1

INTRODUCTION TO ANDROID

Unit Structure

1.0 OBJECTIVE

There are three main objectives for a mobile app: User Growth, Engagement, and Brand Awareness.

1.1 INTRODUCTION

Android is an open source and Linux-based **Operating System** for mobile devices such as smartphones and tablet computers. Android was developed by the *Open Handset Alliance*, led by Google, and other companies.

Android offers a unified approach to application development for mobile devices which means developers need only develop for Android, and their applications should be able to run on different devices powered by Android.

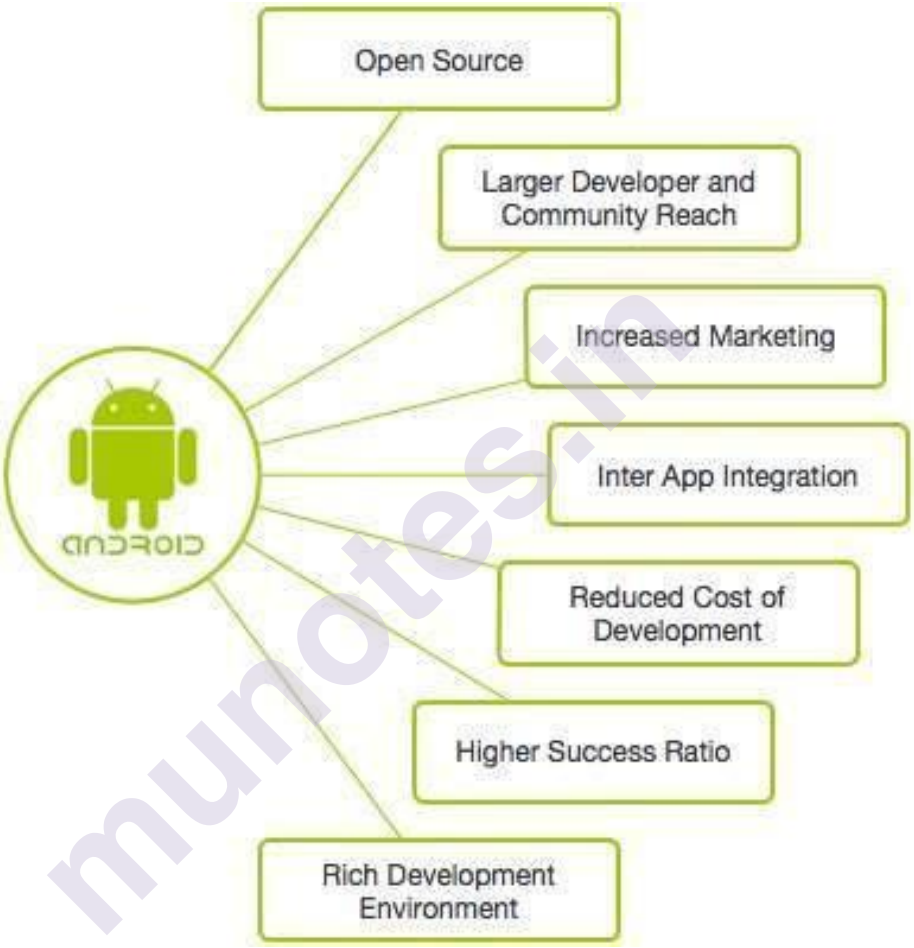
The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008.

On June 27, 2012, at the Google I/O conference, Google announced the next Android version, 4.1 **Jelly Bean**. Jelly Bean is an incremental

update, with the primary aim of improving the user interface, both in terms of functionality and performance.

The source code for Android is available under free and open source software licenses. Google publishes most of the code under the Apache License version 2.0 and the rest, Linux kernel changes, under the GNU General Public License version 2.

Why Android ?



Features of Android:

Android is a powerful operating system competing with Apple 4GS and supports great features. Few of them are listed below:

Sr.No.	Feature & Description
1	**Beautiful UI** Android OS basic screen provides a beautiful and intuitive user interface.
2	**Connectivity** GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.

3	**Storage** SQLite, a lightweight relational database, is used for data storage purposes.
4.	**Media support** H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP.
5.	**Messaging** SMS and MMS
6.	**Web browser** Based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3.
7.	**Multi-touch** Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero.
8.	**Multi-tasking** User can jump from one task to another and same time various application can run simultaneously.
9.	**Resizable widgets** Widgets are resizable, so users can expand them to show more content or shrink them to save space.
10.	**Multi-Language** Supports single direction and bi-directional text.
11.	**GCM** Google Cloud Messaging (GCM) is a service that lets developers send short message data to their users on Android devices, without needing a proprietary sync solution.
12.	**Wi-Fi Direct** A technology that lets apps discover and pair directly, over a high-bandwidth peer-to-peer connection.
13.	**Android Beam** A popular NFC-based technology that lets users instantly share, just by touching two NFC-enabled phones together.

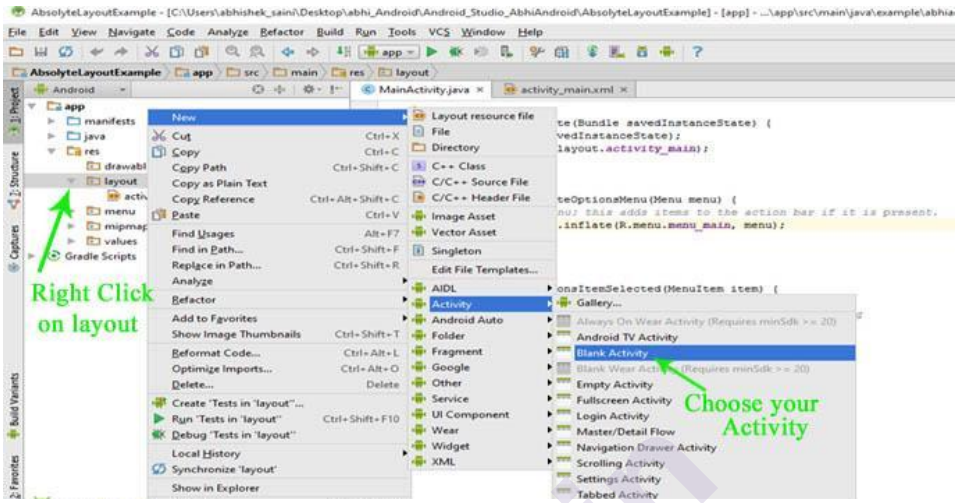
Creating the activity in android:

How to Create New Activity in Android Studio:

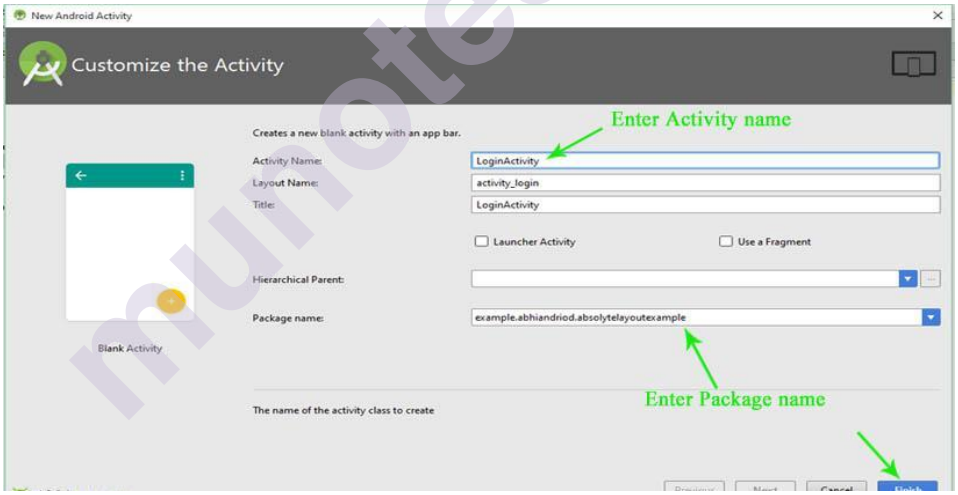
We create New Activity in Android Studio to create XML file for designing UI and java file coding. Below are the steps to create new Activity in Android Studio:

How To Create New Activity in Android Studio:

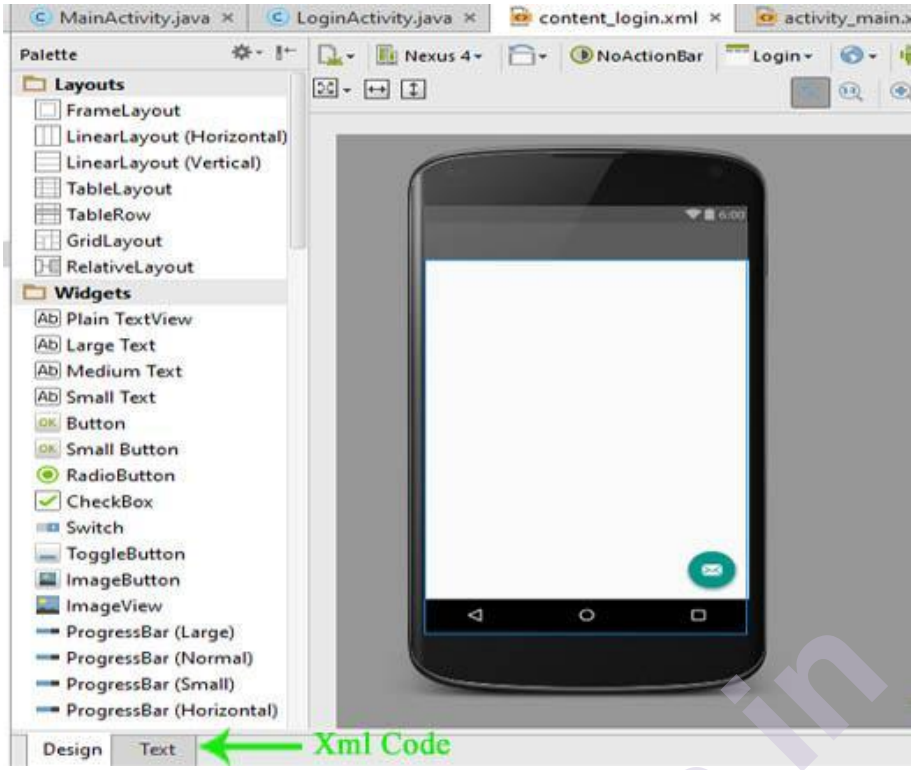
Step 1: Firstly, click on app > res > layout > Right Click on layout. After that Select New > Activity and choose your Activity as per requirement. Here we choose Blank Activity as shown in figure below.



Step 2: After that Customize the Activity in Android Studio. Enter the "Activity Name" and "Package name" in the Text box and Click on Finish button.



Step 3: After that your new Activity in Layout will be created. Your XML Code is in Text and your Design Output is in Design.



Design user interface with views, working:

In Android applications, various types of ViewGroups are used to design UI.

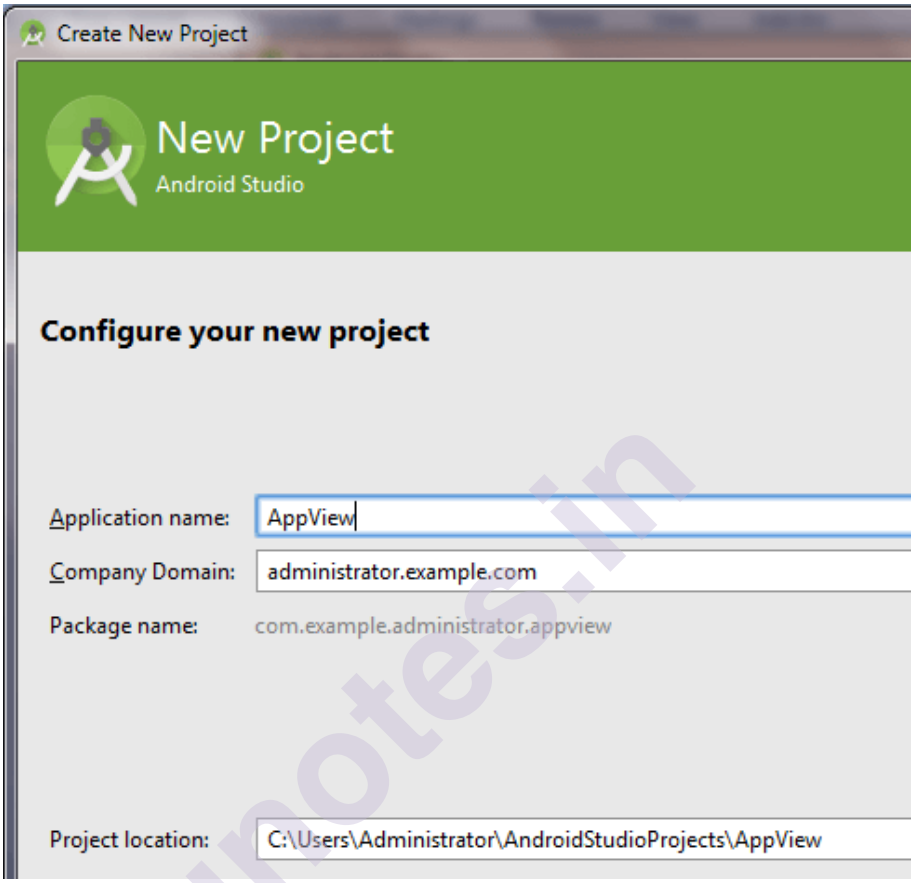
1. Basic Views
2. Picker Views
3. List Views
4. Specialized Fragments
5. Analog and Digital Clock Views

In Android applications, the following are Basic Views.

- TextView
- EditText
- Button
- ImageButton
- CheckBox
- ToggleButton
- RadioButton
- RadioGroup

Implementation:

Create a new Android project called AppView. By default, it creates main.xml file located in the res/layout folder, which contains a <TextView> element.




```
1.  <LinearLayout
2.      xmlns:android="http://schemas.android.com/apk/res/android"
3.      xmlns:tools="http://schemas.android.com/tools"
4.
5.      android:layout_width="fill_parent"
6.      android:layout_height="match_parent"
7.      android:orientation="vertical"
8.      tools:context=".MainActivity">
9.  <TextView android:text="@string/hello_world"
10.     android:layout_width="fill_content"
11.     android:layout_height="wrap_content" />
12. </LinearLayout >
```

The TextView is used to display text/caption to the user. This is the most basic View and very frequently used in an application.

The next View is a subclass of TextView and it is EditText. This View allows the user to edit the text displayed.

```
<EditText
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:id="@+id/txtUserName" />
```

Button represents a push-button widget.

```
<Button
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:id="@+id/btnAdd"
   android:text="Add"/>
```

ImageButton is similar to Button View except that it displays an image with text.

```
<ImageButton
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:id="@+id/imgButton"
   android:src="@drawable/abc_ic_menu_copy_mtrl_am_alpha"
/>
```

CheckBox is a type of button that has two states; i.e., checked or unchecked.

```
<CheckBox
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:id="@+id/chkIndia"
   android:text="India"
/>
<CheckBox
```

```
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/chkUS" style="?android:attr/starStyle"
    android:text="US"
/>
```

RadioGroup and RadioButton, both have two states: either checked or unchecked. A RadioGroup is used to group together one or more RadioButton Views, thereby allowing only one RadioButton to be checked within the RadioGroup.

```
<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:id="@+id/rdoGroup">
<RadioButton
    android:id="@+id/rbMale"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Male"/>
<RadioButton
    android:id="@+id/rbFemale"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Female"/>
</RadioGroup>
```

ToggleButton displays checked/unchecked states using a light indicator.

```
<ToggleButton
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/toggleButton"/>
```

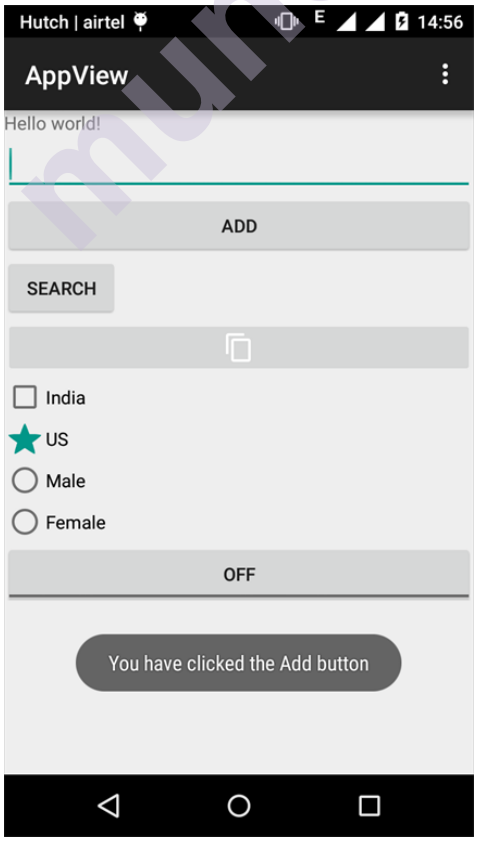
Now, add some code as below to handle View events for elements, like Button, CheckBox etc.

```
1.  Button buttonAdd = (Button) findViewById(R.id.btnAdd);
2.  buttonAdd.setOnClickListener(new View.OnClickListener()
3.    {
4.      @Override
5.      public void onClick(View view)
6.      {
7.          DisplayMessage("You have clicked the Add button");
8.      }
9.  });
```

Write a common method to display the text message as below.

```
1.  private void DisplayMessage(String textMessage) {
2.      Toast.makeText(getBaseContext(), textMessage, Toast.LENGTH
_SHORT).show();
3.  }
```

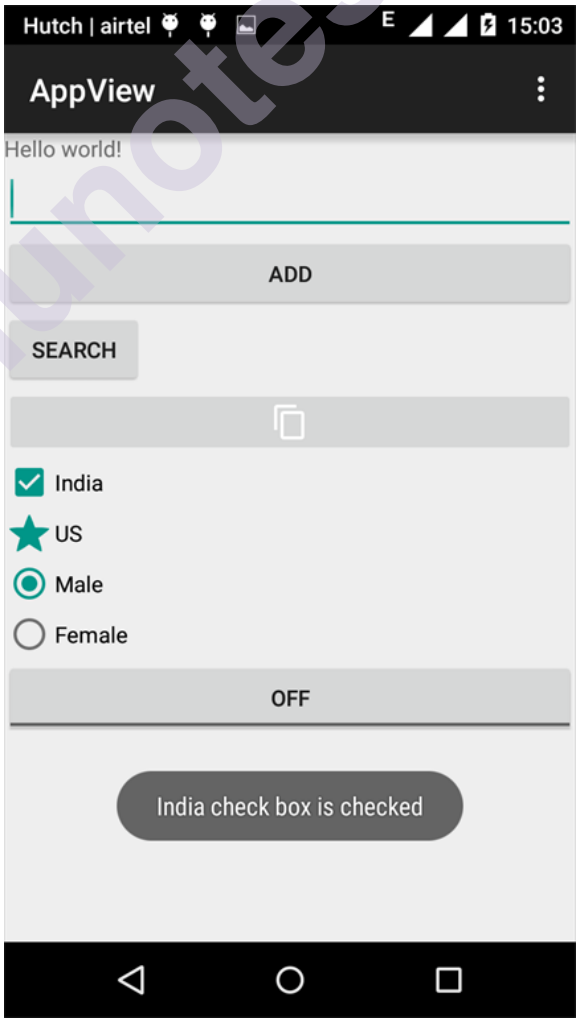
Now, run the application by pressing F11.



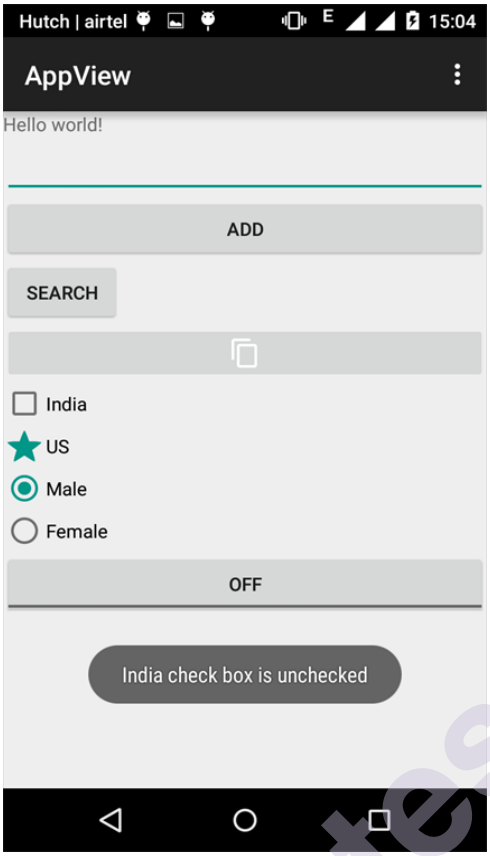
To handle the View events for element CheckBox, add the below code.

```
1.  CheckBox checkBox = (CheckBox) findViewById(R.id.chkIndia);
2.  checkBox.setOnClickListener(new View.OnClickListener()
3.  {
4.      @Override
5.      public void onClick(View view)
6.      {
7.          if (((CheckBox) view).isChecked()) DisplayMessage("India ch
eck box is checked");
8.          else DisplayMessage("India check box is unchecked");
9.      }
10. });
```

Now, run the application by pressing F11. For example, I have checked the India CheckBox.

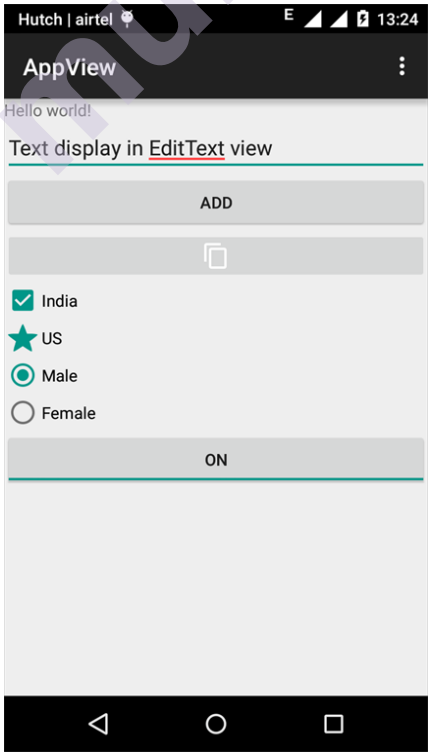


Now, uncheck the India CheckBox.



Explanation:

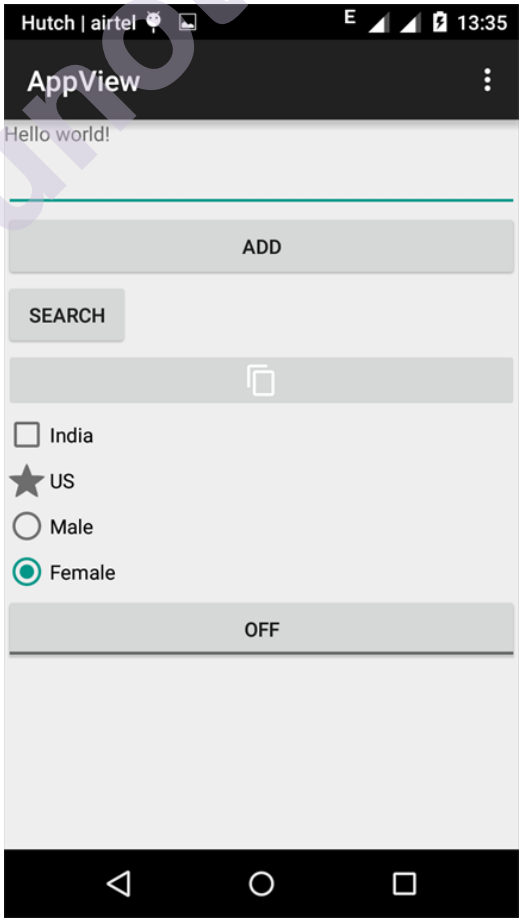
Run the application by pressing F11 and see the result of the above implemented code.



All the Views are relatively straightforward because they are listed using the <LinearLayout> element. So, they are stacked on top of each other when they are displayed in the activity.

```
1.  <Button
2.      android:layout_width="fill_parent"
3.      android:layout_height="wrap_content"
4.      android:id="@+id/btnAdd"
5.      android:text="Add"/>
6.
7.  <Button
8.      android:layout_width="wrap_content"
9.      android:layout_height="wrap_content"
10.     android:id="@+id/btnSearch"
11.     android:text="Search"/>
```

In the above code for first Button, the layout_width attribute is set to fill_parent so that its width occupies the entire width of the screen.



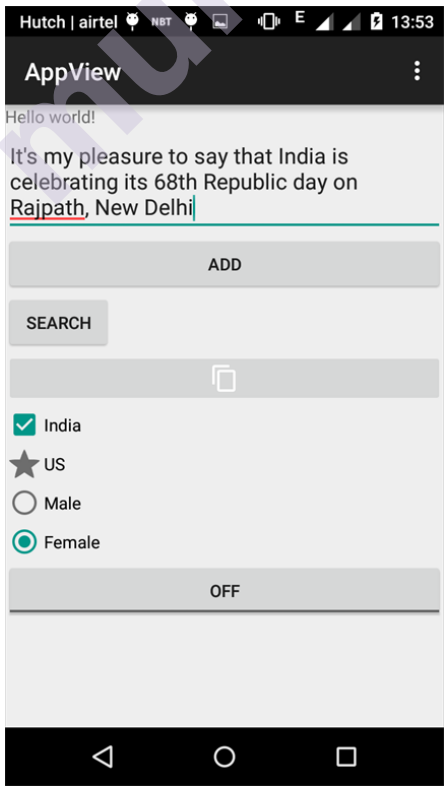
But for the second Button, the layout_width attribute is set to wrap_content so that its width will be the width of its content.

The ImageButton displays a button with an image. The image is set through the src attribute.

```
1.  <ImageButton
2.      android:layout_width="fill_parent"
3.      android:layout_height="wrap_content"
4.      android:id="@+id/imgButton"
5.      android:src="@drawable/abc_ic_menu_copy_mtrl_am_alpha"
6.  />
```

EditText displays a rectangular region where the user can enter some text. Set the attribute of layout_height to wrap_content so that if the user enters a long string of text, its height will automatically be adjusted to fit the content.

```
1.  <EditText
2.      android:layout_width="fill_parent"
3.      android:layout_height="wrap_content"
4.      android:id="@+id/txtUserName"
5.  />
```



The CheckBox displays a checkbox that users can tap on to check or uncheck.

```
<CheckBox
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/chkIndia"
    android:text="India"
/>
<CheckBox
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/chkUS" style="?android:attr/starStyle"
    android:text="US"
/>
```

In the above code, style attribute is set to display another image rather than default one.

There is RadioGroup that encloses two RadioButtons. This is very important because radio buttons are usually used to present multiple options to the user for selection. When a RadioButton in a RadioGroup is selected, all the other RadioButtons are automatically unselected.

```
<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:id="@+id/rdoGroup">
<RadioButton
    android:id="@+id/rbMale"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Male"/>
<RadioButton
```

```
12.    android:id="@+id/rbFemale"
13.    android:layout_width="fill_parent"
14.    android:layout_height="wrap_content"
15.    android:text="Female"/>
16. </RadioGroup>
```

In the above code, RadioButtons are displayed vertically. To display the list horizontally, just change the orientation attribute to horizontal. If you change the orientation attribute to horizontal, you will have to change the layout_width attribute to wrap_content.

The ToogleButton displays a rectangular button that users can toggle either ON or OFF, by tapping.

```
1.  <ToggleButton
2.     android:layout_width="fill_parent"
3.     android:layout_height="wrap_content"
4.     android:id="@+id/toggleButton"/>
```

In the above code, each element has the id attribute with a particular value as we can see for the Button View.

```
1.  <Button
2.     android:layout_width="wrap_content"
3.     android:layout_height="wrap_content"
4.     android:id="@+id/btnSearch"
5.     android:text="Search"/>
```

The id attribute is an identifier for a View so that it may later be retrieved using the View.findViewById() methods.

To handle View events for Button element, add the code given below.

```
1.  Button buttonAdd=(Button) findViewById(R.id.btnAdd);
```

The setOnClickListener() method registers a callback to be invoked later when the View is clicked.

```
1.  buttonAdd.setOnClickListener(new View.OnClickListener() {
2.     @Override
3.     public void onClick(View view) {
4.        DisplayMessage("You have clicked the Add button");
```

```
5.      }
6.  });
```

The onClick() method is called when the View is clicked.

The next element is CheckBox for which code is added to handle their View events. To determine the state of the CheckBox, you must have to typecast the argument of the onClick() method to a CheckBox and then click its isChecked() method to see if it is clicked or not.

```
1.  CheckBox checkBox = (CheckBox) findViewById(R.id.chkIndia);
2.  checkBox.setOnClickListener(new View.OnClickListener() {
3.      @Override
4.      public void onClick(View view) {
5.          if (((CheckBox) view).isChecked()) DisplayMessage("India ch
eck box is checked");
6.          else DisplayMessage("India check box is unchecked");
7.      }
8.  });
```

1.2 SUMMARY

It is currently used in various devices such as mobiles, tablets, televisions, etc. Android provides a rich application framework that allows us to build innovative apps and games for mobile devices in a Java language environment.

1.3 REFERENCES

- Mobile Computing with Android - A Practical Approach Paperback – 4 February 2015 by Rajesh Vartak (Author)
- Mobile Computing (ebook) by Kumkum Garg
- Mobile Computing

  Author(s): Sudeep Tanwar, Sushil Kumar Singh, Sudhanshu Tyagi

1.4 UNIT END EXERCISE

Create a basic calculator app using android studio.

*****

2

BASIC CONTROLS AND UI COMPONENT

Unit Structure

2.1 AIM

Creating android application for generating user interface for student Registration and feedback form by using all basic UI controls.

2.2 OBJECTIVE

The objective of this module is to study and understands basic controls and user interface component of android.

2.3 THEORY

Android provides a various type of controls such as buttons, text fields, seek bars, check box, zoom buttons, toggle buttons, and many more that you can use to design your user interface. Input controls are the interactive components for your user interface in android app.

Android UI controls -Basic Types:

1) **TextView:** It is used to display text to the user. Basically, it is a small label that is displaying some information.

2) **EditText:** It is a predefined subclass of TextView that include editing capability.

3) **Button:** It is a push button that can be pressed or clicked by the user to perform some action.

4) **Checkbox:** They are a group of selectable option that are not mutually exclusive (each check box is independent of each other).

5) **RadioButton:** RadioButton can have two states, either checked or unchecked.

Android UI controls -List Types:

1) **Spinner:** It is a drop-down list that allows users to select one value from a set of values.

2) **AutoCompleteTextView**: It is a editable text view which is used to show the list of suggestions based on the user typing text. The list of suggestions will be shown as a dropdown menu from which the user can choose an item to replace the content of the textbox.

3) **MultiAutoCompleteTextView:** It is similar to AutoCompleteTextView, except that it shows multiple value.

Android UI controls -Advanced Types:

1) **ImageButton:** It is similar to push button that can be pressed or clicked to perform some action but it comes with an image as source.

2) **ToggleButton:** A toggle button allows the user to change a setting between two states, that is on and off.

3) **RadioGroup:** It is used to group together one or more RadioButton.

4) **ProgressBar:** when you are performing task in the background, this control is used to provides visual progress.

5) **TimePicker:** It is used to select a time of the day.

6) **DatePicker:** It is used to select a date of the day.

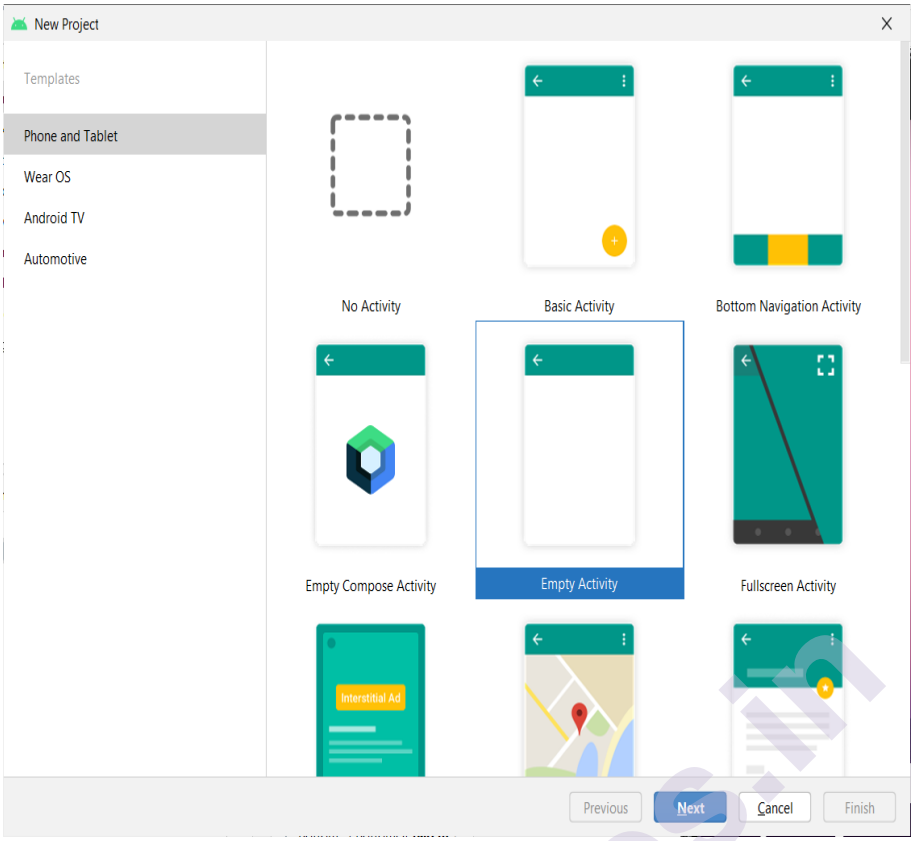
2.4 PROGRAM 1

1) Program to demonstrate use of different text control, RadioGroup, RadioButton, Checkbox and Button control by creating Registration form

1) To create your new Android project, click on **Create New Project** on the **Welcome to Android Studio** window.

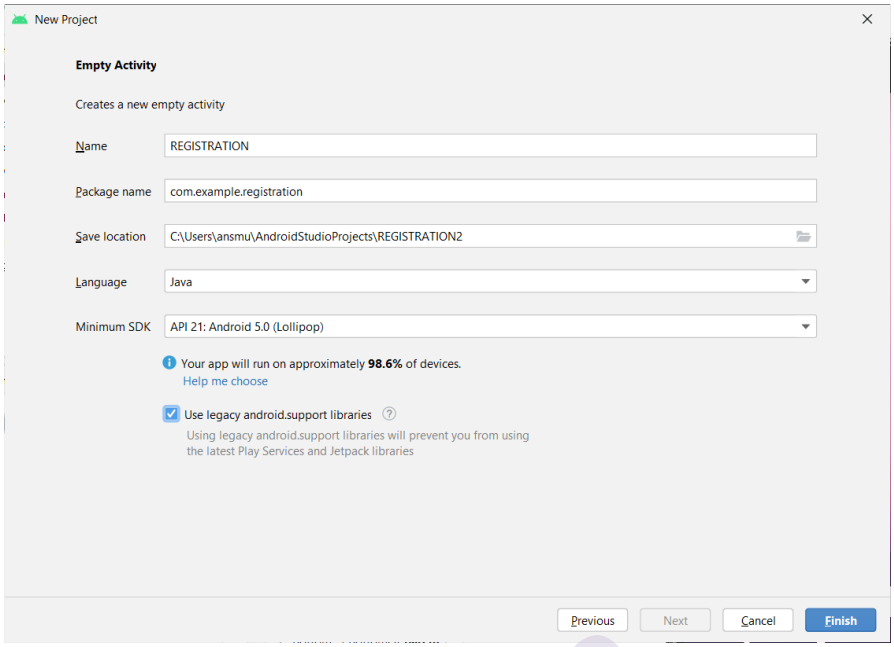
2) If you have a project already opened, select **File > New > New Project**.

3) In the **Select a Project Template** window, select **Empty Activity** andclick **Nex**t.



4) In the **Configure your project** window, complete the following:

- Enter "Registration" in the **Name** field.
- Enter "com.example.registration" in the **Package name** field.
- If you'd like to place the project in a different folder, change its **Save** location.
- Select either **Java** or **Kotlin** from the **Language** drop-down menu.
- Select the lowest version of Android you want your app to support in the **Minimum SDK** field.
- If your app will require legacy library support, mark the **Use legacy android.support libraries** checkbox.
- Leave the other options as they are.

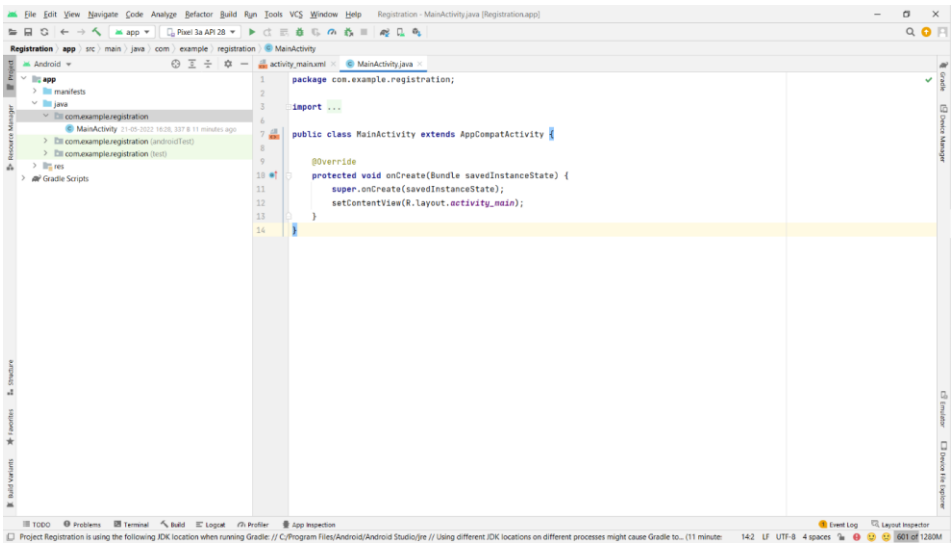


5) Click on finish

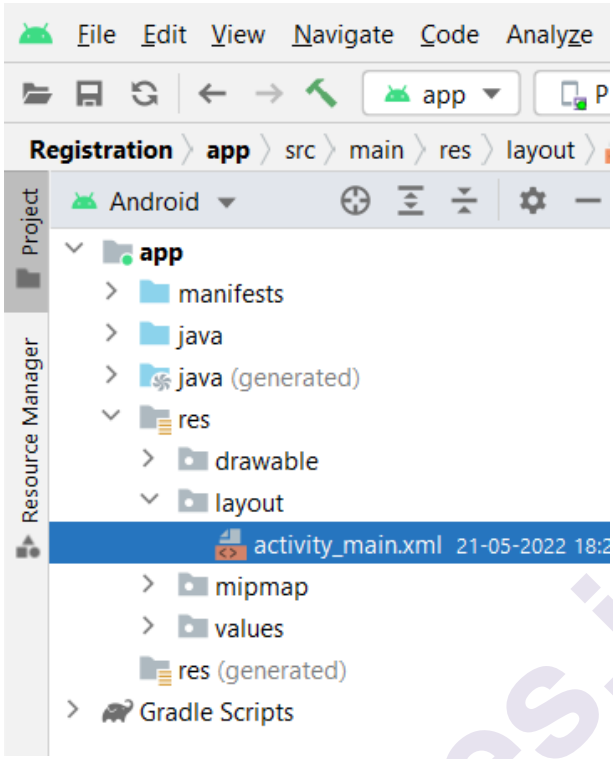
In this program we are going to use the following controls:

1. ImageButton
2. Text(Plain Text)
3. Text (E Mail)
4. Text (Phone)
5. Text (Postal Address)
6. CheckBox
7. RadioGroup with two RadioButton
8. Button

After some processing time, the Android Studio main window appears as follows:



As in this practical we have to make use of different UI controls we will click on **res** folder and then open **activity_main.xml** file and start coding.



We can design activity_main.xml file in three different modes: **Code mode, split mode and Design mode.**

Code for activity_main.xml file

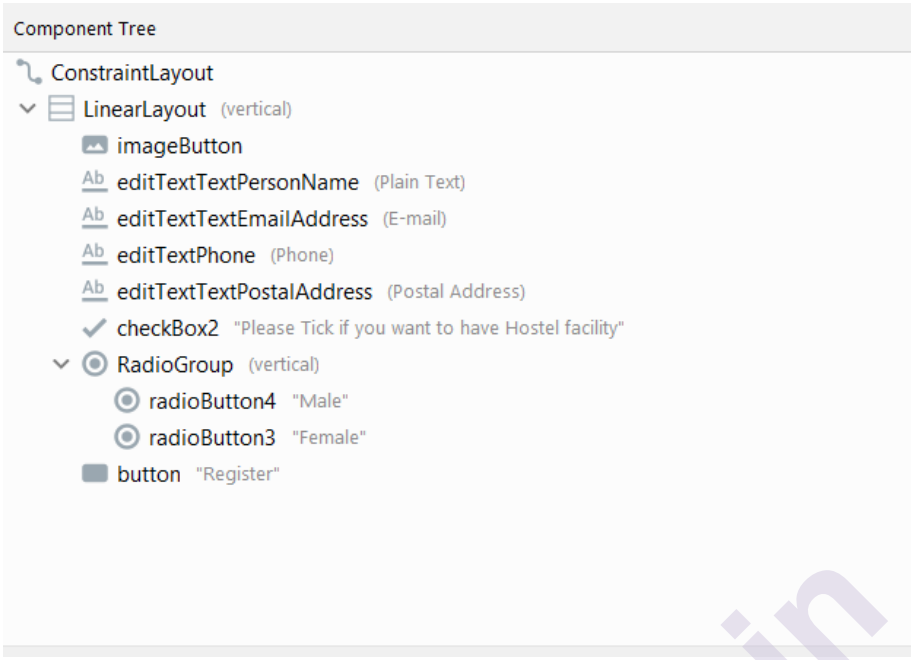
```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:app="http://schemas.android.com/apk/res-auto"

    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    tools:context=".MainActivity">


    <LinearLayout

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:orientation="vertical"
```

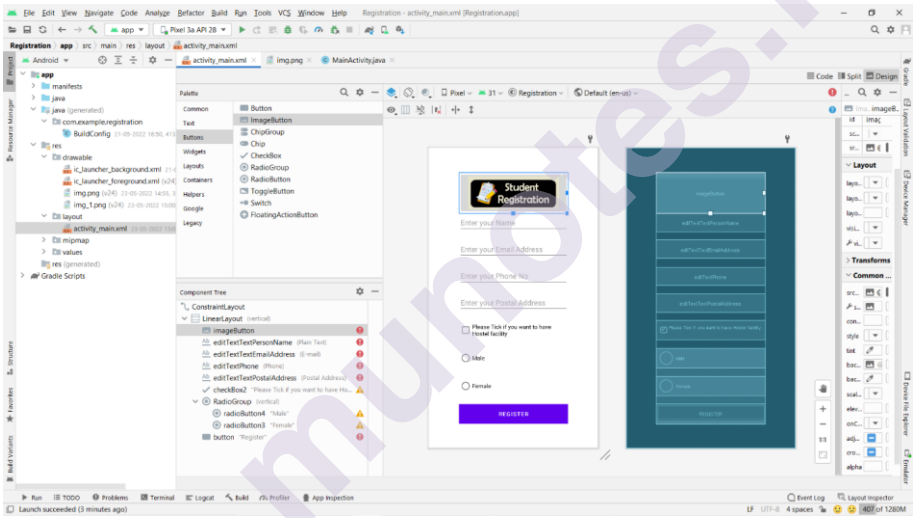
```
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent">

<ImageButton
   android:id="@+id/imageButton"
   android:layout_width="match_parent"
   android:layout_height="100dp"
   android:scaleType="fitCenter"
   app:srcCompat="@drawable/img" />

<EditText
   android:id="@+id/editTextTextPersonName"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:ems="20"
   android:inputType="textPersonName"
   android:layout_marginBottom="20dp"
   android:hint="Enter your Name" />

<EditText
   android:id="@+id/editTextTextEmailAddress"
   android:layout_width="265dp"
   android:layout_height="wrap_content"
   android:hint="Enter your Email Address"
   android:ems="10"
   android:layout_marginBottom="20dp"
   android:inputType="textEmailAddress" />
```

```
<EditText
    android:id="@+id/editTextPhone"
    android:layout_width="265dp"
    android:layout_height="wrap_content"
    android:ems="10"
    android:layout_marginBottom="20dp"
    android:hint="Enter your Phone No"
    android:inputType="phone" />

<EditText
    android:id="@+id/editTextTextPostalAddress"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:layout_marginBottom="20dp"
    android:hint="Enter your Postal Address"
    android:inputType="textPostalAddress" />

<CheckBox
    android:id="@+id/checkBox2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="20dp"
    android:text="Please Tick if you want to have Hostel facility" />

<RadioGroup
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
```

```
        <RadioButton
            android:id="@+id/radioButton4"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginBottom="20dp"
            android:text="Male" />

        <RadioButton
            android:id="@+id/radioButton3"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginBottom="20dp"
            android:text="Female" />

    </RadioGroup>

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#F1A7A7"
        android:text="Register" />

  </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Component Tree:





Activity_main.xml in design mode

Note:

In this program we are only designing user interface and not providing any functionality to it. That is why, writing java files is not required.

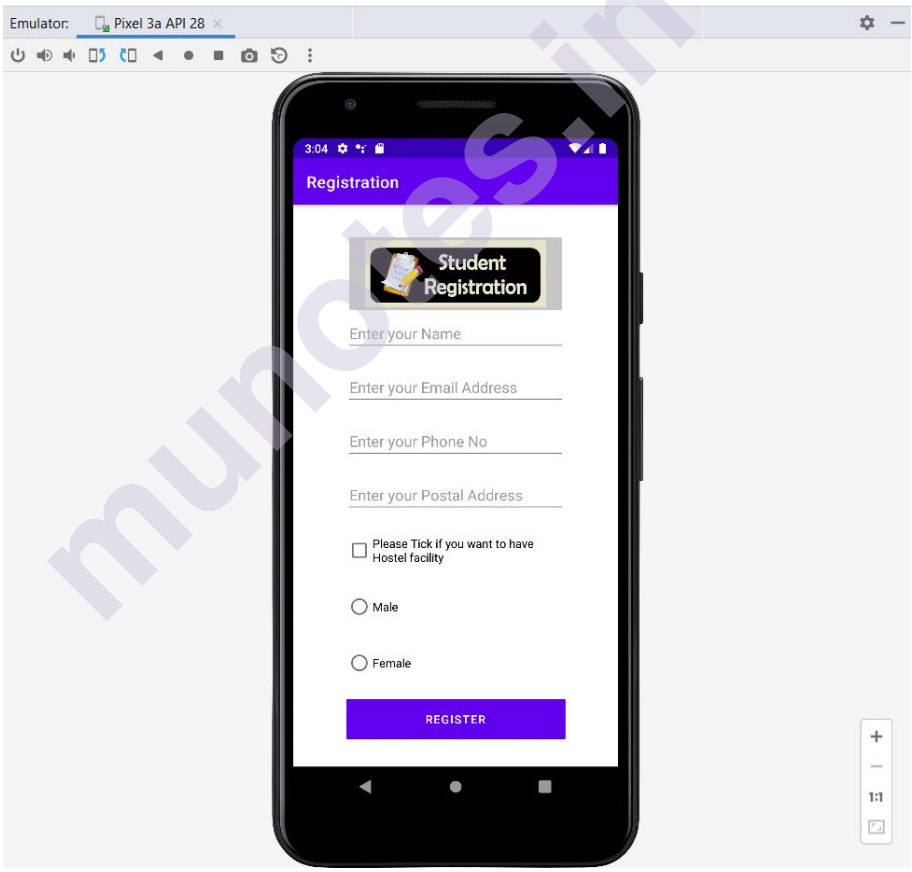
As in this practical we have use ImageButton Control, first we have to add image in drawable folder. Then that image will be displayed on the button.

MainActivity.java (This is autogenerated code,we can add our code as per our program in this file)

```
package com.example.registration;


import androidx.appcompat.app.AppCompatActivity;
```

```
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

2.4.1 Output:



2.5 PROGRAM 2

1) Program to demonstrate use of Spinner, AutoCompleteTextView, multiline text and TextView control by creating Feedback form

In this program we are going to use the following controls:

1. Text View

2. Text (Plain Text)
3. spinner
4. AutoCompleteTextView
5. Multiline Text
6. Button

Coding:

activity_main.xml

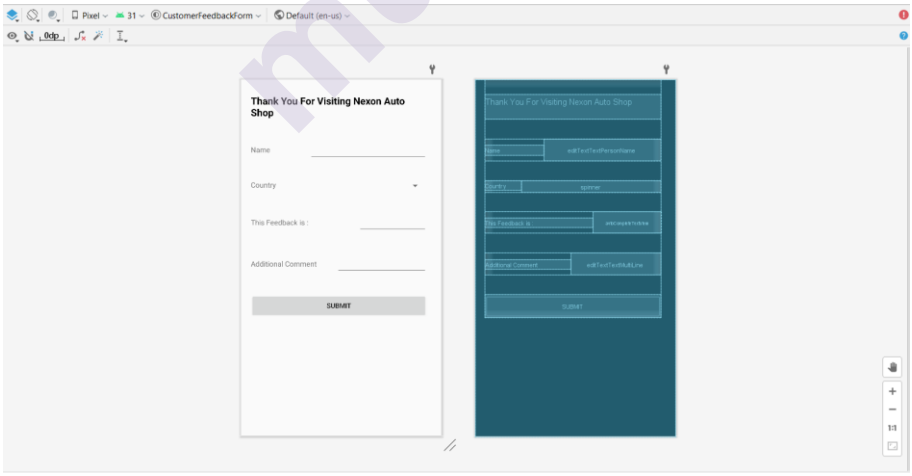
```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        tools:ignore="MissingConstraints"
        android:layout_marginLeft="20dp"
        android:layout_marginRight="30dp">

        <TextView
            android:id="@+id/textView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Thank You For Visiting Nexon Auto Shop"
            android:layout_marginBottom="40dp"
```

```
    android:layout_marginTop="30dp"
    android:textColor="@color/black"
    android:textSize="20sp"
    android:textStyle="bold"
    />

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_marginBottom="40dp">

    <TextView
        android:id="@+id/textView2"
        android:layout_width="90dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Name"
        android:textSize="15sp"/>

    <EditText
        android:id="@+id/editTextTextPersonName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:ems="10"
        android:inputType="textPersonName" />

</LinearLayout>
```

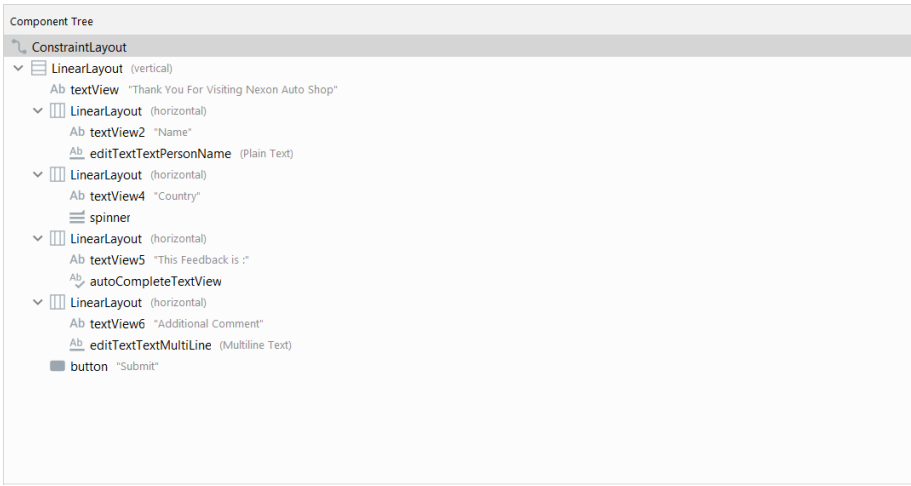
```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_marginBottom="40dp">

    <TextView
        android:id="@+id/textView4"
        android:layout_height="wrap_content"
        android:layout_width="150dp"
        android:layout_weight="1"
        android:text="Country"
        android:textSize="15sp"/>

    <Spinner
        android:id="@+id/spinner"

        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1" />
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_marginBottom="40dp">
```

```
<TextView
    android:id="@+id/textView5"
    android:layout_width="100dp"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="This Feedback is :"
    android:textSize="15sp"/>

  <AutoCompleteTextView
    android:id="@+id/autoCompleteTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1" />

</LinearLayout>

<LinearLayout
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:orientation="horizontal"
  android:layout_marginBottom="40dp">

  <TextView
    android:id="@+id/textView6"
    android:layout_width="200dp"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="Additional Comment"
    android:textSize="15sp"/>
```

```
        <EditText
            android:id="@+id/editTextTextMultiLine"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:ems="10"
            android:gravity="start|top"
            android:inputType="textMultiLine" />
    </LinearLayout>

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Submit" />

</LinearLayout>
</android.support.constraint.ConstraintLayout>
```



Design mode



Component Tree

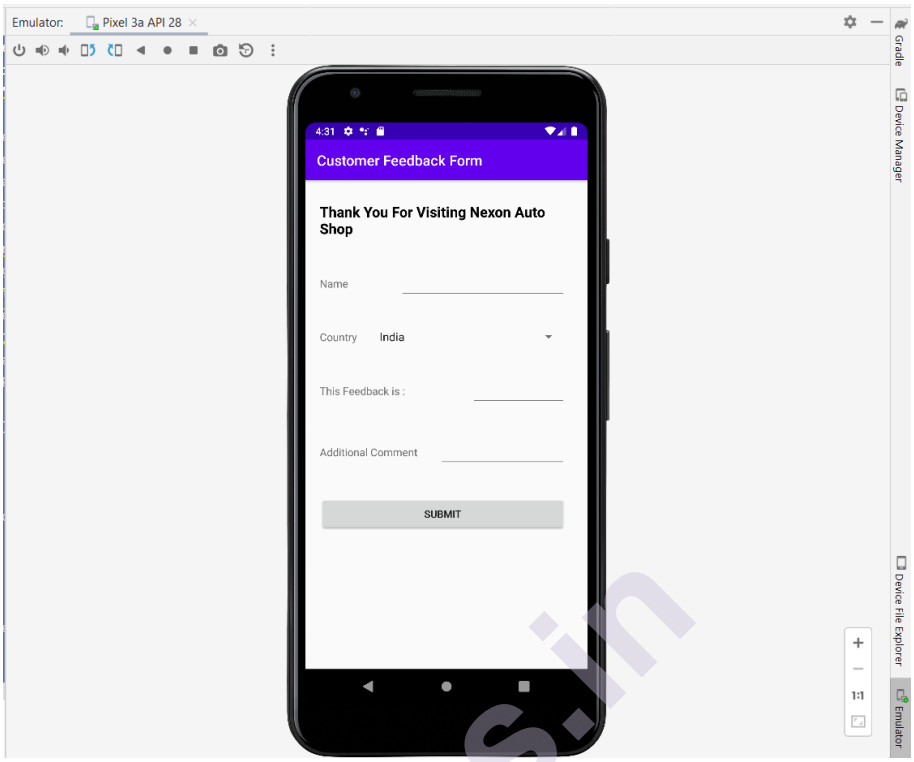
Note: For the functioning of the spinner and AutoCompleteTextView control, Code has to be written in mainActivity.java

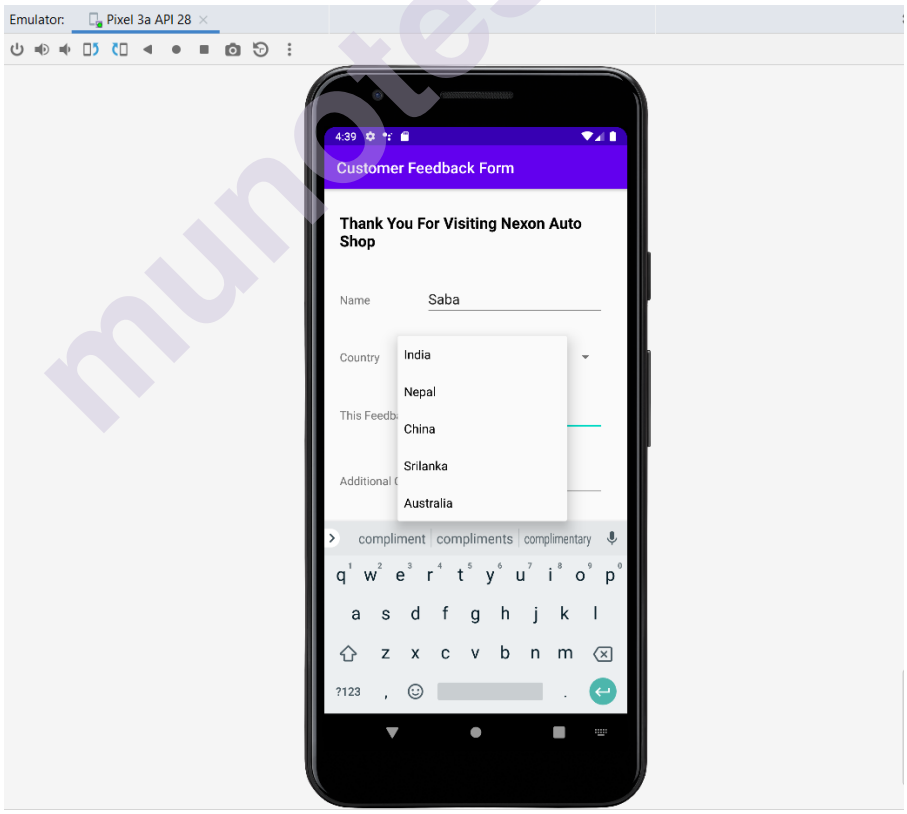
MainActivity.java

```
package com.example.customerfeedbackform;


import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.AdapterView;

import android.widget.AdapterView.OnItemSelectedListener;

import android.widget.ArrayAdapter;

import android.widget.Spinner;

import android.widget.Toast;

import android.widget.AutoCompleteTextView;


public class MainActivity extends AppCompatActivity implements OnItemSelectedListener{

   String[] Country={"India","Nepal","China","Srilanka","Australia"};

   String[] feedback = { "suggestion", "compliment", "complaint", "other"};

   @Override
```

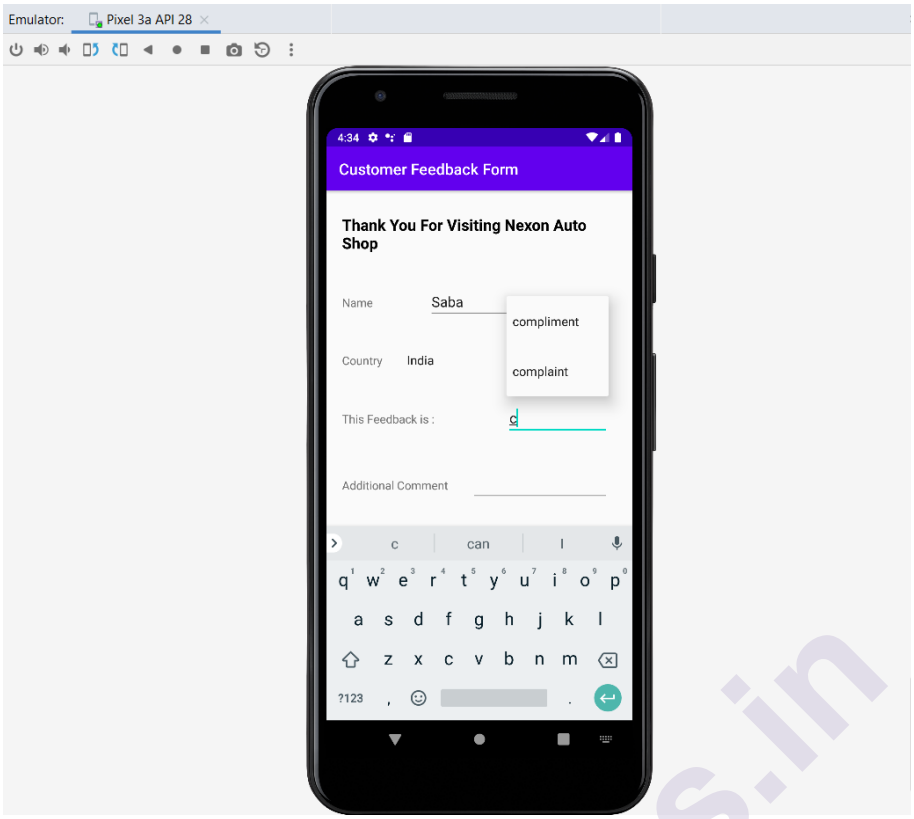
```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Spinner spin = (Spinner) findViewById(R.id.spinner);
        spin.setOnItemSelectedListener(this);
        ArrayAdapter aa = new ArrayAdapter(this,android.R.layout.simple_spinner_item,Country);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                android.R.layout.simple_dropdown_item_1line, feedback);
        AutoCompleteTextView actv = (AutoCompleteTextView)findViewById(R.id.autoCompleteTextView);
        actv.setThreshold(1);
        actv.setAdapter(adapter);

aa.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
//Setting the ArrayAdapter data on the Spinner
        spin.setAdapter(aa);
    }

    @Override
    public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l) {

    }

    @Override
    public void onNothingSelected(AdapterView<?> adapterView) {

    }
}
```

2.5.1 Output:

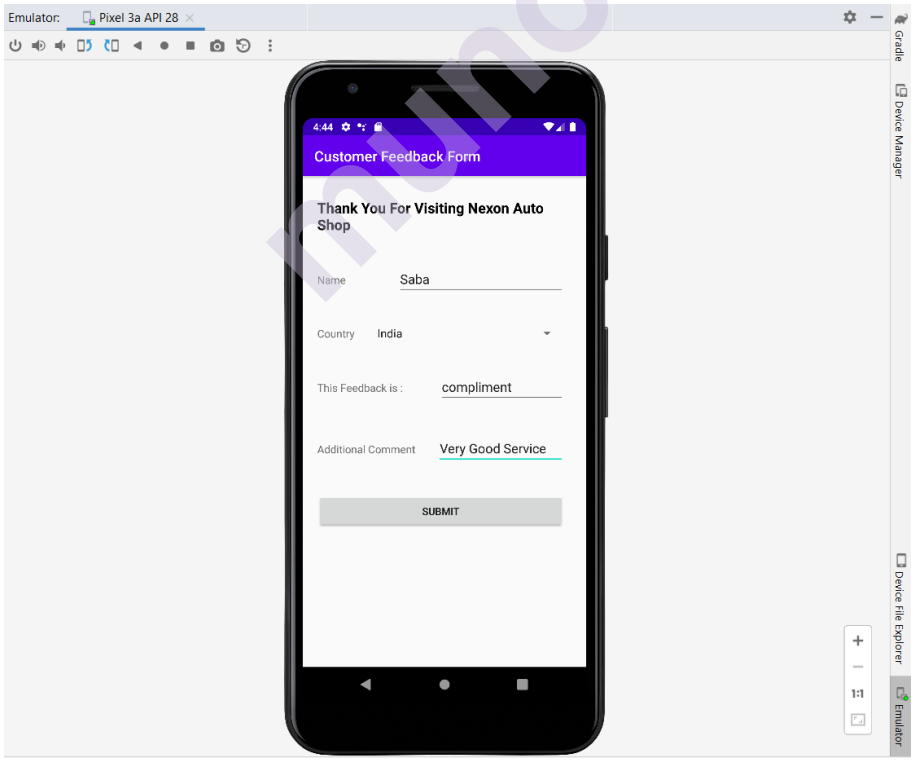




In the above output we can see the use of spinner control. For the selection of country, spinner control is used. If we click on the drop down button we will get country name.



In the above output we can see the use of AutoCompleteTextView control. As user has typed c ,two suggestion is given "Compliment" and "complaint".



2.6 QUESTIONS

1. The layout or design of an android application is saved in

   A. .text file

   B. .java file

   C. .dex file

   D. .xml file

2. Which of the following is not an Android component (i.e. a point from which the system can enter your application)?

   A. Service

   B. Activity

   C. Layout

   D. Content Provider

3. What Activity method you use to retrieve a reference to an android view by using the id attribute of a resource XML?

   A. findViewByReference(int id);

   B. findViewById(int id)

   C. retrieveResourceById(int id)

   D. findViewById(String id)

4. Select correct UI control: A drop-down list that allows users to select one value from a set.

   A. List

   B. Spinner

   C. DropdownList

   D. Choice

5. In which folder the activity_main.xml a layout file is available.

   A. res/drawable

   B. res/layout

   C. res/values

   D. none of the above

6. Which file contains compilesdkVersion, buildtoolVersion, minsdkVersion, versionName, etc

   A. Build.gradle

   B. res

   C. mainfest.xml

   D. activity_main.xml

2.7 Self Learning Topic: Methods of all Control Class

Each control in android has their own set of methods that help us to configure those controls. It is very important to study various methods so that a powerful android application can be created.

*****

3

DATA BASE CONNECTIVITY

3.1 PRACTICAL NO 1: INTERNAL STORAGE

1.1 Aim:

Creating android program to demonstrate the use of Internal Storage.

1.2 Objective:

The main objective of this practical is to study and understand the concept of internal storage in android.

1.3 Theory:

Android provides many types of storage for application, so that application can store the data. Internal storage is the storage on the device memory that is used to store private data. Internal Storage are the primary memory of your phone. By default, these files are private and are accessed by only your application and get deleted, when user delete the application. When files are stored in internal storage these files can only be accessed by the application itself not by other applications. These files in storage exist till the application stays over the device, as you uninstall associated files get removed automatically.

Modes of Internal Storage:

1. **MODE_PRIVATE:** In this mode the data stored earlier is always overridden by the current data i.e., every time when you try to write to a file, the previous content is overridden by the new content.
2. **MODE_APPEND:** In this mode the data is appended to the existing content i.e., keep adding data at the end of the file without removing the previous content.

Writing file:

In order to use internal storage to write some data in the file, call the openFileOutput() method with the name of the file and the mode.

Syntax:

FileOutputStream fOut = openFileOutput("file name here",MODE_WORLD_READABLE);

Reading file:

In order to read from the file you just created, call the openFileInput() method with the name of the file. It returns an instance of FileInputStream.

Syntax:

FileInputStream fin = openFileInput(file);

1.4 Program:

Create a new project and name it InternalStorage.

Activity_main.xml

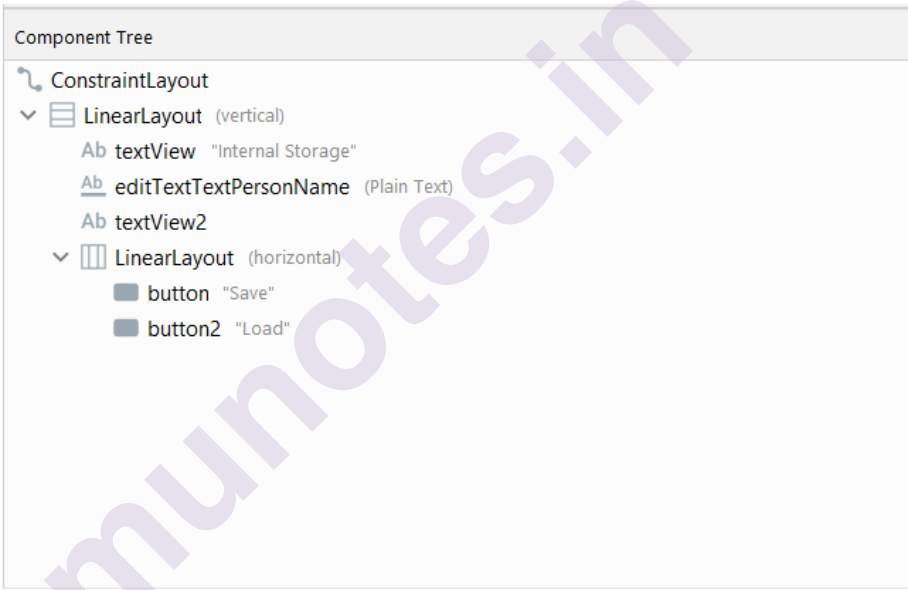
<?xml version="1.0" encoding="utf-8"?>

```
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:layout_marginRight="20sp"
        android:layout_marginLeft="20sp">

        <TextView
            android:id="@+id/textView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Internal Storage"
            android:layout_marginTop="20sp"
            android:layout_marginBottom="20sp"
            android:textSize="40sp"
            android:textColor="@color/black"
            android:textStyle="bold"
            />

        <EditText
            android:id="@+id/editTextTextPersonName"
            android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"
    android:hint="Enter Text"
    android:layout_marginTop="20sp"
    android:layout_marginBottom="20sp"
    android:textSize="30sp"/>

<TextView
    android:id="@+id/textView2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Read"
    android:layout_marginTop="20sp"
    android:layout_marginBottom="20sp"
    android:textSize="30sp"/>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Save" />
```

```
        <Button
            android:id="@+id/button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Load" />
    </LinearLayout>
</LinearLayout>
</android.support.constraint.ConstraintLayout>
```

Component Tree:



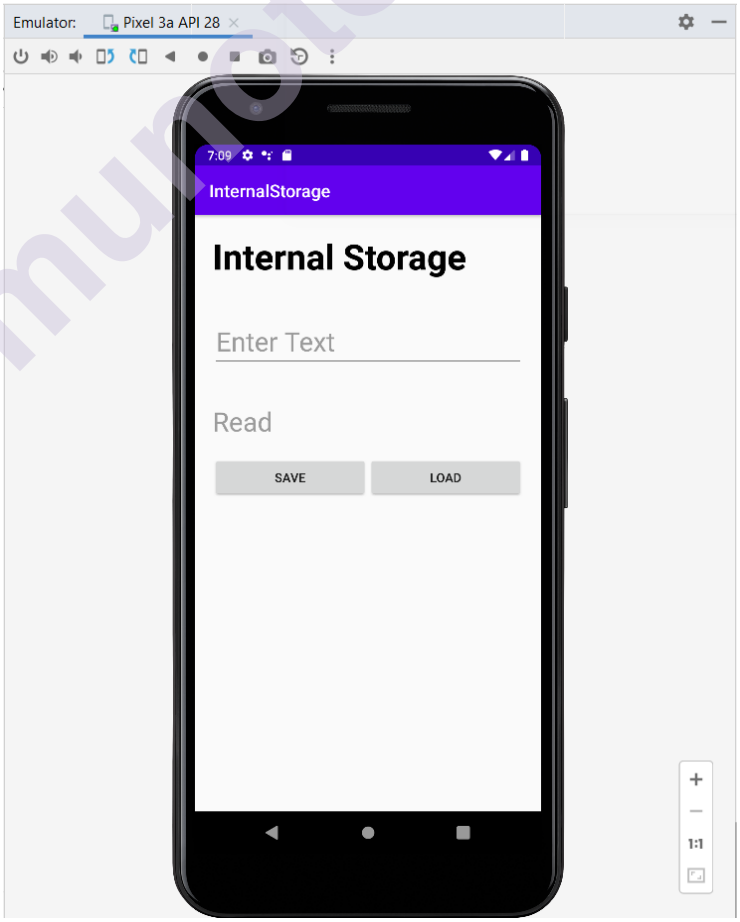
MainActivity.java

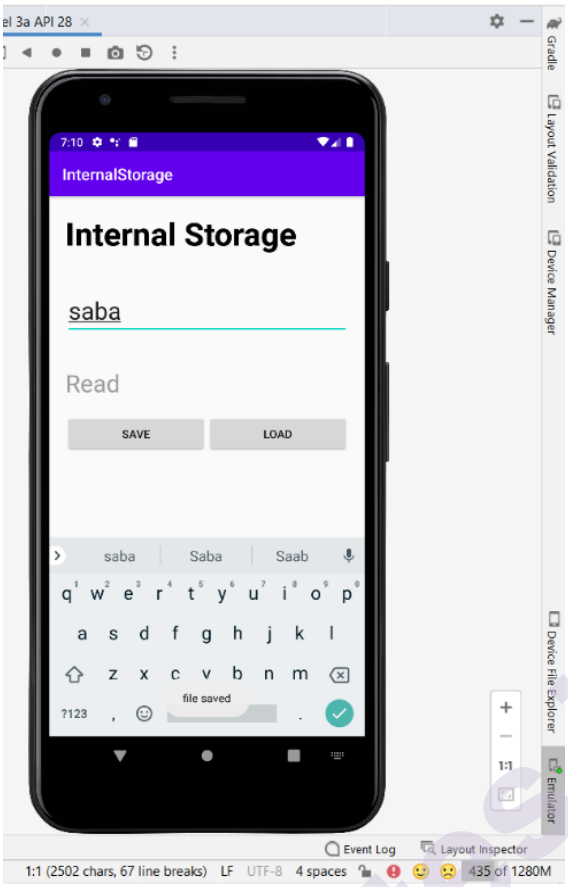
```
package com.example.internalstorage;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
import java.io.FileInputStream;
import java.io.FileOutputStream;
```

```
public class MainActivity extends AppCompatActivity {
    Button b1,b2;
    TextView tv;
    EditText ed1;
    String data;
    private String file = "mydata";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        b1=(Button)findViewById(R.id.button);
        b2=(Button)findViewById(R.id.button2);
        ed1=(EditText)findViewById(R.id.editTextTextPersonName);
        tv=(TextView)findViewById(R.id.textView2);
        b1.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                data=ed1.getText().toString();
                try {
                    FileOutputStream fOut =
openFileOutput(file,MODE_APPEND);
                    fOut.write(data.getBytes());
                    fOut.close();
                    Toast.makeText(getBaseContext(),"file
saved",Toast.LENGTH_SHORT).show();
                }
                catch (Exception e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        });

        b2.setOnClickListener(new View.OnClickListener() {
```

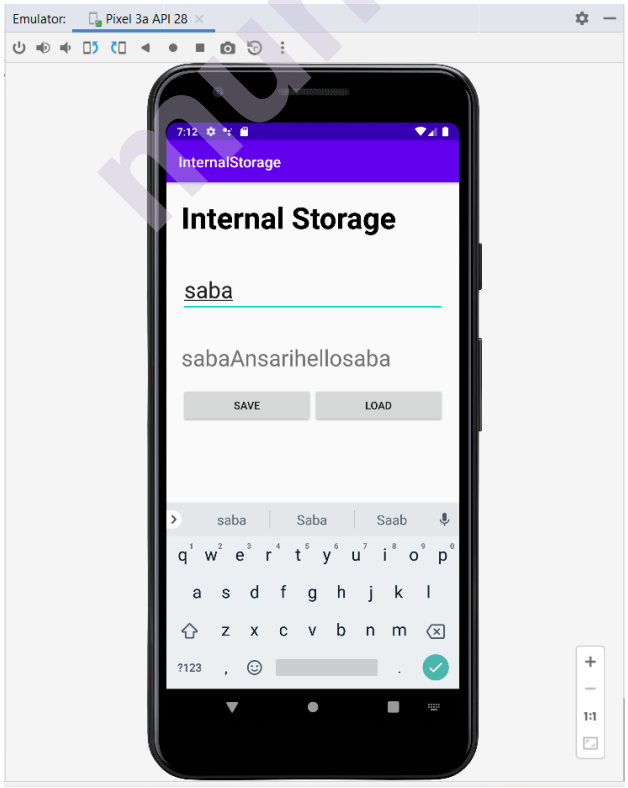
```
        @Override
        public void onClick(View v) {
            try {
                FileInputStream fin = openFileInput(file);
                int c = 0;
                String t="";
                while( (c= fin.read()) != -1) t += Character.toString((char) c);
                tv.setText(t);
                Toast.makeText(getBaseContext(),"file
read",Toast.LENGTH_SHORT).show();
            }
            catch(Exception e){
            }
        }
    });   }
}
```

1.5 Output:

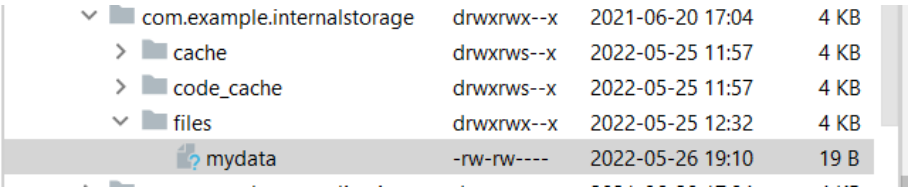




Once we click on load button, all the data saved in internal strorage file will be displayed, As we are using Append mode ,it is showing previous data also.



To view the file in internal storage click on Device File Explorer->Data->Data->com.example.internalstorage->files->myData



3.2 PRACTICAL NO 2: EXTERNAL STORAGE

3.2.1 Aim:

Creating android program to demonstrate the use of external Storage.

3.2.2 Objective:

The main objective of this practical is to study and understand the concept of external storage in android.

3.2.3 Theory:

Depending on our requirement we have different storage options available in android such as shared preferences, internal storage, external storage, SQLite storage, etc. to store and retrieve the application data.

Developers can use the various options available to store data depending on the space required, privacy of data and reliable data access. The data files saved over external storage devices are publicly accessible on shared external storage using USB mass storage transfer. Data files stored on external storage using a FileOutputStream object can be read using a FileInputStream object.

It is very important to add external storage the permission to read and write. For that you need to add permission in android Manifest file.

Open AndroidManifest.xml file and add permissions to it just after the package name.

```
 <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

  <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

3.2.4 Program:

Create a new android application using android studio and give name as ExternalStorage.

Providing Access Permission to External Storage:

To read and write data to external storage, the app required WRITE_EXTERNAL_STORAGE and READ_EXTERNAL_STORAGE

system permission. These permissions are added to the AndroidManifest.xml file. Add these permissions just after the package name.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.externalstrorage">

   <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
   <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE"  />

   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:roundIcon="@mipmap/ic_launcher_round"
      android:supportsRtl="true"
      android:theme="@style/Theme.ExternalStrorage">
      <activity
         android:name=".ViewData"
         android:exported="false"                                    />
      <activity
         android:name=".MainActivity"
         android:exported="true">
         <intent-filter>
            <action      android:name="android.intent.action.MAIN"      />

            <category
android:name="android.intent.category.LAUNCHER"                      />
         </intent-filter>
      </activity>
   </application>

</manifest>
```

Once we have added the permission open activity_main.xml file from \res\layout folder path and write the code as shown below.

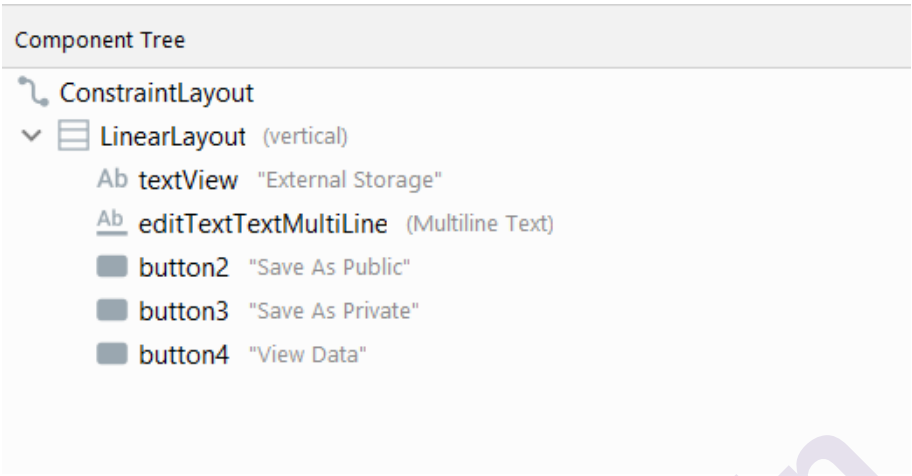
activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
```

```
android:layout_height="match_parent"
tools:context=".ViewData">

<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"      android:layout_marginLeft="20sp"
   android:layout_marginRight="20sp">

   <TextView
      android:id="@+id/textView3"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Retreiving          Saved          Text          Data"
      android:layout_marginTop="30dp"
      android:layout_marginBottom="30dp"/>

   <TextView
      android:id="@+id/textView4"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="TextView"
      android:layout_marginTop="30dp"
      android:layout_marginBottom="30dp"/>

   <Button
      android:id="@+id/button"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Retreive                Private                data"
      android:layout_marginTop="30dp"
      android:layout_marginBottom="30dp"
      android:onClick="showPrivateData"/>

   <Button
      android:id="@+id/button5"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Retreive                Public                Data"
      android:layout_marginTop="30dp"
      android:layout_marginBottom="30dp"
      android:onClick="showPublicData"/>

   <Button
      android:id="@+id/button6"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Home"
      android:layout_marginTop="30dp"
      android:layout_marginBottom="30dp"
```

```
        android:onClick="back"/>
    </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Component Tree:



Create a new Empty Activity:

Now we will create another empty activity by right clicking on App folder on the top->new->Activity->EmptyActivity and name it as ViewData.

We use this activity to display the saved data from the external storage.

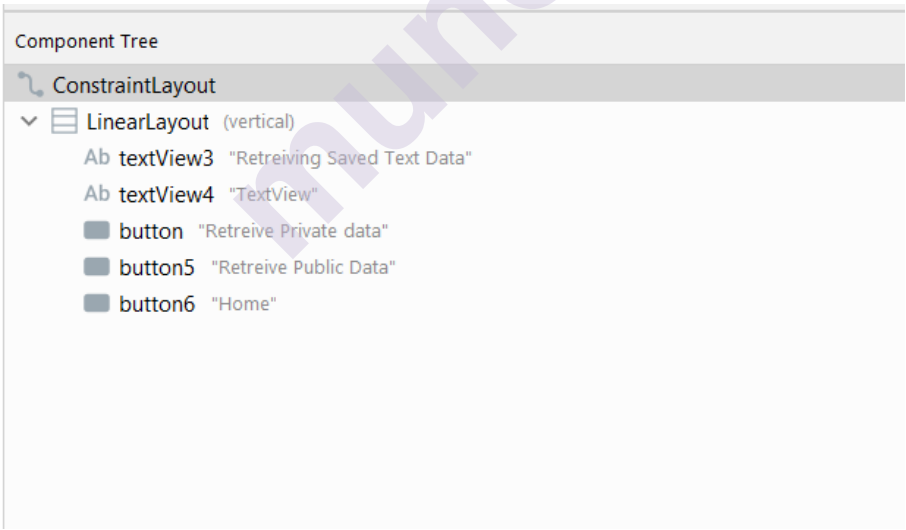
Code for Activity_View_Data.xml

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:app="http://schemas.android.com/apk/res-auto"

    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    tools:context=".ViewData">


    <LinearLayout

        android:layout_width="match_parent"

        android:layout_height="match_parent"

        android:orientation="vertical" android:layout_marginLeft="20sp"

        android:layout_marginRight="20sp">
```

```
<TextView
    android:id="@+id/textView3"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Retreiving Saved Text Data"
    android:layout_marginTop="30dp"
    android:layout_marginBottom="30dp"/>

<TextView
    android:id="@+id/textView4"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="TextView"
    android:layout_marginTop="30dp"
    android:layout_marginBottom="30dp"/>

<Button
    android:id="@+id/button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Retreive Private data"
    android:layout_marginTop="30dp"
    android:layout_marginBottom="30dp"
    android:onClick="showPrivateData"/>

<Button
    android:id="@+id/button5"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

```
        android:text="Retreive Public Data"
        android:layout_marginTop="30dp"
        android:layout_marginBottom="30dp"
        android:onClick="showPublicData"/>

    <Button
        android:id="@+id/button6"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Home"
        android:layout_marginTop="30dp"
        android:layout_marginBottom="30dp"
        android:onClick="back"/>
  </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Component Tree:



MainActivity.java:

```
package com.example.externalstrorage;

import static android.os.Environment.DIRECTORY_DOWNLOADS;
```

```
import androidx.appcompat.app.AppCompatActivity;

import android.Manifest;
import android.content.Intent;
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;
import androidx.core.app.ActivityCompat;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

public class MainActivity extends AppCompatActivity {
   private int EXTERNAL_STORAGE_PERMISSION_CODE = 23;
   EditText editText;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      editText = (EditText) findViewById(R.id.editTextTextMultiLine);



   }
   public void savePublicly(View view) {
      // Requesting Permission to access External Storage
      ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.READ_EXTERNAL_STORAGE},
            EXTERNAL_STORAGE_PERMISSION_CODE);
      String editTextData = editText.getText().toString();
      // getExternalStoragePublicDirectory() represents root of external
storage, we are using DOWNLOADS
```

```
    // We can use following directories: MUSIC, PODCASTS, ALARMS, RINGTONES, NOTIFICATIONS, PICTURES, MOVIES
    File folder = Environment.getExternalStoragePublicDirectory(DIRECTORY_DOWNLOADS);

    // Storing the data in file with name as sabaext.txt
    File file = new File(folder, "sabaext.txt");
    writeTextData(file, editTextData);
    editText.setText("");
  }
  public void savePrivately(View view) {
    String editTextData = editText.getText().toString();

    // Creating folder with name ExternalStorage
    File folder = getExternalFilesDir("ExternalStorage");

    // Creating file with name xyz.txt
    File file = new File(folder, "xyz.txt");
    writeTextData(file, editTextData);
    editText.setText("");
  }
  public void viewInformation(View view) {
    // Creating an intent to start a new activity
    Intent intent = new Intent(MainActivity.this, ViewData.class);
    startActivity(intent);
  }
  private void writeTextData(File file, String data) {
    FileOutputStream fileOutputStream = null;
    try {
      fileOutputStream = new FileOutputStream(file);
      fileOutputStream.write(data.getBytes());
```

```
            Toast.makeText(this, "Done" + file.getAbsolutePath(),
Toast.LENGTH_SHORT).show();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (fileOutputStream != null)
            {
                try {
                    fileOutputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

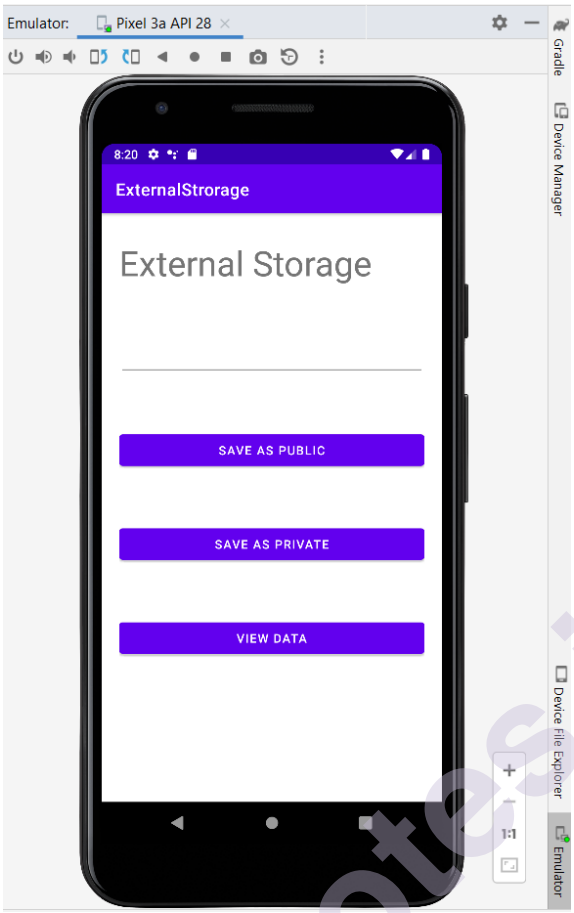
ViewData.java:

```
package com.example.externalstrorage;

import static android.os.Environment.DIRECTORY_DOWNLOADS;

import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.widget.TextView;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class ViewData extends AppCompatActivity {
    TextView textView;
```

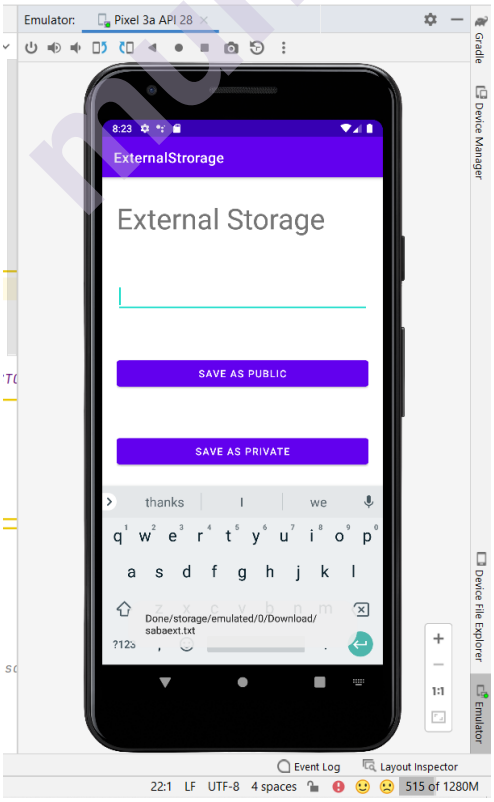
```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_view_data);
        textView = (TextView) findViewById(R.id.textView4);


    }
    public void showPublicData(View view) {
        // Accessing the saved data from the downloads folder
        File folder =
Environment.getExternalStoragePublicDirectory(DIRECTORY_DOWNL
OADS);


        // sabaext represent the file data that is saved publicly
        File file = new File(folder, "sabaext.txt");
        String data = getdata(file);
        if (data != null) {
            textView.setText(data);
        } else {
            textView.setText("No Data Found");
        }
    }
    public void showPrivateData(View view) {


        // ExternalStorage represent the folder name to access privately saved
data
        File folder = getExternalFilesDir("ExternalStorage");


        // xyz.txt is the file that is saved privately
        File file = new File(folder, "xyz.txt");
        String data = getdata(file);
        if (data != null) {
            textView.setText(data);
        } else {
            textView.setText("No Data Found");
        }
```

```
    }
    public void back(View view) {
        Intent intent = new Intent(ViewData.this, MainActivity.class);
        startActivity(intent);
    }
    // getdata() is the method which reads the data
    // the data that is saved in byte format in the file
    private String getdata(File myfile) {
        FileInputStream fileInputStream = null;
        try {
            fileInputStream = new FileInputStream(myfile);
            int i = -1;
            StringBuffer buffer = new StringBuffer();
            while ((i = fileInputStream.read()) != -1) {
                buffer.append((char) i);
            }
            return buffer.toString();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (fileInputStream != null) {
                try {
                    fileInputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return null;
    }
}
```

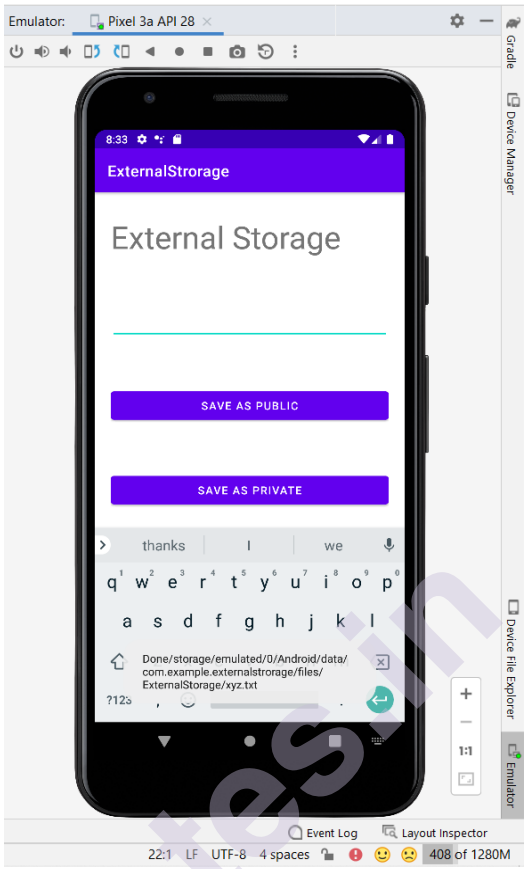
3.2.5 Output:



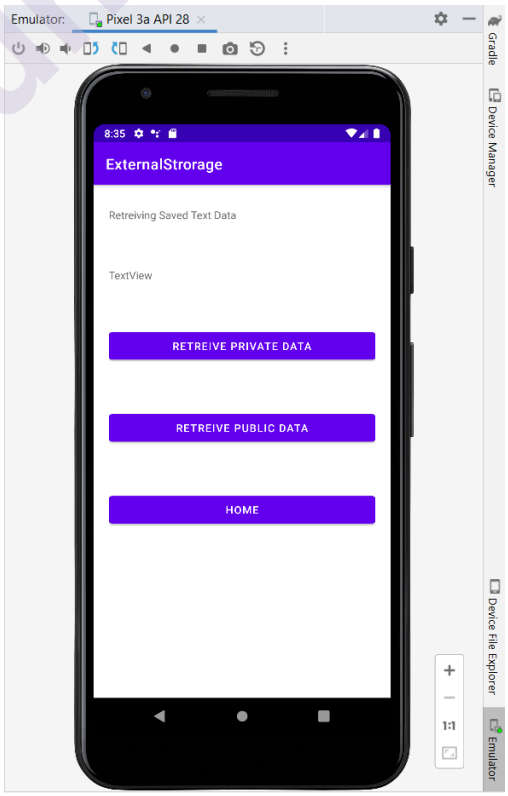
when Clicked on Save as Public button:



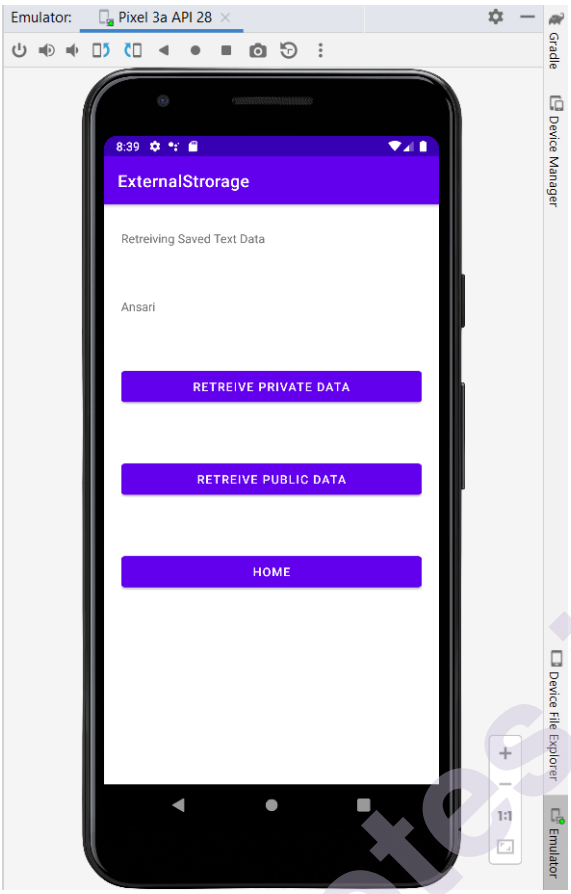
When Clicked on Save As Private Button:



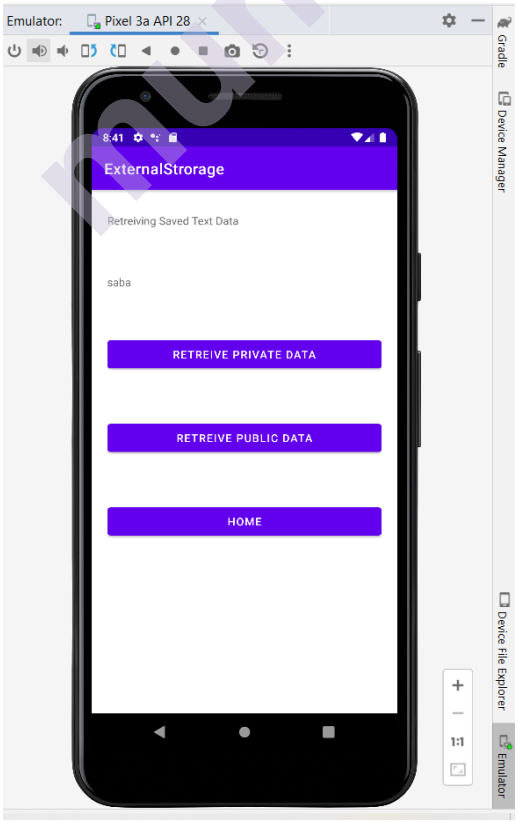
When clicked on View Data button is clicked another activity that view data activity is started.



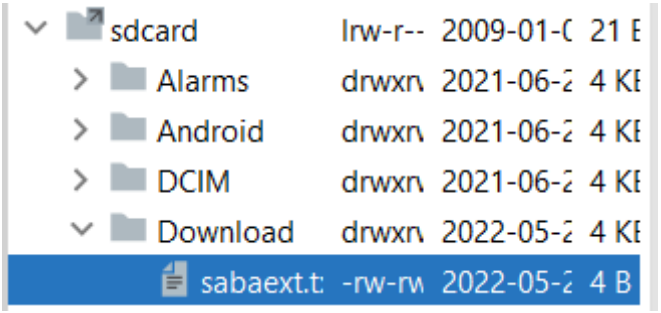
When clicked on retrieve Private data Button:



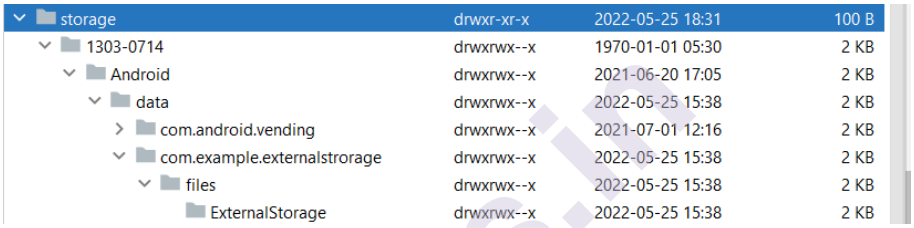
When we clicked on Retrieve public data button:



To View data saved as publicly click on Device File Explorer->SD Card->Downloads and you will see sabaext.txt file.



To View data saved as privately click on Device File Explorer->Storage->Android->Data->com.examples.externalstorage(your Application folder) ->Files. In files folder you can see folder is created



3.3 PRACTICAL NO 3: SHARED PREFERENCES

3.3.1 Aim:

Creating android program to demonstrate the use of Shared preferences.

3.3.2 Objective:

The main objective of this practical is to study and understand the concept of shared preferences in android.

3.3.3 Theory:

Shared Preferences is one the way of storing data of an application. Shared Preferences is used to save and retrieve data in the form of key, value pair.

In order to use shared preferences, we have to call a method getSharedPreferences() that returns a SharedPreference instance pointing to the file that contains the values of preferences.

SharedPreferences sharedpreferences = getSharedPreferences(MyPREFERENCES, Context.MODE_PRIVATE);

The first parameter is the key and the second parameter is the MODE.

The various Modes that are available are as follows:

1. **MODE_APPEND:** This will append the new preferences with the already existing preferences.

2. **MODE_ENABLE_WRITE_AHEAD_LOGGING**: Database open flag. When it is set, it would enable write ahead logging by default.
3. **MODE_MULTI_PROCESS:** This method will check for modification of preferences even if the shared preference instance has already been loaded.
4. **MODE_PRIVATE:** In this mode, the file can only be accessed using calling application.
5. **MODE_WORLD_READABLE:** This mode allows other application to read the preferences
6. **MODE_WORLD_WRITEABLE:** This mode allows other application to write the preferences

You can save something in the sharedpreferences by using SharedPreferences.Editor class.

3.3.4 Program:

Create a new android application using android studio and give name as sharedpreferences. Once we create an application, open activity_main.xml file from \res\layout folder path and write the code as shown below.

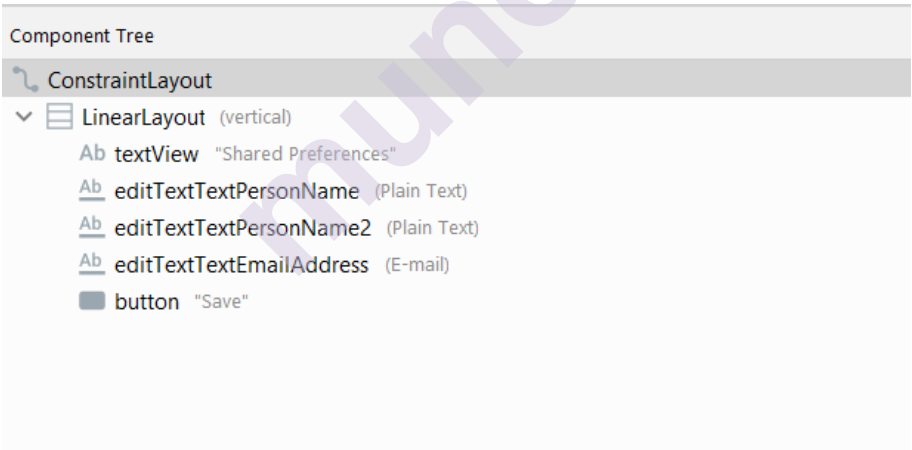
Activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:layout_marginLeft="30sp"
        android:layout_marginRight="30sp">
```

```
<TextView
    android:id="@+id/textView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Shared Preferences"
    android:textSize="40sp"
    android:textStyle="bold"
    android:textColor="@color/black"/>

<EditText
    android:id="@+id/editTextTextPersonName"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:layout_marginBottom="20dp"
    android:ems="10"
    android:hint="Enter your Name"
    android:inputType="textPersonName" />

<EditText
    android:id="@+id/editTextTextPersonName2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"
    android:hint="Enter your Age"
    android:layout_marginTop="20dp"
    android:layout_marginBottom="20dp"/>

<EditText
    android:id="@+id/editTextTextEmailAddress"
    android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textEmailAddress"
        android:hint="Enter Your Email"
        android:layout_marginTop="20dp"
        android:layout_marginBottom="20dp"/>

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Save"
        android:layout_marginTop="20dp"
        android:layout_marginBottom="20dp"/>
  </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Component tree:



MainActivity.java:

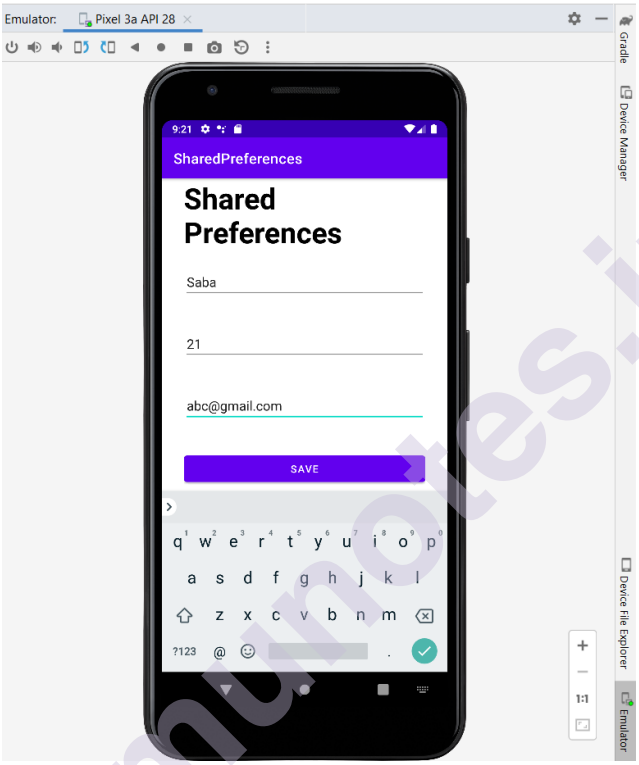
```
package com.example.sharedpreferences;
import androidx.appcompat.app.AppCompatActivity;
import android.content.Context;
import android.os.Bundle;
import android.content.SharedPreferences;
import android.view.View;
```

```
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
   private EditText name, age,email;
   Button b1;
   public static final String MyPREFERENCES = "MyPrefsData" ;
   public static final String Name = "nameKey";
   public static final String Age = "ageKey";
   public static final String Email = "emailKey";
   SharedPreferences sharedpreferences;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      name =(EditText) findViewById(R.id.editTextTextPersonName);
      age = (EditText)findViewById(R.id.editTextTextPersonName2);
      email=(EditText) findViewById(R.id.editTextTextEmailAddress);
      b1=(Button) findViewById(R.id.button);
      sharedpreferences = getSharedPreferences(MyPREFERENCES,
Context.MODE_PRIVATE);

      b1.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            String n  = name.getText().toString();
            String a  = age.getText().toString();
            String e  = email.getText().toString();

            SharedPreferences.Editor editor = sharedpreferences.edit();
            editor.putString(Name, n);
            editor.putString(Age, a);
            editor.putString(Email, e);
            editor.commit();
```

```
Toast.makeText(MainActivity.this,"Thanks",Toast.LENGTH_LONG).sho
w();
            }
        });
    }
}
```

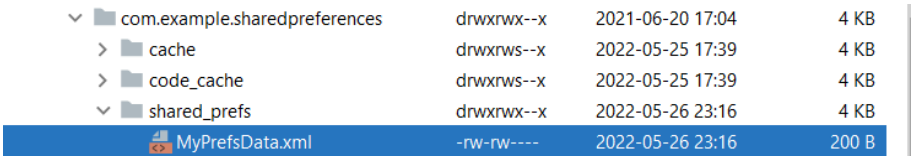
3.3.5 Output:

Step1: User enters the data in text field



Now when you press save button, the text will be saved in the shared preferences. Now press back button and exit the application.

To Check Shared preferences, click on



Device File Explorer->Data->Data->Com.example.sharedpreferences(your application folder)-> shared_prefs->MyPrefsData.xml

Content of MyPrefsData.xml:

MyPrefsData.xml

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="ageKey">21</string>
    <string name="nameKey">saba</string>
    <string name="emailKey">abc@gmail.com</string>
</map>
```

3.4 PRACTICAL NO 4: CONTENT PROVIDER

3.4.1 Aim:

Creating android program to demonstrate the use of Content Provider.

3.4.2 Objective:

The main objective of this practical is to study and understand the concept of Content Provider in android.

3.4.3 Theory:

Android provide very important component to serves the purpose of a relational database to store the data of applications called content provider. The role of the content provider in the android system is like a central repository in which data of the applications are stored, and it enables other applications to securely access and modifies that data based on the user requirements.

Android system allows the content provider to use different ways to store application data and the data can be stored in a database, in files, or even over a network.

Content providers allows you to centralize content in one place and many different applications can access it as needed. A content provider behaves just like a database where you can query it, edit its content, as well as add or delete content using insert (), update(), delete (), and query () methods. In most cases this data is stored in an SQLite database.

Your content provider class must extends ContentProvider class and must implement a standard set of APIs that enable other applications to perform transactions.

The following are the six methods of ContentProvider class that has to be overridden by your content provider class:

1. onCreate() This method is called when the provider is started.
2. query() This method receives a request from a client. The result is returned as a Cursor object.
3. insert()This method inserts a new record into the content provider.

4. delete() This method deletes an existing record from the content provider.
5. update() This method updates an existing record from the content provider.
6. getType() This method returns the MIME type of the data at the given URI.

3.4.4 Program:

Create a new project: Click on File, then New => New Project->Empty Activity.Save project as contentprovider.

Creating the Content Provider class:

1. Click on File, then New => Other => ContentProvider.
2. Name the ContentProvider.In this practical we have named it as MyStudentProvider.
3. Define authority (it can be anything for example in this practical it is "**com.example.contentprovider.student**")
4. Select Exported and Enabled option
5. Choose the language as Java

This class extends the ContentProvider base class and override the six abstract methods. Below is the complete code to define a content provider.

MyStudentProvider.java:

```
package com.example.contentprovider;


import android.content.ContentProvider;

import android.content.ContentValues;

import android.database.Cursor;

import android.net.Uri;

import android.content.ContentUris;

import android.content.Context;

import android.content.UriMatcher;

import android.database.sqlite.SQLiteDatabase;

import android.database.sqlite.SQLiteException;

import android.database.sqlite.SQLiteOpenHelper;
```

```
import android.database.sqlite.SQLiteQueryBuilder;
import java.util.HashMap;
public class MyStudentProvider extends ContentProvider {
   public MyStudentProvider() {
   }
   static final String PROVIDER_NAME =
"com.example.contentprovider.student";
   static final String URL = "content://" + PROVIDER_NAME + "/users";

   // parsing the content URI
   static final Uri CONTENT_URI = Uri.parse(URL);

   static final String id = "id";
   static final String name = "name";
   static final int uriCode = 1;
   static final UriMatcher uriMatcher;
   private static HashMap<String, String> values;

   static {

      // to match the content URI
      // every time user access table under content provider
      uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);

      // to access whole table
      uriMatcher.addURI(PROVIDER_NAME, "users", uriCode);

      // to access a particular row
      // of the table
      uriMatcher.addURI(PROVIDER_NAME, "users/*", uriCode);
```

```
    }
    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        int count = 0;
        switch (uriMatcher.match(uri)) {
            case uriCode:
                count = db.delete(TABLE_NAME, selection, selectionArgs);
                break;
            default:
                throw new IllegalArgumentException("Unknown URI " + uri);
        }
        getContext().getContentResolver().notifyChange(uri, null);
        return count;
    }

    @Override
    public String getType(Uri uri) {
        // TODO: Implement this to handle requests for the MIME type of the
data
        // at the given URI.
       // throw new UnsupportedOperationException("Not yet
implemented");
        switch (uriMatcher.match(uri)) {
            case uriCode:
                return "vnd.android.cursor.dir/users";
            default:
                throw new IllegalArgumentException("Unsupported URI: " +
uri);
        }
    }
```

```
    @Override
    public Uri insert(Uri uri, ContentValues values) {
        long rowID = db.insert(TABLE_NAME, "", values);
        if (rowID > 0) {
            Uri _uri = ContentUris.withAppendedId(CONTENT_URI, rowID);
            getContext().getContentResolver().notifyChange(_uri, null);
            return _uri;
        }
        throw new SQLiteException("Failed to add a record into " + uri);
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
                      String[] selectionArgs) {
        int count = 0;
        switch (uriMatcher.match(uri)) {
            case uriCode:
                count = db.update(TABLE_NAME, values, selection, selectionArgs);
                break;
            default:
                throw new IllegalArgumentException("Unknown URI " + uri);
        }
        getContext().getContentResolver().notifyChange(uri, null);
        return count;
    }

    @Override
```

```
public boolean onCreate() {
   Context context = getContext();
   DatabaseHelper dbHelper = new DatabaseHelper(context);
   db = dbHelper.getWritableDatabase();
   if (db != null) {
      return true;
   }
   return false;
}

@Override
public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {
   SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
   qb.setTables(TABLE_NAME);
   switch (uriMatcher.match(uri)) {
      case uriCode:
         qb.setProjectionMap(values);
         break;
      default:
         throw new IllegalArgumentException("Unknown URI " + uri);
   }
   if (sortOrder == null || sortOrder == "") {
      sortOrder = id;
   }
   Cursor c = qb.query(db, projection, selection, selectionArgs, null,
         null, sortOrder);
   c.setNotificationUri(getContext().getContentResolver(), uri);
   return c;
}

private SQLiteDatabase db;
```

```
// declaring name of the database
static final String DATABASE_NAME = "UserDB";

// declaring table name of the database
static final String TABLE_NAME = "Users";

// declaring version of the database
static final int DATABASE_VERSION = 1;

// sql query to create the table
static final String CREATE_DB_TABLE = " CREATE TABLE " +
TABLE_NAME
        + " (id INTEGER PRIMARY KEY AUTOINCREMENT, "
        + " name TEXT NOT NULL);";

// creating a database
private static class DatabaseHelper extends SQLiteOpenHelper {

    // defining a constructor
    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null,
DATABASE_VERSION);
    }

    // creating a table in the database
    @Override
    public void onCreate(SQLiteDatabase db) {

        db.execSQL(CREATE_DB_TABLE);
    }

    @Override
```

```
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

        // sql query to drop a table
        // having similar name
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
    }
}
```

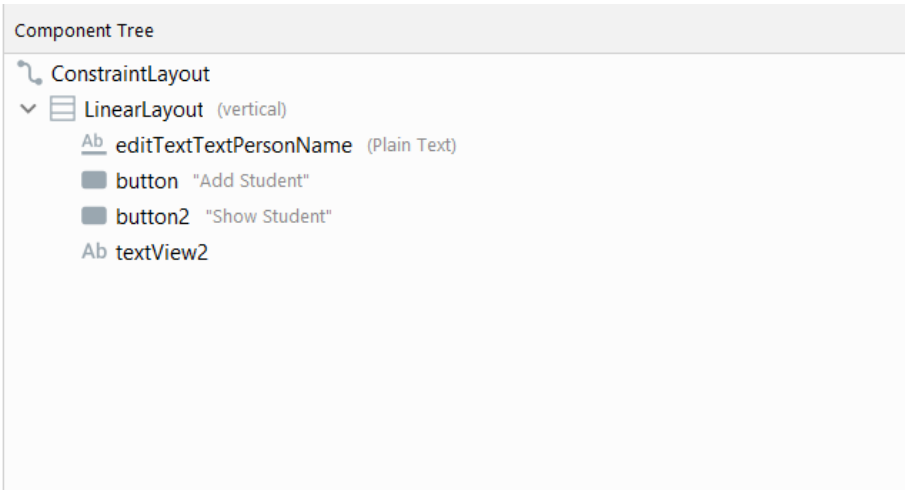
Design the activity_main.xml layout

Activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:layout_marginRight="30sp"
        android:layout_marginLeft="30sp">

        <EditText
            android:id="@+id/editTextTextPersonName"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ems="10"
            android:inputType="textPersonName"
```

```
        android:hint="Enter Student Name"
        android:layout_marginBottom="20dp"
        android:layout_marginTop="20dp"
        tools:ignore="Autofill,HardcodedText" />

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Add Student"
        android:layout_marginBottom="20dp"
        android:layout_marginTop="20dp"
        android:onClick="onClickAddDetails"
        tools:ignore="HardcodedText,UsingOnClickInXml" />

    <Button
        android:id="@+id/button2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Show Student"
        android:layout_marginBottom="20dp"
        android:layout_marginTop="20dp"
        android:onClick="onClickShowDetails"
        tools:ignore="HardcodedText,UsingOnClickInXml" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:clickable="false"/>
  </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Component Tree:



Modify the MainActivity file:

mainActivity.java

```
package com.example.contentprovider;


import androidx.appcompat.app.AppCompatActivity;


import android.annotation.SuppressLint;
import android.os.Bundle;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.net.Uri;
import android.view.MotionEvent;
import android.view.View;
import android.view.inputmethod.InputMethodManager;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {


    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        InputMethodManager imm = (InputMethodManager)getSystemService(Context.INPUT_METHOD_SERVICE);

imm.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(), 0);
        return true;
    }
    public void onClickAddDetails(View view) {
        // class to add values in the database
        ContentValues values = new ContentValues();

        // fetching text from user
        values.put(MyStudentProvider.name, ((EditText) findViewById(R.id.editTextTextPersonName)).getText().toString());

        // inserting into database through content URI
        getContentResolver().insert(MyStudentProvider.CONTENT_URI, values);

        // displaying a toast message
        Toast.makeText(getBaseContext(), "New Record Inserted", Toast.LENGTH_LONG).show();
    }

    @SuppressLint({"SetTextI18n", "Range"})
    public void onClickShowDetails(View view) {

        // inserting complete table details in this text field
```

```
        TextView resultView= findViewById(R.id.textView2);


        // creating a cursor object of the
        // content URI
        @SuppressLint("Recycle") Cursor cursor =
getContentResolver().query(Uri.parse("content://com.example.contentpro
vider.student/users"), null, null, null, null);


        // iteration of the cursor
        // to print whole table
        if(cursor.moveToFirst()) {
            StringBuilder strBuild=new StringBuilder();
            while (!cursor.isAfterLast()) {

strBuild.append("\n").append(cursor.getString(cursor.getColumnIndex("id
"))).append("-
").append(cursor.getString(cursor.getColumnIndex("name")));
                cursor.moveToNext();
            }
            resultView.setText(strBuild);
        }
        else {
            resultView.setText("No Records Found");
        }
    }
}
```

Create another application to access the Content Provider:

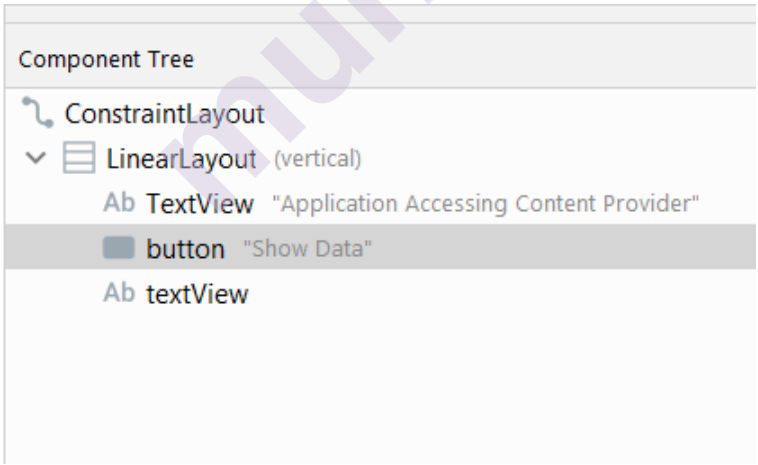
1. Create a new Project
2. Click on File, then New => New Project.
3. Select language as Java.
4. Choose empty activity as a template
5. Select the minimum SDK as per your need.
6. Name this application as accessingcontentprovider

Activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:layout_marginLeft="30sp"
        android:layout_marginRight="30sp">

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Application Accessing Content Provider"
            android:textSize="30sp"
            android:textColor="@color/black"
            android:textStyle="bold"
            android:layout_marginTop="20dp"
            android:layout_marginBottom="20dp"
            tools:ignore="HardcodedText" />

        <Button
            android:id="@+id/button"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Show Data"
```

```
        android:onClick="onClickShowDetails"
        android:layout_marginTop="20dp"
        android:layout_marginBottom="20dp"
        tools:ignore="HardcodedText" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:layout_marginBottom="20dp"
        android:textSize="20sp"
        android:textColor="@color/black"
        android:textStyle="bold"/>
  </LinearLayout>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Component Tree:



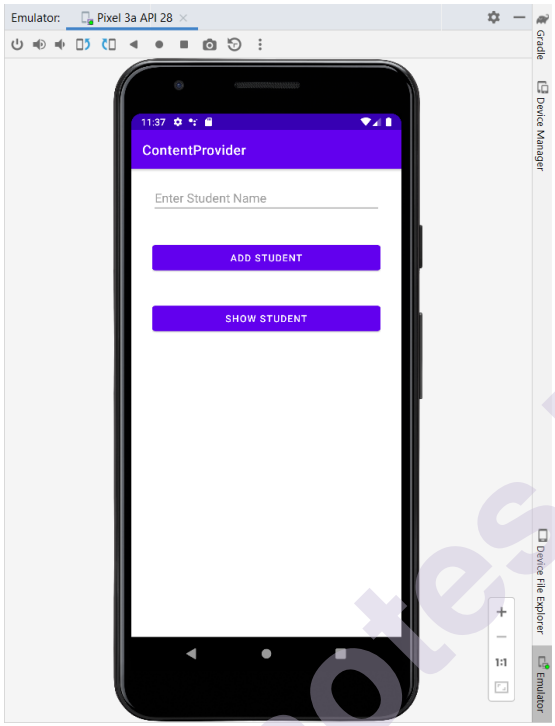
MainActivity.java:

```
package com.example.accessingcontentprovider;

import androidx.appcompat.app.AppCompatActivity;
```

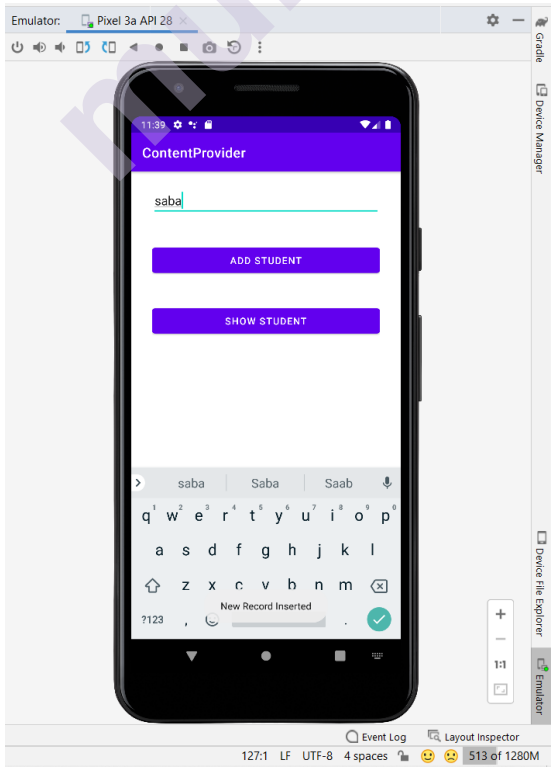
```
import android.annotation.SuppressLint;
import android.os.Bundle;
import android.database.Cursor;
import android.net.Uri;
import android.view.View;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
   @SuppressLint({"Range", "SetTextI18n"})
   public void onClickShowDetails(View view) {
      // inserting complete table details in this text field
      TextView resultView= (TextView) findViewById(R.id.textView);

      // creating a cursor object of the
      // content URI
      @SuppressLint("Recycle") Cursor cursor =
getContentResolver().query(Uri.parse("content://com.example.contentpro
vider.student/users"), null, null, null, null);

      // iteration of the cursor
      // to print whole table
      if(cursor.moveToFirst()) {
         StringBuilder strBuild=new StringBuilder();
         while (!cursor.isAfterLast()) {

strBuild.append("\n").append(cursor.getString(cursor.getColumnIndex("id
"))).append("-
").append(cursor.getString(cursor.getColumnIndex("name")));
            cursor.moveToNext();
         }
         resultView.setText(strBuild);
      }
      else {
```

```
            resultView.setText("No Records Found");
        }
    }
}
```

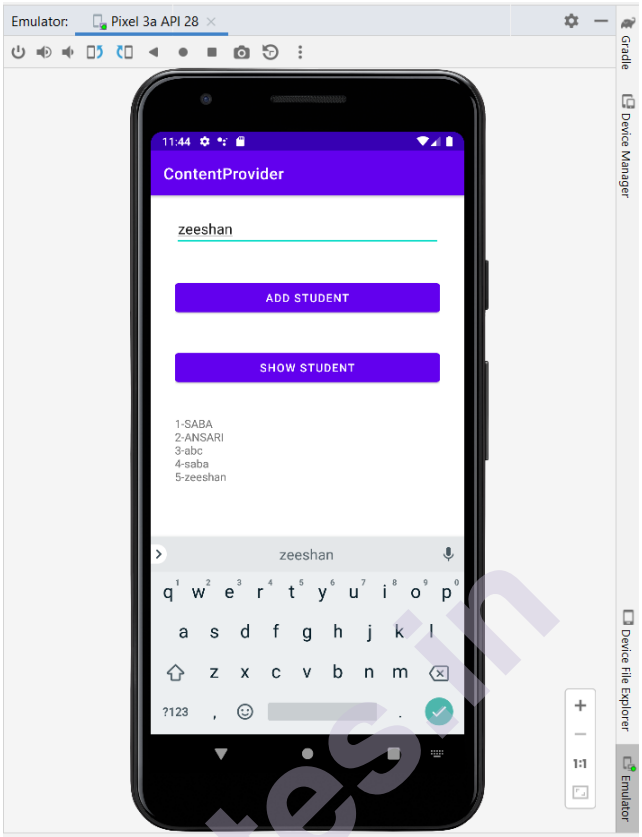
3.4.5 Output: First Run Content Provider:



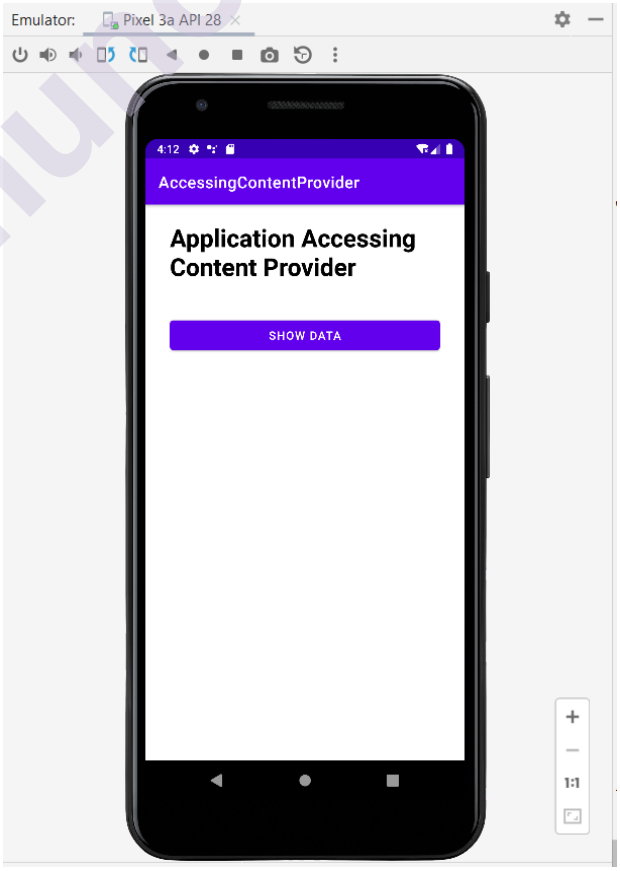
Application:



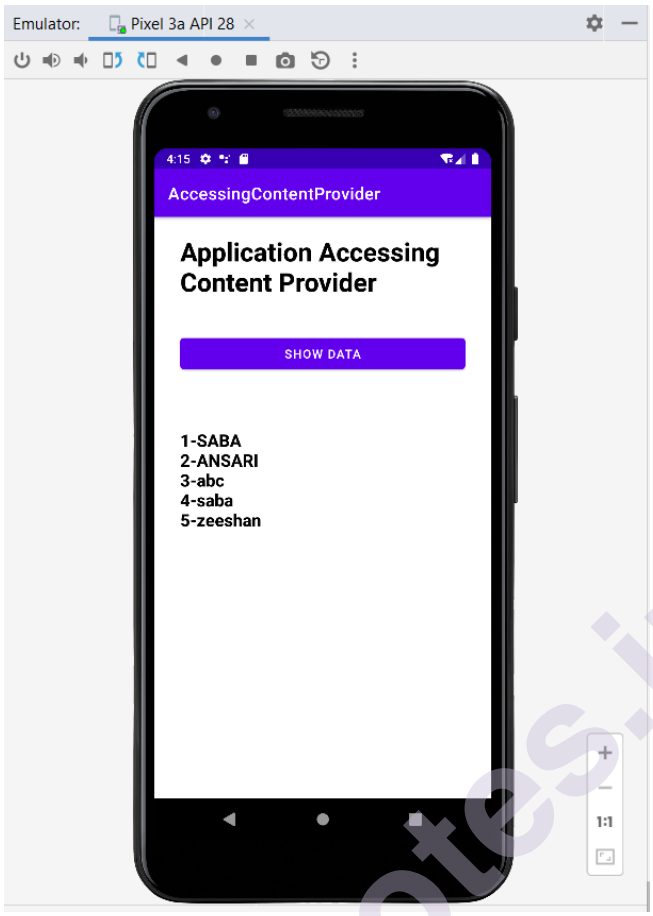
When clicked on Show Student Button:



Now Run AccessContent provider application:



When Show Data button is clicked following output is generated:



3.5 PRACTICAL NO 5: CRUD OPERATION USING SQLITE

3.5.1 Aim:

Creating android program to perform CRUD operation on SQLite database.

3.5.2 Objective:

The main objective of this practical is to study and understand the concept of SQlite database in android.

3.5.3 Theory:

SQLite is an open-source relational database. It is used to perform database operations such as storing, manipulating or retrieving persistent data from the database on android devices.

It is embedded in android by default. So, there is no need to perform any database setup or administration task.

SQLiteOpenHelper class provides the functionality to use the SQLite database.

It is used for database creation and version management. For performing any database operation, you have to override the methods onCreate () and onUpgrade () methods of SQLiteOpenHelper class.

SQLiteDatabase class contains methods to be performed on SQLite database such as create, update, delete, select etc.

Some of the methods of SQLiteDatabase class are as follows:

1. **void execSQL(String sql):** executes the sql query not select query.
2. **long insert (String table, String nullColumnHack, ContentValues values):** This method inserts a record in the database. The table specifies the table name, nullColumnHack doesn't allow completely null values. If second argument is null, android will store null values if values are empty. The third argument specifies the values to be stored.
3. **int update(String table, ContentValues values, String whereClause, String[] whereArgs):** it will updates a row.
4. **Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy):** returns a cursor over the resultset.

Whenever an application needs to store large amount of data then using sqlite is more preferable than other repository system like SharedPreferences or saving data in files.

3.5.4 Program:

Create a New Project and Name it CRUDOperations.

Open res -> layout -> activity_main.xml (or) main.xml and add following code:

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"

   xmlns:app="http://schemas.android.com/apk/res-auto"

   xmlns:tools="http://schemas.android.com/tools"

   android:layout_width="match_parent"

   android:layout_height="match_parent"

   tools:context=".MainActivity">
```

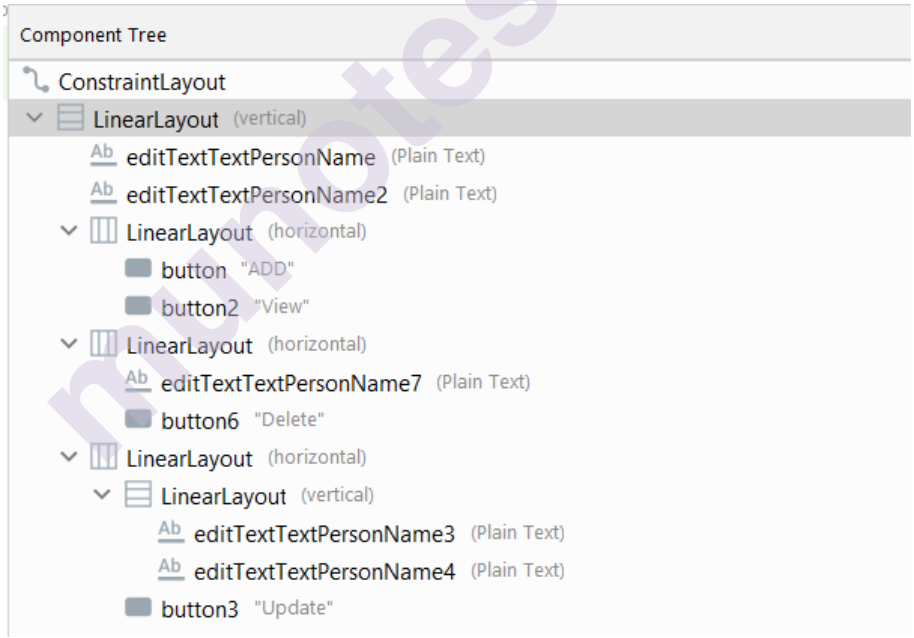
```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" android:layout_marginLeft="30sp"
    android:layout_marginRight="30sp">

    <EditText
        android:id="@+id/editTextTextPersonName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Enter User Name"
        android:inputType="textPersonName"
        android:layout_marginTop="40sp"
        android:layout_marginBottom="20sp"/>

    <EditText
        android:id="@+id/editTextTextPersonName2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Enter Password"
        android:inputType="textPersonName"
        android:layout_marginTop="20sp"
        android:layout_marginBottom="20sp"
        />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_marginTop="40sp"
        android:layout_marginBottom="40sp">
```

```
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="ADD"
        android:onClick="addUser"
        />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="View"
        android:onClick="viewdata"/>

</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_marginTop="40sp"
    android:layout_marginBottom="40sp">

    <EditText
        android:id="@+id/editTextTextPersonName7"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:ems="10"
        android:hint="Enter User Name to be Deleted"
        android:inputType="textPersonName"
         />
```

```
    <Button
        android:id="@+id/button6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Delete"
        android:onClick="delete"/>
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="200dp"
    android:layout_weight="1"
    android:orientation="horizontal"
    android:layout_marginTop="40sp"
    android:layout_marginBottom="40sp">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <EditText
            android:id="@+id/editTextTextPersonName3"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ems="10"
            android:inputType="textPersonName"
            android:hint="Enter Current Username" />

        <EditText
            android:id="@+id/editTextTextPersonName4"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ems="10"
```

```
            android:inputType="textPersonName"
            android:hint="Enter New User Name" />
        </LinearLayout>

        <Button
            android:id="@+id/button3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Update" android:layout_marginTop="30sp"
            android:onClick="update"/>
    </LinearLayout>

  </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Component tree:



Now open app -> java -> package -> MainActivity.java and add the below code.

MainActivity.java:

```
package com.example.crudoperation;


import androidx.appcompat.app.AppCompatActivity;
```

```
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;


public class MainActivity extends AppCompatActivity {
   EditText Name, Pass , updateold, updatenew, delete;
   MyDatabaseAdapter helper;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      Name= (EditText) findViewById(R.id.editTextTextPersonName);
      Pass= (EditText) findViewById(R.id.editTextTextPersonName2);
      updateold=                                          (EditText)
findViewById(R.id.editTextTextPersonName3);
      updatenew=                                          (EditText)
findViewById(R.id.editTextTextPersonName4);
      delete = (EditText) findViewById(R.id.editTextTextPersonName7);


      helper = new MyDatabaseAdapter(this);
   }

   public void addUser(View view)
   {
      String t1 = Name.getText().toString();
      String t2 = Pass.getText().toString();
      if(t1.isEmpty() || t2.isEmpty())
      {
         Message.message(getApplicationContext(),"Enter Both Name and
Password");
      }
      else
      {
         long id = helper.insertData(t1,t2);
```

```
            if(id<=0)
            {
                Message.message(getApplicationContext(),"Insertion
Unsuccessful");
                Name.setText("");
                Pass.setText("");
            } else
            {
                Message.message(getApplicationContext(),"Insertion
Successful");
                Name.setText("");
                Pass.setText("");
            }
        }
    }
    public void viewdata(View view)
    {
        String data = helper.getData();
        Message.message(this,data);
    }

    public void update( View view)
    {
        String u1 = updateold.getText().toString();
        String u2 = updatenew.getText().toString();
        if(u1.isEmpty() || u2.isEmpty())
        {
            Message.message(getApplicationContext(),"Enter Data");
        }
        else
        {
            int a= helper.updateName( u1, u2);
            if(a<=0)
            {
                Message.message(getApplicationContext(),"Unsuccessful");
                updateold.setText("");
```

```
            updatenew.setText("");
        } else {
            Message.message(getApplicationContext(),"Updated");
            updateold.setText("");
            updatenew.setText("");
        }
    }

}
public void delete( View view)
{
    String uname = delete.getText().toString();
    if(uname.isEmpty())
    {
        Message.message(getApplicationContext(),"Enter Data");
    }
    else{
        int a= helper.delete(uname);
        if(a<=0)
        {
            Message.message(getApplicationContext(),"Unsuccessful");
            delete.setText("");
        }
        else
        {
            Message.message(this, "DELETED");
            delete.setText("");
        }
    }
}

}
```

Next step is to create a java class myDatabaseAdapter. Java

MyDatabaseAdapter.java:

```
package com.example.crudoperation;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;


  public class MyDatabaseAdapter {
    myDbHelper myhelper;
    public MyDatabaseAdapter(Context context)
    {
      myhelper = new myDbHelper(context);
    }

    public long insertData(String name, String pass)
    {
      SQLiteDatabase dbb = myhelper.getWritableDatabase();
      ContentValues contentValues = new ContentValues();
      contentValues.put(myDbHelper.NAME, name);
      contentValues.put(myDbHelper.MyPASSWORD, pass);
      long id = dbb.insert(myDbHelper.TABLE_NAME, null ,
contentValues);
      return id;
    }

    public String getData()
    {
      SQLiteDatabase db = myhelper.getWritableDatabase();
      String[] columns =
{myDbHelper.UID,myDbHelper.NAME,myDbHelper.MyPASSWORD};
      Cursor cursor
=db.query(myDbHelper.TABLE_NAME,columns,null,null,null,null,null);
      StringBuffer buffer= new StringBuffer();
```

```
        while (cursor.moveToNext())
        {
          int cid =cursor.getInt(cursor.getColumnIndex(myDbHelper.UID));
          String name =cursor.getString(cursor.getColumnIndex(myDbHelper.NAME));
          String password =cursor.getString(cursor.getColumnIndex(myDbHelper.MyPASSWORD) );
          buffer.append(cid+ "   " + name + "   " + password +" \n");
        }
        return buffer.toString();
    }

    public  int delete(String uname)
    {
        SQLiteDatabase db = myhelper.getWritableDatabase();
        String[] whereArgs ={uname};

        int count =db.delete(myDbHelper.TABLE_NAME ,myDbHelper.NAME+" = ?",whereArgs);
        return  count;
    }

    public int updateName(String oldName , String newName)
    {
        SQLiteDatabase db = myhelper.getWritableDatabase();
        ContentValues contentValues = new ContentValues();
        contentValues.put(myDbHelper.NAME,newName);
        String[] whereArgs= {oldName};
        int count =db.update(myDbHelper.TABLE_NAME,contentValues, myDbHelper.NAME+" = ?",whereArgs );
        return count;
    }

    static class myDbHelper extends SQLiteOpenHelper
    {
```

```
private static final String DATABASE_NAME = "myDatabase"; // Database Name

private static final String TABLE_NAME = "myTable";   // Table Name

private static final int DATABASE_Version = 1;    // Database Version

private static final String UID="_id";    // Column I (Primary Key)

private static final String NAME = "Name";    //Column II

private static final String MyPASSWORD= "Password";    // Column III

private static final String CREATE_TABLE = "CREATE TABLE "+TABLE_NAME+

        " ("+UID+" INTEGER PRIMARY KEY AUTOINCREMENT, "+NAME+" VARCHAR(255) ,"+ MyPASSWORD+" VARCHAR(225));";

private static final String DROP_TABLE ="DROP TABLE IF EXISTS "+TABLE_NAME;

private Context context;


public myDbHelper(Context context) {

   super(context, DATABASE_NAME, null, DATABASE_Version);

   this.context=context;

}


public void onCreate(SQLiteDatabase db) {


   try {

      db.execSQL(CREATE_TABLE);

   } catch (Exception e) {

      Message.message(context,""+e);

   }

}


@Override

public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

   try {
```

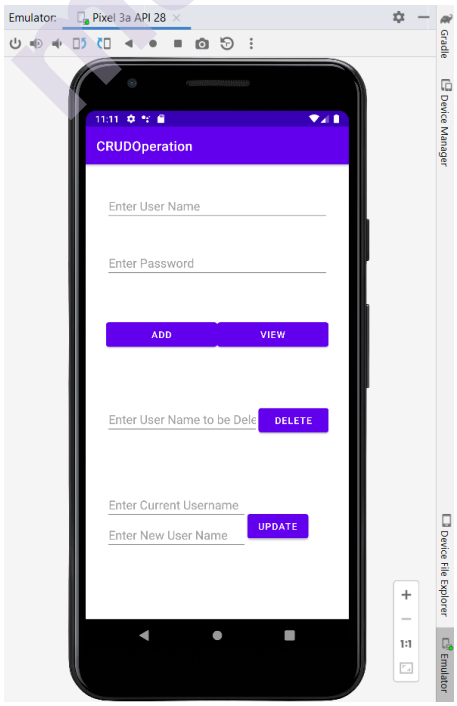
```
            Message.message(context,"OnUpgrade");
            db.execSQL(DROP_TABLE);
            onCreate(db);
        }catch (Exception e) {
            Message.message(context,""+e);
        }
    }
}
```

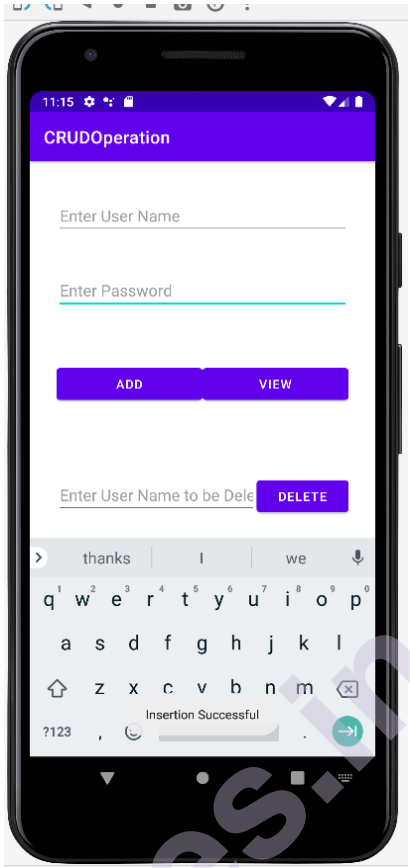
Next step is to create another java class Message.class

Message.java:

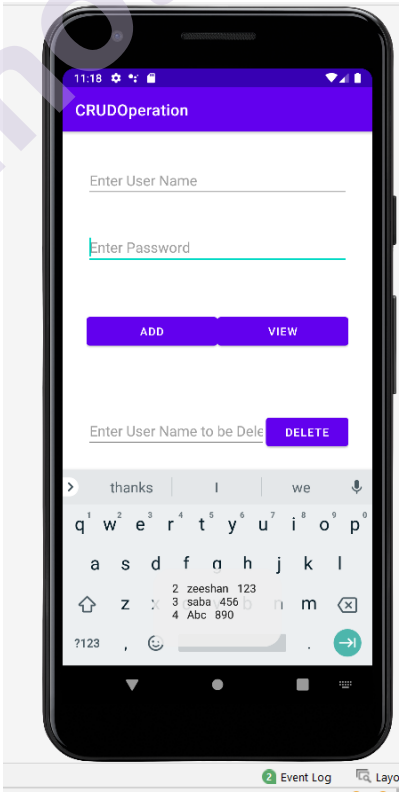
```
package com.example.crudoperation;

import android.content.Context;
import android.widget.Toast;

public class Message {
    public static void message(Context context, String message) {
        Toast.makeText(context, message, Toast.LENGTH_LONG).show();
    }
```

3.5.5 Output:

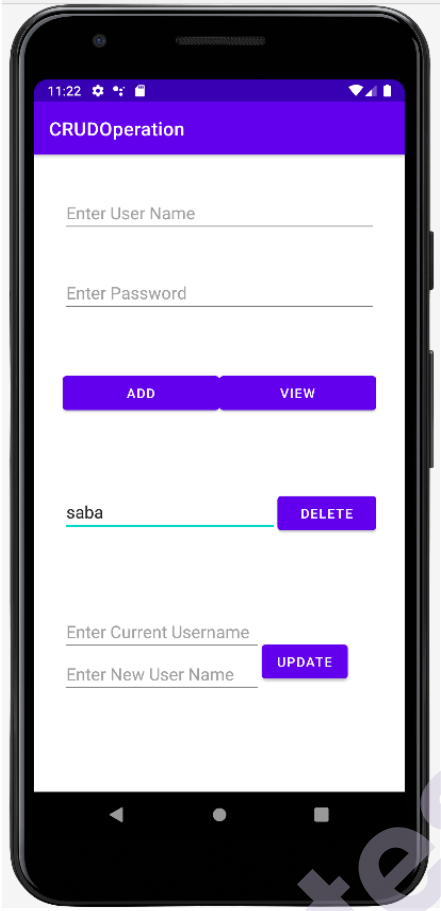




Above output is generated when Add button is clicked.

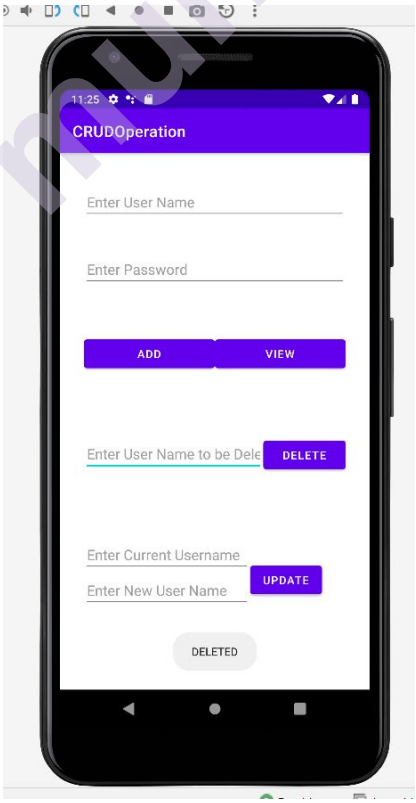


Above output is generated when View Button is pressed.

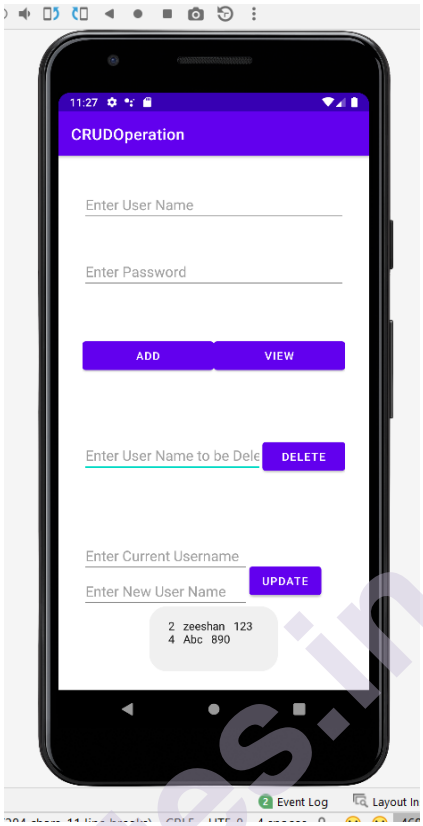


Entering user name saba to delete it.

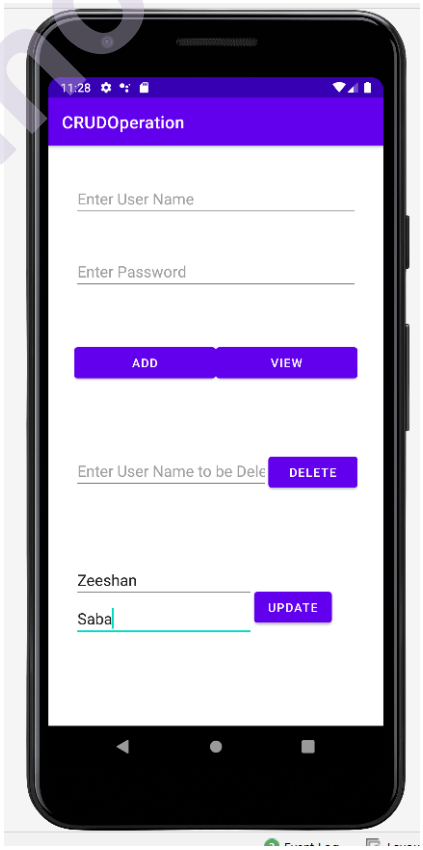
User name is Deleted.

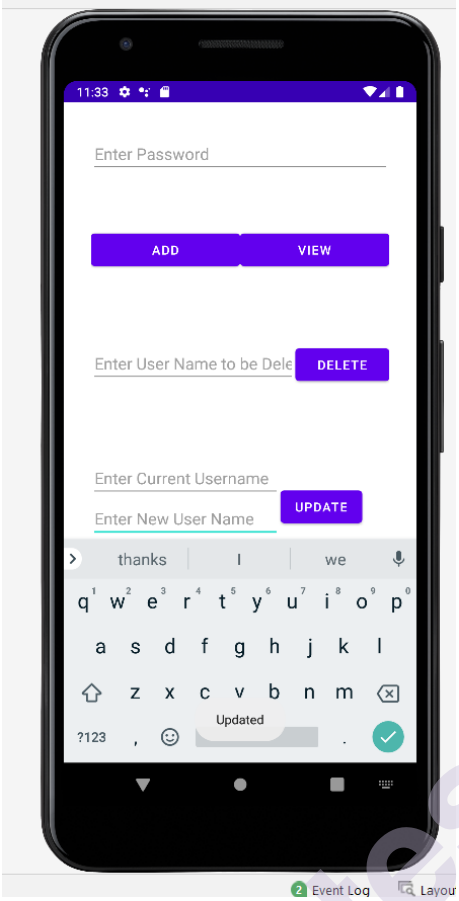


Now again View the data in the database.



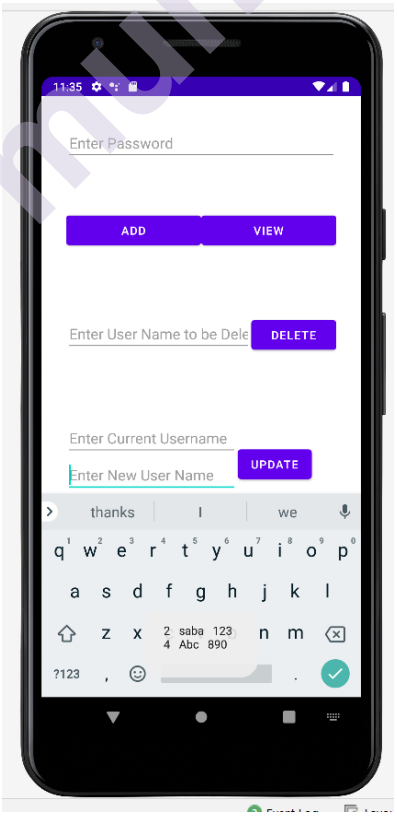
Next we will update the username Zeeshan with new user name saba





After clicking on update button

Again we will see the data by clicking on the update button



3.6 QUESTIONS

1. What is/are storage option available in android?

   A) SharedPreferences

   B) SQLiteDatabase

   C) ContentProvider

   D) All of the above.

2. Which objects stores the data only in key-value pair?

   A) SharedPreferences

   B) SQLiteDatabase

   C) ContentProvider

   D) None of the above.

3. To write files on the external storage, which permission you will write in AndroidManifest.xml file

   A) WRITE_STORAGE

   B) WRITE_EXTERNAL_DATA

   C) WRITE_EXTERNAL_STORAGE

   D) None of the above.

4. To check whether the media(external storage) is available, which method of Environment class you will use?

   A) getExternalStorageState()

   B) getExternalStorage()

   C) getExternalStateData()

   D) None of the above.

5. If you want share the data from one application to other applications,then which object you will use?

   A) SQLiteDatabases

   B) InternalStorage

   C) SharedPreferences

   D) ContentProvider

6. What is the full form of AVD in Android?

   A) Android Virtual Device

   B) Android Virtual Display

   C) Actual Virtual Display

   D) All of the above.

*****

4

INTRODUCTION TO GRAPHICS, ANIMATION AND MULTIMEDIA

Unit Structure

4.0 OBJECTIVE

One of the main goals and objectives of a graphic designer is **to design a project based on the individual needs of a client**. Prior to beginning the design phase of a project, a graphic designer speaks with a client to determine a project's overall goal, purpose and desired appearance.

The biggest use for animation is **for entertainment**. Animation is used on the TV, on your phones, and all over the internet. In television, animation is mostly used to occupy children, as is gives them something to laugh about and keep them entertained for long periods of time.

Multimedia **introduces how multimedia can be used in various application areas**. It provides a solid foundation to the students so that they can identify the proper applications of multimedia, evaluate the appropriate multimedia systems and develop effective multimedia applications.

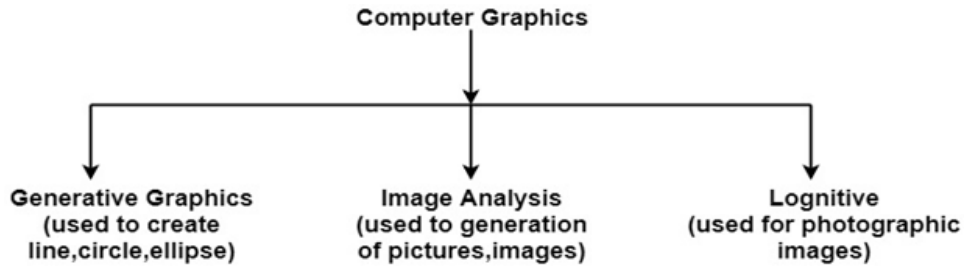
4.1 INTRODUCTION

It is the use of computers to create and manipulate pictures on a display device. It comprises of software techniques to create, store, modify, represents pictures.

Why computer graphics used?

Suppose a shoe manufacturing company want to show the sale of shoes for five years. For this vast amount of information is to store. So a lot of time and memory will be needed. This method will be tough to understand by a common man. In this situation graphics is a better alternative. Graphics tools are charts and graphs. Using graphs, data can be represented in pictorial form. A picture can be understood easily just with a single look.

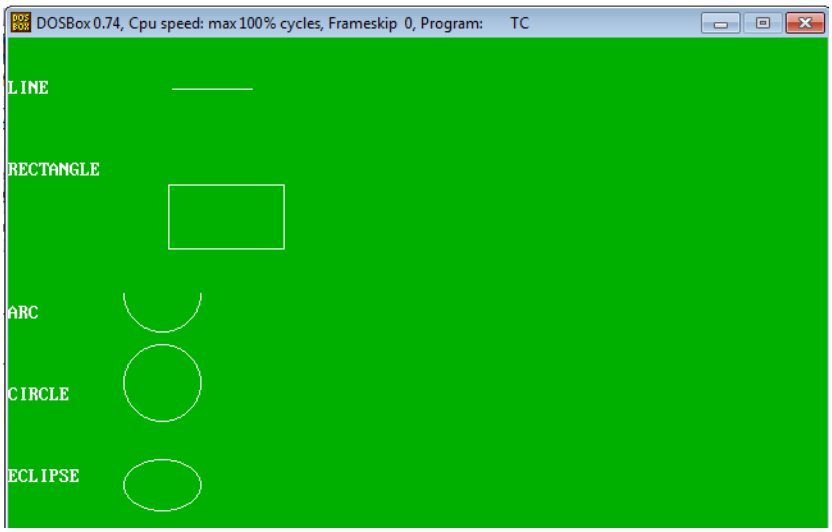
Interactive computer graphics work using the concept of two-way communication between computer users. The computer will receive signals from the input device, and the picture is modified accordingly. Picture will be changed quickly when we apply command.



Write a Program to draw basic graphics construction like line, circle, arc, ellipse and rectangle.

```
1.  #include<graphics.h>
2.  #include<conio.h>
3.  void main()
4.  {
5.      intgd=DETECT,gm;
6.      initgraph (&gd,&gm,"c:\\tc\\bgi");
7.      setbkcolor(GREEN);
8.      printf("\t\t\t\n\nLINE");
9.      line(50,40,190,40);
10.     printf("\t\t\n\n\n\nRECTANGLE");
11.     rectangle(125,115,215,165);
12.     printf("\t\t\t\n\n\n\n\n\n\nARC");
13.     arc(120,200,180,0,30);
14.     printf("\t\n\n\n\nCIRCLE");
15.     circle(120,270,30);
16.     printf("\t\n\n\n\nECLIPSE");
17.     ellipse(120,350,0,360,30,20);
18.     getch();
19. }
```

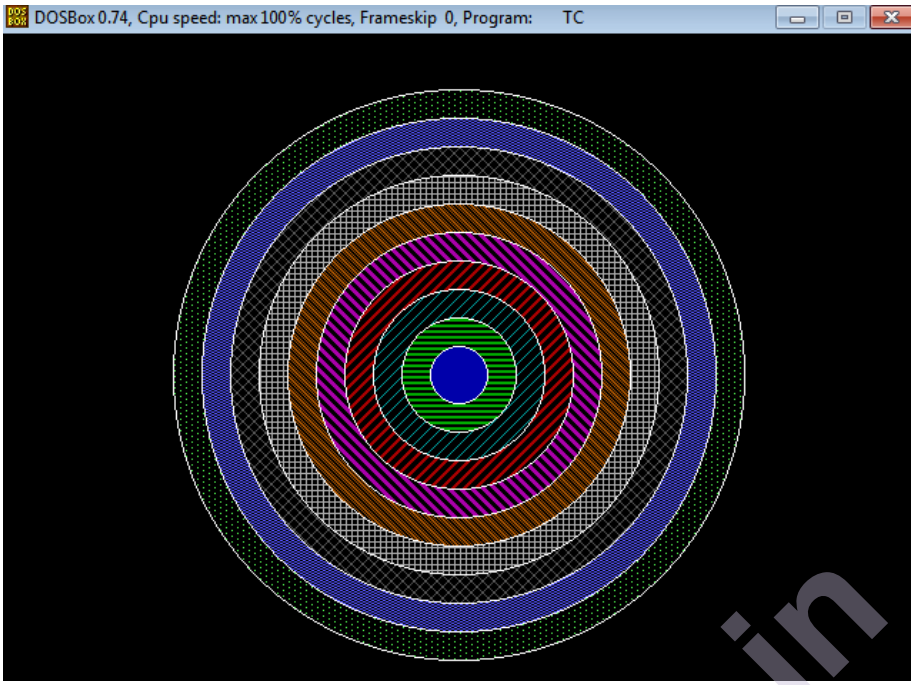
Output:



Write a Program to draw animation using increasing circles filled with different colors and patterns.

```
1.  #include<graphics.h>
2.  #include<conio.h>
3.  void main()
4.  {
5.     intgd=DETECT, gm, i, x, y;
6.     initgraph(&gd, &gm, "C:\\TC\\BGI");
7.     x=getmaxx()/3;
8.     y=getmaxx()/3;
9.     setbkcolor(WHITE);
10.    setcolor(BLUE);
11.    for(i=1;i<=8;i++)
12.        {
13.       setfillstyle(i,i);
14.       delay(20);
15.       circle(x, y, i*20);
16.       floodfill(x-2+i*20,y,BLUE);
17.    }
18.    getch();
19.    closegraph();
20. }
```

Output:



Animation:

In todays world which is filled with full of imagination and visualizations, there are some areas which is covered with the word animation. When this word will come in anyone's mind they always create a picture of cartoons and some Disney world shows. As we already know that among kids, animation movies are very popular like Disney world, Doraemon, etc. All the cartoons and animation pictures are type of animation which made from a thousands of single pictures add together and play according to steps.

As we shift our mind to past some decades than we will find that all animation are created either by hand or painting also we see that some puppet like structure were created to show the animation these type of animation are real world animations on the other hand in that technical world the digital animation will evolve. Now a days to create an animation there have developed so many tools to create scenes with ease and in a short time period some tools are as Blenders3D, Maya, cinema 4D etc, useful to create scenes.

There are many types of animation as we see in our TV's or we see many shows and pictures that are mainly differ from a real shows and films but we properly divide animation that are based on there creation such as 2D animation, 3D animation, paper animation, puppet animation, traditional animation, etc.

Definition of Animation:

Animation is the process of creating an illusion of motion and shape change by means of rapid display of various type of pictures that were made to create a single scene.

Principles of Animation:

Before doing animation, every animator should follow these principles to create a good animation. These principles was evolved from past animation techniques but these principle are also very useful and essential for doing animation. In 1981 two bright Disney animators Ollie Johnston and Frank Thomas introduced twelve basic principles of animation to produce more realistic works. These principle are also applicable on present computer animations.

There are 12 basic principles of animation, they are:

1. **Squash and Stretc:** This is the most important principle of animation, it gives the sense of weight and volume to draw an object.
2. **Anticipation:** In this principle animator will create a starting scene like that it shows that something will happen, almost nothing happens suddenly.
3. **Staging:** Animator creates such type of scene which attract audience so that audience's attention is directed toward that scene.
4. **Straight Ahead:** In this principle, all frames are drawn from beginning to the end and then fill all the interval or scene.
5. **Flow through and overlapping action :** Two object's action have different speed in any scene can easily describe this principle.
6. **Slow in and slow out:** When an abject have maximum acceleration in between and resist on the beginning and end will show this principle's working.
7. **Arc:** Arcs are present in almost all animation as no object will follow straight line and follows some arc in its action.
8. **Secondary action:** As with one character's action second character move shows the multiple dimension of an animation.
9. **Timing:** For playing a given action a perfect timing is very important.
10. **Exaggeration:** This principle creates extra reality in the scene by developing a proper animation style.
11. **Solid drawing:** In this principle, any object will created into 3D form to get realistic visualization of scene.

Appeal: –
Any character need not be as same as any real character but it somewhat seems to be like that which create a proper thinking in the audience's mind.

Write a program in C for bouncing ball animation using graphics.h header file

In this program, we will draw a red color ball move it vertically up and down like a bouncing ball. We will use below mentioned functions in this program.

Function	Description
initgraph	It initializes the graphics system by loading the passed graphics driver then changing the system into graphics mode.
getmaxx	It returns the maximum X coordinate in current graphics mode and driver.
setcolor	It changes the current drawing colour. Default colour is white. Each color is assigned a number, like BLACK is 0 and RED is 4. Here we are using colour constants defined inside graphics.h header file.
setfillstyle	It sets the current fill pattern and fill color.
Circle	It draws a circle with radius r and centre at (x, y).
floodfill	It is used to fill a closed area with current fill pattern and fill color. It takes any point inside closed area and color of the boundary as input.
cleardevice	It clears the screen, and sets current position to (0, 0).
Kbhit	It is used to determine whether a key is pressed or not. It returns a non-zero value if a key is pressed otherwise zero.
Delay	It is used to suspend execution of a program for a M milliseconds.
closegraph	It unloads the graphics drivers and sets the screen back to text mode.

C program for bouncing ball graphics animation:

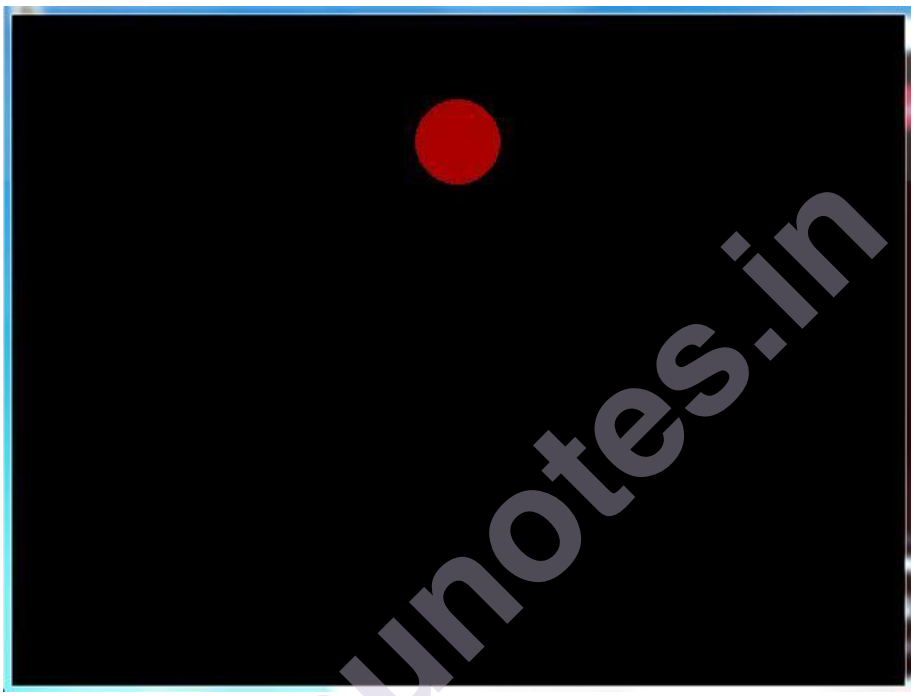
In this program, we first draw a red color ball on screen having center at (x, y) and then erases it using cleardevice function. We again draw this ball at center (x, y + 5), or (x, y - 5) depending upon whether ball is moving down or up. This will look like a bouncing ball. We will repeat above steps until user press any key on keyboard.

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <dos.h>

int main() {
 int gd = DETECT, gm;
 int i, x, y, flag=0;
 initgraph(&gd, &gm, "C:\\TC\\BGI");

 /* get mid positions in x and y-axis */
 x = getmaxx()/2;
 y = 30;



 while (!kbhit()) {
  if(y >= getmaxy()-30 || y <= 30)
     flag = !flag;
     /* draws the gray board */
     setcolor(RED);
     setfillstyle(SOLID_FILL, RED);
     circle(x, y, 30);
     floodfill(x, y, RED);

 /* delay for 50 milli seconds */
 delay(50);

 /* clears screen */
 cleardevice();
 if(flag){
     y = y + 5;
 } else {
     y = y - 5;
 }
    }
```

```
    getch();
    closegraph();
    return 0;
}
```

Program: Output

Here is the screenshot of bouncing ball.



Multimedia:

Multimedia is an interactive media and provides multiple ways to represent information to the user in a powerful manner. It provides an interaction between users and digital information. It is a medium of communication.Some of the sectors where multimedias is used extensively are education, training, reference material, business presentations, advertising and documentaries.

So far we have been using C language for simple console output only. Most of us are unaware that using C++, low level graphics program can also be made. This means we can incorporate shapes,colors and designer fonts in our program. This article deals with the steps to enable the DevC++ compiler to generate graphics .

Configuring DevC++

- **Step 1:** Download the DevC++ version 5.11.
- **Step 2:** Download the Graphics header files, and etc stuff.

- **Step 3:** Extract the contents of the rar file.
- **Step 4:** Go to the location where DevC++ is installed. For me its D drive. Go inside the MinGW64 folder. Copy the graphics.h and winbgim.h in the include folder and D:\Dev-Cpp\MinGW64\x86_64-w64-mingw32\include folder.
- **Step 5:**Copy the libbgi.a file into lib folder and in D:\Dev-Cpp\MinGW64\x86_64-w64-mingw32\lib folder.
- **Step 6:** Copy the ConsoleAppGraphics.template, ConsoleApp_cpp_graph.txt files and paste them inside the template folder of the devc++ installer location.

Now we are done with configuring of the DevC++ to support graphics programming. We shall write our very first graphics program now.

Running the first graphics program:

1. Open DevC++. Click file ->New ->Project.
2. Make sure you get the Console Graphics option. However, we are not going to click on it.
3. Choose Empty Project option and Give a project name and make sure the selected language is C++.
4. Copy the following code to the editor window.
5. #include<graphics.h>
6. #include <conio.h>
7. int main()
8. {
9. int gd = DETECT, gm;
10. initgraph(&gd,&gm, "C:\\tc\\bgi");
11. circle(300,300,50);
12. closegraph();
13. getch();
14. }

Go to “Project” menu and choose “Project Options” (or just press ALT+P).

Go to the “Parameters” tab In the “Linker” field, enter the following text:

-lbgi

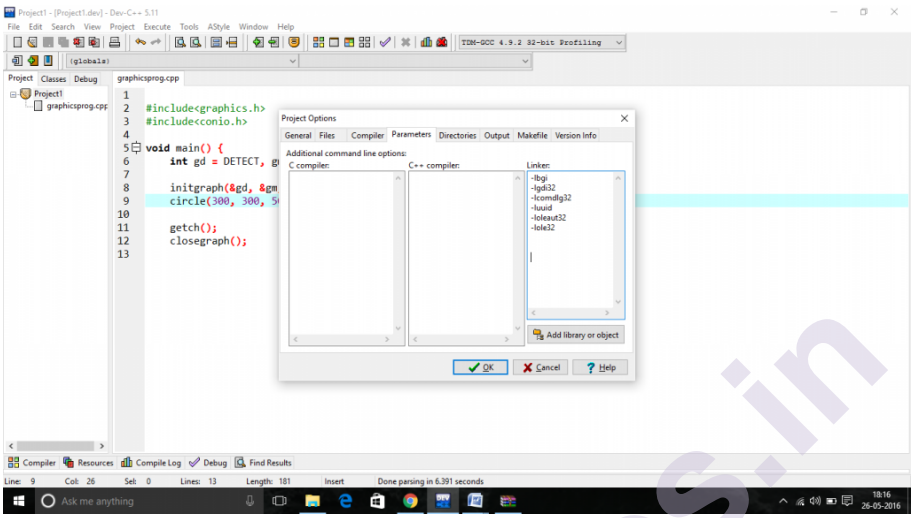
-lgdi32

-lcomdlg32

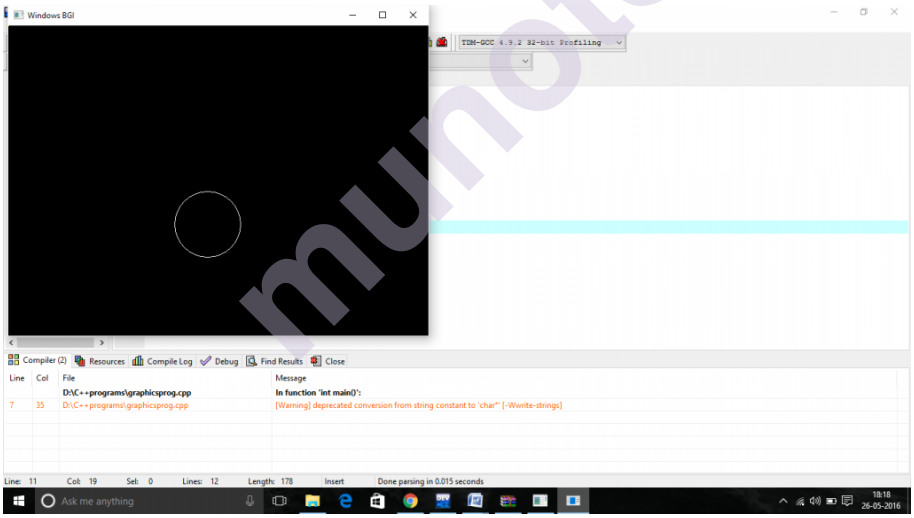
-luuid

-loleaut32

-lole32



Click OK and Compile and run the project and you'll get this output:



4.2 SUMMARY

Computer graphics is receiving much attention in the development of interactive educational software, multimedia systems, and many other applications. It not only adds a new dimension to such applications but also makes them more exciting and dynamic. Furthermore, the use of computer graphics is already well accepted in computer science education. If skillfully and relevantly used, it can be an important component of computer-assisted instruction, which is an educational application area with tremendous potential.

4.3 REFERENCES

- Animation for Beginners

  Basic Principles of Animation for Motion Graphics

  (Lisa Lee)
- Fundamentals of Computer Graphics

  Steve Marschner, Peter Shirley
- Introduction to Computer Graphics 2018 Edition

  Darrell Hajek

4.4 UNIT END EXERCISE

1. Write a C graphics program for moving car animation.
2. Write a C Program to Calculate Area and Perimeter of a Rectangle.

*****

5

LOCATION BASED SERVICES

Unit Structure

5.0 LOCATION-BASED SERVICES

Location-Based Services (LBS) are present in Android to provide you with features like current location detection, display of nearby places, geofencing, etc. It fetches the location using your device's GPS, Wifi, or Cellular Networks.

Components of Location-Based Services:

LocationManager Class: It is used to get Location Service access from the system.

LocationListener Interface: It receives updates from the Location Manager class.

LocationProvider: It is the class that provides us with the location for our devices.

Location Class: Its objects carry information about the location. The information includes latitude, longitude, accuracy, altitude, and speed.

The Location Object:

The Location object represents a geographic location which can consist of a latitude, longitude, time stamp, and other information such as bearing, altitude and velocity.

Sr.No.	Method	Description
1	**Float distanceTo (Location dest)**	Returns the approximate distance in meters between this location and the given location.
2	**float getAccuracy()**	Get the estimated accuracy of this location, in meters.

3	**double getAltitude()**	Get the altitude if available, in meters above sea level.
4	**float getBearing()**	Get the bearing, in degrees.
5	**double getLatitude()**	Get the latitude, in degrees.
6	**double getLongitude()**	Get the longitude, in degrees.
7	**float getSpeed()**	Get the speed if it is available, in meters/second over ground.
8	**boolean hasAccuracy()**	True if this location has an accuracy.
9	**boolean hasAltitude()**	True if this location has an altitude.
10	**boolean hasBearing()**	True if this location has a bearing.
11	**boolean hasSpeed()**	True if this location has a speed.
12	**void reset()**	Clears the contents of the location.
13	**void setAccuracy(float accuracy)**	Set the estimated accuracy of this location, meters.
14	**void setAltitude(double altitude)**	Set the altitude, in meters above sea level.
15	**void setBearing(float bearing)**	Set the bearing, in degrees.
16	**void setLatitude(double latitude)**	Set the latitude, in degrees.
17	**void setLongitude(double longitude)**	Set the longitude, in degrees.
18	**void setSpeed(float speed)**	Set the speed, in meters/second over ground.
19	**String toString()**	Returns a string containing a concise, human - readable description of this object.

5.1 DISPLAYING MAPS

Write a program to find your location in the Map

```
package com.abhiandroid.GoogleMaps.googlemaps;

import android.Manifest;

import android.content.Context;

import android.content.pm.PackageManager;

import android.location.Address;

import android.location.Criteria;
```

```
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationManager;
import android.os.Build;
import android.os.Bundle;
import android.support.v4.app.ActivityCompat;
import android.support.v4.app.FragmentActivity;
import android.support.v4.content.ContextCompat;
import android.widget.Toast;
import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.location.LocationListener;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationServices;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.BitmapDescriptorFactory;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;
import com.abhiandroid.GoogleMaps.googlemaps.R;


import java.io.IOException;
import java.util.List;
import java.util.Locale;
public class MapsActivity extends FragmentActivity implements
OnMapReadyCallback,
    GoogleApiClient.ConnectionCallbacks,
```

```
    GoogleApiClient.OnConnectionFailedListener,
    LocationListener {
  public static final int MY_PERMISSIONS_REQUEST_LOCATION =
99;
  GoogleApiClient mGoogleApiClient;
  Location mLastLocation;
  Marker mCurrLocationMarker;
  LocationRequest mLocationRequest;
  private GoogleMap mMap;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_maps);

    if (android.os.Build.VERSION.SDK_INT >=
Build.VERSION_CODES.M) {
      checkLocationPermission();
    }
    SupportMapFragment mapFragment = (SupportMapFragment)
        getSupportFragmentManager()
            .findFragmentById(R.id.map);

    mapFragment.getMapAsync(this);
  }
  @Override
  public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
    mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
    mMap.getUiSettings().setZoomControlsEnabled(true);
    mMap.getUiSettings().setZoomGesturesEnabled(true);
```

```
        mMap.getUiSettings().setCompassEnabled(true);
        //Initialize Google Play Services
        if (android.os.Build.VERSION.SDK_INT >=
Build.VERSION_CODES.M) {
            if (ContextCompat.checkSelfPermission(this,
                    Manifest.permission.ACCESS_FINE_LOCATION)
                    == PackageManager.PERMISSION_GRANTED) {
                buildGoogleApiClient();
                mMap.setMyLocationEnabled(true);
            }
        } else {
            buildGoogleApiClient();
            mMap.setMyLocationEnabled(true);
        }
    }
    protected synchronized void buildGoogleApiClient() {
        mGoogleApiClient = new GoogleApiClient.Builder(this)
                .addConnectionCallbacks(this)
                .addOnConnectionFailedListener(this)
                .addApi(LocationServices.API)
                .build();
        mGoogleApiClient.connect();
    }
    @Override
    public void onConnected(Bundle bundle) {
        mLocationRequest = new LocationRequest();
        mLocationRequest.setInterval(1000);
        mLocationRequest.setFastestInterval(1000);
mLocationRequest.setPriority(LocationRequest.PRIORITY_BALANCED
_POWER_ACCURACY);
```

```
        if (ContextCompat.checkSelfPermission(this,
                Manifest.permission.ACCESS_FINE_LOCATION)
                == PackageManager.PERMISSION_GRANTED) {

LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient,
                    mLocationRequest, this);
        }
    }
    @Override
    public void onConnectionSuspended(int i) {
    }
    @Override
    public void onLocationChanged(Location location) {
        mLastLocation = location;
        if (mCurrLocationMarker != null) {
            mCurrLocationMarker.remove();
        }
//Showing Current Location Marker on Map
        LatLng latLng = new LatLng(location.getLatitude(), location.getLongitude());
        MarkerOptions markerOptions = new MarkerOptions();
        markerOptions.position(latLng);
        LocationManager locationManager = (LocationManager)
                getSystemService(Context.LOCATION_SERVICE);
        String provider = locationManager.getBestProvider(new Criteria(), true);
        if (ActivityCompat.checkSelfPermission(this,
                Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&
                ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_COARSE_LOCATION)
```

```
                != PackageManager.PERMISSION_GRANTED) {
            return;
        }
        Location locations =
locationManager.getLastKnownLocation(provider);
        List<String> providerList = locationManager.getAllProviders();
        if (null != locations && null != providerList && providerList.size() >
0) {
            double longitude = locations.getLongitude();
            double latitude = locations.getLatitude();
            Geocoder geocoder = new Geocoder(getApplicationContext(),
                Locale.getDefault());
            try {
                List<Address> listAddresses =
geocoder.getFromLocation(latitude,
                    longitude, 1);
                if (null != listAddresses && listAddresses.size() > 0) {
                    String state = listAddresses.get(0).getAdminArea();
                    String country = listAddresses.get(0).getCountryName();
                    String subLocality = listAddresses.get(0).getSubLocality();
                    markerOptions.title("" + latLng + "," + subLocality + "," +
state
                        + "," + country);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

markerOptions.icon(BitmapDescriptorFactory.defaultMarker(BitmapDesc
riptorFactory.HUE_BLUE));
        mCurrLocationMarker = mMap.addMarker(markerOptions);
```

```
        mMap.moveCamera(CameraUpdateFactory.newLatLng(latLng));
        mMap.animateCamera(CameraUpdateFactory.zoomTo(11));
        if (mGoogleApiClient != null) {

LocationServices.FusedLocationApi.removeLocationUpdates(mGoogleAp
iClient,
                this);
        }
    }
    @Override
    public void onConnectionFailed(ConnectionResult connectionResult) {
    }
    public boolean checkLocationPermission() {
        if (ContextCompat.checkSelfPermission(this,
                Manifest.permission.ACCESS_FINE_LOCATION)
                != PackageManager.PERMISSION_GRANTED) {

            if (ActivityCompat.shouldShowRequestPermissionRationale(this,
                    Manifest.permission.ACCESS_FINE_LOCATION)) {
                ActivityCompat.requestPermissions(this,
                        new
String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                        MY_PERMISSIONS_REQUEST_LOCATION);
            } else {
                ActivityCompat.requestPermissions(this,
                        new
String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                        MY_PERMISSIONS_REQUEST_LOCATION);
            }
            return false;
        } else {
```

```
            return true;
        }
    }
    @Override
    public void onRequestPermissionsResult(int requestCode,
                                           String permissions[], int[] grantResults) {
        switch (requestCode) {
            case MY_PERMISSIONS_REQUEST_LOCATION: {
                if (grantResults.length > 0
                    && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                    if (ContextCompat.checkSelfPermission(this,
                        Manifest.permission.ACCESS_FINE_LOCATION)
                        == PackageManager.PERMISSION_GRANTED) {
                        if (mGoogleApiClient == null) {
                            buildGoogleApiClient();
                        }
                        mMap.setMyLocationEnabled(true);
                    }
                } else {
                    Toast.makeText(this, "permission denied",
                        Toast.LENGTH_LONG).show();
                }
                return;
            }
        }
    }
}
```

Output:

Now run the App. If you are connected to internet and provide access to your location then in Map you will see your current location.

5.2 GETTING LOCATION DATA

Step 1: Create a new project in Android Studio, go to File ⇒ New Project and fill all required details to create a new project.

Step 2: Add the following code to res/layout/activity_main.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity">
<TextView
  android:layout_marginTop="20dp"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="Get Current Location and City Name"
  android:textAlignment="center"
  android:layout_centerHorizontal="true"
  android:textSize="20sp" />
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/textView"
  android:layout_centerInParent="true"
  android:textSize="16sp"
  android:textStyle="bold"/>
</RelativeLayout>
```

Step 3: Add the following dependency in Gradle

```
implementation 'com.google.android.gms:play-services-location:17.0.0'
```

Step 4: Add the following code to src/MainActivity.java

```
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;
import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.location.Geocoder;
import android.location.Location;
import android.os.Bundle;
import android.os.Handler;
import android.os.ResultReceiver;
import android.util.Log;
import android.widget.TextView;
import android.widget.Toast;
import com.google.android.gms.location.FusedLocationProviderClient;
import com.google.android.gms.location.LocationCallback;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationResult;
import com.google.android.gms.location.LocationServices;
public class MainActivity extends AppCompatActivity {
  private FusedLocationProviderClient fusedLocationClient;
  private static final int LOCATION_PERMISSION_REQUEST_CODE
= 2;
  private LocationAddressResultReceiver addressResultReceiver;
  private TextView currentAddTv;
  private Location currentLocation;
```

```
private LocationCallback locationCallback;
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  addressResultReceiver = new LocationAddressResultReceiver(new
Handler());
  currentAddTv = findViewById(R.id.textView);
  fusedLocationClient =
LocationServices.getFusedLocationProviderClient(this);
  locationCallback = new LocationCallback() {
    @Override
    public void onLocationResult(LocationResult locationResult) {
      currentLocation = locationResult.getLocations().get(0);
      getAddress();
    }
  };
  startLocationUpdates();
}
@SuppressWarnings("MissingPermission")
private void startLocationUpdates() {
  if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) !=
  PackageManager.PERMISSION_GRANTED) {
    ActivityCompat.requestPermissions(this, new
    String[]{Manifest.permission.ACCESS_FINE_LOCATION},
    LOCATION_PERMISSION_REQUEST_CODE);
  }
  else {
    LocationRequest locationRequest = new LocationRequest();
```

```
        locationRequest.setInterval(2000);
        locationRequest.setFastestInterval(1000);

locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCUR
ACY);
        fusedLocationClient.requestLocationUpdates(locationRequest,
locationCallback, null);
    }
  }
  @SuppressWarnings("MissingPermission")
  private void getAddress() {
    if (!Geocoder.isPresent()) {
      Toast.makeText(MainActivity.this, "Can't find current address, ",
      Toast.LENGTH_SHORT).show();
      return;
    }
    Intent intent = new Intent(this, GetAddressIntentService.class);
    intent.putExtra("add_receiver", addressResultReceiver);
    intent.putExtra("add_location", currentLocation);
    startService(intent);
  }
  @Override
  public void onRequestPermissionsResult(int requestCode, @NonNull
String[] permissions, @NonNull
int[] grantResults) {
  if (requestCode == LOCATION_PERMISSION_REQUEST_CODE) {
    if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
      startLocationUpdates();
    }
     else {
```

```
        Toast.makeText(this, "Location permission not granted, " + "restart the app if you want the
        feature", Toast.LENGTH_SHORT).show();
      }
    }
  }
  private class LocationAddressResultReceiver extends ResultReceiver {
    LocationAddressResultReceiver(Handler handler) {
      super(handler);
    }
    @Override
    protected void onReceiveResult(int resultCode, Bundle resultData) {
      if (resultCode == 0) {
        Log.d("Address", "Location null retrying");
        getAddress();
      }
      if (resultCode == 1) {
        Toast.makeText(MainActivity.this, "Address not found, ", Toast.LENGTH_SHORT).show();
      }
      String currentAdd = resultData.getString("address_result");
        showResults(currentAdd);
      }
    }
    private void showResults(String currentAdd) {
      currentAddTv.setText(currentAdd);
    }
    @Override
    protected void onResume() {
      super.onResume();
```

```
    startLocationUpdates();
  }
  @Override
  protected void onPause() {
    super.onPause();
    fusedLocationClient.removeLocationUpdates(locationCallback);
  }
}
```

Step 5: Create a new java class GetaddressIntentService.java and add the following code

```
package app.com.sample;
import android.app.IntentService;
import android.content.Intent;
import android.location.Address;
import android.location.Geocoder;
import android.location.Location;
import android.os.Bundle;
import android.os.ResultReceiver;
import android.util.Log;
import java.util.List;
import java.util.Locale;
import java.util.Objects;
import androidx.annotation.Nullable;
public class GetAddressIntentService extends IntentService {
  private static final String IDENTIFIER = "GetAddressIntentService";
  private ResultReceiver addressResultReceiver;
  public GetAddressIntentService() {
    super(IDENTIFIER);
  }
  @Override
```

```
protected void onHandleIntent(@Nullable Intent intent) {
  String msg;
  addressResultReceiver = Objects.requireNonNull(intent).getParcelableExtra("add_receiver");
  if (addressResultReceiver == null) {
    Log.e("GetAddressIntentService", "No receiver, not processing the request further");
    return;
  }
  Location location = intent.getParcelableExtra("add_location");
  if (location == null) {
    msg = "No location, can't go further without location";
    sendResultsToReceiver(0, msg);
  return;
}
Geocoder geocoder = new Geocoder(this, Locale.getDefault());
List<Address> addresses = null;
try {
  addresses = geocoder.getFromLocation(location.getLatitude(), location.getLongitude(), 1);
}
catch (Exception ioException) {
  Log.e("", "Error in getting address for the location");
}
if (addresses == null || addresses.size() == 0) {
  msg = "No address found for the location";
  sendResultsToReceiver(1, msg);
}
else {
  Address address = addresses.get(0);
```

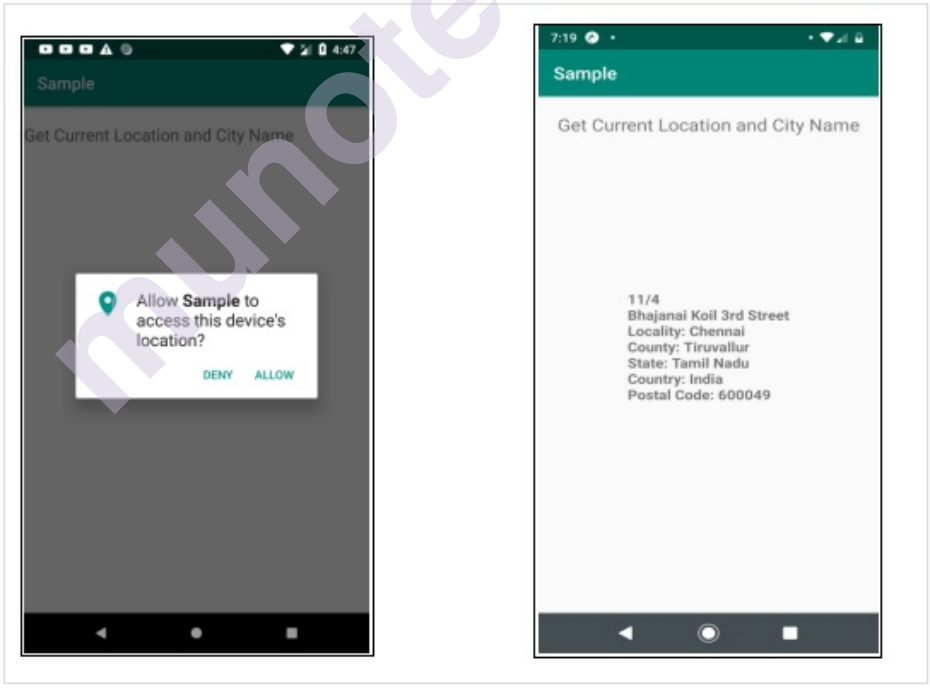
```
    String addressDetails = address.getFeatureName() + "\n" +
address.getThoroughfare() + "\n" +
    "Locality: " + address.getLocality() + "\n" + "County: " +
address.getSubAdminArea() + "\n" +
    "State: " + address.getAdminArea() + "\n" + "Country: " +
address.getCountryName() + "\n" +
    "Postal Code: " + address.getPostalCode() + "\n";
    sendResultsToReceiver(2, addressDetails);
  }
}
private void sendResultsToReceiver(int resultCode, String message) {
  Bundle bundle = new Bundle();
  bundle.putString("address_result", message);
  addressResultReceiver.send(resultCode, bundle);
  }
}
```

Step 6: Add the following code to androidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="app.com.sample">
  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
```

```
            <category    android:name="android.intent.category.LAUNCHER"
/>

        </intent-filter>

      </activity>

    <service android:name=".GetAddressIntentService" />

</application>

<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />

</manifest>
```

To run the app from the android studio, open one of your project's activity files and click the RunPlay Icon icon from the toolbar. Select your mobile device as an option and then check your mobile device which will display your default screen**:**



5.3 MONITORING A LOCATION

Android Location API

There are two ways to get a users location in our application:

1. android.location.LocationListener :

2. com.google.android.gms.location.LocationListener

LocationListener:

The LocationListener interface, which is part of the Android Locations API is used for receiving notifications from the LocationManager when the location has changed. The LocationManager class provides access to the systems location services.

a. The LocationListener class needs to implement the following methods.

b. onLocationChanged(Location location) : Called when the location has changed.

c. onProviderDisabled(String provider) : Called when the provider is disabled by the user.

d. onProviderEnabled(String provider) : Called when the provider is enabled by the user.

e. onStatusChanged(String provider, int status, Bundle extras) : Called when the provider status changes.

The android.location has two means of acquiring location data:

1. **LocationManager.GPS_PROVIDER:** Determines location using satellites. Depending on the conditions, this provider may take a while to return a location fix

2. **LocationManager.NETWORK_PROVIDER:** Determines location based on the availability of nearby cell towers and WiFi access points. This is faster than GPS_PROVIDER

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="https://schemas.android.com/apk/res/android"
   xmlns:tools="https://schemas.android.com/tools"
   android:id="@+id/activity_main"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context="com.journaldev.gpslocationtracking.MainActivity">

   <Button
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/btn"
        android:layout_centerInParent="true"
        android:text="GET LOCATION" />
</RelativeLayout>

package com.journaldev.gpslocationtracking;

import android.annotation.TargetApi;
import android.content.DialogInterface;
import android.content.pm.PackageManager;
import android.os.Build;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

import java.util.ArrayList;
import                                                    static
android.Manifest.permission.ACCESS_COARSE_LOCATION;
import static android.Manifest.permission.ACCESS_FINE_LOCATION;
public class MainActivity extends AppCompatActivity {
   private ArrayList permissionsToRequest;
   private ArrayList permissionsRejected = new ArrayList();
   private ArrayList permissions = new ArrayList();
   private final static int ALL_PERMISSIONS_RESULT = 101;
   LocationTrack locationTrack;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
```

```
        permissions.add(ACCESS_FINE_LOCATION);
        permissions.add(ACCESS_COARSE_LOCATION);
        permissionsToRequest = findUnAskedPermissions(permissions);
        //get the permissions we have asked for before but are not granted..
        //we will store this in a global list to access later.
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
            if (permissionsToRequest.size() > 0)
                requestPermissions(permissionsToRequest.toArray(new
String[permissionsToRequest.size()]), ALL_PERMISSIONS_RESULT);
        }
        Button btn = (Button) findViewById(R.id.btn);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                locationTrack = new LocationTrack(MainActivity.this);
                if (locationTrack.canGetLocation()) {
                    double longitude = locationTrack.getLongitude();
                    double latitude = locationTrack.getLatitude();
                    Toast.makeText(getApplicationContext(), "Longitude:" +
Double.toString(longitude) + "\nLatitude:" + Double.toString(latitude),
Toast.LENGTH_SHORT).show();
                } else {
                    locationTrack.showSettingsAlert();
                }

            }
        });

    }
    private ArrayList findUnAskedPermissions(ArrayList wanted) {
        ArrayList result = new ArrayList();
        for (String perm : wanted) {
            if (!hasPermission(perm)) {
                result.add(perm);
            }
        }
```

```
        return result;
    private boolean hasPermission(String permission) {
        if (canMakeSmores()) {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                return (checkSelfPermission(permission) == PackageManager.PERMISSION_GRANTED);
            }
        }
        return true;
    }
    private boolean canMakeSmores() {
        return (Build.VERSION.SDK_INT > Build.VERSION_CODES.LOLLIPOP_MR1);
    }
    @TargetApi(Build.VERSION_CODES.M)
    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
        switch (requestCode) {
            case ALL_PERMISSIONS_RESULT:
                for (String perms : permissionsToRequest) {
                    if (!hasPermission(perms)) {
                        permissionsRejected.add(perms);
                    }
                }
                if (permissionsRejected.size() > 0) {
                    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                        if (shouldShowRequestPermissionRationale(permissionsRejected.get(0))) {
                            showMessageOKCancel("These permissions are mandatory for the application. Please allow access.",
                                    new DialogInterface.OnClickListener() {
                                        @Override
                                        public void onClick(DialogInterface dialog, int which) {
```

```
                    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {

requestPermissions(permissionsRejected.toArray(new String[permissionsRejected.size()]), ALL_PERMISSIONS_RESULT);
                                }
                            }
                        });
                    return;
                }
            }

        }
        break;
    }
  }
  private void showMessageOKCancel(String message, DialogInterface.OnClickListener okListener) {
    new AlertDialog.Builder(MainActivity.this)
        .setMessage(message)
        .setPositiveButton("OK", okListener)
        .setNegativeButton("Cancel", null)
        .create()
        .show();
  }
  @Override
  protected void onDestroy() {
    super.onDestroy();
    locationTrack.stopListener();
  }
}
```

5.4 BUILD LOCATION TRACKER

```
public class LocationTrack extends Service implements LocationListener
{
private final Context mContext;
  boolean checkGPS = false;
```

```
    boolean checkNetwork = false;
    boolean canGetLocation = false;
    Location loc;
    double latitude;
    double longitude;
    private static final long MIN_DISTANCE_CHANGE_FOR_UPDATES
= 10;
    private static final long MIN_TIME_BW_UPDATES = 1000 * 60 * 1;
    protected LocationManager locationManager;
    public LocationTrack(Context mContext) {
      this.mContext = mContext;
      getLocation();
    }
    private Location getLocation() {
      try {
        locationManager = (LocationManager) mContext
            .getSystemService(LOCATION_SERVICE);
        // get GPS status
        checkGPS = locationManager
            .isProviderEnabled(LocationManager.GPS_PROVIDER);
        // get network provider status
        checkNetwork = locationManager

.isProviderEnabled(LocationManager.NETWORK_PROVIDER);
        if (!checkGPS && !checkNetwork) {
          Toast.makeText(mContext, "No Service Provider is available",
Toast.LENGTH_SHORT).show();
        } else {
          this.canGetLocation = true;
          // if GPS Enabled get lat/long using GPS Services
          if (checkGPS) {
            if (ActivityCompat.checkSelfPermission(mContext,
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(mContext,
Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
```

```
                // TODO: Consider calling
                //    ActivityCompat#requestPermissions
                // here to request the missing permissions, and then
overriding
                //          public void onRequestPermissionsResult(int
requestCode, String[] permissions,
                //                                  int[] grantResults)
                // to handle the case where the user grants the permission.
See the documentation
                // for ActivityCompat#requestPermissions for more details.
            }
            locationManager.requestLocationUpdates(
                LocationManager.GPS_PROVIDER,
                MIN_TIME_BW_UPDATES,
                MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
            if (locationManager != null) {
              loc = locationManager

.getLastKnownLocation(LocationManager.GPS_PROVIDER);
                if (loc != null) {
                  latitude = loc.getLatitude();
                  longitude = loc.getLongitude();
                }
            }
        }
        /*if (checkNetwork) {



            if (ActivityCompat.checkSelfPermission(mContext,
Manifest.permission.ACCESS_FINE_LOCATION)                !=
PackageManager.PERMISSION_GRANTED                        &&
ActivityCompat.checkSelfPermission(mContext,
Manifest.permission.ACCESS_COARSE_LOCATION)              !=
PackageManager.PERMISSION_GRANTED) {
                // TODO: Consider calling
                //    ActivityCompat#requestPermissions
                // here to request the missing permissions, and then
overriding
```

```
                    // public void onRequestPermissionsResult(int
requestCode, String[] permissions,
                    //                                 int[] grantResults)
                    // to handle the case where the user grants the permission.
See the documentation
                    // for ActivityCompat#requestPermissions for more details.
                }
                locationManager.requestLocationUpdates(
                    LocationManager.NETWORK_PROVIDER,
                    MIN_TIME_BW_UPDATES,
                    MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
if (locationManager != null) {
                    loc = locationManager

.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
                }
                if (loc != null) {
                    latitude = loc.getLatitude();
                    longitude = loc.getLongitude();
                }
            }*/
              } catch (Exception e) {
        e.printStackTrace();
      }
      return loc;
   }
   public double getLongitude() {
      if (loc != null) {
         longitude = loc.getLongitude();
      }
      return longitude;
   }
   public double getLatitude() {
      if (loc != null) {
         latitude = loc.getLatitude();
      }
```

```
        return latitude;
    }
    public boolean canGetLocation() {
        return this.canGetLocation;
    }
    public void showSettingsAlert() {
        AlertDialog.Builder alertDialog = new
AlertDialog.Builder(mContext);
        alertDialog.setTitle("GPS is not Enabled!");
        alertDialog.setMessage("Do you want to turn on GPS?");
        alertDialog.setPositiveButton("Yes", new
DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int which) {
                Intent intent = new
Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
                mContext.startActivity(intent);
            }
        });
        alertDialog.setNegativeButton("No", new
DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int which) {
                dialog.cancel();
            }
        });
        alertDialog.show();
    }
    public void stopListener() {
        if (locationManager != null) {
            if (ActivityCompat.checkSelfPermission(mContext,
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(mContext,
Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
                // TODO: Consider calling
                //    ActivityCompat#requestPermissions
                // here to request the missing permissions, and then overriding
```

```
            //    public void onRequestPermissionsResult(int requestCode, String[] permissions,
            //                                           int[] grantResults)
            // to handle the case where the user grants the permission. See the documentation
            // for ActivityCompat#requestPermissions for more details.
            return;
        }
        locationManager.removeUpdates(LocationTrack.this);
    }
  }
  @Override
  public IBinder onBind(Intent intent) {
    return null;
  }
  @Override
  public void onLocationChanged(Location location) {
  }
  @Override
  public void onStatusChanged(String s, int i, Bundle bundle) {
  }
  @Override
  public void onProviderEnabled(String s) {
  }
  @Override
  public void onProviderDisabled(String s) {
  }
}
```

QUIZ

1) Which of the following usually stores all user-related data that is also relevant to GSM mobile systems?

   . **VLR**

   a. HMR

   b. CMR

   c. SIM

2) In which one of the following codes with specific characteristics can be applied to the transmission?

   . **CDMA**

   a. GPRS

   b. GSM

   c. All of the above

3) Which of the following offers packet mode data transfer service over the cellular network?

. **TCP**

a. GPRS

b. GSM

c. None of the above

4) Which one of the following enables us to use the entire bandwidth simultaneously?

. **TDMA**

a. CDMA

b. FDMA

c. All of the above

5) In the Cellular Network, on which of the following, the cell's shape depends?

. **Political conditions**

a. Social Conditions

b. Environment Condition

c. None of the above

6) In a Cellular network, which of the following is used to use the same frequency for others?

. **Frequency hopping**

a. Frequency reuse

b. Frequency planning

c. None of the above

7) Which of the following uses wireless as the mode of communication for transferring or exchanging data between various mobiles over a short-range?

. **Ad hoc computing**

a. Mobile computing

b. Bluetooth technology

c. None of the above

8) The main aim of the file system is to support_____

. **Transparent access to data**

a. Efficient access to data

b. Consistent access to data

c. All of the above

9) In a distributed system, a client sends the request, and the server provides_____

. **Data**

a. Service

b. Information

c. All of the above

10) One of the essential challenges of distributed systems apply to DFS is/are

. **Migration of data**

a. Concurrent access to data

b. Replication of data

c. All of the above

11) Which one of the following can be considered as the features of CODA?

. **A disconnected operation for mobile computing**

a. It is freely available under a liberal license

b. It provides high performance through client-side persistent caching

c. All of the above

12) Which one of the file systems applies gossip protocols?

. **CODA**

a. Ficus

b. Rover

c. None of the above

13) Which of the following modes are supported by the Mio-NFS?

. **Connected**

a. Loosely connected

b. Disconnected

c. All of the above

14) Which of the following can be considered as the advantage of using frequency reuse?

. **The same spectrum can be allocated to the other networks**

a. Only a limited spectrum is required

b. Increase capacity

c. All of the above

15) Which of the following can be considered as the primary function of snooping TCP?

. **To buffer data close to the mobile host to perform fast local retransmission in case of packet loss.**

a. Congestion control

b. Flow control

c. None of the above

16) In which one of the following, the slow and fast hopping is used?

. **GSM**

a. GPRS

b. FHSS

c. None of the above

17) Mobile Computing allows transmission of data from one wireless-enabled device to another_

. **Any device**

a. Wired device

b. Wireless-enabled device

c. one of the above

18) Which of the following can be considered as the drawbacks of the Mobile and Wireless Devices?

. **Smaller keypads**

a. Consumes power rapidly

b. Requires a big power source

c. All of the above

19) In general, a mobile computing environment can also be considered as the type of _____ environment.

. **Grid computing**

a. Mobile computing

b. Distributed computing

c. None of the above

20) Which of the following is a fundamental principle of wireless communication?

. **Electromagnetic waves**

a. Microwaves

b. Both A and B

c. None of the abov

21) Which of the following statements is correct about the FHSS?

. **FHSS is a type of narrowband signal**

a. It uses the 78 frequency in the 2.4 GHz

b. It is referred as Frequency Hopping Spread Spectrum

c. All of the above

22) Which of the following is required to transmit the digital information using a certain frequency by translating it into an analog signal?

. **Demodulation**

a. Modulation

b. QPSK

c. BSPK

23) When was the 2G communication introduced in the market?

. **1982**

a. 1984

b. 1986

c. 1988

24) Which of the following is supported data rates of 4G Networks?

. **1024kbps**

a. 100mbs

b. 200mbps

c. None of the above

25) Which one of the following is considered as the GSM supplementary service?

. **Emergency number**

a. SMS

b. Call forwarding

c. All of the above

VIDEO LINKS

1. Location management | Mobile computing. https://www.youtube.com/watch?v=2TFn8oylzYE
2. Location Management. https://www.youtube.com/watch?v=cZHDLWtPg9k
3. Location Management In Mobile Computing. https://www.youtube.com/watch?v=FDOpxGkn58g
4. Location Management In Mobile Computing. https://www.youtube.com/watch?v=FDOpxGkn58g
5. Location Update in GSM ll Explained with Example in Hindi. https://www.youtube.com/watch?v=lVzSFMeKvvc
6. Location management in Mobile Computing. https://www.youtube.com/watch?v=v8GhpYg2CEg
7. Location Management: Principles and Techniques. https://www.youtube.com/watch?v=eDkhct72i5A
8. Mobility Management in Mobile Communication. https://www.youtube.com/watch?v=jnMlNW7SeFU

9. Location mgmt. https://www.youtube.com/watch?v=pJWd4izK7Z8
10. Mobility management | Hand-off. https://www.youtube.com/watch?v=ASU5nT3cTfs
11. Mobile assigned handoff, Intersystem handoff, Vehicle locating methods. https://www.youtube.com/watch?v=y1d2WyJqlFM
12. TDMA| Mobile Computing. https://www.youtube.com/watch?v=GZLJ8wRwRj0
13. Adaptation in mobile computing. https://www.youtube.com/watch?v=ueUeJ76DDgk
14. Location Management in Mobile Computing Network Projects. https://www.youtube.com/watch?v=3z1BgeI8z3g
15. Location Management Case Studies. https://www.youtube.com/watch?v=x9fAoTlHpTI

MOOCS

1. Programming Mobile Applications for Android Handheld Systems: Part 1. Coursera. https://www.coursera.org/learn/android-programming
2. Google Maps Web Services Proxy for Mobile Applications. Coursera. https://www.coursera.org/projects/googlecloud-google-maps-web-services-proxy-for-mobile-applications-guuf0
3. Introduction to Mobile and Cloud Computing. Alison. https://alison.com/course/introduction-to-mobile-and-cloud-computing-revised
4. Mobile Computing Online Course. youth4work. https://www.ed.youth4work.com/Course/479-Mobile-Computing-Online-Course
5. Mobile Computing Course And Certification. SIIT. https://siit.co/courses/mobile-computing-course-and-certification/718
6. CLOUD COMPUTING AND MOBILE COMPUTING. CSIT. https://www.csit.edu.in/program/cloud-computing-and-mobile-computing
7. Mobile App Development with Flutter & Dart (iOS and Android). UDEMY. https://www.udemy.com/course/mobile-app-development-with-flutter/

REFERENCES

1. https://www.tutorialspoint.com/android/android_location_based_services.htm
2. https://www.section.io/engineering-education/creating-a-location-tracking-app-using-firebase-and-google-maps-in-android/
3. https://www.tutorialspoint.com/the-simplest-way-to-get-the-user-s-current-location-on-android
4. https://www.journaldev.com/13325/android-location-api-tracking-gps
5. https://www.javatpoint.com/android-google-map-displaying-current-location
6. https://www.geeksforgeeks.org/how-to-get-user-location-in-android/

*****

6

REST API INTEGRATION

Unit Structure

6.0 REST API integration

REST API integration refers to connecting with third-party applications using HTTPS requests to access and transmit data. With several cloud applications, mobile apps, and IoT devices emerging on the digital horizon, businesses are discovering practical uses for these streaming sources.

Features of REST API:

REST is not a standard; it simply defines the principles of architecture to implement network applications or services. However, the implementation of REST is based on standards: HTTP, XML, etc.

- Client-Server. REST services must be based on a Client-Server architecture.
- No condition.
- Cache-enabled information.
- Consistent interface.
- Resource access by name.
- Related resources.
- Answer in a known format.

Client-Server:

REST services must be based on a Client-Server architecture. A server that includes the resources and conditions, and clients who access them

No condition:

REST Services can be scaled to achieve high performance to span the demand from all possible clients. This requires server farms to be implemented with charge balance and failover or different server levels to minimize the response time for the clients.

Cache-enabled information:

To improve network traffic efficiency, the answers from the server should be cache-enabled. This information is used by REST clients to decide whether they perform a local copy of the resource including time and date of the latest resource condition change.

Consistent interface:

One of the main features of REST Web services is the explicit use of HTTP (HyperText Transfer Protocol) methods. These methods are indicated in the HTTP header by the client and are as follows:

- **GET:** collects information from a resource
- **PUT:** modifies or updates the condition of a resource
- **POST:** creates a new resource in the server
- **DELETE:** removes a resource of the server

Resource access by name:

A REST system is made of resources accessed via URL, and these must be intuitive, predictive and easy to understand and set.

Related resources:

The resources available in the server are usually interrelated. As a consequence, the condition information of a resource should allow access to other resources.

Answer in a known format:

The representation of a resource reflects its current condition and its attributes when the client has made the request. The result can represent simply the value of a variable on a certain time, a database register or other type of information. In either case, the information must be delivered to the client on a format understandable for both parts, and included in the HTTP body. The most usual formats are JSON (JavaScript Object Notation) and XML (Extensible Markup Language), and others like CSV (Comma Separated Values) are also accepted.

Challenges of REST API Testing:

1. Securing REST API Parameter Combinations. ...
2. Validating REST API Parameters. ...
3. Maintaining Data Formatting. ...
4. Securing API Call Sequences. ...
5. Setting up an Automated REST API Test. ...
6. REST API Error Reporting.

1. Securing REST API Parameter Combinations:

The purpose of API parameters is to pass data values through API endpoints via data requests. Choosing the wrong REST API parameter combinations can trigger faulty program states that might potentially expose APIs to external attacks or cause crashes.

One of the best ways to ensure the security of a REST API, is to test all of its parameter combinations. Therefore, testing approaches that can automatically generate test cases for these parameters are particularly helpful to secure REST APIs, especially in large projects with many dependencies.

2. Validating REST API Parameters:

Even with automated testing tools, the sheer number of combinations is often too big to cover. Only white-box testing solutions are smart enough to pick values that are known from experience to cause problems. This way they can automatically generate inputs that try to cover all relevant parameter combinations.

3. Maintaining Data Formatting:

Automated testing solutions often allow for parsing of the API documentation and generate test cases based on this. If you test your API continuously and somehow documentation and implementation are out of

sync this, would be easily recognizable and this makes it easier to overcome the challenges.

4. Securing API Call Sequences:

When calling an API, a client application sends multiple requests, which must be called in the correct order. If the requests are handled in the wrong order, the program will return an error. An example of this would be the error that comes up when an API call to delete an object is made before the call to create it.

5. Setting up an Automated REST API Test:

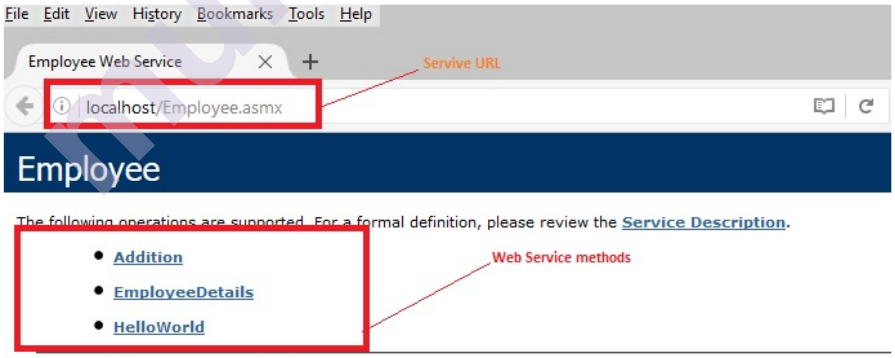
The initial configuration is the part of automated REST API testing that requires the most manual effort. While it is possible to build a continuous REST API testing cycle with open-source software, experience shows that this is usually vastly time-consuming.

6. REST API Error Reporting:

Conventional black-box testing tools can't measure the test coverage during REST API testing, which greatly reduces the value of test reports. White-box testing approaches enable testers to generate inputs that cover large parts of REST APIs, while providing detailed error reports and code-coverage visibility.

6.1 CONSUMING WEB SERVICE USING HTTPWEBREQUEST

Step 1: Find the Web Service SOAP Request Body.



After clicking on web method it shows SOAP request and SOAP Response body, find out the SOAP request body which is shown in the following image.

The above image shows the all details how to make SOAP request and also showing parameter values which is require to invoke service using HttpWebRequest .

HTTP Method: It's nothing but the what type of HTTP request you wants to make to the service such as GET , POST ,PUT or delete.

Host: It defines the url where is the service hosted , in our scenario our host is localhost , i.e http://localhost/Employee.asmx.

Content-Type: It defines what type of content type request you are making such as XML or json now in our scenario its text/xml.

Content-Length: It defines the content length of request body.

SOAPAction: This is very important attribute to identify specific web method and call it from multiple web methods .

SOAP Body: It contains the request and response body.

I hope you learned about SOAP request parameter , Now copy the soap envelope part to use it as SOAP request which we will use in our console application.

Step 2: Create the Console application to call Web Service

"Start" - "All Programs" - "Microsoft Visual Studio 2015".

"File" - "New" - "Project..." then in the New Project window, console "C#" - ".

Give the project a name, such as "UsingSOAPRequest" or another as you wish and specify the location.

Now open Program.cs file and write the following method to create HttpWebRequest as:

```
public HttpWebRequest CreateSOAPWebRequest()

  {

    //Making Web Request
```

```
        HttpWebRequest Req = (HttpWebRequest)WebRequest.Create(@"http://localhost/Employee.asmx");
        //SOAPAction
        Req.Headers.Add(@"SOAPAction:http://tempuri.org/Addition");
        //Content_type
        Req.ContentType = "text/xml;charset=\"utf-8\"";
        Req.Accept = "text/xml";
        //HTTP method
        Req.Method = "POST";
        //return HttpWebRequest
        return Req;
    }

Invoke the web service using SOAP request bo
public void InvokeService(int a, int b)
{
    //Calling CreateSOAPWebRequest method
    HttpWebRequest request = CreateSOAPWebRequest();

    XmlDocument SOAPReqBody = new XmlDocument();
    //SOAP Body Request
    SOAPReqBody.LoadXml(@ "<?xml version="
     "1.0"
     " encoding="
     "utf-8"
     "?>< soap: Envelope xmlns: soap = ""
     http: //schemas.xmlsoap.org/soap/envelope/"" xmlns:xsi=""http://www.w3.org/2001/XMLSchema-instance"" xmlns:xsd=""http://www.w3.org/2001/XMLSchema"">
< soap: Body >
< Addition xmlns = ""
     http: //tempuri.org/"">
< a > " + a + @" < /a>< b > " + b + @" < /b>< /Addition>< /soap:Body>< /soap:Envelope>");
     using(Stream stream = request.GetRequestStream()) {
      SOAPReqBody.Save(stream);
```

```
    }
    //Geting response from request
    using(WebResponse Serviceres = request.GetResponse()) {
      using(StreamReader rd = new StreamReader(Serviceres.GetResponseStream())) {
        //reading stream
        var ServiceResult = rd.ReadToEnd();
        //writting stream result on console
        Console.WriteLine(ServiceResult);
        Console.ReadLine();
      }
    }
  }
```

Now call above method from main method by writing following code as:

```
static void Main(string[] args)
  {
    //creating object of program class to access methods
    Program obj = new Program();
    Console.WriteLine("Please Enter Input values..");
    //Reading input values from console
    int a = Convert.ToInt32(Console.ReadLine());
    int b = Convert.ToInt32(Console.ReadLine());
    //Calling InvokeService method
    obj.InvokeService(a, b);
  }
```

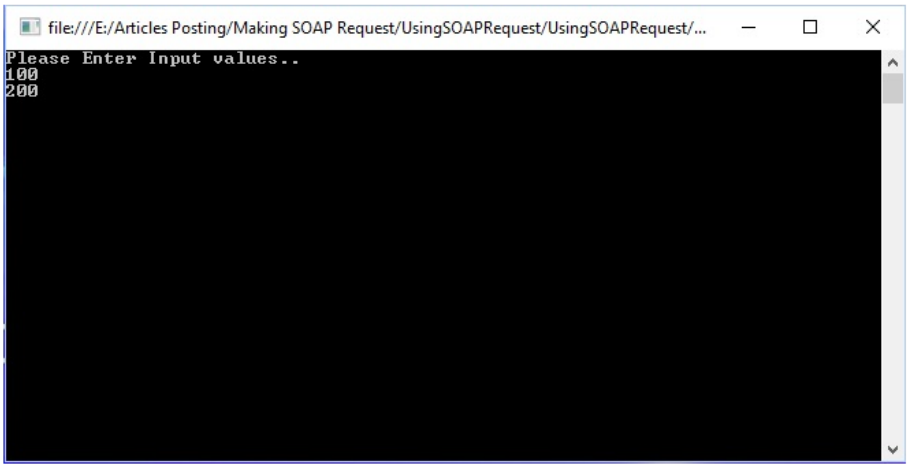
Now whole code in Program.cs file will be look like as follows:

```
using System;
using System.IO;
using System.Net;
using System.Xml;

namespace UsingSOAPRequest
{
  public class Program
  {
```

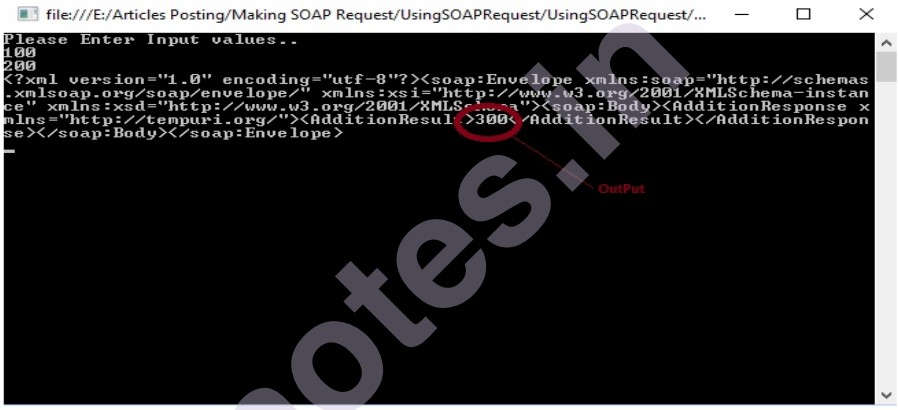
```
static void Main(string[] args)
{
   //creating object of program class to access methods
   Program obj = new Program();
   Console.WriteLine("Please Enter Input values..");
   //Reading input values from console
   int a = Convert.ToInt32(Console.ReadLine());
   int b = Convert.ToInt32(Console.ReadLine());
   //Calling InvokeService method
   obj.InvokeService(a, b);
}
public void InvokeService(int a, int b)
{
   //Calling CreateSOAPWebRequest method
   HttpWebRequest request = CreateSOAPWebRequest();
   XmlDocument SOAPReqBody = new XmlDocument();
   //SOAP Body Request
   SOAPReqBody.LoadXml(@"<?xml version=""1.0"" encoding=""utf-8""?>
<soap:Envelope xmlns:soap=""http://schemas.xmlsoap.org/soap/envelope/"" xmlns:xsi=""http://www.w3.org/2001/XMLSchema- instance"" xmlns:xsd=""http://www.w3.org/2001/XMLSchema"">
<soap:Body>
<Addition xmlns=""http://tempuri.org/"">
<a>" + a + @"</a>
<b>" + b + @"</b>
</Addition>
</soap:Body>
</soap:Envelope>");
   using (Stream stream = request.GetRequestStream())
   {
      SOAPReqBody.Save(stream);
   }
   //Geting response from request
   using (WebResponse Serviceres = request.GetResponse())
```

```
        {
            using (StreamReader rd = new StreamReader(Serviceres.GetResponseStream()))
            {
                //reading stream
                var ServiceResult = rd.ReadToEnd();
                //writting stream result on console
                Console.WriteLine(ServiceResult);
                Console.ReadLine();
            }
        }
    }
    public HttpWebRequest CreateSOAPWebRequest()
    {
        //Making Web Request
        HttpWebRequest Req = (HttpWebRequest)WebRequest.Create(@"http://localhost/Employee.asmx");
        //SOAPAction
        Req.Headers.Add(@"SOAPAction:http://tempuri.org/Addition");
        //Content_type
        Req.ContentType = "text/xml;charset=\"utf-8\"";
        Req.Accept = "text/xml";
        //HTTP method
        Req.Method = "POST";
        //return HttpWebRequest
        return Req;
    }
  }
}
```

Now run the console application, then the console windows will be look like as follows and enter the input values as:



Now press enter button , it will give the response in XML format are as follows:



6.2 CONCEPTS OF JSON WEB SERVICES

A JSON web service uses the term in its generic sense.

There are some important differences between SOAP and JSON:

- The content of a SOAP message is XML data, whereas a JSON message contains JSON data. JSON and XML are different encoding mechanisms for describing structured data.
- JSON is easy to integrate in JavaScript applications, but XML isn't. This makes JSON a preferred data format with many mobile application developers.
- SOAP provides a mechanism to add Headers to a message, and a family of specifications for qualities of service (such as security configuration, and distributed transactions).
- SOAP web services have an explicit error format involving SOAP Fault messages. There's no equivalent for JSON.

There are also many similarities between JSON and SOAP:

- The CICS implementation of JSON is derived from the SOAP architecture, and shares many of the concepts and artifacts.
- Both involve offline utility programs that assist with mapping application data to and from the external data representation. For SOAP there is DFHLS2WS and DFHWS2LS, for JSON there is DFHLS2JS and DFHJS2LS.
- The deployment mechanism for both technologies involve a PIPELINE resource, a WEBSERVICE resource, and a URIMAP resource.

JSON schema:

One disadvantage of JSON compared to SOAP is the difficulty in documenting the structure of a JSON interface. SOAP web services have the advantage of WSDL documents, together with XML schemas.

CICS implementation of JSON based web services:

CICS supports three modes of JSON web service, z/OS® Connect , Request-Response and RESTful. CICS also supports a programmatic scenario in which applications can transform JSON data to and from COBOL style data formats themselves.

z/OS Connect:

z/OS Connect enables CICS programs to be called with a JavaScript Object Notation (JSON) interface. z/OS Connect is IBM®'s premiere technology for implementing JSON Services and APIs in CICS. It is available in three versions, and supports several deployment options.

z/OS Connect Enterprise Edition enables API developers to construct JSON APIs from JSON services. The APIs are constructed and packaged with the Eclipse-based API Editor that is provided with z/OS Connect EE, then deployed to the z/OS Connect runtime.

Request-Response:

The Request-Response JSON pattern is very similar to that of SOAP based web services in CICS. The web service is implemented using a PROGRAM in CICS. The PROGRAM has input and output data formats, described using language structures (such as COBOL copybooks), and CICS is responsible for transforming incoming JSON messages into application data, and linking to the application.

A Request-Response mode JSON web service can be developed in either bottom-up mode or top-down mode. In bottom-up mode an existing CICS PROGRAM is exposed as a JSON web service using the DFHLS2JS JSON Assistant. In top-down mode a new JSON web service can be developed to implement an interface described using existing JSON

schemas. In top-down mode, the DFHJS2LS JSON Assistant is used to generate new language structures, and an application must be created to use them.

The Request-Response pattern may be used to build JSON Web Services that target either Commarea or Channel attached CICS PROGRAMs.

RESTful:

The concept of a RESTful JSON web service is described more completely in Concepts of RESTful JSON web services. A RESTful JSON web service implements the architectural principles of the REpresentational State Transfer (REST) design pattern. This design pattern is unlikely to be relevant for existing CICS applications, so is available only in top-down mode.

A JSON schema can be processed by DFHJS2LS in RESTful mode. An application must be written to implement the service and it will have to behave differently depending on the HTTP method that was used for the incoming request.

CICS implements a pure style of RESTful application, where the data format for POST (create) GET (inquire) and PUT (replace) are the same.

An example response that implements JSON:API

```
{
 "links": {
  "self": "http://example.com/articles",
  "next": "http://example.com/articles?page[offset]=2",
  "last": "http://example.com/articles?page[offset]=10"
 },
 "data": [{
  "type": "articles",
  "id": "1",
  "attributes": {
   "title": "JSON:API paints my bikeshed!"
  },
  "relationships": {
   "author": {
    "links": {
     "self": "http://example.com/articles/1/relationships/author",
     "related": "http://example.com/articles/1/author"
```

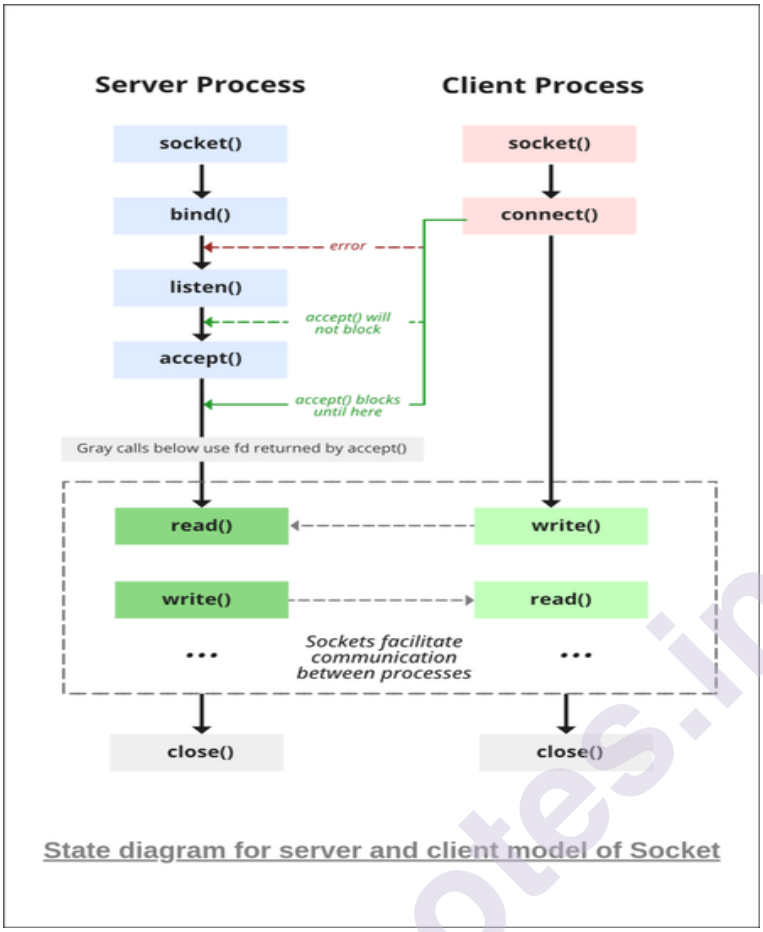
```
      },
      "data": { "type": "people", "id": "9" }
    },
    "comments": {
      "links": {
        "self": "http://example.com/articles/1/relationships/comments",
        "related": "http://example.com/articles/1/comments"
      },
      "data": [
        { "type": "comments", "id": "5" },
        { "type": "comments", "id": "12" }
      ]
    }
  },
  "links": {
    "self": "http://example.com/articles/1"
  }
}],
"included": [{
  "type": "people",
  "id": "9",
  "attributes": {
    "firstName": "Dan",
    "lastName": "Gebhardt",
    "twitter": "dgeb"
  },
  "links": {
    "self": "http://example.com/people/9"
  }
}, {
  "type": "comments",
  "id": "5",
```

```
    "attributes": {
      "body": "First!"
    },
    "relationships": {
      "author": {
        "data": { "type": "people", "id": "2" }
      }
    },
    "links": {
      "self": "http://example.com/comments/5"
    }
  }, {
    "type": "comments",
    "id": "12",
    "attributes": {
      "body": "I like XML better"
    },
    "relationships": {
      "author": {
        "data": { "type": "people", "id": "9" }
      }
    },
    "links": {
      "self": "http://example.com/comments/12"
    }
  }]
}
```

6.2 SOCKET PROGRAMMING IN C/C++

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server.

State diagram for server and client model:



State diagram for server and client model of Socket

Stages for server:

1. Socket creation:

int sockfd = socket(domain, type, protocol)

sockfd: socket descriptor, an integer (like a file-handle)

domain: integer, specifies communication domain.

type: communication type

SOCK_STREAM: TCP(reliable, connection oriented)

SOCK_DGRAM: UDP(unreliable, connectionless)

2. Setsockopt:

This helps in manipulating options for the socket referred by the file descriptor sockfd.

int setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t optlen);

3. Bind:

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

4. Listen:

int listen(int sockfd, int backlog);

5. Accept:

int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

Stages for Client:

Socket connection: Exactly same as that of server's socket creation

Connect: The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.

int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

Implementation:

Here we are exchanging one hello message between server and client to demonstrate the client/server model.

Server.c

C

```
// Server side C/C++ program to demonstrate Socket
// programming
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define PORT 8080
int main(int argc, char const* argv[])
{
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[1024] = { 0 };
```

```
char* hello = "Hello from server";

// Creating socket file descriptor
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0))
   == 0) {
   perror("socket failed");
   exit(EXIT_FAILURE);
}

// Forcefully attaching socket to the port 8080
if (setsockopt(server_fd, SOL_SOCKET,
            SO_REUSEADDR | SO_REUSEPORT, &opt,
            sizeof(opt))) {
   perror("setsockopt");
   exit(EXIT_FAILURE);
}
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

// Forcefully attaching socket to the port 8080
if (bind(server_fd, (struct sockaddr*)&address,
        sizeof(address))
   < 0) {
   perror("bind failed");
   exit(EXIT_FAILURE);
}
if (listen(server_fd, 3) < 0) {
   perror("listen");
   exit(EXIT_FAILURE);
}
if ((new_socket
    = accept(server_fd, (struct sockaddr*)&address,
            (socklen_t*)&addrlen))
   < 0) {
```

```
        perror("accept");
        exit(EXIT_FAILURE);
    }
    valread = read(new_socket, buffer, 1024);
    printf("%s\n", buffer);
    send(new_socket, hello, strlen(hello), 0);
    printf("Hello message sent\n");

  // closing the connected socket
    close(new_socket);
  // closing the listening socket
    shutdown(server_fd, SHUT_RDWR);
    return 0;
}
```

client.c

C

```
// Client side C/C++ program to demonstrate Socket
// programming
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define PORT 8080

int main(int argc, char const* argv[])
{
    int sock = 0, valread, client_fd;
    struct sockaddr_in serv_addr;
    char* hello = "Hello from client";
    char buffer[1024] = { 0 };
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        return -1;
```

```
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary
    // form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)
        <= 0) {
        printf(
            "\nInvalid address/ Address not supported \n");
        return -1;
    }

    if ((client_fd
         = connect(sock, (struct sockaddr*)&serv_addr,
                   sizeof(serv_addr)))
        < 0) {
        printf("\nConnection Failed \n");
        return -1;
    }
    send(sock, hello, strlen(hello), 0);
    printf("Hello message sent\n");
    valread = read(sock, buffer, 1024);
    printf("%s\n", buffer);

    // closing the connected socket
    close(client_fd);
    return 0;
}
```

Compiling:

gcc client.c -o client

gcc server.c -o server

Output:

Client:Hello message sent

Hello from server

Server:Hello from client

Hello message sent

6.3 OKHTTP ANDROID

Initially Android had only two HTTP clients:

1. HttpURLConnection and
2. Apache HTTP Client; for sending and receiving data from the web.

OkHttp android provides an implementation of HttpURLConnection and Apache Client interfaces by working directly on a top of java Socket without using any extra dependencies.

OkHttp Android Advantages:

1. Connection pooling
2. Gziping
3. Caching
4. Recovering from network problems
5. Redirects
6. Retries
7. Support for synchronous and asynchronous calls

OkHttp Android Example Code:

The MainActivity.java for Synchronous Calls is given below.

```
package com.journaldev.okhttp;
import android.os.AsyncTask;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
```

```
import org.json.JSONException;
import org.json.JSONObject;

import java.io.IOException;

import okhttp3.Call;
import okhttp3.Callback;
import okhttp3.MediaType;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.RequestBody;
import okhttp3.Response;
public class MainActivity extends AppCompatActivity {

   OkHttpClient client = new OkHttpClient();
   TextView txtString;
   public String url= "https://reqres.in/api/users/2";
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      txtString= (TextView)findViewById(R.id.txtString);
      OkHttpHandler okHttpHandler= new OkHttpHandler();
      okHttpHandler.execute(url);
   }
   public class OkHttpHandler extends AsyncTask {
      OkHttpClient client = new OkHttpClient();
      @Override
      protected String doInBackground(String...params) {
         Request.Builder builder = new Request.Builder();
         builder.url(params[0]);
         Request request = builder.build();
         try {
            Response response = client.newCall(request).execute();
            return response.body().string();
```

```
            }catch (Exception e){
                e.printStackTrace();
            }
            return null;
        }
        @Override
        protected void onPostExecute(String s) {
            super.onPostExecute(s);
            txtString.setText(s);
        }
    }
}
```

For Asynchronous Calls the MainActivity.java should be defined as:

```
package com.journaldev.okhttp;
import android.os.AsyncTask;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import org.json.JSONException;
import org.json.JSONObject;
import java.io.IOException;
import okhttp3.Call;
import okhttp3.Callback;
import okhttp3.MediaType;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.RequestBody;
import okhttp3.Response;
public class MainActivity extends AppCompatActivity {

    TextView txtString;
    public String url= "https://reqres.in/api/users/2";
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txtString= (TextView)findViewById(R.id.txtString);
        try {
            run();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    void run() throws IOException {
        OkHttpClient client = new OkHttpClient();
        Request request = new Request.Builder()
                .url(url)
                .build();
        client.newCall(request).enqueue(new Callback() {
            @Override
            public void onFailure(Call call, IOException e) {
                call.cancel();
            }
            @Override
            public void onResponse(Call call, Response response) throws
IOException {
                final String myResponse = response.body().string();
                MainActivity.this.runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        txtString.setText(myResponse);
                    }
                });
            }
        });
    }
}
```

We've used a test API from here.

The response string returned is of the JSON format that gets printed on the screen.

You can try out other open source API's like Github API, Stackoverflow etc.

OkHttp Query Parameters Example:

```
HttpUrl.Builder urlBuilder = HttpUrl.parse("https://httpbin.org/get).newBuilder();

urlBuilder.addQueryParameter("website", "www.journaldev.com");

urlBuilder.addQueryParameter("tutorials", "android");

String url = urlBuilder.build().toString();


Request request = new Request.Builder()
        .url(url)
        .build();
```

The above url was obtained from https://resttesttest.com/.

OkHttp Android Headers Example:

If there are any authenticated query parameters, they can be added in the form of headers as shown below:

```
Request request = new Request.Builder()
   .header("Authorization", "replace this text with your token")
   .url("your api url")
   .build();
```

Processing the JSON Response:

We can parse the JSON data to get the relevant params and display them in a TextView as below code.

```
client.newCall(request).enqueue(new Callback() {
        @Override
        public void onFailure(Call call, IOException e) {
           call.cancel();
        }
        @Override
```

```
        public void onResponse(Call call, Response response) throws
IOException {
            final String myResponse = response.body().string();
            MainActivity.this.runOnUiThread(new Runnable() {
               @Override
               public void run() {
                  try {
                     JSONObject json = new JSONObject(myResponse);

txtString.setText(json.getJSONObject("data").getString("first_name")+    "
"+json.getJSONObject("data").getString("last_name"));
                  } catch (JSONException e) {
                     e.printStackTrace();
                  }
               }
            });

         }
      });
```

OkHttp Android POST Example:

To post a data to the server we need to build our request in the following way.

```
public class MainActivity extends AppCompatActivity {
   public String postUrl= "https://reqres.in/api/users/";
   public String postBody="{\n" +
         "    \"name\": \"morpheus\",\n" +
         "    \"job\": \"leader\"\n" +
         "}";
   public static final MediaType JSON =
MediaType.parse("application/json; charset=utf-8");
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
        try {
            postRequest(postUrl,postBody);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    void postRequest(String postUrl,String postBody) throws IOException
{
        OkHttpClient client = new OkHttpClient();
        RequestBody body = RequestBody.create(JSON, postBody);
        Request request = new Request.Builder()
                .url(postUrl)
                .post(body)
                .build();
        client.newCall(request).enqueue(new Callback() {
            @Override
            public void onFailure(Call call, IOException e) {
                call.cancel();
            }
            @Override
            public void onResponse(Call call, Response response) throws
IOException {
                Log.d("TAG",response.body().string());
            }
        });
    }
}
```

In the above code, we've used the MediaType class that's a part of OkHttp to define the type of data being passed. We've used the test API URL from https://reqres.in/.

The post(RequestBody body) method is called on the RequestBuilder with the relevant value.

The Log displays the following response.

{"name":"morpheus","job":"leader","id":"731","createdAt":"2017-01-03T17:26:05.158Z"}.

6.4 COMPARING RETROFIT 2.0 VS. VOLLEY

Ease-of-use:

Retrofit is dead-simple to use. It essentially lets you treat API calls as simple Java method calls, so you only define which URLs to hit and the types of the request/response parameters as Java classes. The entire network call + JSON/XML parsing is completely handled by it (with help from Gson for JSON parsing), along with support for arbitrary formats with pluggable serialization / deserialization.

```
// Retrofit turns your HTTP API into a Java interface
public interface GitHubService {
  @GET("users/{user}/repos")
  Call<List<Repo>> listRepos(@Path("user") String user);
}
// the Retrofit class generates an implementation of the above interface
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://api.github.com/")
    .build();
GitHubService service = retrofit.create(GitHubService.class);
// each Call from the created GitHubService can make a synchronous or
// asynchronous HTTP request to the remote webserver
Call<List<Repo>> repos = service.listRepos("octocat");
// let's use the Call asynchronously
call.enqueue(new Callback<List<Contributor>>() {
  @Override void onResponse(Response<List<Contributor>>
contributors) {
    // ...
  }
  @Override void onFailure(Throwable t) {
    // ...
  }
});
```

Volley involves a little more boilerplate in general. Out-of-the-box, the only response types supported are String, Image, JSONObject, and JSONArray. To avoid the burden of unpacking JSONObjects yourself and use it like Retrofit, you need to include this GsonRequest<T> class, which basically represents a request whose response is converted to type T by Gson.

A short code sample:

```
// instantiate the RequestQueue
RequestQueue queue = Volley.newRequestQueue(this);
String url ="http://www.google.com";
// request a string response asynchronously from the provided URL
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
        new Response.Listener<String>() {
   @Override
   public void onResponse(String response) {
      // ...
   }
}, new Response.ErrorListener() {
   @Override
   public void onErrorResponse(VolleyError error) {
      // ...
   }
});
// add the request to the RequestQueue
queue.add(stringRequest);
```

Image loading:

Retrofit does not come with support for loading images from the network by itself. Another small library called Picasso is recommended for this. I really don't consider this a drawback because (a) Picasso is just as easy to use as Retrofit, and, (b) it avoids over-burdening the core library and bloating your APK if you don't need image loading features -- indeed this might be considered an advantage if you're looking for a smaller library over ease-of-use for images.

Volley has rudimentary support for image loading, which suffices for use-cases with a small number of images and not too many feature requirements.

So in this category it really comes down to Picasso vs Glide. Both libraries have a nearly identical API and feature set. The main differences are that:

- Glide caches images after resizing so it loads faster and uses less memory (demo), but may end up saving multiple copies of the same image at different sizes. Picasso saves the full image and resizes it on load, which is less memory- and CPU-efficient, but if you're displaying images at several sizes it will use up less disk space.
- Glide supports GIFs and locally-stored videos, Picasso doesn't.

Performance and caching:

- Both libraries use thread pools and execute multiple requests in parallel, and both support the venerable OkHttp library.
- Retrofit: caching should "just work" if your server sets the correct cache control headers. If not, or if you're trying to do something unusual, you may be out of luck. The only lever you can turn here is your HTTP client layer (OkHttp, HttpUrlConnection, etc), as in this example.
- Volley definitely has a more elaborate and flexible caching mechanism. Glide leverages it to great effect for caching bitmaps.

Request prioritization, cancellation and retrying:

- **Retrofit:** does not support setting priority, although there are hacks available; supports cancellation since v2 (which is in late beta now); supports manual request retries.
- **Volley:** supports prioritization with a little extra code; supports cancellation and customizable retry with exponential backoff out-of-the-box.

POST requests + multipart uploads:

- Retrofit has full support for POST requests and multipart file uploads, with a sweet API to boot.
- Volley supports POST requests but you'll have to convert your Java objects to JSONObjects yourself (e.g., with Gson). Also supports multipart requests but you need to add these additional classes or equivalent. The experience is not as polished as with Retrofit, but probably a bit more flexible as a result.

Extensibility:

- Retrofit has a well-designed, intuitive API with a small number of extension points for common use-cases, e.g., for (de)serialization and request threading.

- Volley on the other hand is very open to extension and even relies on that fact for common features like request priority, setting up a global long-lived RequestQueue, and bitmap caching.

Dynamic URLs:

- Retrofit supports dynamic URLs since v2.
- Volley obviously supports dynamic URLs.

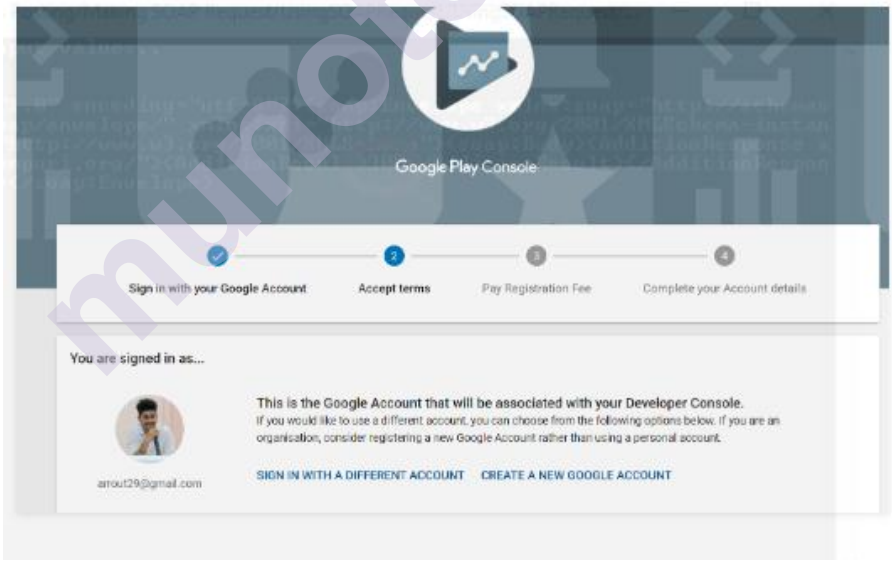
Large file downloads / streaming:

Neither Retrofit nor Volley is designed for this, and you will probably run into out-of-memory issues if you try. Both recommend using DownloadManager instead, which supports resuming and progress notifications.

6.5 HOW TO PUBLISH AN ANDROID APP ON GOOGLE PLAY STORE: A STEP-BY-STEP GUIDE:

Step 1: Create a Google Developer account:

A developer account is must be needed to upload an app on the Google Play Store, and the process is very simple. Just go through Google Play Store and do as instructed.

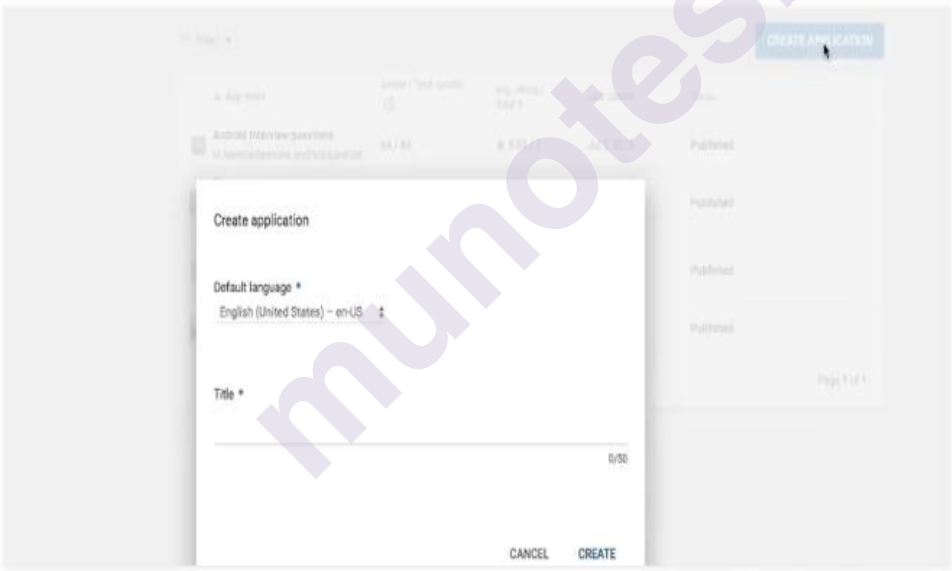


The account can be created in four simple steps:

- Sign-In with Your Google Account
- Accept Terms
- Pay Registration Fee of $25.
- Complete Your Account Details

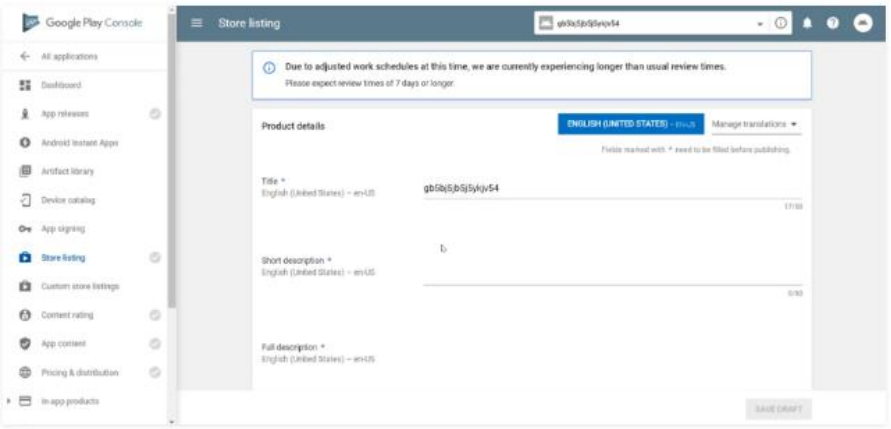
Step 2: After you completed step 1 you will be redirected to this page where you have to click on the **CREATE APPLICATION button.**

Once you click on it a pop up will be shown like this where you have to choose your Default language and Title of your app. Then click on the CREATE button.

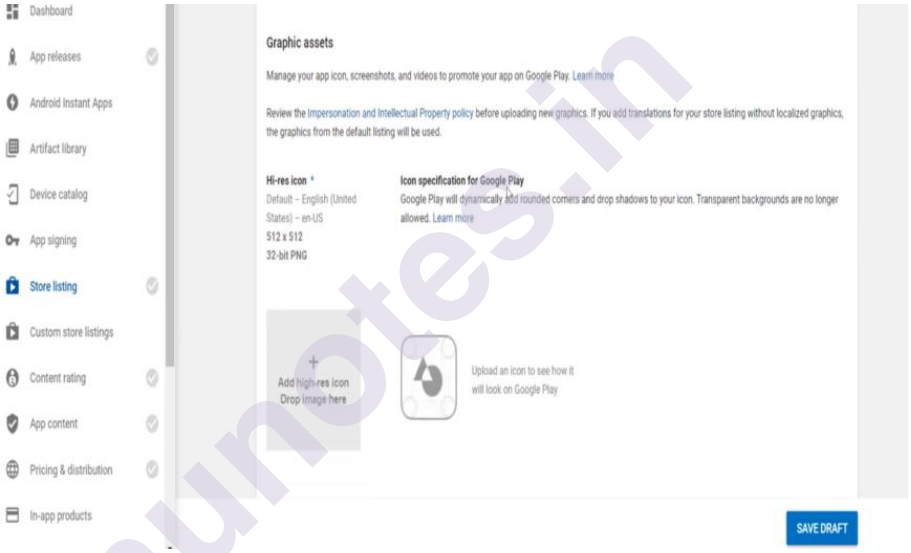


Step 3: Store listing

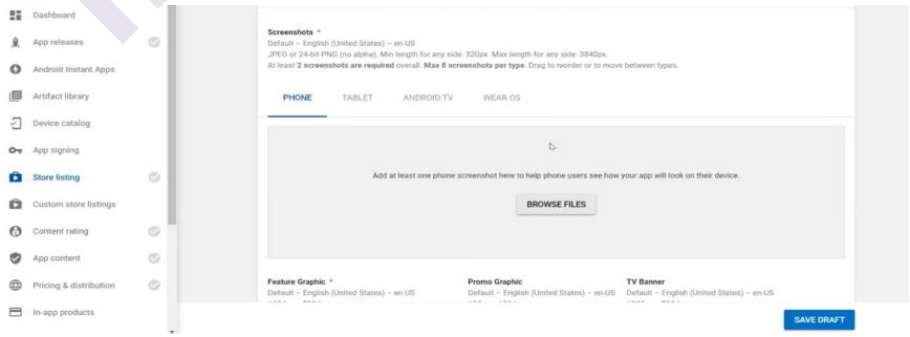
After you completed step 2 you will be redirected to this page where you have to provide the Short description and Full description of your App.



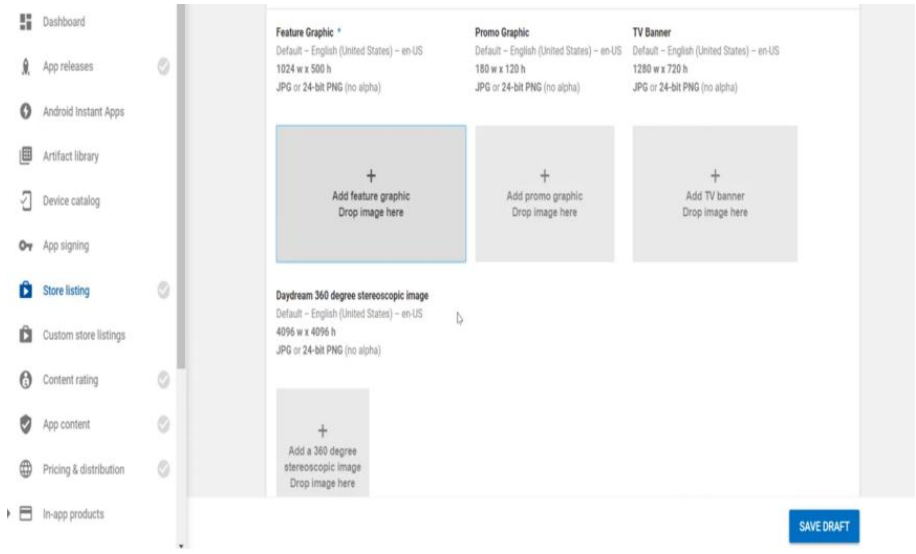
Then you scroll down the page and now you have to add the **Hi-res icon** of your app.



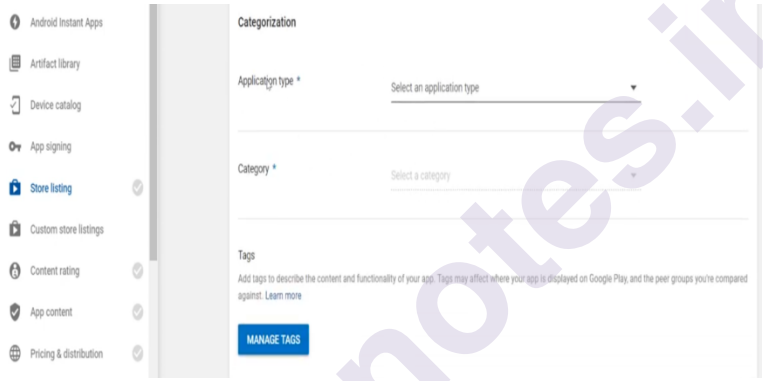
Then you have to provide the **Screenshots** of your app.



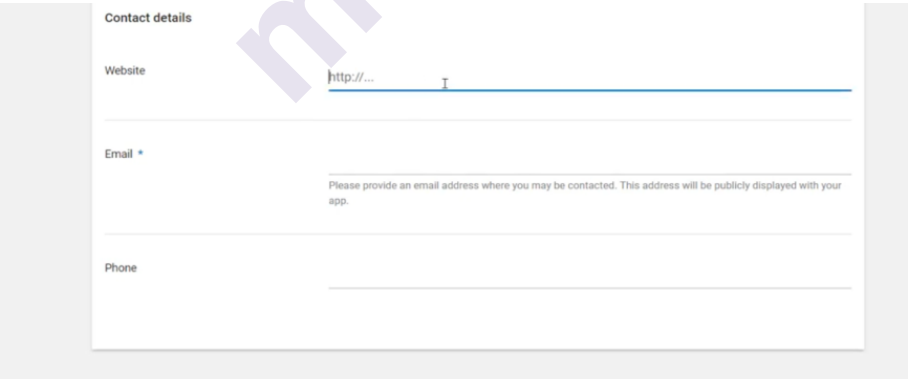
Ant next thing you have to provide is the **Feature Graphic** of your app. Note that this graphic is then used everywhere your app is featured on Google Play.



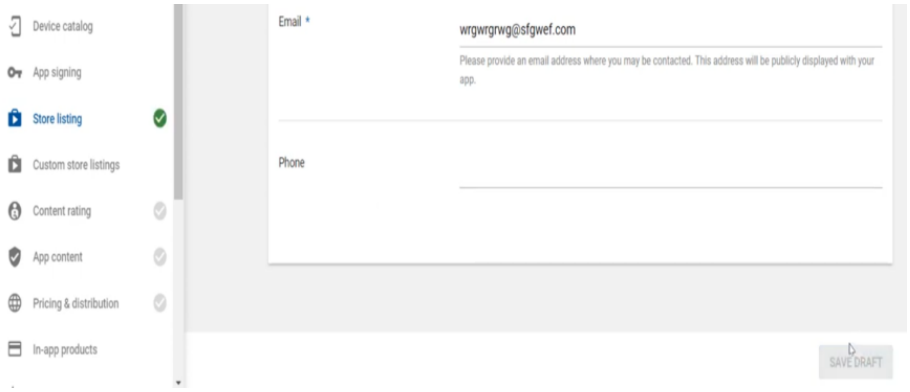
Then come to **Categorization** part where you have to provide your **Application type** and **Category** of your app.



Then come to **Contact details** part where you have to provide your **Website(if any)**, **email,** and **Phone** of yours.



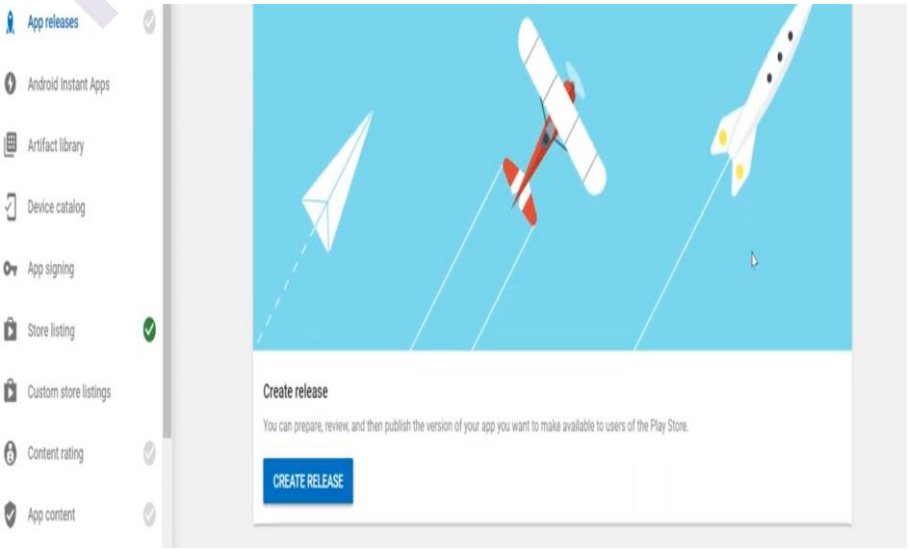
And finally when you click on **SAVE DRAFT** button you can see that **Store listing** tab is now become turned to green and you are done for Store listing.



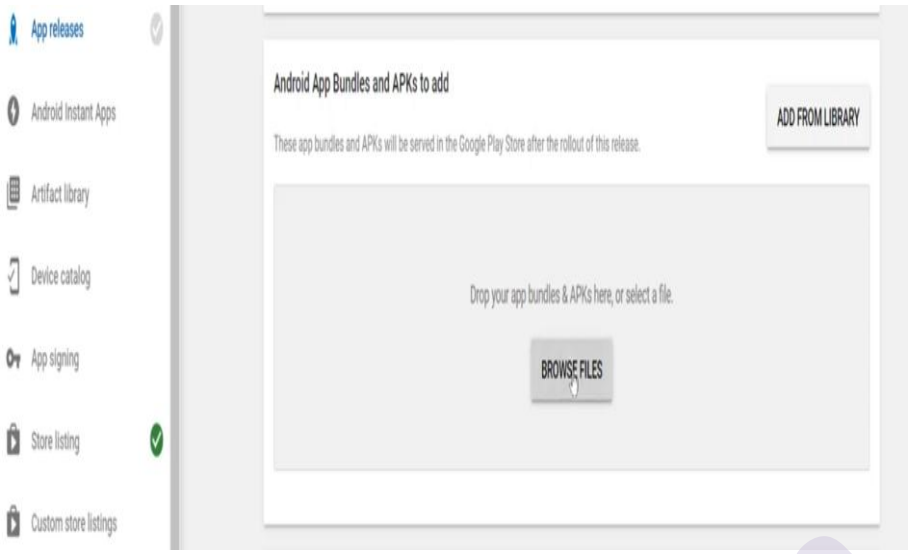
Step 4: App release:

After completing step 3 go to **App releases** then scroll down to **Production track** and click on **MANAGE** button.

After redirecting to the next page click on the **CREATE RELEASE** button.



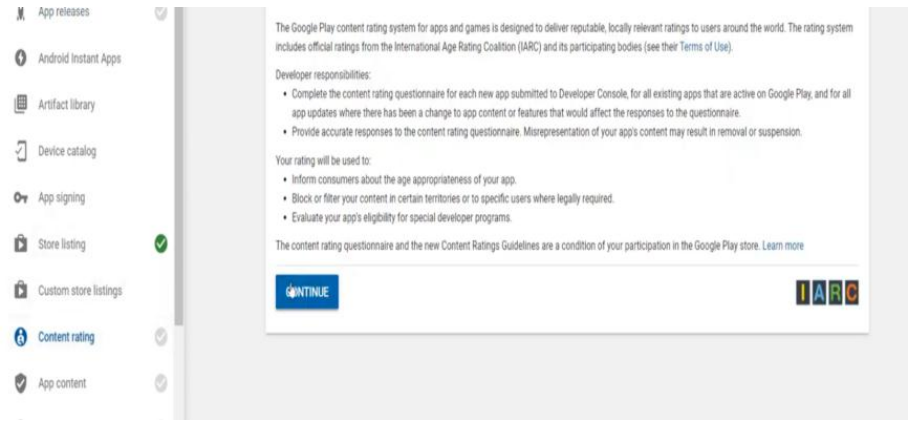
After that on the next page, you have to upload your APK file in **Android App Bundles and APKs to add** section.



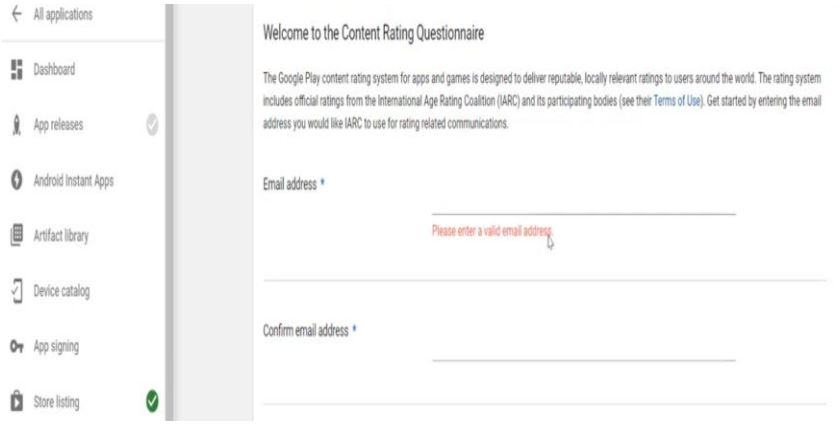
After that simply click on the **SAVE** button.

Step 5: Content rating:

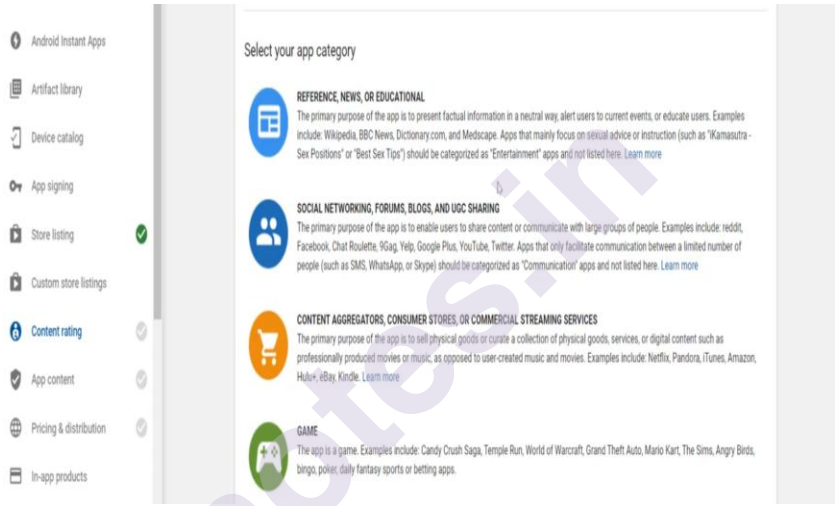
Now after completing step 4 go to **Content rating** and click on **CONTINUE** button.



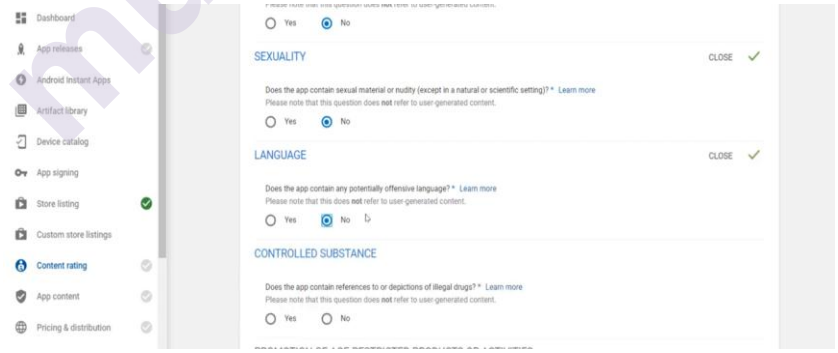
After that fill your email address as well as confirm the email address.



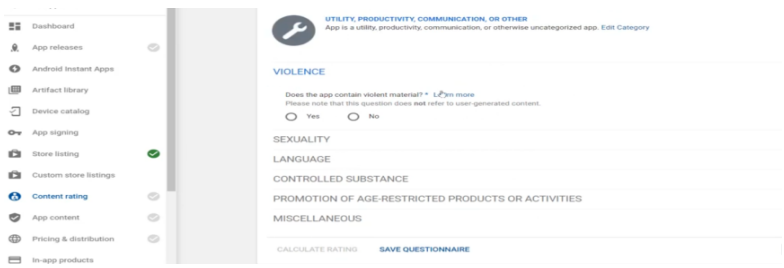
And then **Select your app category.**



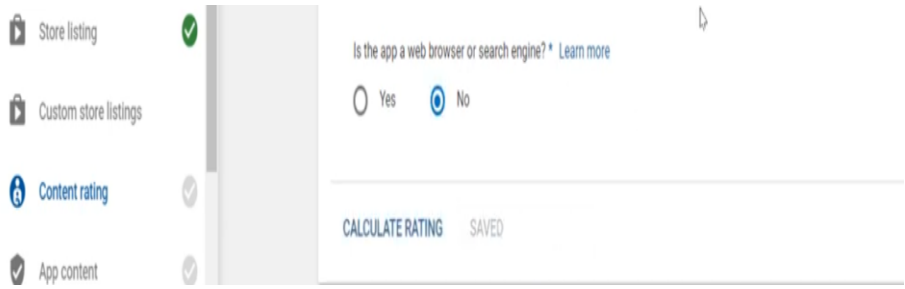
After selecting your app category make sure that you read all of these and answer them correctly.



And after answering them correctly don't forget to click on **SAVE QUESTIONNAIRE** button.



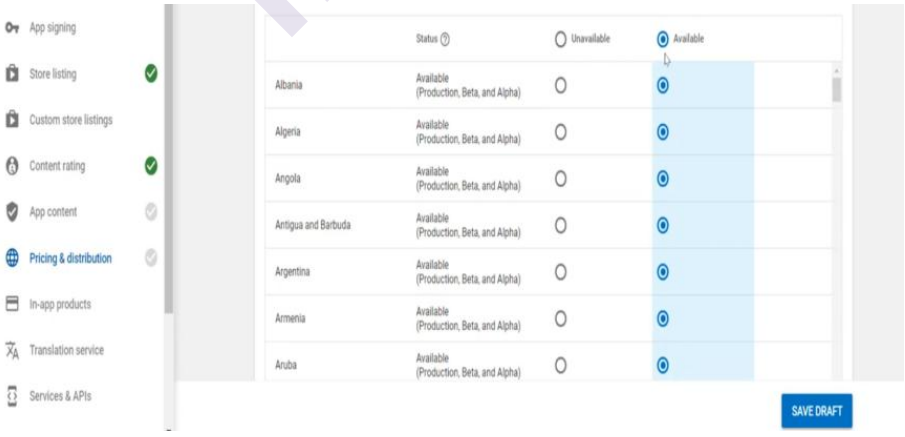
Once you saved all those things then click on **CALCULATE RATING** button.



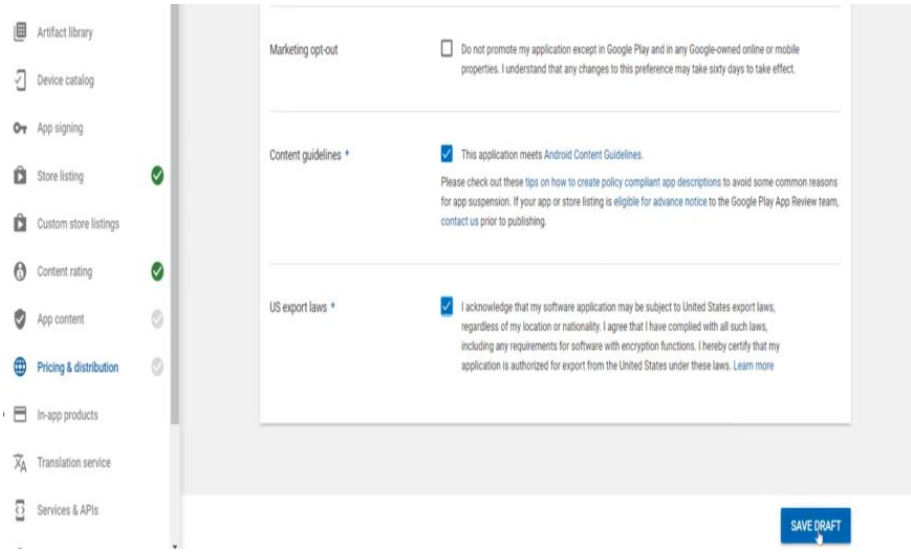
When you redirected to another page scroll down and click on **APPLY RATING** button. And you are done for **Content rating** section. Don't forget to notice that **Content rating** section is now become turned to green.

Step 6: Pricing & distribution:

Then go to the **Pricing & distribution** section. Then select the country in which you want to available your app.

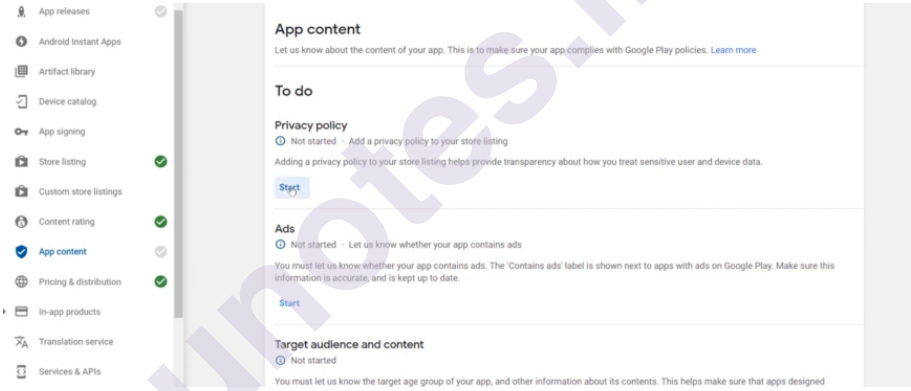


Then go down and down and check out the **Content guidelines and US export laws** section by marking them tick mark. And click on the **SAVE DRAFT** button. Don't forget to notice that **Pricing & distribution** section is now become turned to green tick.

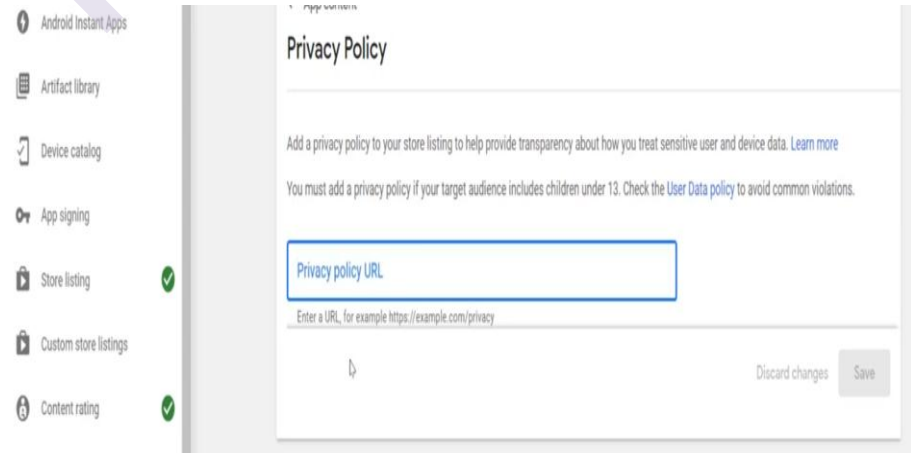


Step 7: App content:

Then come to the **App content** section. And in the **Privacy policy** section click on the **Start** button.

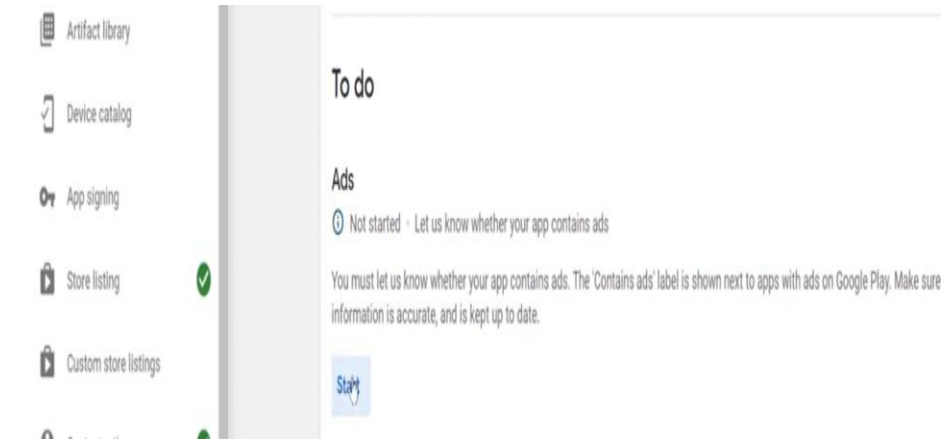


And then provide a valid **Privacy policy URL**. **Note that google will check this**.

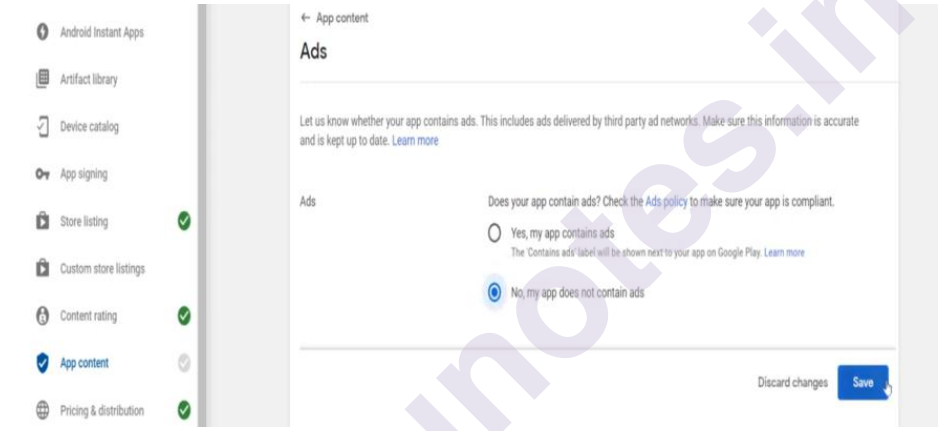


Then go back and continue further steps by clicking **start** button in **Ads** section.

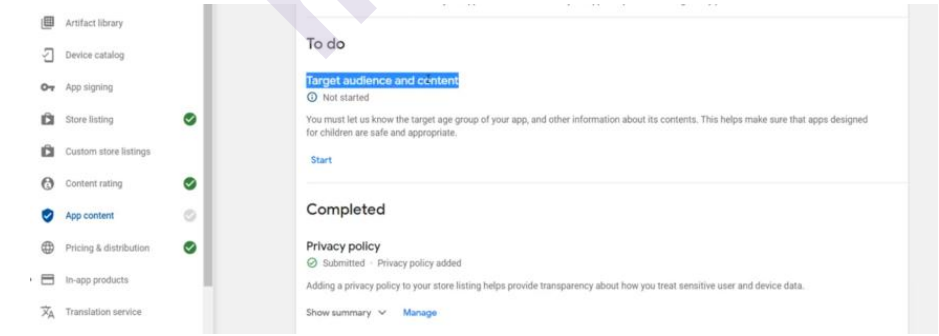
Then go back and continue further steps by clicking **start** button in **Ads** section.



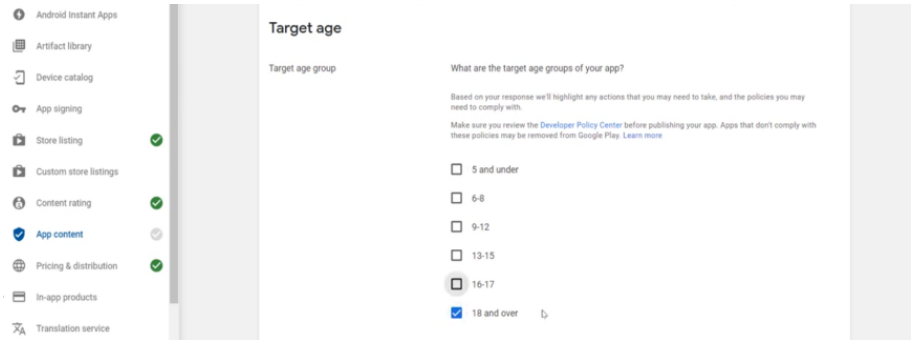
Then select **does your app contain ads or not?** And click on **SAVE** button.



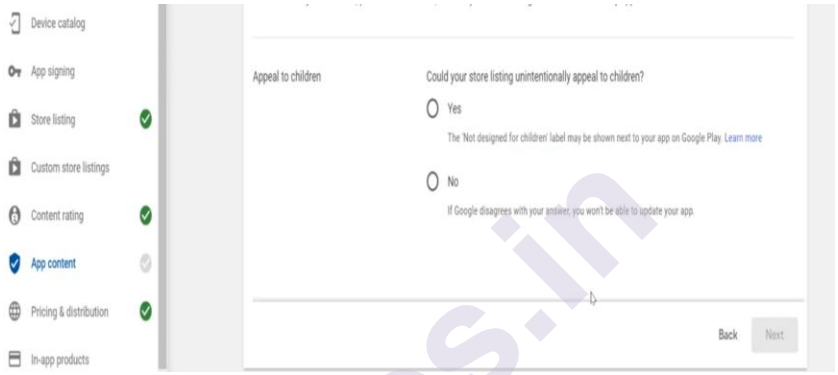
Then again go back and continue further steps by clicking **start** button in **Target audience and content** section.



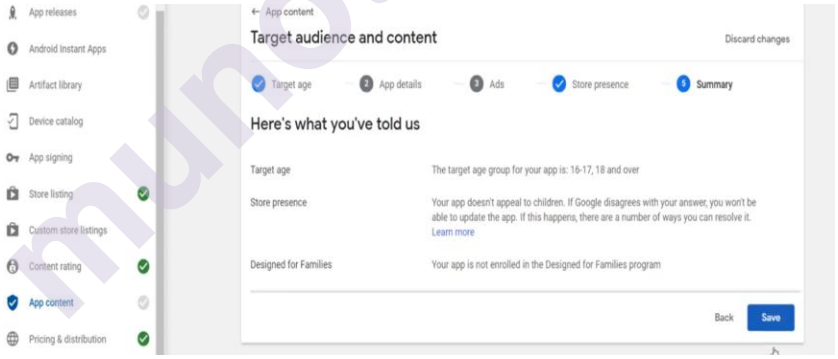
In the next page select the **Target age** group and scroll down and click on the **Next** button.



Then check the **Appeal to children** section. And click on the **Next** button.

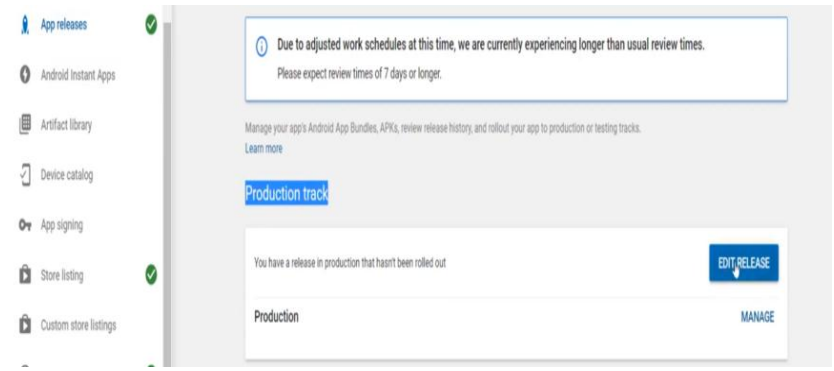


On the next page click on the **Save** button and you are done for **App content section**.

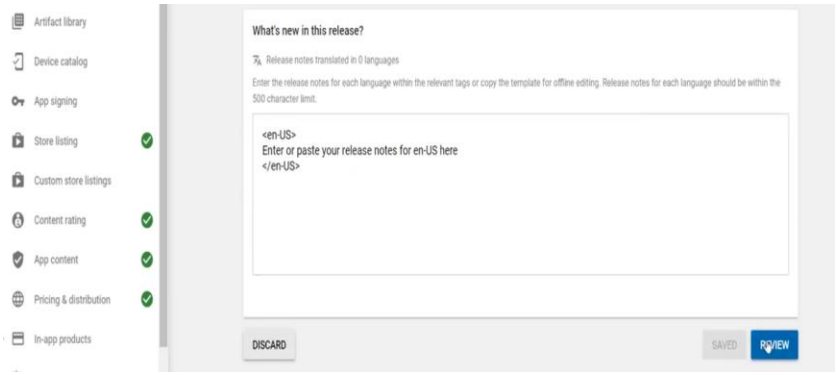


Step 8: App releases:

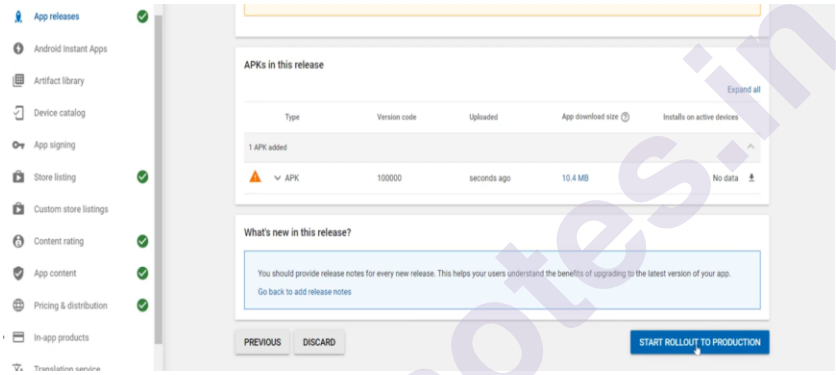
Again go back to the **App releases** section. And in the **Production track** click on the **EDIT RELEASE** button.



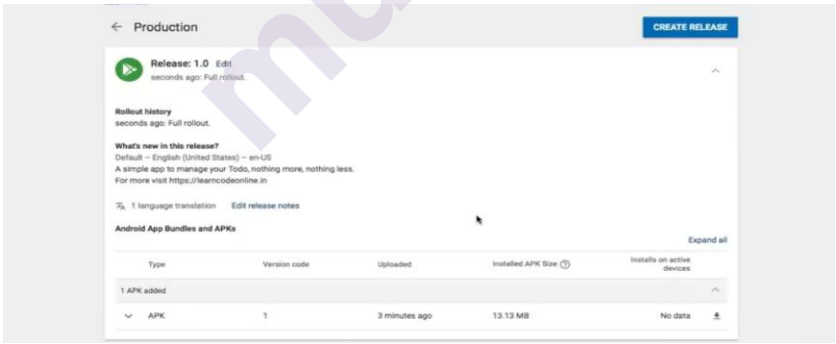
Then on the next page go down and down and click on the **REVIEW** button.



And finally, on the next page click on the **START ROLLOUT TO PRODUCTION** button to send your app to review. And you are finally done.



After usually 4 to 5 days they will review your app and let you know to either approve or reject your app.



6.6 EXERCISES

1. Creating a REST API
2. Access OCI Language with REST APIs
3. Viewpoint Sessions RESTful API
4. Viewpoint SSL Certificates RESTful API

QUIZ

Q1 - Which of the following is a components of a Web Service architecture?

A - SOAP

B - UDDI

C - WSDL

D - All of the above.

Q2 - Which of the following is the benefits of Web services being loosely coupled?

A - The web service interface can change over time without compromising the client's ability to interact with the service.

B - Adopting a loosely coupled architecture tends to make software systems more manageable and allows simpler integration between different systems.

C - Both of the above

D - None of the above.

Q3 - Which of the following is correct about Service Description layer in Web Service Protocol Stack?

A - This layer is responsible for describing the public interface to a specific web service.

B - Currently, service description is handled via the Web Service Description Language (WSDL).

C - Both of the above.

D - None of the above.

Q4 - Which of the following is correct about BEEP protocol?

A - This is a promising alternative to HTTP.

B - BEEP is a new Internet Engineering Task Force (IETF) framework for building new protocols.

C - Both of the above.

D - None of the above.

Q5 - Which of the following is correct about SOAP?

A - SOAP is language independent.

B - SOAP is simple and extensible.

C - Both of the above.

D - None of the above.

Q6 - Which of the following is true about REST?

A - REST is web standards based architecture and uses HTTP Protocol for data communication.

B - It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods.

C - REST was first introduced by Roy Fielding in 2000.

D - All of the above.

Q7- Which of the following protocol is used by RESTful web services as a medium of communication between client and server?

A - HTTP

B - FTP

C - Gopher

D - None of the above.

Q8- Which of the following is advantage of RESTful web service being stateless?

A - Web services can treat each method request independently.

B - Web services need not to maintain client's previous interactions. It simplifies application design.

C - As HTTP is itself a statelessness protocol, RESTful Web services work seamlessly with HTTP protocol.

D - All of the above.

Q9- Which of the following HTTP method should be read only in nature?

A - GET

B - DELETE

C - POST

D - PUT

Q10 - Which of the following directive of Cache Control Header of HTTP response provides indication to server to revalidate resource if max-age has passed?

A - must-revalidate

B - Private

C - no-cache/no-store

D - max-age

Q11 - Which of the following HTTP Status code means NOT MODIFIED, used to reduce network bandwidth usage in case of conditional GET requests?

A - 200

B - 201

C - 204

D - 304

Q12 - Which of the following annotation of JAX RS API is used to annotate a method used to create resource?

A - @Path

B - @GET

C - @PUT

D - @POST

Q13 - Which of the following annotation of JAX RS API binds the parameter passed to method to a query parameter in path?

A - @PathParam

B - @QueryParam

C - @MatrixParam

D - @HeaderParam

Q14 - A RESTful web service client sends a message in form of a Gopher Request and server responds in form of a HTTP Response.

A - true

B - false

Q15 - Response Header - Contains metadata for the HTTP Response message as key-value pairs.

A - true

B - false

Q16 - Which methods are commonly used in Server Socket class?

a) Public Output Stream get Output Stream ()

b) Public Socket accept ()

c) Public synchronized void close ()

d) Public void connect ()

Q17 - Which constructor of Datagram Socket class is used to create a datagram socket and binds it with the given Port Number?

a) Datagram Socket(int port)

b) Datagram Socket(int port, Int Address address)

c) Datagram Socket()

d) Datagram Socket(int address)

Q18 - The client in socket programming must know which information?

a) IP address of Server

b) Port number

c) Both IP address of Server & Port number

d) Only its own IP address

Q19 - The URL Connection class can be used to read and write data to the specified resource that is referred by the URL.

a) True

b) False

Q20 - Datagram is basically just a piece of information but there is no guarantee of its content, arrival or arrival time.

a) True

b) False

Q21 - TCP, FTP, Telnet, SMTP, POP etc. are examples of _____

a) Socket

b) IP Address

c) Protocol

d) MAC Address

Q22 - What does the java.net.InetAddress class represent?

a) Socket

b) IP Address

c) Protocol

d) MAC Address

Q23 - The flush () method of Print Stream class flushes any un-cleared buffers in the memory.

a) True

b) False

Q24 - Which classes are used for connection-less socket programming?

a) Datagram Socket

b) Datagram Packet

c) Both Datagram Socket & Datagram Packet

d) Server Socket

Q25 - In Inet Address class, which method returns the host name of the IP Address?

a) Public String get Hostname()

b) Public String getHostAddress()

c) Public static InetAddress get Localhost()

d) Public getByName()

VIDEO LINKS

1. How to Integrate REST API into a Custom Mobile Application. https://www.youtube.com/watch?v=n7onrlkovAQ
2. REST API - Introducing REST. https://www.youtube.com/watch?v=HeXQ98sogs8
3. What is REST API ? https://www.youtube.com/watch?v=o7ePQ_M_jsc
4. REST API - REST Constraints. https://www.youtube.com/watch?v=JYNYv8jJQTE

5. What is an API ? Simply Explained. https://www.youtube.com/watch?v=XGa4onZP66Q

6. What is API and API integration? https://www.youtube.com/watch?v=N4TCoZ069jk

7. Salesforce Apex Rest API Integration. https://www.youtube.com/watch?v=hNG8zZDgp7Y

8. What is an API? https://www.youtube.com/watch?v=E0Qqpn8ymko

9. What is an API? | Rest API | Web Service. https://www.youtube.com/watch?v=zeN41a4RVF0

10. What is REST API? | Web Service. https://www.youtube.com/watch?v=qVTAB8Z2VmA

11. Build a Mobile App with Custom API Backend in 5 Minutes. https://www.youtube.com/watch?v=l5WmM3atjKk

12. How to Call REST API in Android. https://www.youtube.com/watch?v=DpEg_UVkv6E

13. What is an API and how does it work? https://www.youtube.com/watch?v=Yzx7ihtCGBs

14. What is an API - Rest API in Android App Development. https://www.youtube.com/watch?v=35ZzyRY7FlU

15. REST API Token Authentication for Mobile Apps. https://www.youtube.com/watch?v=v4db49yJPIU

16. What is a REST API? https://www.youtube.com/watch?v=lsMQRaeKNDk

17. REST Vs Web Socket Explained in Hindi. https://www.youtube.com/watch?v=u_0Vzmj1Bxo

MOOCS

1. Building RESTful APIs Using Node.js and Express. Coursera. https://www.coursera.org/learn/building-restful-apis-using-nodejs-and-express

2. Postman - Intro to APIs (without coding). Coursera. https://www.coursera.org/projects/laura-gemmell-intro-postman-apis

3. REST API Courses. Udemy. https://www.udemy.com/topic/rest-api/

4. Rest API. EDX. https://www.edx.org/learn/rest-api

5. Designing RESTful APIs. UDACITY. https://www.udacity.com/course/designing-restful-apis--ud388

6. Learn the basics of REST APIs. LinkedIn. https://www.linkedin.com/learning/learning-rest-apis/welcome?autoplay=true&u=89447330

7. Application Programming Interface (API). IBM. https://www.ibm.com/cloud/learn/api

REFERENCES

1. https://www.astera.com/type/blog/rest-api-integration/

2. https://www.comunicacionesindustrialeslogitek.com/en/features-of-rest-services/

3. https://www.code-intelligence.com/blog/challenges-rest-api-testing

4. https://www.compilemode.com/2018/07/consuming-web-service-using-Http-Web-Request.html

5. https://jsonapi.org/

6. https://www.journaldev.com/13629/okhttp-android-example-tutorial

7. https://vickychijwani.me/retrofit-vs-volley/

*****

7.1

INTRODUCTION TO DART AND FLUTTER

Unit Structure

7.1.0 OBJECTIVES

After this chapter, you will be:

- Understand the fundamentals of the Dart programming language.
- Learn how to set up your own environment, and finally, learn how to get started with it.
- Getting to know the principles and tools of the Dart language.
- Understanding why Flutter uses Dart.

- Able to work on Android Development, iOS Development, and Web Development using the Flutter Framework.

7.1.1 INTRODUCTION

The Dart language is present at the core of the Flutter framework. A modern framework such as Flutter requires a high-level modern language to be capable of providing the best experience to the developer and making it possible to create awesome mobile applications. Understanding Dart is fundamental to working with Flutter; developers need to know the origins of the Dart language, how the community is working on it, its strengths, and why it is the chosen programming language to develop with Flutter. In this chapter, you will review the basics of the Dart language and be provided with some links to resources that can help you on your Flutter journey. You will review Dart built-in types and operators, and how Dart works with object-oriented programming (OOP). By understanding what the Dart language provides, you will be able to comfortably experiment with the Dart environment by yourself and expand your knowledge.

7.1.2 AN OVERVIEW OF DART LANGUAGE

Dart is the **open-source** programming language originally developed by Google. It is meant for both server side as well as the user side. The Dart SDK comes with its compiler – the Dart VM and a utility dart2js which is meant for generating Javascript equivalent of a Dart Script so that it can be run on those sites also which don't support Dart. Dart is Object-oriented language and is quite similar to that of Java Programming. Dart is extensively use to create single-page websites and web-applications. Best example of dart application is Gmail. Dart provides the most typical operators for manipulating variables. Its built-in types are the most common ones found in high-level programming languages, with a few particularities. Also, control flows and functions are very similar to typical ones.

7.1.2.1 Why Dart?

The Dart language, developed by **Google**, is a programming language that can be used to develop web, desktop, server-side, and mobile applications. Dart is the programming language used to **code Flutter apps,** enabling it to provide the best experience to the developer for the creation of high-level mobile applications.

7.1.2.2 Features of Dart:

Dart aims to aggregate the benefits of most of the high-level languages with mature language features, including the following:

- **Productive tooling:** This includes tools to analyze code, integrated development environment (IDE) plugins, and big package ecosystems.

- **Garbage collection:** This manages or deals with memory deallocation (mainly memory occupied by objects that are no longer in use).
- **Type annotations (optional):** This is for those who want security and consistency to control all of the data in an application.
- **Statically typed:** Although type annotations are optional, Dart is **type-safe** and uses type inference to analyze types in runtime. This feature is important for finding bugs during compile time.
- **Portability:** This is not only for the web (transpiled to JavaScript), but it can be natively compiled to ARM and x86 code.

7.1.2.3 The Evolution of Dart:

Unveiled in 2011, Dart has been evolving ever since. Dart saw its stable release in 2013, with major changes included in the release of Dart 2.0 toward the end of 2018:

- It was focused on web development in its conception, with the main aim of replacing JavaScript: However, now Dart is focused on mobile development areas as well as on Flutter.
- It tried solving JavaScript's problems: JavaScript doesn't provide the robustness that many consolidated languages do. So, Dart was launched as a mature successor to JavaScript.
- It offers the best performance and better tools for large-scale projects: Dart has modern and stable tooling provided by IDE plugins. It's been designed to get the best possible performance while keeping the feel of a **dynamic language.**
- It is molded to be robust and flexible: By keeping the type annotations optional and adding OOP features, Dart balances the two worlds of flexibility and robustness.

Dart is a great modern cross-platform, general-purpose language that continually improves its features, making it more mature and flexible. That's why the Flutter framework team chose the Dart language to work with.

7.1.2.4 UNDERSTANDING WHY FLUTTER USES DART:

The Flutter framework aims to be a game-changer in mobile app development, providing all of the tools needed by the developer to make awesome applications with no drawbacks in performance and scalability. Flutter has, in its core structure, multiple concepts focused on app performance and the user interface. To deliver the best output to the developing world with high performance that compares to the official native SDKs, Flutter uses the support of Dart to provide tools that contribute to developer productivity in the development phase and

to build applications optimized for publication. Let's understand why Dart was the perfect choice for the Flutter framework.

Combines Ahead-of-Time and Just-in-Time Compilation:

In creation, developers typically need to decide on a certain kind of compilation for their programming language. That is, whether the program is to be executed before or during runtime. Ahead-of-Time (AOT) compiled programs generally run quicker since they are compiled before execution. However, in reality, with ahead of time compilation, the development itself is decelerated. Just-in-Time (JIT) compilation, on the other hand, helps quicken development processes but reduces the application's initialization speed, as the compiler executes the code at or just before runtime. Overcoming these issues, Flutter uses JIT compilation for development and AOT for launching the application, thus ensuring an optimal end-user experience.

Eliminates XML Files:

Dart has a declarative and programmable layout that is easy to read and visualize. Hence, Flutter doesn't require a separate declarative layout language like XML. It is easy for Flutter to provide advanced tooling since all the layout in one language and in a central place.

Eliminates the JavaScript Bridge:

The application runs seamlessly on a user's gadget as Dart compiles and executes directly into native code, without an intermediary bridge (e.g., JavaScript to Native). This allows Dart to render hitchless animations, while user interfaces are better even at 60fps. Dart can further perform object allocation and garbage collection without the acquisition of locks.

Scaffolding:

The scaffold in Flutter is very distinct from that of iOS or React Native or even the Android UI. Among other things, it stretches to fill in the space available. Usually, this means it will fill the entire screen of its device window. The scaffold incorporates APIs for an app bar, a floating button, drawers, and the bottom sheets to enforce the graphic interface arrangement of the primary content design.

Incorporates HTTP:

To host HTTP resources from the internet, Dart incorporates an abstraction named Future. Here, the http.get() function returns a 'Future' that contains a 'Response'. While the Future is a core class in Dart to deal with asynchronous functions, a future-object reflects a possible value or error and will be visible in the Future at any point.

Maturity:

Despite being a relatively new language, Dart is not poor or lacking in resources. On the contrary, in version 2, it already has various modern language resources that help the developer to write effective high-level code

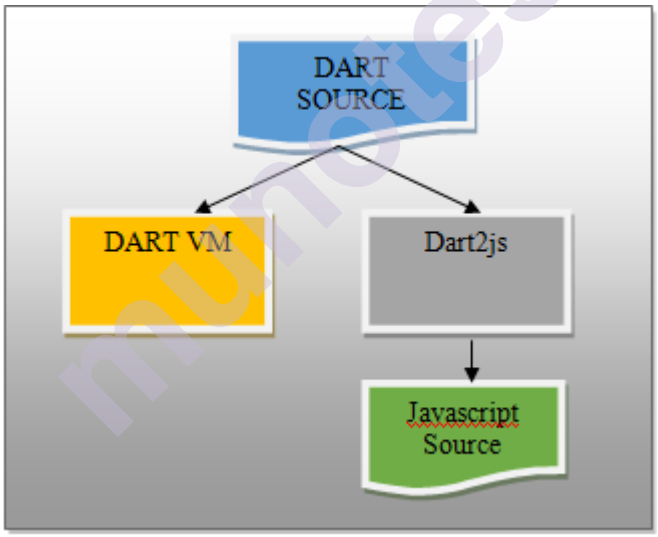
Easy learning:

Dart is a new language for many developers, and learning a new framework and a new language at the same time can be challenging. However, Dart makes this task simple by not reinventing concepts, just fine-tuning them and trying to make them as effective as possible for designated tasks.Dart is inspired by many modern and mature languages such as Java, JavaScript, C#, Swift, and Kotlin .

7.1.2.5 How Dart Works:

Dart language's flexibility comes from, the way Dart code run. This is done in two ways:

- Dart Virtual Machines (VMs)
- JavaScript compilations



Javascript Source

Dart Vm and Javascript Compilation:

Dart code can be run in a Dart-capable environment. A Dart-capable environment provides essential features to an app, such as the following:

- Runtime systems
- Dart core libraries
- Garbage collectors

The execution of Dart code operates in two modes—Just-In-Time (JIT) compilation or

Ahead-Of-Time (AOT) compilation:

- A JIT compilation is where the source code is loaded and compiled to native machine code by the Dart VM on the fly. It is used to run code in the command line interface or when you are developing a mobile app in order to use features such as debugging and hot reloading.
- An AOT compilation is where the Dart VM and your code are precompiled and the VM works more like a Dart runtime system, providing a garbage collector and various native methods from the Dart software development kit (SDK) to the application.

Hands-On Dart:

The way Flutter is designed is heavily influenced by the Dart language. So, knowing this language is crucial for success in the framework. Let's start by writing some code to understand the basics of the syntax and the available tools for Dart development.

Dartpad:

The easiest way to start coding is to use the DartPad tool (https://dartpad.dartlang.org/). It is a great online tool to learn and experiment with Dart's language features. It supports

Dart's core libraries, except for VM libraries such as dart:io. This is what the tool looks like:

Dart Development Tools:

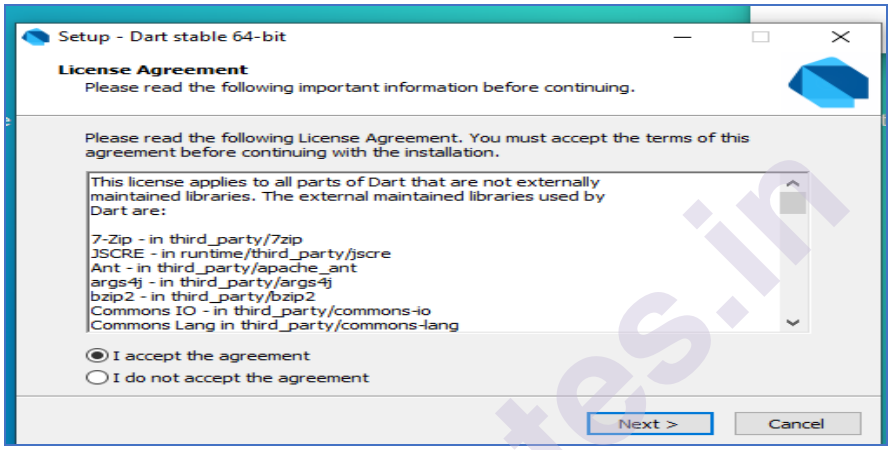
DartPad is a perfect way to start experimenting with the language without any extra effort. Since you will soon want to learn advanced things such as writing on files or using custom libraries, you'll need to have a development environment configured for that. We can do Installation of Dart on our Windows/Linux/MAC machine. Flutter is based on Dart and you can develop Dart code by having a Flutter development environment. The most common IDEs used for Dart and Flutter development are Visual Studio Code or VS Code (for the web

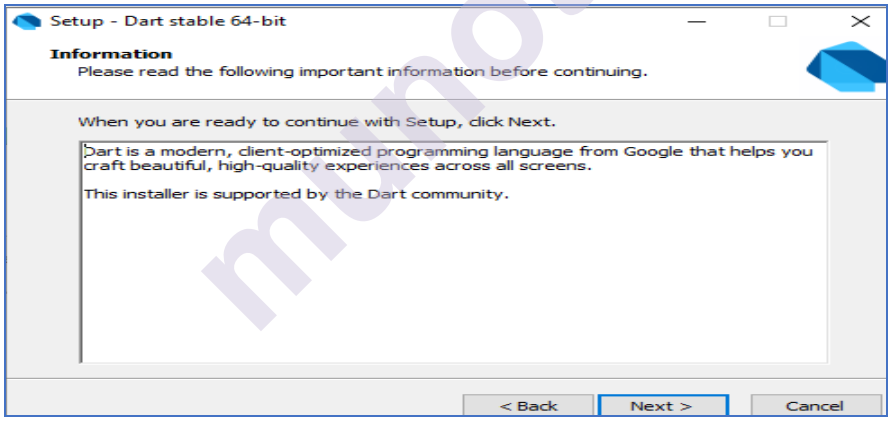
and Flutter) and Android Studio or any JetBrains IDE such as WebStorm (which is web-focused). All of the Dart functionalities of these IDEs are based on official tools, so it doesn't matter what you choose—the provided tools will be mostly the same. The Dart SDK provides specialized tools for each development ecosystem, such as web and server-side programming.

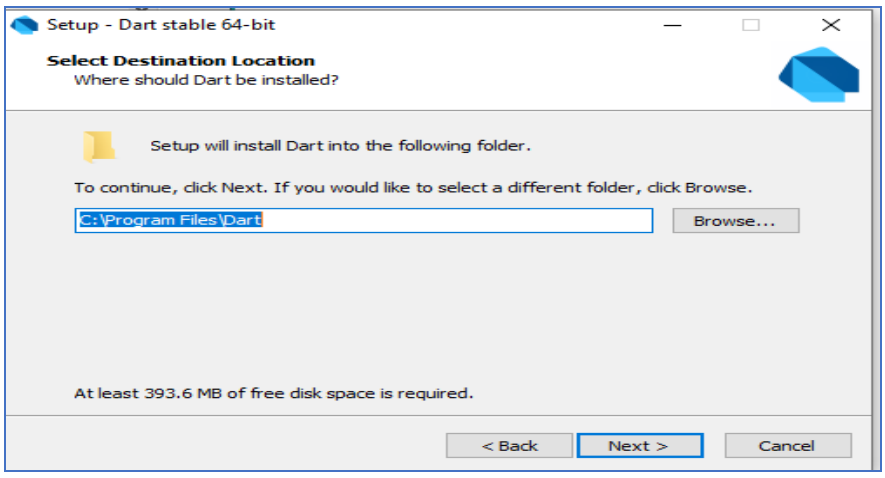
7.1.2.6 Installing Dart:

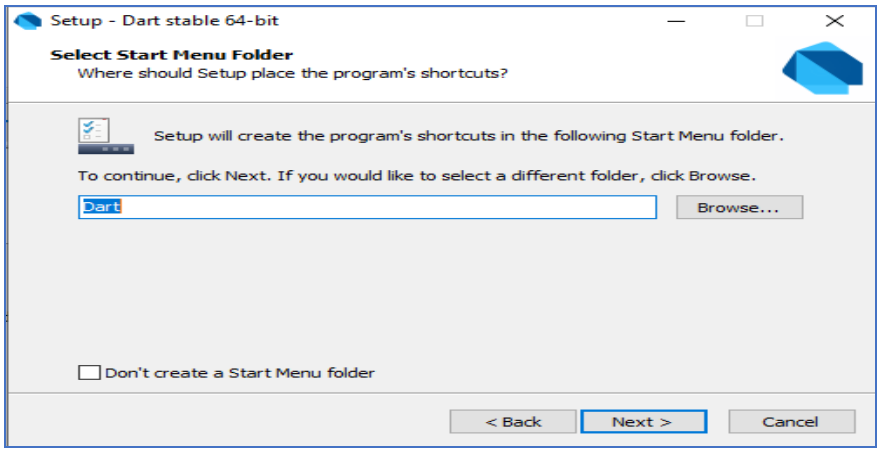
Lets learn the installation first. Below a step by step process to get Dart installed on Windows machine.

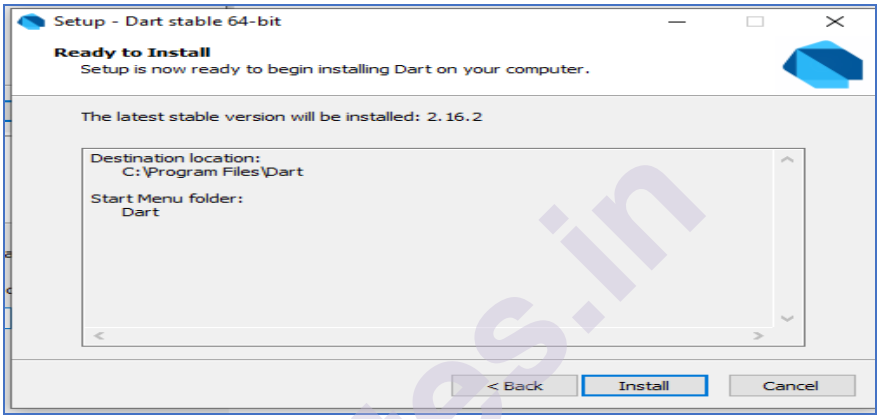
Installation for Windows Os:

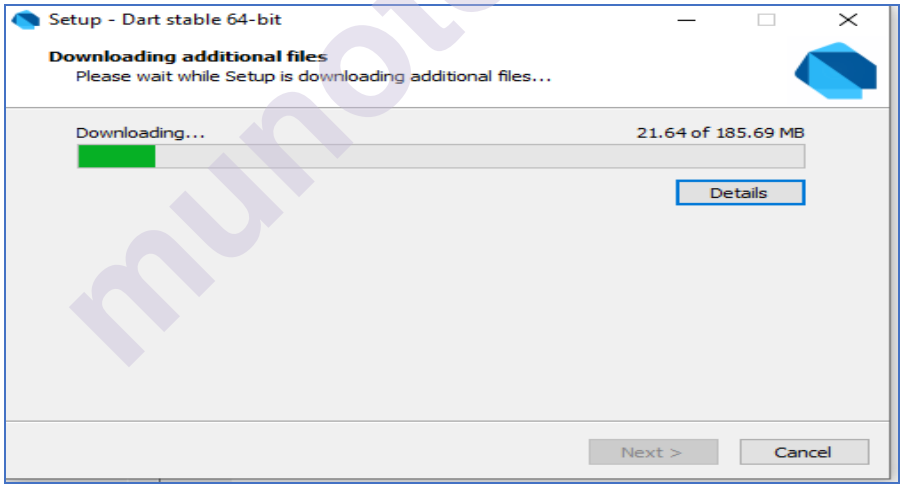


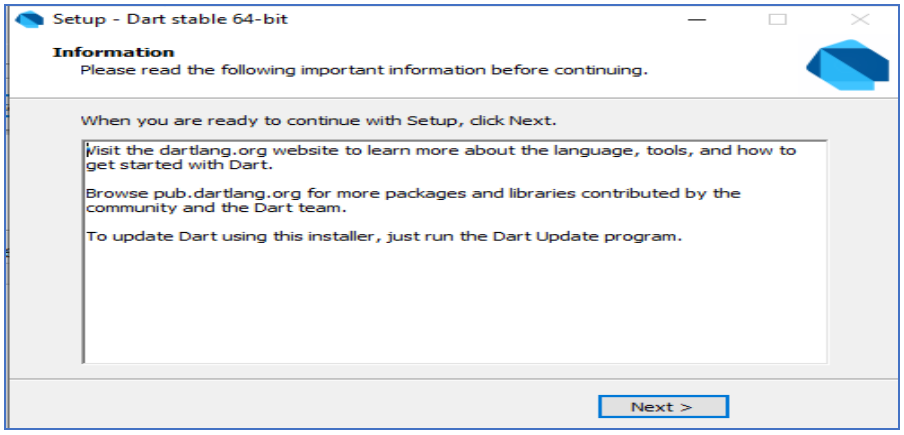


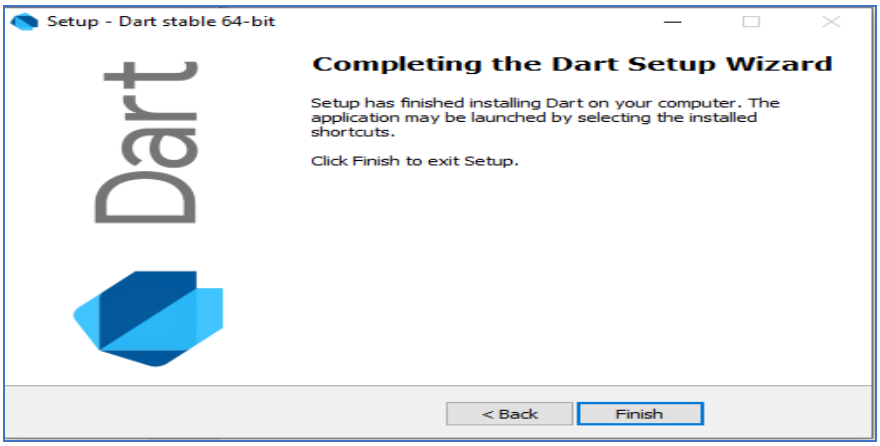



Setup - Dart stable 64-bit
Select Start Menu Folder
Where should Setup place the program's shortcuts?
Setup will create the program's shortcuts in the following Start Menu folder.
To continue, click Next. If you would like to select a different folder, click Browse.
Dart
Browse...
Don't create a Start Menu folder
< Back
Next >
Cancel

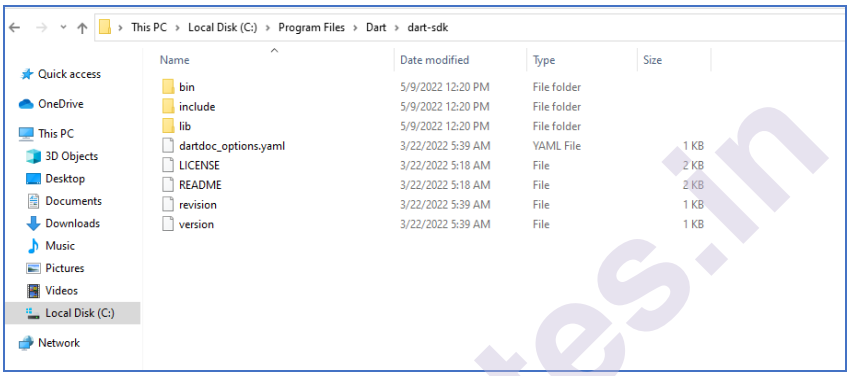

Setup - Dart stable 64-bit
Ready to Install
Setup is now ready to begin installing Dart on your computer.
The latest stable version will be installed: 2.16.2
Destination location:
C:\Program Files\Dart
Start Menu folder:
Dart
< Back
Install
Cancel


Setup - Dart stable 64-bit
Downloading additional files
Please wait while Setup is downloading additional files...
Downloading...
21.64 of 185.69 MB
Details
Next >
Cancel

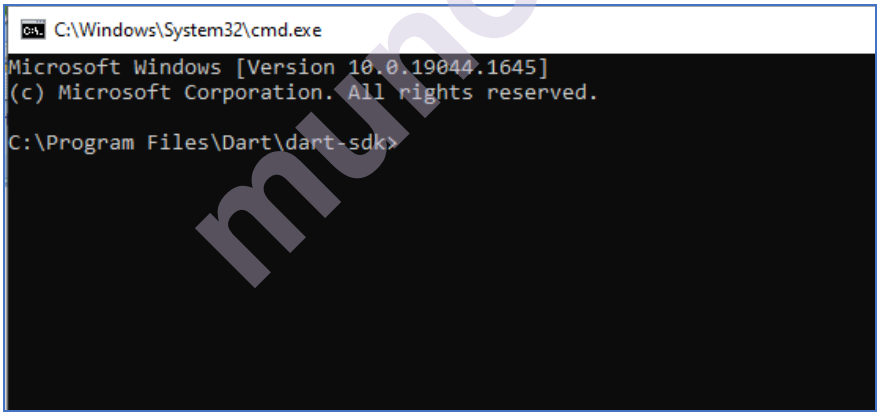

Setup - Dart stable 64-bit
Information
Please read the following important information before continuing.
When you are ready to continue with Setup, click Next.
Visit the dartlang.org website to learn more about the language, tools, and how to get started with Dart.
Browse pub.dartlang.org for more packages and libraries contributed by the community and the Dart team.
To update Dart using this installer, just run the Dart Update program.
Next >



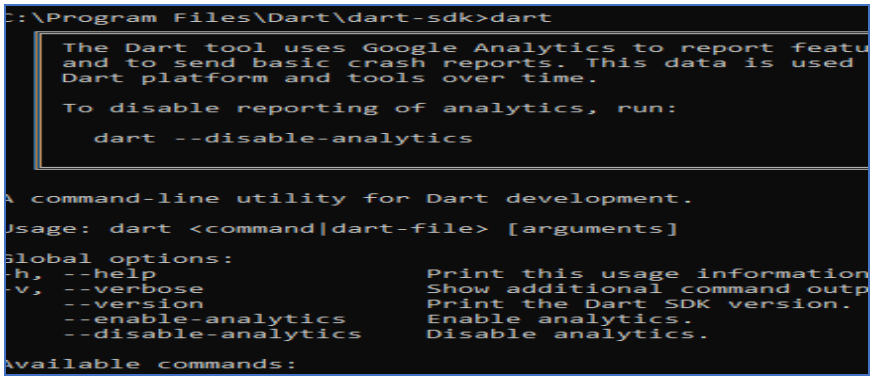
After This Go To File Location As Shown Below:



Now open the location in command-prompt

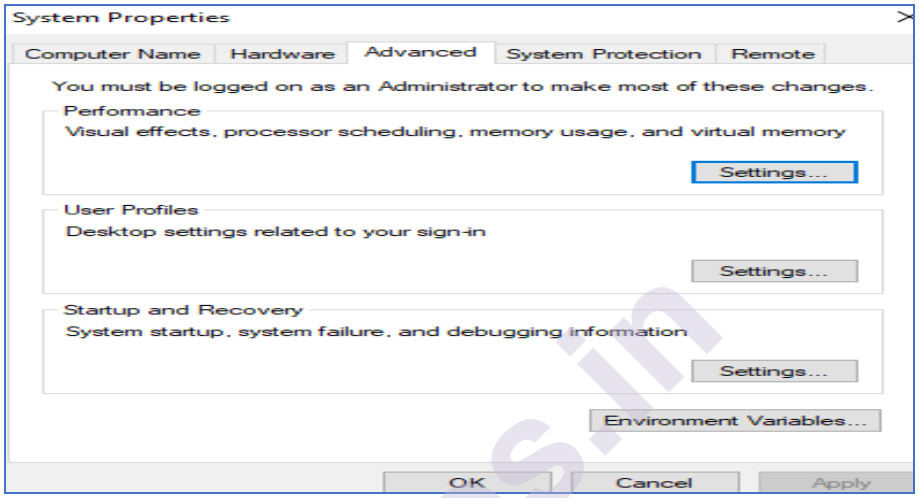


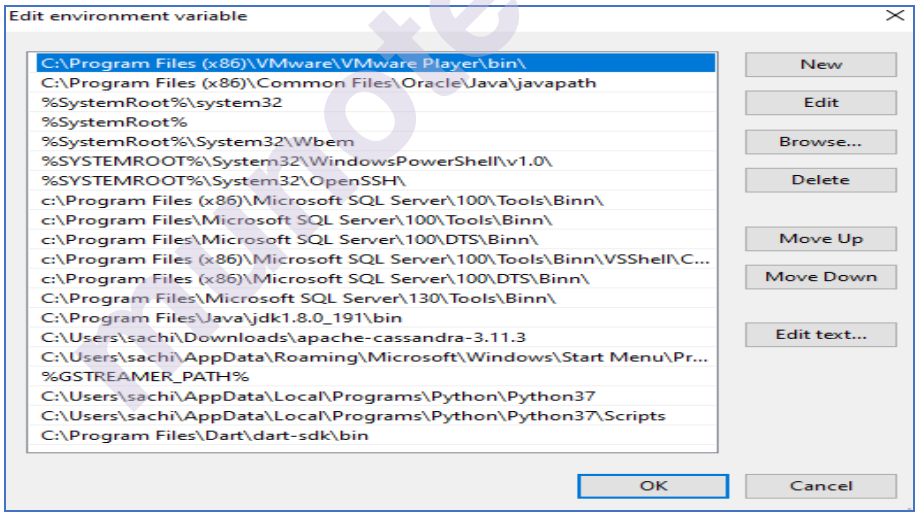
Now run the dart Command



Next **ADD DART PATH TO PATH ENVIRONMENT VARIABLE:**

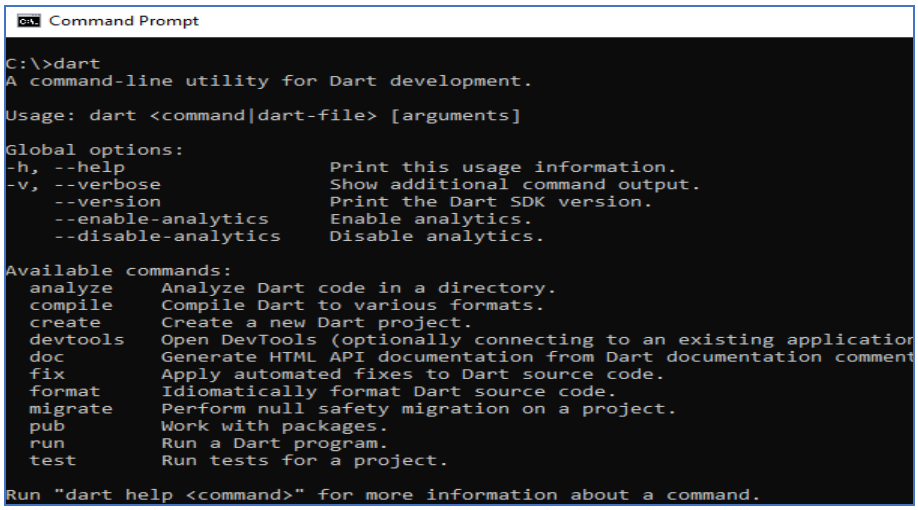
As of now, you can run dart command only from bin folder of dart sdk. To run dart command from anywhere in your file system, add dart bin path to PATH environment variable. Open Environment Variables. Under System variables, click on Path and click Edit button. Edit environment variable window appears. Click on New and paste the dart sdk bin path as shown below.





Now, Restart Command Prompt.

Close the existing command prompt window and open a new command prompt. Just run the dart command from any working directory. We shall run from C



:\.

For Linux OS:

If you're using Debian/Ubuntu on AMD64 (64-bit Intel), you can choose one of the following options, both of which can update the SDK automatically when new versions are released.

1. Install using apt-get
2. Install a Debian package

To Install using apt-get: Perform the following one-time setup:

- $ sudo apt-get update
- $ sudo apt-get install apt-transport-https
- $sudo sh -c 'wget -qO- https://dl-ssl.google.com/linux/linux_signing_key.pub | apt-key add -'
- $sudo sh -c 'wget -qO- https://storage.googleapis.com/download.dartlang.org/linux/debian/dart_stable.list > /etc/apt/sources.list.d/dart_stable.list'

Then install the Dart SDK:

- $ sudo apt-get update
- $ sudo apt-get install dart

Install a Debian package:

Download Dart SDK as Debian package in the .deb package format.

Modify PATH for access to all Dart binaries. After installing the SDK, add its bin directory to your PATH. For example, use the following command to change PATH in your active terminal session:

- export PATH="$PATH:/usr/lib/dart/bin"

To change the PATH for future terminal sessions, use a command like this:

- echo 'export PATH="$PATH:/usr/lib/dart/bin"' >> ~/.profile

7.1.3 INTRODUCING THE STRUCTURE OF THE DART LANGUAGE

If you already know some programming languages inspired by the old C language or have some experience of JavaScript, much of the Dart syntax will be easy for you to understand. Dart provides the most typical operators for manipulating variables. Its built-in types are the most common ones found in high-level programming languages, with a few particularities. Also, control flows and functions are very similar to typical ones. Let's review some of the structure of the Dart programming language starting with basic syntax.

7.1.3.1 Dart Basic Syntax:

Syntax in any program determines the rules of writing a programming language. We can think of it as the grammar of the programming language. Each language specification defines its syntax. Each dart line must end with a semicolon. The darts program consists of - variables and operators, class, function, expression and programming structure, decision making, a loop structure, remarks, Libraries, and packages. The dart syntax is similar to the C language. We will write a very basic Dart program.

- First Code in Dart: In dart main() function is predefined method and acts as the entry point to the application. A dart script needs the main() method for execution of the code. Open a text editor and write the following code.
- Save the file as first.dart or some_file_name_you_like.dart. Open Command Prompt and go to the folder, in which the first.dart file is saved.
- Execute the following command to run first.dart present in the current working directory.


```
Run "dart help <command>" for more information about a command.
See https://dart.dev/tools/dart-tool for detailed documentation.

C:\Users\sachi\Desktop\IDOL>dart first.dart
Welcome to IDOL MCA COURSE

C:\Users\sachi\Desktop\IDOL>
```

```
void main()
{
  print('Hello World');
}
```

The program is run successfully and **"Welcome to IDOl MCA COURSE"** string is printed to console. Now, we shall look into the program and analyze the code. Following is our dart program.

1. Every Dart application has a main function main().Dart main() function is the entry point to the Dart application.
2. void represents that the function returns nothing.
3. Empty parenthesis after main () represents that currently our main function does not take any arguments.
4. The body of main() function is enclosed in curly braces { } .
5. print() is a high level Dart function that prints to console.

Lastly, Execution of program:

1. Online Compiler: The online compiler which support Dart is Dart Pad.
2. IDE: The IDEs which support Dart are WebStorm, Intellij, Eclipse, etc. Among them WebStorm from Jetbrains is available for Mac OS, Windows and Linux.
3. The dart program can also be compile through terminal by executing the code dart file_name.dart.

Now Let us take one more basic Dart program:-

second - Notepad

File Edit Format View Help

```
void printInteger (int aNumber)
 {
  print('The number is $aNumber.');  // Print to console
}
// This is where the app starts executing.
void main()
 {
  var number = 42; // Declare and initialize a variable.
  printInteger(number); // Call a function.
}
```

```
C:\Users\sachi\Desktop\IDOL>dart first.dart
Welcome to IDOL MCA COURSE

C:\Users\sachi\Desktop\IDOL>dart second.dart
The number is 42.

C:\Users\sachi\Desktop\IDOL>
```

```
// Define a function.
void printInteger (int aNumber)
 {
  print('The number is $aNumber.');  // Print to
console.
}
// This is where the app starts executing.
void main()
 {
  var number = 42; // Declare and initialize a
variable.
  printInteger(number); // Call a function.
}
```

Lets Analyze the the following code. This code uses many of Dart's most basic features:

Here's what this program uses that applies to all (or almost all) Dart apps:

1. // This is a comment: - A single-line comment. Dart also supports multi-line and document comments.
2. **Void :** A special type that indicates a value that's never used. Functions like printInteger() and main() that don't explicitly return a value have the void return type.
3. **Int:** Another type, indicating an integer. Some additional built-in types are String, List, and bool.
4. **42:** A number literal. Number literals are a kind of compile-time constant.
5. **print():** A handy way to display output.
6. **'...' (or "..."):** A string literal.
7. $variableName (or ${expression}): String interpolation: including a variable or expression's string equivalent inside of a string literal.

8. **main():** The special, required, top-level function where app execution starts.

9. **Var:** A way to declare a variable without specifying its type. The type of this variable (int) is determined by its initial value (42).

7.1.3.2 Dart Identifiers:

An identifiers is a name given to program elements such as variables, array, class and functions etc. An identifier is a sequence of letters, digits, and underscores, the first character of which can not be a digit. Following rules must be kept in mind while naming an identifier.

- The first character must be an alphabet (uppercase or lowercase) or can be an underscore.
- An identifier can not start with a digit.
- All succeeding characters must be alphabets or digits.
- No special characters or punctuation symbol is allowed except the underscore"_" or a dollar sign ($).
- No two successive underscores are allowed.
- Keywords can not be used as identifiers.

Note: Dart is a case-sensitive programming language, which means "Abc" and "abc" are not the same.

7.1.3.3 Dart Semantics:

Printing and String Interpolation:

The print() function is used to print output on the console, and $expression is used for the string interpolation. Below is an example.

prac - Notepad

File Edit Format View Help

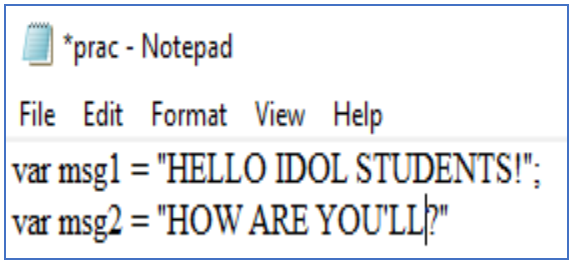
```
void main()
{
   var name = "SHYAM MOHAN";
   var roll_no = 24;
   print("My name is ${name} My roll number is ${roll_no}");
}
```

```
C:\Users\sachi\Desktop\IDOL>dart prac.dart
My name is SHYAM MOHAN My roll number is 24
```

Semicolon in Dart:

The semicolon is used to terminate the statement that means, it indicates the statement is ended here. It is mandatory that each statement should be terminated with a semicolon(;). We can write multiple statements in a single line by using a semicolon as a delimiter. The compiler will generate an error if it is not use properly.

Example:



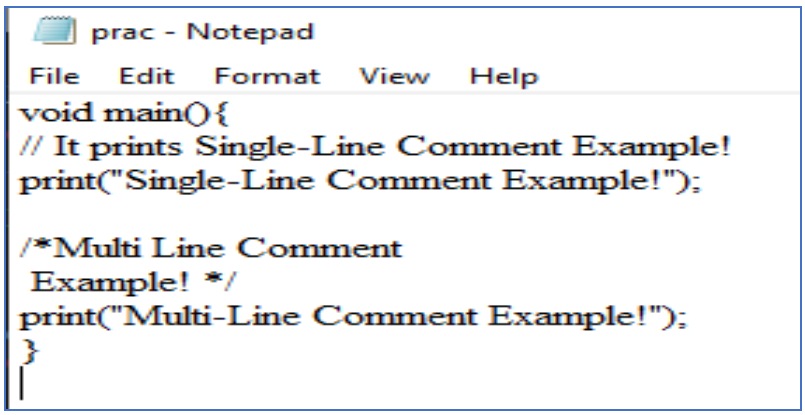
Dart Whitespace and Line Breaks:

The Dart compiler ignores whitespaces. It is used to specify space, tabs, and newline characters in our program. It separates one part of any statement from another part of the statement. We can also use space and tabs in our program to define indentation and provide the proper format for the program. It makes code easy to understand and readable.

Dart Comments:

Comments are a set of statements that are not executed by the Swift compiler and interpreter. The use of comments makes it easy for humans to understand the source code. There are two types of comments:

- **Dart Single-line Comments:** A '//' (double forward slash) is used to specify a single line comment, which extends up to the newline character.
- **Dart Multi-line Comments:** If you want to comment multiple lines then you can do it using /* and */, everything in between from /* to */ is ignored by the compiler-



```
C:\Users\sachi\Desktop\IDOL>dart prac.dart
Single-Line Comment Example!
Multi-Line Comment Example!
```

Block in Dart:-

The block is the collection of the statement enclosed in the curly braces. In Dart, we use curly braces to group all of the statements in the block. Consider the following syntax.

```
{//start of the block
//block of statement(s)
}// end of the block
```

7.1.3.4 Dart Operators:

The operators are special symbols that are used to carry out certain operations on the operands. The Dart has numerous built-in operators which can be used to carry out different functions, for example, '+' is used to add two operands. Operators are meant to carry operations on one or two operands. To make it more easier we can even think, operators as nothing more than methods defined in classes with a special syntax. So, when you use operators such as x == y, it is as though you are invoking the x.==(y) method to compare equality. As you might have noted, we are invoking a method on x, which means x is an instance of a class that has methods. In Dart, everything is an Object instance; any type you define is also an Object instance. The following are the various types of operators in Dart:

1. Arithmetic
2. Increment and decrement
3. Equality and relational
4. Type checking and casting
5. Logical operators
6. Bits manipulation
7. Null-safe and null-aware (modern programming languages provide this operator to facilitate null value handling).

Let's look at each one in more detail.

1. Arithmetic operators:

Dart comes with many typical operators that work like many languages; this includes the following:

- +: This is for the addition of numbers.
- -: This is for subtraction.
- *: This is for multiplication.
- /: This is for division.
- ~/: This is for integer division. In Dart, any simple division with / results in a double value. To get only the integer part, you would need to make some kind of transformation (that is, type cast) in other programming languages; however, here, the integer division operator does this task.
- %: This is for modulo operations (the remainder of integer division).
- - :expression: This is for negation (which reverses the sign of expression).

Some operators have different behavior depending on the left operand type; for example, the + operator can be used to sum variables of the num type, but also to concatenate strings. This is because they were implemented differently in the corresponding classes as pointed out before. Dart also provides shortcut operators to combine an assignment to a variable after another operation. The arithmetic or assignment shortcut operators are +=, -=, *=, /=, and ~/=.

2. Increment and decrement operators:

The increment and decrement operators are also common operators and are implemented in number type, as follows:

- ++var or var++ to increment 1 into var
- --var or var—to decrement 1 from var

The Dart increment and decrement operators don't have anything different to typical languages. A good application of increment and decrement operators is for count operations on loops.

3. Equality and relational operators:

The equality Dart operators are as follows:

- ==: For checking whether operands are equal
- !=: For checking whether operands are different

For relational tests, the operators are as follows:

- >: For checking whether the left operand is greater than the right one
- <: For checking whether the left operand is less than the right one

- >=: For checking whether the left operand is greater than or equal to the right one
- <=: For checking whether the left operand is less than or equal to the right one

Note:- In Dart, unlike Java and many other languages, the == operator does not compare memory references but rather the content of the variable

4. Type checking and casting:

Dart has optional typing, as you already know, so type checking operators may be handy for checking types at runtime:

- is: For checking whether the operand has the tested type
- is!: For checking whether the operand does not have the tested type

The output of this code will be different depending on the context of the execution. In DartPad, the output is true for the check of the double type; this is due to the way JavaScript treats numbers and, as you already know, Dart for the web is precompiled to JavaScript for execution on web browsers. There's also the as keyword, which is used for typecasting from a supertype to a subtype, such as converting num into int.

5. Logical operators:

Logical operators in Dart are the common operators applied to bool operands; they can be variables, expressions, or conditions. Additionally, they can be combined with complex expressions by combining the results of the expressions. The provided logical operators are as follows:

- !expression: To negate the result of an expression, that is, true to false and false to true.
- ||: To apply logical OR between two expressions.
- &&: To apply logical AND between two expressions.

6. Bits manipulation:

Dart provides bitwise and shift operators to manipulate individual bits of numbers, usually with the num type. They are as follows:

- &: To apply logical AND to operands, checking whether the corresponding bits are both 1.
- |: To apply logical OR to operands, checking whether at least one of the corresponding bits is 1.

- ^: To apply logical XOR to operands, checking whether only one (but not both) of the corresponding bits is 1.
- ~operand: To invert the bits of the operand, such as 1s becoming 0s and 0s becoming 1s.
- <<: To shift the left operand in x bits to the left (this shifts 0s from the right).
- >>: To shift the left operand in x bits to the right (discarding the bits from the left).

Like arithmetic operators, the bitwise ones also have shortcut assignment operators, and they work in the exact same way as the previously presented ones; they are <<=, >>=, &=, ^=, and |=.

7. Null-safe and null-aware operators:

Following the trend on modern OOP languages, Dart provides a null-safe syntax that evaluates and returns an expression according to its null/non-null value. The evaluation works in the following way**: if expression1 is non-null, it returns its value; otherwise, it evaluates and returns the value of expression2: expression1 ?? eexpression2.**

In addition to the common assignment operator, =, and the ones listed in the corresponding operators, Dart also provides a combination between the assignment and the null-aware expression; that is, the ??= operator, which assigns a value to a variable only if its current value is null. Dart also provides a null-aware access operator, **?.**, which prevents accessing null object members.

7.1.3.5 Variables and Data Types:

Variables are used to represent reserved memory locations that is used to store values, when we create a variable we are a suppose to allocate some memory space for that variable. Dart is a statically typed programming language. This means that variables always have a specific type and that type cannot change. Every variable have data type associated to it, data type for a variable defines. Dart uses var keyword to declare the variable. The syntax of var is defined below,

Eg:- **Var name = 'Dart';**

final and const :

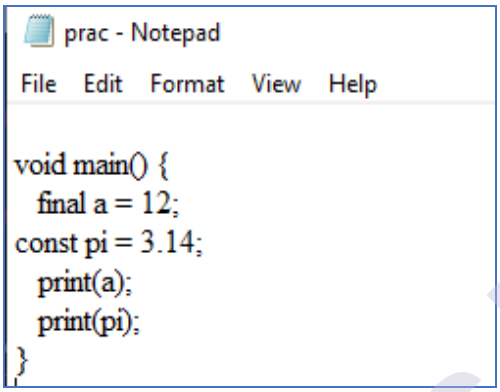
The final and const keyword are used to declare constants. A variable will never intend to change its value after it is assigned, and you can use the final and const ways for declaring this:

Eg:- **final value = 1;**

The value variable cannot be changed once it's initialized:

Eg:- **const value = 1;**

Just like the final keyword, the value variable cannot be changed once it's initialized, and its initialization must occur together with a declaration. In addition to this, the const keyword defines a compile-time constant. As a compile-time constant, the const values are known at compile time. They also can be used to make object instances or Lists immutable, as follows: const list = const [1, 2, 3] **and** const point = const Point(1,2). This will set the value of both variables during compile time, turning them into completely immutable variables. Example for Final and Constant is given below for you to try:



Dart Datatype:

The data type classification is as given below:

Data Type	Keyword	Description
Number	int, double, num	Numbers in Dart are used to represent numeric literals
Strings	String	Strings represent a sequence of characters
Booleans	Bool	It represents Boolean values true and false
Lists	List	It is an ordered group of objects
Maps	Map	It represents a set of values as key-value pairs

1. Number:

The number in Dart Programming is the data type that is used to hold the numeric value. Dart numbers can be classified as:

- The int data type is used to represent whole numbers.
- The double data type is used to represent 64-bit floating-point numbers.
- The num type is an inherited data type of the int and double types.

2. String:

It used to represent a sequence of characters. It is a sequence of UTF-16 code units. The keyword string is used to represent string literals. String values are embedded in either single or double-quotes.

3. Boolean:

It represents Boolean values true and false. The keyword bool is used to represent a Boolean literal in DART.

4. List:

List data type is similar to arrays in other programming languages. A list is used to represent a collection of objects. It is an ordered group of objects.

5. Map:

The Map object is a key and value pair. Keys and values on a map may be of any type. It is a dynamic collection.

Below are some speacial Data Type for your reference:-

6. Dart Runes:

Dart string is a sequence of Unicode UTF-16 code units, 32-bit Unicode values within a string are represented using a special syntax. A rune is an integer representing a Unicode code point. For example, the heart character '♥ is represented using corresponding unicode equivalent \u2665, here \u stands for unicode and the numbers are hexadecimal, which is essentially an integer. If the hex digits are more or less than 4 digits, place the hex value in curly brackets ({ }). For example, the laughing emoji '□' is represented as \u{1f600}.

7. Dart Symbols:

Dart Symbol object used to refer an operator or identifier declared in a Dart program. Dart symbol are commonly used in APIs that refer to identifiers by name, because an identifier name can changes but not identifier symbols. Symbol for an identifier can be created using a hash (#) followed by the identifier name.

Note: If the type of a variable is not specified, the variable's type is dynamic. The dynamic keyword is used as a type annotation explicitly.

Dart Dynamic Type:

Dart is an optionally typed language. If the type of a variable is not explicitly specified, the variable's type is dynamic. The dynamic keyword can also be used as a type annotation explicitly.

Dynamic:

If the variable type is not defined, then its default type is dynamic. The following example illustrates the dynamic type variable

```
void main() {
   dynamic name = "Dart";
   print(name);
}
```

Literals:

You can use the [] and {} syntaxes to initialize variables such as lists and maps, respectively. A literal is a notation to represent a fixed value in programming languages. These are some examples of literals provided by the Dart language for creating objects of the provided built-in types:

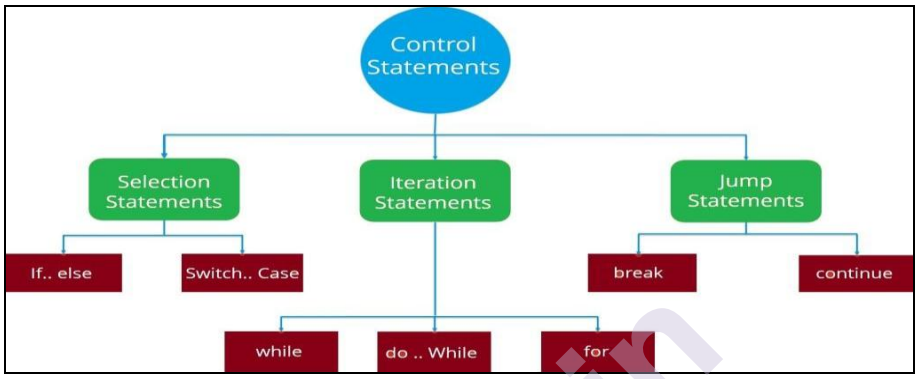
Type	Literal example
Int	10, 1, -1, 5, and 0
Double	10.1, 1.2, 3.123, and -1.2
Bool	true and false
String	"Dart", 'Dash', and """multiline String"""
List	[1,2,3] and ["one", "two", "three"]
Map	{"key1": "val1", "b": 2}

7.1.3.6 Dart Control Flow Statements:

Control flow or flow of control is the order in which instructions, statements and function calls being executed or evaluated when a program is running. The control flow statements are also called as Flow Control Statements. In Dart, statements inside your code are generally executed sequentially from top to bottom, in the order that they appear. It is not always the case your program statements to be executed straightforward one after another sequentially, you may require to execute or skip certain set of instructions based on condition, jump to another statements, or execute a set of statements repeatedly. In Dart, control flow statements are used to alter, redirect,

or to control the flow of program execution based on the application logic. In Dart, Control flow statements are mainly categorized in following types

1. Selection statements.
2. Iteration statements
3. Jump statements.



Dart Selection Statements:

In Dart, Selection statements allow you to control the flow of the program during run time on the basis of the outcome of an expression or state of a variable. Selection statements are also referred to as Decision making statements. They evaluates single or multiple test expressions which results in "TRUE" or "FALSE". The outcome of the test expression/condition helps to determine which block of statement(s) to executed if the condition is "TRUE" or "FALSE" otherwise. In Dart, we have following selection statements –

1. Dart if Statement
2. Dart If..else Statement
3. Dart Else if Ladder Statement
4. Nested if Statement
5. Dart Switch Case Statement

If Statement:

This type of statements simply checks the condition and if it is true the statements within it is executed but if it in is not then the statements are simply ignored in the code. Syntax is as given below:

```
If ( condition )
{
  // body of if
}
```

If…else Statement:

This type of statement simply checks the condition and if it is true, the statements within is executed but if not then else statements are executed. Syntax is as given below:

```
If ( condition )
{
  // body of if
}
Else {
  // body of else
}
```

Else…if Ladder:

This type of statement simply checks the condition and if it is true the statements within it is executed but if it in is not then other if conditions are checked, if they are true then they are executed and if not then the other if conditions are checked. This process is continued until the ladder is completed. Syntax is as given below:

```
If ( condition1 )
{
  // body of if
}
Else if ( condition2 )
{
  // body of if
}
Else {
  // statement
}
```

Nested if Statement:

This type of statements checks the condition and if it is true then the if statement inside it checks its condition and if it is true then the statements are executed otherwise else statement is executed. Syntax is as given below:

```
If ( condition1 )
{
  If ( condition2 )
{
     // Body of if
  }
  Else
{
    // Body of else
  }
}
```

Switch Case:

In Dart, switch-case statements are a simplified version of the nested if-else statements. Its approach is the same as that in Java. Syntax is as given below:

```
switch ( expression )
 {
  case value1:
{
    // Body of value1
  } break;
  case value2:
 {
    //Body of value2
  } break;
  default:
 {
    //Body of default case
  } break;
}
```

The default case is the case whose body is executed if none of the above cases matches the condition. Rules to follow in switch case:

- There can be any number of cases. But values should not be repeated.
- The case statements can include only constants. It should not be a variable or an expression.
- There should be a flow control i.e break within cases. If it is omitted than it will show error.
- The default case is optional.
- Nested switch is also there thus you can have switch inside switch.

Dart Iteration Statements:

In Dart, Iteration statements are used to execute the block of code repeatedly for a specified number of times or until it meets a specified condition. Iteration statements are commonly known as loops or looping statements. In Dart, we have following iteration statements available:

1. for loop
2. for… in loop
3. for each loop
4. while loop
5. do-while loop

Let us understand a simple example about the usage of control statements and loops

For loop:

For loop in Dart is similar to that in Java and also the flow of execution is the same as that in Java. Syntax is given below:

```
for (initialization; condition; test expression)
{
        // Body of the loop
}
```

Control flow: Control flow goes as:

- Initialization
- Condition
- Body of loop
- Test expression

The first is executed only once i.e in the beginning while the other three are executed until the condition turns out to be false.

For…in loop:

For…in loop in Dart takes an expression or object as an iterator. It is similar to that in Java and its flow of execution is also the same as that in Java.

Syntax is as given below:

```
for (var in expression)
{
  // Body of loop
}
```

1. **For each … loop :** The for-each loop iterates over all elements in some container/collectible and passes the elements to some specific function. Syntax is as given below:-

```
Collection.foreach(void
f(value))
```

Parameters: F(value): It is used to make a call to the f function for each element in the collection.

2. **While loop:** The body of the loop will run until and unless the condition is true.

Syntax is as given below:

```
While(condition)
{
   Text expression;
   // Body of loop
}
```

3. **Do..while loop:** The body of the loop will be executed first and then the condition is tested. Syntax is as given below:-

```
Do
{
   Text expression;
   // Body of loop
}while(condition);
```

Dart Jump Statements:

Jump statements are used to alter or transfer the control to other section or statements in your program from the current section. In Dart, we have following types of jump statements –

1. Dart Break Statement
2. Dart Continue Statement

Break :

The break statement inside any loop gives you way to break or terminate the execution of loop containing it, and transfers the execution to the next statement following the loop.If break statement is used in nested loops, only the immediate loop will be broken. Break statement can be used inside a For loop, While loop and Do-while loop statements. Syntax is as given below:-

Eg:- **Break;**

Refer the example:

prac - Notepad

File Edit Format View Help

```
void main()
{
   int count = 1;

   while (count <= 10) {
      print("Hi, you are inside loop $count");
      count++;

      if (count == 4) {
         break;
      }
   }
   print("Hi, you are out of while loop");
}
```

```
C:\Users\sachi\Desktop\IDOL>dart prac.dart
Hi, you are inside loop 1
Hi, you are inside loop 2
Hi, you are inside loop 3
Hi, you are out of while loop
```

Continue Statement:

While the break is used to end the flow of control, continue on the other hand is used to continue the flow of control. When a continue statement is encountered in a loop it doesn't terminate the loop but rather jump the flow to next iteration. If continue statement is used in nested loops, only the immediate loop is continued with. Continue statement can be used inside a For loop, While loop and Do-while loop statements. Syntax is as given below

Eg:- **Continue;**

prac - Notepad

File Edit Format View Help

```
void main()
{
   for (int i = 1; i <= 10; ++i) {

      if (i == 2) {
         print("Hi, you are inside loop $i");
         continue;
      }
   }

   print("Hi, you are out of loop");
}
```

```
C:\Users\sachi\Desktop\IDOL>dart prac.dart
Hi, you are inside loop 2
Hi, you are out of loop
```

7.1.3.7 Functions:

Function is a set of statements that take inputs, do some specific computation and produces output. Functions are created when certain statements are repeatedly occurring in the program and a function is created to replace them. Functions makes it easy to divide the complete program into sub-units that perform a specific task for that program, this way it enhance the modular approach and increase the code re-usability of the program. We pass information in function call as its parameter and the function can either returns some value to the point it where it called from or returns nothing.

Advantages of function:

- It enhance the modularity of the program.
- It enhance the re usability.
- It makes development easy as development can be shared in team.
- It reduces the coupling.
- It reduces duplication.

Defining the function in Dart:

A function must be defined prior to use otherwise this will show a compile time error as the main() function is unaware of the function, its argument list and the return type. Call to the function cannot be made unless it is defined. Below is the general syntax of a Dart function. Syntax is as given below:

```
Return_type function_name (
parameters )
 {
  // Body of function
  Return value;
}
```

In the above syntax:

- **function_name:** defines the name of the function.
- **parameters:** It represents the list of the parameters need to be passed when function call made.
- **Return_type:** defines the datatype in which output is going to come.
- **Return value:** defines the value to be returned from the function.

Calling function In Dart:

In Dart, once a function is defined, later you can invoke or call this function inside main() function body. A function can be invoked simply by its name with argument list if any. The function is called as per the syntax given below:-

Eg: Function_name (argument_list);

- In the above syntax:
- **Function_name :-** defines the name of the function.
- **Argument list :-** is the list of the parameter that the function requires.

Dart Passing Arguments to Function:

In Dart, when a function is invoked it may be provided with some information as per the function prototype is called as argument (parameter). The number of values passed and the data type of the value passed must match the number of parameters and the data type of the parameter defined during its declaration . Otherwise, the compiler throws an error.

```
void main() {
     ... .. ...
     sum = add( num1 , num2 );   // Actual parameters: num1 and num2
    ... .. ...
}

int add( int n1 , int n2 ) {      // Formal parameters: n1 and n2
     ... .. ...
     result = n1 + n2;                                    ;
    ... .. ...                                            ;
}
```

Let us look into a simple function in Dart as shown here:

prac - Notepad

File Edit Format View Help

```
void main(){
  print("Dart Program To Add Two Numbers Using Function.");
  var sum = add(10,20);
  print("Sum Of Given No. Is : ${sum}");
}

int add(int n1, int n2){
   int result;
   result = n1+n2;
   return result;
}
```

```
C:\Users\sachi\Desktop\IDOL>dart prac.dart
 Dart Program To Add Two Numbers Using Function.
Sum Of Given No. Is : 30

C:\Users\sachi\Desktop\IDOL>
```

Example:

prac - Notepad

File Edit Format View Help

```
void main() {
  print( factorial(3));
}
factorial(number) {
  if (number <= 0) {
    // termination case
    return 1;
  } else {
    return (number * factorial(number - 1));
    // function invokes itself
  }
}
```

```
C:\Users\sachi\Desktop\IDOL>dart prac.dart
6

C:\Users\sachi\Desktop\IDOL>
```

```
return_type func_name()
{
  //Statement(s)
  return value;
}
```

Now Let us also study the basic syntax of the following:

1. Dart Function With No Parameters and Return Value.
2. Dart function without Parameters and Without Return Value.

Dart Function With No Parameters and Return Value :- Syntax is as given below:

Above is the general syntax of a Dart function With No Parameters and Return Value, here:

- **func_name:** It is replaced with the name of the function.
- **return_type:** It represents return_type of the function. The return_type can be any valid data type. The data type of the value returned must match the return type of the function.

Dart function without Parameters and Without Return value Syntax is as given below:

```
fun_name()
{
//statements
}
Or
void fun_name()
{
//statements
}
```

- **void:** It is used to represent no return value.
- **fun_name:** It is replaced with the name of the function.

Function parameters:

A function can have two types of parameters: optional and required. Additionally, as with most modern programming languages, these parameters can be named on call to make the code more readable. The parameter type doesn't need to be specified; in this case, the parameter assumes the dynamic type:

Required parameters:

This simple function definition with parameters is achieved by just defining them in the same way as most other languages. In the following function, both name and additionalMessage are required parameters, so the caller must pass them when calling it:

```
sayHello(String name, [String additionalMessage]) =>
"Hello $name.$additionalMessage";
```

Optional positional parameters:

```
sayHello(String name, String additionalMessage) =>
"Hello $name.$additionalMessage";
```

Sometimes, not all parameters need to be mandatory for a function, so it can define optional parameters as well. The optional positional parameter definition is done by using the [ ] syntax. Optional positional parameters must go after all of the required parameters, as follows:

Optional named parameters:

The optional named parameter definition is done by using the { } syntax. They must also go after all of the required parameters:

```
sayHello(String name, {String additionalMessage}) =>
"Hello $name.$additionalMessage";
```

Dart Anonymous Functions:

- Every function in dart is an object. In Dart, most of the functions we have seen are named functions; we can also create nameless function knows as **anonymous function, lambda or closure**.
- An anonymous function is like named function but they do not have name associated with it. An anonymous function can have zero or more parameters with optional type annotations. We can assign the anonymous function to a variable, and then we can retrieve or access the value of the closure based on our requirement.

```
(parameter_list)
 {
  statement(s)
}
```

- An Anonymous function contains an independent block of the code, and that can be passed around in our code as function parameters. In short, if your function has only one expression to return then to quickly represent it in just one line you can use Anonymous/ Lambda function. The syntax is as follows.

Dart Recursive Function:

Recursive functions are quite similar to the other functions, but difference is to calling itself recursively. A recursive function repeats multiple times until it returns the final output. It allows programmers to solve complex problems with minimal code. Syntax is as given below:

```
void recurse()
 {
  //statement(s)
 recurse();
//statement(s);
}
void main()
{
   //statement(s)
  recurse();
 //statement(s)
}
```

How does recursion works?

Let's understand the concept of the recursion of the example of a factorial of a given number. In the following example, we will evaluate the factorial of n numbers. It is the series of the multiplication as follows.

Factorial of n (n!) = n*(n-1)*(n-2).....1

Characteristics of Recursive Function

- A recursive function is a unique form of a function where function calls itself.
- A valid base case is required to terminate the recursive function.
- It is slower than the iteration because of stack overheads.

Lexical Scope:

The Dart scope is determined by the layout of the code using curly braces like many programming languages; the inner functions can access variables all the way up to the global level. Variables/Closure functions are only accessed inside the same block that is defined. Variable declared in curly braces are available to access inside the same block as well as nested scopes. You can define the nested blocks using {} or functions In Dart, Block, or {} define the scope In the Dart file, Variables declare outside the main function are global variables Global Variables are accessible to all functions including the main function.

lex - Notepad

File Edit Format View Help

```
bool topVariable = true;

void main() {
  var inside_Main = true;
  // Defining Nested Function

  void myFunction() {
    var inside_Function = true;

  void nestedFunction() {
      var inside_NestedFunction = true;
      // This function is using all variable of the previous functions.
      assert(topVariable);
      assert(inside_Main);
      assert(inside_Function);
      assert(inside_NestedFunction);
    }
  }
}
```

7.1.3.8 Dart Exception Handling:

EXCEPTION:

An exception is an error that takes place inside the program. When an exception occurs inside a program the normal flow of the program is disrupted and it terminates abnormally, displaying the error and exception stack as output. So, an exception must be taken care to prevent the application from termination. To prevent the program from exception we make use of try/on/catch blocks in Dart.

Dart try-catch is used to execute a block of code that could throw an exception, and handle the exception without letting the program terminate. If an exception, thrown by any of the code, is not handled via catch block, then the program could terminate because of the exception. Every built-in exception in Drat comes under a pre-defined class named Exception. To prevent the program from exception we make use of try/on/catch blocks in Dart. In this topic, we will learn how to use try-catch in Dart code, to catch the exceptions thrown, if any, by a piece of code enclosed in the try block. The syntax of try-catch to catch any type of exception is:

```
try {
  // program that might throw an
exception
}
on Exception1 {
  // code for handling exception
1
}
catch Exception2 {
  // code for handling exception
2
}
```

Final block:

The final block in dart is used to include specific code that must be executed irrespective of error in the code. Although it is optional to include finally block if you include it then it should be after try and catch block are over. Syntax is as given below:

```
finally
{
  ...
}
```

Eg:

exception - Notepad

File Edit Format View Help

```
void main() {
int i= 10;
try{
        var result = i ~/ 0;
        print(result);
}
catch(e){
        print(e);
}
finally {
        print("Code is at end.");
}
}
```

```
C:\Users\sachi\Desktop\IDOL>dart exception.dart
IntegerDivisionByZeroException
Code is at end.
```

Custom Exception:

Unlike other languages, in Dart one can create a custom exception. To do, so we make use of throw new keyword in the dart. Let us understand this with an example of Creating custom exceptions in the dart.

custom - Notepad

File Edit Format View Help

```
void displayName(str) {
  if (str.length > 0) {
    print(str);
  } else {
    throw new Exception('Name is empty.');
  }
}

void main() {
  var name = '';
  displayName(name);
  print('Bye');
}
```

```
C:\Users\sachi\Desktop\IDOL>dart custom.dart
Unhandled exception:
Exception: Name is empty.
#0      displayName (file:///C:/Users/sachi/Desktop/IDOL/custom.dart:5:5)
#1      main (file:///C:/Users/sachi/Desktop/IDOL/custom.dart:11:3)
#2      _delayEntrypointInvocation.<anonymous closure> (dart:isolate-patch/isolate_patch.dart:297:19)
#3      _RawReceivePortImpl._handleMessage (dart:isolate-patch/isolate_patch.dart:192:12)
```

In the following example, we write a method to display a string passed to it. If the given string is empty, this method throws a custom Exception, i.e., EmptyNameException. We handle calls to this function using a try-catch block

custom - Notepad

File Edit Format View Help

```
class EmptyNameException implements Exception {
  String cause;
  EmptyNameException(this.cause);
}

void displayName(str) {
  if (str.length > 0) {
    print(str);
  } else {
    throw new EmptyNameException('Name is empty.');
  }
}

void main() {
  var name = '';
  try {
    displayName(name);
  } on EmptyNameException {
    print('Given name is empty.');
  }
  print('MY OWN CUSTOM EXCEPTION');
}
```

```
C:\Users\sachi\Desktop\IDOL>dart custom.dart
Given name is empty.
MY OWN CUSTOM EXCEPTION
```


7.1.3.9 Data Structures, Collections, And Generics:

Dart provides multiple kinds of structures to manipulate a of values. Dart lists are widely used even in the most simple use cases. Generics are an concept when working with collections of data tied to a specific type, such as List or Map, for example. They ensure a collection will have homogeneous values by specifying the type of data it can hold.

Dart Collection:

Collection is a group of multiple objects represented as a single unit. In Dart, the “dart:collection” library encompasses classes and utilities that supplement the collection support in dart:core. The “dart:collection” library is a collection of interfaces and classes which helps in storing and processing the data efficiently. Dart collection can be classified as

Iterating Collections:

The dart::core library provides the iterator class, which enables the easy collection traversal. As we know that, every collection contains an iterator property. This property returns an iterator that point to the objects in the collection.

Dart Generics:

are the same as the Dart collections, which are used to store the homogenous data. As we discussed in the Dart features, it is an optionally typed language. By default, Dart Collections are the heterogeneous type. In other words, a single Dart collection can hold the values of several data types. However, a Dart collection can be also stored the homogenous values or same type values. The Dart Generics provides the facility to enforce a limit on the data type of the values that can be stored by the collection. These collections can be referred to as the type-safe collections. Type safety is a unique feature of the Dart programming, which makes sure that a memory block can only contain the data of a specific data type. The generics are a way to support type-safety implementation for all Dart collections. The pair of the angular bracket is used to declare the type-safe collection. The angular bracket consists of the data-types of the collection. The syntax is given below.

```
Collection_name <data_type> identifier = new
Collection_name<data_type>
```

When and Why to Use Generics:

The use of generics can help a developer to maintain and keep collection behavior under control. When we use a collection without specifying the allowed element types, it is our responsibility to correctly insert the elements. This, in a wider context, can become expensive, as we need to implement validations to prevent wrong insertions and to document it for a team.

7.1.4 OOPS CONCEPT AND CLASSES & PACKAGES IN DART PROGRAMMING

Introduction to OOP in Dart:

In Dart, everything is an object, including the built-in types. Upon defining a new class, even when you don't extend anything, it will be a descendant of an object. Dart implicitly does this for you. Dart is called a true object-oriented language; and supports the concepts of class, object, interfaces, inheritance, mixins, and abstract classes etc. In Dart, a class can be defined as a blueprint or prototype of associated objects. Class is a wrapper that binds/encapsulates the data and methods together; which can be later accessed by objects of that class. We can think of a class as user-defined data type, that describes

the behavior and characteristics shared by all of its instances. Once a class has been defined, we can create instance or objects of that class which has access to class properties and methods. Even functions are objects, which means that you can do the following:

- Assign a function as a value of a variable.
- Pass it as an argument to another function.
- Return it as a result of a function as you would do with any other type, such as String and int.

This is known as having first-class functions because they're treated the same way as other types. Another important point to note is that Dart supports single inheritance on a class, similar to Java and most other languages, which means that a class can inherit directly from only a single class at a time. A class can implement multiple interfaces and extend multiple classes using mixins.

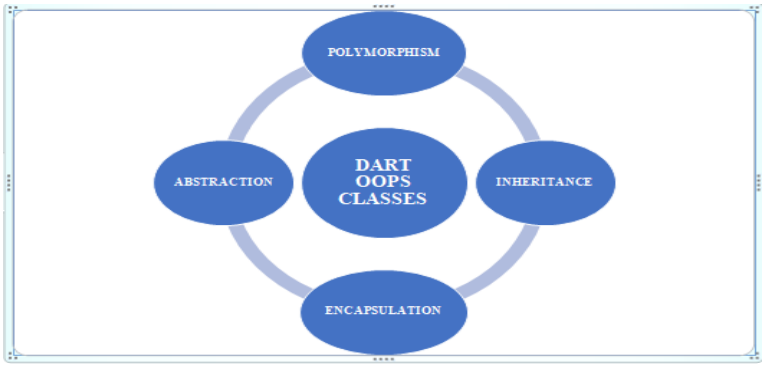
7.1.4.1 Oops Artificats & Features:

Oops Artifacts:

1. Class: This is a blueprint for creating an object.
2. Interface: This is a contract definition with a set of methods available on an object. Although there is no explicit interface type in Dart, we can achieve the interface purpose with abstract classes.
3. Enumerated class: This is a special kind of class that defines a set of common constant values.
4. Mixin: This is a way of reusing a class's code in multiple class hierarchies.

Dart OOP features:

Every programming language can provide the OOP paradigm in its own way, with partial or full support, by applying some or all of the following principles:



Below is the brief introduction of these oops concepts.

1. Class:

Dart classes are defined as the blueprint of the associated objects. A Class is a user-defined data type that describes the characteristics and behavior of it. To get all properties of the class, we must create an object of that class. Dart classes can have both instance members (methods and fields) and class members (static methods and fields). Dart classes do not support constructor overloading, but you can use the flexible function argument specifications from the language (optional, positional, and named) to provide different ways to instantiate a class. Also, you can have named constructors to define alternatives. In Dart, a class can be defined using the class keyword followed by the class name; and the class body enclosed by a pair of curly braces ({}). The syntax of the class is given below:-

```
class class_name
 {
   // Body of class
}
```

In the above syntax:

- Class is the keyword use to initialize the class.
- class_name is the name of the class.
- Body of class consists of fields, constructors, getter and setter methods, etc.

2. Object:

The starting point of OOP, objects, are instances of defined classes. In Dart, as has already been pointed out, everything is an object, that is, every value we can store in a variable is an instance of a class. Besides that, all objects also extend the Object class, directly or indirectly. An object is a real-life entity such as a table, human, car, etc. The object has two characteristics - state and behavior. Let's take an example of a car which has a name, model name, price and behavior moving, stopping, etc. The object-oriented programming offers to identify the state and behavior of the object. We can access the class properties by creating an object of that class. In Dart, The object can be created by using a new keyword followed by class name. The syntax is given below.

```
var objectName = new
ClassName(<constructor_arguments>)
```

In the above syntax:

- new is the keyword use to declare the instance of the class
- object_name is the name of the object and its naming is similar to the variable name in dart.
- class_name is the name of the class whose instance variable is been created.
- <constructor_arguments> are the input which are needed to be pass if we are willing to call a constructor. The argument should be passed values if the class constructor is parameterized.

After the object is created, there will be the need to access the fields which we will create. We use the dot(.) operator for that purpose.

```
// For accessing the property
object_name.property_name;

// For accessing the method
object_name.method_name();
```

3. Inheritance:

Dart supports inheritance, which is used to create new classes from an existing class. The class that to be extended is called parent /superclass, and the newly created class is called child/subclass. Dart permits single direct inheritance. Dart has special support for mixins, which can be used to extend class functionalities without direct inheritance, simulating multiple inheritances, and reusing code. Dart does not contain a final class directive like other languages; that is, a class can always be extended (have children). Dart supports the following types of Inheritance :

- Single (one child class is inherited by one parent class only).
- Multi level (child class can inherit from another child class).

Dart does not support Multiple Inheritance. The super keyword is used to refer to immediate parent of a class. The keyword can be used to refer to the super class version of a method or a variable**.** Dart provides extends keyword to inherit the properties of parent class in child class. The syntax is given below.

```
class child_class_name extends parent_class_name
```

4. Abstraction:

Following inheritance, abstraction is the process whereby we define a type and its essential characteristics, moving to specialized types from parent ones. Dart contains abstract classes that allow a definition of what something does/provides, without caring about how this is implemented. Dart has the powerful implicit interface concept, which also makes every class an interface, allowing it to be implemented by others without extending it.

5. Encapsulation:

Data Encapsulation is binding data and functions that use data into one unit. It is also referred to as data hiding and information hiding. Dart does not contain access restrictions explicitly, like the famous keywords used in Java—protected, private, and public. In Dart, encapsulation occurs at the library level instead of at the class level. Dart creates implicit getters and setters for all fields in a class, so you can define how data is accessible to consumers and the way it changes. In Dart, if an identifier (class, class member, top-level function, or variable) starts with an underscore(_), it's private to its library. Syntax is as given below:-

```
_identifier
```

```
library loggerlibrary;
void _log(message)
{
  print("Log method called in
  the loggerlibrary
  message:$message");
}
```

Example for your reference:

In above example we have defined a library with a private function.

6. Polymorphism:

Polymorphism is achieved through inheritance and it represents the ability of an object to copy the behavior of another object. It means that one object can have multiple forms.

subclasses or child classes usually override instance methods, getters and setters. For example, the int type is also a num type. Dart allows overriding parent methods to change their original behavior. Dart does not allow overloading in the way you may be familiar with. You cannot define the same method twice with different arguments. You can simulate overloading by using flexible argument definitions (that

is, optional and positional, as seen in the previous Functions section) or not use it at all. We can use @override to indicate that we are overriding a member. As Dart doesn't allow overloading to overcome this we can use argument definitions like optional and positional.

7. Abstract Class:

A class that contains one or more abstract method is called an abstract class. We can declare the abstract class using the abstract keyword followed by class declaration. The syntax is given below.

```
abstract class ClassName
{
  //Body of abstract class
}
```

7.1.4.2 Dart Packages:

Dart package is the collection of a well-organized, independent, and reusable code unit. Applications might be needed to implement third-party libraries or packages. The package generally contains a set of classes, functions, or unit of code for specific tasks along with the compiled program and sample data. Dart provides an extensive set of default packages that load automatically when the dart console starts. However, if we need packages other than the default packages then its need to installed and loaded explicitly in order to use. When a package is loaded, it can be used in the whole Dart environment.

Dart Package Manager:

Every language provides the functionality for handling the external packages such as Nuget for .NET, Gradle or Maven for Java, npm for Node.js, etc. Dart has the inbuilt package manager, which is called a pub. It is basically used to organize, manage the third-party libraries, tools, dependencies, and also used to install the packages in the repository.

Every Dart application consists of a pubspec.yaml file which includes the metadata of the file. The metadata of the package holds the author, version, application name, and description. The full form of the yaml is a Yet Another Markup Language. The pubspec.yaml is used to download the various libraries that application needs during programming. The pubspec.yaml file must look like as follows.

```
name: 'vector_victor'
version: 0.0.1
description: An absolute bare-bones web app.
...
dependencies: browser: '>=0.10.0 <0.11.0'
```

Sr. No	Description
pub get	It is used to get all packages of application dependent on.
pub upgrade	It is used to upgrade all application dependencies to the modern version.
pub build	It is used to construct your web application, and it will create a build folder, with all related scripts in it.
pub help	It is used to get help related to all pub commands or when we stuck while programming.

The Dart IDE's provides the inbuilt support for using the pub that consist of creating, downloading, updating, and publishing packages, Otherwise we can use the pub command line. The following is the list of the few important pub commands.

Installing a Package: The following steps are defining the installation of the package in the project.

- **Step 1:** Write the package name in the dependencies section of project's pubspec.yaml file. Then run the below command to find the package installed in project.

- The above command will download the package under the packages folder in the application directory.

We have added the **xml**: to the project dependencies. This will install Dart XML package in the project.

Importing libraries from packages:

Step 2: Once the package is installed in our project, we need to import the required libraries from the package in our program as follows –

7.1.5 SUMMARY

In this chapter, we have finished our introduction to the Dart language we presented the available tools to start your Dart language studies, discovered what a basic Dart program looks like, and learned about the basic Dart code syntax and structure. We also demonstrated how the Dart SDK works, and the tools it provides that help with Flutter app development and make the Flutter framework succeed in its objectives. We also reviewed some important concepts of the Dart language Additionally, we reviewed functions, try- catch mechanism and introduced Dart OOP concept, classes and packages.

7.1.6 QUESTIONS

1. Explain in brief about DART? Explain with a simple program?
2. What Are The Various Types Of Operators In Dart?
3. Explain recursion with an example in Dart?
4. Explain control flow statements in dart with syntax?
5. Explain the concepts of OOPS in Dart?
6. Explain Lamba Function?
7. Write a note on Classes in Dart and what Is The Syntax To Declare Class?
8. Explain Packages in Dart?

7.1.7 CHAPTER END EXERCISE

1. Create a program that asks the user to enter their name and their age. Print out a message that tells how many years they have to be 100 years old.
2. Ask the user for a number. Depending on whether the number is even or odd, print out an appropriate message to the user.
3. Make a two-player Rock-Paper-Scissors game against computer.(HINT:- Ask for player's input, compare them, print out a message to the winner.)
4. Generate a random number between 1 and 100. Ask the user to guess the number, then tell them whether they guessed too low, too high, or exactly right.(HINT :- Keep track of how many guesses the user has taken, and when the game ends, print this out.

7.1.8 REFERENCES

1. https://www.dartlang.org/articles/dart-vm/numeric-computation.
2. You can read more about generic syntax here: https://github.com/dart lang/sdk/blob/master/pkg/dev_compiler/doc/GENERIC_METHODS.md.
3. Alessandro Biessek Flutter for Beginners: An Introductory Guide to Building Cross-platform Mobile Applications with Flutter and Dart 2 Packt Publishing Ltd. ISBN. 9781788990523.

*****

7.2

INTRODUCTION TO DART AND FLUTTER

Unit Structure

7.2.0 OBJECTIVES

After this chapter:

- You will learn the history of the Flutter framework, how and why it was created, and its evolution so far.
- You will learn how its community is contributing to it, and how and why it has grown quickly in the last few years.
- You will be introduced to the main features of Flutter, with short comparisons to other frameworks.
- You will learn to make a basic Flutter project.

7.2.1 INTRODUCTION

Flutter – a simple and high performance framework based on Dart language, provides high performance by rendering the UI directly in the operating system's canvas rather than through native framework. Flutter also offers many ready to use widgets (UI) to create a modern application. These widgets are optimized for mobile environment and designing the application using widgets is as simple as designing HTML. Flutter application is itself a widget. Flutter widgets also supports animations and gestures. The application logic is based on reactive programming. Widget may optionally have a state. By changing the state of the widget, Flutter will automatically (reactive programming) compare the widget's state (old and new) and render the widget with only the necessary changes instead of re-rendering the whole widget.

Features of Flutter:

Flutter framework offers the following features to developers:

- Modern and reactive framework.
- Uses Dart programming language and it is very easy to learn.
- Fast development.
- Beautiful and fluid user interfaces.
- Huge widget catalog.
- Runs same UI for multiple platforms.
- High performance application.

Advantages of Flutter:

- Flutter comes with beautiful and customizable widgets for high performance and outstanding mobile application. It fulfills all the custom needs and requirements.
- Dart has a large repository of software packages which lets you to extend the capabilities of your application.

- Developers need to write just a single code base for both applications (both Android and iOS platforms). Flutter may to be extended to other platform as well in the future.
- Flutter needs lesser testing. Because of its single code base, it is sufficient if we write automated tests once for both the platforms.
- Flutter's simplicity makes it a good candidate for fast development. Its customization capability and extendibility makes it even more powerful.
- With Flutter, developers has full control over the widgets and its layout.
- Flutter offers great developer tools, with amazing hot reload.

Disadvantages of Flutter:

Despite its many advantages, flutter has the following drawbacks in it

- Since it is coded in Dart language, a developer needs to learn new language (though it is easy to learn).
- Modern framework tries to separate logic and UI as much as possible but, in Flutter, user interface and logic is intermixed. We can overcome this using smart coding and using high level module to separate user interface and logic.
- Flutter is yet another framework to create mobile application. Developers are having a hard time in choosing the right development tools in hugely populated segment.

7.2.2 COMPARISONS WITH OTHER MOBILE APP DEVELOPMENT FRAMEWORKS

Why Flutter and Not Some Other Framework:

Although it's relatively new, Flutter has experienced a great deal of experimentation and evolution over the years. It was called Sky, at its first appearance at the Dart Developer Summit 2015 presented by Eric Seidel. It was presented as the evolution of some previous Google experiments to create something better for mobile in terms of development and user experience, with the main goal of rendering with high performance. Presented as Flutter in 2016, and with its first alpha release in May 2017, it was already building for iOS and Android systems. Then it started to get mature and community adoption began to grow. It evolved from community feedback to its first stable release at the end of 2018. There are many mobile development frameworks out there that seek a common goal: to build native mobile apps for Android and iOS with a single code base. Some of those frameworks are widely adopted by the community and provide similar solutions to the problems they purport to solve.

7.2.2.1 The Problems Flutter Wants to Solve:

Since the beginning of the Flutter framework, it was intended to provide a better experience to the user through high-performance execution, but that's not the only promise of Flutter.

The development experience was also focused on addressing some of the problems of multiple platform mobile development:

1. **Long/more expensive development cycles:**

To be able to cope with market demand, you must choose to build for a single platform, or create multiple teams. This has some consequences in terms of cost, multiple deadlines, and different capabilities of native frameworks

2. **Multiple languages to learn:**

If a developer wants to develop for multiple platforms, they must learn how to do something in one OS and programming language, and later, the same thing on another OS and programming language. This certainly impacts in the developer's productive time.

3. **Long build/compile time**:

Some developers may already have experienced how build time may have an impact on productivity. In Android, for example, some developers experience multiple long build times after a few minutes of coding (this is evolving, and it's a lot better now, but it was already causing a lot of pain).

4. **Existing cross-platform solutions side effects**:

You adopt an existing cross platform framework (that is, React Native, Xamarin, Ionic, Cordova) in an attempt to work around the preceding problems, but with that come some side effects, such as a performance impact, a design impact, or a user experience impact.

Now let's see how Flutter counters these problems.

7.2.2.2 Differences between Existing Frameworks

There are a large number of high-quality and well-accepted frameworks and technologies.

Some of them are as follows:

- Xamarin
- React Native
- Ionic
- Cordova

Flutter has benefits that make space for itself, not necessarily by overcoming the other frameworks, but by already being at least on the same level as native frameworks:

- High performance
- Full control of the user interface
- Dart language
- Being backed by Google
- Open source framework
- Developer resources and tooling

High Performance:

Right now, it is hard to say that Flutter's performance is always better than all of the other frameworks in practice, but it's safe to say that its aim is to be. For example, its rendering layer was developed with a high frame rate in mind. Some of the existing frameworks rely on JavaScript and HTML rendering, which might cause overheads in performance because everything is drawn in a webview (a visual component like a web browser). Some use Original Equipment Manufacturer (OEM) widgets but rely on a bridge to request the OS API to render the components, which creates a bottleneck in the application because it needs an extra step to render the user interface (UI).

What Makes Flutter's Performance Great:

Flutter owns the pixels: Flutter renders the application pixel by pixel , interacting directly with the Skia graphics engine. No extra layers or additional OS API calls: As Flutter owns the app rendering, it he does not need additional calls to use the OEM widgets, so no bottleneck. Flutter is compiled to native code: Flutter uses the Dart AOT compiler to produce native code. That means there's no overhead in setting up an environment to interpret Dart code on the fly, and it runs just like a native app, starting more quickly than frameworks that need some kind of interpreter.

Full Control of the UI:

The Flutter framework chooses to do all the UI by itself rendering the visual components directly to the canvas, requiring nothing more than the canvas from the platform so it's not limited by rules and conventions. Most of the time, frameworks just reproduce what the platform offers in another way. For example, other webview-based cross-platform frameworks reproduce visual components using HTML elements with CSS styling. Other frameworks emulate the creation of the visual components and pass them to the device

platform, which will render the OEM widgets like a natively developed app .Refer the below points for more understanding:

- **Ruling all the pixels on the device:** Frameworks limited by OEM widgets will reproduce at most what a native developed app would, as they use only the platform's available components. On the other hand, frameworks based on web technologies may reproduce more than platform-specific components, but may also be limited by the mobile web engine available on the device. By getting the control of the UI rendering, Flutter allows the developer to create the UI in their own way by exposing an extensible and rich Widgets API, which provides tools that can be used to create a unique UI with no drawbacks in performance and no limits in design.

- **Platform UI kits:** By not using OEM widgets, Flutter can break the platform design, but it does not. Flutter is equipped with packages that provide platform design widgets, the Material set in Android, and Cupertino in iOS.

- **Achievable UI Design requirements:** Flutter provides a clean and robust API with the ability to reproduce layouts that are faithful to the design requirements. Unlike web-based frameworks that rely on CSS layout rules that can be large and complicated and even conflicting, Flutter simplifies this by adding semantic rules that can be used to make to complex but efficient and beautiful layouts.

- **Smoother look and feel:** In addition to native widget kits, Flutter seeks to provide a native platform experience where the application is running, so fonts, gestures, and interactions are implemented in a platform-specific way, bringing a natural feel the user, like a native application.

Dart:

Flutter's main goals is to be a high-performance alternative to existing cross-platform frameworks. But not only that; to significantly improve the mobile developer's experience was one of the crucial points of the project. Dart seems to be the perfect match to the framework for the following reasons:

- **Dart AOT and JIT compilation:** Dart is flexible enough to provide different ways of running the code, so Flutter uses Dart AOT with performance in mind when compiling a release version of the application, and it uses JIT with sub-second compilation of code in development time, aiming for fast workflows and code changes. **Dart Just in time compilation (JIT) and Ahead of time (AOT)** refers to when the compilation phase takes place. In AOT, code is compiled before running. In JIT, code is compiled while running. (Check out the Dart introduction section in the first chapter of this module).

- **High performance:** Due to Dart's support for AOT compilation, Flutter does not require a slow bridge between realms (for example, non-native to native), which makes Flutter apps also start up much more quickly. Also, Flutter uses a functional-style flow with short-lived objects, and this means a lot of short-lived allocations. Dart garbage collection works without locks, helping with fast allocation.

- **Easy learning:** Dart is a flexible, robust, modern, and advanced language. Although it's still evolving, the language has a well-defined object-oriented framework with familiar functionalities to dynamic and static languages, an active community, and well-structured documentation.

- **Declarative UI:** In Flutter, we use a declarative style to lay out widgets, what means that widgets are immutable from and are only lightweight "blueprints". To change the UI, a Widget triggers a rebuild on itself and its subtree. In the opposite imperative style (the most common), we can change specific component properties after they are created.

- **Dart syntax to layout:** Different from many frameworks that have a separate syntax for layout, in Flutter, the layout is created writing Dart code, aiming for greater flexibility and ease to create a developer environment, with tools for debugging layout rendering performance.

Being Backed By Google:

Flutter is a brand new framework, and this means that it does not have a big section of the mobile development market yet, but this is changing, and the outlook for the next few years is highly positive. Being backed by Google, the framework has all the tools it needs to succeed in the community, with support from the Google team, presence at big events such as Google IO, and investments into continuous improvement in the code base. From the launch of the third Beta version at Google IO 2018 to the first stable release launched during the Flutter Live Event at the end of 2018, its growth is evident:

- More than 200 million users of Flutter apps.
- More than 3,000 Flutter apps on the Play Store.
- More than 250,000 new developers.
- The 34th most popular software repository on GitHub—it was in the Top 15 at the beginning of 2019.

Fuchsia OS and Flutter:

It's not a secret anymore that Google is working on its new Fuchsia OS as a replacement for the Android OS. One thing to pay attention to

is that Fuchsia OS may be a universal Google system to run on more just than mobile phones, and this directly affects Flutter adoption. This is because Flutter will be the first method of developing mobile apps for the new Fuchsia OS, and, not only this, the UI of the system is being developed with it. With the system targeting more devices than just smartphones, as seems to be the case, Flutter will certainly have a lot of improvements. The growth of the framework's adoption is directly related to the new Fuchsia OS. Google has announced that Android apps will be compatible with the new OS, making the transition to and adoption of Flutter significantly easier.

Open Source Framework:

Having a big company such as Google behind it is fundamental to a framework such as Flutter (see React, for example, which is maintained by Facebook). In addition, community support becomes even more important as it becomes more popular. By being open source, the community and Google can work together to:

- Help with bug fixes and documentation through code collaboration
- Create new educational content about the framework
- Support documentation and usage
- Make improvement decisions based on real feedback

Improving the developer experience is one of the main goals of the framework. Therefore, in addition to being close to the community, the framework provides great tools and resources for developers.

Developer Resources and Tooling:

The focus on developers in the Flutter framework goes from documentation and learning resources to providing tools to helping with productivity:

- **Documentation and learning resources:** Flutter websites are rich for developers coming from other platforms, including many examples and use cases, for example, the famous Google Codelabs (https://codelabs.developers.google. com/?cat=Flutter).
- **Command-line tools and IDE integration:** Dart tools that help with analyzing, running, and managing dependencies are also part of Flutter. Besides that, Flutter also has commands to help with debugging, deploying, inspecting layout rendering and integration with IDEs through Dart plugins.
- **Easy start**: Flutter comes with the flutter doctor tool, which is a command-line tool that guides the developer through the system setup by indicating what isneeded in order to be ready to set up a Flutter environment.

- **Hot reload:** This is the feature that has been taking the focus in presentations about the framework. By combining the capabilities of the Dart language (such as JIT compilation) and the power of Flutter, it is possible for the developer to instantly see design changes made to code in the simulator or device. In Flutter, there is no specific tool for layout preview. Hot reload makes it unnecessary.

7.2.3 FLUTTER COMPILATION (DART)

The way an application is built is fundamental to how it will perform on the target platform. This is an important step regarding performance. Even though you do not necessarily need to know this for every kind of application, knowing how the application is built helps you to understand and measure possible improvements. Flutter relies on the AOT compilation of Dart for release mode and the JIT compilation of Dart for development/debug mode. Dart is one of very few languages that's capable of being compiled to both AOT and JIT, and for Flutter this is great.

Development Compilation:

During development, Flutter uses JIT compilation in development mode. This enables important development features such as hot reload. Due to the power of Dart's compiler, interactions between the code and the simulator/device are really fast, and debugging info helps developers step into the source code.

Release Compilation:

In release mode, debugging information is not necessary, and the focus is performance. Flutter uses a technique that's common to game engines. By using AOT mode, Dart code is compiled to native code, and the app loads the Flutter library and delegates rendering, input, and event handling to it by using the Skia engine.

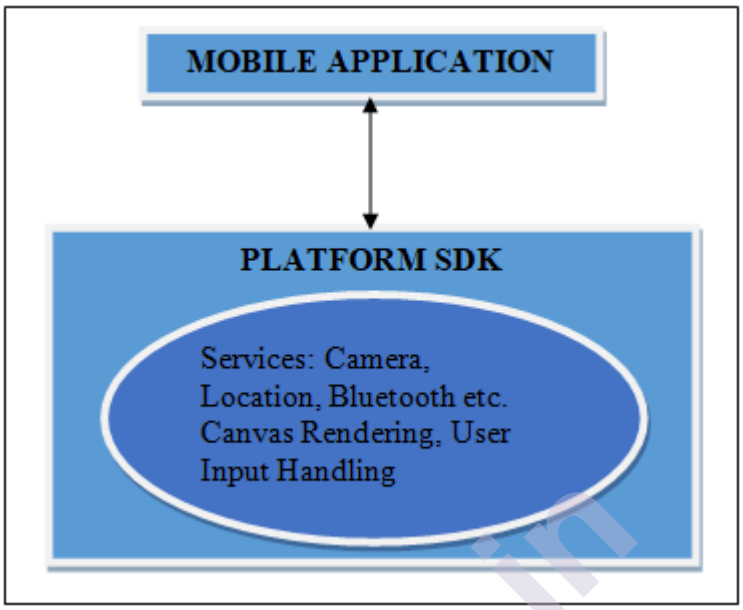
Supported Platforms:

By now, Flutter supports ARM Android devices running at least on Jelly Bean 4.1.x version, and iOS devices from iPhone 4S or newer. Of course, Flutter apps can normally be run on simulators. Google is intending to port the Flutter runtime to the web by using the Dart capability of compiling to JavaScript. Initially called Hummingbird, this project now is known as "Flutter for web".

7.2.4 FLUTTER RENDERING

Flutter is a UI framework that enables the rendering of a set of code on multiple terminals. It adopts a simple rendering pipeline design and provides a rendering engine. Therefore, it provides a better performance and user experience than React Native, Weex, WebView, and other solutions. One of the main aspects that makes Flutter unique

is the way that it draws the visual components to the screen. The big difference is how the application communicates with the platform's SDK, what it asks the SDK to do, and what it does by itself:



The platform SDK can be seen as the interface between applications and the operation system and services. Each system provides its own SDK with its own capabilities and is based on a programming language (that is, Kotlin/Java for the Android SDK and Swift/Objective C for the iOS SDK).

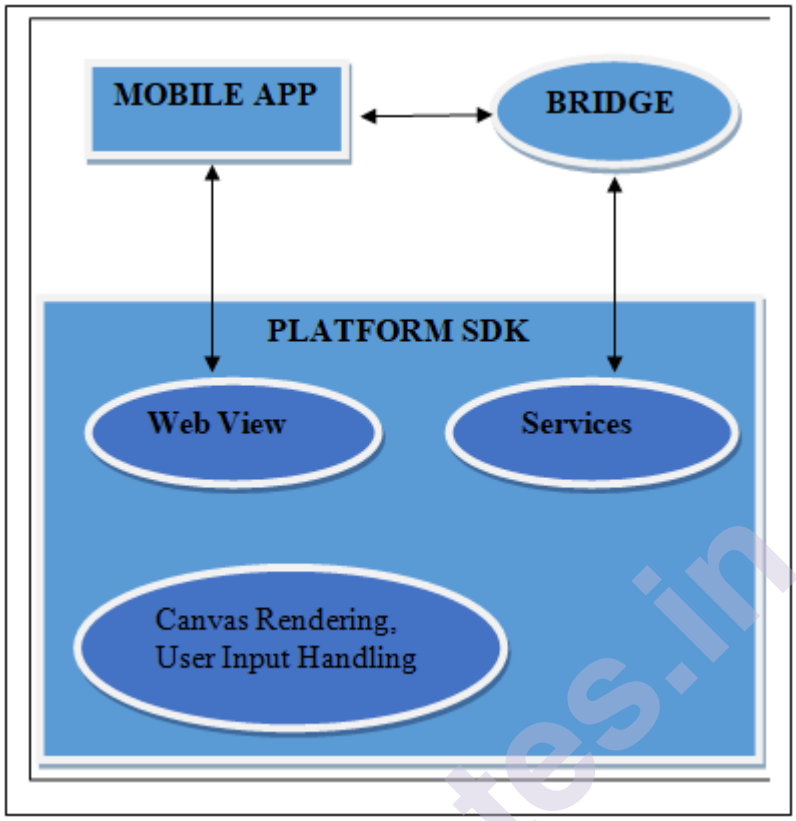
7.2.4.1 Web-Based Technologies:

Flutter is Google's UI library that was initially launched to develop native, performant, and beautiful mobile applications. However, the vision behind Flutter isn't restricted to just mobile apps, it's to build user interfaces for any and every screen with cross platform development. The web itself is a flexible platform, but Flutter is ideal for building web applications like PWAs or SPAs and bringing your existing mobile app to the web. At the beginning of 2019, Flutter for Web was originally in a separate repo of its own. This repo merged into the official flutter branch on Sept 10, 2019, with the launch of Flutter v1.9. The Flutter developer community is very active and is making rapid progress to boost performance and bring the Flutter Web to a stable release.

Flutter web performance is measured by primarily two things. The first is the ability to render and manipulate large amounts of data and the second is transitions, effects, and animations.

Dart is able to handle long lists flawlessly and I was able to scroll through 100k records in a breeze. In general, Flutter for Web satisfies the former requirement but pales in comparison to modern JS frameworks in the latter. We have already seen frameworks that use

webviews to reproduce a UI by combining HTML and CSS. In terms of platform usage, it would look like this:

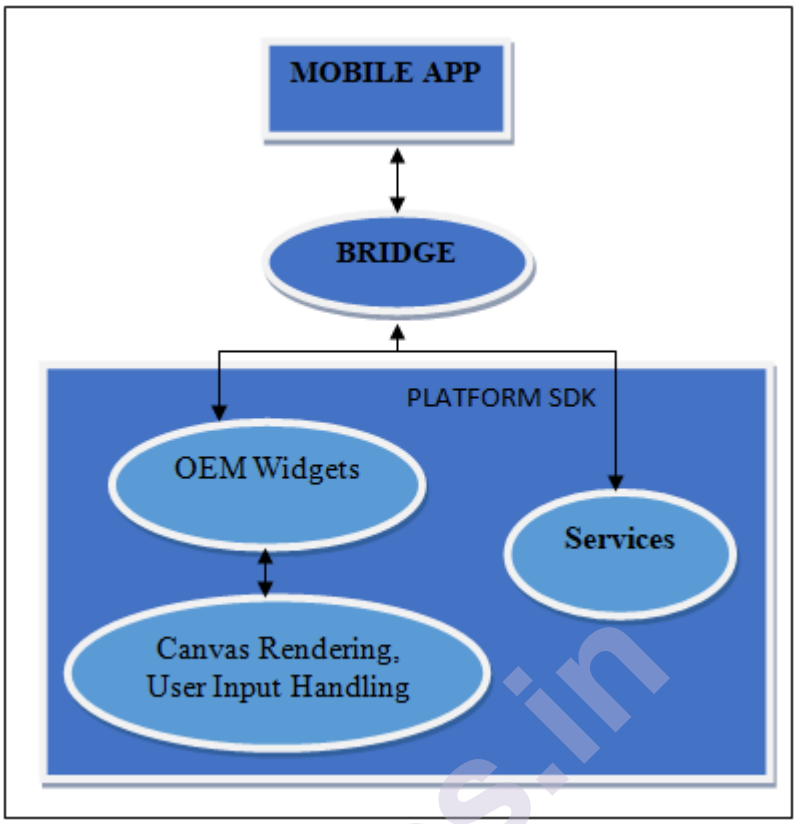


The application does not know how the rendering is done by the platform; the only thing it needs is the webview widget on which it will render the HTML and CSS code.

7.2.4.2 Framework and Oem Widgets:

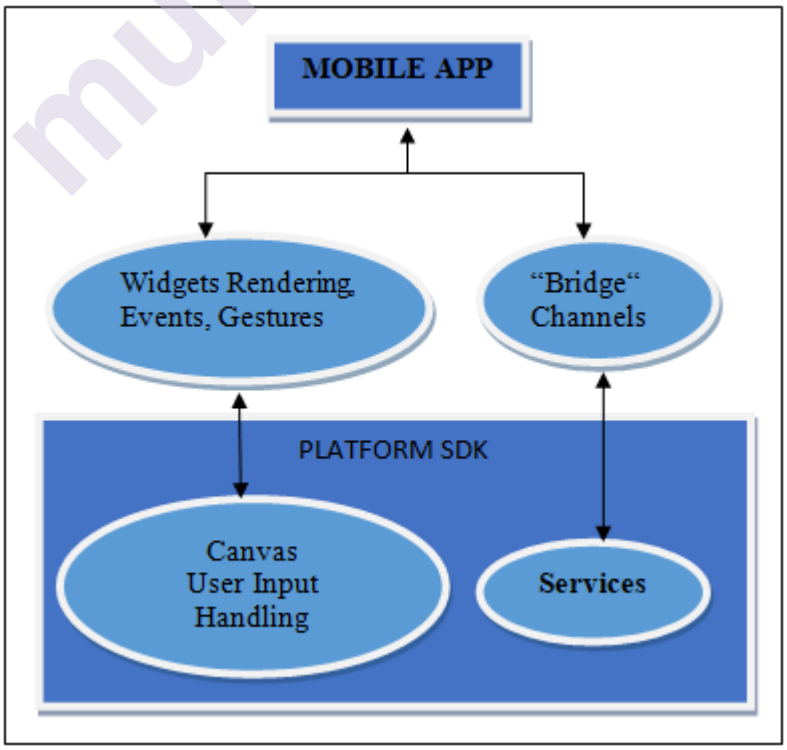
Another way of rendering widgets is by adding a layer above the platform widgets, but not

changing the way the system renders visual components effectively. In this mode of rendering, the work is done by the SDK like a normal native app, but before it, the layout is defined by an additional step in the framework language. Every change in the UI causes communication between the application code and the native code that's responsible for calling the platform's SDK, working like an intermediary. Like the previous technique, it may cause a small overhead for the application, maybe a little bit bigger than the previous one, because rendering occurs often, and therefore so does the communication.



7.2.1.3 Flutter – Rendering By Itself:

Flutter chooses to do all the hard work by itself. The only thing it needs from the platform's SDK is access to Services APIs and a canvas to draw the UI on:



Flutter moves the widgets and rendering to the app, from where it gets the customization and extensibility. Through a canvas, it can draw anything and also access events to handle user inputs and gestures by itself. The bridge in Flutter is done by platform.

7.2.5 FLUTTER ARCHITECTURE

The architecture of the Flutter framework. The Flutter architecture mainly comprises of four components.

1. Flutter Engine
2. Foundation Library
3. Widgets
4. Design Specific Widgets

Flutter Engine:

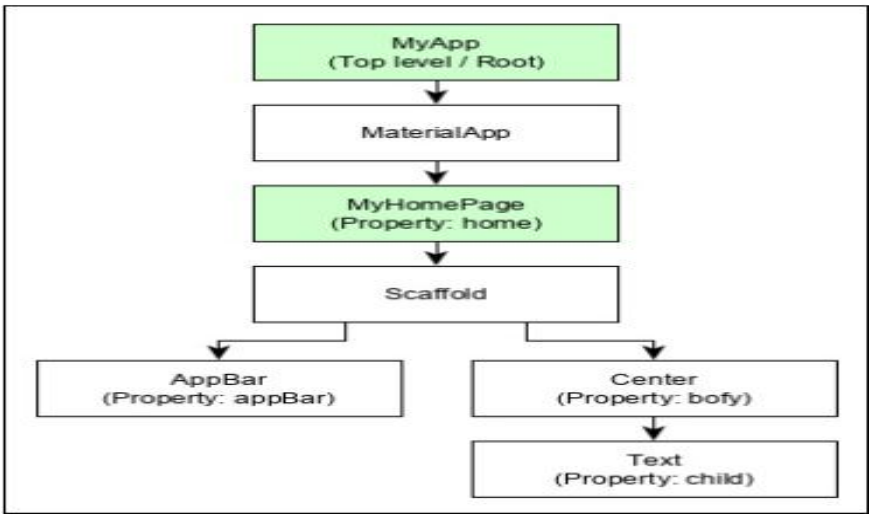
It is a portable runtime for high-quality mobile apps and primarily based on the C++ language. It implements Flutter core libraries that include animation and graphics, file and network I/O, plugin architecture, accessibility support, and a dart runtime for developing, compiling, and running Flutter applications. It takes Google's open-source graphics library, Skia, to render low-level graphics.

Foundation Library:

It contains all the required packages for the basic building blocks of writing a Flutter application. These libraries are written in Dart language.

Widgets:

In Flutter, everything is a widget, which is the core concept of this framework. Widget in the Flutter is basically a user interface component that affects and controls the view and interface of the app. It represents an immutable description of part of the user interface and includes graphics, text, shapes, and animations that are created using widgets. The widgets are similar to the React components. In Flutter, the application is itself a widget that contains many sub widgets. It means the app is the top-level widget, and its UI is build using one or more children widgets, which again includes sub child widgets. This feature helps you to create a complex user interface very easily. In the below given flutter architecture, we can see that all the components are widgets that contain child widgets. Thus, the Flutter application is itself a widget.



Design Specific Widgets:

The Flutter framework has two sets of widgets that conform to specific design languages. These are Material Design for Android application and Cupertino Style for IOS application.

Gestures:

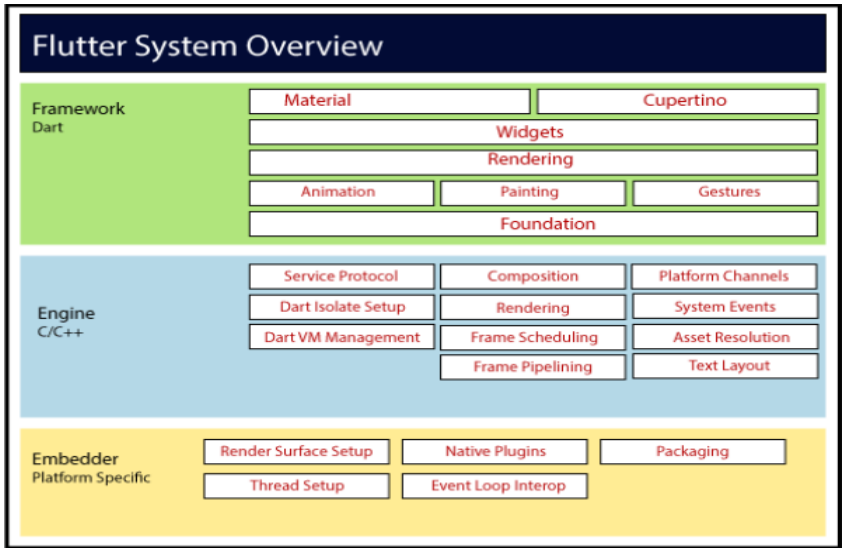
It is a widget that provides interaction (how to listen for and respond to) in Flutter using GestureDetector. GestureDector is an invisible widget, which includes tapping, dragging, and scaling interaction of its child widget. We can also use other interactive features into the existing widgets by composing with the GestureDetector widget.

State Management:

Flutter widget maintains its state by using a special widget, StatefulWidget. It is always auto re-rendered whenever its internal state is changed. The re-rendering is optimized by calculating the distance between old and new widget UI and render only necessary things that are changes.

Layers:

Layers are an important concept of the Flutter framework, which are grouped into multiple categories in terms of complexity and arranged in the top-down approach. The topmost layer is the UI of the application, which is specific to the Android and iOS platforms. The second topmost layer contains all the Flutter native widgets. The next layer is the rendering layer, which renders everything in the Flutter app. Then, the layers go down to Gestures, foundation library, engine, and finally, core platform-specific code. The following diagram specifies the layers in Flutter app development.



Source:https://www.theregister.com/2020/09/23/flutter_targets_the_windows_desktop/

7.2.6 WIDGETS INTRODUCTION

Understanding Flutter widgets is essential if you want to work with it. Flutter takes control of rendering and does this with extensibility and customization in mind, intending to add power to the developer's hands. Widgets can be understood as the visual (but not only that) representation of parts of the application. Many widgets are put together to compose the UI of an application. Imagine it as a puzzle in which you define the pieces. The intention of widgets is to provide a way for your application be modular, scalable, and expressive with less code and without imposing limitations. The main characteristics of the widgets UI in Flutter are composability and immutability.

Composability:

Flutter chooses composition over inheritance, with the goal of keeping each widget simple and with a well-defined purpose. Meeting one of the framework's goals, flexibility, Flutter allows the developer to make many combinations to achieve incredible results.

Immutability:

Flutter is based on the reactive style of programming, where the widget instances are short lived and change their descriptions (whether visually or not) based on configuration changes, so it reacts to changes and propagates these changes to its composing widgets, and so on. A Flutter widget may have a state associated with it, and when the associated state changes, it can be rebuilt to match the representation. The terms state and reactive are well known in the React style of programming, disseminated by Facebook's famous React library

Everything Is A Widget:

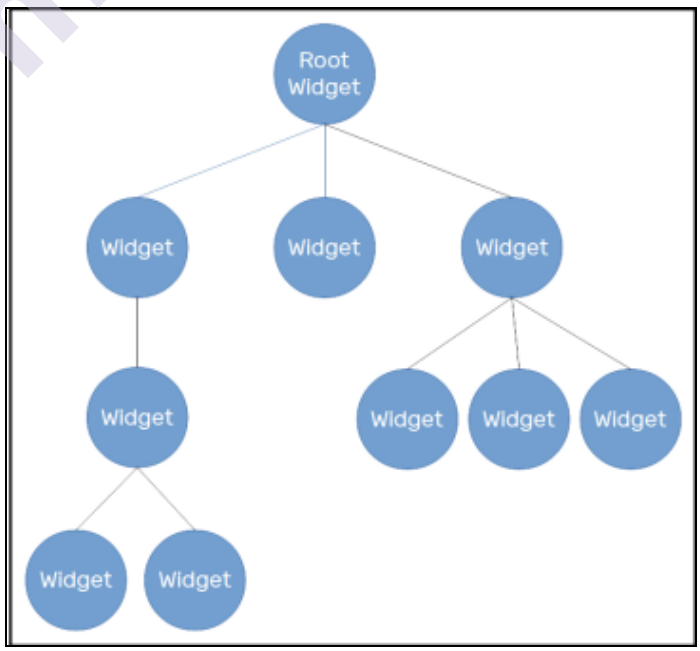
Flutter widgets are everywhere in an application. Maybe not everything is a widget, but almost everything is. Even the app is a widget in Flutter, and that's why this concept is so important. A widget represents a part of a UI, but it does not mean it's only something that is visible. It can be any of the following:

- A visual/structural element that is a basic structural element, such as the Button or Text widgets
- A layout specific element that may define the position, margins, or padding, such as the Padding widget
- A style element that may help to colorize and theme a visual/structural element, such as the Theme widget
- An interaction element that helps to respond to interactions in different ways, such as the GestureDetector widget.

Widgets are the basic building blocks of an interface. To build a UI properly, Flutter organizes the widgets in a widget tree.

7.2.6.1 The Widget Tree:

- This is another important concept in Flutter layouts. It's where widgets come to life. The widget tree is the logical representation of all the UIs widgets. It is computed during layout (measurements and structural info) and used during rendering (frame to screen) and hit testing (touch interactions), and this is the things Flutter does best. By using a lot of optimization algorithms, it tries to manipulate the tree as little as possible, reducing the total amount of work spent on rendering, aiming for greater efficiency:



- Widgets are represented in the tree as nodes. It may have a state associated with it; every change to its state results in rebuilding the widget and the child involved. As you can see above, the tree's child structure is not static, and it's defined by the widgets description. The children relations in widgets are what makes the UI tree; it exists by composition, so it's common to see Flutter's built-in widgets exposing child or children properties, depending on the purpose of the widget.

- The widget tree does not work alone in the framework. It has the help of the element tree; a tree that relates to the widget tree by representing the built widget on the screen, so every widget will have a corresponding element in the element tree after it is built. The element tree has an important task in Flutter.

- It helps to map on-screen elements to the widget tree. Also, it determines how widget rebuilding is done in update scenarios. When a widget changes and needs to be rebuilt, this will cause an update on the corresponding element. The element stores the type of the corresponding widget and a reference to its children elements. In the case of repositioning, for example, a widget, the element will check the type of the corresponding new widget, and in a match it will update itself with the new widget description.

7.2.7 HELLO FLUTTER

Let us create a simple Flutter application to understand the basics of creating a flutter application in the Android Studio.

7.2.7.1 Installation in Windows:

In this section, let us see how to install Flutter SDK and its requirement in a windows system.

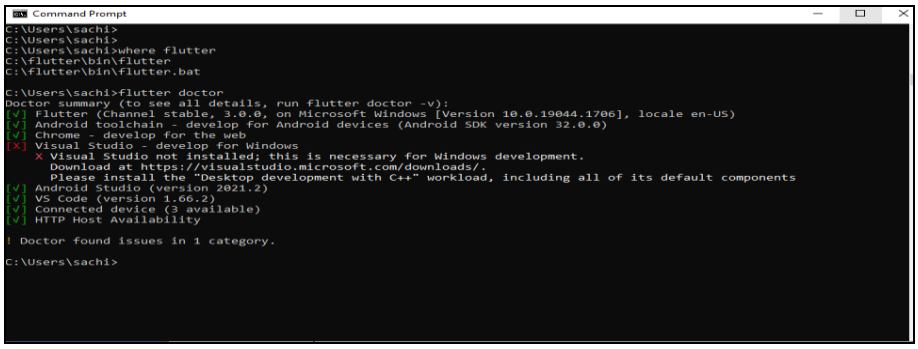
- **Step 1:** Go to URL, https://flutter.dev/docs/get-started/install/windows and download the latest Flutter SDK. As of April 2019, the version is 1.2.1 and the file is flutter_windows_v1.2.1-stable.zip.

- **Step 2:** Unzip the zip archive in a folder, say C:\flutter\

- **Step 3:** Update the system path to include flutter bin directory.

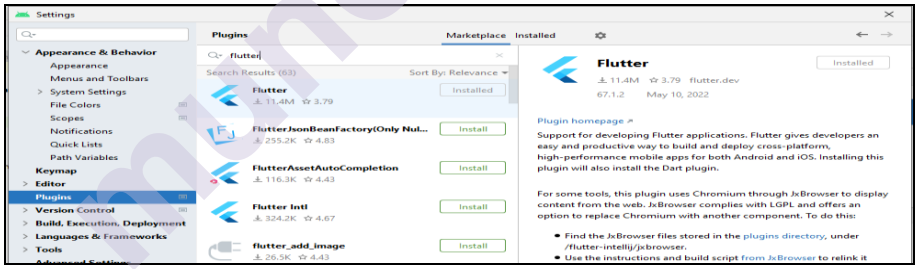
- **Step 4:** Flutter provides a tool, flutter doctor to check that all the requirement of flutter development is met.

- **Step 5:** Running the above command will analyze the system and show its report as shown below –



- **Step 6:** Install the latest Android SDK, if reported by flutter doctor
- **Step 7:** Install the latest Android Studio, if reported by flutter doctor
- **Step 8:** Start an android emulator or connect a real android device to the system.
- **Step 9:** Install Flutter and Dart plugin for Android Studio. It provides startup template to create new Flutter application, an option to run and debug Flutter application in the Android studio itself, etc.,
- **Step 10:** Open Android Studio. :- Click File → Settings → Plugins.

(Select the Flutter plugin and click Install.)



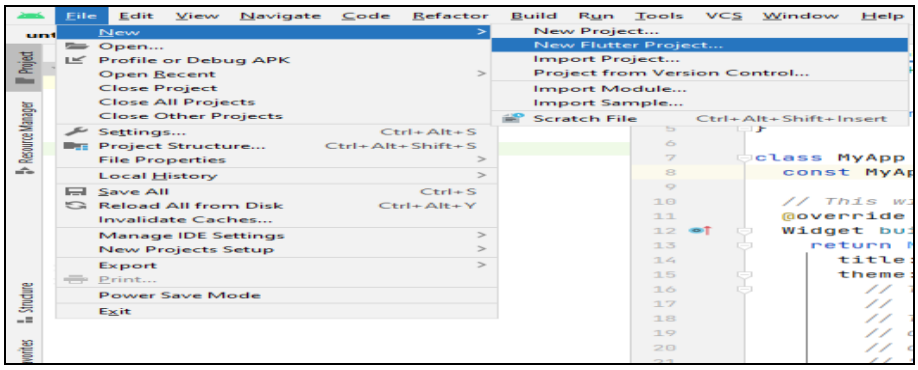
Click Yes when prompted to install the Dart plugin. Restart Android studio.

7.2.7.2 Creating First Flutter Application:

To create simple Flutter application in the Android Studio.

Step 1: Open Android Studio.

Step 2: Create Flutter Project. For this, click File → New → New Flutter Project.

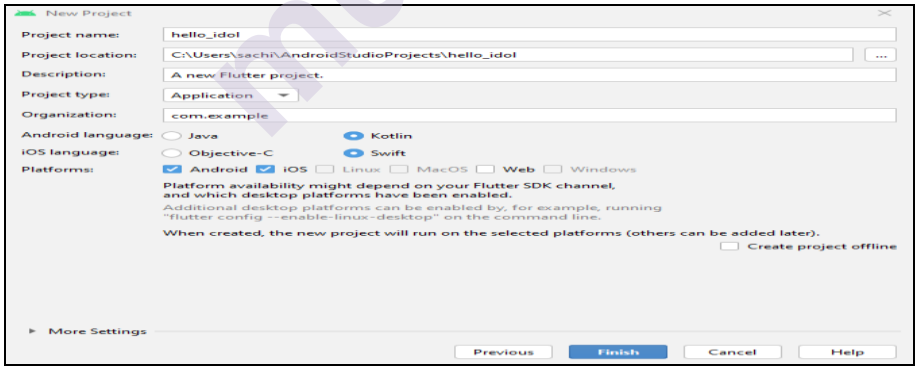


Step 3: Select SDK PATH(configure Flutter SDK Path: <path_to_flutter_sdk>)

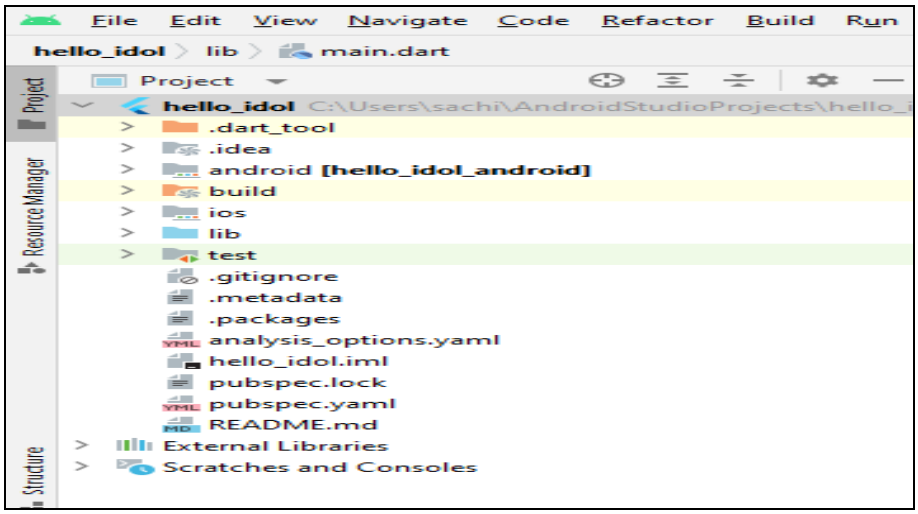
Step 4: Configuring Flutter Application

1. **Project name**:

hello_idol; Project Location: <path_to_project_folder> and **Description**: A new Flutter project



Clicking on the Finish button will create a fully working flutter application with minimal functionality. Below is image provided to show you the directory structure of the flutter application created.



In the image you can see the various folders and components of a flutter application structure. Lets understand structure and various components of the application.

- **.idea:** This is the folder at the very top of the project structure, it holds the Android studio configuration.
- **.android:** This folder holds auto generated source code for complete android project while creating flutter application for android. When the Flutter project is compiled into the native code, this will be injected into this Android project to generate native Android application.
- **.ios:** This folder holds auto generated source code for complete iOS project while creating flutter application for iOS. When the Flutter project is compiled into the native code, this will be injected into this iOS project to generate native iOS application.
- **.lib:** The lib stands for the library. It is the main working directory where we will do our project work. This folder mainly holds all of our Dart files written using flutter framework while creating our flutter application. By default, it contains a main.dart file for our flutter application.
- **lib/main.dart:** By default lib folder contains a main.dart file which works as entry point for our flutter application.
- .test : This folder contains Dart code to perform automated testing of our flutter application.
- **test/widget_test.dart:** This file contains sample code.
- **.gitignore:** A git file containing a list of files and folders that should be ignored by Git version control.

- **.metadata:** It is an auto generated file by the flutter tools, typically holds metadata or properties of our flutter application. You do not need to edit this file manually at any time.
- **.packages:** It is an auto generated file contain a list of packages for tracking dependencies of our Flutter project.
- **hello_app.iml**: It is always named as your Flutter project's name. It is a project file contains additional settings of the project used by Android studio.
- **pubspec.yaml:** It is a project configuration file that is used by Pub, Flutter package manager.
- **pubspec.lock :** It is an auto generated file by Pub, Flutter package manager based on the .yaml file.
- **README.md:** It is an auto generated file that holds project description.

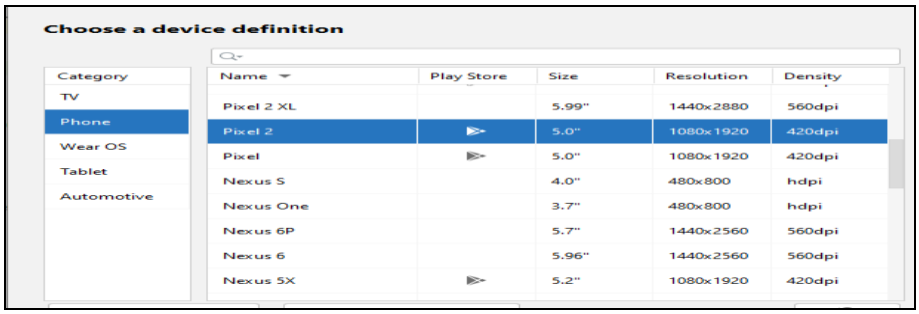
Step 5: Now open the lib/main.dart file and replace the code with the following code.

README.md | main.dart

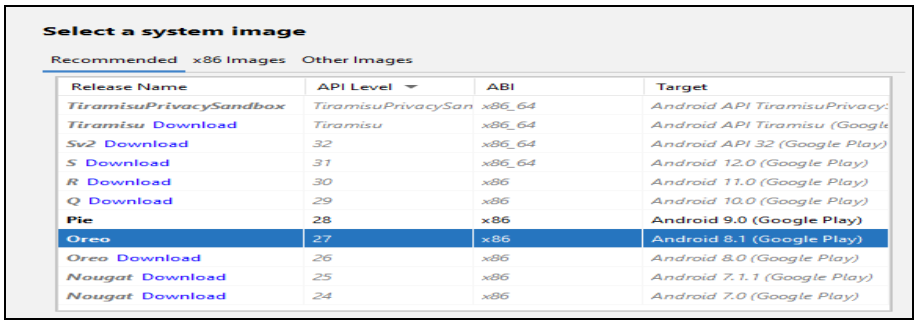
```dart
import 'package:flutter/material.dart';

void main() {
  runApp(
  MaterialApp(
    title : "IDOL FLUTTER FIRST PRACTICAL",
    home: Scaffold(
      appBar: AppBar(title: Text("IDOL SAMPLE PRACTICAL"),),
      body: Material(
        color: Colors.white,
        child: Center(
          child:Text("HELLO IDOL STUDENT")

    )  // Center

      )  // Material

    )  // Scaffold
    )  // MaterialApp
  );
}

```

Step 6: BEFORE RUNNING THE ABOVE CODE SETUP AVD(Android Virtual Device Manager)

1. Now, go to drop-menu Tools and chose AVD Manager
2. Create Virtual Device:- You will get a pop-up window like this one and there go for "+ Create Virtual Device.." in the left down corner.
3. Choose category and virtual device:- Again you will get a new one pop-up window and now is to you to choose device definition. You have category: (TV, Phone, WearOS, Tablet , Automotive). You probably need to choose the Phone category, but that depends on you and your project. Go for the Phone category and choose one phone that you want.

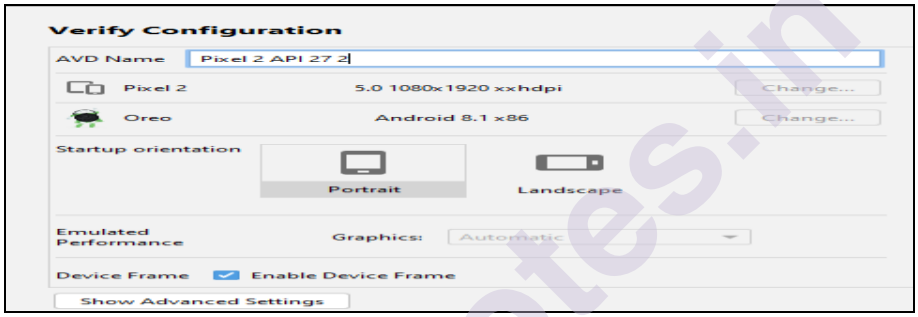


System Images: Now we need to choose System Image for our device. Android System Images are just versions of Android that you can run on your computer. (I HAVE **SELECTED OREO**).



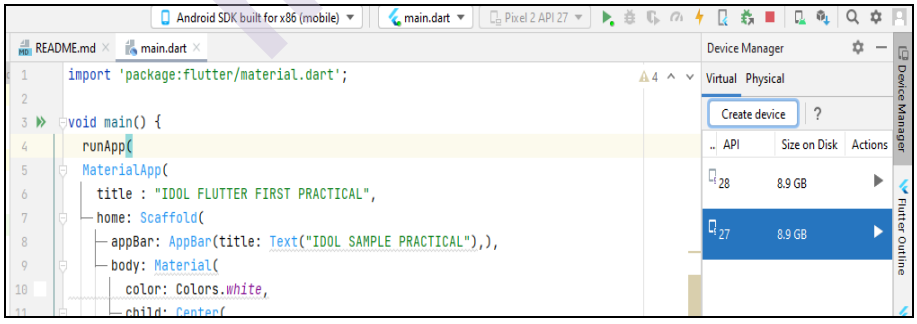
4. Android Virtual Device:

Then, finish step you can change the name of your device. Click Finish.



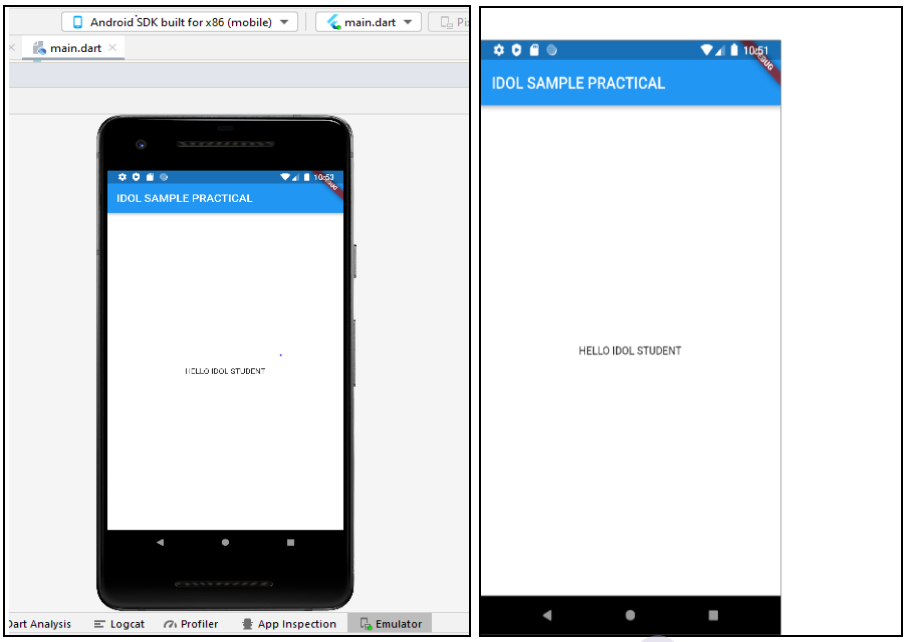
5. Launch Android Virtual Device:

Now you can launch this AVD in the emulator. Click that. And it will show you a phone, maybe you need to wait for a few seconds but be patient. After you launch this AVD in the emulator, go to drop-menu and choose (mobile) and also choose main.dart.



6. Last step: Run 'main.dart' (Shift+F10):

Run it. It can take a few seconds/minutes. Be patient. After that this main.dart code will be run on your virtual device.



7.2.8 FLUTTER USER INTERFACE USING WIDGETS

Widgets are immutable. That is, they cannot be changed. Any properties that they contain are final and can only be set when the widget is initialized. This keeps them lightweight so that it's inexpensive to recreate them when the widget tree changes. There are broadly two types of widgets in the flutter:

1 Stateless Widget (Stateless widgets are widgets that don't store any state. That is, they don't store values that might change.)

2 Stateful Widget (The second type of widget is called a stateful widget. That means it can keep track of changes and update the UI based on those changes.)

State:

The state of an app can very simply be defined as anything that exists in the memory of the app while the app is running. This includes all the widgets that maintain the UI of the app including the buttons, text fonts, icons, animations, etc The State is the information that can be read synchronously when the widget is built and might change during the lifetime of the widget. In other words, the state of the widget is the data of the objects that its properties (parameters) are sustaining at the time of its creation (when the widget is painted on the screen). The state can also change when it is used for example when a CheckBox widget is clicked a check appears on the box. What are these stateful and stateless widgets and how do they differ from one another.

7.2.8.1 Stateful Versus Stateless Widgets:

Widgets play an important role in Flutter application development. They are the pieces that form the UI; they are the code representation of what is visible to the user. UIs are almost never static; they change frequently, as you know. Although immutable by definition, widgets are not meant to be final – after all, we are dealing with a UI, and a UI will certainly change during the life cycle of any application. That's why Flutter provides us with two types of widgets: stateless and stateful. The big difference between these is in the way the widget is built. It's the developer's responsibility to choose which kind of widget to use in each situation to compose the UI in order to make the most of the power in the widget rendering layer of Flutter.

Stateless Widgets:

A typical UI will be composed of many widgets, and some of them will never change their properties after being instantiated. They do not have a state; that is, they do not change by themselves through some internal action or behavior. Instead, they are changed by external events on parent widgets in the widgets tree. So, it's safe to say that stateless widgets give control of how they are built to some parent widget in the tree. The following is a representation of a stateless widget:

So, the child widget will receive its description from the parent widget and will not change it by itself. In terms of code, this means that stateless widgets have only final properties defined during construction, and that's the only thing that needs to be built on the device screen. Below is the basic structure of a stateless widget. Stateless widget overrides the build() method and returns a widget. For example, we use Text or the Icon is our flutter application where the state of the widget does not change in the runtime. It is used when the UI depends on the information within the object itself.

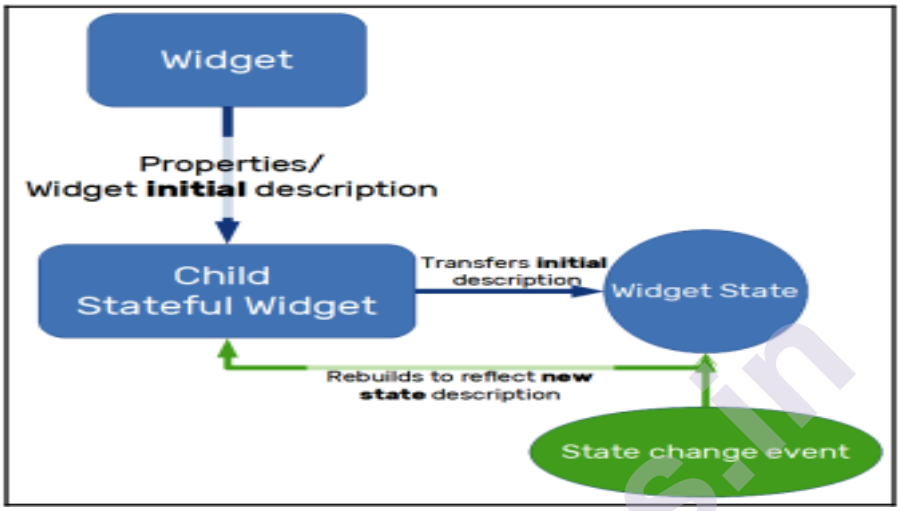
README.md | main.dart | widget_test.dart

```dart
import 'package:flutter/material.dart';


void main() => runApp(const IDOLMCA());

class IDOLMCA extends StatelessWidget {
  const IDOLMCA({Key? key}) : super(key: key);

  @override Widget build(BuildContext context)
  {return MaterialApp(
    home: Scaffold(
      backgroundColor: Colors.grey,
      appBar: AppBar(backgroundColor: Colors.green,
        title: const Text("IDOL MCA"), ),  // AppBar
      body: const Center(child: Text("Stateless Widget"),
      ),  // Center
    ),  // Scaffold
  );  // MaterialApp
  }}

```

So let us see what this small code snippet tells us. The name of the stateless widget is IDOLMCA which is being called from the runApp() and extends a stateless widget. Inside this IDOLMCA a build function is overridden and takes BuildContext as a parameter. This BuildContext is unique to each and every widget as it is used to locate the widget inside the widget tree.

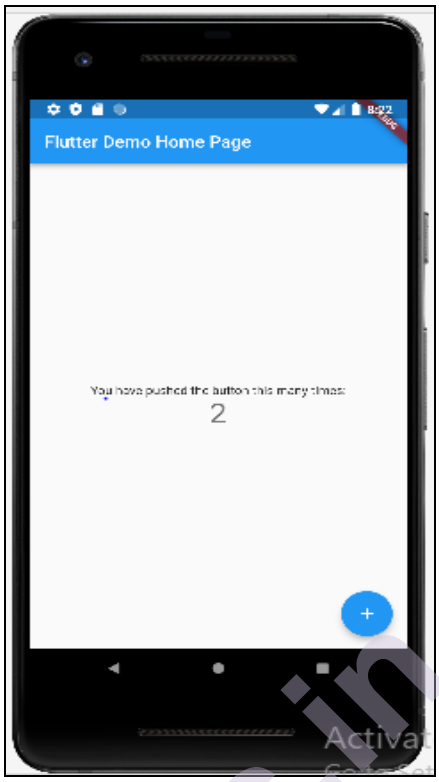
Stateful Widgets:

Unlike stateless widgets, which receive a description from their parents that persist during the widgets' lifetime, stateful widgets are meant to change their descriptions dynamically during their lifetimes. By definition, stateful widgets are also immutable, but they have a company State class that represents the current state of the widget. It is shown in the following diagram:



By holding the state of the widget in a separate State object, the framework may rebuild it whenever needed without losing its current associated state. The element in the elements tree holds a reference of the corresponding widget and also the State object associated with it. The State object will notify when the widget needs to be rebuilt and then cause an update in the elements tree, too.

```dart
import 'package:flutter/material.dart';


void main() => runApp(MyApp());

class MyApp extends StatefulWidget {
  @override
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```



Differences Between Stateless And Stateful Widget:

Stateless Widget:

- Stateless Widgets are static widgets.
- They do not depend on any data change or any behavior change.
- Stateless Widgets do not have a state, they will be rendered once and will not update themselves, but will only be updated when external data changes.
- For Example: Text, Icon, RaisedButton are Stateless Widgets.

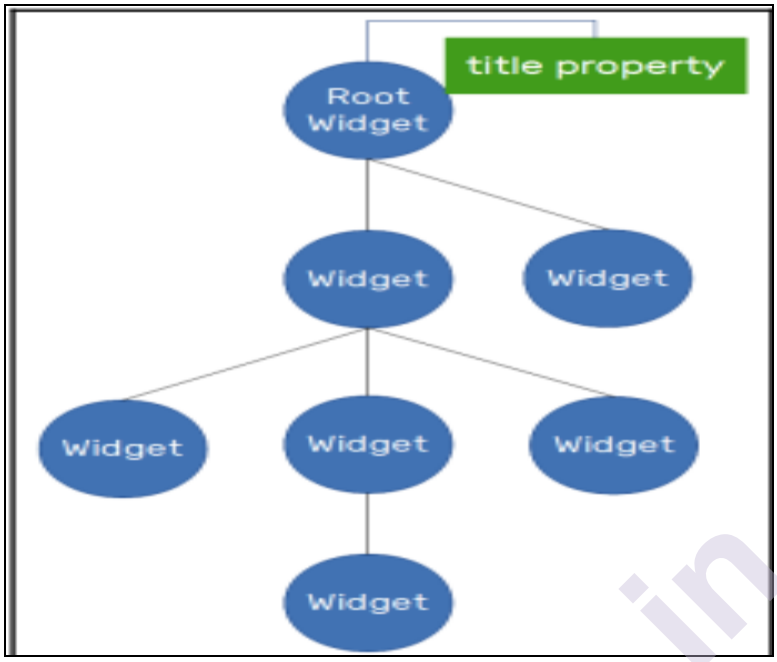
Stateful Widget:

- Stateful Widgets are dynamic widgets.
- They can be updated during runtime based on user action or data change.
- Stateful Widgets have an internal state and can re-render if the input data changes or if Widget's state changes.
- For Example: Checkbox, Radio Button, Slider are Stateful Widgets.

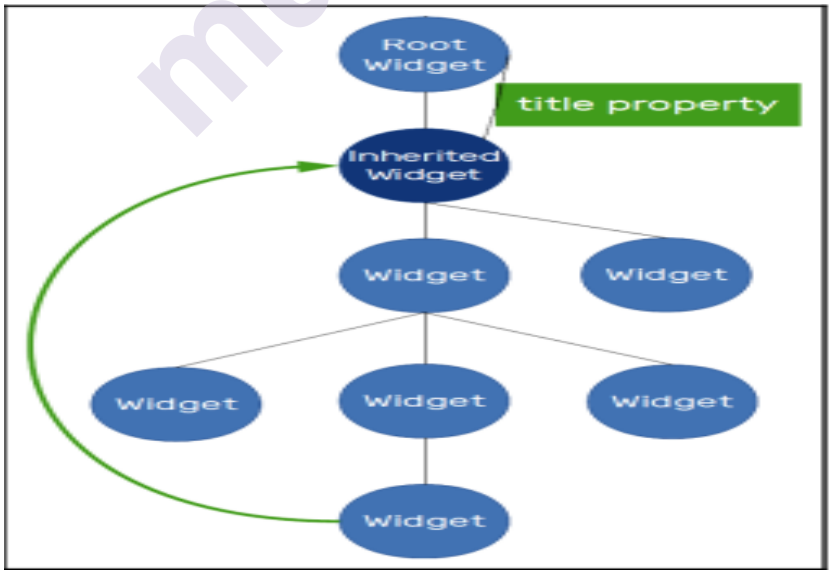
7.2.8.2 Inherited Widgets:

Besides statelessWidget and statefulWidget, there is one more type of widget in the Flutter framework, InheritedWidget. Sometimes, one widget may need to have access to data up the tree, and in such a case,

we would need to replicate the information down to the interested widget. This process is shown in the following diagram:



Let's suppose some of the widgets down the tree need to access the title property from the root widget. To do that, with statelessWidget or statefulWidget, we would need to replicate the property in the corresponding widgets and pass it down through the constructor. It can be annoying to replicate the property on all child widgets so that the value reaches the interested widget. To address this problem, Flutter provides the InheritedWidget class, an auxiliary kind of widget that helps to propagate information down the tree as shown in the following diagram:



By adding a InheritedWidget to the tree, any widget below it can access the data it exposes by using the inheritFromWidgetOfExactType(InheritedWidget) method of

BuildContext class that receives an InheritedWidget type as parameter and uses the tree to find the first ancestral widget of the requested type.

7.2.8.3 Widget Key Property:

If you take a look at both constructors of statelessWidget and statefulWidget classes, you will notice a parameter named key. This is an important property for widgets in Flutter. It helps in the rendering from the widgets tree to the element tree. Besides the type and a reference to the corresponding widget, this element also holds the key that identifies the widget in the tree. The key property helps to **preserve the state of a widget between rebuilds**. The most common usage of key is when we are dealing with collections of widgets that have the same type; so, without keys, the element tree would not know which state corresponds to which widget, as they would all have the same type. For example, whenever a widget changes its position or level in the widgets tree, matching is done in the elements tree to see what needs to be updated in the screen to reflect the new widget structure. When a widget has a state, it needs the corresponding state to be moved around with it. In brief, that is what a key helps the framework to do. By holding the key value, the element in question will know the corresponding widget state that needs to be with it.

7.2.8.4 Built-In Widgets:

Flutter has a big focus on UI, and because of this, it contains a large catalog of widgets to allow the construction of customized interfaces according to your needs. The available widgets of Flutter go from simple ones, such as the Text widget in the Flutter counter application example, to complex widgets that help to design dynamic UI with animations and multiple gesture handling.

1. **Accessibility:** These are the set of widgets that make a flutter app more easily accessible.
2. **Animation and Motion:** These widgets add animation to other widgets.
3. **Assets, Images, and Icons:** These widgets take charge of assets such as display images and show icons.
4. **Async:** These provide async functionality in the flutter application.
5. **Basics:** These are the bundle of widgets which are absolutely necessary for the development of any flutter application.
6. **Cupertino:** These are the ios designed widgets.
7. **Input:** This set of widgets provide input functionality in a flutter application.

8. **Interaction Models:** These widgets are here to manage touch events and route users to different views in the application.
9. **Layout**: This bundle of widgets helps in placing the other widgets on the screen as needed.
10. **Material Components:** This is a set of widgets that mainly follow material design by Google.
11. **Painting and effects:** This is the set of widgets which apply visual changes to their child widgets without changing their layout or shape.
12. **Scrolling:** This provides sacrollability of to a set of other widgets that are not scrollable by default.
13. **Styling:** This deals with the theme, responsiveness, and sizing of the app.
14. Text: This display text.

Lets study some widgets in details

- **The Text widget:** Text displays a string of text allowing styling:

```
Text(
 "This is a text",
)
```

The most common properties of the Text widget are as follows:

1. **style:** A class that composes the styling of a text. It exposes properties that allow changing the text color, background, font family (allowing the usage of a custom font from assets, line height, font size, and so on.
2. **textAlign:** Controls the text horizontal alignment, giving options such as center aligned or justified, for example.
3. **maxLines:** Allows specifying a maximum number of lines for the text that will be truncated if the limit is exceeded.
4. **overflow:** Will define how the text will be truncated on overflows, giving options such as specifying a max-lines limit. It can be by adding an ellipsis at the end, for example

The Images:

When you create an app in Flutter, it includes both code and assets (resources). An asset is a file, which is bundled and deployed with the app and is accessible at runtime. The asset can include static data, configuration files, icons, and images. The Flutter supports many image formats, such as JPEG, WebP, PNG, GIF, animated WebP/GIF,

BMP, and WBMP. Displaying images is the fundamental concept of most of the mobile apps. Flutter has an Image widget that allows displaying different types of images in the mobile application.

How to display the image in Flutter:- To display an image in Flutter, do the following steps:

Step 1: First, we need to create a new **folder** inside the root of the Flutter project and named it assets. We can also give it any other name if you want.

Step 2: Next, inside this folder, add one image manually.

Step 3: Update the pubspec.yaml file. Suppose the image name is tablet.png, then pubspec.yaml file is:

assets:

- assets/tablet.png

- assets/background.png

If the assets folder contains more than one image, we can include it by specifying the directory name with the slash (/) character at the end.

flutter:

assets:

- assets/

Step 4: Finally, open themain.dart file and insert your code. The Image property from the widget specifies ImageProvider. The image to be shown can come from different sources.

The below are commonly used properties:

- **height/width**: To specify the size constraints of an image
- **repeat**: To repeat the image to cover the available space
- **alignment:** To align the image in a specific position within its bounds
- **fit:** To specify how the image should be inscribed into the available space

Material Design And Ios Cupertino Widgets

Many of the widgets in Flutter are descended in some way from a platform-specific guideline.: Material Design or iOS Cupertino. This helps the developer to follow platformspecific guidelines in the easiest possible way. Flutter, for example, does not have a Button widget;

instead, it provides alternative button implementations for Google Material Design and iOS Cupertino guidelines.

Buttons:

On the Material Design side, Flutter implements the following button components:

- **RaisedButton**: A Material Design raised button. A raised button consists of a rectangular piece of material that hovers over the interface.
- **FloatingActionButton:** A floating action button is a circular icon button that hovers over content to promote a primary action in the application.
- **FlatButton:** A flat button is a section printed on a Material widget that reacts to touches by splashing/rippling with color.
- **IconButton:** An icon button is a picture printed on a Material widget that reacts to touches by splashing/rippling.

Ink, from the Material Design guidelines website, can be explained as follows: "Component that provides a radial action in the form of a visual ripple expanding outward from the user's touch."

- **DropDownButton**: Shows the currently selected item and an arrow that opens a menu for selecting another item.
- **PopUpMenuButton:** Displays a menu when pressed.

Scaffold:

Scaffold implements the basic structure of a Material Design or iOS Cupertino visual layout. For Material Design, the Scaffold widget can contain multiple Material Design components:

- **body:** The primary content of the scaffold. Its displayed below AppBar, if any.
- **AppBar**: An app bar consists of a toolbar and potentially other widgets.
- **TabBar:** A Material Design widget that displays a horizontal row of tabs. This is generally used as part of AppBar.
- **TabBarView**: A page view that displays the widget that corresponds to the currently selected tab. Typically used in conjunction with TabBar and used as a body widget
- **BottomNavigationBar:** Bottom navigation bars make it easy to explore and switch between top-level views in a single tap.
- **Drawer:** A Material Design panel that slides in horizontally from the edge of a scaffold to show navigation links in an application.

In iOS Cupertino, the structure is different with some specific transitions and behaviors.

The available iOS Cupertino classes are CupertinoPageScaffold and CupertinoTabScaffold, which are composed typically with the following:

- **CupertinoNavigationBar**: A top navigation bar. It's typically used with CupertinoPageScaffold.
- **CupertinoTabBar:** A bottom tab bar that is typically used with CupertinoTabScaffold.

Dialogs:

Both Material Design and Cupertino dialogs are implemented by Flutter. On the Material Design side, they are SimpleDialog and AlertDialog; on the Cupertino side, they are CupertinoDialog and CupertinoAlertDialog.

Text Fields:

Text fields are also implemented in both guidelines, by the TextField widget in Material Design and by the CupertinoTextField widget in iOS Cupertino. Both of them display the keyboard for user input. Some of their common properties are as follows:

- **autofocus:** Whether the TextField should be focused automatically (if nothing else is already focused).
- **enabled:** To set the field as editable or not.
- **keyboardType:** To change the type of keyboard displayed to the user when editing.

Selection widgets:

The available control widgets for selection in Material Design are as follows:

- **Checkbox** allows the selection of multiple options in a list.
- **Radio** allows a single selection in a list of options.
- **Switch** allows the toggle (on/off) of a single option.
- **Slider** allows the selection of a value in a range by moving the slider thumb.

On the iOS Cupertino side, some of these widget functionalities do not exist; however, there are some alternatives available:

- **CupertinoActionSheet:** An iOS-style modal bottom action sheet to choose an option among many.

- **CupertinoPicker**: Also a picker control. It's used to select an item in a short list.
- **CupertinoSegmentedControl**: Behaves like a radio button, where the selection is a single item from an options list.
- **CupertinoSlider:** Similar to Slider in Material Design.
- **CupertinoSwitch:** This is also similar to Material Design's Switch.

Date and Time Pickers:

For Material Design, Flutter provides date and time pickers through showDatePicker and showTimePicker functions, which builds and displays the Material Design dialog for the corresponding actions. On the iOS Cupertino side, the CupertinoDatePicker and CupertinoTimerPicker widgets are provided, following the previous CupertinoPicker style.

7.2.9 FLUTTER LISTS

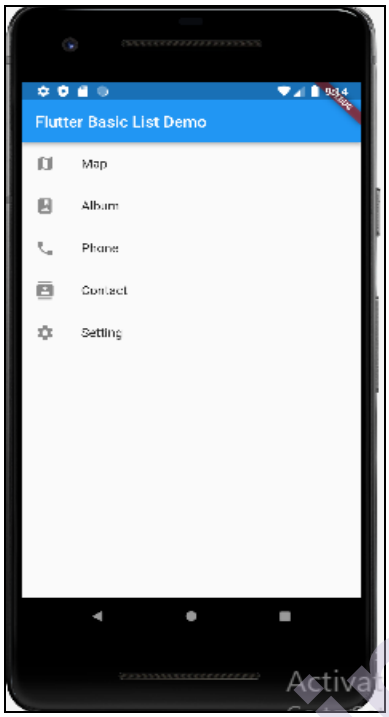
Lists are the most popular elements of every web or mobile application. They are made up of multiple rows of items, which include text, buttons, toggles, icons, thumbnails, and many more. We can use it for displaying various information such as menus, tabs, or to break the monotony of pure text files. Flutter allows you to work with Lists in different ways, which are given below:

1. Basic Lists
2. Long Lists
3. Grid Lists
4. Horizontal Lists

1. Basic Lists:

Flutter includes a ListView widget for working with Lists, which is the fundamental concept of displaying data in the mobile apps. The ListView is a perfect standard for displaying lists that contains only a few items. ListView also includes ListTitle widget, which gives more properties for the visual structure to a list of data.

The following example displays a basic list in the Flutter application.



Android SDK built for x86 (mobile)

README.md | main.dart | widget_test.dart

```dart
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final appTitle = 'Flutter Basic List Demo';

    return MaterialApp(
      title: appTitle,
      home: Scaffold(
        appBar: AppBar(
          title: Text(appTitle),
        ), // AppBar
        body: ListView(
          children: <Widget>[
            ListTile(
              leading: Icon(Icons.map),
              title: Text('Map'),
            ), // ListTile
            ListTile(
              leading: Icon(Icons.photo_album),
              title: Text('Album'),
            ), // ListTile
            ListTile(
              leading: Icon(Icons.phone),
              title: Text('Phone'),
            ), // ListTile
            ListTile(
              leading: Icon(Icons.contacts),
              title: Text('Contact'),
            ), // ListTile
            ListTile(
              leading: Icon(Icons.settings),
              title: Text('Setting'),
            ), // ListTile
          ], // <Widget>[]
        ), // ListView
      ), // Scaffold
    ); // MaterialApp
  }
}
```

2. Working with Long Lists:

Sometimes you want to display a very long list in a single screen of your app, then, in that case, the above method for displaying the lists is not perfect. To work with a list that contains a very large number of

items, we need to use a ListView.builder() constructor. The main difference between ListView and ListView.builder is that ListView creates all items at once, whereas the ListView.builder() constructor creates items when they are scrolled onto the screen.

3. Creating a Grid Lists:

Sometimes we want to display the items in a grid layout rather than the normal list that comes one after next. A GridView widget allows you to create a grid list in Flutter. The simplest way to create a grid is by using the GridView.count() constructor, which specifies the number of rows and columns in a grid. The simplest way to get started using grids is by using the GridView.count() constructor, because it allows you to specify how many rows or columns you'd like. To visualize how GridView works, generate a list of 100 widgets that display their index in the list.

README.md | main.dart | widget_test.dart

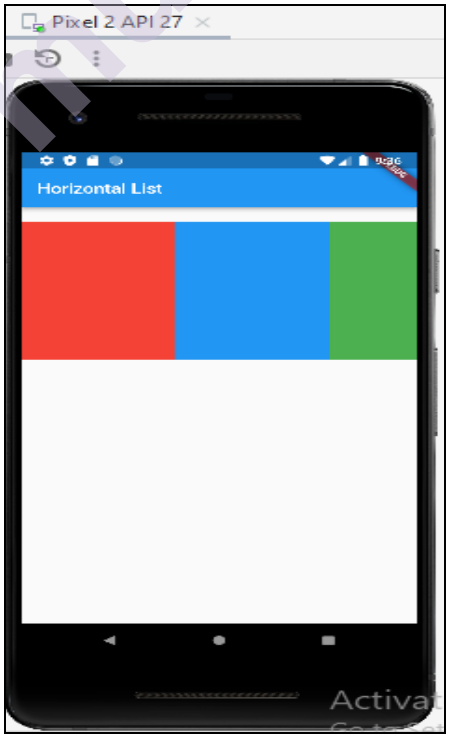
```dart
import 'package:flutter/material.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    const title = 'Grid List';

    return MaterialApp(
      title: title,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(title),
        ), // AppBar
        body: GridView.count(
          // Create a grid with 2 columns. If you change the scrollDirection to
          // horizontal, this produces 2 rows.
          crossAxisCount: 2,
          // Generate 100 widgets that display their index in the List.
          children: List.generate(100, (index) {
            return Center(
              child: Text(
                'Item $index',
                style: Theme.of(context).textTheme.headline5,
              ), // Text
            ); // Center
          }), // List.generate
        ), // GridView.count
      ), // Scaffold
    ); // MaterialApp
  }
}
```

4. Creating a Horizontal List:

The ListView widget also supports horizontal lists. Sometimes we want to create a list that can scroll horizontally rather than vertically. In that case, ListView provides the horizontal scrollDirection that overrides the vertical direction. You might want to create a list that scrolls horizontally rather than vertically. The ListView widget supports horizontal lists. Use the standard ListView constructor, passing in a horizontal scrollDirection, which overrides the default vertical direction.

ADME.md | main.dart | widget_test.dart

```dart
import 'package:flutter/material.dart';
void main() => runApp(const MyApp());
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    const title = 'Horizontal List';
    return MaterialApp(
      title: title,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(title),
        ), // AppBar
        body: Container(
          margin: const EdgeInsets.symmetric(vertical: 20.0),
          height: 200.0,
          child: ListView(
            // This next line does the trick.
            scrollDirection: Axis.horizontal,
            children: <Widget>[
              Container(
                width: 160.0,
                color: Colors.red,
              ), // Container
              Container(
                width: 160.0,
                color: Colors.blue,
              ), // Container
              Container(
                width: 160.0,
                color: Colors.green,
              ), // Container
              Container(
                width: 160.0,
                color: Colors.yellow,
              ), // Container
              Container(
                width: 160.0,
                color: Colors.orange,
              ), // Container
            ], // <Widget>[]
          ), // ListView
        ), // Container
      ), // Scaffold
    ); // MaterialApp
  }
}
```



7.2.10 FLUTTER NAVIGATION

Navigation and routing are some of the core concepts of all mobile application, which allows the user to move between different pages. We know that every mobile application contains several screens for displaying different types of information. For example, an app can have a screen that contains various products. When the user taps on that product, immediately it will display detailed information about that product.

7.2.10.1 Route:

Apps are the new trend. The number of apps available in the play stores nowadays are quite a lot. The apps display their content in a full-screen container called pages or screens. In flutter, the pages or screens are called as the Routes. In android, these pages/screens are referred as Activity and in iOS, it is referred as ViewController. But, in flutter, routes are referred as Widgets. In Flutter, a Page / Screen is called as a Route.

Creating routes:

A route can be written in the form of a "Class" in Dart using object-oriented concepts. Each route can be written as a separate class having its own contents and UI. In Flutter, the screens and pages are known as routes, and these routes are just a widget. In Android, a route is similar to an Activity, whereas, in iOS, it is equivalent to a ViewController. In any mobile app, navigating to different pages defines the workflow of the application, and the way to handle the navigation is known as routing. Flutter provides a basic routing class MaterialPageRoute and two methods Navigator.push() and Navigator.pop() that shows how to navigate between two routes. The following steps are required to start navigation in your application.

- Step 1: First, you need to create two routes.
- Step 2: Then, navigate to one route from another route by using the Navigator.push() method.
- Step 3: Finally, navigate to the first route by using the Navigator.pop() method.

7.2.10.2 NAVIGATOR

As the name suggests, Navigator is a widget that helps us to navigate between the routes. The navigator follows stack method when dealing with the routes. Based on the actions made by the user, the routes are stacked one over the other and when pressed back, it goes to the most recently visited route. Navigator is a widget that follows a stack discipline.

Defining Home:

While navigating, the first thing that we need to do is to define or initialize the "home page". The home page can be any route according to our need. The home usually will be placed in the bottom of the navigator stack. Now let's see how to initialize our HomeRoute() as our home page:

```
void main()
{
runApp(MaterialApp(
 home: HomeRoute(),
));
}
```

Navigating to a Page:

Since, we have defined our Home, all the remaining is to navigate from home to another route of the app. For that the navigator widget has a method called Navigator.push(). This method pushes the route on top of the home, thereby displaying the second route. The code for pushing a route into the stack is as follows:

```
// Within the `HomeRoute`
widget
onPressed: () {
Navigator.push( context,
      MaterialPageRoute(buil
der: (context) =>
SecondRoute()),
);
}
```

Navigating Back to Home: Now we have arrived to our destination, but how do we go back home? For that, the navigator has a method called Navigator.pop(). This helps us to remove the present route from the stack, so that we go back to our home route. This can be done as follows:

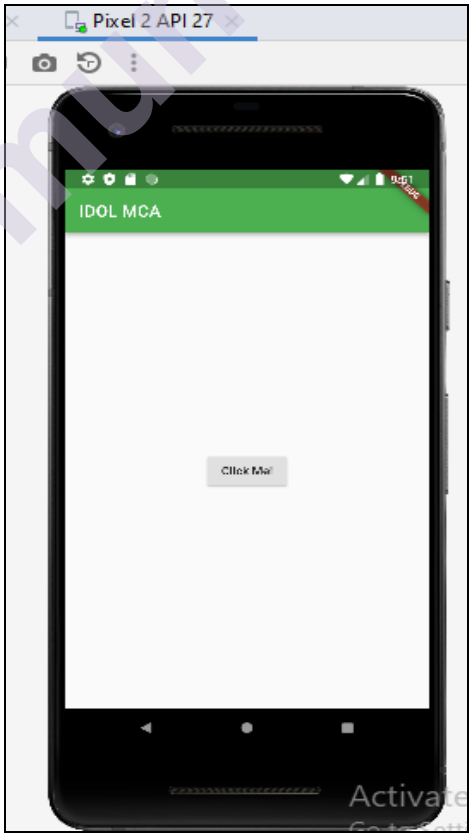
```
// Within the SecondRoute
widget
onPressed: () {
Navigator.pop(context);
}
```

Create two routes:

Here, we are going to create two routes for navigation. In both routes, we have created only a single button. When we tap the button on the first page, it will navigate to the second page. Again, when we tap the button on the second page, it will return to the first page. The code are as follows:

README.md | main.dart | widget_test.dart

```
import 'package:flutter/material.dart';

void main() {
  runApp(MaterialApp(
    home: HomeRoute(),
  ));  // MaterialApp
}
class HomeRoute extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("IDOL MCA"),
        backgroundColor: Colors.green,
      ),  // AppBar
      body: Center(
        child: RaisedButton(
          child: Text('Click Me!'),
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(builder:
                  (context) => SecondRoute()),  // MaterialPageRoute
            );
          }
        ),  // RaisedButton
      ),  // Center
    );  // Scaffold
  }
}

class SecondRoute extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Click Me Page"),
        backgroundColor: Colors.green,
      ),  // AppBar
      body: Center(
        child: RaisedButton(
          onPressed: () {
            Navigator.pop(context);
          },
          child: Text('Home!'),
        ),  // RaisedButton
      ),  // Center
    );  // Scaffold
  }
}
```



7.2.11 UNDERSTANDING BUILT-IN LAYOUT WIDGETS & EFFECTS

Some widgets seem not to appear on screen to the user, but if they are in the widget tree, they will be there somehow, affecting how a child widget looks (such as how it is positioned or styled, for example). To position a button in the bottom corner of the screen, for example, we could specify a positioning related to the screen, but as you may have noticed, buttons and other widgets do not have a Position property. So, you might be asking yourself, "How are widgets organized on the screen?" The answer is widgets again. That's right! Flutter provides widgets to compose the layout itself, with positioning, sizing, styling, and so on

7.2.11.1 Containers:

Displaying a single widget onscreen is not a good way to organize a UI. We will usually layout a list of widgets that are organized in a specific way; to do so, we use container widgets. The most common containers in Flutter are the Row and Column widgets. They have a

children property that expects a list of widgets to display in some direction (that is, a horizontal list for Row, or a vertical list for Column). Another widely used widget is the Stack widget, which organizes children in layers, where one child can overlap another child partially or totally.

If you have developed some kind of mobile application before, you may have already used lists and grids. Flutter provides classes for both of them: namely, the ListView and GridView widgets. Also, other less typical but nonetheless important container widgets are available, such as Table, for example, which organizes children in a tabular layout.

7.2.11.2 Effects:

Gestures, Animations, and Transformations:

Flutter provides widgets for anything related to UI. For example, gestures such as scrolling or touches will all be related to a widget that manages gestures. Animations and transformations, such as scaling and rotation, are also all managed by specific widgets.

Styling and Positioning:

You can also create great effect in flutter with styling and positioning.The task of positioning a child widget in a container, such as a Stack widget, for example, is done by using other widgets. Flutter provides widgets for very specific tasks. Centering widget inside a container is done by wrapping it into a Center widget. Aligning a child widget relative to a parent can be done with the Align widget, where you specify the desired position through its alignment property. Another useful widget is Padding, which allows us to specify a space around the given child. The functionalities of these widgets are aggregated in the Container widget, which combines those common positioning and styling widgets to apply them to a child directly, making the code much cleaner and shorter.

7.2.12 BUILDING LAYOUT

Since the core concept of Flutter is Everything is widget, Flutter incorporates a user interface layout functionality into the widgets itself. Flutter provides quite a lot of specially designed widgets like Container, Center, Align, etc., only for the purpose of laying out the user interface. Widgets build by composing other widgets normally use layout widgets.

Layout a widget:

The following steps show how to layout a widget:

- **Step 1:** First, you need to select a Layout widget.
- **Step 2:** Next, create a visible widget.

- **Step 3:** Then, add the visible widget to the layout widget.
- **Step 4:** Finally, add the layout widget to the page where you want to display.

7.2.12.1 Type of Layout Widgets:

Layout widgets can be grouped into two distinct category based on its child:

1. Widget supporting a single child
2. Widget supporting multiple child

Let us learn both type of widgets and its functionality in the upcoming part.

Single Child Widgets:

In this category, widgets will have only one widget as its child and every widget will have a special layout functionality. For example, Center widget just centers it child widget with respect to its parent widget and Container widget provides complete flexibility to place it child at any given place inside it using different option like padding, decoration, etc., Single child widgets are great options to create high quality widget having single functionality such as button, label, etc.,

Layout elements like padding and rows are also widgets.

Padding:

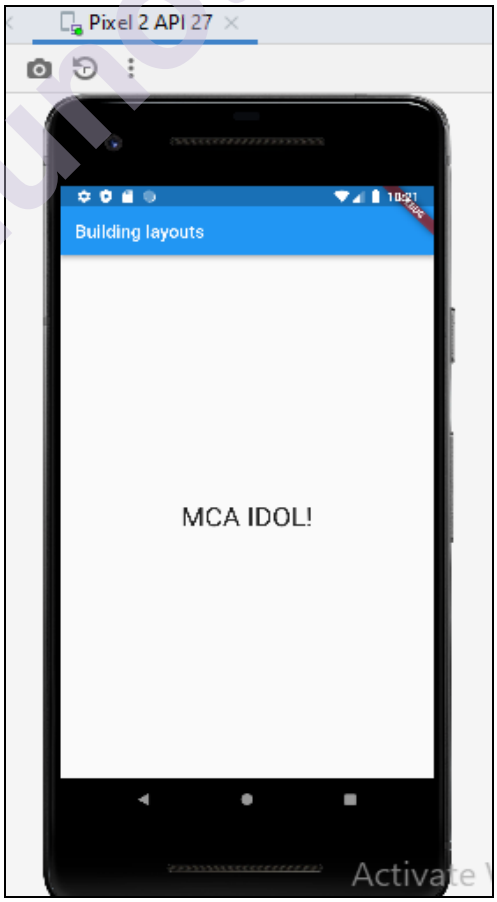
Thinking of padding as a widget is strange when you first come to Flutter. At least in iOS and Android world, padding is a parameter. But in Flutter, if you want add some extra space around a widget, then you wrap it in a Padding widget. The EdgeInsets parameter is used to specify the amount of padding. Here all was used to indicate that every side (left, top, right, and bottom) should have equal padding. If you want them to have different values, then you can use only instead of all.

Alignment:

To center a widget, the concept is the same as it was for padding. This time you just wrap your widget with a Center widget. You can type it out or there is a shortcut menu option for it, too. In addition to centering it, I added some styling to the Text widget so that the font size is more visible.

README.md | main.dart | widget_test.dart

```dart
import 'package:flutter/material.dart';

// entry point for the app.
// the => operator is shorthand for {} when there is only one line of code
void main() => runApp(MyApp());

// the root widget of our application
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text("Building layouts"),
        ),  // AppBar
        body: myLayoutWidget(),
      ),  // Scaffold
    );  // MaterialApp
  }
}

Widget myLayoutWidget() {
  return Center(
    child: Text(
      "MCA IDOL!",
      style: TextStyle(fontSize: 30),
    ),  // Text
  );  // Center
}
```



We can modify our above application to build the home page using basic layout widgets as specified below:

Container:

Generic, single child, box based container widget with alignment, padding, border and margin along with rich styling features. Container widget is the top level or root widget. Container is configured using decoration and padding property to layout its content. BoxDecoration has many properties like color, border, etc., to decorate the Container widget and here, color is used to set the color of the container , padding of the Container widget is set by using dgeInsets class, which provides the option to specify the padding value.

Center:

Simple, Single child container widget, which centers its child widget. Center is the child widget of the Container widget. Again, Text is the child of the Center widget. Text is used to show message and Center is used to center the text message with respect to the parent widget, Container.

We can also use

- **FittedBox:** It scales the child widget and then positions it according to the specified fit.
- **AspectRatio:** It attempts to size the child widget to the specified aspect ratio.

The list of different types of single child widgets are:

- **Container:** It is the most popular layout widget that provides customizable options for painting, positioning, and sizing of widgets.
- **Padding:** It is a widget that is used to arrange its child widget by the given padding. It contains EdgeInsets and EdgeInsets.fromLTRB for the desired side where you want to provide padding.
- **Center:** This widget allows you to center the child widget within itself.
- **Align:** It is a widget, which aligns its child widget within itself and sizes it based on the child's size. It provides more control to place the child widget in the exact position where you need it.
- **SizedBox**: This widget allows you to give the specified size to the child widget through all screens.
- **AspectRatio:** This widget allows you to keep the size of the child widget to a specified aspect ratio.

- **Baseline:** This widget shifts the child widget according to the child's baseline.
- **ConstrainedBox:** It is a widget that allows you to force the additional constraints on its child widget. It means you can force the child widget to have a specific constraint without changing the properties of the child widget.
- **CustomSingleChildLayout:** It is a widget, which defers from the layout of the single child to a delegate. The delegate decides to position the child widget and also used to determine the size of the parent widget.
- **FittedBox:** It scales and positions the child widget according to the specified fit.
- **FractionallySizedBox:** It is a widget that allows to sizes of its child widget according to the fraction of the available space.
- **IntrinsicHeight and IntrinsicWidth:** They are a widget that allows us to sizes its child widget to the child's intrinsic height and width.
- **LimitedBox:** This widget allows us to limits its size only when it is unconstrained.
- **Offstage:** It is used to measure the dimensions of a widget without bringing it on to the screen.
- **OverflowBox:** It is a widget, which allows for imposing different constraints on its child widget than it gets from a parent. In other words, it allows the child to overflow the parent widget.

Multiple Child Widgets:

The multiple child widgets are a type of widget, which contains more than one child widget, and the layout of these widgets are unique. For example, Row widget laying out of its child widget in a horizontal direction, and Column widget laying out of its child widget in a vertical direction. If we combine the Row and Column widget, then it can build any level of the complex widget. In this category, a given widget will have more than one child widgets and the layout of each widget is unique. For example, Row widget allows the laying out of its children in horizontal direction, whereas Column widget allows laying out of its children in vertical direction. By composing Row and Column, widget with any level of complexity can be built. Let us learn some of the frequently used widgets in this section.

- **Row:** It allows to arrange its child widgets in a horizontal direction.
- **Column:** It allows to arrange its child widgets in a vertical direction.

- **ListView:** It is the most popular scrolling widget that allows us to arrange its child widgets one after another in scroll direction.
- **GridView**: It allows us to arrange its child widgets as a scrollable, 2D array of widgets. It consists of a repeated pattern of cells arrayed in a horizontal and vertical layout.
- **Expanded**: It allows to make the children of a Row and Column widget to occupy the maximum possible area.
- **Table:** It is a widget that allows us to arrange its children in a table based widget.
- **Flow:** It allows us to implements the flow-based widget.
- **Stack:** It is an essential widget, which is mainly used for overlapping several children widgets. It allows you to put up the multiple layers onto the screen.

README.md | main.dart | widget_test.dart

```dart
import 'package:flutter/material.dart';
void main() => runApp(MyApp());
class MyApp extends StatelessWidget {
  // It is the root widget of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Multiple Layout Widget for IDOL',
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        // This is the theme of your application.
        primarySwatch: Colors.blue,
      ), // ThemeData
      home: MyHomePage(),
    ); // MaterialApp
  }
}
class MyHomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: Container(
        alignment: Alignment.center,
        color: Colors.white,
        child: Stack(
          children: <Widget>[
            // Max Size
            Container(
              color: Colors.blue,
            ), // Container
            Container(
              color: Colors.pink,
              height: 400.0,
              width: 300.0,
            ), // Container
            Container(
              color: Colors.yellow,
              height: 220.0,
              width: 200.0,
            ) // Container
          ], // <Widget>[]
        ), // Stack
      ),); } } // Container, Center
```

7.2.13 SUMMARY

In this chapter, we finally started playing with the Flutter framework. First, we learned some important concepts about Flutter, mainly the concepts of widgets. We saw that widgets are the central part of the Flutter world, where the Flutter team continually works to improve existing widgets and add new ones. This is because the widget concept is everywhere, from rendering performance to the final result on screen. We also saw how to start a Flutter application project with the framework tools, the basic project structure of files, and the peculiarities of the pubspec file. We also learned how to run a project on an emulator. We have seen each of the available Flutter widget types and their differences. Stateless widgets do not get rebuilt frequently by the framework; on the other hand, Stateful widgets get rebuilt every time its associated State object

changes (which could be when the setState() function is used, for example). We have also seen that Flutter comes with many widgets that can be combined to build unique UIs, and that also they do not

need to be visual components on the user's screen; they can be layout, styling, and even data widgets, such as InheritedWidget

7.2.14 QUESTIONS

1. What is flutter? Write the advantages of using flutter.
2. Explain the flutter widget and write its importance
3. Write the difference between runApp() and main() in flutter.
4. Explain Flutter User Interface?
5. Explain Flutter Architecture?
6. What are the types of widgets?
7. Explain Navigation in Flutter
8. Explain types of Lists in Brief?

7.2.15 CHAPTER END EXERCISE

1. Create a Bar chart using Flutter application?
2. Create a Stateless Widget(say, Myapp),add scaffold widget to hold the containers and add container containing the images inside a ListView constructor?

7.2.16 REFERENCES

1. : https://flutter.dev/docs/get-started/flutter-for/declarative
2. https://flutter.dev/docs/resources/faq#how-does-flutterrun-my-code-on-android.
3. https://flutter.dev/docs/resources/faq#how-does-flutter-run-my-code-on-ios.
4. https://docs.flutter.io/flutter/widgets/Element-class.html.

*****

8

DATA HANDLING

Unit structure

8.0 OBJECTIVES

After this Chapter you will be able to learn and understand,

- The basic functionality of JavaScript Object Notation (JSON) and to develop the data interchange format.
- How to structure data by using the JSON file format.
- How to create model classes to handle JSON serialization.
- How to use the Future class to save, read, and parse JSON files.
- Manage the style of your widgets (Like Colors, Text appearance (fonts, weights, underlines, etc.), Borders (thicknesses, patterns, corner radii), Background images and Applying shapes to a Container.
- State Management in Flutter.

8.1 INTRODUCTION

In this chapter, you'll learn how to persist data—that is, save data on the device's local storage directory—across app launches by using the JSON file format and saving the file to the local iOS and Android filesystem. JavaScript Object Notation (JSON) is a common open-standard and language-independent file data format with the benefit of being human-readable text. Persisting data is a two-step process; first you use the File class to save and read data, and second, you parse the data from and to a JSON format. You'll create a class to handle saving and reading the data file that uses the File class. You'll also create a class to parse the full list of data by using json.encode and json.decode and a class to extract each record. And you'll create another class to handle passing an action and an individual journal entry between pages.

Furthemore we will study about Flutter Form widgets. Forms are an integral part of all modern mobile and web applications. It is mainly used to interact with the app as well as gather information from the users. Form widgets allows users to input data and provides them interaction capability with the application with that we will also study form validation. Validating user input is an essential part of app development. This process makes the app more secure and checks whether the information provided by the user is what we anticipate them to provide so as not to introduce bugs into our application. Flutter comes with a native way of validating user inputs using the Form and TextFormField widget.

We will than study about styling that affects the appearance of widgets. Things like Colors, Text appearance (fonts, weights, underlines, etc.), Borders (thicknesses, patterns, corner radii), Background images and applying shapes to a Container. Further we will also study about state management. Managing state in an application is one of the most important and necessary process in the life cycle of an application. It centralizes all the states of various UI controls to handle data flow across the application.

8.2 UNDERSTANDING THE JSON FORMAT

JSON or JavaScript Object Notation is a lightweight text-based open standard designed for human-readable data interchange. The JSON format is text-based and is independent of programming languages, meaning any of them can use it. It's the most widely used format for Representational State Transfer (REST)-based APIs that servers provide. If you talk to a server that has a REST API, it will most likely return data in a JSON format. It's a great way to exchange data between different programs because it is human-readable text. JSON uses the key/value pair, and the key is enclosed in quotation marks followed by a colon and then the value like "id":"100". You use a comma (,) to separate multiple key/value pairs.

Table 8.1 shows some examples.

KEY	COLON	VALUE
"id"	:	"100"
"quantity"	:	3
"in_stock"	:	true

The types of values you **can** use are Object, Array, String, Boolean, and Number. Objects are declared by curly ({}) brackets, and inside you use the key/value pair and arrays. You declare arrays by using the square ([]) brackets, and inside you use the key/value or just the value.

Table 8.2 shows some examples.

TYPE	SAMPLE

Object	{
	"id": "100",
	"name": "Vacation"
	}

Array with values only	["Family", "Friends", "Fun"]

Array with key/value	[
	{
	"id": "100",
	"name": "Vacation"
	},
	{
	"id": "102",
	"name": "Birthday"
	}
	]

Object with array	{
	"id": "100",
	"name": "Vacation",
	"tags": ["Family", "Friends", "Fun"]
	}

Multiple objects with arrays	{
	"journals":[
	{
	"id":"4710827",
	"mood":"Happy"
	},
	{
	"id":"427836",
	"mood":"Surprised"
	},
	],
	"tags":[
	{
	"id": "100",
	"name": "Family"
	},
	{
	"id": "102",
	"name": "Friends"
	}
	]
	}

8.2.1 Simple Example of Json File:

The following is an example of the JSON file that you'll create for the journal application. The JSON file is used to save and read the journal data from the device local storage area, resulting in data persistence over app launches. You have the opening and closing curly brackets declaring an object. Inside the object, the journal's key contains an array of objects separated by a comma. Each object inside the array is a journal entry with key/value pairs declaring the id, date, mood, and note. The id key value is used to uniquely identify each journal entry and isn't displayed in the UI. How the value is obtained depends on the project requirement; for example, you can use sequential numbers or calculate a unique value by using characters and numbers (universally unique identifier [UUID]).

```
{
 "journals":[
 {
 "id":"470827",
 "date":"2019-01-13 00:27:10.167177",
 "mood":"Happy",
 "note":"Cannot wait for family night."
 },
 {
 "id":"427836",
 "date":"2019-01-12 19:54:18.786155",
 "mood":"Happy",
 "note":"Great day watching our favorite
shows."
 },
 ],
}
```


8.2.2 Uses of JSON:

- It is used while writing JavaScript based applications that includes browser extensions and websites.
- JSON format is used for serializing and transmitting structured data over network connection.
- It is primarily used to transmit data between a server and web applications.
- Web services and APIs use JSON format to provide public data.
- It can be used with modern programming languages.

8.2.3 Characteristics of JSON:

- JSON is easy to read and write.
- It is a lightweight text-based interchange format.
- JSON is language independent.

8.3 USING DATABASE CLASSES TO WRITE, READ, AND SERIALIZE JSON

To create reusable code to handle the database routines such as writing, reading, and serializing (encoding and decoding) data, you'll place the

logic in classes. You'll create four classes to handle local persistence, with each class responsible for specific tasks.

- The **DatabaseFileRoutines** class uses the File class to retrieve the device local document directory and save and read the data file.
- The **Database** class is responsible for encoding and decoding the JSON file and mapping it to a List.
- The **Journal** class maps each journal entry from and to JSON.
- The **JournalEdit** class is used to pass an action (save or cancel) and a journal entry between pages.

The **DatabaseFileRoutines** class requires you to import the **dart:io** library to use the **File** class responsible for saving and reading files. It also requires you to import the **path_provider** package to retrieve the local path to the document directory. The **Database** class requires you to import the **dart:convert** library to decode and encode JSON objects.

The first task in local persistence is to retrieve the directory path where the data file is located on the device. Local data is usually stored in the application documents directory; for iOS, the folder is called **NSDocumentDirectory**, and for Android it's **AppData**. To get access to these folders, you use the **path_provider** package (Flutter plugin). You'll be calling the **getApplicationDocumentsDirectory()** method, which returns the directory giving you access to the path variable.

```
Future<String> get _localPath async {
 final directory = await getApplicationDocumentsDirectory();
 return directory.path;
}
```

Once you retrieve the path, you append the data filename by using the File class to create a File object. You import the **dart:io** library to use the File class, giving you a reference to the file location.

```
final path = await _localPath;
Final file = File('$path/local_persistence.json');
```

Once you have the File object, you use the **writeAsString()** method to save the file by passing the data as a **String** argument. To read the file, you use the readAsString() method without any arguments. Note that the file variable contains the documents folder's path and the data filename.

```
// Write the file
file.writeAsString('$json');
// Read the file
String contents = await file.readAsString();
```

As you learned in the "Understanding the JSON Format" section, you use a JSON file to save and read data from the device local storage. The JSON file data is stored as plain text (strings). To save data to the JSON file, you use serialization to convert an object to a string. To read data from the JSON file, you use deserialization to convert a string to an object. You use the **json.encode()** method to serialize and the **json.decode()** method to deserialize the data. Note that both the json.encode() and json.decode() methods are part of the **JsonCodec** class from the dart:convert library. To serialize and deserialize JSON files, you import the **dart:convert** library. After calling the **read-AsString()** method to read data from the stored file, you need to parse the string and return the JSON object by using the json.decode() or jsonDecode() function. Note that the jsonDecode() function is shorthand for json.decode().

```
// String to JSON object
final dataFromJson = json.decode(str);
// Or
final dataFromJson = jsonDecode(str);
```

To convert values to a JSON string, you use the json.encode() or jsonEncode() function. Note that jsonEncode() is shorthand for json.encode(). It's a personal preference deciding which approach to use; in the exercises, you'll be using json.decode() and json.encode().

```
// Values to JSON string
json.encode(dataToJson);
// Or
jsonEncode(dataToJson);
```

8.4 FORMATTING DATES

To format dates, you use the intl package (Flutter plugin) providing internationalization and localization. The full list of available date formats is available on the intl package page site at https://pub.dev/packages/intl. For our purposes, you'll use the DateFormat class to help you N format and parse dates. You'll use the DateFormat named constructors to format the date according to the specification. To format a date like Jan 13, 2022, you use the DateFormat.yMMD() constructor, and then you pass the date to the format argument, which expects a DateTime. If you pass the date as a String, you use DateTime.parse() to convert it to a DateTime format.

```
// Formatting date examples
print(DateFormat.d().format(DateTime.parse('2022-
01-13')));
print(DateFormat.E().format(DateTime.parse('2022
-01-13')));
print(DateFormat.y().format(DateTime.parse('2022-
01-13')));
print(DateFormat.yMEd().format(DateTime.parse('
2022-01-13')));
print(DateFormat.yMMMEd().format(DateTime.pa
rse('2022-01-13')));
print(DateFormat.yMMMMEEEEd().format(DateTi
me.parse('2022-01-13')));

I/flutter (19337): 13
I/flutter (19337): Thu
I/flutter (19337): 2022
I/flutter (19337): Thu, 1/13/2022
I/flutter (19337): Thu, Jan 13, 2022
I/flutter (19337): Thursday, January 13, 2022
```

To build additional custom date formatting, you can chain and use the add_*() methods (substitute the * character with the format characters needed) to append and compound multiple formats. The following sample code shows how to customize the date format.

```
// Formatting date examples with the add_* methods
print(DateFormat.yMEd().add_Hm().format(DateTime.p
arse('2022-01-13 10:30:15')));
print(DateFormat.yMd().add_EEEE().add_Hms().format
(DateTime.parse('2022-01-13
10:30:15')));
I/flutter (19337): Thu, 1/13/2022 10:30
I/flutter (19337): 1/13/2022 Thursday 10:30:15
```

8.4.1 Sorting A List of Dates:

You learned how to format dates easily, but how would you sort dates? The journal app that you'll create requires you to show a list of entries, and it would be great to be able to display the list sorted by date. In particular, you want to sort dates to show the newest first and oldest last, which is known as DESC (descending) order. Our journal entries are

displayed from a List, and to sort them, you call the List().sort() method. The List is sorted by the order specified by the function, and the function acts as a Comparator, comparing two values and assessing whether they are the same or whether one is larger than the other- such as the dates 2022-01-20 and 2022-01-22 in Table 8.3. The Comparator function returns an integer as negative, zero, or positive. If the comparison—for example, 2022-01-20 > 2022-01-22- is true, it returns 1, and if it's false, it returns -1. Otherwise (when the values are equal), it returns 0.

TABLE 8.3: Sorting Dates

COMPARE	TRUE	SAME	FALSE
date2.compareTo(date1)	1	0	-1
2022-01-20 > 2022-01-22			-1
2022-01-20 < 2022-01-22	1		
2022-01-22 = 2022-01-22		0	

8.5 FLUTTER FORMS

Forms are an integral part of all modern mobile and web applications. It is mainly used to interact with the app as well as gather information from the users. They can perform many tasks, which depend on the nature of your business requirements and logic, such as authentication of the user, adding user, searching, filtering, ordering, booking, etc. A form can contain text fields, buttons, checkboxes, radio buttons, etc. Flutter provides a Form widget to create a form. The form widget acts as a container, which allows us to group and validate the multiple form fields. When you create a form, it is necessary to provide the GlobalKey. This key uniquely identifies the form and allows you to do any validation in the form fields. Each individual form field should be wrapped in a FormField widget, with the Form widget as a common ancestor of all of those. Call methods on FormState to save, reset, or validate each FormField that is a descendant of this Form. To obtain the FormState, you may use Form.of with a context whose ancestor is the Form, or pass a GlobalKey to the Form constructor and call GlobalKey.currentState.

8.5.1 Creating Form:

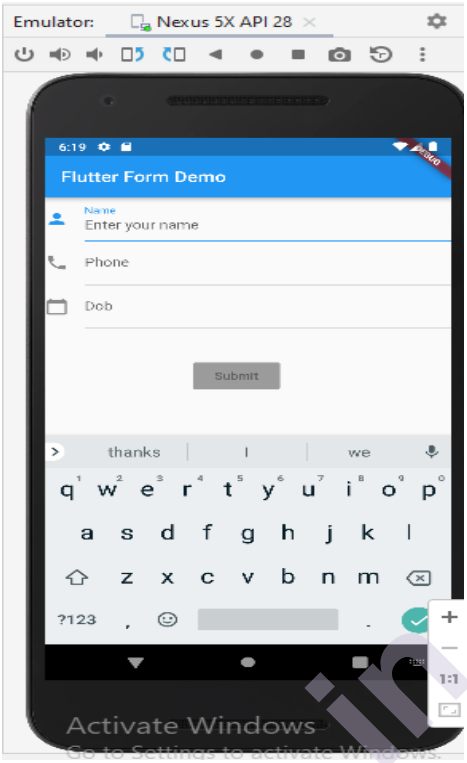
Flutter provides a Form widget to create a form. The form widget acts as a container, which allows us to group and validate the multiple form fields. When you create a form, it is necessary to provide the GlobalKey. This key uniquely identifies the form and allows you to do any validation in the form fields. The form widget uses child widget TextFormField to provide the users to enter the text field. This widget renders a material design text field and also allows us to display validation errors when they occur.

Let us create a form. First, create a Flutter project and replace the following code in the main.dart file. In this code snippet, we have created a custom class named MyCustomForm. Inside this class, we define a global key as _formKey. This key holds a FormState and can use to

retrieve the form widget. Inside the build method of this class, we have added some custom style and use the TextFormField widget to provide the form fields such as name, phone number, date of birth, or just a normal field. Inside the TextFormField, we have used InputDecoration that provides the look and feel of your form properties such as borders, labels, icons, hint, styles, etc. Finally, we have added a button to submit the form.

```
class MyCustomFormState extends State<MyCustomForm> {

  final _formKey = GlobalKey<FormState>();

  @override
  Widget build(BuildContext context) {
    // Build a Form widget using the _formKey created above.
    return Form(
      key: _formKey,
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: <Widget>[
          TextFormField(
            decoration: const InputDecoration(
              icon: const Icon(Icons.person),
              hintText: 'Enter your name',
              labelText: 'Name',
            ),  // InputDecoration
          ),  // TextFormField
          TextFormField(
            decoration: const InputDecoration(
              icon: const Icon(Icons.phone),
              hintText: 'Enter a phone number',
              labelText: 'Phone',
            ),  // InputDecoration
          ),  // TextFormField
          TextFormField(
            decoration: const InputDecoration(
              icon: const Icon(Icons.calendar_today),
              hintText: 'Enter your date of birth',
              labelText: 'Dob',
            ),  // InputDecoration
          ),  // TextFormField
          new Container(
              padding: const EdgeInsets.only(left: 150.0, top: 40.0),
              child: new RaisedButton(
                child: const Text('Submit'),
                onPressed: null,
              )),  // RaisedButton, Container
```


This form contains three field name, phone number, date of birth, and submit button.



8.5.2 Form Validation:

Validation is a method, which allows us to correct or confirms a certain standard. It ensures the authentication of the entered data. The Flutter SDK provides us with an out-of-the-box widget and functionalities to make our lives easier when using form validation. Validating forms is a common practice in all digital interactions. To validate a form in a flutter, we need to implement mainly three steps.

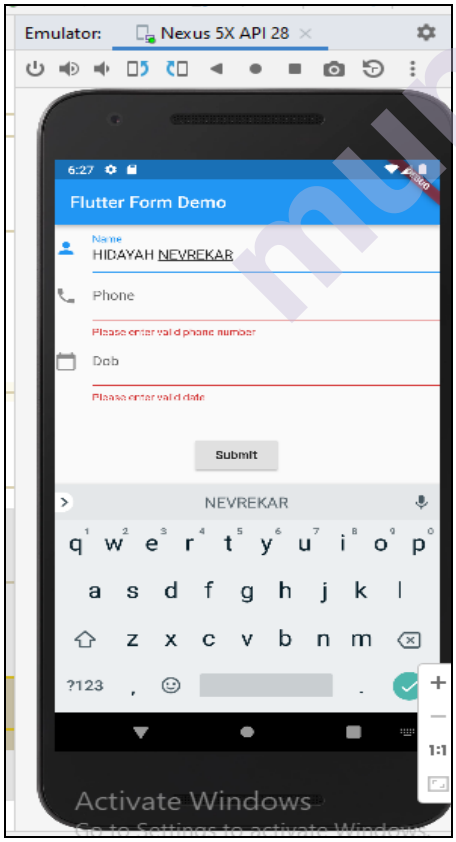
1. Use the Form widget with a global key.
2. Use TextFormField to give the input field with validator property.
3. Create a button to validate form fields and display validation errors.

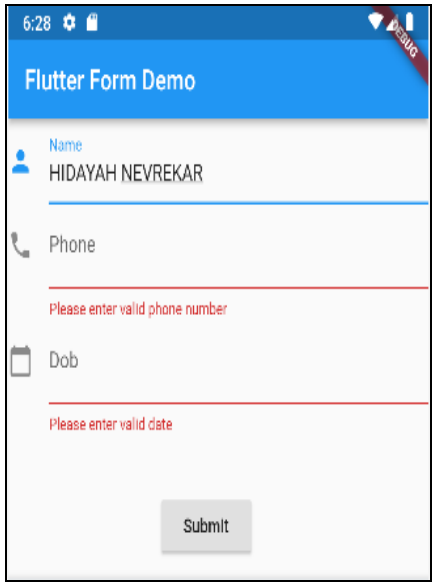
Let us understand it with the following example. In the below code, we have to use validator() function in the TextFormField to validate the input properties. If the user gives the wrong input, the validator function returns a string that contains an error message; otherwise, the validator function return null. In the validator function, make sure that the TextFormField is not empty. Otherwise, it returns an error message. The validator() function can be written as below code snippets:

```
validator: (value) {
      if (value.isEmpty) {
            return 'Please enter some text';
      }
      return null;
},
```

```
  ),  // InputDecoration
  validator: (value) {
    if (value!.isEmpty) {
      return 'Please enter valid phone number';
    }
    return null;
  },
),  // TextFormField
TextFormField(
  decoration: const InputDecoration(
    icon: const Icon(Icons.calendar_today),
    hintText: 'Enter your date of birth',
    labelText: 'Dob',
  ),  // InputDecoration
  validator: (value) {
    if (value!.isEmpty) {
      return 'Please enter valid date';
    }
    return null;
  },
),  // TextFormField
Container(
    padding: const EdgeInsets.only(left: 150.0, top: 40.0),
    child: new RaisedButton(
      child: const Text('Submit'),
      onPressed: () {
        // It returns true if the form is valid, otherwise returns false
        if (_formKey.currentState!.validate()) {
          // If the form is valid, display a Snackbar.
          Scaffold.of(context)
              .showSnackBar(SnackBar(content: Text('Data is in processing.')));
        }
      },
    )),  // RaisedButton, Container
  // <Widget>[]
```

In this form, if you left any input field blank, you will get an error message like below screen.





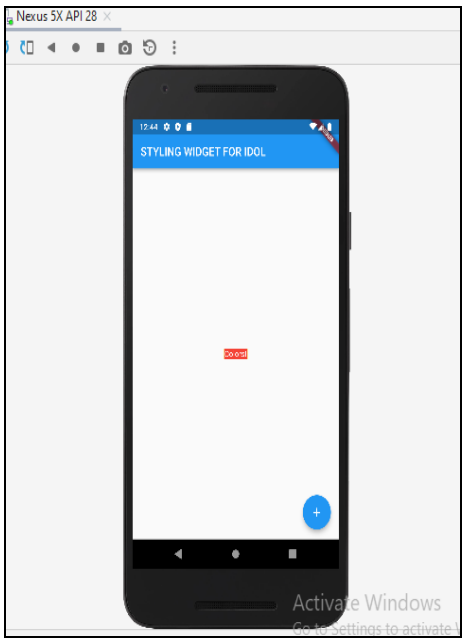
8.6 STYLING YOUR WIDGETS

Flutter has borrowed the best ideas from Android, iOS development, and web development, especially from React and the dynamicness of JavaScript. But it doesn't copy their techniques exactly. Flutter does things its own way, and it is a mistake to take the web analogies too far. It's to our benefit to grasp how Flutter is different. First, Google has made Material Design very popular in Android development and all across the Web. And while Flutter's default look conforms to Material Design, don't let anyone tell you that you're forced to use it. That's a popular but untrue misconception. Second, Flutter's styling is not CSS. Whereas CSS has certain properties that are passed down to their children, Flutter styles are not inherited. You cannot set a font family on your custom widget for example, and have all of the Texts and TextFields and buttons beneath it suddenly begin rendering with that font. To make that kind of thing happen, it is possible to use Themes.

Finally, Flutter, like Dart, is very verbose – it takes a lot of code to express something you'd think should be simple. And unsurprisingly, styling is no different. Just rest assured that some very smart people have written Flutter and have darn good reasons for it being as wordy as it is, safety and completeness being just two. It's a fact of life. We just want you to be aware and prepared for it.

8.6.1 A Word about Colors:

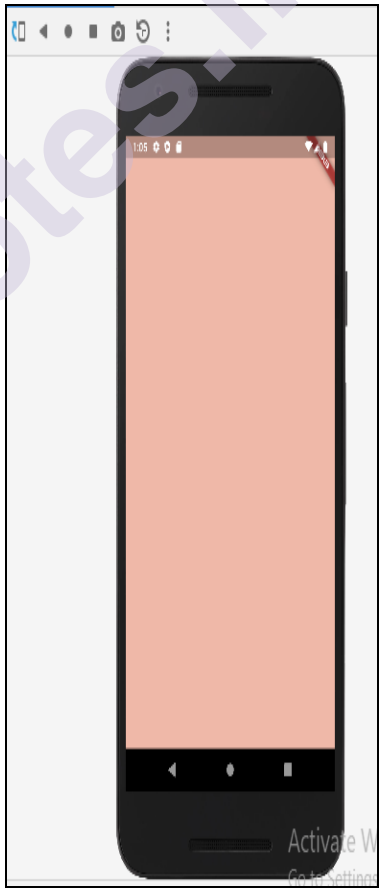
Most Flutter styles are very narrowly applied; they only make sense for certain tightly defined situations. On the other hand, colors are applied nearly everywhere (Figure 8-1). Borders, text, backgrounds, icons, buttons, and shadows all have colors. And they're all specified in the same manner. For example, here's white Text in a red container with a yellow border, and all of those widgets are colored identically with the syntax "color: Colors. somethingOrOther":



```
child: Container(
 child: Text(
 'Colors!',
 style: TextStyle(color:
Colors.white,),
 ),
 decoration: BoxDecoration(
 color: Colors.red,
 border: Border.all(
 color: Colors.yellow,
 )),
),
```

You can create random colors on click using the below code:

```dart
import 'dart:math';
import 'package:flutter/material.dart';

void main() {
  runApp(MaterialApp(home: MyPage()));
}

class MyPage extends StatefulWidget {
  @override
  _MyPageState createState() => new _MyPageState();
}

class _MyPageState extends State<MyPage> {
  final Random _random = Random();

  Color _color = Color(0xFFFFFFFF);
  void changeColor() {
    setState(() {
      _color = Color.fromARGB(
        //or with fromRGBO with fourth arg as _random.nextDouble(),
        _random.nextInt(256),
        _random.nextInt(256),
        _random.nextInt(256),
        _random.nextInt(256),
      );
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: InkWell(
        onTap: changeColor,
        child: Container(
          color: _color,
        ), // Container
      ), // InkWell
    ); // Scaffold
  }
}
```



So you can create any of the 16+ million colors with Color.fromRGBO(red, green, blue, opacity) where each of the three RGB colors is a number between 0 and 255 and the opacity is 1.0 forfully opaque and 0.0 for fully transparent.

8.6.2 Styling Text:

There are two topics regarding the appearance of Text: TextStyle and Custom Fonts.

1. **TextStyle:** Text widgets have a style property which takes a TextStyle object.

You'll simply set the style property to an instance of a TextStyle widget and set properties. TextStyle supports about 20 properties. Here are the most useful:

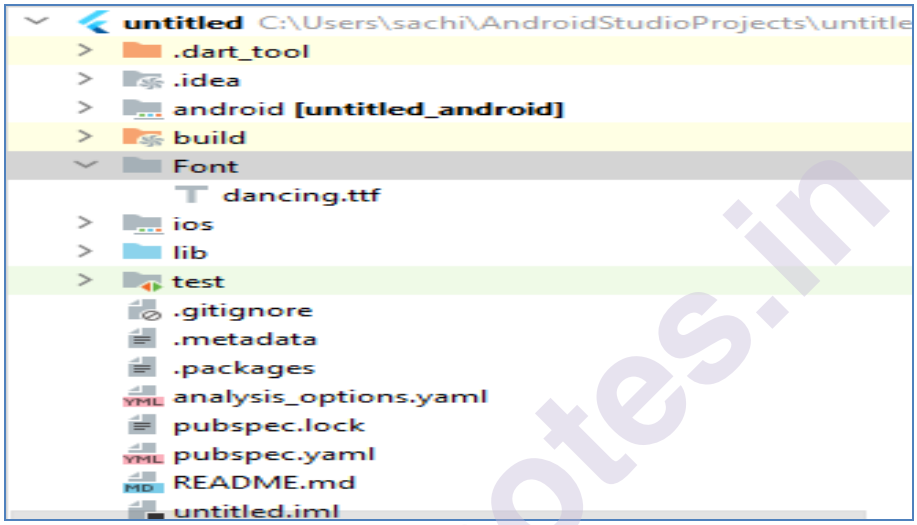
- Color: Any of the valid 16+ million colors.
- Decoration: TextDecoration.underline, overline, strikethrough, none.
- fontSize: A double. The number of pixels tall to make the characters. Default size 14.0 pixels.
- fontStyle: FontStyle.italic or normal.
- fontWeight: FontWeight.w100-w900. Or bold (which is w700) or normal (which is w400).
- fontFamily: A string.
- fontFamily is a bigger topic. There are some fonts that are built-in like Courier, Times New Roman, serif, and a bunch more. How many more? It depends on the type of device on which the app is running.

Since we don't have control over the users' devices, the best practice is for you to stick to the default font family unless you install and use a custom font. Let's talk about how to do that.

2. CUSTOM FONTS:

Certain designers call for custom fonts when they design scenes. It turns out with Flutter, using custom fonts is easy to implement, and they work cross-platform. It involves three steps:

1. Download the custom font files which are in ttf, woff, or woff2 format. These are customarily stored in a root-level folder called fonts, but the name is up to you.



8.6.3 How to Use A Custom Font In A Flutter App?

1. **Add a font to your project:-** Right click on your project folder and go to New > Directory. Call it Font. It should be in your projects root directory. Copy and paste your font into the new assets directory. I'm just using a single font in my example, the regular dancing script font. I renamed it to dancing.ttf.

2. **Register the font:** Open your pubspec.yaml file. Add the fonts info under the flutter section. Indentation is mandatory.

```
flutter:
  fonts:
  - family: dancing
    fonts:
    - asset: Font/dancing.ttf
```

3. **Use the font in your theme or widget**: Now you can use your font by setting the fontFamily attribute in the TextStyle widget to whatever you called it in pubspec.yaml. For this example, I called it dancing.

```
Text(
  'HELLO IDOL STUDENTS',
  style: TextStyle(fontFamily: 'dancing',
  ),
)
```

4. **Restart your app:** Whenever I add assets I find I need to do a full stop and restart. Otherwise I get errors.

8.6.4 Container Decorations:

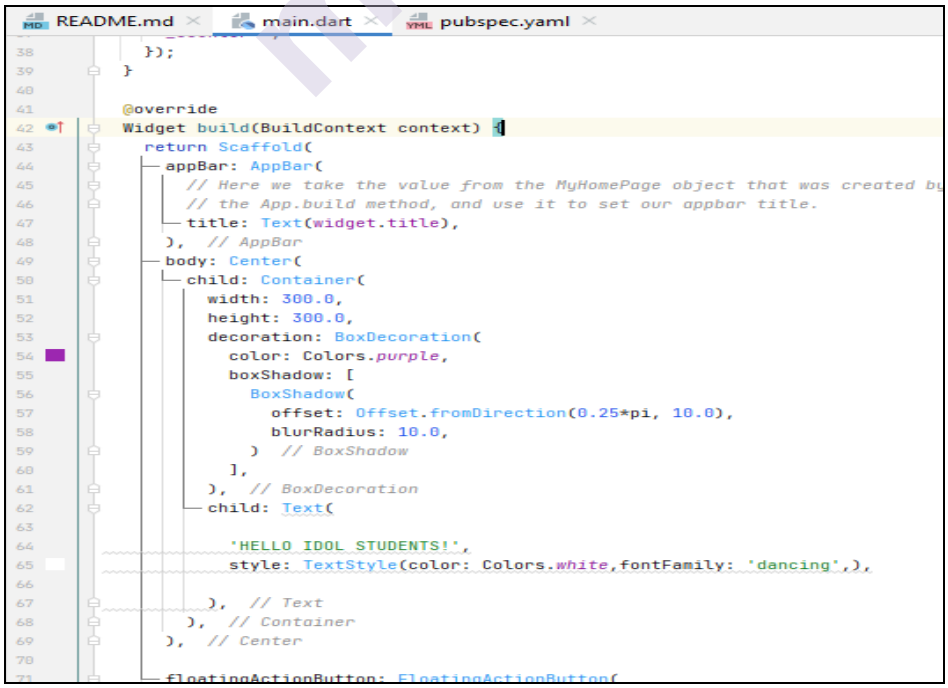
How do you add borders to Text? You can't. How about a background to an Icon? Nope. They don't have the capacity to have those decorations. But you know what does? A Container. When you have styling problems like these, the answer is almost always to wrap widgets in a Container and put a decoration on the Container. Container widget is one of the main layout widgets in flutter, however, it has many properties that can change its look entirely. One of such properties is decoration. The decoration property returns another class BoxDecoration. Containers have a catch-all styling property called decoration. The BoxDecoration class helps us with variety of ways to draw a box. As a result, we can use decorations to change the look of a container. Here's an example of how to put a shadow on a container:


```
child: Container(
 width: 300.0,
 height: 300.0,
 decoration: BoxDecoration(
 color: Colors.purple,
 boxShadow: [
 BoxShadow(
 offset:
Offset.fromDirection(0.25*p
i, 10.0),
 blurRadius: 10.0,
 )
 ],
 ),
),
```

WITH A BOX SHADOW

Flutter – Container Box Shadow

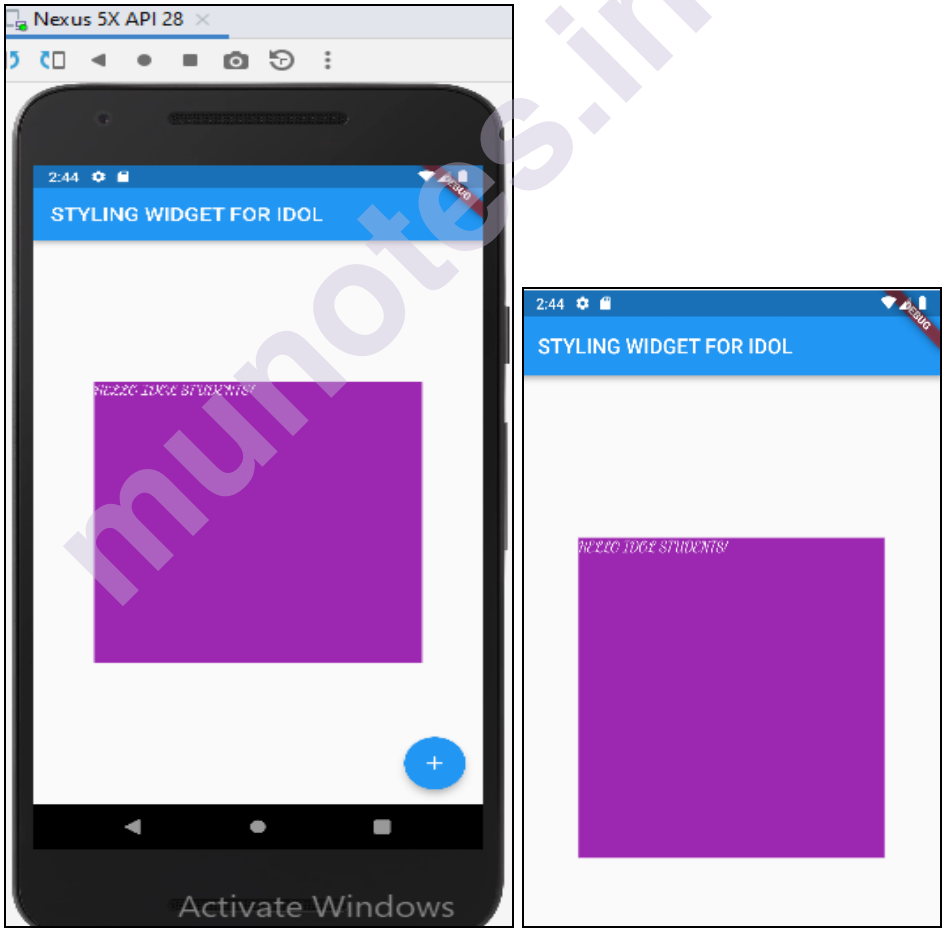
To set box shadow for Container widget, set its decoration property with a BoxDecoration object. Inside BoxDecoration, set boxShadow property with required array of BoxShadow objects as shown below.The main part of the code (WITH A BOX SHADOW) Screenshot is shared below for your reference:-



The main part of the code (WITHOUT A BOX SHADOW) Screenshot is shared below for your reference:-

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      // Here we take the value from the MyHomePage object that was created by
      // the App.build method, and use it to set our appbar title.
      title: Text(widget.title),
    ),  // AppBar
    body: Center(
      child: Container(
        width: 300.0,
        height: 300.0,
        decoration: BoxDecoration(
          color: Colors.purple,

        ),  // BoxDecoration
        child: Text(

          'HELLO IDOL STUDENTS!',
          style: TextStyle(color: Colors.white,fontFamily: 'dancing',),

        ),  // Text
      ),  // Container
    ),  // Center
```

Without A Box Decoration:



And this is a terrific example of the wordiness with Flutter. In the Web, this would have been done in 17 characters. But in Flutter we have to remember that boxShadow is an array of BoxShadows, each of which has an offset which takes a direction expressed in radians, a size expressed in pixels, and the blur radius is in pixels also. Blur radius may call for additional explanation. The blur radius is the distance over which the

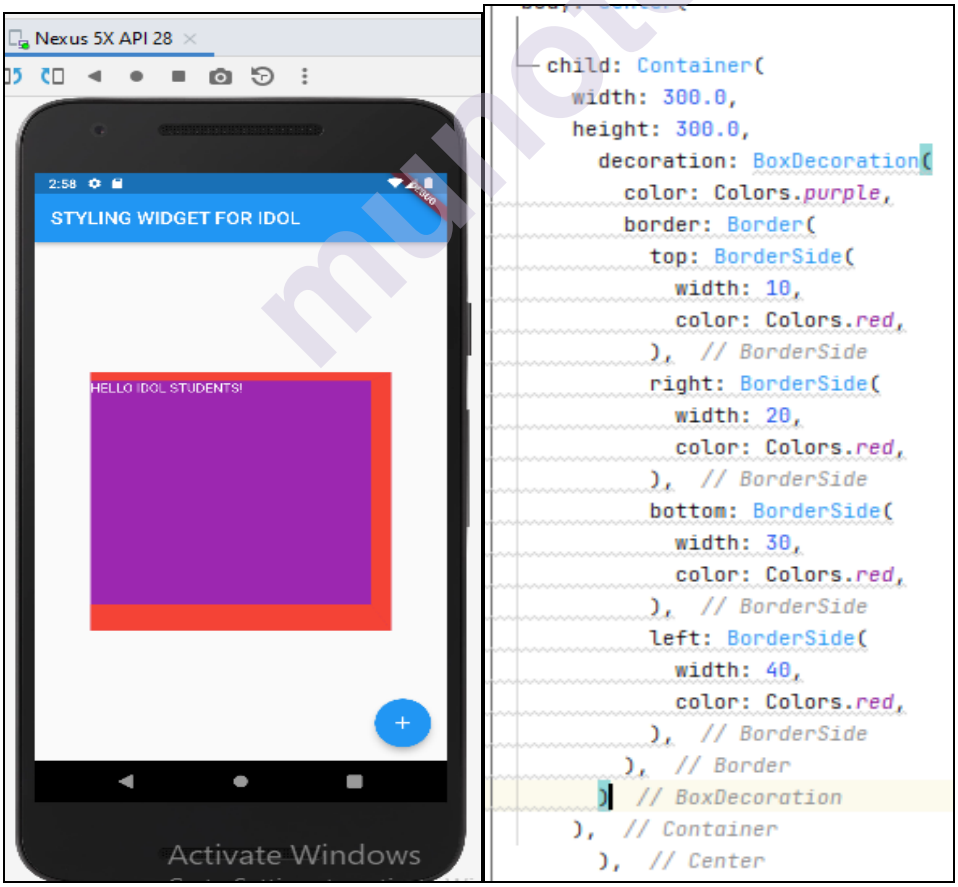
shadow dissipates. It's like putting a lampshade on a lamp. Without a shade, the light is harsh and shadows are crisp. With a lampshade, the light is softer and the shadows are also. The larger the blur radius, the softer the shadow.

8.6.5 Other Decorations:

There are a number of other decorations available. Let's look at the most useful ones, border, borderRadius, and BoxShape.

Border:

BorderRadius is a built-in widget in flutter. Its main functionality is to add a curve around the border-corner of a widget. There are in total of five ways in which we can use this widget, the first is by using BorderRadius.all, the radius for all the corners are the same here. The second way is by using BorderRadius.Circle, here we need to specify radius only once which would be a double value. The third way is by using BorderRadius.horizontal, here we can specify different border-radius for the left and the right side. The fourth way is by using BorderRadius.only, it can take a different radius for all four border corners. And the last way is by using BorderRadius.vertical, which can give a different radius to the upper and the lower portion of the border. Just like you used a BoxDecoration for shadows, you also use them to put a border on a container. Here's a red border with four different widths:-



BorderRadius:

BorderRadius is a built-in widget in flutter. Its main functionality is to add a curve around the border-corner of a widget. There are in total of five ways in which we can use this widget, the first is by using BorderRadius.all, the radius for all the corners are the same here. The second way is by using BorderRadius. Circle, here we need to specify radius only once which would be a double value. The third way is by using BorderRadius.horizontal, here we can specify different border-radius for the left and the right side. The fourth way is by using BorderRadius.only, it can take a different radius for all four border corners. And the last way is by using BorderRadius.vertical, which can give a different radius to the upper and the lower portion of the border.Rounded corners are a favorite look. You can make a Container rounded even if it doesn't have a border (Figure below). You do this with BorderRadius:

```
),  // AppBar
body: Center(

  child: Container(
    width: 300.0,
    height: 300.0,
      decoration: BoxDecoration(
        color: Colors.purple,
         borderRadius: BorderRadius.only(
        topLeft: Radius.circular(20.0),
        topRight: Radius.circular(60.0),

        ),  // BorderRadius.only
      ),  // BoxDecoration

    ),  // Container
      ),  // Center
```

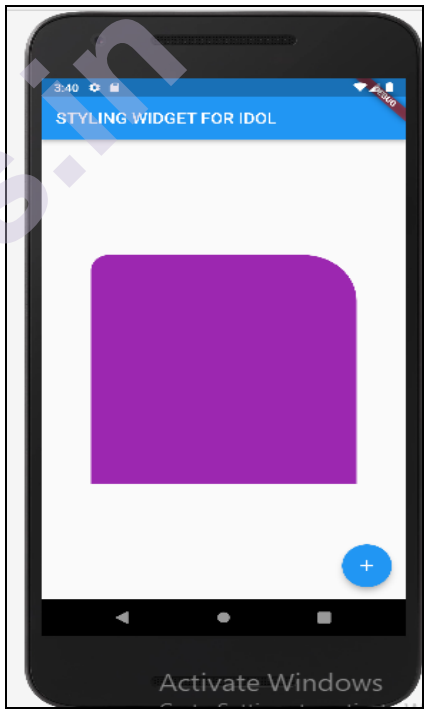


Figure :-BorderRadius on two corners

We only gave it a topLeft and a topRight radius, but there is also a bottomLeft and bottomRight property. And although we appreciate the flexibility, it is not typical to use it. We ordinarily specify all four the same.

```
return Scaffold(
  appBar: AppBar(
    // Here we take the value from the MyHomePage object that was cr
    // the App.build method, and use it to set our appbar title.
    title: Text(widget.title),
  ), // AppBar
  body: Center(

    child: Container(
      width: 300.0,
      height: 300.0,
        decoration: BoxDecoration(
          color: Colors.purple,
            borderRadius: BorderRadius.all(Radius.circular(20.0),


          ), // BorderRadius.all
        ), // BoxDecoration

      ), // Container
        ), // Center
```

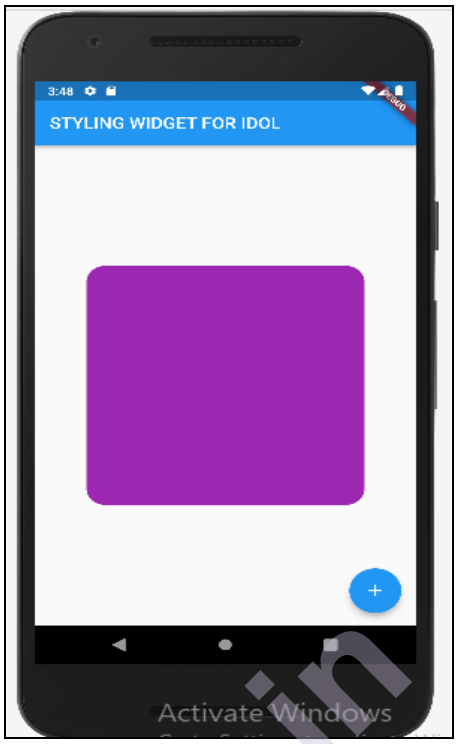


Figure BorderRadius on all four corners

BOXSHAPE:

Your containers don't have to always be rectangles. When you need it to be another shape, you can make it so with BoxShape or CustomPainter. BoxShape is much easier to use, but it only supports circles.

With CustomPainter, Flutter gives you access to low-level graphics painting. Best of all, painting in Flutter is fast and efficient. For example, Flutter's hot reload feature makes it easy to iterate quickly until you get exactly the look you want. CustomPainter is way more complex, but it allows infinite shapes. Flutter's CustomPaint widget enables you to pour your imagination onto the digital canvas. You can draw almost anything on the screen with this low-level painting API. It's similar to drawing something on paper, but instead of a pencil and paper, you have an API and a canvas on which to draw. It would be distracting to get too deep into the details of CustomPainter

```
@override
Widget build(BuildContext context) {

  return Scaffold(
    appBar: AppBar(
      // AppBar
      title: Text(widget.title),
    ),
    body: Center(
      child: Container(
        width: 300.0,
        height: 300.0,
          decoration: BoxDecoration(
            shape: BoxShape.circle,
            color: Colors.deepOrange,
            ),
          ),
        ),
```

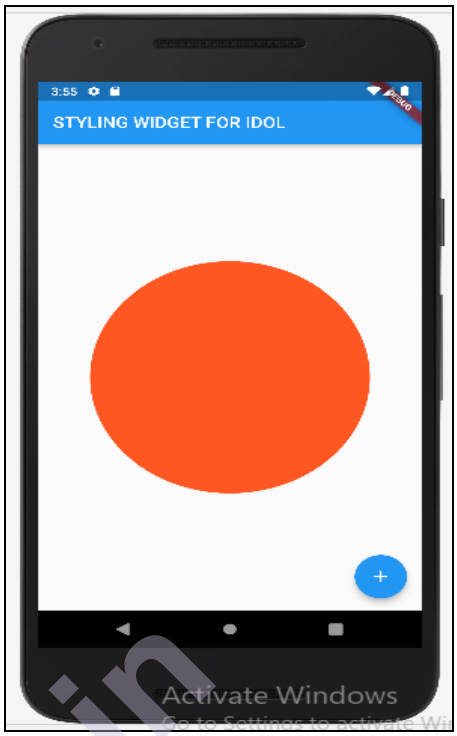


Figure: BoxShape. circle makes your Container round

8.6.6 HOW TO USE CustomPaint?

CustomPaint is a widget in Flutter that generates a canvas on which to draw during the paint phase. The canvas has a coordinate system that matches the coordinate system of the CustomPaint object. First, CustomPaint asks its painter to paint on the canvas. After it paints its child, the widget asks the foregroundPainter property to paint. The painters are restricted to a rectangular area that starts at the origin and encompasses a region of a specified size. If they venture outside this allotted space, there may not be enough memory to rasterize the painting commands.

```
import 'package:flutter/material.dart';

void main() {
  runApp(const MyApp());
}
class SupermanShieldPainter extends CustomPainter {
  @override
  void paint(Canvas canvas, Size size) {
    canvas.drawPath(Path()
      ..moveTo(25, 0)
      ..lineTo(125, 0)
      ..lineTo(150,25)
      ..lineTo(75, 125)
      ..lineTo(0,25)
      ..lineTo(25,0),
        Paint()
      ..style=PaintingStyle.fill..color = Colors.red
    );
  }
  @override
  bool shouldRepaint(SupermanShieldPainter oldDelegate) => false;
}
```

```
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Custom paint Demo'),
    ),  // AppBar
    body: Container(
      child: Center(
        child: CustomPaint(

      child: CustomPaint(
        size: Size(200, 200),
        painter: SupermanShieldPainter(),
      ),  // CustomPaint
    ),  // CustomPaint
    ),  // Center
    ),  // Container
    );  // Scaffold

  }
}
```

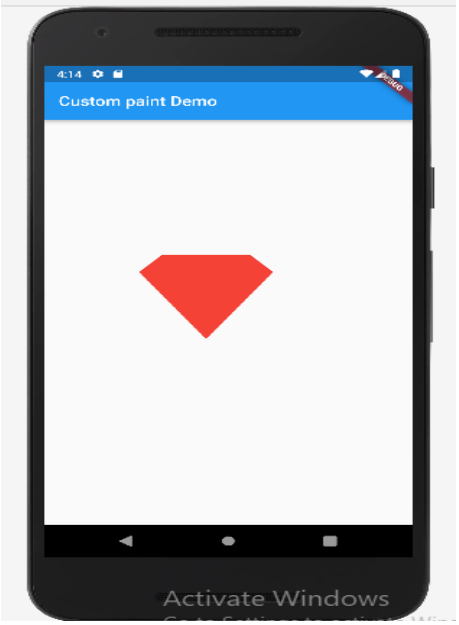


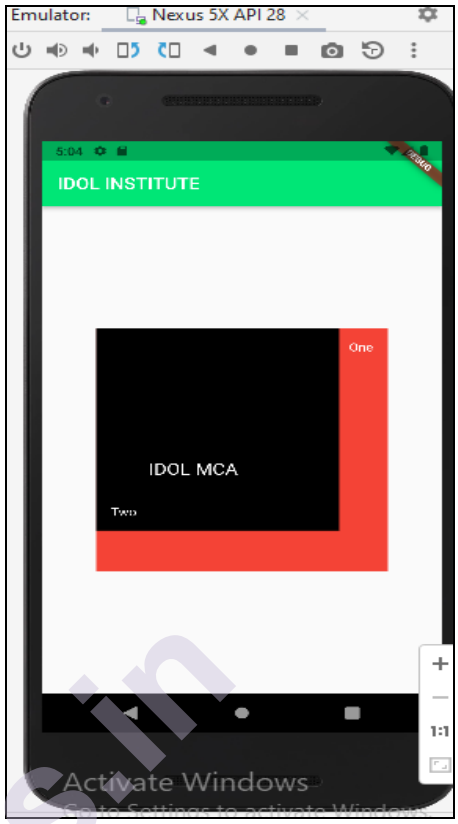
Figure: Using a CustomPainter

8.6.7 Stacking Widgets:

The stack is a widget in Flutter. It contains a list of widgets and places them on top of each other. And it places their children on top of each other like a stack of books. In other words, stack developers would overlap multiple widgets on one screen. As you can add different images or colors using containers in it.

If you ever want two or more things to occupy the same x- and y-position on the screen, reach for the Stack widget. The stack widget enables us to lay down a widget and then put another widget in front of it and another one in front of that one and so on. Obviously the ones added later will have a higher z-index, thereby “occulting” (hiding) the one behind it. Basically it allows you to ... well ... stack the inner widgets.Using a Stack, you can create some really cool layouts. In fact, Material Cards rely on Stacks a lot because they embrace background images with text on top of it.

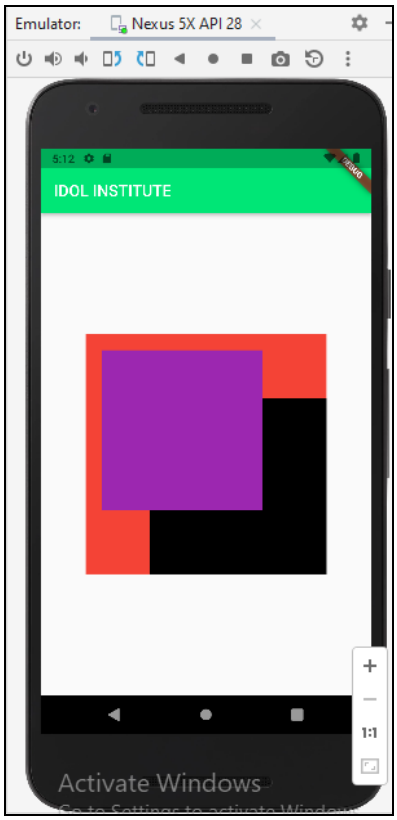
```
child: Stack(
  children: <Widget>[
    Container(
      width: 300,
      height: 300,
      color: Colors.red,
      padding: EdgeInsets.all(15.0),
      alignment: Alignment.topRight,
      child: Text(
        'One',
        style: TextStyle(color: Colors.white),
      ), //Text  // Text
    ), //Container  // Container
    Container(
      width: 250,
      height: 250,
      color: Colors.black,
      padding: EdgeInsets.all(15.0),
      alignment: Alignment.bottomLeft,
      child: Text(
        'Two',
        style: TextStyle(color: Colors.white),
      ), //Text  // Text
    ), //Container  // Container
    Container(
      height: 200,
      width: 200,
      padding: EdgeInsets.all(15.0),
      alignment: Alignment.bottomCenter,
      decoration: BoxDecoration(
        image: DecorationImage(
            image: NetworkImage(
                "idol.jpg") //NetworkImage  // NetworkImage
        ), //DecorationImage  // DecorationImage
      ), //BoxDecoration  // BoxDecoration
      child: Text(
        "IDOL MCA",
```



8.6.7 Positioned Widget:

In our preceding example, the texts laid out decently because a Column centers its children and the Expanded pushed the Texts to the top and bottom.

```
void main() {
  runApp(MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        title: Text('IDOL INSTITUTE'),
        backgroundColor: Colors.greenAccent[400],
      ), //AppBar  // AppBar
      body: Center(
        child: SizedBox(
          width: 300,
          height: 300,
          child: Center(
            child: Stack(
              fit: StackFit.expand,
              clipBehavior: Clip.antiAliasWithSaveLayer,

              children: <Widget>[
                Container(
                  width: 300,
                  height: 300,
                  color: Colors.red,
                ), //Container  // Container
                Positioned(
                  top: 80,
                  left: 80,
                  child: Container(
                    width: 250,
                    height: 250,
                    color: Colors.black,
                  ),  // Container
                ), //Container  // Positioned
                Positioned(
                  left: 20,
                  top: 20,
                  child: Container(
                    height: 200,
                    width: 200,
                    color: Colors.purple,
```



In this app, we have wrapped the purple and the black Container with a Positioned widget, so these children widgets are now positioned widgets. In the Stack widget, the fit property is set to StackFit.expand which will force all its children widgets to take maximum space available to them. The clip property is set to antiAliasWithSaveLayer, which avoid any bleeding edges. And the overflow is set to visible, to make the overflowing parts visible. Now, that we have wrapped containers with the Positioned widget we need to specify their position. For the black Container, the top and the left properties are set to 80 each, which makes it overflow out of the SizedBox and the red Container. But as the overflow is set to visible, so we are able to see the overflowing portion of the black Container. And for the purple Container, the top and left are set to 20 each, which makes it a bit offset when compared to the first example.

8.6.8 Card Widget:

- Card is a build-in widget in flutter which derives its design from Google's Material Design Library. The functionality of this widget on screen is, it is bland space or panel with round corners and a slight elevation on the lower side. It comes with many properties like color, shape, shadow color, etc which lets developers customize it the way they like. You may have noticed that we used a Card widget in our preceding example.

- A Card feels like the right thing to do in this situation, but it is by no means required. A Flutter Card widget was created to implement the Material Design look and feel, having properties like color for the background color, elevation for a drop shadow size, borderOnForeground for the border, and margin for spacing around it. A card is a sheet used to represent the information related to each other, such as an album, a geographical location, contact details, etc.

- A card in Flutter is in rounded corner shape and has a shadow. We mainly used it to store the content and action of a single object. In this article, we are going to learn how to create a card widget in Flutter. We will also learn how to customize the card widget. Card creation in Flutter is very simple. We just need to call the card constructor and then pass a widget as child property for displaying the content and action inside the card.Granted, all of those could also be accomplished with a Container. But if you want to do it with a standard look and feel, a Card makes it easy:

```
Card(
 elevation: 20.0,
 child: Text("This is text in a card",
 style: Theme.of(context).textTheme.display3),
),
```

8.6.9 Themes:

Themes are the preset packages that contain the graphical appearances to our website or mobile app screen. It makes the user interface more attractive. We mainly use themes for sharing the colors and fonts styles throughout the app. In mobile development, it becomes mandatory to add the Light and Dark theme for our app. Today's most people prefer the dark version of the theme over the light version theme because it makes them more comfortable to their eyes and increases battery life. In Flutter, we can either use Theme widgets that contain the colors and font styles for a specific area of the application or define app-wide themes. The app-wide themes are also Theme widgets, which are created in the root of our app under the MaterialApp widget.

After defining a theme, we can use it in any widget wherever we need it in the app. Material widgets in Flutter can also use our Theme to set the font styles and background colors for AppBars, Buttons, Buttons, Checkboxes, and many more. A consistent use of styling creates a pleasant app that exudes quality. And a great way of staying consistent is simply to stick to a Theme. A Theme in Flutter is a grouping of styles in logically-defined groups that can be applied together. This way, not only does your app have a consistent look and feel throughout, but you can easily change the theme in one (1!) place, MaterialApp, and it propagates to all children:

```
class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
 return MaterialApp(
 title: 'Ch 8 - Styling Text',
 theme: ThemeData(primarySwatch: Colors.yellow),
 home: HomeWidget(),
 );
 }
}
```

Applying Theme Properties:

Remember, you don't have to do anything in order to use the themes on almost every widget in your app. Nothing. In fact, that's the whole idea of setting a Theme and some of the underlying properties; your app just absorbs them when they're rendered. The theme becomes their default look and feel. But what if you want to overtly apply a theme? For instance, you have a Text widget at the top of a scene and you want it to function as a page heading. Or maybe below that somewhere you want a secondlevel heading. Perhaps a sub-heading somewhere. How do you tell these special Text widgets that they are supposed to be drawn in a special way? Remember the style property? Text widgets have a style property that takes a TextStyle object. But you can access well-known text styles from the theme like so:

```
Text('title', style: Theme.of(context).textTheme.title),
Text('subtitle', style: Theme.of(context).textTheme.subtitle),
Text('headline', style:
Theme.of(context).textTheme.headline),
Text('subhead', style:
Theme.of(context).textTheme.subhead),
```

You have the Material textThemes in Table given below to choose from:

Table: Material theme text styles

Text theme Name	Description
body1	Most of the text you see. This will be the default style if you don't explicitly apply one.
body2	Slightly thicker body text.
Button	The default font on buttons, typically all caps and spread out a bit
Caption	For photo captions.
display1	The biggest headline (aka headline1)
display2	The 2nd biggest headline (aka headline2)
display3	The 3rd biggest headline (aka headline2)
display4	The smallest headline (aka headline4)
Headline	Your go-to style for headlines (aka headline5)
Subhead	For subheadings. Usually right below a heading
Title	(aka headline6)
Subtitle	For sub-subheadings. Usually right below a title
Overline	Rarely used. For introducing a headline

Figure below shows what they all look like

Figure:- How the Material text styles look on a device

8.7 MANAGING STATE

State is widget data whose change requires a re-render. - Rap Payne ;-).

StatelessWidgets might have data, but that data either doesn't change or doesn't change how the screen looks while the widget is alive. Sure, it may change when Flutter destroys and recreates the widget, but that doesn't count. To be state, it must change while the widget is active, and that change requires a re-render in order to stay current. Flutter gives us certain widgets that are stateful out of the box

- AppBar
- BottomNavigationBar
- Checkbox
- DefaultTabController
- Dismissible
- DrawerController
- DropdownButton
- EditableText
- Form
- FormField
- GlowingOverscrollIndicator
- Image
- InputDecorator
- MonthPicker
- Navigator
- ProgressIndicator
- Radio
- RefreshIndicator
- Scaffold
- Scrollbar
- Slider
- Switch
- TextField
- YearPicker

. and many more. These all have internal data that must be maintained and monitored so that as the data changes, we re-render the widget to reflect the said change. Let's take a simple example: a TextField widget. Yes, we're talking about the built-in widget that's kind of like a textbox on the Web; the user can type characters into it. You realize of course that as the user types, the widget is keeping track of and displaying the stuff that they're typing. That, my friend, is state.

8.7.1 What Goes In A Statefulwidget?

Here's the shape of a StatefulWidget:

```
class Foo extends StatefulWidget {
 @override
 _FooState createState() => _FooState();
}
class _FooState extends State<Foo> {
 //Private variables here are considered the 'state'
 @override
 Widget build(BuildContext context) {
 return someWidget;
```

A stateful widget looks pretty complex, but once you get used to its structure, it becomes second nature. We traditionally write it in one Dart file, but it always consists of two classes: the widget class and a state

class. The widget class inherits from StatefulWidget and is public because it is the thing that will be placed in other widgets. The state class is always private because the current widget is the only thing that will ever see this class. The state class is responsible to...

1. Define and maintain the state data.
2. Define the build() method – It knows how to draw the widget on screen.
3. Define any callback functions needed for data gathering or event handling.

What does that leave for the widget class? Not much. The widget class just kind of gets out of the way.

So then why separate them? There are two reasons. First, the single responsibility principle (the SRP) suggests that we should have one thing responsible for drawing the widget and another thing responsible for dealing with data. That's just good software design. Other frameworks suggest that you separate UI from state management, but most don't enforce it. Flutter does.

Second is performance. Redrawing takes time. Recalculating state takes time. When we separate them like this, we are giving the processor a chance to handle these two things independently. Sometimes a redraw doesn't need to happen just because state changes. So we save the cycles of redrawing. Also, when we redraw, Flutter creates and draws a whole new widget.

The old widget in memory is no longer needed so it is dereferenced and eventually garbage collected. That's awesome but state is still needed. If Flutter retains that old state object, it can be reused instead of being garbage collected and recreated. By separating these objects, Flutter decouples them so they can each be handled in its own most efficient way.

8.7.3 The Most Important Rule About State!

When you change any state value, you should do it ...

1. In the state class.
2. Inside a function call to setState():

```
setState(() {
 // Make all changes to state variables here...
 _value = 42; // <-- ... Like this
});
```

- setState() takes a function which is run ... uh ... soon. The Flutter subsystem batches changes and runs them all at a time that it decides is optimal. This is extremely efficient because, among other reasons, it will reduce the number of screen redraws.

- setState() not only sets the variables in the most efficient and controlled way, but it always forces a re-render of this widget to occur. It invokes build() behind the scenes. The end result: When you change a value, the widget redraws itself and your user sees the new version. Note that if this widget has subwidgets inside of it (aka inner widgets), they'll be in the build() method, so a call to setState() actually redraws everything in this widget including all of its subtrees.

8.7.3 Passing State Down:

Technically, you can't pass state from a host widget into an inner widget because state only exists within a widget. But we definitely want to pass data down. That data may be stateful data in the host widget, and it may be moved to state in the inner widget. Flutter provides us an object called widget which represents the StatefulWidget. In other words, if there is a variable called "x" in the StatefulWidget, it is visible in the State class as "widget.x":

```
class Foo extends StatefulWidget {
 final String passedIn;
 // Value passed in from its host
 ColorValueChanger({Key key, this.passedIn}) :
super(key: key);
 _FooState createState() => new _FooState();
}
class _FooState extends State<Foo> {
 @override
 Widget build(BuildContext context) {
 return Text(widget.passedIn,);
 }
}
```


8.7.4 Lifting State Back Up:

Flutter has one-way data flow. Period. Data can only flow down from a host widget to an inner widget. We've been doing this for, what, about 200 pages now? But sometimes we need data to flow from an inner back up to a host.For instance, let's say we have a Login.dart widget with username password textfields and a submit button.

We'd place this Login in other widgets provided that the user is not already logged in. The business logic to log in must be in the Login widget itself. But when they successfully log in, we really need to let the host widget – or even all widgets – know they are now authenticated. The token needs to be passed back up.

Don't pass the data up. Pass the handler method down! In Dart (as in JavaScript), functions are first-class objects. This means that their references can be passed around like data. This also means that you can pass a function from a host widget down into inner widgets. Now that the

inner widget has a handle to this function, it can call it as if it were its own. But of course when the inner widget calls it, if it passes a value into that function, the value is seen in the host where the function is defined. This technique is called lifting the state up.

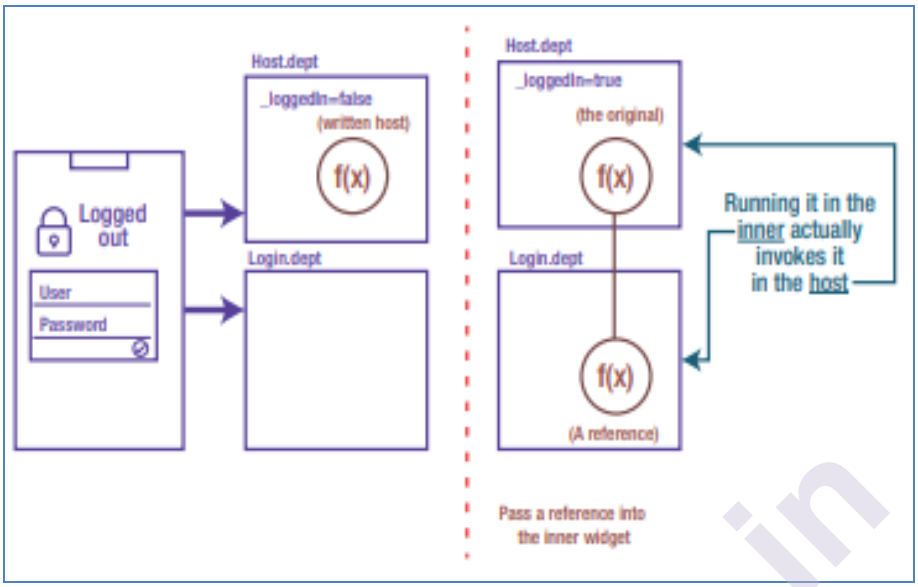


Figure: Lifting state up

8.7.5 When Should We Use State?

The very best way to avoid complex state is to avoid having state at all. Just about every expert agrees that if you can avoid state altogether, do. But it can be confusing as to when you need state and when you don't. For example, the label on our color picker is data within the component. Should that be state? No, of course not; it doesn't change. How about a loop counter on a for loop? Nope; it never affects anything in the build() method, so it doesn't need to be put in a setState(). State can sometimes be simplified or eliminated.

8.7.6 Advanced State Management:

What we've looked at in this module would work as advertised even when the widget tree gets infinitely deep. But please realize that as your app gets bigger and bigger, state management become more and more complex. When it gets too complex, you may be better served by using a more advanced state management pattern. These patterns are not always easy to learn, but at some point in your app's growth, they become worth the effort to master.

- **Inheritedwidget:**

This is a relatively simple solution, maybe too simple for most needs. InheritedWidget is a built-in Flutter widget. Essentially it creates a small set of global variables that are made available in a controlled way to all descendants in its tree. Several of the other methods (ScopedModel, Provider, Redux) are wrappers around InheritedWidget.

Pros: No library to install or keep track of.

Cons: There is some duplication between the InheritedWidget and the underlying StatefulWidget. That's a shame. Also the entire subtree is rerendered when any data changes

- **BLoC:**

BLoC is an acronym for Business Logic Component, and it's less of a solution than a design pattern. BLoC was created by Google so naturally; it was embraced by the Flutter community.

Pros: Lots of folks in the community can and will help you. It is a solid, well-vetted pattern.

Cons: You have to write everything yourself; it's neither built-in nor a library. It can be hard to know where to inject a BLoC.

- **ScopedModel:**

ScopedModel is a library "shamelessly borrowed" from the Fuchsia4 codebase by Brian Egan. ScopedModel creates data models with the ability to register listeners. Each model notifies its listeners when the data has changed so they can update. Clever design.

Pros: Does its job of separating presentation and data very well.

Cons: There is talk of ScopedModel being combined with Google's flutter_provide which seems to be a more modern and simpler approach to state management.

- **Hooks:**

An implementation of React hooks by Rémi Rousselet of Paris called flutter_hooks. You no longer use StatefulWidgets at all. Instead you inherit from a HookWidget which is stateless (therefore simpler) but allows you to create and access custom functions that read and write state values. It even comes with some pre-baked hooks that you don't have to write.

Pros: Greatly simplifies your formerly stateless widgets.

Cons: Learning curve. It isn't obvious how they work and the rules for use unless you're already familiar with React hooks.

- **Provider:**

At the time of this writing, there's some confusion between Provider, also written by Rémi Rousselet, and a similarly named one written by Filip Hracek and the good folks at Google. Filip freely admits that Rémi's package "is more feature-full.

Pros: A very robust and capable package that is comparatively

simple to use. In the near future, I expect this to become the go-to state management library for developers who don't already have a leaning toward Redux and/or hooks because of prior experience with the React ecosystem.

Cons: Not (yet) as popular as some of the others.

Note that there is a lot of confusion between this package and flutterprovide created by Google because of the naming. The latter one, taken from the Fuchsia codebase and open-sourced, may be combined with ScopedModel and deprecated.

- **Redux:**

Like a few others on this list, Redux is a library borrowed from other technologies and ported to Dart. Redux has a deep history coming from the world of React via Facebook. There are several implementations, but the most popular is here: flutter_redux. Also written by the prolific Brian Egan.

Cons: Very steep learning curve.

Pros: Very performant. Very scalable. Many React developers already know Redux.js. The learning curve flattens significantly for them.

These packages all solve the same problem in different ways, some similarly and others using wildly different strategies. No one has any expectations that you'll have anything more than an awareness that there are tools out there.

8.8 SUMMARY

- JSON serialization is a very mundane task. But if we want our apps to work correctly, it's very important that we do it right and pay attention to details: use jsonEncode() and jsonDecode() from 'dart:convert' to serialize JSON data.
- JSON is an open-standard format used on the web and in mobile clients, especially with REST APIs. In mobile apps, JSON code is usually parsed into the model objects that your app will work with.
- We have also understood that options for styling things in Flutter are near infinite. Flutter styling resembles what you may have seen in CSS, but is by no means the same. First, it is more verbose. And second, it doesn't inherit. Some people may resent these characteristics, but others will like the cleanness that it creates.
- We have also seen the styling options that Flutter provides, especially when you think about how they're organized in Themes so we can present a consistent, professional look and feel throughout our app.
- We have also seen that there are clear times when a widget needs to maintain its own status via the data that is contained within it. When

we do, we call this state and we call the widget a stateful widget. Stateful widgets are by their nature more complex than stateless widgets so we try to avoid them if we can. Additionally the more stateful widgets we have, the more state needs to be passed around between the widgets. This can get very complex very quickly so we look to tools and techniques like BLoC, Redux, ScopedModel, and Provider to tame state.

8.9 QUESTIONS

1. Explain JSON FORMAT?
2. Write a note on Database Classes to write, read and serialize JSON?
3. Explain in brief form Validation?
4. Explain STACKING Widgets?
5. Write in your own words about THEMES in flutter?
6. Explain ADVANCED STATE MANAGEMENT?
7. What are the most important rules about STATE?

8.10 CHAPTER END EXCERCISE

1. Create a flutter project and add two TextField widgets customized by two different decorations.
2. Create a flutter project using decoration property of Container (apply multiple properties to the container).
3. Create a flutter project with StatefulWidget.

8.11 REFERENCES

1. Alessandro Biessek Flutter for Beginners: An Introductory Guide to Building Cross-platform Mobile Applications with Flutter and Dart 2 Packt Publishing Ltd.ISBN. 9781788990523.
2. Rap Payne Beginning App Development with Flutter: Create Cross-Platform Mobile Apps Apress, ISBN 978-1-4842-5181-2.
3. Marco L. Napoli Beginning Flutter: A Hands On Guide to App Development John Wiley &Sons, ISBN:- 1119550823, 9781119550822.
4. https://blog.logrocket.com/flutter-form-validation-complete-guide/.
5. https://flutter.dev/

*****

9

CASE STUDY ON IOS APP DEVELOPMENT

Introduction swift programming concept, objective c.

Unit Structure

9.1 OBJECTIVES

To extend the knowledge and skill in **iOS app development** in creating more complex and capable mobile apps and work with data from a server and explore new **iOS** APIs.

9.2 INTRODUCTION

Swift is a general-purpose, multi-paradigm, object-oriented, functional, imperative and block structured language. It is built using a modern approach to safety, software design patterns by Apple Inc.**.** It is a programming language for iOS application, macOS application, watchOS application, tvOS application. Swift is famously Apple's products language, but it is not an Apple-only language. We

can use it on several other platforms. It is open source, so porting the language to other platforms does not require any permission or licensing, and you can find Swift projects to create Web servers and APIs.

9.3 EVOLUTION OF SWIFT PROGRAMMING LANGUAGE

Swift language was developed by 'Chris Lattner'with an aim to resolve difficulties existed in Objective C. It was introduced at Apple's 2014 Worldwide Developers Conference (WWDC) with version Swift 1.0. Soon, It underwent an upgrade to version 1.2 during 2014. Swift 2.0 was introduced at WWDC 2015. Initially, version 2.2 was made open-source software under the Apache License 2.0 on December 3, 2015, for Apple and Linux platforms. It is primarily designed to work with Apple's Cocoa and Cocoa Touch frameworks and the large body of existing Objective-C code written for Apple products. Swift language has gone through major changes since its release from version names 1.0, 2.0, 3.0 and 4.0 and later. The current released version is Swift 4.2 with a beta version of Swift 4.3 and Xcode 10.

Sample program:

```
// Basic Swift Program

import UIKit


var str1 = "Hello World !!!"

print (str1)
```

Output:

```
Hello World !!!
```

9.4 ADVANTAGES

Swift is in the main an Object-Oriented language with some interesting features inspired by functional-style languages. Here are some of the highlights:

- c-like syntax.
- Type safe. It's strongly, staticly typed and checks types at compile time. It uses type inference so you don't need to declare types for everything.
- Supports lots of data-types natively, including lists, dictionaries, tuples, enumerations, structs and classes.
- Uses automatic reference counting for memory management.

- Strong support for closures. Functions are first-class objects.
- Supports interactive programming and scripting using repl and playgrounds.
- Compiles to fast native code.
- Object-Oriented: classes, protocols (similar to java interfaces), extensions (similar to ruby mixins).
- Generics for generic programming.
- Some limited support for pattern matching.
- Functional. There's enough functional goodness in Swift that you can eschew OO and program in a functional style if you want to.

9.5 SWIFT PROGRAMMING LANGUAGE

The features of Swift are designed to work together to create a powerful language. Additional features of Swift include:

- Closures unified with function pointers
- Tuples and multiple return values
- Generics
- Concise and fast iteration over a range or collection
- Structs that support methods, extensions, and protocols
- Functional programming patterns, e.g., map and filter
- Powerful error handling built-in
- Advanced control flow with do, guard, defer, and repeat keywords

9.6 MEMORY MANAGEMENT

Swift uses Automatic Reference Counting (ARC) to manage memory. Earlier, Apple used to require manual memory management in Objective-C, but after introducing ARC in 2011 memory allocation and de-allocation became easier.

Swift is managed as a collection of projects, each with its repositories. The current list of projects includes:

- The Swift compiler command-line tool
- The standard library bundled as part of the language
- Core libraries that provide higher-level functionality

- The Swift REPL included LLDB debugger
- Xcode playground support to enable playgrounds in Xcode.
- The Swift package manager for distributing and building Swift source code

9.7 VARIABLE

Variables let us assign a value to a label, and are defined using the var keyword:

var name = "Roger"

var age = 8

Once a variable is defined, we can change its value:

age = 9

Variables that you do not want to change can be defined as constants, using the let keyword:

let name = "Roger"

let age = 8

Once a variable is defined, it is bound to that type, and you cannot assign to it a different type, unless you explicitly convert it.

You can't do this:

var age = 8

age = "nine"

9.8 OBJECTS

In Swift, everything is an object. Even the 8 value we assigned to the age variable is an object.

Swift has 3 different **object types**, which we'll see more in details later on: **classes**, **structs** and **enums**.

Those are very different, but they have one thing in common: to object type, we can **add methods**, and to any value, of any object type, we can **send messages**.

In some languages, objects are a special type. But in Swift, everything is an object and this leads to one particular feature: every value can receive messages.

Each type can have multiple functions associated to it, which we call **methods**.

For example, talking about the 8 number value, we can call its isMultiple method, to check if the number is a multiple of another number:

9.9 SWIFT OPERATORS

Then we can divide operators based on the kind of operation they perform:

- assignment operator
- arithmetic operators
- compound assignment operators
- comparison operators
- range operators
- logical operators

Additionally, some more advanced ones, including nil-coalescing, ternary conditional, overflow, bitwise and pointwise operators.

9.10 CONDITIONALS

if conditionals

if statements are the most popular way to perform a conditional check. We use the if keyword followed by a boolean expression, followed by a block containing code that is ran if the condition is true:

```
let condition = true
if condition == true {
   // code executed if the condition is true
}
```

An else block is executed if the condition is false:

```
let condition = true
if condition == true {
   // code executed if the condition is true
} else {
   // code executed if the condition is false
}
```

You can optionally wrap the condition validation into parentheses if you prefer:

```
if (condition == true) {
   // ...
```

```
}
```

switch case conditionals

Switch statements are a handy way to create a conditional with multiple options:

```
var name = "Roger"

switch name {
case "Roger":
   print("Hello, mr. Roger!")
default:
   print("Hello, \(name)")
}
```

When the code of a case ends, the switch exits automatically.

A switch in Swift needs to cover all cases. If the tag, name in this case, is a string that can have any value, we need to add a default case, mandatory.

Otherwise with an enumeration, you can simply list all the options:

```
enum Animal {
   case dog
   case cat
}

var animal: Animal = .dog

switch animal {
case .dog:
   print("Hello, dog!")
case .cat:
   print("Hello, cat!")
}
```

A case can be a Range:

```
var age = 20

switch age {
case 0..<18:
```

```
    print("You can't drive!!")
default:
    print("You can drive")
}
```

Ternary conditionals:

The ternary conditional operator is a shorter version of an if expression. It allows us to execute an expression if a condition is true, and another expression if the condition is false.

Here is the syntax:

`condition` ? `value if true` : `value if false`

Example:

```
let num1 = 1
let num2 = 2


let smallerNumber = num1 < num2 ? num1 : num2


// smallerNumber == 1
```

9.11 LOOPS

For-in loops

For-in loops can be used to iterate a specific amount of times, using a range operator:

```
for index in 0...3 {
  //iterate 4 times, `index` is: 0, 1, 2, 3
}
```

You can iterate over the elements of an array or set:

```
let list = ["a", "b", "c"]
for item in list {
  // `item` contains the element value
}
```

And on the elements of a dictionary:

```
let list = ["a": 1, "b": 2, "c": 2]
for (key, value) in list {
  // `key` contains the item key
```

```
    // `value` contains the item value
}
```

while loops:

A while loop can be used to iterate on anything, and will run while the condition while:

```
while [condition] {
    //statements...
}
```

The condition is checked at the start, before the loop block is executed.

Example:

```
var item = 0
while item <= 3 { //repeats 3 times
    print(item)
    item += 1
}
```

repeat-while loops:

A repeat-while in Swift is similar to the while loop, but in this case the condition is checked at the end, after the loop block, so the loop block is executed at least once. Then the condition is checked, and if it is evaluated as true, the loop block is repeated:

```
repeat {
    //statements...
} while [condition]
```

Example:

```
var item = 0
repeat { //repeats 3 times
    print(item)
    item += 1
} while item < 3
```

Loop control transfer statements

Swift provides you 2 statements that you can use to control the flow inside a loop: continue and break

continue is used to stop the current iteration, and run the next iteration of the loop.

break ends the loop, not executing any other iteration.

9.12 OPTIONALS

Constants need to be initialized when declaring them, and variables need to be initialized before use. So where's the Objective-C nil equivalent? Swift introduces **optional values**. Optional values can have a value or be nil. If you take a look at the following code, you will notice that x was assigned an Optional value of 2014. This means that Swift compiler was aware that x might also be nil.

```
var s = "2014"
var x = s.toInt()
print(x)    // Optional(2014)
```

If you make a change in this code and assign the value "abc" to s, which cannot be converted to an Integer, you will notice that x is now anil.

```
var s = "abc"
var x = s.toInt()
print(x)    // nil
```

The return type of toInt() function is Int?, which is an **optional Int**. Let's try to call a standard function on x:

```
var x = "2014".toInt()
print(x.successor()) // error
```

The compiler signals an error, since x is an **optional and could potentially be nil**. We have to test x first, and make sure that the successor function is invoked on a real number, and not a nil value:

```
var x = "2014".toInt()
if x != nil
{
  print(x!.successor())    // 2015
}
```

Note that we have to unwrap x by appending an exclamation mark (!). When we are sure that x contains a value, we can access it. Otherwise we will get a runtime error. We can also do what Swift calls **optional binding**, converting the optional into a non-optional variable

```
let x = "123".toInt()
if let y = x
{
```

```
    print(y)
}
```

The code in the if statement will only execute if x has a value, and assign it to y. Note that we don't need to unwrap y, it's type is not optional since we know x is not nil.

Check Apple's Swift tutorial to read more details on optionals and nice features like optional chaining

9.13 STRING INTERPOLATION

In Objective-C formatting strings is usually done with the stringWithFormat: method:

```
NSString *user = @"Gabriel";

int days = 3;

NSString *s = [NSString stringWithFormat:@"posted by %@ (%d days ago)", user, days];
```

Swift has a feature called **string interpolation** to do the same, but it is more compact and easier to read:

```
let user = "Gabriel"

let days = 3

let s = "posted by \(user) \(days) ago"
```

You can also use expressions:

```
let width = 2

let height = 3

let s = "Area for square with sides \(width) and \(height) is \(width*height)"
```

To learn more about Swift's string interpolation and other new features, go here.

9.14 FUNCTIONS

Function definition in Swift is different from C. A sample function definition is below:

```
func someFunction(s:String, i: Int) -> Bool
{
    ...    // code
}
```

Swift functions are first-class types. This means that you can assign functions to variables, pass them as parameters to other functions, or make them return types:

```
func stringLength(s:String) -> Int
{
   return countElements(s)
}

func stringValue(s:String) -> Int
{
   if let x = s.toInt()
   {
      return x
   }
   return 0
}

func doSomething(f:String -> Int, s:String) -> Int
{
   return f(s).successor()
}

let f1 = stringLength
let f2 = stringValue

doSomething(f1, "123")   // 4
doSomething(f2, "123")   // 124
```

Again, Swift infers the types of f1 and f2 (String -> Int), although we could have defined them explicitly:

```
let f1:String -> Int = stringLength
```

Functions can also return other functions:

```
func compareGreaterThan(a: Int, b: Int) -> Bool
{
   return a > b
}
```

```
func compareLessThan(a: Int, b: Int) -> Bool
{
   return a < b
}
func comparator(greaterThan:Bool) -> (Int, Int) -> Bool
{
   if greaterThan
   {
      return compareGreaterThan
   }
   else
   {
      return compareLessThan
   }
}

let f = comparator(true)
println(f(5, 9))
```

A guide to functions in Swift can be found here.

9.15 ENUMERATIONS

Enumerations in Swift are much more powerful than in Objective-C. As Swift structs, they can have methods, and are passed by value:

```
enum MobileDevice : String
{
   case iPhone = "iPhone", Android = "Android", WP8 = "Windows Phone8", BB = "BlackBerry"

   func name() -> String
   {
      return self.toRaw()
   }
}
let m = MobileDevice.Android
```

```
print(m.name())    // "Android"
```

Unlike Objective-C, Swift enumerations can assign Strings, characters or floats as values for each member, besides integers. The convenient toRaw() method returns the value assigned to each member.

Enumerations can also be parameterized:

```
enum Location
{
   case Address(street:String, city:String)
   case LatLon(lat:Float, lon:Float)

   func description() -> String
   {
      switch self
      {
      case let .Address(street, city):
         return street + ", " + city
      case let .LatLon(lat, lon):
         return "(\(lat), \(lon))"
      }
   }
}

let loc1 = Location.Address(street: "2070 Fell St", city: "San Francisco")

let loc2 = Location.LatLon(lat: 23.117, lon: 45.899)

print(loc1.description())       // "2070 Fell St, San Francisco"

print(loc2.description())       // "(23.117, 45.988)"
```

More information about enumerations is available here.

9.16 TUPLES

Tuples group multiple values into a single compound value. The values within a tuple can be of any type and do not have to be of the same type as each other.

```
let person = ("Gabriel", "Kirkpatrick")

print(person.0) // Gabriel
```

You can also name the individual tuple elements:

```
let person = (first: "Gabriel", last: "Kirkpatrick")

print(person.first)
```

Tuples are extremely convenient as return types for functions that need to return more than one value:

```
func intDivision(a: Int, b: Int) -> (quotient: Int, remainder: Int)
{
   return (a/b, a%b)
}
print(intDivision(11, 3))    // (3, 2)
let result = intDivision(15, 4)
print(result.remainder)    // 3
```

Unlike in Objective-C, Swift supports pattern matching in a switch statement:

```
let complex = (2.0, 1.1)    // real and imaginary parts

switch complex
{
   case (0, 0):
      println("Number is zero")
   case (_, 0):
      println("Number is real")
   default:
      println("Number is imaginary")
}
```

In the second case we don't care about the real part of the number, so we use an _ to match anything. You can also check for additional conditions in each case. For that we need to bind the pattern values to constants:

```
let complex = (2.0, 1.1)


switch complex
{
   case (0, 0):
      println("Number is zero")
   case (let a, 0) where a > 0:
```

```
    println("Number is real and positive")
  case (let a, 0) where a < 0:
    println("Number is real and negative")
  case (0, let b) where b != 0:
    println("Number has only imaginary part")
  case let (a, b):
    println("Number is imaginary with distance \(a*a + b*b)")
}
```

Note how we need to bind only the values we are going to use in the comparison or in the case statement.

To read more about tuples, go here.

9.17 CLASSES AND STRUCTURES

Unlike Objective-C, Swift does not require you to create separate interface and implementation files for custom classes and structures. As you learn Swift, you will learn to define a class or a structure in a single file, and the external interface to that class or structure is automatically made available for other code to use.

Defining Classes:

Class definitions are very simple:

```
class Bottle
{
  var volume: Int = 1000

  func description() -> String
  {
    return "This bottle has \(volume) ml"
  }
}
let b = Bottle()
print(b.description())
```

As you can see, declaration and implementation are in the same file. Swift no longer uses header and implementation files. Let's add a label to our example:

```
class Bottle
{
```

```
  var volume: Int = 1000
  var label:String

  func description() -> String
  {
     return "This bottle of \(label) has \(volume) ml"
  }
}
```

The compiler will complain because label is a non-optional variable and will not hold a value when a Bottle is instantiated. We need to add an initializer:

```
class Bottle
{
  var volume: Int = 1000
  var label:String

  init(label:String)
  {
     self.label = label
  }
  func description() -> String
  {
     return "This bottle of \(label) has \(volume) ml"
  }
}
```

Or, we could use Optional type for a property, which does not to be initialized. In the following example we made volume an Optional Integer:

```
class Bottle
{
  var volume: Int?
  var label:String

  init(label:String)
  {
```

```
        self.label = label
    }

    func description() -> String
    {
        if self.volume != nil
        {
            return "This bottle of \(label) has \(volume!) ml"
        }
        else
        {
            return "A bootle of \(label)"
        }
    }
}
```

Structures:

The Swift language also has structs, but they are much more flexible than in Objective-C. The following code tutorial defines a struct:

```
struct Seat
{
    var row: Int
    var letter:String

    init (row: Int, letter:String)
    {
        self.row = row
        self.letter = letter
    }

    func description() -> String
    {
        return "\(row)-\(letter)"
    }
}
```

Like classes in Swift, structures can have methods, properties, initializers, and conform to protocols. The main difference between classes and structures is that **classes are passed by reference, while structs are passed by value**.

This example demonstrates passing classes by reference:

```
let b = Bottle()

print(b.description())    // "b" bottle has 1000 ml

var b2 = b

b.volume = 750

print(b2.description())    // "b" and "b2" bottles have 750 ml
```

If we try similar case with struct, you will notice that variables are passed by value:

```
var s1 = Seat(row: 14, letter:"A")

var s2 = s1

s1.letter = "B"

print(s1.description())    // 14-B

print(s2.description())    // 14-A
```

When should we use struct and when should we use class? As in Objective-C and C, use structs when you need to group a few values, and expect them to be copied rather than referenced. For example, complex numbers, 2D or 3D points, or RGB colors.

An instance of a class is traditionally known as an object. However, Swift classes and structures are much closer in functionality than in other languages, and much functionality can apply to instances of either a class or a structure type. Because of this, the more general term used in Swift reference is instance, which applies to any of these two.

Learn the basics of Swift classes and structures here.

9.18 PROPERTIES

As we saw earlier, properties in Swift are declared with the var keyword inside a class or struct definition. We can also declare constants with a let statement.

```
struct FixedPointNumber

{

   var digits: Int
```

```
    let decimals: Int
}

var n = FixedPointNumber(digits: 12345, decimals: 2)
n.digits = 4567    // ok
n.decimals = 3     // error, decimals is a constant
```

Also keep in mind that class properties are strongly referenced, unless you prefix them with the weak keyword. However there are some subtleties with weak non-optional properties, so read the automatic reference counting chapter in Apple's Swift guide.

Computed Properties:

Computed properties do not actually store a value. Instead, they provide a getter and an optional setter to retrieve and set other properties and values indirectly.

The following code provides a sample of a computed value sign:

```
enum Sign
{
    case Positive
    case Negative
}

struct SomeNumber
{
    var number:Int
    var sign:Sign
    {
        get
        {
            if number < 0
            {
                return Sign.Negative
            }
            else
            {
                return Sign.Positive
```

```
        }
    }

    set (newSign)
    {
        if (newSign == Sign.Negative)
        {
            self.number = -abs(self.number)
        }
        else
        {
            self.number = abs(self.number)
        }
    }
  }
}
```

We can also define read-only properties by just implementing a getter:

```
struct SomeNumber
{
  var number:Int
  var isEven:Bool
  {
    get
    {
        return number % 2 == 0
    }
  }
}
```

In Objective-C, properties are usually backed by an instance variable, declared explicitly or automatically created by the compiler. In Swift, on the other hand, **a property doesn't have a corresponding instance variable**. That is, the backing store of a property cannot be accessed directly. Suppose we have this in Objective-C

```
// .h
@interface OnlyInitialString : NSObject
```

```
@property(strong) NSString *string;

@end

// .m

@implementation OnlyInitialString

- (void)setString:(NSString *newString)
{
   if (newString.length > 0)
   {
      _string = [newString substringToIndex:1];
   }
   else
   {
      _string = @"";
   }
}

@end
```

Since, in Swift, computed properties don't have a backing store, we need to do something like this:

```
class OnlyInitialString
{
   var initial:String = ""
   var string:String
   {
      set (newString)
      {
         if countElements(newString) > 0
         {
            self.initial =
newString.substringToIndex(advance(newString.startIndex, 1))
```

```
        }
        else
        {
            self.initial = ""
        }
    }
    get
    {
        return self.initial
}
  }
}
```

9.19 SUMMARY

Swift is a powerful and intuitive programming language for iOS, iPadOS, macOS, tvOS, and watchOS. Writing Swift code is interactive and fun, the syntax is concise yet expressive, and Swift includes modern features developers love. Swift code is safe by design, yet also produces software that runs lightning-fast.

9.20 SAMPLE QUESTIONS

1. What is an optional and which problem do optionals solve?
2. Summarize the main differences between a structure and a class.
3. What are generics and How are they useful in SWIFT?
4. In some cases, you can't avoid using implicitly unwrapped optionals. When? Why?
5. What are the various ways to unwrap an optional? How do they rate in terms of safety?
6. What's the difference between nil and .none?
7. Can you add a stored property to a type by using an extension? How or why not?
8. What is a protocol in Swift?
9. Can you describe a circular reference in Swift? How can you solve it?
10. In Swift enumerations, what's the difference between raw values and associated values?

9.21 REFERENCES

- Wei-Meng Lee ,BEGINNING Android™ 4 Application Development , John Wiley & Sons Crosspoint Boulevard Indianapolis ,ISBN: 978-1-118-24067-0
- Reto Meier, Professional Android™ Application Development ,Wiley Publishing, ISBN: 978-0-470-56552-0,www.wiley.com
- ZigurdMednieks, Laird Dornin, G. Blake Meike, and Masumi and Masumi Nakamura, Programming Android , Gravenstein Highway North, Sebastopol
- W. Frank Ableson, RobiSen, Chris King, C. Enrique Ortiz, Dreamtech Press Android in action, Third Edition, ISBN 9781617290503
- Alessandro Biessek Flutter for Beginners: An Introductory Guide to Building Cross-platform Mobile Applications with Flutter and Dart 2 Packt Publishing Ltd.
- Marco L. Napoli Beginning Flutter: A Hands On Guide to App Development John Wiley &Sons
- Rap Payne Beginning App Development with Flutter: Create Cross-Platform Mobile Apps Apress
- https://android.google.com/:/
- https://codelabs.developers.google.com/codelabs/first-flutter-app-pt1/#0
- https://flutter.dev/docs/reference/tutorials
- https://flutter.dev/docs/get-started/learn-more
- https://opensourceforu.com/?s=Flutter
- https://developer.apple.com/library/archive/referencelibrary/GettingStarted/DevelopiOSAppsSwift/
- https://developer.apple.com/ios/
- https://www.apple.com/in/ios/ios-13/

*****