Unit I

1

INTRODUCTION TO JAVA PROGRAMMING

Unit Structure

1.0 OBJECTIVES

In this chapter, you will be going to learn following topics:

- Introduction to java programming and its various features
- Java architecture
- JVM Components
- Different Java platforms
- How to write and execute a java program?
- Different terms used in java language like variables, keywords, identifiers

1.1 INTRODUCTION

What is Java?

Java is a programming language and platform. Java is High-level, object-oriented programming language developed by sun microsystems. It is simple programming language. Java Language is used to developed Standalone Applications, Web Applications, Enterprise Applications and Mobile Applications. Java Provides a runtime Environment (JRE) and Java API, that's why it is called as a platform.

Features of Java language:

1) **Simple:** Java was designed to be a very easy to learn and understand and use effectively. If you understand the basic concepts of object-oriented programming, learning java will be easier.

2) **Object- Oriented:** Java is purely an object-oriented programming language. In Java, everything is an object which has some data and its own behavior.

Following are some basic concepts of OOPs:

- **Class:** Class is collection of objects. You can create multiple objects of a class. It represents properties and methods that are common to all objects of one type.
- **Object:** Any entity that has state and behaviors is called as an object. Object can be defined as instance of a class.
- **Inheritance:** Inheritance is a mechanism in java which allows one class to inherit properties and behavior of another class (parent class).
- **Polymorphism:** If one task is performed in many different ways, it is called as polymorphism. In java method overloading and method overriding is used to implement polymorphism.
- **Abstraction:** Hiding internal details and showing only functionality is known as abstraction. For example, we don't know the internal processing of car, when it is moving.

- **Encapsulation:** Binding or wrapping code and data together into a single unit are known as encapsulation. For example, capsule wrapped it with so many medicines.

3) **Robust:** Java is robust because it provides strong memory management and automatic garbage collection. Java Virtual machine is responsible to manage allocation and deal location of memory for objects. It will automatically free the memory for such objects which have not been used in java application. It also provides compile time and run-time error checking mechanism.

4) **Platform Independent:** Java code is not complied into platform specific machines. Java Code can be executed on multiple platforms like Windows, Linux, Mac/OS etc.

5) **Secured:**We can develop secured and virus free applications. Java is secured because java programs always get executed inside java virtual machine which has no interaction with OS.

6) **Architecture-Neutral:** Java has no implementation dependent features. In java size of the primitive data type is fixed. Once byte code is generated it can be executed on 32-bit architecture as well as 64-bit architecture.

7) **Portable:**Java is writing once and run anywhere language. Java byte code generated after compilation can be executed on any machine.

8) **High- Performance:**Java is an interpreted language. It faster than other traditional interpreted languages.

9) **Distributed:** Java is distributed because it supports creating distributed applications. Using RMI and EJB you can create distributed applications in java.

10) **Multi-Threaded:** Java multi-threading allows you to create multiple threads to executed simultaneously. With this feature we can divide program in multiple threads that will execute different tasks at once.

11) **Dynamic:** Java support dynamic loading of classes.

1.2 HISTORY

Java language was Initially developed by James Gosling, Patrick, Chris Wart hand Mike Sheridan at Sun Microsystems in 1991.This language was initially called “Oak” but was renamed “Java” in 1995. Initially it was developed to create software to be embedded in various electronic devices like microwave ovens, remote controls, set-top boxes etc.

Sun micro system releases first version JDK 1.0 in Jan 1996. It provides Write once and run anywhere functionality. After first release they have added various new features in each new release version. Second version JDK 1.1 was released in Feb 1997.

On 13 November 2006 Sun Micro system released some part of Java Virtual Machine(JVM) as free and open source software, under the General Public License. On May 8, 2007, Sun finished the process of making all of its JVM Code available under free and open source distribution term, except small portion of the code to which sun did not have copyright. In 2010 java was acquired by Oracle Corporation. Later on, Oracle Corporation keeps on releasing new versions by adding many new features in each version. Oracle released current version JDK SE 16 on 16 march, 2021.

1.3JAVA ARCHITECTURE AND ITS COMPONENTS

Java Architecture includes components JVM, JRE and JDK. It combines the process of compilation and Interpretation. It defines all the processes involves in creating a java program. Architecture explains all steps like creating program, how program is compiled and how it is executed.

Java architecture performs following steps:

1) In Java, Program execution is a two-step process which include Compilation and Interpretation.
2) In first Step, Java compiler converts the source code into byte code.
3) In second step, JVM converts byte code into machine code.
4) Then machine code is executed by Operating System.

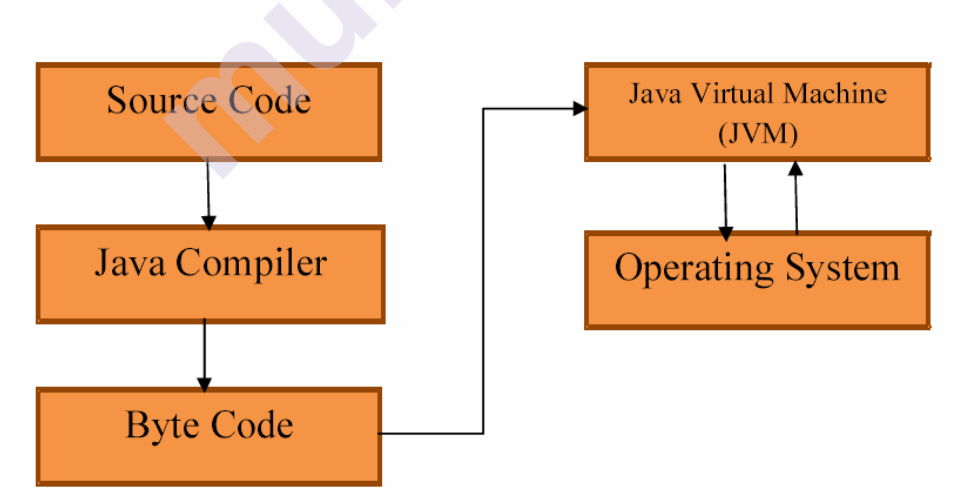


Fig. Java Architecture

Components of Java Architecture

Java Architecture includes three main components:

1) Java Virtual Machine (JVM)
2) Java Runtime Environment (JRE)
3) Java Development Kit (JDK)

1.3.1 The Java Virtual Machine

Java virtual machine is a part of Java Runtime Environment. The primary function of JVM is to execute the java bytecode. It is a specification that provides java runtime environment in which java bytecode is executed. JVM is responsible to load java code in memory and calls main method present in java code and then execute that method.

Java language provides a feature of Write Once Run Anywhere. Which means we can write code once and execute it anywhere and on any operating system. Java program executes on any platform only because of Java Virtual Machine.

1.3.1.1 JavaVirtual Machine Components

JVM performs following operations:

Loads java byte code, verifies code, execute code, provides runtime Environment

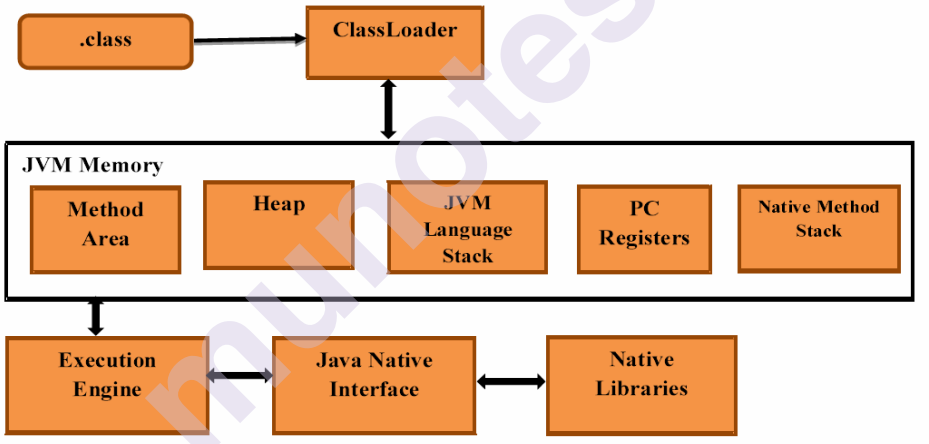


Fig. JVM Architecture

1) **Class Loader:** Class Loader is used to loads the .class files into memory. Whenever we execute the java program, class loader loads it first.

There are three built in class loader are available in Java.

1) **Bootstrap Class Loader:** Bootstrap class loader is super class of Extension Class Loader. It loads all the packages included in Java Standard Edition like java.lang package, java.io package, java.util package, java.net package and so on.
2) **Extension Class Loader:** Extension classloader is subclass of bootstrap classloader and superclass of application classloader. It

loads all jar files present in the $JAVA_HOME/jre/lib/ext directory.

3) **Application Class Loader:** Application Classloader is subclass of the Extension Classloader. It loads files from classpath. By default, classpath is set to a current working directory. You can able to modify classpathby using -cp or -classpath command.

2) Method Area: Method Area stores the all the class level data such as runtime constant pool, fields, method data and code for methods. There is only one method area par JVM.

3) Heap: Heap memory created when JVM Start up. It stores all the objects created during program execution and their corresponding instance variable. It is the run-time data area from which memory for all class instances and array is allocated. There is only one heap area par JVM.

4) Stack: Java Stack memory stores Stack frames, local variable, method calls, partial results. Whenever a new thread is created in the JVM, a separate JVM stack is also created at the same time for that thread. A new frame is created whenever a method is invoked and it is deleted when method invocation process is completed.

5) Program Counter (PC) Registers: Program Counter Register stores address of the currently executing JVM Instruction. JVM Support Multiple threads, so each thread has it own PC register. Once the instruction is executed PC register is updated with new instruction.

6) Native Method Stack: It contains all the native methods used in application. These methods are written any other language than java like c, c++ etc. For each thread separate native Method Stack is created.

7) Execution Engine: Once the bytecode has been loaded into memory, next step is to execute the program. Execution engine is responsible to execute the program. Before executing program, bytecode need to be converted into machine language instructions. The JVM uses interpreter or JIT Compiler to converts byte code into machine language instructions.

8) Execution engine includes:

Interpreter: The interpreter read and executes byte code instructions line by line.

Just-In-Time Compiler: it is used to improved performance. JIT complies the parts of the bytecode that have similar functionality at the same time. It reduces amount of time needed for compilation.

Garbage Collector: The Garbage Collector collects and remove unused objects from the heap area. It is the process of automatically

destroying unused objects and making runtime memory available to use for some other objects.

9) **Java Native Interface:** Java Native Interface(JNI) provides an interface to communicate with another application written in another language like c, c++ etc. Java provides execution of native code through JNI.

10) **Native Libraries:** Java Native Libraries provides a collection of libraries which are written in other programming languages needed by execution engine.

1.3.2 Java Runtime Environment (JRE)

Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide a runtime environment where we can execute our java program. JRE is part of JDK means it get install automatically along with JDK. JRE includes JVM, set of libraries, plugins and applet support. JRE initiates JVM for its execution.

1.3.3 Java Development Kit (JDK)

We need JDK for execution of java programs and for developing java applications. Java Development Kit is a software development environment which is used to developed Java applications. It includes JRE, a compiler, an interpreter, aa debugger and other tools like document generator etc.

1.4 THE JAVA API

Java API (Application Programming Interface) provides a large collection of packages. Each package includes a large collection of classes and interfaces along with their methods.

Java API Packages:

java.io	Provides Classes for system input and output through data streams and the file system.
java.net	Provides classes for implementing networking application.
java.util	It contains date and time facilities and collection framework and classes.
java.awt	Provides a large list of classes for creating user interfaces and for painting graphics and images.
java.sql	Provides classes and interface required for accessing and processing data stored in data based
javax.swing	Provides a set of lightweight components required for creating graphical user interface.

Java.math	Provides classes and interfaces for performing arbitrary-precision integer arithmetic and arbitrary-precision decimal arithmetic..

1.5 JAVA PLATFORM

Java platforms is a collection of programs that helps programmer to develop and run java programs written in java programming language. It is hardware and software environment in which programs runs. Java platform includes an JVM, a compiler and set of libraries called as API.
Java Platforms:

1) **Java SE (Standard Edition):** It is a java programming platform. It includes java programming APIs like java.io, java.net, java.util,java.math, etc.

2) **Java EE (Enterprise Edition):** It is an enterprise platform which is usedto developed web applications and enterprise applications.it includes java technologies like Servlet, JSP, Web Services, EJB etc.

3) **Java ME (Micro Edition):** Java micro edition used to develop mobile applications.

1.6 LAMBDA EXPRESSIONS

Lambda Expression is an anonymous method (unnamed method). Lambda expression is used to implement a method defined by functional interface. Lambda expression is treated as function.

Lambda expression provides an implementation of functional interface. Functional interface is an interface which has only one abstract method. A method which includes only declaration not any implementation is called as abstract method. Functional interface is also known as Single Abstract Method (SAM) interface.

Syntax for creating functional interface:

```
interface interface_name
{
        Return_typemethod_name();  //abstract method
}
```

Example:

1)
```
interface Draw
{
        void circle();
}
```

2) interface Addition
```
{
        add(int a, int b);
}
```

Lambda Expression Syntax:
(argument list) ->{ body }
-> This operator is called as lambda operator or arrow operator. This operator divides lambda expression in two parts, left side includes parameters required by lambda expression and right side include body, which specifies actions of the lambda expression. Body part includes expressions or statements for lambda expression.

Examples:1) () -> 123.14
This lambda expression returns constant value 123.14.
2) () -> 12*6

Program:
```
interface Area
{
        void  findArea();
}
public class LambdaExpressionDemo1
{
        public static void main(String arg[])
        {
                Area obj1;
                int r = 3;
                //Lambda Expression without any argument
                obj1 = () ->System.out.println("Circle Area:" + (3.14*r*r));
                obj1.findArea();

                int side = 5;
        obj1 = () ->System.out.println("Square Area: " +( side*side));
                obj1.findArea();
        }

}
```

Output:

Circle Area:28.259999999999998
Square Area: 25

Lambda expression with parameter:
1) (n) -> n*n
2) (r) -> 3.14*r*r

Program:

```
interface Calculate
{
        int  display(int p, int q);
}
public class LambdaExpressionDemo2
{

        public static void main(String arg[])
        {
                Calculate obj1;
        obj1 = (a,b) -> (a+b);          //lambda expression with argument

                System.out.println("Addition: " + obj1.display(10,20));

                obj1 = (h,w) -> (h*w);
                System.out.print("Rectangle Area:" + obj1.display(5,4));

        }

}
```

Output:

Addition: 30
Rectangle Area:20

1.7 METHOD REFERENCES

Lambda Expression is used to create anonymous methods. Method references is similar to lambda expression. Sometimes, a lambda expression only calls an existing method. When you want to refer existing method by name then you can use method references. Method reference provides a way to refer to a method without executing. Whenever you are using a lambda expression just to refer a method, you can replace that lambda expression with method reference.

Types of method references:

1) Method reference to a static method:

You can refer a static method created in class.

To create static method reference use following syntax:

Class_name :: methodName

Program:

```
interface Calculate                     //functional interface
{
        void cube(int a);               //abstract method
```

```
}
class StaticReference
{
        public static void display(int b)
        {
                System.out.println("Cube: " + (b*b*b));
        }
        public static void main(String arg[])
        {
                Calculate c = StaticReference :: display;
                c.cube(5);
        }
}
```

Output:
Cube: 125

2) Method Reference to anInstance Method

To create instance method reference use following syntax:

Object_reference :: methodName

This syntax is similar to static method reference, except it uses a object reference instead of class name.

Program:

```
interface Calculate
{
        void square(int a);
}
class InstanceReference
{
        public void display(int b)
        {
                System.out.println("Square: " + (b*b));
        }
        public static void main(String arg[])
        {
                InstanceReference instance =new  InstanceReference();
                Calculate c = instance :: display;
                c.square(15);
        }
}
```

Output:
Square: 225

3) Method Reference to a constructor

Constructor: Constructor is a one kind of function in java which has same name as the class name. It does not have any return type. Constructor will get called automatically after creating an object (instance) of a class. Constructor is mostly used to initialized object of class. Every time you will create object of a class, the constructor will get executed.

Syntax: class_name() {}

Example of creating constructor:

```
Class Demo
{
        Demo()          //constructor
        {

                //Statements
        }
}
```

Similarly, you are creating references to a method, you can also create a reference to a constructor.

Syntax for creating reference to a constructor:
Class_name :: new
Program:

```
interface Calculate
{
        void display();
}
public class MyConstructor
{
        MyConstructor()
        {
                System.out.println("Creating reference to a constructor");
        }
        public static void main(String arg[])
        {
                Calculate c = MyConstructor :: new;
                c.display();
        }
}
```

Output:
Creating reference to a constructor

1.8 TYPE ANNOTATIONS

Java Annotations are used to provide supplementary information about your program. Java Annotations represents a metadate or information attached with class, interface, methods, instance variables and constructor etc. Annotations can be used by compiler to detect errors or suppress warnings. Annotations can be applied while declaring methods, fields, classes, constructor and other program elements.

All the annotations in java start with '@' symbol. Annotations do not change action or execution of a compiled program. Annotations are not considered as comments as they change the way the compiler treats a program.

Types of Java Annotations:

1) **Marker Annotation:** The purpose of market annotation is to mark a declaration. Marker annotation do not contain any members or do not consist of any data. Marker interface in java does not consists of any method, for declaring this interface we can use marker annotation. @Override and @Deprecated are the examples of Marker Annotation
   Example of declaring marker Annotation:
   - @TestAnnotation()
   - @interface MyAnnotation{}

2) Single Value Annotation:

Single Value Annotation contain only one value. They allow shorthand form of specifying the value to of the member. While using this annotation, we only need to specify the value for that member and also do not need to specify the name of the member.

Example:

```
@TestAnotation(name = "amit")
@MyAnnotation(value= 10)
```

3) Multi Value Annotation:

Multi value Annotation consists of multiple data members, values and pairs.

Example of declaring Multi value annotation:

1)
```
@interfaceMyAnnotation
{
        int id();
        String name();
        String class();
}
```

2)
```
// you can also provide default value to each member
@interface MyAnnotation2
{
        int id() default 101;
```

String name() “Pranali”;
}

3) @TestAnnotaion(id = 1, name= “shruti”, class = “SYIT”)
4) @Author(name= “balguru swami” , date = “28/8/2021”)

Predefined Java Annotations used in java code:

1) **@Deprecated:** The @Deprecated Annotation is marker annotation that indicates the element is deprecated and should no longer be used. The compiler generated a warning whenever program uses a method, class and field with @Deprecated annotation.

2) **@Override:** the @override annotation informs the compiler that element is meant to override an element declared in a superclass.

3) **@SuppressWarnings:** The @SuppressWarnings annotation instruct the compiler to suppress warning that are generated while program executes. The java language specification lists two categories of warnings: Deprecation and Unchecked.

   Example: To suppress particular category of warning, we use
   @SupressWarning(“Deprecation”)
   To suppress multiple category of warnings, we use
   @SupressWarning({“Deprecation”, “Unchecked”})

4) **@Safevarargs:** This annotation asserts that the annotated method or constructor does not perform unsafe operations on its varargs (variable number of argument).

5) **@Functional Interface:** This annotation is introduced in JDK SE 8, indicates that the type declaration is intended to be a functional interface.

Predefined Java annotation applied to other Annotation:

1) **@Retention:** Retention annotation specifies how the marked annotation is stored:
   - **@RetentionPolicy.SOURCE**: The marked annotation is retained only in the source level and is ignored by the compiler.
   - **@RetentionPolicy.CLASS**: The marked annotation is retained by the compiler at compile time, but it is ignored by JVM.
   - **@RetentionPolicy.RUNTIME**: The marked annotation is retained by the JVM so that it can be used by the runtime environment.

2) **@Documented:**Documented annotation indicated that whenever the specified annotation is used those elements should be documented using the Javadoc tool.

3) **@Terget:** Target annotation marks another annotation to restrict what kind of java elements the annotation can be applied to. It specifies one of the following element types as its value:
    - @ElementType.ANNOTATION_TYPE
    - @ElementType.CONSTRUCTOR
    - @ElementType.FIELD
    - @ElementType.LOCAL_VARIABLE
    - @ElementType.METHOD
    - @ElementType.PACKAGE
    - @ElementType.PARAMETER
    - @ElementType.TYPE

4) **@Inherited:**This annotation indicates that the annotation type can be inherited from the super class
5) **@Repeatable:** repeatable annotation indicates that the marked annotation can be applied more than once to the same declaration or type use.

1.9 METHOD PARAMETER REFLECTION

Method contains executable code which may be invoked, once you called method. By default, .class file generated after compilation does not store formal parameter names of any method or constructor. Method Parameter Reflection provides you feature in which you can able to store names of formal parameters. The java.lang.reflect package provides you all classes required for method parameter reflection. Method and Parameter class are mainly used to store and read formal parameter.

Method class:

The java.lang.reflect.Method class provides APIs to access information about a method's modifiers, return type, parameter, annotations and thrown exception. It is also used to invoked methods. Reflected method can be a class method or instance method.

Method Class Methods

int getModifiers()	Returns java language modifiers for the executable represented by objects.
String getName()	Return the name of the method represented by this method object.
Annotation[][] getParameterAnnotations()	It returns array of array of annotations that represents the annotations of the formal parameter, in declaration order, of the executable represented by object.
Class<?>getReturnType()	It returns a class object that represents the formal return type of the method represented by this Method Object

int getParameterCount()	Returns a number of formal parameters for the executable represented by this object.
Class<?>[] getParameterTypes()	Returns an array of Class objects that represents the formal parameter types, in declaration order, of the executable represented by object.

Parameter Class:

It is used to obtain information about method parameters, including its name, and modifier. To store formal parameter names in a particular .class file, and thus enable the reflection API to retrieve formal parameter names, compile the source file with the -parameters option to the javac compiler.

Parameter Class Methods:

int getModifiers()	Get the modifier flags for the parameter represented by this parameter object.
String getName()	Return the name of the parameter.
Type getParameterizedType()	Return a type object that identifies the parameterized type for the parameter represented by this parameter object.
Class<?>getType()	Return a class object that identifies the declared type for the parameter represented by this parameter object.
booleanisNamePresent()	Return true if the parameter has a name according to class file.
booleanisImplicit()	Return true if this parameter is implicitly declared in source code.
booleanisSynthetic()	Return true is this parameter is neither implicitly nor explicitly declared in source code.

Program:

Calculate.java

```
public class Calculate
{
        int add(int a, int b, int c)
        {
                return (a+b+c);
        }
        int sub(int a, int b)
        {
```

```
                return (a-b);
        }
}
```

Compile this class with following command:
javac -parameters Calculate.java
MethodReflection.java

```
import java.lang.reflect.*;
public class MethodReflection
{
        public static void main(String arg[])
        {
        Calculate c = new Calculate();
        Class obj1 = c.getClass();              //creating instance of Class

        // returns array of methods from calculate class

        Method[] m1 = obj1.getDeclaredMethods();

        for(Method m2 :m1)
        {

                System.out.print(m2.getName());//returns name of method

                Parameter p[] = m2.getParameters(); // returns parameter
        od each method
                for(Parameter p2: p)
                {
                        System.out.print(" " + p2.getParameterizedType());
                        System.out.print(" " +p2.getName());

                }
                System.out.println();
                System.out.println(" no of parameter : " +
m2.getParameterCount());
                System.out.println(" Return Type : " +
m2.getReturnType());
        }

   }
}
```

Command to compile and execute the program
Compile: javac MethodReflection.java
Execute: java MethodReflection

1.10 SETTING THE PATH ENVIRONMENT VARIABLE

The Path is the system variable that your operating system uses to locate needed executables from the command line. If you have saved your java file in JDK/bin directory, the path is not required to be set. If you save your java files in other directory then you need to set path of JDK.

There are two ways to set the JDK path:

1) **Temporary:**It will be valid till your command prompt is open.

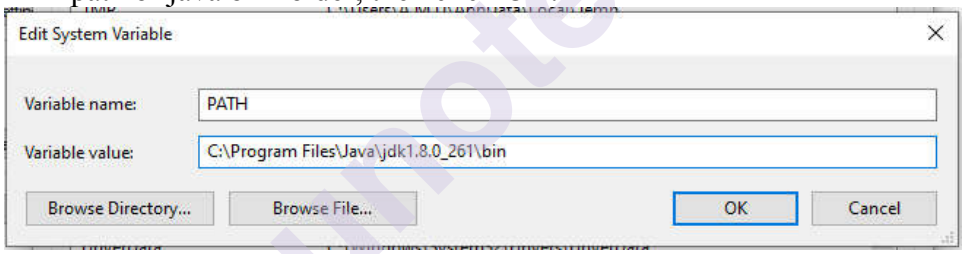
Steps to set temporary path of JDK:

- Open the command prompt.
- Copy the path of JDK/bin Directory
- Write the command in command prompt: set path = copied_path
- Example: **Set path=" C:\Program Files\Java\jdk1.8.0_261\bin"**

2) Permanent:

Steps to set Permanent path of JDK in Windows 10 and Windows 8:

1) Open My Computer properties.
2) Click on Advanced System Setting
3) Click on Environment Variable options.
4) In system variable section, click on new option.
5) Specify variable name as Path and in place of variable value specify path of java bin folder, then click OK.



1.11 JAVA COMPILER AND INTERPRETER

Java program execution is a two-step process i.e., Compilation and interpretation.

Java Compiler: Compiler Translate the source code written by programmer into byte code. The bytecode generated after compilationis save with .class file extension. File name is same as a class name. Compiler also responsible to check source code for errors. It cannot fix errors if present in program; it generates error message and programmer have to solve that all errors. If your program is having no errors then compiler convert it into byte code.

Java Interpreter:

Java Interpreter is a computer program, which implements the JVM specification and helps to actually execute the java bytecode. After compilation compiler generates bytecode (.class file), interpreter takes this

byte code and translate it into machine code. Then interpreter executes instruction available in machine code line by line.

1.12 JAVA CLASS FILE

Java class file contains bytecode. It is saved with **.class** extension. This file automatically gets generated after successful compilation of the program. It has a same name as the class name. This file is executed on Java Virtual Machine. If your java program contains more than one class, then in such cases after compilation we get same number of .class files as the number of classes present in java program. It is saved in same directory where your java file is stored.

1.13 JAVA PROGRAMS

Structure of java program:

Java source code file saved with .java extension. Source code includes one class definition. We can also include multiple classes t single source code file. Class includes one or more methods. All the methods need to be defined inside the class. Within the curly braces of the method, you can write instructions that method should be performed. Method includes collection of statements. Every Java program must have at least one class and one main method in each program.

```
class Example
{
        public static void main(String arg[])
        {
                System.out.print("we are learning java programming");
        }
}
```

Program should be saved with Example.java file name.

Parameters use in above java program:

class: class is inbuild keyword use to declare class in java. Class keyword followed with class_name.

public: It is a access specifier which represents the visibility of main method. Public means accessible everywhere.

static: It is defined keyword in java. If you declared any method as a static in java, then this method should be called without creating any object of the class. Main method is executed by JVM, so there is no need to create any class object to called main method.

void: It is one return type. Void means not returning any value.

main(): main is starting point of program execution. Java program execution starts from main method.

String arg[]: It is argument to the method. Main method must be array of string. This will be use to read command line argument.

System.out.print(): It is used to print statement on output screen. System is a class, out is an object. Print method is used to print statement.
Java program will be executed on command prompt.
Command used to compile java program:

C:\>javac classname.java
example: javacExample.java

This javac compiler create a file with name Example.class stored in save location where you have saved java file. It is the byte code version of original program. This byte code is latter on executed by Java Virtual Machine.

command use to execute the java program:
c:\>java classname
example: java Example
when you run this program, following output is displayed.
Output:
we are learning java programming

1.14 JAVA APPLICATIONS

There are four different types of applications we can developed using java language.

1) **Standalone Application:**Standalone Application also called as desktop application or windows-based application. This kind of software we need to install on every system where you want to use that software. Examples of standalone applications are office suite, media player and all other programming software like turbo++ compiler, python etc. AWT and SWING in java used to developed standalone applications.

2) **Web Application:** Like standalone applications which need to be install on machine, Web Applications we can be access through internet. It is installed on the server and users can used it through internet. Currently, Servlet, JSP, Struts Framework, Spring Framework, Hibernate Framework etc.

3) **Enterprise Application:** An Application that is distributed in nature are called as enterprise application. Examples of enterprise applications are banking application, Skype, call center and customer support application, Health Information Management system etc. Technologies used to develop Enterprise Application

include Java Persistence API, Java Transaction API, Java Enterprise Bean, Java Mail etc.

4) **Mobile Application:**An Applications which is created for mobile devices are called as Mobile Application. It will work only on smart phones.

1.15 WHITESPACE AND CASE SENSITIVITY

Whitespace: Java is free form language. There is no need to follow any special indentation while writing program like you are doing in python programming language. This means you can write entire java program in one line. Whitespace in java includes a space, tab and new line.

Case Sensitivity: Java is case sensitive language. In java program add variable will be different from ADD variable. If you are using some inbuild methods in program then it needs to use in same defined format.
Example: int add;
int ADD;
Both will be considered as completely different variables.

1.16 IDENTIFIERS AND KEYWORDS

Identifiers: Identifier is a sequence of one or more characters. Identifiers are used to name things, like variables, methods and classes. An Identifier is descriptive sequence of uppercase and lowercase letters, numbers, and underscore sign and dollar-sign.

Rules for defining Identifiers:

- You can use combination uppercase and lower-case letters, numbers, dollar-sign and underscore sign for creating variable name. No other special characters are allowed to use while creating identifiers.
- Identifier name must not be start with digit. Example “111add” will not be consider as valid identifier.
- Identifiers are case sensitive.
- Identifier name should be start with either alphabet, underscore sign or dollar-sign.
- Inbuild keywords in java cannot be used as an identifiers.

Example: varaible1, $add, _test2;
This allare the valid identifiers in java language.

Example: 1test, lower-case, ok/cancel;
This all are invalid identifiers in java. Wecannot able to use dash or any other special character while creating identifiers.

Keywords: Keywords are the reserved words in java. Keywords cannot be used as identifiers, means you cannot use it as variable name, method name and class name. The keywords const and goto are reserved but currently not in use. The keyword strictfp is added in JDK 1.2 version. The keyword assert is added in JDK 1.4 version. Enum keyword is added in JDK 5.0.

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Java reserved other four names like null, true, false and var. you may not use these words as variable, class and method name.

Example:
```
class Example2
{
        public static void main(String arg[])
        {
                int x=10;
                int y=20;
                int z= x+y;
                System.out.print("Addition:" + z);
        }
}
```

Output:

Addition: 30

In this example, Example2, arg, x,y and z are called as identifiers which are user defined.

Class, public, static, void, main, String is called as keywords which are predefined in java.

1.17 COMMENTS

Comments are statements in java which are not executed by java compiler and interpreter. Comments are used to make the program more readable by adding details of the code. You can add small description about your program in comments section. You can use comments to provide description about variable, methods, class and any other statement.

There are three types of comments in java:

1) **Single Line Comment:** Single Line comment is use to add single line description for your code. It is easy to use. Single Line comment start with two forwordslashes (//). Any statement written in front of this slash is not executed by compiler.
Syntax: // this is single line comment

2) **Multi Line Comment:** Multi Line comment is use to comment multiple lines of code. It is mostly used to provide explanation about particular method or your program. Multi line comments start with /* and ends with */. The statements written inside these comments are ignore by the compiler and interpreter.

Syntax: /* This is multiline comment
This part is not executed.
This will be used to provide description about the program.
*/

Example: class Comment Example
```
{
    public static void main(String arg[])


{
    /* In this example, we will learn how to create variables and
        How to display value of variable on output screen. */
    double a =4.6755;// we have created variable here
            System.out.print("Variable a:" + a);
    }
}
```

3) **Documentation Comment:** This type of comment is used produced an HTML file that documents your program. These comments are usually used to write a code for project or software application, since it helps to generate documentation page for reference, which can be used for getting information about present methods, classes and arguments etc. Documentation comments start with /** and ends with */.

Syntax:
```
/** Comment start
*You can used various tag to specify a parameters or method or heading
*we can use various HTML tags here.
*
*/
```

1.18 BRACES AND CODE BLOCK

Curly Braces {}: Used to defined values of during array initialization. It is also used to defined block of code for classes, interface. Method and constructor bodies also enclosed in curly braces. Braces are used to group statements in an if statement, a loop and other control statements.

Example: class Example3
```
{
        //statements
}
```

Brackets []: It is used to declare array type. It is also used to access individual elements from array list. And dereferencing array values.

Parentheses (): It is used to declare method and constructor. It is used to provide parameters in method definition and invocation. It is also used to defined order of operation in expression.

1.19 VARIABLES AND VARIABLE NAME

Variable is the basic unit of storage in java program. It is like a container which holds value while java program is executed. A variable is the name given to a memory location. A variable is assigned with data type. A value stored in variable can be changed during program execution. A variable can be defined by the combination of an identifier, a type and an option initializer. All the variables have a scope, which defines their visibility and the lifetime.

Syntax for Variable Declaration:

Type identifier; // variable declaration only
Type identifier = value; // variable with initialization

Here, type defines what type of data is stored in this variable. It defines data type of variable. Identifier is the name of the variable. You can initialize variable by assigning equal sign after name of variable.

Few Examples of variable declaration:

```
int a, b, c;// declaring three int variables
int d=3, e;// declaring two int variable and initializing one int variable d
double d=3.14;// declaring and initializing double variable
char c='a';//declaring and initializing char variable
```

There are three types of variables you can create in java.

1) **Local variable:** The variable declare inside the body of method or constructor is called local variable. Scope of local variable exists only within the block where it is declared. We can access variable only

within same block. You cannot able to use local variable outside of that method.

2) **Instance Variable:**Variable which is declare inside the class and outside the method is called as instance variable. Instance variables are created when an object of class is created and destroyed when object get destroyed. Value of instance variable is instance specific. You can use access specifier to defined scope of instance variable. According to defined access specifier visibility and lifetime of this variable get changed.

3) **Static Variable:** A variable that is declared with static keyword is called static variable. This variable is also declared inside the class. For static variable, memory allocated only once when class is loaded in the memory. Like instance variable static variable is not instance specific. Static variable is created when you want to share same value among all objects of the class. Means Value of static variable is similar for all objects of class.

Programs:

1) Write a java program to find out area of triangle.

```
class Triangle
{
        public static void main(String arg[])
        {
                Double pi=  3.14, r = 3, area;
                area = 3.14*r*r;
                System.out.print("Area of Triangle" + area);
        }
}
```

Output:

Area of Triangle: 28.26

2) Program to calculate simple Interest

```
public class SimpleInterest
{
   public static void main(String args[])
   {
      float p =2000, r = 6, t =3, sinterest;
         //simple interest formula (principal* rate of interest * time
period)/100
sinterest = (p * r * t) / 100;
System.out.print("Simple Interest is: " +sinterest);
   }
}
```

Output:

Simple Interest is: 360.0

1.20 SUMMARY

In this chapter, we learn introduction to java language, features of java, Java virtual machine, Java Runtime Environment, Java Development Kit, LambdaExpression, Method References, Writing and executing java programs, identifiers, keywords, java API and variables with examples.

1.21 LIST OF REFERENCES

Java, A Beginner's Guide, Eighth Edition, Herbert Schildt, McGraw Hill Publisher

Java: The Complete Reference, Eleventh Edition, Herbert Schildt, McGraw Hill Publisher

1.22 BIBLIOGRAPHY

- https://www.javatpoint.com/java-tutorial
- https://www.geeksforgeeks.org/java
- https://www.beginnersbook.com/java-tutorial-for-beginners-with-examples/
- http://www.programiz.com/java-programming/
- https://docs.oracle.com/javase/tutorial/

1.23 MODEL QUESTIONS

1) Which component is used to compile, debug and execute java program?
   a) JVM
   b) JRE
   c) JDK
   d) JIT

2) Which component is responsible for converting bytecode into machine specific code?
   a) JVM
   b) JRE
   c) JDK
   d) JIT

3) What is extension of java byte code?
   a) .txt
   b) .class
   c) .java
   d) .js

4) JRE stand for?
   a) Java Runnable Environment
   b) Java Runtime Environment
   c) Java Runtime Extension
   d) None of the above

5) ____ are the reserved words in java?
   a) Keywords
   b) Variables
   c) Identifiers
   d) All of the above

6) _____ is a sequence of one and more characters?
   a) Identifiers
   b) Keywords
   c) Variables
   d) Literals

2

DATA TYPES

Unit Structure

2.0 OBJECTIVE

In this chapter, you will be going to learn following topics:

- Primitive data types supported in java.
- Creating string in java using String class and different methods of string class.
- How to convert primitive type to object type.
- Use of different types operators available in java.

2.1 INTRODUCTION

Data types and Operator are the main fundamental elements of java Programming. In this chapter we will learn different data types supported

in java language. Java is strongly type language, which means that you must need to declare variable before they can be used. This declaration includes variable's type and name. Data types defines type of value stored in variable.

Another concept we will going to learn in this chapter is operators and types of operators supported in java. Java provide a rich set of operators. Java supports all basic operators and some additional operators for handing special condition.

2.2 PRIMITIVE DATA TYPES

Data Types: When you define a variable in java, you must need to defined what kind of value that variable will stored. This will be defined with the help of data types. Based on this information compiler will decide how much space to allocate in the memory for the variable.

Java programming language defines eight primitive data types: byte, short. int, float, double, char, boolean. A primitive type is predefined in java language. Primitive types are the most basic data types available in java. Size of these primitive data type is fixed; it does change from one operating system to another. This primitive type can be placed into four group.

Integers: This group includes byte, short, int and long, which represents whole numbers. All these integers support signed, positive and negative values.

Floating- point numbers: This group include float and double data type, which represents numbers with fractional value. Floating point numbers are used when evaluating expression that require fractional precision.

Character: this group include char data type.

Boolean: This group include Boolean data type, which includes binary value like true/false.

Data Type	Size	Default value
byte	1 byte	0
short	2 byte	0
int	4 byte	0
long	8 byte	0
float	4 byte	0.0f
double	8 byte	0.0d
char	2 byte	\u0000

boolean	1 bit	false

Fig. Primitive Data Types

byte:

It is a smallest integer type. The Byte data type is 8 bits (1 byte) signed two's complement integer. It has value range lies between -128 to 127. Its default value is zero.

Syntax of declaring byte variable:

byte variable1;

short:

It is a signed 16 bits (2 byte) two's complement integer. It has value range lies between -32,768 to 32,767. Its default value is zero.

Syntax:

short t, s;

int:

The int data type is 32 bits (4 bytes) signed two's complement integer. It has the value range lies between- (2^{31})to (2^{31} -1). Its default value is zero. It is most commonly used data type.

Syntax:

int a, b;

int c=2;

long:

The long data type is 64 bits (8 byte) signed two's complement integer. It has value range lies between – (2^{63}) to (2^{63}-1). Its default value is zero.

Example:

```
class DataTypesExample1
{
        public static void main(String arg[])
        {
        byte b = -100;
        short num = 150;
        int a = 1;
        long l = 4562455555555L;
        System.out.println("Byte Value :" + b);
        System.out.println("Integer Value:" + a );
        System.out.println(" Int value: " + a );
        System.out.println("Long value: " + l );
        }
}
```

Output:

```
Byte Value :-100
Integer Value:1
 Int value: 1
```

Long value: 4562455555555

float:

The float data type is single precision 32 bits (4 byte) floating point. It is useful for storing smaller precision but not suited for large degree of precision. Its default value is 0.0f.

Syntax:

Float f = 3.14f, a;

double:

The double data type is double precision 64 bits (8 byte) floating point. It is useful for storing floating point values with large precision. Its default value is 0.0d.

Syntax:

double d = 45.78521457;

char:

The char is 16 bits (2 byte) Unicode character. It has the value range lie between 0 to 65,535. This data type is used to stored characters.

Unicode defines a fully international character set that can represents all of the characters that are available in all human languages. Its default value is '\u0000'.

Syntax:

char c = 'g';

Boolean:

The Boolean data type is used to stored logical value. It stores only two possible values: true and false. It represents only one bit of information. Values of Boolean variable can not be converted into any other type. Its default value is false.

Example:

```
class  DataTypesExample2
{
      public static void main(String arg[])
      {
        float f = 22.14f;
      double d = 12.11122452;
      char c='g';
      boolean flag = true;
      System.out.println("float value:" + f);
      System.out.println("double value:" +d);
      System.out.println("Char value:"+ c);
      System.out.println("boolean value:"+ flag);
      }
```

}

Output:
float value:22.14
double value:12.11122452
Char value:g
booleanvalue:true

2.3 OBJECT REFERENCE TYPES

In java, not- primitive data type is also known as object reference type. Java object reference type holds the references of dynamically created class objects and provide a means to access those objects stored in memory. For example, if you have created class Example and you have created its object d, then d is called as object reference type. All the reference types are subclass of type java.lang.Object. Object reference type includes class, Annotation, Array, Enumeration and Interface. Like primitive data type it is not predefined.

Reference Type	Description
Class	Class is a collection of objects. You can create multiple objects of class.
Annotation	It provides a way to associate metadate with program elements.
Array	It provides a data structure that stores the elements of similar type.
Enumeration	A reference to a set of objects that represents a related set of choices.
Interface	It is implemented by java classes. interface provides you collection of abstract methods.

Fig. Object Reference Types

2.4 STRINGS

Java String is an object that represents a sequence of characters. String class is used to create string object in java. It is defined in java.lang package. Java string is immutable. Once string is created, you cannot able to change its value.

There are two ways to create string in java:
1) Using String Literal
Examples: String name = “Amit”;
String address = “thane”;

2) Using new keyword

Examples: String s1 = new String("Welcome");
String s2 = new String("we are learning java programming");

Program:

```
class StringExample
{
   public static void main(String arg[])
   {
      //creating string using string literal
      String name = "Kunal";
      System.out.println("Name: " + name);

      //creating string using new keyword
      String s = new String("java is easy to learn");
      System.out.println(s);
   }
}
```

Output:

Name: Kunal
Java is easy to learn

String class also provides you lots of methods to perform different operations on string.

List of Methods:

Method	**Description**
toUpperCase(String str)	Convert the given string in uppercase.
toLowerCase(String str)	Convert the given string in lowercase.
startWith(String prefix)	Test if given string starts with specified prefix. True or false result.
endsWith(String Suffix)	Test if given string ends with specified suffix. Return true or false result.
charAt(int index)	Return the character value for specified index number.
length()	Return the length of this string.
substring(int beginindex)	Return new string that is substring of the given string.
replace()	Used to replace all the occurrences of old char in this string with new char.
contains(CharSequence s)	Return true or false after matching specified

	character sequence in given string.
indexOf(int ch)	Return the index value of specified char.
equals(String str)	Compares the given string with this specified string. Return true or false result.
trim()	Use to remove leading and trailing whitespaces in this string.
valueOf(int value) valueOf(float value) valueOf(double value) valueOf(long value)	Used to convert given type to string.
concat(String str)	Concatenates specified string at the end of this string.

Example:

```
class StringMethods
{
   public static void main(String arg[])
   {
      String s="java programming";
      String s1= " PYTHON";
      System.out.println("Upper Case : " + s.toUpperCase());
      System.out.println("Lowercase : " +s1.toLowerCase());
      System.out.println("Trim : " + s.trim());
      System.out.println( "Start With : " + s.startsWith("Sa"));
      System.out.println("Ends with: " + s.endsWith("g"));
      System.out.println("Char at: " + s.charAt(3));
      System.out.println( "length: " + s.length());
      System.out.println("substring : " +s.substring(2,4));
      String r=s.replace("java","python");
      System.out.println(r);
      System.out.println("Contains: " + r.contains("java"));
      int index1=s.indexOf("a");
      System.out.println("Index OF : " + index1);
      int index2=s.lastIndexOf('g');
      System.out.println("last Index of: " + index1);
          String str1 = "Hello";
          String str2 = "Javatpoint";

        // Concatenating one string
```

```
        String str4 = str1.concat("javatpoint");
      System.out.println(str4);
      }
  }
```

Output:

Upper Case : JAVA PROGRAMMING
Lowercase : python
Trim : java programming
Start With : false
Ends with: true
Char at: a
length: 16
substring :va
python programming
Contains: false
Index OF : 1
last Index of: 1
Hellojavatpoint

2.5 AUTOBOXING AND UNBOXING

Autoboxing:

The Automatic conversion of primitive type into its corresponding wrapper class type is called Autoboxing. For example, converting an int to an Integer. Converting an float to Float type.

Unboxing:

The automatic conversion of object of wrapper type into its corresponding primitive type is called as unboxing. For example, converting Integer object into int type, Double object into double type.

List of primitive type and their corresponding wrapper type:

Primitive Type (Data Types)	Wrapper Type (Classes)
byte	Byte
short	Short
int	Int
long	Long
float	Float
double	Double
char	Char
boolean	Boolean

Example:

```
class AutoboxingEx
{
        public static void main(String arg[])
        {
          int a=20;
          Integer n1 = new Integer(a);   // autoboxing
          Integer n2 = a;              //autoboxing
          System.out.println("n1 : " +n1);
          System.out.println( "n2:  " + n2);

          Integer n3 = new Integer(50); //unboxing
          int b = n3;
          System.out.print("b: " +b);
        }
}
```

Output:

```
n1 : 20
n2:  20
b: 50
```

2.6 OPERATORS AND PROPERTIES OF OPERATORS

Operators are used to performed some mathematical and logical operations. Java provides a rich set of operators. All basic operators are classified into four types; relational, arithmetic, logical and bitwise. It also provides you some additional operators like conditional operator, assignment operator and increment and decrement operator.

2.6.1 Arithmetic Operators

Arithmetic operators are used to perform operations like addition, subtraction, multiplication and division. These are the basic mathematical operator.

List of arithmetic operators:

Operators	Description
+	**Addition**
-	**Subtraction**
*	**Multiplication**
/	**Division**
%	**Modulus division**

Example:

```
class ArithmeticOperator
{
        public static void main(String arg[])
        {
         double a=20, b=4;
         System.out.println("Addition: " + (a+b));
         System.out.println("Subtraction: " + (a-b));
         System.out.println("Multiplication: " + (a*b));
         System.out.println("Division: " + (a/b));
         System.out.println("Modulus: " + (a%b));
        }
}
```

Output:

Addition: 24.0
Subtraction: 16.0
Multiplication: 80.0
Division: 5.0
Modulus: 0.0

2.6.2 Assignment Operators

Assignment operator is one of the most common operators; you are using in java program. It is used to assign value on its right to the operand (variable) on its left.
Symbol: =

Example: int a = 4;
double d = 3.2;

Compound Assignment Operators:

Operator	Description
+=	**Addition Assignment:**It work by adding the current value of the variable on left to the value on the right and then assign result to the operand on the left.
-=	**Subtraction Assignment:** It work by subtracting the current value of the variable on left to the value on the right and then assign result to the operand on the left.

/=	**Division Assignment:** It work by dividing the current value of the variable on left to the value on the right and then assign result to the operand on the left.
*=	**Multiplication Assignment:** It work by multiplying the current value of the variable on left to the value on the right and then assign result to the operand on the left.
%=	**Modulus Assignment:** It work by dividing the current value of the variable on left to the value on the right and then assign remainder to the operand on the left.

Program:

```
class AssignmentOperator
{
        public static void main(String arg[])
        {
         int num = 60;              // simple asignment operator
         double a=20, b=30;
          a += 10;         // it is similar to  the expression a= a+10
          b -= 5;// it is similar to b= b-5;
         System.out.println(" Assignment operator: " + num);
         System.out.println("Addition Assignment: " + a);
         System.out.println("Subtraction Assignment: " + b);
        }
}
```

Output:

Assignment operator: 60
Addition Assignment: 30.0
Subtraction Assignment: 25.0

2.6.3 Increment and Decrement Operators

Increment operator is used to increase value of operand by one. Decrement operator is used to decrease value of operand by one. Both operators required only one operand.These operators can be used in either prefix and postfix form. In prefix form, the operand is incremented and decremented before the value is obtain for used in the expression. In postfix form, the operand is incremented and decremented after the value is obtain for used in the expression.

Operator	Description
++	Increment Operator
--	Decrement Operator

Program:

```
class Operator
{
        public static void main(String arg[])
        {
         int num1 = 20, num2 = 10;
         System.out.println(num1++);          //  num1++ is post-increment
         System.out.println(++num1);          // ++num1is pri-increment
         System.out.println(num2--);          //it is post-decrement
         System.out.println(--num2);          // it is pri-decrement
        }
}
```

Output:

20
22
10
18

Relational Operators

Relational operators are used to determine relationship between two variables, like value of one operand is greater then, equal to or less than another operand. The relational operator generates true and false result.

Operator	Description
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than and equal to
<=	Less than and equal to

Fig. Relational Operators

Program:

```
class Relationaloperator
{
        public static void main(String arg[])
        {
        int a=10, b=30, c=20;
        System.out.println("a<b : " + (a<b));
        System.out.println("a>b :" + (a>b));
        System.out.println("a==b :" +(a==b));
        System.out.println(" a!=b : " + (a!=b));
        }
}
```

Output:

a<b : true

a>b :false

a==b :false

a!=b : true

2.6.4 Logical Operators

Logical operators are used to perform logical AND, OR and NOT operation. Its function is similar to AND gate, OR gate and NOT gate in digital electronics. Logical AND gate return true result if both the expressions are true. Logical OR gate return true result if either one expression is true. Logical Not return true if expression is false and vice versa.

Operator	Description
&&	Logical AND
||	Logical OR
!	Logical NOT

Fig. Logical Operators

Program:

```
class LogicalOperator
{
        public static void main(String arg[])
        {
        int a=10, b=30, c=20;
        System.out.println("Logical AND: " + (a<c  && a<b) );
        System.out.println("Logical OR: " + (a>c  || a<b) );
        System.out.println("Logical OR: " + (a>c  && a<b) );
        System.out.print("Logical NOT:" + !(a>b));
        }
}
```

Output:

Logical AND: true

Logical OR: true

Logical OR: false

Logical NOT:true

2.6.5 Bitwise Operators

Bitwise operators operate on binary value. It returns result by performing manipulation on individual bit of a number. In Bitwise OR operation, it performs bit by bit OR operation of input value and it return 1 if either of the bits is 1 otherwise it returns 0.

In bitwise AND operation, it performs bit by bit AND operation of input value and it return 0 if either of the bits is 0 otherwise it returns 1.

In bitwise XOR operation, it performs bit by bit XOR operation of input value and it return 1 if corresponding bits different, else it gives zero if bits are same. Bitwise Complement operator returns one's complement of input value.

Operator	Description
&	Bitwise AND
|	Bitwise inclusive OR
^	Bitwise Exclusive OR
~	Bitwise Complement

Program:

```
class BitwiseOperator
{
        public static void main(String arg[])
        {
        int a=5, b=12;
        System.out.println("Bitwise AND: " + (a&b) );
        System.out.println("Bitwise  Inclusive OR: " + (a|b) );
        System.out.println(" Bitwise Exclusive OR: " + (a^b));
        System.out.print("Bitwise Complement:" + (~a));
        }
}
```

Output:

Bitwise AND: 4
Bitwise Inclusive OR: 13
Bitwise Exclusive OR: 9
Bitwise Complement:-6

2.6.6 Conditional Operator

Operator	Description
&&	Logical or Conditional AND
||	Logical or Conditional OR
?:	Ternary operator

Ternary operator is similar to If-then-else statement in java. A ternary operator evaluates the test condition and execute the block of code based on result of the condition.

Syntax: condition ? expression1 : expression2

It first checks the test condition and execute expression1 if condition is true, else it execute expression2.
Program:

```
class TernaryOperator
{
        public static void main(String args[])
        {
         int marks=40;
         String result = (marks>40) ? "PASS" : "FAIL";
         System.out.print(result);

        }
}
```

Output:
FAIL

2.7 SUMMARY

In this chapter, we learn primitive data types, object reference types, creating string in java and different methods of string class, converting primitive date type to object type and vice-versa, operators and their different types.

2.8 LIST OF REFERENCES

Java, A Beginner's Guide, Eighth Edition, Herbert Schildt, McGraw Hill Publisher

Java: The Complete Reference, Eleventh Edition, Herbert Schildt, McGraw Hill Publisher

2.9 BIBLIOGRAPHY

- https://www.javatpoint.com/java-tutorial
- https://www.geeksforgeeks.org/java
- https://www.beginnersbook.com/java-tutorial-for-beginners-with-examples/
- http://www.programiz.com/java-programming/
- https://docs.oracle.com/javase/tutorial/

2.10 MODEL QUESTIONS

1) How many primitive types are there in java?
   a) 5
   b) 6
   c) 7
   d) 8

2) Size of int type in java?
   a) 8 bit
   b) b) 4 bit
   c) 16 bit
   d) d) 32 bit

3) The smallest integer type is ___ and its size is ___ bits.
   a) short, 16
   b) byte, 16
   c) short, 8
   d) byte, 8

4) which of the following is a symbol of assignment operator?
   a) ==
   b) =
   c) !
   d) ~

5) Size of double data type in java?
   a) 4 byte
   b) 8 byte
   c) 2 byte
   d) 1 byte

6) What is the range of byte data type in java?
   a) - 128 to 127
   b) -128 to 128
   c) -127 to 128
   d) -127 to 127

7) Increment operator increases value of operand by which value?
   a) 2
   b) 3
   c) 1
   d) 4

8) Which of the following is not conditional operator?
   a) &&
   b) ||
   c) ?:
   d) ~

9) What is the size of short data type in java?
   a) -128 to 127
   b) -32768 to 32767
   c) $-(2^{31})$ to $(2^{31}-1)$
   d) None of this

10) In java, byte, short, int and long all of these are ____ integers.
   a) Signed
   b) Unsigned
   c) Both of the above
   d) None of the above

Unit II

3

CONTROL STATEMENTS

Unit Structure

3.1. INTRODUCTION

Java compiler executes the code through and through. The assertions in the code are executed by the request where they show up. Nonetheless, Java gives proclamations that can be utilized to control the progression of Java code. Such proclamations are called control stream explanations. It is one of the crucial components of Java, which gives a smooth progression of program.

3.2. JAVA CONTROL STATEMENTS

In Java, program is a bunch of articulations and which are executed consecutively all together in which they show up. In that assertions, some

estimation have need of executing for certain conditions and for that we need to give control to that assertions. All in all, Control explanations are utilized to furnish the progression of execution with condition. In this unit, we will gain proficiency with the control structure exhaustively. Java provides three types of control flow statements.

Decision Making statements

- if statements
- switch statement

Loop statements

- do while loop
- while loop
- for loop
- for-each loop

Jump statements

- break statement
- continue statement

3.2.1. Decision-Making statements:

As the name recommends, dynamic proclamations choose which articulation to execute and when. Dynamic articulations assess the Boolean articulation and control the program stream contingent on the consequence of the condition gave. There are two sorts of dynamic explanations in Java, i.e., If articulation and switch proclamation.

3.2.2. If Statement

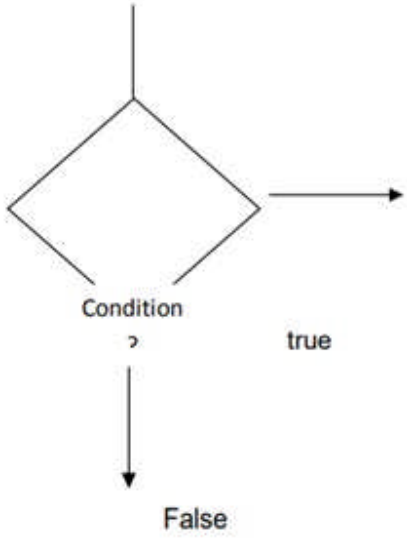
In Java, the "if" articulation is utilized to assess a condition. The control of the program is redirected relying on the particular condition. The state of the If articulation gives a Boolean worth, either obvious or bogus. In Java, there are four kinds of if-proclamations given underneath.

- Simple if statement
- if-else statement
- if-else-if ladder
- Nested if-statement

Let's understand the if-statements one by one.

Simple if statement:

It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.



Syntax of if statement is given below.

if(condition) {
statement 1; //executes when condition is true
}
Statement-a;

In proclamation block, there might be single explanation or various articulations. Assuming the condition is valid, articulation square will be executed. Assuming the condition is bogus, proclamation square will preclude and articulation a will be executed. Consider the accompanying model wherein we have utilized the if articulation in the java code.

1) Student.java

```
public class Student {
public static void main(String[] args) {
int x = 10;
int y = 12;
if(x+y > 20)
{
        System.out.println("x + y is greater than 20");
}
}
}
```

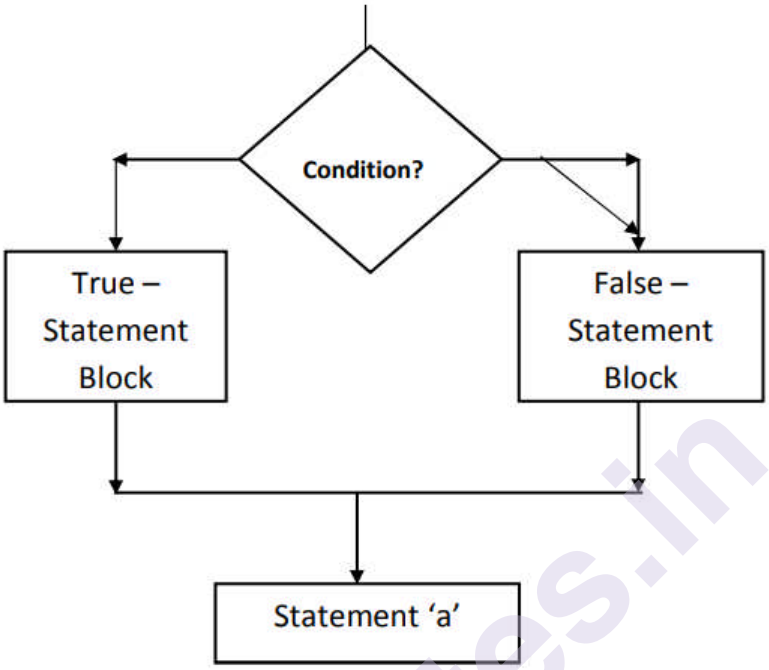
Output:

x + y is greater than 20

3.2.3. if-else statement

The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

Following figure shows the flow of statement.



Syntax:

```
if(condition) {
statement 1; //executes when condition is true
}
else{
statement 2; //executes when condition is false
}
Statement-a;
```

If the condition is true then True - statement block will be executed. If the condition is false then False - statement block will beexecuted. In both cases the statement-a will always executed.

Consider the following example.

1) Student.java

```
public class Student {
public static void main(String[] args) {
int x = 10;
int y = 12;
if(x+y < 10)
```

```
{
System.out.println("x + y is less than      10");
}
else {
System.out.println("x + y is greater than 20");
}
}
}
```

Output:

x + y is greater than 20

2) write a program to check whether the number is divisible by 2 or not.

```
import java.io.*;
classdivisorDemo
{
public static void main(String[ ] args)
{
int a =11;
if(a%2==0)
{
System.out.println(a +" is divisible by 2");
}
else
{
System.out.println(a+" is not divisible by 2");
}
}
}
```

Output:

11 is not divisible by 2

3.2.4.if-else-if ladder

The if-else-if explanation contains the if-articulation followed by different else-if proclamations. At the end of the day, we can say that it is the chain of if-else articulations that make a choice tree where the program might enter in the square of code where the condition is valid. We can likewise characterize an else explanation toward the finish of the chain.

Syntax of if-else-if statement is given below.

```
if(condition 1)
{
```

```
statement 1; //executes when condition 1 is true
}
else if(condition 2)
{
statement 2; //executes when condition 2 is true
}
else {
statement 2; //executes when all the conditions are false
}
```

Consider the following example.

1) Student.java

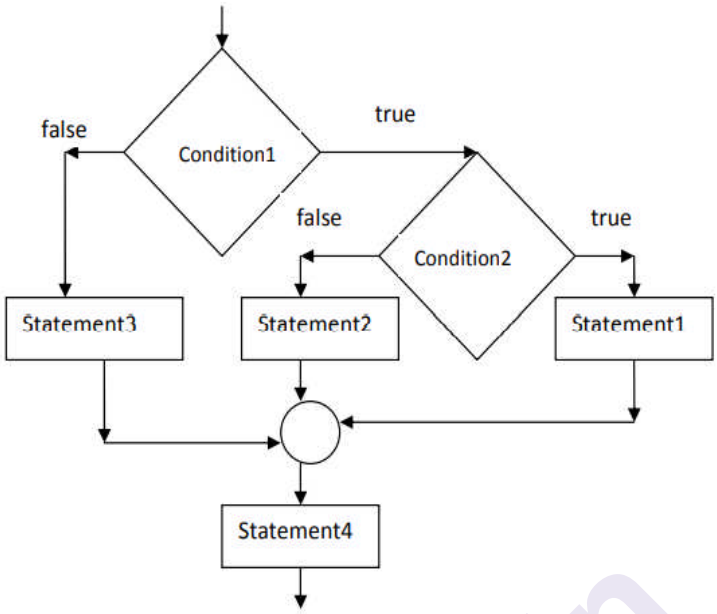
```
public class Student {
public static void main(String[] args) {
String city = "Delhi";
if(city == "Meerut")
 {
        System.out.println("city is meerut");
}
else if (city == "Noida")
{
        System.out.println("city is noida");
}
else if(city == "Agra")
 {
        System.out.println("city is agra");
}
else
{
        System.out.println(city);
}
}
}
```

Output:

Delhi

3.2.5. Nested if-statement

In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.



Syntax of Nested if-statement is given below.

```
if(condition 1) {
statement 1; //executes when condition 1 is true
if(condition 2) {
statement 2; //executes when condition 2 is true
}
else{
statement 2; //executes when condition 2 is false
}
}
statement 4;
```

If the condition1 is true then it will be goes for condition2. If the condition2 is true then statement block1 will be executed otherwise statement2 will be executed. If the condition1 is false then statement block3 will be executed. In both cases the statement4 will always executed.

Consider the following example.

1) Student.java

```
public class Student {
public static void main(String[] args) {
String address = "Delhi, India";

if(address.endsWith("India"))
{
```

```
        if(address.contains("Meerut"))
        {
                System.out.println("Your city is Meerut");
        }
        else if(address.contains("Noida"))
        {
                System.out.println("Your city is Noida");
        }
        else
        {
                System.out.println(address.split(",")[0]);
        }
}
Else
 {
        System.out.println("You are not living in India");
}
}
}
```

Output:

Delhi

3.3. SWITCH STATEMENT:

In Java, Switch articulations are like if-else-if explanations. The switch explanation contains various squares of code called cases and a solitary case is executed dependent on the variable which is being exchanged. The switch proclamation is simpler to use rather than if-else-if explanations. It likewise improves the intelligibility of the program.

Focuses to be noted with regards to switch proclamation:

The case factors can be int, short, byte, roast, or list. String type is likewise upheld since rendition 7 of Java

Cases can't be copy

Default explanation is executed when any of the case doesn't coordinate with the worth of articulation. It is discretionary.

Break explanation ends the switch block when the condition is fulfilled.

It is discretionary, if not utilized, next case is executed.

While utilizing switch proclamations, we should see that the case articulation will be of a similar kind as the variable. Nonetheless, it will likewise be a consistent worth.

The syntax to use the switch statement is given below.

```
switch (expression){
   case value1:
    statementblock1;
    break;
case value 2:
    statement block 2;
    break;
case value 3:
    statementblock 3;
    break;
   .
   .
   .
   case valueN:
    statementN;
    break;
   default:
    default statement;
}
```

The condition is byte, short, character or a number. value1,value-2,value-3,… are steady and is called as marks. Every one of these qualities be inimitable or special with the assertion. Proclamation block1, Articulation block2, Explanation block3,..are rundown of explanations which contain one articulation or more than one proclamations. Case mark is consistently end with ":" (colon).

Consider the following example to understand the flow of the switch statement.

1) Student.java

```
public class Student implements Cloneable
{
public static void main(String[] args)
{
int num = 2;
switch (num)
{
case 0:
System.out.println("number is 0");
break;
```

```
case 1:
System.out.println("number is 1");
break;
default:
System.out.println(num);
}
}
}
```

Output:

2

While utilizing switch articulations, we should see that the case articulation will be of a similar kind as the variable. Be that as it may, it will likewise be a steady worth. The switch allows just int, string, and Enum type factors to be utilized.

Program: write a program for bank account to perform following operations.

-Check balance

-withdraw amount

-deposit amount

1) For example:

```
import java.io.*;
classbankac
{
public static void main(String args[]) throws Exception
{
intbal=20000;
intch=Integer.parseInt(args[0]);
System.out.println("Menu");
System.out.println("1:check balance");
System.out.println("2:withdraw amount... plz enter choiceand amount");
System.out.println("3:deposit amount... plz enter choiceand amount");
System.out.println("4:exit");
switch(ch)
{
case 1:
        System.out.println("Balance is:"+bal);
break;
case 2:
        int w=Integer.parseInt(args[1]);
        if(w>bal)
```

```
        {
                System.out.println("Not sufficient balance");
        }
        bal=bal-w;
        System.out.println("Balance is"+bal);
break;

case 3:
        int d=Integer.parseInt(args[1]);
        bal=bal+d;
        System.out.println("Balance is"+bal);
break;

default:
break;
}
}
}
```

Output:

Menu
1:check balance
2:withdraw amount... plz enter choice and amount
3:deposit amount... plz enter choice and amount
4:exit
Balance is:20000

3.4. LOOP STATEMENTS

In programming, now and again we need to execute the square of code more than once while some condition assesses to valid. Be that as it may, circle explanations are utilized to execute the arrangement of guidelines in a rehashed request. The execution of the arrangement of directions relies on a specific condition.

In Java, we have three kinds of circles that execute also. Notwithstanding, there are contrasts in their grammar and condition really looking at time.

- for loop
- while loop
- do-while loop

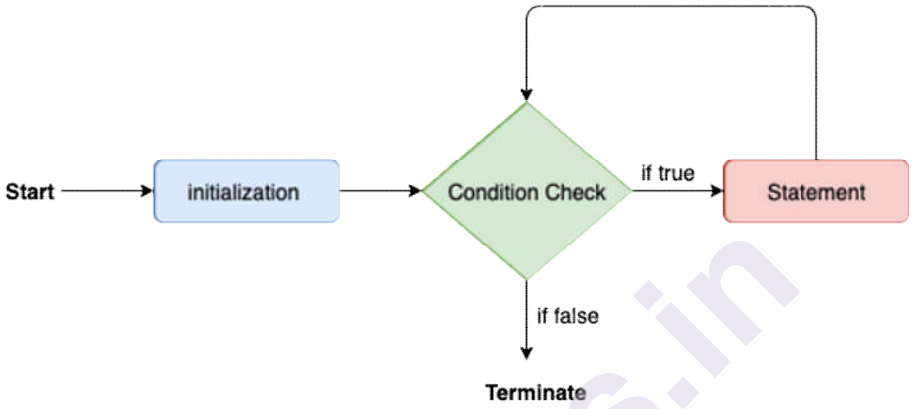
Let's understand the loop statements one by one.

3.4.1. Java for loop

In Java, for circle is like C and C++. It empowers us to introduce the circle variable, actually look at the condition, and addition/decrement in a solitary line of code. We utilize the for circle just when we precisely know the occasions, we need to execute the square of code.

for(initialization, condition, increment/decrement)
{
//block of statements
}

The flow chart for the for-loop is given below.



Consider the following example to understand the proper functioning of the for loop in java.

1) Calculation.java

```
public class Calculattion
{
public static void main(String[] args)
{
// TODO Auto-generated method stub
int sum = 0;
for(int j = 1; j<=10; j++)
{
        sum = sum + j;
}
System.out.println("The sum of first 10 natural numbers is " + sum);
}
}
```

Output:

The sum of first 10 natural numbers is 55

3.4.2. Java for-each loop

Java gives an improved to circle to navigate the information structures like exhibit or assortment. In the for-each circle, we don't have

to refresh the circle variable. The linguistic structure to utilize the for-each circle in java is given beneath.

```
for(data_type var : array_name/collection_name)
{
//statements
}
```

Consider the following example to understand the functioning of the for-each loop in Java.

1) Calculation.java

```
public class Calculation
{
public static void main(String[] args)
{
// TODO Auto-generated method stub
String[] names = {"Java","C","C++","Python","JavaScript"};
System.out.println("Printing the content of the array names:\n");
for(String name:names)
{
System.out.println(name);
}
}
}
```

Output:

```
Printing the content of the array names:
Java
C
C++
Python
JavaScript
```


3.4.3. Java while loop

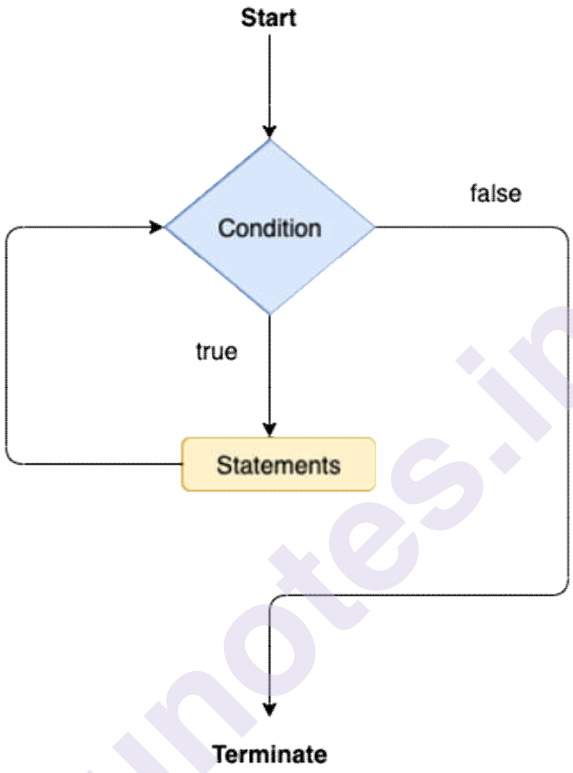
The while circle is likewise used to emphasize over the quantity of explanations on numerous occasions. Notwithstanding, in the event that we don't have the foggiest idea about the quantity of emphasess ahead of time, it is prescribed to utilize some time circle. Not at all like for circle, the instatement and addition/decrement doesn't occur inside the circle proclamation in while circle.

It is otherwise called the passage controlled circle since the condition is checked toward the beginning of the circle. On the off chance that the condition is valid, the circle body will be executed; any other way, the assertions after the circle will be executed.

The grammar of the while circle is given underneath.

```
while(condition)
{
//looping statements
}
```

The flow chart for the while loop is given in the following image.



Consider the following example.

Calculation .java

```
public class Calculation {
public static void main(String[] args) {
// TODO Auto-generated method stub
int i = 0;
System.out.println("Printing the list of first 10 even numbers \n");
while(i<=10)
 {
System.out.println(i);
i = i + 2;
}
}
}
```

Output:

Printing the list of first 10 even numbers

0

2

4

6

8

10

3.4.4. Java do-while loop

The do-while circle really takes a look at the condition toward the finish of the circle subsequent to executing the circle explanations. At the point when the quantity of cycle isn't known and we need to execute the circle once, we can utilize do-while circle.

It is otherwise called the exit-controlled circle since the condition isn't checked ahead of time. The sentence structure of the do-while circle is given beneath.

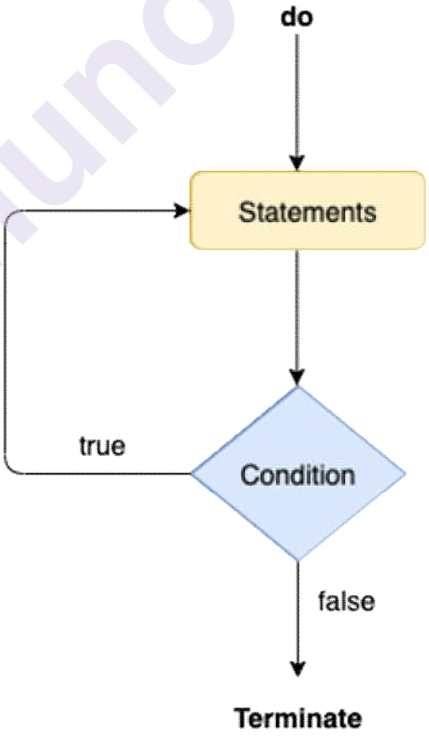
do

{

//statements

} **while** (condition);

The flow chart of the do-while loop is given in the following image.



Consider the following example to understand the functioning of the do-while loop in Java.

1) Calculation.java

```
public class Calculation
 {
public static void main(String[] args)
 {
// TODO Auto-generated method stub
int i = 0;
System.out.println("Printing the list of first 10 even numbers \n");
do
{
System.out.println(i);
i = i + 2;
}while(i<=10);
}
}
```

Output:

```
Printing the list of first 10 even numbers
0
2
4
6
8
10
```

3.4.5. Jump Statements

Hop articulations are utilized to move the control of the program to the particular assertions. At the end of the day, bounce articulations move the execution control to the next piece of the program. There are two kinds of hop proclamations in Java, i.e., break and proceed.

3.4.5.1. Java break statement

As the name recommends, the break proclamation is utilized to break the current progression of the program and move the control to the following assertion outside a circle or switch explanation. Be that as it may, it breaks just the inward circle on account of the settled circle.

The break articulation can't be utilized autonomously in the Java program, i.e., it must be composed inside the circle or switch explanation.

The break statement example with for loop

Consider the following example in which we have used the break statement with the for loop.

1) BreakExample.java

```
public class Break Example
 {
```

```
 public static void main(String[] args)
 {
// TODO Auto-generated method stub
for(int i = 0; i<= 10; i++)
{
System.out.println(i);
if(i==6) {
break;
}
}
}
}
```

Output:

```
0
1
2
3
4
5
6
```

break statement example with labeled for loop

1) Calculation.java

```
public class Calculation {
public static void main(String[] args)
 {
// TODO Auto-generated method stub
a:
for(int i = 0; i<= 10; i++) {
b:
for(int j = 0; j<=15;j++) {
c:
for (int k = 0; k<=20; k++) {
System.out.println(k);
if(k==5) {
break a;
}
}
}
}
}
}
```

Output:

0

1

2

3

4

5

3.4.5.2. Java continue statement

Unlike break statement, the continue statement doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

Consider the following example to understand the functioning of the continue statement in Java.

```
public class Continue Example
 {
  public static void main(String[] args)
{
// TODO Auto-generated method stub

for(int i = 0; i<= 2; i++) {

for (int j = i; j<=5; j++) {

if(j == 4)
{

continue;
}
System.out.println(j);
}
}
}
}
```

Output:

0

1

2

3

5
1
2
3
5
2
3
5

3.5. SUMMARY:

- In this Chapter, we covered control flow statement
- In Decision-Making statements, we learned if, if-else,if-else-if ladder, Nested if-statement, Switch Statement
- In this chapter we study about Looping statement like for loop, for-each loop, while loop, do-while loop, also jump statement break statement, continue statement.

3.6. LIST OF REFERENCES

1. Java 2: The Complete Reference, Fifth Edition, Herbert Schildt, Tata McGraw Hill.
2. An Introduction to Object oriented Programming with JAVA, C THOMAS WU3. www.javatpoint.com

3.7. QUESTION

1) What mean by Decision control statement?
2) Explain If Else statement with suitable example.
3) Explain Nested if statement with suitable example.
4) Explain loop control statement
5) What is mean by switch control statement with example.
6) Explain loop control statement with example.

❖❖❖❖

4

CLASSES

Unit Structure

4.1 OBJECTIVE

In this lesson of Java Tutorial, you will learn...

- How to create class
- How to create method
- How to create constructor

4.2 CLASS

Definition: A class is a collection of objects of similar type. Once a class is defined, any number of objects can be produced which belong to that class.

Class Declaration

classclassname

{

…

ClassBody

…

}

Objects are instances of the Class. Classes and Objects are very much related to each other. Without objects you can't use a class.

A general class declaration:

```
class name1
{
        //public variable declaration
        voidmethodname()
        {
                //body of method…
                //Anything
        }
}
```

Now following example shows the use of method.

```
class Demo
{
        privateintx,y,z;
        public void input()
        {
                x=10;
                y=15;
        }
        public void sum()
        {
                z=x+y;
        }
        public void print_data()
        {
                System.out.println(“Answer is =” +z);
        }
        public static void main(String args[])
        {
                Demo object=new Demo();
                object.input();
                object.sum();
                object.print_data();
        }
}
```

In program,

```
Demo object=new Demo();
object.input();
object.sum();
object.print_data();
```

In the first line we created an object.

The three methods are called by using the dot operator. When we call a method the code inside its block is executed.

The dot operator is used to call methods or access them.

4.2.1 Creating "main" in a separate class

We can create the main method in a separate class, but during compilation you need to make sure that you compile the class with the "main" method.

```
class Demo
{
        privateintx,y,z;
        public void input() {
                x=10;
                y=15;
        }
        public void sum()
        {
                z=x+y;
        }
        public void print_data()
        {
                System.out.println("Answer is =" +z);
        }
}
classSumDemo
{
        public static void main(String args[])
        {
                Demo object=new Demo();
                object.input();
                object.sum();
                object.print_data();
        }
}
```

Use of dot operator

We can access the variables by using dot operator.

Following program shows the use of dot operator.

```
classDotDemo
{
        intx,y,z;
```

```
        public void sum(){
                z=x+y;
        }
        public void show(){
                System.out.println("The Answer is "+z);
        }
}
class Demo1
{
        public static void main(String args[]){
                DotDemo object=new DotDemo();
                DotDemo object2=new DotDemo();
                object.x=10;
                object.y=15;
                object2.x=5;
                object2.y=10;
                object.sum();
                object.show();
                object2.sum();
                object2.show();
}}
```

output :

```
C:\cc>javac Demo1.java
C:\cc>java Demo1
The Answer is 25
The Answer is 15
```

- **Instance Variable**

All variables are also known as instance variable. This is because of the fact that each instance or object has its own copy of values for the variables. Hence other use of the "dot" operator is to initialize the value of variable for that instance.

4.2.2 Methods with parameters

Following program shows the method with passing parameter.

```
classprg
{
        int n,n2,sum;
        public void take(intx,int y)
        {
                n=x;
                n2=y;
```

```
        }
        public void sum()
        {
                sum=n+n2;
        }
        public void print()
        {
                System.out.println("The Sum is"+sum);
        }
}
class prg1
{
        public static void main(String args[])
        {
                prgobj=new prg();
                obj.take(10,15);
                obj.sum();
                obj.print();
        }
}
```

4.2.3 Methods with a Return Type

At the point when technique return some worth that is the sort of that strategy. For Instance: a few techniques are with boundary yet that strategy didn't return any worth that implies kind of technique is void. Furthermore, in the event that strategy return whole number worth, the sort of technique is a whole number.

Following program shows the method with their return type.

```
class Demo1
{
int n,n2;
public void take( intx,int y)
{
        n=x;
        n=y;
}
publicintprocess()
{
        return (n+n2);
}
}
```

```
Class prg
{
        public static void main(String args[])
        {
                int sum;
                Demo1 obj=new Demo1();
                obj.take(15,25);
                sum=obj.process();
                System.out.println("The sum is"+sum);
        }
}
```

Output:

The sum is25

4.2.4 Method Overloading

Method overloading means method name will be same but each method should be different parameter list.

```
class prg1
{
        int x=5,y=5,z=0;
        public void sum()
        {
                z=x+y;
                System.out.println("Sum is "+z);
        }
        public void sum(inta,int b)
        {
                x=a;
                y=b;
                z=x+y;
                System.out.println("Sum is "+z);
        }
        Public intsum(int a)
        {
                x=a;
                z=x+y;
                return z;
        }
}
class Demo
{
```

```
        public static void main(String args[])
        {
                prg1obj=new prg1();
                obj.sum();
                obj.sum(10,12);
                System.out.println(+obj.sum(15));
        }
}
```

Output:

sum is 10
sum is 22
27

4.2.5 Passing Objects as Parameters

Objects can even be passed as parameters.

```
class para123
{
        int n,n2,sum,mul;
        public void take(intx,int y)
        {
                n=x;
                n2=y;
        }
        public void sum()
        {
                sum=n+n2;
                System.out.println("The Sum is"+sum);
        }
        public void take2(para123 obj)
        {
                n=obj.n;
                n2=obj.n2;
        }
        public void multi()
        {
                mul=n*n2;
                System.out.println("Product is"+mul);
        }
}
classDemoPara
{
```

```
        public static void main(String args[])
        {
                para123ob=new para123();
                ob.take(3,7);
                ob.sum();
                ob.take2(ob);
                ob.multi();
        }
}
```

Output:

```
C:\cc>javac DemoPara.java
C:\cc>java DemoPara
The Sum is10
Product is21
```

We have defined a method "take2" that declares an object named obj as parameter. We have passed ob to our method. The method "take2" automatically gets 3,7 as values for n and n2.

4.2.6 Passing Values to methods and Constructor:

These are two different ways of supplying values to methods.

Classified under these two titles -

1.Pass by Value

2.Pass by Address or Reference

 Pass by Value-When we pass a data type like int, float or any other datatype to a method or some constant values like(15,10). They are all passed by value. A copy of variable's value is passed to the receiving method and hence any changes made to the values do not affect the actual variables.

```
class Demopbv
{
int n,n2;
public void get(intx,int y)
{
x=x*x; //Changing the values of passed arguments
y=y*y; //Changing the values of passed arguments
}
}
class Demo345
{
        public static void main(String args[])
        {
```

```
                inta,b;
                a=1;
                b=2;
                System.out.println("Initial Values of a & b "+a+" "+b);
                Demopbvobj=new Demopbv();
                obj.get(a,b);
                System.out.println("Final Values "+a+" "+b);
        }
}
```

Output:

C:\cc>javac Demo345.java

C:\cc>java Demo345

Initial Values of a & b 1 2

Final Values 1 2

 Pass by Reference

Objects are always passed by reference. When we pass a value by reference, the reference or the memory address of the variables is passed. Thus any changes made to the argument causes a change in the values which we pass.

Demonstrating Pass by Reference---

```
class Pass_by_Ref
{
        int n,n2;
        public void get(inta,int b)
        {
                n=a;
                n2=b;
        }
        public void doubleit(pass_by_ref temp)
        {
                temp.n=temp.n*2;
                temp.n2=temp.n2*2;
        }
}
class apply7
{
        public static void main(String args[])
        {
                int x=5,y=10;
                pass_by_refobj=new pass_by_ref();
                obj.get(x,y); //Pass by Value
```

```
                System.out.println("Initial Values are-- ");
                System.out.println(+obj.n);
                System.out.println(+obj.n2);
                obj.doubleit(obj); //Pass by Reference
                System.out.println("Final Values are");
                System.out.println(+obj.n);
                System.out.println(+obj.n2);
        }
}
```

4.2.7 Abstract Classes

Definition: An abstract class is a class that is declared as abstract. It may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclass. An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

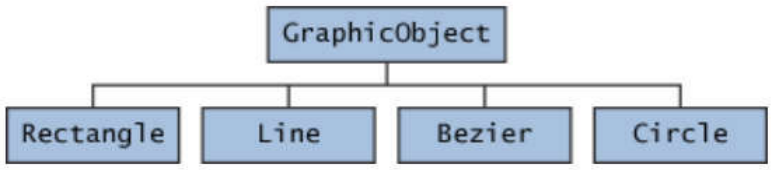
abstract void studtest (introllno, double test fees);

If a class includes abstract methods, the class itself must be declared abstract, as in:

```
public abstract class Graphic Object
{
// declare fields
// declare non-abstract methods
abstract void draw();
}
```

When an abstract class is subclass, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, the subclass must also be declared abstract.

For example: In an object-oriented drawing application, you can draw circles, rectangles, lines, Bezier curves, and many other graphic objects. These objects all have certain states (for example: position, orientation, line color, fill color) and behaviors (for example: move To, rotate, resize, draw) in common. Some of these states and behaviors are the same for all graphic objects—for example: position, fill color, and move To. Others require different implementations—for example, resize or draw. All Graphic Objects must know how to draw or resize themselves; they just differ in how they do it. This is a perfect situation for an abstract super class. You can take advantage of the similarities and declare all the graphic objects to inherit from the same abstract parent object—for example, Graphic Object, as shown in the following figure.



How to implement above diagram concept with source code:

```
abstract class Graphic Object
{
int x, y;
...
Voidmove To (intnew X, intnew Y)
{
...
}
abstract void draw();
abstract void resize();
}
```

Each non-abstract subclass of Graphic Object, such as Circle and Rectangle, must provide implementations for the draw and resize methods:

```
class Circle extends Graphic Object {
void draw() {
...
}
void resize() {
...
}
}
class Rectangle extends Graphic Object {
void draw() {
...
}
void resize() {
...
}
}
```

Abstract classes are those which can be used for creation of objects. However their methods and constructors can be used by the child or extended class. The need for abstract classes is that you can generalize the super class from which child classes can share its methods. The subclass of an abstract class which can create an object is called as "concrete class".

For example:

```
abstract class A
{
        abstract void method1();
        void method2()
        {
        System.out.println("this is real method");
        }
}
class B extends A
{
        void method1()
        {
        System.out.println("B is execution of method1");
        }
}
class demo
{
        public static void main(String arg[])
        {
                B b=new B();
                b.method1();
                b.method2();
        }
}
```


4.2.8 Extending the class:

Inheritance allows to subclass or child class to access all methods and variables of parent class.

Syntax:

```
Class subclass name extends super class name
{
        Varables;
        Methods;
        …..
}
```

For example: calculate area and volume by using In hertance.

```
class data
{
        int l;
        int b;
```

```
        data(int c, int d)
        {
                l=c;
                b=d;
        }
        int area( )
        {
                return(l*b);
        }
}
class data2 extends data
{
        int h;
        data2(intc,int d, int a)
        {
                super(c,d);
                h=a;
        }
        int volume()
        {
                return(l*b*h);
        }
}
class DataDemo
{
        public static void main(String args[])
        {
                data2 d1=new data2(10,20,30);
                int area1=d1.area(); //superclass method
                int volume1=d1.volume( );// subclass method
                System.out.println("Area="+area1);
                System.out.println("Volume="+volume1);
        }
}
```

Output:

```
C:\cc>javac dataDemo.java
C:\cc>java dataDemo
Area=200
Volume=6000
```

"Is A" - is a subclass of a super class (ex: extends)

"Has A" - has a reference to (ex: variable, ref to object).

o Access Control –

Away to limit the access others have to your code.

 Same package - can access each others' variables and methods, except for private members.

 Outside package - can access public classes. Next, can access members that are public. Also, can access protected members if the class is a subclass of that class.

Same package - use package keyword in first line of source file, or no package keyword and in same directory.

o Keywords -

1. public - outside of package access.
2. [no keyword] - same package access only.
3. protected - same package access. Access if class is a subclass of, even if in another package.
4. private - same class access only.

4.3 SUMMARY:

In this unit, we learn the concept of class and how to create method and how to pass parameters by value and by reference and method overloading with example. In this unit, we also learn the concept of inheritance.

4.4 LIST OF REFERENCES

1. Java 2: The Complete Reference, Fifth Edition, Herbert Schildt, Tata McGraw Hill.
2. An Introduction to Object oriented Programming with JAVA, C THOMAS WU
3. www.javatpoint.com

4.5. QUESTION

1. Explain class in detail with suitable example.
2. Explain Methods with a Return Type with suitable example.
3. What is method overloading explain with suitable example.
4. Write a short note on abstract class.

5

CONSTRUCTORS

Unit Structure

5.1. INTRODUCTION

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java:

1) Default constructor

2) Parameterized constructor.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

5.1.1. Rules for creating constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

5.1.2. Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

<class_name>(){}

Example of default constructor

1. Java Default Constructor

```
//Java Program to create and call a default constructor
class Bike1{
//creating a default constructor
Bike1(){System.out.println("Bike is created");}
//main method
public static void main(String args[]){
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

Output:

Bike is created

2. Example of default constructor that displays the default values

```
//Let us see another example of default constructor
//which displays the default values
class Student3{
int id;
String name;
//method to display the value of id and name
void display(){System.out.println(id+" "+name);}

public static void main(String args[]){
//creating objects
Student3 s1=new Student3();
Student3 s2=new Student3();
//displaying values of the object
```

```
s1.display();
s2.display();
}
}
```

Output:

```
0 null
0 null
```

In the above class, you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

5.1.3. Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor. The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
//Java Program to demonstrate the use of the parameterized constructor.
class Student4{
int id;
   String name;
   //creating a parameterized constructor
Student4(inti,String n){
id = i;
name = n;
   }
   //method to display the values
void display(){System.out.println(id+" "+name);}
public static void main(String args[]){
   //creating objects and passing values
   Student4 s1 = new Student4(111,"Karan");
   Student4 s2 = new Student4(222,"Aryan");
   //calling method to display the values of object
s1.display();
s2.display();
```

```
  }
}
```

Output:

111 Karan

222 Aryan

5.1.4. Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Example of Constructor Overloading

```
//Java program to overload constructors
class Student5{
int id;
   String name;
int age;
   //creating two arg constructor
Student5(inti,String n){
id = i;
name = n;
   }
   //creating three arg constructor
Student5(inti, Stringn, int a){
id = i;
name = n;
age=a;
   }
void display(){System.out.println(id+" "+name+" "+age);}

public static void main(String args[]){
   Student5 s1 = new Student5(111,"Karan");
   Student5 s2 = new Student5(222,"Aryan",25);
s1.display();
s2.display();
  }
}
```

Output:

111 Karan 0

222 Aryan 25

5.1.5. Copy Constructor

There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in Java. They are:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

In this example, we are going to copy the values of one object into another using Java constructor.

```
//Java program to initialize the values from one object to another object.
class Student6{
int id;
   String name;
   //constructor to initialize integer and string
Student6(inti,String n){
id = i;
name = n;
   }
   //constructor to initialize another object
Student6(Student6 s){
id = s.id;
name =s.name;
   }
void display(){System.out.println(id+" "+name);}

public static void main(String args[]){
   Student6 s1 = new Student6(111,"Karan");
   Student6 s2 = new Student6(s1);
s1.display();
s2.display();
  }
}
```

Output:

111 Karan

111 Karan

5.1.6. Super Keyword

The super keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

super is used to refer immediate parent class instance variable.

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

```
class Animal{
String color="white";
}
class Dog extends Animal{
String color="black";
voidprintColor(){
System.out.println(color);//prints color of Dog class
System.out.println(super.color);//prints color of Animal class
}
}
class TestSuper1{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
}}
```

Output:

```
black
white
```

In the above example, Animal and Dog both classes have a common property color. If we print color property, it will print the color of current class by default. To access the parent property, we need to use super keyword.

5.1.7.static keyword

The static keyword in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested classes. The static keyword belongs to the class than an instance of the class.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested class

1) Java static variable

If you declare any variable as static, it is known as a static variable.

- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

Advantages of static variable

It makes your program memory efficient (i.e., it saves memory).

6.1.8. Understanding the problem without static variable

```
class Student{
introllno;
   String name;
   String college="ITS";
}
```

Suppose there are 500 students in my college, now all instance data members will get memory each time when the object is created. All students have its unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all objects. If we make it static, this field will get the memory only once.

1) Example of static variable

```
//Java Program to demonstrate the use of static variable
class Student{
introllno;//instance variable
  String name;
static String college ="ITS";//static variable
  //constructor
Student(int r, String n){
rollno = r;
name = n;
  }
  //method to display the values
```

```
void display (){System.out.println(rollno+" "+name+" "+college);}
}
//Test class to show the values of objects
public class TestStaticVariable1{
public static void main(String args[]){
 Student s1 = new Student(111,"Karan");
 Student s2 = new Student(222,"Aryan");
 //we can change the college of all objects by the single line of code
 //Student.college="BBDIT";
s1.display();
s2.display();
 }
}
```

Output:

```
111 Karan ITS
222 Aryan ITS
```

2) Program of counter by static variable

As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

```
//Java Program to illustrate the use of static variable which
//is shared with all objects.
class Counter2{
staticint count=0;//will get memory only once and retain its value

Counter2(){
count++;//incrementing the value of static variable
System.out.println(count);
}

public static void main(String args[]){
//creating objects
Counter2 c1=new Counter2();
Counter2 c2=new Counter2();
Counter2 c3=new Counter2();
}
}
```

Output:

1

2

3

5.1.9. Java static method

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

Example of static method

```
//Java Program to demonstrate the use of a static method.
class Student{
introllno;
    String name;
static String college = "ITS";
    //static method to change the value of static variable
static void change(){
college = "BBDIT";
    }
    //constructor to initialize the variable
Student(int r, String n){
rollno = r;
name = n;
    }
    //method to display values
void display(){System.out.println(rollno+" "+name+" "+college);}
}
//Test class to create and display the values of object
public class TestStaticMethod{
public static void main(String args[]){
Student.change();//calling change method
   //creating objects
   Student s1 = new Student(111,"Karan");
```

```
    Student s2 = new Student(222,"Aryan");
    Student s3 = new Student(333,"Sonoo");
    //calling display method
s1.display();
s2.display();
s3.display();
    }
}
```

Output:

```
111 Karan BBDIT
222 Aryan BBDIT
333 Sonoo BBDIT
```

5.2. GARBAGE COLLECTION

Java applications obtain objects in memory as needed. It is the task of garbage collection (GC) in the Java virtual machine (JVM) to automatically determine what memory is no longer being used by a Java application and to recycle this memory for other uses. Because memory is automatically reclaimed in the JVM, Java application developers are not burdened with having to explicitly free memory objects that are not being used. The GC operation is based on the premise that most objects used in the Java code are short-lived and can be reclaimed shortly after their creation. Because unreferenced objects are automatically removed from the heap memory, GC makes Java memory-efficient.

Garbage collection frees the programmer from manually dealing with memory deallocation. As a result, certain categories of application program bugs are eliminated or substantially reduced by GC:

- Dangling pointer bugs, which occur when a piece of memory is freed while there are still pointers to it, and one of those pointers is dereferenced. By then the memory may have been reassigned to another use with unpredictable results.
- Double free bugs, which occur when the program tries to free a region of memory that has already been freed and perhaps already been allocated again.
- Certain kinds of memory leaks, in which a program fails to free memory occupied by objects that have become unreachable, which can lead to memory exhaustion.

5.2.1. Two types of garbage collection

- **A minor or incremental garbage collection** is said to have occurred when unreachable objects in the young generation heap memory are removed.
- **A major or full garbage collection** is said to have occurred when the objects that survived the minor garbage collection and copied into the old generation or permanent generation heap memory are removed. When compared to young generation, garbage collection happens less frequently in old generation.

To free up memory, the JVM must stop the application from running for at least a short time and execute GC. This process is called "**stop-the-world**." This means all the threads, except for the GC threads, will stop executing until the GC threads are executed and objects are freed up by the garbage collector.

Modern GC implementations try to minimize blocking "stop-the-world" stalls by doing as much work as possible on the background (i.e. using a separate thread), for example marking unreachable garbage instances while the application process continues to run.

Garbage collection consumes CPU resources for deciding which memory to free. Various garbage collectors have been developed over time to reduce the application pauses that occur during garbage collection and at the same time to improve on the performance hit associated with garbage collection.

The traditional Oracle HotSpot JVM has four ways of performing the GC activity:

- **Serial** where just one thread executed the GC
- **Parallel** where multiple minor threads are executed simultaneously each executing a part of GC
- **Concurrent Mark Sweep (CMS),** which is similar to parallel, also allows the execution of some application threads and reduces the frequency of stop-the-world GC
- **G1** which is also run in parallel and concurrently but functions differently than CMS

Many JVMs, such as Oracle HotSpot, JRockit, OpenJDK, IBM J9, and SAP JVM, use stop-the-world GC techniques. Modern JVMs like Azul Zing use **Continuously Concurrent Compacting Collector (C4)**, which eliminates the stop-the-world GC pauses that limit scalability in the case of conventional JVMs.

5.3. SUMMARY

In this chapter we study about constructor, Rules for creating constructor, Default Constructor, Constructor Overloading, Copy Constructor.

Also we study about Super Keyword, static keyword, static method, Last point of this chapter is Garbage Collection and types of Garbage Collection

5.4 LIST OF REFERENCES

1. Java 2: The Complete Reference, Fifth Edition, Herbert Schildt, Tata McGraw Hill.
2. An Introduction to Object oriented Programming with JAVA, C THOMAS WU
3. www.javatpoint.com

5.5 QUESTIONS

1. Write a short note on constructor.
2. Explain default contractors with suitable Example.
3. Explain Parameterized constructor with suitable example.
4. Explain static keyword.
5. write a short note on garbage collection

Unit III

6

INHERITANCE

Unit Structure

6.0 OBJECTIVE

To understand the concept of Inheritance, Types of Inheritance, Abstract class, Difference between abstract class and interface, etc…

6.1.INTRODUCTION INHERITANCE

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

6.1.1. Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

6.1.2. Terms used in Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

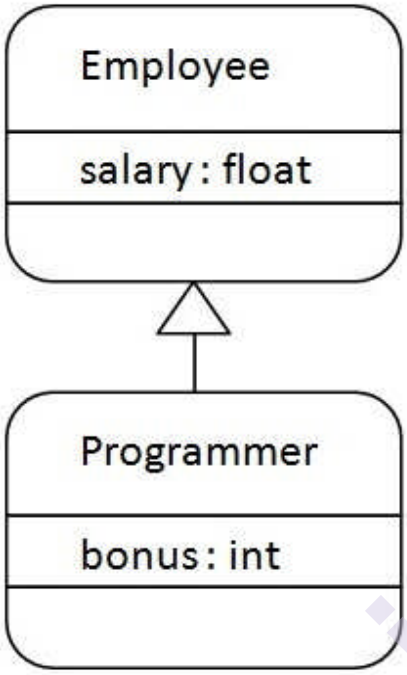
6.1.3.The syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

Java Inheritance Example



As displayed in the above figure, Programmer is the subclass and Employee is the superclass. The relationship between the two classes is **Programmer IS-A Employee**. It means that Programmer is a type of Employee.

```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

Output:

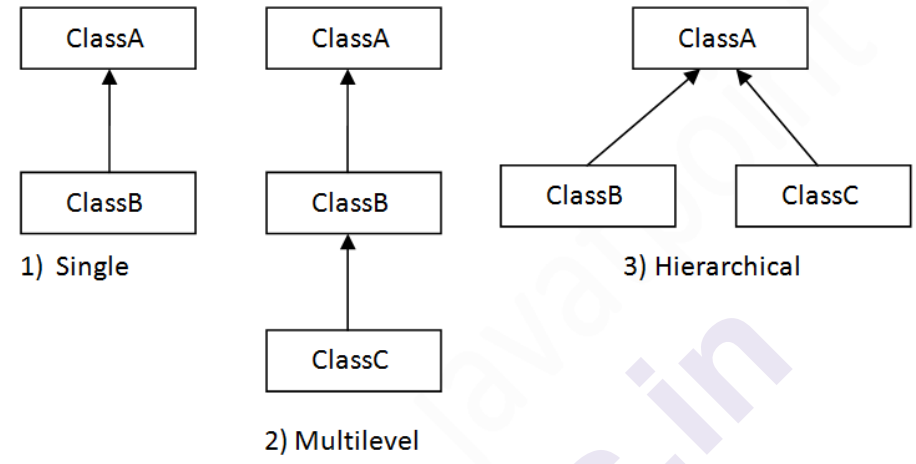
```
Programmer salary is:40000.0
Bonus of programmer is:10000
```

In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

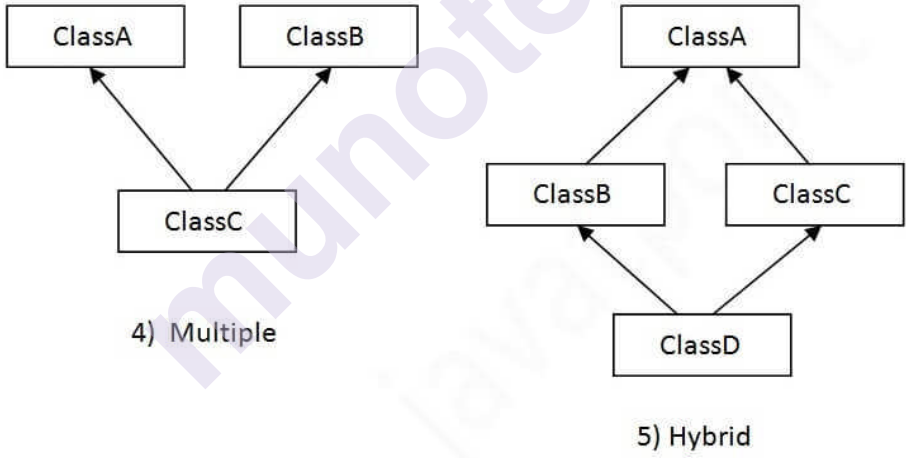
6.2. TYPES OF INHERITANCE IN JAVA

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



When one class inherits multiple classes, it is known as multiple inheritance. For Example:



.

6.2.1. Single Inheritance Example

When a class inherits another class, it is known as a *single inheritance*. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

File: TestInheritance.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
```

```
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

Output:

```
barking...
eating...
```

6.3. MULTILEVEL INHERITANCE EXAMPLE

When there is a chain of inheritance, it is known as *multilevel inheritance*. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

File: TestInheritance2.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Output:

```
weeping...
barking...
eating...
```

6.4. HIERARCHICAL INHERITANCE

When two or more classes inherits a single class, it is known as *hierarchical inheritance*. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

File: TestInheritance3.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Output:

```
meowing...
eating...
```

6.5. MULTIPLE INHERITANCE IS NOT SUPPORTED IN JAVA

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

```
class A{
void msg(){System.out.println("Hello");}
```

```
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

 public static void main(String args[]){
   C obj=new C();
   obj.msg();//Now which msg() method would be invoked?
}
}
```

Output:

Compile Time Error

6.6. ABSTRACT CLASS IN JAVA

A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

Before learning the Java abstract class, let's understand the abstraction in Java first.

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

6.6.1. Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.

- It can have final methods which will force the subclass not to change the body of the method.

Example of abstract class
abstract class A{}

A method which is declared as abstract and does not have implementation is known as an abstract method.

Example of abstract method
abstract void printStatus();//no method body and abstract

Example of Abstract class that has an abstract method

In this example, Bike is an abstract class that contains only one abstract method run. Its implementation is provided by the Honda class.

```
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```

Output:
running safely

example of Abstract class in java
File: TestBank.java

```
abstract class Bank{
abstract int getRateOfInterest();
}
class SBI extends Bank{
int getRateOfInterest(){return 7;}
}
class PNB extends Bank{
int getRateOfInterest(){return 8;}
}

class TestBank{
public static void main(String args[]){
Bank b;
b=new SBI();
```

```
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
b=new PNB();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
}}
```

Output:

Rate of Interest is: 7 %

Rate of Interest is: 8 %

6.7. ABSTRACT CLASS HAVING CONSTRUCTOR, DATA MEMBER AND METHODS

An abstract class can have a data member, abstract method, method body (non-abstract method), constructor, and even main() method.

File: TestAbstraction2.java

```
//Example of an abstract class that has abstract and non-abstract methods
 abstract class Bike{
   Bike(){System.out.println("bike is created");}
   abstract void run();
   void changeGear(){System.out.println("gear changed");}
 }
//Creating a Child class which inherits Abstract class
 class Honda extends Bike{
 void run(){System.out.println("running safely..");}
 }
//Creating a Test class which calls abstract and non-abstract methods
 class TestAbstraction2{
 public static void main(String args[]){
  Bike obj = new Honda();
  obj.run();
  obj.changeGear();
 }
}
```

Output:

```
bike is created
      running safely..
      gear changed
```

6.8. INTERFACE IN JAVA

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

- Java Interface also **represents the IS-A relationship**.
- It cannot be instantiated just like the abstract class.
- we can have **default and static methods** in an interface.
- we can have **private methods** in an interface.

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

How to declare an interface?

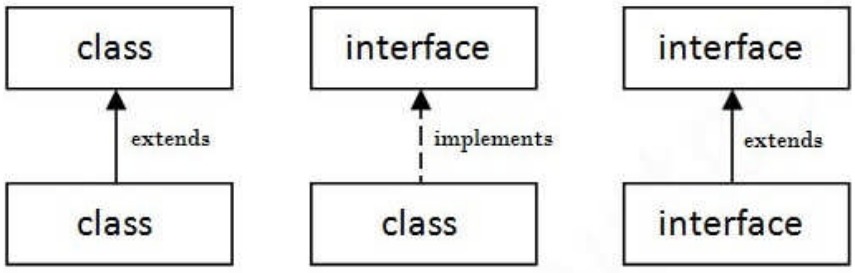
An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

Syntax:

```
interface <interface_name>{

   // declare constant fields
   // declare methods that abstract
   // by default.
}
```

6.9. THE RELATIONSHIP BETWEEN CLASSES AND INTERFACES

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



Java Interface Example

In this example, the Printable interface has only one method, and its implementation is provided in the A6 class.

```
interface printable{
void print();
}
class A6 implements printable{
public void print(){System.out.println("Hello");}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

Output:
Hello

Java Interface Example: Bank

Let's see another example of java interface which provides the implementation of Bank interface.

File: TestInterface2.java

```
interface Bank{
float rateOfInterest();
}
class SBI implements Bank{
public float rateOfInterest(){return 9.15f;}
}
class PNB implements Bank{
public float rateOfInterest(){return 9.7f;}
}
class TestInterface2{
public static void main(String[] args){
Bank b=new SBI();
System.out.println("ROI: "+b.rateOfInterest());
}}
```

Output:
ROI: 9.15

6.10 MULTIPLE INHERITANCE IN JAVA BY INTERFACE

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.

Multiple Inheritance in Java

```
interface Printable{
void print();
}
interface Showable{
void show();
}
class A7 implements Printable,Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 }
}
```

Output:

Hello
Welcome

6.11. INTERFACE INHERITANCE

A class implements an interface, but one interface extends another interface.

```
 interface Printable{
 void print();
 }
```

```
interface Showable extends Printable{
void show();
}
class TestInterface4 implements Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
TestInterface4 obj = new TestInterface4();
obj.print();
obj.show();
 }
}
```

Output:

Hello
Welcome

6.12. DEFAULT METHOD IN INTERFACE

We can have method body in interface. But we need to make it default method. Let's see an example:

File: TestInterfaceDefault.java

```
interface Drawable{
void draw();
default void msg(){System.out.println("default method");}
}
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class TestInterfaceDefault{
public static void main(String args[]){
Drawable d=new Rectangle();
d.draw();
d.msg();
}}
```

Output:

drawing rectangle
default method

6.13. STATIC METHOD IN INTERFACE

We can have static method in interface. Let's see an example:

File: TestInterfaceStatic.java

```
interface Drawable{
void draw();
static int cube(int x){return x*x*x;}
}
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}

class TestInterfaceStatic{
public static void main(String args[]){
Drawable d=new Rectangle();
d.draw();
System.out.println(Drawable.cube(3));
}}
```

Output:

```
drawing rectangle
27
```

6.14. DIFFERENCE BETWEEN ABSTRACT CLASS AND INTERFACE

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.

4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable {
	void draw();
	}

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

6.15. SUMMARY:

In this chapter we discuss the detailed concept of Inheritance, types of inheritance, Super keyword, Abstract classes, how to use interfaces, Static Method in Interface, Static Method in Interface, Difference between abstract class and interface, etc..

6.16. QUESTIONS:

1) Explain the term used in Inheritance.
2) What are the types of Inheritance?
3) Explain multiple inheritances with satiable example.
4) Why multiple inheritances is not supported in java.
5) Write a short note on abstract class.
6) Write a Difference between abstract class and interface.

6.17 REFERENCE:

1) Java™: The Complete Reference, Seventh Edition
2) www.javatpoint.com

7

PACKAGES

Unit Structure

7.0 OBJECTIVE

In this chapter, you will be going to learn:

- Concept of package
- Built-In packages available
- Creating and using user defined packages

7.1 INTRODUCTION

Package in java is a mechanism to encapsulate a group of classes, sub packages and interfaces. Package in java can be categorized in two forms, built- in package and user defined package. Packages are used for:

- Preventing naming conflicts.
- Making searching, usage of classes, interfaces, enumerations and annotations easier.
- Package provides access protection.

7.2 CREATING PACKAGES

The package keyword is used to create a package in java. We can add multiple classes and interfaces to a created package by using package name at the top of the program and saving it in package directory.

Example:

1) **//SAVE AS Addition.java**

```
package mypack;
public class Addition
{
public void add(double a,double b)
{
        System.out.println("Addition :" + (a+b));
 }
}
```

Compiling Java Package:

Syntax: javac-d directory javafilename

javac -d . Addition.java

Here, -d specifies the destination where to put the generated class file. To keep the package in the same directory , use .(dot).

2) // save as Factorial.java

```
package mypack;
import java.util.*;
public class Factorial
{
  int i,fact=1;
public void fact(int number)
{
  for(i=1;i<=number;i++)
  {
     fact=fact*i;
  }
System.out.println("Factorial of "+number+" is: "+fact);
 }
}
```

Compiling Java Package:

javac -d .Factorial.java

7.3 DEFAULT PACKAGES

Package	Description
java.io	Provides system input and output through data stream, serialization and the file system. Example, Input Stream, Output Stream, Print Writer, Writer etc.
java.lang	Provides classes that are fundamental to the design of the java programming language. Example, Object, System, Runnable etc.
java.sql	Provides the API for accessing and processing data stored in a data source. Example, Statement, ResultSet, Drier Manager etc.
java.time	The main API for Dates, times, instances and durations.
java.util	Contains collection Framework.
java.applet	Contains classes for creating applet
java.awt	Contains the classes for implementing the components for graphical user interface.
java.net	Contains the classes for supporting networking operations.

7.4 IMPORTING PACKAGES

There are two ways to access the package from outside the package.

1) Using packagename.*

Syntax: import packagename.*

Using this syntax, all the classes and interfaces of this package will be accessible but not subpackage.

Example: import mypack.*;

2) Using packagename.classname

Syntax import packagename.classname

If you import package.classname then only declared class of this package will be accessible.

Example: import mypack.Factorial;
import mypack.addition;

7.5 USING A PACKAGE

Example:

```
import java.util.Scanner;
import mypack.Factorial;
import mypack.Addition;

class TestPackage
{
public static void main(String arg[])
{
        Factorial f=new Factorial();
        System.out.println("enter number to find out factorial ");
        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        f.fact(n);

        System.out.println("Enter 2 numbers: ");
        Addition a=new Addition();
        double p=sc.nextDouble();
        double q=sc.nextDouble();

        a.add(p,q);
}
}
```

Output:

```
enter number to find out factorial
5
Factorial of 5 is: 120
Enter 2 numbers:
5
6
Addition :11.0
```

7.6 SUMMARY

In this chapter, we learn concept of package, built in collection of packages available in java language, creating and accessing user defined packages.

7.7 LIST OF REFERENCES

Java, A Beginner's Guide, Eighth Edition, Herbert Schildt, McGraw Hill Publisher

Java: The Complete Reference, Eleventh Edition, Herbert Schildt, McGraw Hill Publisher

7.8 BIBLIOGRAPHY

- https://www.javatpoint.com/java-tutorial
- https://www.geeksforgeeks.org/java
- https://www.beginnersbook.com/java-tutorial-for-beginners-with-examples/
- http://www.programiz.com/java-programming/
- https://docs.oracle.com/javase/tutorial/

7.9 MODEL QUESTIONS

1. Which of these keywords is used to define packages in Java?
a) pkg
b) Pkg
c) package
d) Package

2. Which of these is a mechanism for naming and visibility control of a class and its content?
a) Object
b) Packages
c) Interfaces
d) None of the Mentioned.

3. Which of these access specifiers can be used for a class so that its members can be accessed by a different class in the different package?
a) Public
b) Protected
c) Private
d) No Modifier

4. Which of the following is the correct way of importing an entire package ‘pkg’?
a) import package.;
b) Import package;
c) import package.*;
d) Import package.*;

5. What is the maximum number of Java Class files that can be kept inside a single Java Package?
a) 8
b) 64
c) 128
d) Unlimited

6. When importing a package, the Class is actually importing ____.
a) Classes or Interfaces from the package
b) Constants
c) Methods
d) None of the above

Unit IV

8

ENUMERATIONS, ARRAYS

Unit Structure :

8.0 INTRODUCTION

An array is a data structure supported by all the programming languages that stores multiple values of the same type in a fixed length structure. Vector implements a dynamic array. It is similar to Array List.

In this chapter we will cover about Enumerations, Arrays . We will discuss about Vector. Then we will cover Operations with Array and Vector supported in Java. At the end we will discuss program of array and vector in Java language.

8.1 OBJECTIVES

After going through this unit, you should be able to:

- Know theory behind Array and Vector
- Write Array Program with its different Types
- Write Vector Program with its different Methods

8.2 JAVA ARRAY

- An array is a collection of similar type of elements that have a contiguousmemory location.
- **Java array** is an object which contains elements of a similar data type.

- It is a data structure where we store similar elements.
- We can store only a fixed set of elements in a Java array.
- Array in java is index - based, the first element of the array is stored at the 0 index.

8.2.0 Advantages

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data Efficiently.
- **Random access:** We can get any data located at an index position.

8.2.1 Disadvantages

- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

8.3 TYPES OF ARRAY IN JAVA

There are two types of array.
- Single Dimensional Array
- Multidimensional Array

8.3.0 Single Dimensional Array in Java

Syntax to Declare an Array in Java

```
1. dataType[] arr;
2. dataType []arr;
3. dataType arr[];
```

Instantiation of an Array in Java

```
1. array RefVar=new datatype[size];
```

Example of Java Array

Let's see the simple example of java array, where we are going to declare, instantiate,initialize and traverse an array.

```
class Testarray{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initializationa
[1]=20;
a[2]=70;
a[3]=90;
 a[4]=10;
//traversing array
```

```
for(int i=0;i<a.length;i++)     //length is the property of array
System.out.println(a[i]);
 }}
```

Output:

```
10
20
70
90
10
```

Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

```
int a[]={33,3,4,5,6};//declaration, instantiation and initialization
```

Let's see the simple example to print this array.

```
//Java Program to illustrate the use of declaration, instantiation
//and initialization of Java array in a single line
class Testarray1{
public static void main(String args[]){
int a[]={33,3,4,5,6};//declaration, instantiation and initialization
//printing array
for(int i=0;i<a.length;i++)//length is the property of array
    System.out.println(a[i]);
     }}
```

Output:

```
33
3
4
5
6
```

8.3.1 Multidimensional Array in Java

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in Java

```
1. dataType[][] arrayRefVar;
2. dataType [][]arrayRefVar;
3. dataType arrayRefVar[][];
4. dataType []arrayRefVar[];
```

Example to instantiate Multidimensional Array in Java

int[][] arr=**new int**[3][3];//3 row and 3 column

Example to initialize Multidimensional Array in Java

```
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;
```

Example of Multidimensional Java Array

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensionalarray.

```
//Java Program to illustrate the use of multidimensional array
class Testarray3
{
public static void main(String args[])
{
//declaring and initializing 2D array
     int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
    //printing 2D array
      for(int i=0;i<3;i++){
                for(int j=0;j<3;j++){
     System.out.print(arr[i][j]+" ");
     }
     System.out.println();
     }
    }}
```

Output:

1 2 3
2 4 5
4 4 5

8.4 JAVA- VECTOR CLASS

Vector implements a dynamic array. It is similar to Array List, but with two differences −

- Vector is synchronized.

- Vector contains many legacy methods that are not part of the collections framework.

Vector proves to be very useful if you don't know the size of the array in advance or you justneed one that can change sizes over the lifetime of a program.

Following is the list of constructors provided by the vector class.

Sr.No.	Constructor & Description
1	**Vector( )** This constructor creates a default vector, which has an initial size of 10.
2	**Vector(int size)** This constructor accepts an argument that equals to the required size, and creates a vector whose initial capacity is specified by size.
3	**Vector(Collection c)** This constructor creates a vector that contains the elements of collection c.
4	**Vector(int size, int incr)** This constructor creates a vector whose initial capacity is specified by size and whose increment is specified by incr. The increment specifies the number of elements to allocate each time that a vector is resized upward.

Apart from the methods inherited from its parent classes, Vector defines the following methods –

Sr.No.	Constructor & Description
1	**void add(int index, Object element)** Inserts the specified element at the specified position in this Vector.
2	**boolean add(Object o)** Appends the specified element to the end of this Vector.
3	**boolean addAll(Collection c)** Appends all of the elements in the specified Collection to the end of this Vector, in the order that they are returned by the specified Collection's Iterator.
4	**boolean addAll(int index, Collection c)** Inserts all of the elements in in the specified Collection into this Vectorat the specified position.

5	**void addElement(Object obj)** Adds the specified component to the end of this vector, increasing itssize by one.
6	**int capacity()** Returns the current capacity of this vector.
7	**void clear()** Removes all of the elements from this vector.
8	**Object clone()** Returns a clone of this vector.
9	**boolean contains(Object elem)** Tests if the specified object is a component in this vector.
10	**boolean containsAll(Collection c)** Returns true if this vector contains all of the elements in the specifiedCollection.
11	**void copy Into(Object[] an Array)** Copies the components of this vector into the specified array.
12	**Object elementAt(int index)** Returns the component at the specified index.
13	**Enumeration elements()** Returns an enumeration of the components of this vector.
14	**void ensure Capacity(int min Capacity)** Increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument.
15	**boolean equals(Object o)** Compares the specified Object with this vector for equality.
16	**Object firstElement()** Returns the first component (the item at index 0) of this vector.
17	**Object get(int index)** Returns the element at the specified position in this vector.
18	**int hash Code()** Returns the hash code value for this vector.
19	**int index Of(Object elem)** Searches for the first occurence of the given argument, testing forequality using the equals method.

20	**int index Of(Object elem, int index)** Searches for the first occurence of the given argument, beginning the search at index, and testing for equality using the equals method.
21	**void insert Element At (Object obj, int index)** Inserts the specified object as a component in this vector at thespecified index.
22	**boolean is Empty()** Tests if this vector has no components.
23	**Object last Element()** Returns the last component of the vector.
24	**int last IndexOf(Object elem)** Returns the index of the last occurrence of the specified object in thisvector.
25	**int last Index Of (Object elem, int index)** Searches backwards for the specified object, starting from the specifiedindex, and returns an index to it.
26	**Object remove(int index)** Removes the element at the specified position in this vector.
27	**boolean remove (Object o)** Removes the first occurrence of the specified element in this vector, If the vector does not contain the element, it is unchanged.
28	**boolean remove All(Collection c)** Removes from this vector all of its elements that are contained in thespecified Collection.
29	**void remove All Elements()** Removes all components from this vector and sets its size to zero.
30	**boolean remove Element(Object obj)** Removes the first (lowest-indexed) occurrence of the argument fromthis vector.
31	**void remove Element At (int index)** remove Element At(int index).
32	**protected void remove Range (int from Index, int to Index)** Removes from this List all of the elements whose index is between from Index, inclusive and to Index, exclusive.

33	**boolean retain All(Collection c)** Retains only the elements in this vector that are contained in thespecified Collection.
34	**Object set(int index, Object element)** Replaces the element at the specified position in this vector with thespecified element.
35	**void set Element At(Object obj, int index)** Sets the component at the specified index of this vector to be the
36	specified object.
37	**void set Size (int new Size)** Sets the size of this vector.
38	**int size()** Returns the number of components in this vector.
39	**List sub List (int from Index, int to Index)** Returns a view of the portion of this List between from Index, inclusive,and to Index, exclusive.
40	**Object[] toArray()** Returns an array containing all of the elements in this vector in thecorrect order.
41	**Object[] to Array(Object[] a)** Returns an array containing all of the elements in this vector in the correct order; the runtime type of the returned array is that of the specified array.
42	**String to String()** Returns a string representation of this vector, containing the Stringrepresentation of each element.

Example

The following program illustrates several of the methods supported by this

collection –

```
import java.util.*;
public class VectorDemo
{
        public static void  main(String args[]){
                // initial size is 3, increment is 2
                Vector v =new Vector(3,2);
                System.out.println("Initial size: "+ v.size());
```

```
        System.out.println("Initial capacity: "+ v.capacity());
        v.addElement(newInteger(1));
        v.addElement(newInteger(2));
        v.addElement(newInteger(3));
        v.addElement(newInteger(4));
        System.out.println("Capacity after four additions: "+
        v.capacity());
        v.addElement(newDouble(5.45));
        System.out.println("Current capacity: "+ v.capacity());
        v.addElement(newDouble(6.08));
        v.addElement(newInteger(7));
        System.out.println("Current capacity: "+ v.capacity());
        v.addElement(newFloat(9.4));
        v.addElement(newInteger(10));
        System.out.println("Current capacity: "+ v.capacity());
        v.addElement(newInteger(11));
        v.addElement(newInteger(12));
        System.out.println("First element:
        "+(Integer)v.firstElement());
        System.out.println("Last element:
        "+(Integer)v.lastElement());
        if(v.contains(newInteger(3))) System.out.println("Vector
        contains 3.");
        // enumerate the elements in the vector.
        Enumeration vEnum = v.elements();
        System.out.println("\nElements in vector:");
        while(vEnum.hasMoreElements())
        System.out.print(vEnum.nextElement()+" ");
        System.out.println();
    }
}
```

This will produce the following result –

Output :

```
Initial size: 0
Initial capacity: 3
Capacity after four additions: 5 Current capacity: 5
Current capacity: 7
Current capacity: 9
First element: 1
Last element: 12
Vector contains 3.
```

Elements in vector:
1 2 3 4 5.45 6.08 7 9.4 10 11 12

8.5 SAMPLE QUESTIONS

Q1. What is the need of Array ? Explain

Q2. What is Array ? Explain with Example.

Q3. What is Array ? what are the Types of Array ?

Q4. What are the Advantages of Vector Class over Java Array ?

Q5. What is vector ? Explain with Example.

Q6. Explain various methods vectors.

Q7. What is the difference between Array and Vector ?

9

MULTITHREADING

Unit Structure :

9.0 INTRODUCTION

A thread is single sequence of execution that can run independently in an application. This unit covers the very important concept of multithreading in programming. Uses of thread in programs are good in terms of resource utilization of the system on which application is running. Multithreaded programming is very useful in network and Internet applications development. In this unit you will learn what is multithreading, how thread works, how to write programs in Java using multithreading. Also, in this unit will be explained about thread-properties, synchronization, and interthread communication.

9.1 OBJECTIVES

After going through this unit, you will be able to:

- describe the concept of multithreading;
- explain the Java thread model;
- create and use threads in program;
- describe how to set the thread priorities;
- use the concept of synchronization in programming, and
- use inter-thread communication in programs.

9.2 MULTITHREADING IN JAVA

Multithreading in java is a process of executing multiple threads simultaneously.

Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation etc.

9.2.0 Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can performmultiple operations at same time.
2) You **can perform many operations together so it saves time**.
3) Threads are **independent** so it doesn't affect other threads if exception occur in asingle thread.

9.3 MULTITASKING

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved by two ways:

- Process-based Multitasking(Multiprocessing)
- Thread-based Multitasking(Multithreading)

1) Process-based Multitasking (Multiprocessing)

- Each process have its own address in memory i.e. each process allocatesseparate memory area.
- Process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another require some time for saving and loadingregisters, memory maps, updating lists etc.

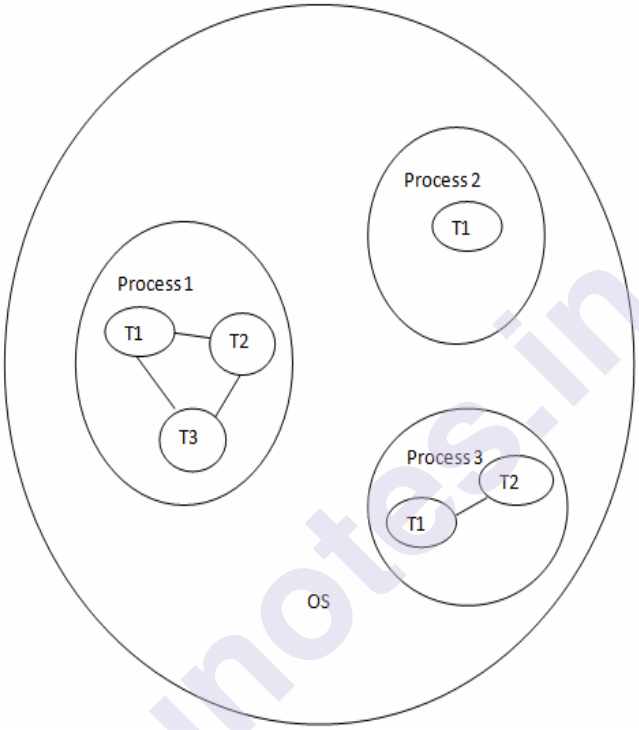
2) Thread-based Multitasking (Multithreading)

- Threads share the same address space.
- Thread is lightweight.
- Cost of communication between the thread is low.

9.4 WHAT IS THREAD IN JAVA

A thread is a lightweight sub process, a smallest unit of processing. It is a separatepath of execution.

Threads are independent, if there occurs exception in one thread, it doesn't affect otherthreads. It shares a common memory area.



9.5 JAVA THREAD CLASS

Thread class is the main class on which java's multithreading system is based. Thread class provide constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

Java Thread Methods

S.N.	Modifie r and Type	Method	Description
1	void	start()	It is used to start the execution of thethread.
2	void	run()	It is used to perform action for a thread.
3	static void	sleep()	It sleeps a thread for the specifiedamount of time.

4	static Thread	current Thread()	It returns a reference to the currentlyexecuting thread object.
5	void	join()	It waits for a thread to die.
6	int	get Priority()	It returns the priority of the thread.
7	void	set Priority()	It changes the priority of the thread.
8	String	get Name()	It returns the name of the thread.
9	void	set Name()	It changes the name of the thread.
10	long	getId()	It returns the id of the thread.
11	boolean	isAlive()	It tests if the thread is alive.
12	static void	yield()	It causes the currently executing thread object to temporarily pause and allow other threads to execute.
13	void	suspend()	It is used to suspend the thread.
14	void	resume()	It is used to resume the suspended thread.
15	void	stop()	It is used to stop the thread.
16	void	destroy()	It is used to destroy the thread group and all of its subgroups.
17	boolean	is Daemon()	It tests if the thread is a daemon thread.
18	void	set Daemon()	It marks the thread as daemon or user thread.
19	void	interrupt()	It interrupts the thread.
20	boolean	is interrupted()	It tests whether the thread has been interrupted.
21	static boolean	interrupted()	It tests whether the current thread has been interrupted.
22	static int	active Count()	It returns the number of active threads in the current thread's thread group.

23	void	check Access()	It determines if the currently running thread has permission to modify the thread.
24	static boolean	hold Lock()	It returns true if and only if the current thread holds the monitor lock on the specified object.
25	static void	dump Stack()	It is used to print a stack trace of the current thread to the standard error stream.
26	Stack Trace Element[]	get Stack Trace()	It returns an array of stack trace elements representing the stack dump of the thread.
27	static int	enumerate()	It is used to copy every active thread's thread group and its subgroup into the specified array.
28	Thread. State	get State()	It is used to return the state of the thread.
29	Thread Group	get Thread Group()	It is used to return the thread group to which this thread belongs
30	String	to String()	It is used to return a string representation of this thread, including the thread's name, priority, and thread group.
31	void	notify()	It is used to give the notification for only one thread which is waiting for a particular object.
32	void	notify All()	It is used to give the notification to all waiting threads of a particular object.

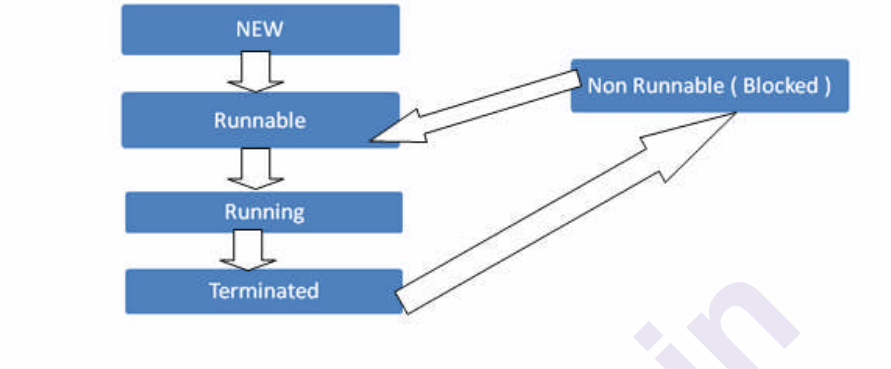
9.6 LIFE CYCLE OF A THREAD (THREAD STATES)

A thread can be in one of the five states. According to sun, there is only 4 states in **thread life cycle in java** new, runnable, non-runnable and terminated. There is no running state.

But for better understanding the threads, we are explaining it in the 5 states.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated



1) New

The thread is in new state if you create an instance of Thread class but before theinvocation of start() method.

2) Runnable

The thread is in runnable state after invocation of start() method, but the threadscheduler has not selected it to be the running thread.

3) Running

The thread is in running state if the thread scheduler has selected it.

4) Non-Runnable (Blocked)

This is the state when the thread is still alive, but is currently not eligible to run.

5) Terminated

A thread is in terminated or dead state when its run() method exits.

How to create thread

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

9.7 THREAD CLASS

Thread class provide constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

Commonly used methods of Thread class:

1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
3. **public void sleep (long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long miliseconds):** waits for a thread to die for the specified miliseconds.
6. **public int getPriority():** returns the priority of the thread.
7. **public int setPriority(int priority):** changes the priority of the thread.
8. **public String get Name():** returns the name of the thread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread current Thread():** returns the reference of currently executing thread.
11. **public int getId():** returns the id of the thread.
12. **public Thread.State getState():** returns the state of the thread.
13. **public boolean isAlive():** tests if the thread is alive.
14. **public void yield():** causes the currently executing thread object totemporarily pause and allow other threads to execute.
15. **public void suspend():** is used to suspend the thread (depricated).
16. **public void resume():** is used to resume the suspended thread(depricated).
17. **public void stop():** is used to stop the thread (depricated).
18. **public boolean is Daemon():** tests if the thread is a daemon thread.
19. **public void set Daemon (boolean b):** marks the thread as daemon or userthread.
20. **public void interrupt():** interrupts the thread.
21. **public boolean is Interrupted():** tests if the thread has been interrupted.

22. **public static boolean interrupted():** tests if the current thread has beeninterrupted.

1) Java Thread Example by extending Thread class

```
class Multi extends Thread
{
public void run()
{
System.out.println("thread is running...");
}
public static void main(String args[])
{
Multi t1=new Multi();
t1.start();
}
}
```

Output: thread is running...

9.8 SYNCHRONIZATION IN JAVA

Synchronization in java is the capability *to control the access of multiple threads to anyshared resource*.

Java Synchronization is better option where we want to allow only one thread to accessthe shared resource.

Why use Synchronization ?
The synchronization is mainly used to
1. To prevent thread interference.
2. To prevent consistency problem.

Types of Synchronization
There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Here, we will discuss only thread synchronization.

Thread Synchronization
There are two types of thread synchronization mutual exclusive and inter-threadcommunication.

- Mutual Exclusive
- Synchronized method.
- Synchronized block.
- static synchronization.
- Cooperation (Inter-thread communication in java)

9.9 MUTUAL EXCLUSIVE

Mutual Exclusive helps keep threads from interfering with one another while sharingdata. This can be done by three ways in java:

- by synchronized method
- by synchronized block
- by static synchronization

```
class Table{
void printTable(int n){//method not synchronized
for(int i=1;i<=5;i++){
System.out.println(n*i);
try{
Thread.sleep(400);
}
catch(Exception e)
{
System.out.println(e);
}}}}
class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}
class TestSynchronization1{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
```

```
t1.start();
t2.start();
}}
```

Output:

```
5
100
10
200
15
300
20
400
25
500
```

9.10 SAMPLE QUESTIONS

Q1. How does multithreading take place on a computer with a single CPU?

Q2. State the advantages of multithreading.

Q3. Explain how a thread is created by extending the Thread class.

Q4. Explain Lifecycle of Thread.

Q5. Explain the need of synchronized method

Q6. List And Explain Various Methods of Thread

10

EXCEPTIONS

Unit Structure :

10.0 INTRODUCTION

During programming in languages like c, c++ you might have observed that even after successful compilation some errors are detected at runtime. For handling these kinds of errors there is no support from programming languages like c, c++. Some error handling mechanisms like returning special values and setting flags are used to determine that there is some problem at runtime. In C++ programming language there is a very basic provision for exception handling. Basically exception handlings provide a safe escape route from problem or clean-up of error handling code. In Java exception handling is the only semantic way to report error .In Java exception is an object, which describes error condition, occurs in a section of code. In this unit we will discuss how exceptions are handled in Java, you will also learn to create your own exception classes in this unit.

10.1 OBJECTIVES

After going through this unit you will be able to:

- describe exception;
- explain causes of exceptions;
- writing programs with exceptions handling;

- use built–in exceptions;
- create your own exception classes.

10.2 EXCEPTION HANDLING IN JAVA

The **exception handling in java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

In this page, we will learn about java exception, its type and the difference betweenchecked and unchecked exceptions.

What is exception
Dictionary Meaning: Exception is an abnormal condition.

In java, exception is an event that disrupts the normal flow of the program. It is anobject which is thrown at runtime.

What is exception handling
Exception Handling is a mechanism to handle runtime errors such as Class Not Found,IO, SQL, Remote etc.

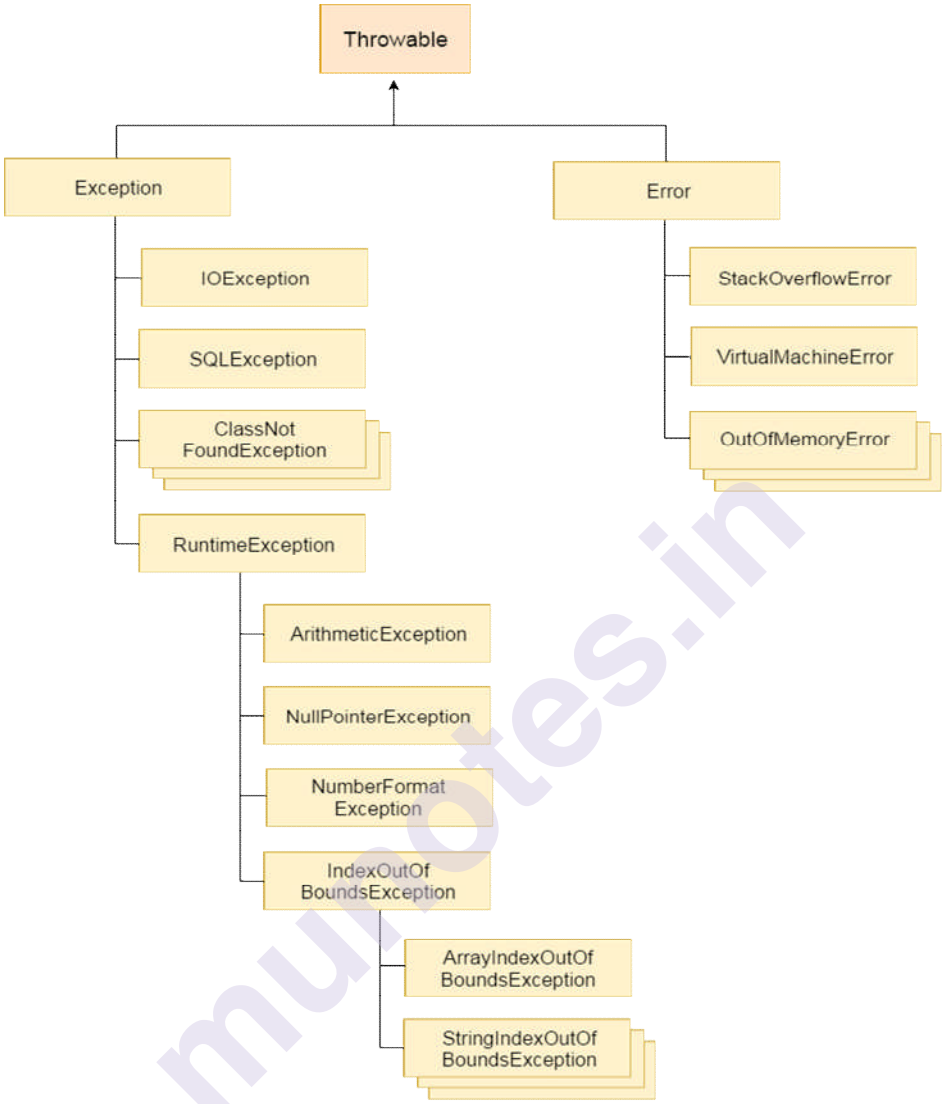
Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. Exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5;//exception occurs
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;
10. statement 10;

Suppose there is 10 statements in your program and there occurs an exception at statement 5, rest of the code will not be executed i.e. statement 6 to 10 will not run. If we perform exception handling, rest of the statement will be executed. That is why we use exception handling in java.

10.3 HIERARCHY OF JAVA EXCEPTION CLASSES



10.4 TYPE OF EXCEPTION

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun micro system says there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

Difference between checked and unchecked exceptions

1) Checked Exception

The classes that extend Throwable class except Runtime Exception and Error are known as checked exceptions e.g. IO

Exception, SQL Exception etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that extend Runtime Exception are known as unchecked exceptions e.g. Arithmetic Exception, Null Pointer Exception, Array Index Out Of Bounds Exception etc.Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

3) Error

Error is irrecoverable e.g. out of Memory Error, Virtua Machine Error, Assertion Error etc.

10.5 COMMOM SCENARIOS WHERE EXCEPTION MAY OCCUR

There are given some scenarios where unchecked exceptions can occur. They are asfollows:

1) Scenario where Arithmetic Exception occurs

If we divide any number by zero, there occurs an Arithmetic Exception.

```
1. int a=10/0;//ArithmeticException
```

2) Scenario where Null Pointer Exception occurs

If we have null value in any variable, performing any operation by the variable occursan Null Pointer Exception.

```
1.      String s=null;
2.      System.out.println (s.length());//Null Pointer Exception
```

3) Scenario where Number Format Exception occurs

The wrong formatting of any value, may occur Number Format Exception. Suppose Ihave a string variable that have characters, converting this variable into digit will occur Number Format Exception.

```
1. String s="abc";
2. int i=Integer.parseInt(s);//Number Format Exception
```

4) Scenario where Array Index Out Of Bounds Exception occurs

If you are inserting any value in the wrong index, it would result

Array Index Out Of Bounds Exception as shown below:

```
1.  int a[]=new int[5];
2.  a[10]=50; //Array Index Out Of Bounds Exception
```

10.6 JAVA EXCEPTION HANDLING KEYWORDS

There are 5 keywords used in java exception handling.

1. try
2. catch
3. finally
4. throw
5. throws

10.7 JAVA TRY BLOCK

Java try block is used to enclose the code that might throw an exception. It must beused within the method.

Java try block must be followed by either catch or finally block.

Syntax of java try-catch

```
try{
//code that may throw exception
}catch(Exception_class_Name ref){}
```

Syntax of try-finally block

```
try{
//code that may throw exception
}finally{}
```

10.8 JAVA CATCH BLOCK

Java catch block is used to handle the Exception. It must be used after the try blockonly.

You can use multiple catch block with a single try.

Problem without exception handling
Let's try to understand the problem if we don't use try-catch block.

```
public class Testtrycatch1{
public static void main(String args[]){
int data=10/0;//may throw exception
System.out.println("rest of the code...");
}
}
```

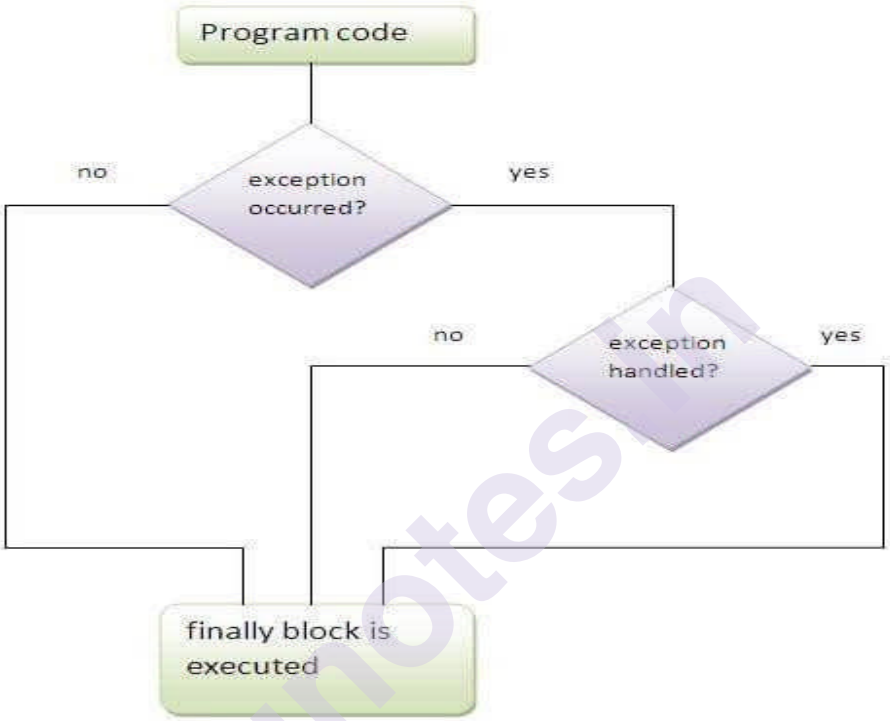
Output:

```
Exception in thread main java.lang. Arithmetic Exception:/ by zero
```

10.9 JAVA FINALLY BLOCK

Java finally block is a block that is used *to execute important code* such as closingconnection, stream etc.

Java finally block is always executed whether exception is handled or not.Java finally block follows try



Usage of Java finally
Let's see the different cases where java finally block can be used.

Case 1
Let's see the java finally example where **exception doesn't occur**.

```
class Test Finally Block{
public static void main(String args[]){
try{
int data=25/5;
System.out.println(data);
}
catch(Nul Pointer Exception e){System.out.println(e);}
finally{System.out.println("finally block is always executed");}
System.out.println("rest of the code...");
}
}
```

Output:5

finally block is always executedrest of the code...

10.10 JAVA THROW EXCEPTION

Java throw keyword
The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

The syntax of java throw keyword is given below.

throw exception;

Let's see the example of throw IO Exception.

throw new IO Exception("sorry device error);

java throw keyword example

In this example, we have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the Arithmetic Exception otherwise print a message welcome to vote.

```
public class TestThrow1{
static void validate(int age){
if(age<18)
throw new Arithmetic Exception("Age not valid");
else
System.out.println("welcome You can vote");
}
public static void main(String args[]){
validate(12);
System.out.println("rest of the code...");
}
}
```

Output:

Exception in thread main java.lang. Arithmetic Exception:not valid

10.11 JAVA THROWS KEYWORD

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an

exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occursany unchecked exception such as Null Pointer Exception, it is programmers fault that he is not performing check up before the code being used.

Syntax of java throws

return_type method_name() **throws** exception_class_name{
//method code }

Which exception should be declared ?
Ans : checked exception only, because:

- **unchecked Exception:** under your control so correct your code.
- **error:** beyond your control e.g. you are unable to do anything if there occursVirtual Machine Error or Stack Over flow Error.

Advantage of Java throws keyword

Now Checked Exception can be propagated (forwarded in call stack). It provides information to the caller of the method about the exception

Java throws example

Let's see the example of java throws clause which describes that checked exceptionscan be propagated by throws keyword.

```
import java.io.IOException;
class Testthrows1{
void m()throws IOException{
throw new IOException("device error");//checked exception

}
void n()throws IOException{
m();
}
void p(){
try{
n();
}catch(Exception e){System.out.println("exception handled");}
}
public static void main(String args[]){
Testthrows1 obj=new Testthrows1();
```

```
obj.p();
System.out.println("normal flow...");
}
}
```

Output:

exception handlednormal flow...

10.12 DIFFRENCE BETWEEN THROW AND THROWS IN JAVA

There are many differences between throw and throws keywords. A list of differencesbetween throw and throws are given below:

Sr.No	throw	throws
1	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare exception.
2	Checked exception cannot propagated using throw only.	Checked exception can be propagated with throws.
3	Throw is followed by an instance.	Throws is followed by class.
4	Throw is used within the method.	Throws is used with the method signature.
5	You cannot throw multiple exceptions.	You can declare multiple exceptions eg. public void method()throws IO Exception, SQL Exception.

10.13 SAMPLE QUESTIONS

Q1. What is an exception? Write any three actions that can be taken after an exception occurs in a program.

Q2. Write a program to catch more than two exceptions.

Q3. Write a partial program to show the use of finally clause.

Q4. Differentiate between checked and unchecked exceptions.

Q5. Explain how you can throw an exception from a method in Java.

Q6. Write a program to create your own exception subclass that throws exception if the sum of two integers is greater that 99.

Q7. Draw and Explain Exception class Hirerchy in detail.

Q8. Differentiate between throw and throws exceptions / Keywords.

11

BYTE STREAMS

Unit Structure :

11.0 INTRODUCTION

Input is any information that is needed by a program to complete its execution. Output is any information that the program must convey to the user. Input and Output are essential for applications development. To accept input a Java program opens a stream to a data source, such as a file or remote socket, and reads the information serially. Whether reading data from a file or from a socket, the concept of serially reading from, and writing to, different data sources is the same. For that very reason, it is essential to understand the features of top-level classes (Java.io.Reader, Java.io.Writer). In this unit you will be working on some basics of I/O (Input–Output) in Java such as Files creation through streams in Java code. A stream is a linear, sequential flow of bytes of input or output data. Streams are written to the file system to create files. Streams can also be transferred over the Internet. In this unit you will learn the basics of Java streams by reviewing the differences between byte and character streams, and the various stream classes available in the Java.io package. We will cover the standard process for standard Input (Reading from console) and standard output (writing to console).

11.1 OBJECTIVES

After going through this unit you will be able to:

- explain basics of I/O operations in Java;
- use stream classes in programming;
- take inputs from console;
- write output on console;
- read from files, and
- write to files.

11.2 JAVA I/O TUTORIAL

Java I/O (Input and Output) is used *to process the input* and *produce the output*.

Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

We can perform **file handling in Java** by Java I/O API.

11.3 STREAM

A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a streambecause it is like a stream of water that continues to flow.

In Java, 3 streams are created for us automatically. All these streams are attached with theconsole.

1) **System.out:** standard output stream

2) **System.in:** standard input stream

3) **System.err:** standard error stream

Let's see the code to print **output and an error** message to the console.

```
System.out.println("simple message");
System.err.println("error message");
```

Let's see the code to get **input** from console.

```
int i=System.in.read();//returns ASCII code of 1st character
System.out.println((char)i);//will print the character
```

11.4 OUTPUT STREAM VS INPUT STREAM

The explanation of Output Stream and Input Stream classes are given below:

Output Stream

Java application uses an output stream to write data to a destination; it may be a file, an array,peripheral device or socket.

Input Stream

Java application uses an input stream to read data from a source; it may be a file, an array,peripheral device or socket.

Let's understand the working of Java Output Stream and Input Stream by the figure given below.

11.5 OUTPUT STREAM

Output Stream class is an abstract class. It is the super class of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Useful methods of Output Stream

Method	Description
public void write(int)throws IOException	is used to write a byte to the current output stream.
public void write(byte[])throws IOException	is used to write an array of byte to thecurrent output stream.
public void flush()throws IOException	flushes the current output stream.
public void close()throws IOException	is used to close the current output stream.

11.6 INPUT STREAM CLASS

Input Stream class is an abstract class. It is the superclass of all classes representing an inputstream of bytes.

Useful methods of Input Stream

Method	Description
public abstract int read()throws IO Exception	reads the next byte of data from the input stream.It returns -1 at the end of the file.
public int available()throws IO Exception	returns an estimate of the number of bytes that can be read from the current input stream.
public void close()throws IO Exception	is used to close the current input stream.

11.7 JAVA FILE OUTPUT STREAM CLASS

Java File Output Stream is an output stream used for writing data to a file.

If you have to write primitive values into a file, use File Output Stream class. You can write byte-oriented as well as character-oriented data through File Output Stream class. But, for character- oriented data, it is preferred to use File Writer than File Output Stream.

File Output Stream class declaration
Let's see the declaration for Java.io.File Output Stream class:

1. **public class** File Output Stream **extends** Output Stream

File Output Stream class methods

Method	Description
protected void finalize()	It is used to clean up the connection with the file outputstream.
void write(byte[] ary)	It is used to write **ary.length** bytes from the byte array to the file output stream.
void write(byte[] ary, int off, int len)	It is used to write **len** bytes from the byte array starting at offset **off** to the file output stream.
void write(int b)	It is used to write the specified byte to the file outputstream.
File Channel get Channel()	It is used to return the file channel object associated with the file output stream.
File Descriptor get FD()	It is used to return the file descriptor associated withthe stream.
void close()	It is used to closes the file output stream.

Java FileOutputStream Example 1: write byte

```
import java.io.FileOutputStream;
public class FileOutputStreamExample {
public static void main(String args[]){
try{
FileOutputStream fos=new FileOutputStream("D:\\test.txt");
fos.write(65);
fos.close();
System.out.println("success...");
}
catch(Exception e)
{System.out.println(e);}
}
 }
```

Output:

```
Success...
```

The content of a text file **test.txt** is set with the data **A**.test.txt

```
A
```

Java FileOutputStream example 2: write string

```
import java.io.FileOutputStream;
public class FileOutputStreamExample {
public static void main(String args[]){
try{
FileOutputStream fos=new FileOutputStream("D:\\test.txt");
String s="Welcome to java.";
byte b[]=s.getBytes();//converting string into byte array
fos.write(b);
fos.close();
System.out.println("success ! check text file...");
}catch(Exception e){System.out.println(e);}
}
}
```

Output:

```
Success...
```

The content of a text file **test.txt** is set with the data **Welcome to java.**

test.txt

```
Welcome to java.
```

11.8 JAVA FILEINPUTSTREAM CLASS

Java File Input Stream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character- stream data. But, for reading streams of characters, it is recommended to use File Reader class.

Java File Input Stream class declaration
Let's see the declaration for java.io.File Input Stream class:

1. **public class** File Input Stream **extends** Input Stream

Java FileInput Stream class methods

Method	Description
int available()	It is used to return the estimated number of bytes that canbe read from the input stream.
int read()	It is used to read the byte of data from the input stream.
int read(byte[] b)	It is used to read up to **b.length** bytes of data from theinput stream.
int read(byte[] b, intoff, int len)	It is used to read up to **len** bytes of data from the inputstream.
long skip(long x)	It is used to skip over and discards x bytes of data from theinput stream.
File Channel get Channel()	It is used to return the unique File Channel object associated with the file input stream.
File Descriptorget FD()	It is used to return the File Descriptor object.
protected voidfinalize()	It is used to ensure that the close method is call when thereis no more reference to the file input stream.
void close()	It is used to closes the stream.

11.9 JAVA BYTE ARRAY OUTPUT STREAM CLASS

Java Byte Array Output Stream class is used to **write common data** into multiple files. In thisstream, the data is written into a byte array which can be written to multiple streams later.

The Byte Array Output Stream holds a copy of data and forwards it to multiple streams. The buffer of Byte Array Output Stream automatically grows according to data.

Java Byte Array Output Stream class declaration
Let's see the declaration for Java.io.Byte Array Output Stream class:

1. **public class** Byte Array Output Stream **extends** Output Stream

Java Byte Array Output Stream class constructors

Constructor	Description
Byte Array Output Stream()	Creates a new byte array output stream with the initial capacity of 32 bytes, though its size increases if necessary.
Byte Array Output Stream (int size)	Creates a new byte array output stream, with a buffer capacity of the specified size, in bytes.

Java Byte Array Output Stream class methods

Method	Description
int size()	It is used to returns the current size of a buffer.
byte[] to Byte Array()	It is used to create a newly allocated byte array.
String to String()	It is used for converting the content into a string decoding bytes using a platform defaultcharacter set.
String to String (String charset Name)	It is used for converting the content into a string decoding bytes using a specified charset Name.
void write(int b)	It is used for writing the byte specified to the bytearray output stream.
void write(byte[] b, int off,int len	It is used for writing **len** bytes from specified byte array starting from the offset **off** to the byte array output stream.
void write To (Output Streamout)	It is used for writing the complete content of a bytearray output stream to the specified output stream.
void reset()	It is used to reset the count field of a byte arrayoutput stream to zero value.
void close()	It is used to close the Byte Array Output Stream.

Example of Java Byte Array Output Stream
Let's see a simple example of java Byte Array Output Stream class to write common data into 2files: f1.txt and f2.txt.

```
import java.io.*;
public class Data Stream Example {
public static void main(String args[])throws Exception{
File Output Stream fos1=new File Output Stream("D:\\test1.txt");
File Output Stream fos2=new File Output Stream("D:\\test2.txt");7.
Byte Array Output Stream bos=new Byte Array Output Stream();
bos.write(65);
bos.writeTo(fout1);
bos.writeTo(fout2);
bos.flush();
bos.close();//has no effect
System.out.println("Success...");
}
}
```

Output:

```
Success...
```

test1.txt:

```
A
```

test2.txt:

```
A
```

11.10 JAVA BYTE ARRAY INPUT STREAM CLASS

The Byte Array Input Stream is composed of two words: Byte Array and Input Stream. As the namesuggests, it can be used to read byte array as input stream.

Java Byte Array Input Stream class contains an internal buffer which is used to **read byte array** asstream. In this stream, the data is read from a byte array.

The buffer of Byte Array Input Stream automatically grows according to data.

Java Byte Array Input Stream class declaration
Let's see the declaration for Java.io.Byte Array Input Stream class:

1. **public class** Byte Array Input Stream **extends** Input Stream

Java Byte Array Input Stream class constructors

Constructor	Description
Byte Array Input Stream(byte[] ary)	Creates a new byte array input stream which uses **ary** as its buffer array.
Byte Array Input Stream(byte[] ary, int offset, int len)	Creates a new byte array input stream which uses **ary** as its buffer array that can read up to specified **len** bytes of data from an array.

Java Byte Array Input Stream class methods

Methods	Description
int available()	It is used to return the number of remaining bytes that can be read from the input stream.
int read()	It is used to read the next byte of data from the inputstream.
int read(byte[] ary, int off, int len)	It is used to read up to len bytes of data from an array of bytes in the input stream.
boolean mark Supported()	It is used to test the input stream for mark and reset method.
long skip(long x)	It is used to skip the x bytes of input from the inputstream.
void mark (int read Ahead Limit)	It is used to set the current marked position in the stream.
void reset()	It is used to reset the buffer of a byte array.
void close()	It is used for closing a Byte Array Input Stream.

Example of Java Byte Array Input Stream
Let's see a simple example of java Byte Array Input Stream class to read byte array as inputstream.

```
import java.io.*;
public class Read Example {
public static void main(String[] args) throws IO Exception {

byte[] buf = { 35, 36, 37, 38 };
// Create the new byte array input stream
Byte Array Input Stream byt = new Byte Array Input Stream(buf);
```

```
int k = 0;
while ((k = byt.read()) != -1) {
//Conversion of a byte into character
char ch = (char) k;
System.out.println("ASCII value is:" + k + "; Special character is: " + ch);
}
}
}
```

Output:

```
ASCII value is:35; Special character is: #
ASCII value is:36; Special character is: $
ASCII value is:37; Special character is: %
ASCII value is:38; Special character is: &
```

Java Char Array Reader Class

The Char Array Reader is composed of two words: Char Array and Reader. The Char Array Reader class is used to read character array as a reader (stream). It inherits Reader class.

Java Char Array Reader class declaration

Let's see the declaration for Java.io.Char Array Reader class:

1. **public class** Char Array Reader **extends** Reader

Java Char Array Reader class methods

Method	Description
int read()	It is used to read a single character
int read(char[] b, int off, intlen)	It is used to read characters into the portion of anarray.
boolean ready()	It is used to tell whether the stream is ready to read.
boolean mark Supported()	It is used to tell whether the stream supports mark()operation.
long skip(long n)	It is used to skip the character in the input stream.
void mark(int read Ahead Limit)	It is used to mark the present position in the stream.
void reset()	It is used to reset the stream to a most recent mark.

Example of Char Array Reader Class:

Let's see the simple example to read a character using Java Char Array Reader class.

```
import java.io.CharArrayReader;
public class CharArrayExample{
public static void main(String[] ag) throws Exception
{
char[] ary = { 'j', 'a', 'v', 'a'};
CharArrayReader reader = new CharArrayReader(ary);
int k = 0;
// Read until the end of a file
while ((k = reader.read()) != -1) {
char ch = (char) k;
System.out.print(ch + " : ");
System.out.println(k);
}
}
}
```

Output :

```
j : 106
a : 97
v : 118
a : 97
```

11.11 JAVA CHAR ARRAY WRITER CLASS

The Char Array Writer class can be used to write common data to multiple files. This class inherits Writer class. Its buffer automatically grows when data is written in this stream. Calling the close() method on this object has no effect.

Java Char Array Writer class declaration
Let's see the declaration for Java.io.Char Array Writer class:

public class Char Array Writer **extends** Writer

Java Char Array Writer class Methods

Method	Description
int size()	It is used to return the current size of thebuffer.
char[] to Char Array()	It is used to return the copy of an input data.
String to String()	It is used for converting an input data toa string.

Char Array Writer append (char c)	It is used to append the specified character to the writer.
Char Array Writer append (Char Sequence csq)	It is used to append the specified character sequence to the writer.
Char Array Writer append (Char Sequence csq, int start, intend)	It is used to append the subsequence of aspecified character to the writer.
void write(int c)	It is used to write a character to the buffer.
void write(char[] c, int off, int len)	It is used to write a character to the buffer.
void write(String str, int off, int len)	It is used to write a portion of string to the buffer.
void write To(Writer out)	It is used to write the content of buffer to different character stream.
void flush()	It is used to flush the stream.
void reset()	It is used to reset the buffer.
void close()	It is used to close the stream.

Example of Char Array Writer Class:
In this example, we are writing a common data to 4 files a.txt, b.txt, c.txt and d.txt.

```
import java.io.CharArrayWriter;
import java.io.FileWriter;
public class CharArrayWriterExample {
public static void main(String args[])throws Exception{
CharArrayWriter out=new CharArrayWriter();
out.write("Welcome to java");
FileWriter f1=new FileWriter("D:\\testa.txt");
FileWriter f2=new FileWriter("D:\\ testb.txt");
FileWriter f3=new FileWriter("D:\\ testc.txt");
FileWriter f4=new FileWriter("D:\\ testd.txt");
out.writeTo(f1);
out.writeTo(f2);
out.writeTo(f3);
out.writeTo(f4);
f1.close();
f2.close();
f3.close();
```

```
f4.close();
System.out.println("Success...");
}
}
```

Output

```
Success...
```

After executing the program, you can see that all files have common data: Welcome tojava.

testa.txt:

```
Welcome to java
```

testb.txt:
c.txt:

```
Welcome to java
```

testd.txt:

```
Welcome to java
```

11.12 JAVA FILE CLASS

The File class is an abstract representation of file and directory pathname. A pathname can beeither absolute or relative.

The File class have several methods for working with directories and files such as creating newdirectories or files, deleting and renaming directories or files, listing the contents of a directoryetc.

Fields

Modifier	Type	Field	Description
static	String	path Separator	It is system-dependent path-separator character, represented as a string for convenience.
static	char	path Separator Char	It is system-dependent path-separator character.
static	String	separator	It is system-dependent default name-separator character, represented as a string for convenience.
static	char	separator Char	It is system-dependent default name-separator character.

Constructors

Constructor	Description
File(File parent, Stringchild)	It creates a new File instance from a parent abstract pathname and a child pathname string.
File(String pathname)	It creates a new File instance by converting the given pathname string into an abstract pathname.
File(String parent,String child)	It creates a new File instance from a parent pathname string and a child pathname string.
File(URI uri)	It creates a new File instance by converting the given file: URI into an abstract pathname.

Useful Methods

Modifier and Type	Method	Description
static File	create Temp File(String prefix, String suffix)	It creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name.
boolean	create New File()	It atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.
boolean	canWrite()	It tests whether the application can modify the file denoted by this abstract pathname.String[]
boolean	canExecute()	It tests whether the application can execute the file denoted by this abstract pathname.
boolean	canRead()	It tests whether the application can read the file denoted by this abstract pathname.
boolean	isAbsolute()	It tests whether this abstract pathname is absolute.

boolean	isDirectory()	It tests whether the file denoted by this abstract pathname is a directory.
boolean	isFile()	It tests whether the file denoted by this abstract pathname is a normal file.
String	getName()	It returns the name of the file or directory denoted by this abstract pathname.
String	getParent()	It returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory.
Path	toPath()	It returns a java.nio.file.Path object constructed from the this abstract path.
URI	toURI()	It constructs a file: URI that represents this abstract pathname.
File[]	listFiles()	It returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname
long	get Free Space()	It returns the number of unallocated bytes in the partition named by this abstract path name.
String[]	list (Filename Filter filter)	It returns an array of strings naming the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
boolean	mkdir()	It creates the directory named by this abstract pathname.

Java File Example 1

```
import java.io.*;
public class FileDemo {
public static void main(String[] args)
{
try {
File file = new File("javaFile123.txt");
if (file.createNewFile()) {
System.out.println("New File is created!");
```

```
} else {
System.out.println("File already exists.");
}
} catch (IOException e) {
e.printStackTrace();
}
}
}
```

Output:

```
New File is created!
```

Java File Example 2

```
import java.io.*;
public class FileDemo2 {
public static void main(String[] args) {
String path = "";
boolean bool = false;
try {
// createing  new files
File file  = new File("testFile1.txt");
file.createNewFile();
System.out.println(file);
// createing new canonical from file object
File file2 = file.getCanonicalFile();
// returns true if the file exists
System.out.println(file2);
bool = file2.exists();
// returns absolute pathname
path = file2.getAbsolutePath();
System.out.println(bool);
// if file exists
if (bool) {
// prints
System.out.print(path + " Exists? " + bool);
}
} catch (Exception e) {
// if any error occurs
e.printStackTrace();
}
}
}
```

Output:

```
testFile1.txt
/home/Work/Project/File/t
estFile1.txttrue
/home/Work/Project/File/testFile1.txt Exists? true
```

Java File Example 3

```
import java.io.*;
public class File Example {
public static void main(String[] args) {
File f=new File("/Users/test/Documents");
String filenames[]=f.list();
for(String filename:filenames){
System.out.println(filename);
}
}
}
```

Output:

bestreturn_orgtest.rtf BIODATA10.pagesBIODATA1.pdf BIODATA9.png struts2jars_test.zip workspace_test

Java File Example 4

```
import java.io.*;
public class FileExample {
public static void main(String[] args) {
File dir=new File("/Users/Test/Documents");
File files[]=dir.listFiles();
for(File file:files){
System.out.println(file.getName()+" Can Write: "+file.canWrite()+"
Is Hidden: "+file.isHidden()+" Length: "+file.length()+" bytes");
}
}
}
```

Output:

apache-tomcat-9.0.0.M19 Can Write: true Is Hidden: false Length: 476 bytes

apache-tomcat-9.0.0.M19.tar Can Write: true Is Hidden: false Length: 13711360 bytes

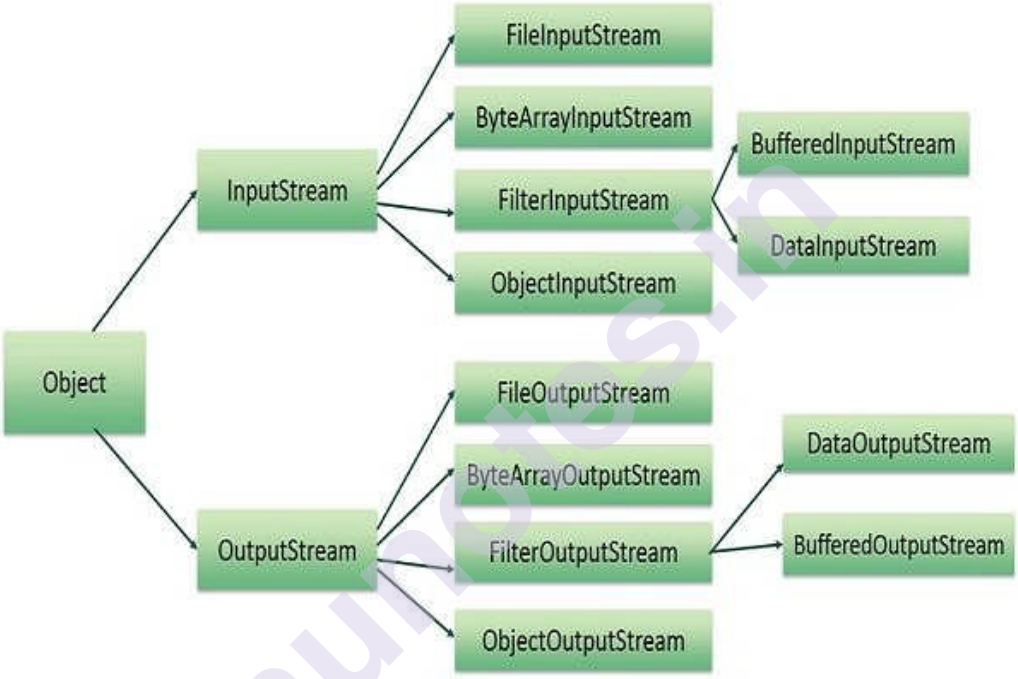
bestreturn_org.rtf Can Write: true Is Hidden: false Length: 389 bytes BIODATA10.pages Can Write: true Is Hidden: false Length: 707985

bytes BIODATA1.pdf Can Write: true Is Hidden: false Length: 69681 bytes BIODATA5.png Can Write: true Is Hidden: false Length: 282125 bytes workspace Can Write: true Is Hidden: false Length: 1972 bytes

11.13 READING AND WRITING FILES

As described earlier, a stream can be defined as a sequence of data. The **Input Stream** is used to read data from a source and the **Output Stream** is used for writing data to a destination.

Here is a hierarchy of classes to deal with Input and Output streams.



The two important streams are **File Input Stream** and **File Output Stream**, which would be discussed in this tutorial.

File Input Stream

This stream is used for reading data from the files. Objects can be created using the keyword **new** and there are several types of constructors available.

Following constructor takes a file name as a string to create an input streamobject to read the file −

Following constructor takes a file object to create an input stream object

```
InputStream f = new FileInputStream("C:/java/hello");
```

toread the file. First we create a file object using File() method as follows −

```
File f = new File("C:/java/hello");
```

```
InputStream f = new FileInputStream(f);
```

Once you have *Input Stream* object in hand, then there is a list of helper methods which can be used to read to stream or to do other operations on the stream.

Sr.No.	Method & Description
1	**public void close() throws IO Exception{}** This method closes the file output stream. Releases any system resources associated with the file. Throws an IO Exception.
2	**protected void finalize()throws IO Exception {}** This method cleans up the connection to the file. Ensures that the close method of this file output stream is called when there are no more references to this stream. Throws an IO Exception.
3	**public int read(int r)throws IO Exception{}** This method reads the specified byte of data from the InputStream. Returns an int. Returns the next byte of data and -1 will be returned if it's the end of the file.
4	**public int read(byte[] r) throws IO Exception{}** This method reads r. length bytes from the input stream into an array. Returns the total number of bytes read. If it is the end of the file, -1 will be returned.
5	**public int available() throws IO Exception{}** Gives the number of bytes that can be read from this file input stream.Returns an int.

11.14 SAMPLE QUESTIONS

Q1. What is stream? Differentiate between stream source and stream destination.

Q2. Write a program for I/O operation using Buffered Input Stream and Buffered Output Stream

Q3. Write a program using File Reader and Print Writer classes for file handling.

Q4. Which class may be used for reading from console?

Q5. Write a program to read the output of a file and display it on console.

Q7.What is Java Stream Explain in with example.

Q8. What is Serialization?

Q9. Explain File Reader class With Example.

Q10. Explain File Writer class With Example.

Q11. Explain File class With Example.

Q12. Explain Char Array Reader class With Example.

Q13. Explain Char Array Writer class With Example.

Q12. Explain Byte Array Input Stream class With Example.

Q13. Explain Byte Array Output Stream class With Example.

UNIT SUMMARY :-

- In this unit we first discussed about an array, which is a fixed-length data structure that can contain multiple objects of the same type. An element within an array can be accessed by its index. Indices begin at 0 and end at the length of the array.

- Also we discuss about how Java goes to great lengths to help you deal with error conditions. In this unit we have discussed how Java's exception mechanisms give a structured way to perform a go-to from the place where an error occurs to the code that knows how to handle the error. In this unit we have discussed different causes of exception, using try, catch, finally, throw and throws clauses in exception handling. This unit deals with ways to handle error conditions in a structured, methodical way. This unit discusses types of exceptions, Throwable class hierarchy, and explains how to write own exception subclasses.

- Also we discuss about working of multithreading in Java. Also you have learned what is the main thread and when it is created in a Java program. Different states of threads are described in this unit. This unit explained how threads are created using Thread class and Runnable interface. It explained how thread priority is used to determine which thread is to execute next. This unit explains concept of synchronization, creating synchronous methods and inter thread communication. It is also explained how object locks are used to control access to shared resources.

- Also we discuss about various methods of I/O streams-binary, character and object in Java. This unit briefs that input and output in the Java language is organised around the concept of streams. All input is done through subclasses of Input Stream and all output is done through subclasses of Output Stream. (Except for Random Access File). We have covered how various streams can be combined together to get the added functionality of standard input and stream input. In

this unit you have also learned the operations of reading from a file and writing to a file. For this purpose objects of File Reader and File Writer classes are used.

UNIT FURTHER READINGS :-

- Core Java for Beginners by Sharanam Shah
- Java: The Complete Reference , Oracle , Author: Herbert Schildt
- E.Balaguruswamy, Programming with Java, second edition, Tata McGraw Hill publications,2000
- Programming with java , Tata McGraw-Hill Education by E.Balgurusamy

Web Refrences :-

www.w3school.com
www.tutorialspoint.com
www.javatpoint.com
www.programiz.com

Unit V

12

EVENT HANDLING

Unit Structure

12.0 OBJECTIVES

This chapter would make you understand the following concepts:

- Define Event Handling in Java
- Describe Event Handling in Java
- Explain Two Event Handling Mechanisms.
- Illustrate the Delegation Event Model in Java
- Describe Components of Event Handling
- Explain steps to handle an event in java

12.1 INTRODUCTION

In this chapter we will learn Event handling which is fundamental to Java programming because it is integral to the creation of applets and other types of GUI-based programs. Change in the state of an object is known as event. Event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components. Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has the code which is known as event handler that is executed when an event occurs.

12.2 EVENT HANDLING

Event handling is prime to Java programming because it's integral to the creation of applets and other sorts of GUI-based programs. Events are supported by a variety of packages, including java.util, java.awt, and java.awt.event. The program response is generated when the user interacts with a GUI-based program.

Two event handling mechanisms in java

1. Original Version of Java (1.0)
2. Modern Versions of Java, beginning with Java version 1.1

The way during which events are handled changed significantly between the first version of Java (1.0) and every one subsequent version of Java, beginning with version 1.1. Although the 1.0 method of event handling remains supported, it's not recommended for brand spanking new programs.

Also, many of the methods that support the old 1.0 event model are deprecated. The modern approach is the way that events should be handled by all-new programs.

12.2.1 Delegation Event Model in Java

The Delegation Event model is defined to handle events in GUI programming languages. The GUI stands for Graphical User Interface, where a user graphically/visually interacts with the system.

The GUI programming is inherently event-driven; whenever a user initiates an activity such as a mouse activity, clicks, scrolling, etc., each is known as an event that is mapped to a code to respond to functionality to the user. This is known as event handling.

In this section, we will discuss event processing and how to implement the delegation event model in Java. We will also discuss the different components of an Event Model.

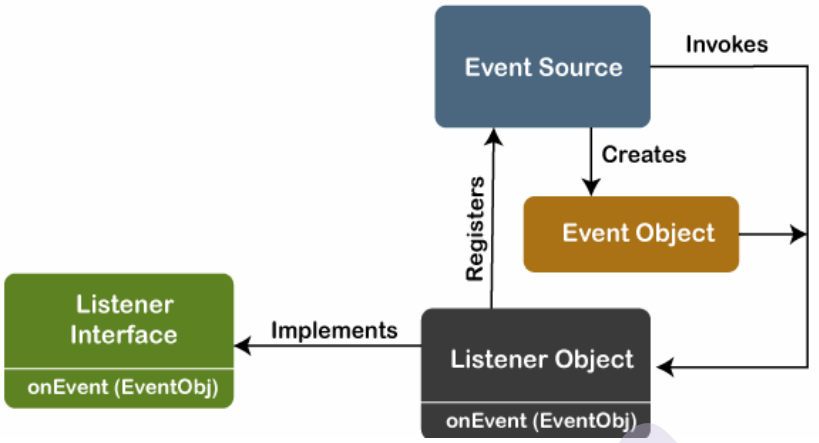
Advantage of using delegation event model

The advantage of this design is that the appliance logic that processes events is cleanly separated from the interface logic that generates those events. An interface element is in a position to "delegate" the processing of an occasion to a separate piece of code. In the delegation event model, listeners must register with a source so as to receive an occasional notification. This provides is a crucial benefit: notifications are sent only to listeners that want to receive them. This is a more efficient way to handle events.

Event Processing in Java

Java support event processing since Java 1.0. It provides support for AWT (Abstract Window Toolkit), which is an API used to develop

the Desktop application. In Java 1.0, the AWT was based on inheritance. To catch and process GUI events for a program, it should hold subclass GUI components and override action() or handleevent() methods. The below image demonstrates the event processing.



But, the modern approach for event processing is based on the Delegation Model. It defines a standard and compatible mechanism to generate and process events. In this model, a source generates an event and forwards it to one or more listeners. The listener waits until it receives an event. Once it receives the event, it is processed by the listener and returns it. The UI elements are able to delegate the processing of an event to a separate function.

The key advantage of the Delegation Event Model is that the application logic is completely separated from the interface logic.

In this model, the listener must be connected with a source to receive the event notifications. Thus, the events will only be received by the listeners who wish to receive them. So, this approach is more convenient than the inheritance-based event model (in Java 1.0).

In the older model, an event was propagated up the containment until a component was handled. This needed components to receive events that were not processed, and it took lots of time. The Delegation Event model overcame this issue.

Basically, an Event Model is based on the following three components:

- Events
- Events Sources
- Events Listeners

12.2.2 Event

An event is an object that describes a phase change during a source. It is often generated as a consequence of an individual interacting with the weather during a graphical interface. Some activities that cause events to be generated are pressing a button, entering a personality via the

keyboard, selecting an item during a list, and clicking the mouse. Events can also occur that aren't directly caused by interactions with an interface. For example, an occasion could also be generated when a timer expires, a counter exceeds a worth, a software or hardware failure occurs, or an operation is completed.

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

12.2.3 Java Event classes and Listener interfaces

Event Classes

The classes that represent events ae at the core of Java's event handling mechanism. At the root of the Java event class hierarchy is **event object**, which is in **java.util**. It is the super class for all events.

THE ACTION EVENT CLASS

An actioneventis generated when a button is pressed, a list item is double-clicked, or a menu item is selected. It defines four integer constants that can be used to identify any modifiers associated with an action event: **ALT_MASK**, **CTRL_MASK**, **META_MASK**, and **SHIFT _ MASK**. There is also an integer constant, **ACTION_ PERFORMED**, which can be used to identify action events.

Event listeners

A listener is an object that's notified when an occasion occurs. It has two major requirements. First, it registered with one or more sources to receive notifications about specific sorts of events. Second, it implements methods to receive and process these notifications. The methods that receive and process events are defined in interfaces found in **java.awt.event**.

Event Classes	Listener Interfaces
Actionevent	Actionlistener
Mouseevent	Mouselistener and mousemotionlistener
Mousewheelevent	Mousewheellistener
Keyevent	Keylistener
Itemevent	Itemlistener
Textevent	Textlistener
Adjustmentevent	Adjustmentlistener
Windowevent	Windowlistener
Componentevent	Componentlistener

Actionlistener

- The class which processes the action event should implement this interface. The object of that class must be registered with a component.

- The object can be registered using the add action listener() method.
- When the action event occurs, that object's action performed method is invoked.

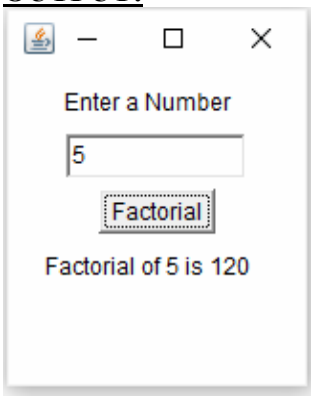
Interface methods

S.N.	Method & Description
1	**Void actionper formed (actionevente)** Invoked when an action occurs.

```
//program to calculate factorial of a number entered in the textfield
Import java.awt.*;
Import java.awt.event.*;

Public class action listenerexextends Frame implements action listener{
   Label l1,l2;
Textfieldt1;
   Button b1;

Actionlistenerex()
   {
Super("Action Listerner Example");
Setsize(200,200);
Setvisible(true);
Setlayout(new flowlayout());

L1=new Label("Enter a Number");
T1=new textfield(10);
B1=new Button("Factorial");
L2=new Label();

Add(l1);
Add(t1);
Add(b1);
Add(l2);

B1.addactionlistener(this);
   }
Public void actionperformed(actionevente)
   {
```

```
        String s=t1.gettext();
Int n=Integer.parseint(s);
Int f=1;
For(int i=1;i<=n;i++)
F=f*i;
L2.settext("Factorial of "+s+" is " +f);
   }
Public static void main(String[] args) {
New actionlistenerex();
   }
}
```

OUTPUT:



Itemlistener

- The class which processes the itemevent should implement this interface.
- The object of that class must be registered with a component.
- The object can be registered using the additemlistener() method.
- When the action event occurs, that object's item state changed method is invoked.

Interface methods

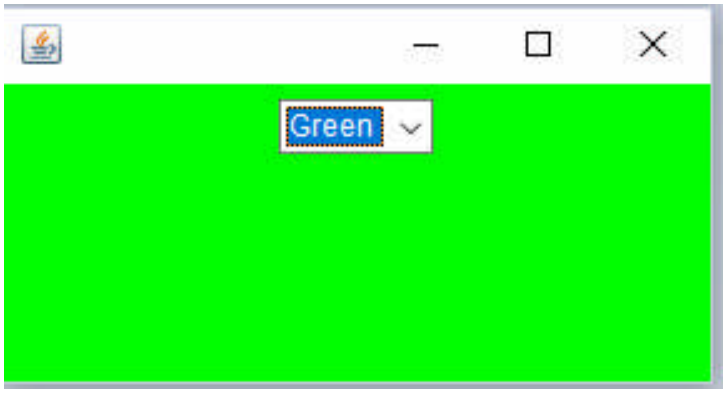
S.N.	Method & Description
1	**Void itemstatechanged(itemevente)** Invoked when an item has been selected or deselected by the user.

// program to change the background color of a frame according to the selection in choice control

```
Import java.awt.*;
Import java.awt.event.*;
```

```
Public class choiceexextends Frame implements itemlistener{
   Choice color;
Choiceex()
   {
Setsize(300,400);
Setvisible(true);
Setlayout(new flowlayout());
Color=new Choice();
Color.add("Red");
Color.add("Green");
Color.add("blue");

Add(color);
Color.additemlistener(this);

   }

Public void itemstatechanged(itemevente)

   {
If(color.getselectedindex()==0)
System.out.println(color.getitemcount());
Else if(color.getselectedindex()==1)
Setbackground(Color.green);
Else if(color.getselecteditem().equalsignorecase("Blue"))
Setbackground(Color.blue);
   }

Public static void main(String[] args) {
New choiceex();
   }
}
```

OUTPUT:



Windowlistener

- The class which processes the window event should implement this interface.
- The object of that class must be registered with a component.
- The object can be registered using the addwindowlistener() method.

Interface Methods

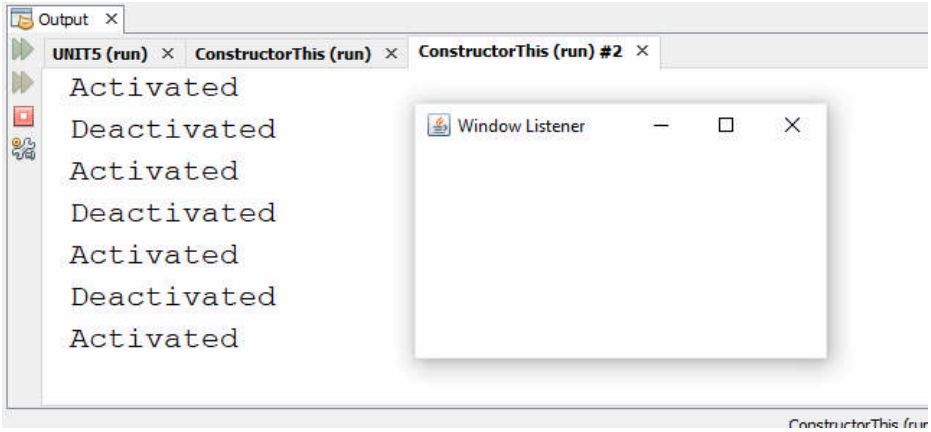
S.N.	Method & Description
1	**Void window activated (windowevente)** Invoked when the Window is set to be the active Window.
2	**Void window closed (windowevente)** Invoked when a window has been closed as the result of calling dispose on the window.
3	**Void window closing (windowevente)** Invoked when the user attempts to close the window from the window's system menu.
4	**Void window deactivated (windowevente)** Invoked when a Window is no longer the active Window.
5	**Void window deiconified (windowevente)** Invoked when a window is changed from a minimized to a normal state.
6	**Void windowiconified(windowevente)** Invoked when a window is changed from a normal to a minimized state.
7	**Void window opened (windowevente)** Invoked the first time a window is made visible.

// program to implement Window Listener methods

```
Import java.awt.*;
Import java.awt.event.*;
Public class windowex2 extends Frame implements windowlistener{
Windowex2()
  {
Super("Window Listener");
Setsize(300,300);
Setvisible(true);
Addwindowlistener(this);
```

```
    }
Public static void main(String[] args) {
New windowex2();
    }

    @Override
Public void windowopened(windowevente) {
System.out.println("Opened");
    }

    @Override
Public void windowclosing(windowevente) {
Dispose();
    }

    @Override
Public void windowclosed(windowevente) {
System.out.println("Closed");

        }

    @Override
Public void windowiconified(windowevente) {
System.out.println("Iconified");

    }

    @Override
Public void windowdeiconified(windowevente) {
System.out.println("deiconified");

    }

    @Override
Public void windowactivated(windowevente) {
System.out.println("Activated");

    }

    @Override
Public void windowdeactivated(windowevente) {
System.out.println("Deactivated");

    }
}
```

OUTPUT:



12.2.4 Using delegation event model

Using the delegation event model is actually quite easy. Just follow these two steps: 1. Implement the appropriate interface in the listener so that it will receive the type of event desired. 2. Implement code to register and unregister (if necessary) the listener as a recipient for the event notifications.

Keylistener

- The class which processes the keyevent should implement this interface. The object of that class must be registered with a component.
- The object can be registered using the add key listener() method.

Interface Methods

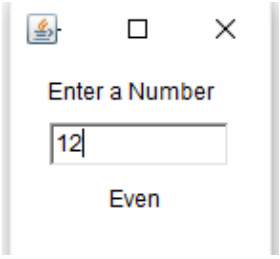
S.N.	Method & Description
1	**Void keypressed(keyevente)** Invoked when a key has been pressed.
2	**Void keyreleased(keyevente)** Invoked when a key has been released.
3	**Void keytyped(keyevente)** Invoked when a key has been typed.

//program to check whether the entered number is even or odd as soon as you typed the number

```
Import java.awt.*;
Import java.awt.event.*;
Public class keylistenerexextends Frame implements keylistener{
 Label l1,l2;
```

```
Textfieldt1;
Keylistenerex()
   {
Super("Key Listener Example");
Setsize(200,200);
Setvisible(true);
Setlayout(new flowlayout());

L1=new Label("Enter a Number");
T1=new textfield(10);
L2=new Label();

Add(l1);
Add(t1);
Add(l2);

T1.addkeylistener(this);

   }
Public void keytyped(keyevente)
   {
   }
Public void keypressed(keyevente)
   {
   }
Public void keyreleased(keyevente)
   {
      String s=t1.gettext();
Int n=Integer.parseint(s);
If(n%2==0)
L2.settext("Even");
Else
L2.settext("Odd");
   }
Public static void main(String[] args) {
New keylistenerex();
   }
}
```

OUTPUT:



Mouse listener

- The class which processes the mouse event should implement this interface.
- The object of that class must be registered with a component.
- The object can be registered using the add mouse listener() method.

Interface Methods

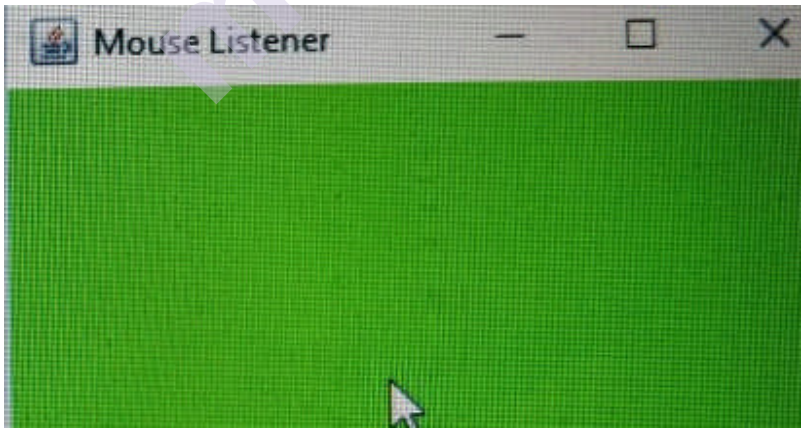
S.N.	Method & Description
1	**Void mouse clicked (mouseevente)** Invoked when the mouse button has been clicked (pressed and released) on a component.
2	**Void mouseentered (mouseevente)** Invoked when the mouse enters a component.
3	**Void mouseexited (mouseevente)** Invoked when the mouse exits a component.
4	**Void mousepressed (mouseevente)** Invoked when a mouse button has been pressed on a component.
5	**Void mousereleased(mouseevente)** Invoked when a mouse button has been released on a component.

//program to change background color of frame on every mouse event

```
Import java.awt.*;
Import java.awt.event.*;
Public class mouselistenerexextends Frame implements mouselistener{

Mouselistenerex(){
Super("Mouse Listener");
```

```
Addmouselistener(this);
Setsize(300,300);
Setlayout(null);
Setvisible(true);
   }
Public void mouseclicked(mouseevente) {
Setbackground(Color.red);
   }
Public void mouseentered(mouseevente) {
Setbackground(Color.green);
   }
Public void mouseexited(mouseevente) {
Setbackground(Color.blue);
   }
Public void mousepressed(mouseevente) {
Setbackground(Color.yellow);
   }
Public void mousereleased(mouseevente) {
Setbackground(Color.black);
   }
Public static void main(String[] args) {
New mouselistenerex();
}
}
```

OUTPUT:



❖ **Mouse motion listener**

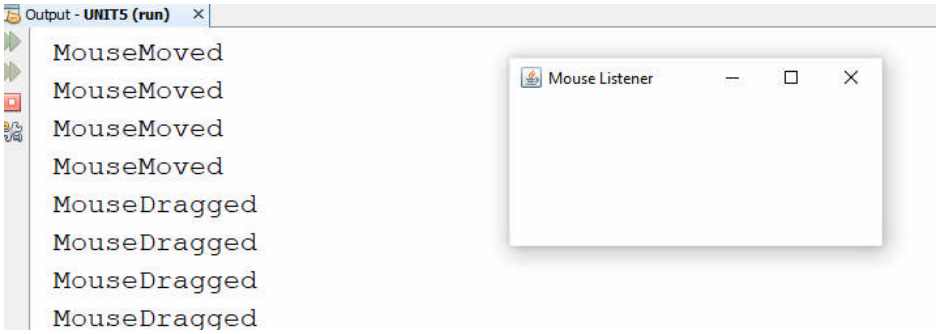
- The inter face mouse motion listener is used for receiving mouse motion events on a component.
- The class that process mouse motion events needs to implements this interface.

S.N.	Method & Description
1	**Void mousedragged(mouseevente)** Invoked when a mouse button is pressed on a component and then dragged.
2	**Void mousemoved(mouseevente)** Invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.

//program to implement Mouse Motion Listener

```
Import java.awt.*;
Import java.awt.event.*;
Public class mouse motion list enerexextends Frame implements
mousemotionlistener{

Mousemotionlistenerex()
   {
Super("Mouse Listener");
Addmousemotionlistener(this);

Setsize(300,300);
Setvisible(true);
   }

Public void mousedragged(mouseevente)
   {
System.out.println("mousedragged");
   }
Public void mousemoved(mouseevente)
   {
System.out.println("mousemoved");
   }

Public static void main(String[] args) {
New mousemotionlistenerex();
   }

}
```

OUTPUT:



13.2.5 adapter classes and inner classes

Java adapter classes provide the default implementation of listener interfaces. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it saves code.

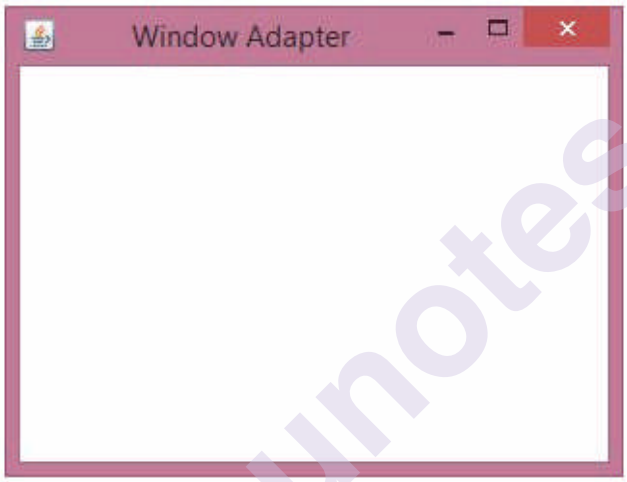
Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

Windowadapter Example

```
Import java.awt.*;
Import java.awt.event.*;
Public class adapterexample{
   Frame f;
Adapterexample(){
F=new Frame("Window Adapter");
F.addwindowlistener(new windowadapter(){
```

```
Public void windowclosing(windowevente) {
F.dispose();
        }
      });

F.setsize(400,400);
F.setlayout(null);
F.setvisible(true);
   }
Public static void main(String[] args) {
New adapterexample();
}
}
```

OUTPUT:



Inner class:

The inner class is defined inside the body of another class (known as an outer class). The class written within is called the nested class, and the class that holds the inner class is called the outer class. Java inner class can be declared private, public, protected, or with default access whereas an outer class can have only public or default access.

We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable. The **Syntax** is given below.

```
Class  Outerclass
{
        Class Innerclass
        {
        //code
         }
}
```

Types of inner classes in java:

1. **Nested Inner Class**
2. **Method Local Inner Class**
3. **Anonymous Inner Class**
4. **Static Nested Class**

Nested inner class in java:

A class created within the class and outside the method is known as **Nested Inner Class**. It can access the private instance variable of the outer class.

Example:

```
Package Demo;

Public class nestedinnerclass{
Class Inner {
Public void show() {
System.out.println("In a nested class method");
    }
  }
Public static void main(String[] args) {
Nestedinnerclass.Inner in = new nestedinnerclass().new Inner();
In.show();
  }
}
```

Method local inner class in java:

A class created within the method of the enclosing class is known as Method Local Inner Class. Since the local inner class is not associated with Object, we can't use private, public, or protected access modifiers with it. The only allowed modifiers are abstract or final.

Example:

```
Package Demo;

Public class methodlocalinnerclass{
Void outermethod() {
System.out.println("Inside outermethod");
    // Inner class is local to outermethod()
Class Inner {
Void innermethod() {
System.out.println("Inside innermethod");
      }
    }
    Inner y = new Inner();
```

```
Y.innermethod();
   }
Public static void main(String[] args) {
Methodlocalinnerclassouter = new methodlocalinnerclass();
Outer.outermethod();
   }
}
```

Anonymous inner class in java:

An inner class declared without a class name is known as an **anonymous inner class.** It is created for implementing an interface or extending class. Since an anonymous class has no name, it is not possible to define a constructor for an anonymous class. Its name is decided by the java compiler.

Example:

```
Package Demo;
Abstract class Animal{
Abstract void dog();
}
Class anonymousinnerclass{
Public static void main(String args[]){
  Animal p=new Animal(){
Void dog(){
System.out.println("Dog is an Animal.");
  }
 };
P.dog();
 }
}
```

Static nested class:

Static nested classes are not technically inner classes. They are like static members of the outer class. A static nested class is the same as any other top-level class and is nested for only packaging convenience. Because this is static in nature so this type of inner class doesn't share any special kind of relationship with an instance of the outer class. A static nested class cannot access non-static members of the outer class.

Example:

```
Package Demo;

Public class staticnestedclass{
Static class Nested_Demo {
Public void my_method() {
```

```
System.out.println("This is my nested class");
      }
    }

Public static void main(String args[]) {
Staticnestedclass.Nested_Demo nested = new
staticnestedclass.Nested_Demo();
Nested.my_method();
    }
}
```

12.3 LET US SUM UP

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

12.4 LIST OF REFERENCES

1. Core Java 8 for Beginners, Vaishali Shah, Sharnam Shah SPD 1st 2015
2. Java: The Complete Reference Herbert Schildt McGraw Hill 9th 2014

12.5 CHAPTER END EXERCISES

1. Explain Delegation Event Model in Java
2. Explain keyadapter class defined in Java with code segment.
3. Explain mouseadapter class defined in Java with code segment.
4. Develop a frame that has three radio buttons Red, Green, Blue. On Click of any one of them background color of the frame should change accordingly.
5. Explain the following interfaces:
   i) Key Listener ii) Mouse Listener
6. What is the use of adapter class in Java? Explain any one of the adapter classes defined in Java.
7. Define Event Handling in Java.
8. Describe Event Handling in Java.
9. Explain Two Event Handling Mechanisms.
10. Illustrate the Delegation Event Model in Java.
11. Describe Components of Event Handling.
12. Explain steps to handle an event in java.

13

ABSTRACT WINDOW TOOLKIT

Unit Structure

13.0 OBJECTIVES

This chapter would make you understand the following concepts:

- Define Abstract Windows Toolkit (AWT) in Java
- Describe Why AWT is platform dependent
- Explain Features of AWT in Java
- Illustrate AWT Hierarchy
- Define AWT Component

13.1 INTRODUCTION

In this chapter we will learn AWT. To develop the GUI based applications we have to use AWT.AWT stands for Abstract Windowing Toolkit. The set of classes and interfaces which are required to develop GUI components together are called “Toolkit”. The GUI components will be used to design GUI programs. Writing a program to display the created GUI components on the windows is called “windowing”. To display the components on the windows we need to take the support of graphics available in the operating system. For a developer, there is no direct interaction with the graphics and hence graphics is “Abstract” to the developer. Every GUI component will have a corresponding “PEER” class which is responsible to interact with the graphics of the operating system.

13.2 ABSTRACT WINDOW TOOLKIT

The Java AWT creates components by calling the subroutines of native platforms. Hence, an AWT GUI application will have the look and feel of Windows OS while running on Windows and Mac OS look and feel when running on Mac and so on. This explains the platform dependency of Abstract Window Toolkit applications.

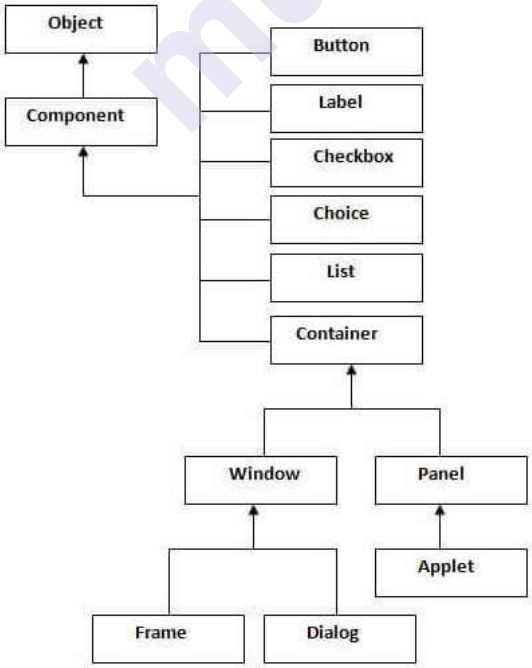
- Due to its platform-dependence and a kind of heavyweight nature of its components, it is rarely used in Java applications these days. Besides, there are also newer frameworks like Swing which are light-weight and platform-independent.
- Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java.
- Java AWT components are platform-dependent i.e. Components are displayed according to the view of operating system. AWT is heavyweight i.e. Its components are using the resources of OS.
- The java.awt package provides classes for AWT API such as textfield, Label, textarea, radiobutton, checkbox, Choice, List etc.

13.2.1 Window Fundamentals

The AWT defines windows according to a class hierarchy that adds functionality and specificity with each level. The two most common windows are those derived from Panel, which is used by applets, and those derived from Frame, which creates a standard application window.

13.2.2 Components, Container, Panel, Window, Frame, Canvas

The hierarchy of Java AWT classes are given below:



As shown in the above figure the root AWT component 'Component' extends from the 'Object' class. The component class is the parent of the other components including Label, Button, List, Checkbox, Choice, Container, etc.

A container is further divided into panels and windows. An Applet class derives from Panel while Frame and Dialog derive from the Window component.

Components are as follows:

- **Container**

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

- **Window**

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

- **Panel**

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

- **Frame**

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

- **Canvas**

The Canvas control represents a blank rectangular area where the application can draw or trap input events from the user. It inherits the Component class.

❖ **Component Class**

Method	**Description**
Public void add(Component c)	Inserts a component on this component.
Public void setsize (int width, int height)	Sets the size (width and height) of the component.
Public void setlayout (layout managerm)	Defines the layout manager for the component.
Public void setvisible (boolean status)	Changes the visibility of the component, by default false.

❖ **Creating Frames**

To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

- By extending Frame class (inheritance)
- By creating the object of Frame class (association)

Creating Frame by Association

The given example creates instance of empty Frame class with "MYFRAME" as title.

```
importjava.awt.*;
class frame1{
frame1(){
frame f=new Frame("MYFRAME");
f.setsize(300,300);  //frame size 300 width and 300 height
f.setvisible(true);     //now frame will be visible, by default not visible
}
public static void main(string args[]){
frame1 f=new frame1();
}}
```

OUTPUT:

Creating Frame by Inheritance

The given example creates instance of empty Frame class with "MYFRAME" as title.

```
import java.awt.*;
class first extends frame{
first(){
super("MYFRAME");  //frame title
setsize(300,300);//frame size 300 width and 300 height
setvisible(true);//now frame will be visible, by default not visible
}
public static void main(string args[]){
first f=new first();
}}
```

OUTPUT:

13.2.3 Components – Labels, Buttons, Check Boxes, Radio Buttons, Choice Menus, Text Fields, Text, Scrolling List, Scrollbars, Panels, Frames

Label

The object of Label class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly.

Constructors	Description
label()	Constructs an empty label.
label(String text)	Constructs a new label with the specified string of text, left justified.
label(string text, int alignment)	Constructs a new label that presents the specified string of text with the specified alignment.

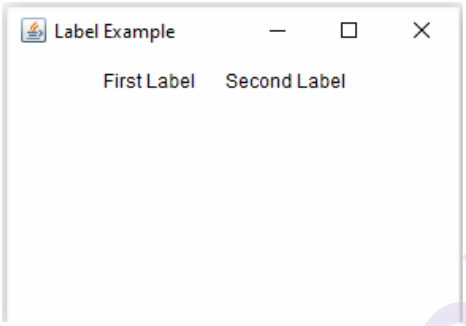
Methods	Description
string gettext()	Gets the text of this label.
void settext(String text)	Sets the text for this label to the specified text.
int getalignment()	Gets the current alignment of this label.
void setalignment (intalignment)	Sets the alignment for this label to the specified alignment.

//creates two labels

```
import java.awt.*;
public class labelex extends frame {
   label l1,l2;
labelex()
   {
super("label example");
setsize(300,300);
```

```
setvisible(true);
setlayout(new flowlayout());
L1=new label("first label");
L2=new label("second label");
add(l1);
add(l2);

   }
public static void main(string[] args) {
new labelex();
   }

}
```

OUTPUT:



Button

The button class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed.

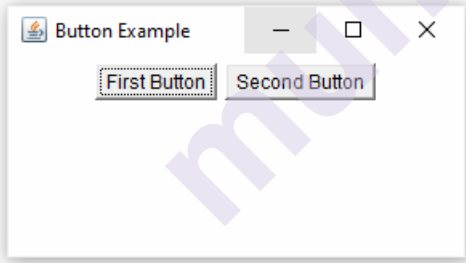
Constructors	Description
button()	Constructs a button with an empty string for its label.
button(string text)	Constructs a new button with specified label.

Methods	Description
void setlabel(string label)	Sets the button's label to be the specified string.
string getlabel()	Gets the label of this button
void add action listener (action listener l)	Adds the specified action listener to receive action events from this button.
void remove action listener (action listener l)	Removes the specified action listener so that it no longer receives action events from this button.

//creating button

```
import java.awt.*;

public class buttonex extends frame {
  button b1,b2;
buttonex()
   {
super("button example");
setsize(300,300);
setvisible(true);
setlayout(new flowlayout());
b1=new button("first button");
b2=new button("second button");
add(b1);
add(b2);

   }
public static void main(string[] args) {
new buttonex();
   }
}
```

OUTPUT



Text field

The object of a text field class is a text component that allows the editing of a single line text. It inherits text component class.

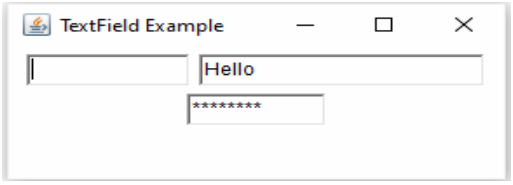
Constructors	Description
text field()	Constructs a new text field.
text field(int columns)	Constructs a new empty text field with the specified number of columns.
text field(string text)	Constructs a new text field initialized with the specified text.
text field(string text, int columns)	Constructs a new text field initialized with the specified text to be displayed, and wide enough to hold the specified number of columns.

Methods	Description
void settext(string t)	Sets the text that is presented by this text component to be the specified text.
string gettext()	Gets the text of the text field.
void setechochar (char c)	Sets the echo character for this text field.
char getechochar()	Gets the character that is to be used for echoing.
void addactionlistener (actionlistener l)	Adds the specified action listener to receive action events from this text field.
int getcolumns()	Gets the number of columns in this text field.
void setcolumns (intcolumns)	Sets the number of columns in this text field.

//creating textfield

```
import java.awt.*;

public class textfieldex extends frame {
textfield t1,t2;
textfieldex()
   {
super("button example");
setsize(300,300);
setvisible(true);
setlayout(new flowlayout());
t1=new textfield(10); // creates empty textfield
t2=new textfield("Hello", 20);
add(t1);
add(t2);
   }
public static void main(string[] args) {
new textfieldex();
   } }
```

OUTPUT



Text area

The object of a text area class is a multi line region that displays text. It allows the editing of multiple line text. It inherits text component class.

➢ **Field**

Following are the fields for java.awt.text area class:

- **Static int SCROLLBARS_BOTH** -- Create and display both vertical and horizontal scrollbars.
- **Static int SCROLLBARS_HORIZONTAL_ONLY** -- Create and display horizontal scrollbar only.
- **Static int SCROLLBARS_NONE** -- Do not create or display any scrollbars for the text area.
- **Static int SCROLLBARS_VERTICAL_ONLY** -- Create and display vertical scrollbar only.

Constructors	**Description**
textarea()	Constructs a new text area with the empty string as text.
textarea (int rows, int columns)	Constructs a new text area with the specified number of rows and columns and the empty string as text.
textfield (string text)	Constructs a new text area initialized with the specified text.
textfield (string text, int rows, int columns)	Constructs a new text area with the specified text, and with the specified number of rows and columns.
textarea (string text, int rows, int columns, int scrollbars)	Constructs a new text area with the specified text, and with the rows, columns, and scroll bar visibility as specified.

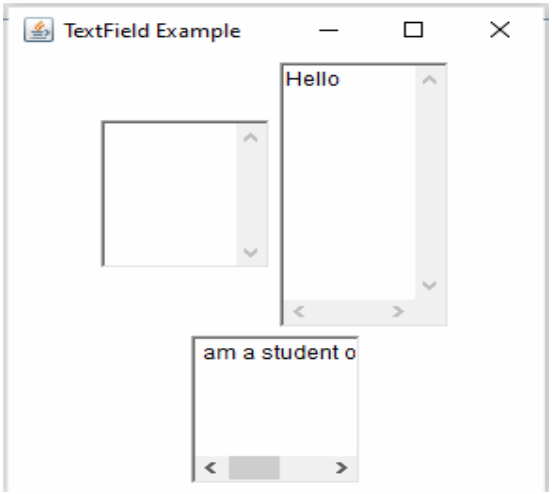
Methods	Description
void settext(string t)	Sets the text that is presented by this text component to be the specified text.
string gettext()	Gets the text of the textarea.
int getcolumns()	Gets the number of columns in this text area.
void setcolumns(int columns)	Sets the number of columns in this text area.
int getrows()	Gets the number of rows in this text area.
void setrows(int rows)	Sets the number of rows in this text area.
void append(string str)	Appends the given text to the text area's current text.
void insert(string str, int pos)	Inserts the specified text at the specified position in this text area.

//creating textarea

```
import java.awt.*;

public class textareaex extends frame {
textarea t1,t2,t3;
textareaex()
   {
super("textfield example");
setsize(300,300);
setvisible(true);
setlayout(new flowlayout());
t1=new textarea(5,10); // creates empty textfield
t2=new textarea("Hello",10, 10);
t3=new textarea("i am a student of VSIT
college",5,10,textarea.SCROLLBARS_HORIZONTAL_ONLY);

add(t1);
add(t2);
add(t3);

   }
```

```
public static void main(string[] args) {
new textareaex();
   }
}
```

OUTPUT:



Checkbox

The Checkbox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a Checkbox changes its state from "on" to "off" or from "off" to "on".

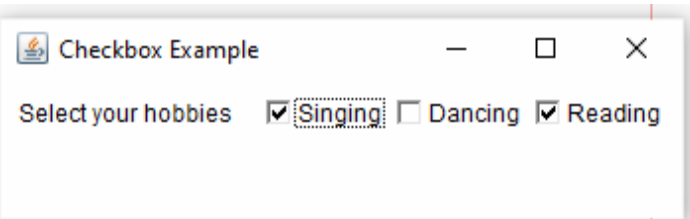
Constructor	Description
checkbox()	Creates a check box with an empty string for its label.
checkbox(string label)	Creates a check box with the specified label.
checkbox(String label, boolean state)	Creates a check box with the specified label and sets the specified state.
checkbox(String label, boolean state, check boxgroup group)	Constructs a Checkbox with the specified label, set to the specified state, and in the specified check box group.
checkbox(string label, checkbox group group, boolean state)	Creates a check box with the specified label, in the specified check box group, and set to the specified state.

Methods	Description
void setlabel (string label)	Sets this check box's label to be the string argument.
string getlabel()	Gets the label of this check box.
void setstate (boolean state)	Sets the state of this check box to the specified state.
boolean getstate()	Determines whether this check box is in the on or off state.
void additemlistener (itemlistener l)	Adds the specified item listener to receive item events from this check box.

//creating Checkbox

```
import java.awt.*;
public class checkboxex extends frame {
   label l1;
   checkbox c1,c2,c3;
checkboxex()
   {
super("checkbox example");
setsize(300,300);
setvisible(true);
setlayout(new flowlayout());
l1=new label("select your hobbies");
c1=new checkbox("singing",true);
c2=new checkbox("dancing",false);
c3=new checkbox("reading",true);
add(l1); add(c1);  add(c2);  add(c3);
   }
public static void main(string[] args) {
new checkboxex();
   }  }
```

OUTPUT:



Checkbox group

The object of checkbox group class is used to group together a set of Checkbox. At a time only one check box button is allowed to be in "on" state and remaining check box button in "off" state. It inherits the object class.

Constructor	Description
Checkbox group() ()	Creates a new instance of checkbox group.

Methods	Description
checkbox getselected checkbox()	Gets the current choice from this check box group.
void setselected checkbox (checkbox box)	Sets the currently selected check box in this group to be the specified check box.

```
//creating radiobuttons
import java.awt.*;
public class radiobuttonex extends Frame {
   label l1;
   checkbox c1,c2,c3;
checkboxgroup cg;
radiobuttonex()
   {
super("radiobutton example");
setsize(300,300);
setvisible(true);
setlayout(new flowlayout());
cg=new checkboxgroup();
l1=new label("select your color");
c1=new checkbox("red",cg,true);
c2=new checkbox("green",cg,false);
c3=new checkbox("blue",cg,false);
add(l1);
add(c1);
add(c2);
add(c3);
   }
public static void main(string[] args) {
new radiobuttonex();
   }

}
```

OUTPUT:

Choice

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits Component class.

Constructor	Description
Choice()	Creates a new choice menu.

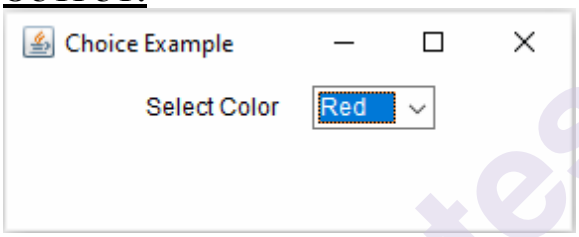
Methods	Description
void add(string item)	Adds an item to this Choice menu.
void additemlistener (itemlistener l)	Adds the specified item listener to receive item events from this Choice menu.
string getitem (int index)	Gets the string at the specified index in this Choice menu.
int getitemcount()	Returns the number of items in this Choice menu.
int getselectedindex()	Returns the index of the currently selected item.
string getselecteditem()	Gets a representation of the current choice as a string.
void insert(string item, int index)	Inserts the item into this choice at the specified position.
void remove(int position)	Removes an item from the choice menu at the specified position.
void remove(string item)	Removes the first occurrence of item from the Choice menu.
void removeall()	Removes all items from the choice menu.

//creating dropdown list

```
import java.awt.*;
public class choiceex extends frame {
   label l1;
   choice c;
choiceex()
   {
super("choice example");
```

```
setsize(300,300);
setvisible(true);
setlayout(new flowlayout());
l1=new label("select color");
c=new choice();
c.add("red");
c.add("green");
c.add("blue");
add(l1);
add(c);
   }
public static void main(string[] args) {
new choiceex();
   } }
```

OUTPUT:



List

The object of List class represents a list of text items. By the help of list, user can choose either one item or multiple items. It inherits Component class.

Constructor	Description
list()	Creates a new scrolling list.
list(int rows)	Creates a new scrolling list initialized with the specified number of visible lines.
list(int rows, boolean multiplemode)	Creates a new scrolling list initialized to display the specified number of rows.

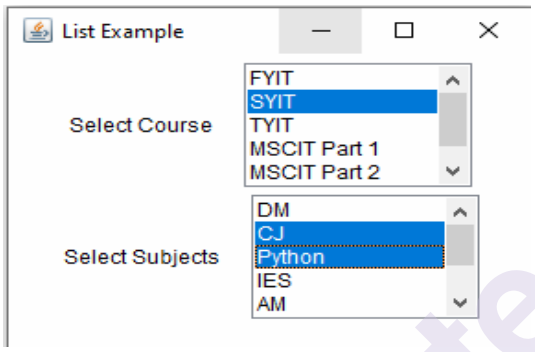
Methods	Description
void add(string item)	Adds the specified item to the end of scrolling list.
void add(string item, int index)	Adds the specified item to the the scrolling list at the position indicated by the index.
int getitemcount()	Gets the number of items in the list.
string getitem(int index)	Gets the item associated with the specified index.

string[] getitems()	Gets the items in the list.
int getselectedindex()	Gets the index of the selected item on the list,
int[] getselectedindexes()	Gets the selected indexes on the list.
string getselecteditem()	Gets the selected item on this scrolling list.
string[] getselecteditems()	Gets the selected items on this scrolling list.
void remove(int position)	Removes the item at the specified position from this scrolling list.
void remove(string item)	Removes the first occurrence of an item from the list.
void removeall()	Removes all items from this list.

```
// creating scrolling list
import java.awt.*;
public class listex extends frame {
   label l1,l2;
   list course,subject;
listex()
   {
super("list example");
setsize(300,300);
setvisible(true);
setlayout(new flowlayout());
l1=new label("select course");
l2=new label("select subjects");
course=new list(5);
subject=new list(5,true); //multi select list
course.add("fyit");
course.add("syit");
course.add("tyit");
course.add("mscit part 1");
course.add("mscit part 2");
course.add("fybms");

subject.add("dm");  //0
subject.add("cj");  //1
subject.add("ies");  //2
subject.add("am");
subject.add("ma");
```

```
subject.add("ip");
subject.add("dbms");
subject.add("python",2);

add(l1);  add(course);  add(l2);
add(subject);
   }
public static void main(string[] args) {
new listex();
   }  }
```

OUTPUT



13.3 PROGRAMS

a. Create an AWT application to create a Frame with a Button named cube, a Label and a Text Field. Click of the button should display cube of that number in the Label.

```
import java.awt.*;
import java.awt.event.*;
public class numex extends frame implements actionlistener{
label l1,l2;
textfield t1;
button b1;
numex()
   {
   super("cube");
setlayout(new flowlayout());
   l1=new label("enter a number:-");
   l2=new label();
   t1=new textfield();
   b1=new button("cube");
   b1.addactionlistener(this);
```

```
    add(l1);
    add(t1);
    add(b1);
    add(l2);
setsize(300,300);
    setvisible(true);
}
    public static void main(string[] args) {
        new numex();
    }

    public void actionperformed(actionevent e) {
        int n=integer.parseint(t1.gettext());
        l2.settext((n*n*n)+"");
    }
}
```

b. Develop a frame that has three radio buttons Red, Green, Blue. On Click of any one of them background color of the frame should change accordingly.

```
import java.awt.*; import java.awt.event.*;
public class colorclass extends frame implements itemlistener
{
 checkbox r1,r2,r3;
checkboxgroup chg;
colorclass()
{
setlayout(new flowlayout());
chg=new checkboxgroup();
r1=new checkbox("red",chg,true);
r2=new checkbox("green",chg,false);
r3=new checkbox("blue",chg,false);
add(r1); add(r2); add(r3);
r1.additemlistener(this); r2.additemlistener(this); r3.additemlistener(this);
setbackground(color.red);
setsize(500,500);
setvisible(true);
}
public static void main(string args[])
{
new colorclass();
}
```

```
@override public void item state changed(item event e)
{
if(e.getsource()==r1)
setbackground(color.red);
else if(e.getsource()==r2)
setbackground(color.green);
else setbackground(color.blue);
}
}
```

c. Write a program to demonstrate the use of Canvas.

Canvas control represents a rectangular area where application can draw something or can receive inputs created by user.

```
 import java.awt.*;
public class canvasexample
{
public canvasexample()
{
frame f= new frame("canvas example");
f.add(new mycanvas());
f.setlayout(null);
f.setsize(400, 400);
f.setvisible(true);
}
public static void main(string args[])
{
new canvasexample();
}
}
class mycanvas extends canvas
{
public mycanvas()
{
setbackground (color.gray);
setsize(300, 200);
}
public void paint(graphics g)
{
g.setcolor (color.red);
g.filloval(75, 75, 150, 75);
}
}
```

13.4 LET US SUM UP

The java.awt.event package provides many event classes and Listener interfaces for event handling. The java.awt package provides a great deal of functionality and flexibility. With the help of this notes, you should get an excellent grasp of the java.awt, java.awt.event.

13.5 LIST OF REFERENCES

1. Core Java 8 for Beginners, Vaishali Shah, Sharnam Shah SPD 1st 2015
2. Java: The Complete Reference Herbert Schildt McGraw Hill 9th 2014

13.6 CHAPTER END EXERCISES

1. Define Component, Panel, Canvas, Window and Frame.
2. Write a short note on checkbox and checkbox group class.
3. List various layouts in AWT and Explain Border Layout with example.
4. Write the constructors and methods of checkbox class. Also explain the use of checkbox group class.
5. Explain any two overloaded constructors and three methods of Label class.
6. Write a short note on Choice and List class.
7. Write a program to display "Good Morning" in blue with font size 20 and font name Times New Roman in bold and italic.
8. Develop a frame that has three radio buttons Red, Green, Blue. On Click of any one of them background color of the frame should change accordingly.
9. Write aprogram to Design a AWT program to print the factorial for an input value.
10. Design a Registration Form.
11. Explain the hierarchy of AWT components.
12. Explain any two overloaded constructors and three methods of class Text Field.
13. What is the use of adapter class in Java? Explain any one of the adapter classes defined in Java.
14. Create an AWT application to create a Frame with a Button named cube, a Label and a Text Field. Click of the button should display cube of that number in the Label.
15. Explain Choice class along with constructors in detail.

14

LAYOUTS

Unit Structure

14.0 OBJECTIVES

This chapter would make you understand the following concepts:

- Flow Layout
- Grid Layout
- Border Layout
- Card Layout
- Programs using layouts

14.1 INTRODUCTION

In this unit we will learn different layouts. Layouts defines how UI elements should be organized on the screen and provide a final look and feel to the GUI.The layout is used to enhance the look and feel of the application. To arrange the components in a container, the various layout classes can be used such as Flow layout and Border Layout. These layouts use relative positioning to place the components on the container, which means the components automatically adjust their position according to the frame size.

14.2 LAYOUT MANAGERS

The layout managers are used to arrange components in a particular manner. Layout manager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. Java.awt.flowlayout
2. Java.awt.gridlayout
3. Java.awt.borderlayout
4. Java.awt.cardlayout

14.2.1 Flow Layout

The flowlayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.

Fields of flowlayout class

- Public static final int LEFT
- Public static final int RIGHT
- Public static final int CENTER
- Public static final int LEADING
- Public static final int TRAILING

Constructors

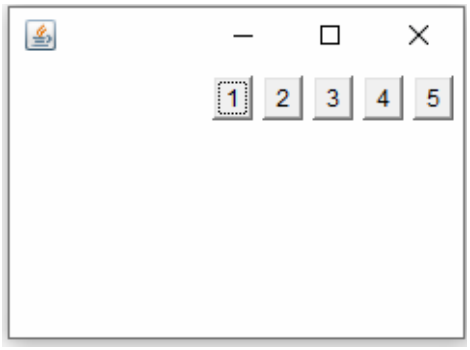
Flowlayout()	Creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
Flowlayout(int align)	Creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
Flowlayout(int align, int hgap, int vgap)	Creates a flow layout with the given alignment and the given horizontal and vertical gap.

Example

```
Import java.awt.*;
Public class flowlayoutex extends Frame{
Button b1,b2,b3,b4,b5;
Flowlayoutex(){

   B1=new Button("1");
   B2=new Button("2");
   B3=new Button("3");
   B4=new Button("4");
   B5=new Button("5");

   Add(b1);add(b2);add(b3);add(b4);add(b5);

Setlayout(new flowlayout(flowlayout.RIGHT));
   //setting flow layout of right alignment
```

```
Setsize(300,300);
   Setvisible(true);
}
Public static void main(String[] args) {
   New flowlayoutex();
}
}
```

OUTPUT:



14.2.2 Grid Layout

The gridlayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

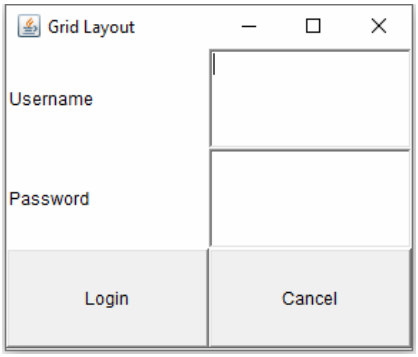
Constructor

Gridlayout()	Creates a grid layout with one column per component in a row.
Gridlayout(int hgap, int vgap)	Creates a grid layout with the given rows and columns but no gaps between the components.
Gridlayout(int rows, int columns, int hgap, int vgap)	Creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

```
// Example
Import java.awt.*;
Import java.awt.event.*;
Public class gridlayoutex extends Frame {
 Label lbname,lbpass;
Textfield txtname,txtpass;
Button btnlogin,btncancel;
Gridlayoutex()
 {
```

```
Super("Grid Layout");
Setsize(300,400);
Setvisible(true);
Setlayout(new gridlayout(3,2));
Lbname=new Label("Username");
Lbpass=new Label("Password");
Txtname=new textfield(20);
Txtpass=new textfield(20);
Txtpass.setechochar('*');
Btnlogin=new Button("Login");
Btncancel=new Button("Cancel");
Add(lbname);
Add(txtname);
Add(lbpass);
Add(txtpass);
Add(btnlogin);
Add(btncancel);

Addwindowlistener(new windowadapter(){
Public void windowclosing(windowevent e)
{
Dispose();
}
});

 }
   Public static void main(String[] args) {
      New gridlayoutex();
  }
}
```

OUTPUT:



14.2.3 Border Layout

The borderlayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The borderlayout provides five constants for each region:

- Public static final int NORTH
- Public static final int SOUTH
- Public static final int EAST
- Public static final int WEST
- Public static final int CENTER

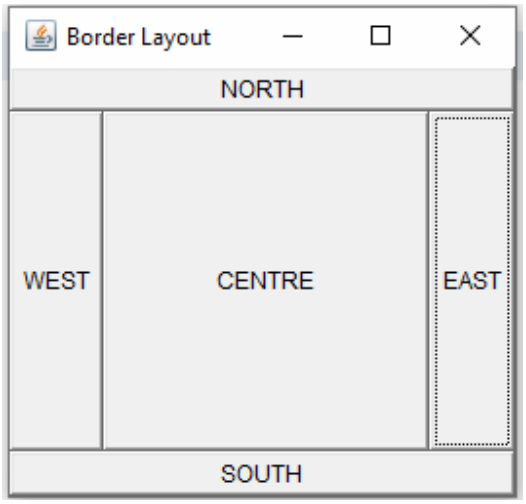
Constructor

Borderlayout()	Creates a border layout but with no gaps between the components.
Jborderlayout(int hgap, int vgap)	Creates a border layout with the given horizontal and vertical gaps between the components.

Example

```
Import java.awt.*;

Public class borderlayoutex extends Frame {
    Button b1,b2,b3,b4,b5;
Borderlayoutex(){
Super("Border Layout");
Setsize(500,500);
Setvisible(true);
B1=new Button("EAST");
B2=new Button("WEST");
B3=new Button("NORTH");
B4=new Button("SOUTH");
B5=new Button("CENTRE");

Add(b1,borderlayout.EAST);
Add(b2,borderlayout.WEST);
Add(b3,borderlayout.NORTH);
Add(b4,borderlayout.SOUTH);
Add(b5,borderlayout.CENTER);
   }
Public static void main(String[] args) {
New borderlayoutex();
   } }
```

OUTPUT:



14.2.4 Card Layout

The cardlayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as cardlayout.

Constructor

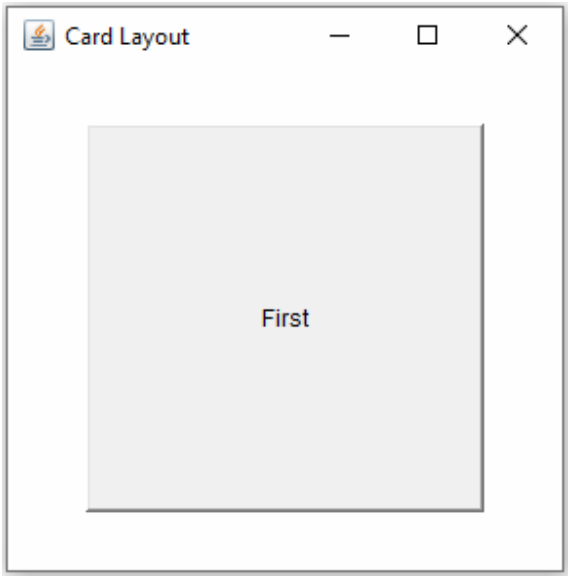
Cardlayout()	Creates a card layout with zero horizontal and vertical gap.
Cardlayout(int hgap, int vgap)	Creates a card layout with the given horizontal and vertical gap.

Method	**Description**
Public void next(Container parent)	Is used to flip to the next card of the given container.
Public void previous(Container parent)	Is used to flip to the previous card of the given container.
Public void first(Container parent)	Is used to flip to the first card of the given container.
Public void last(Container parent)	Is used to flip to the last card of the given container.
Public void show(Container parent,String name)	Is used to flip to the specified card with the given name.

// Example

```
Import java.awt.*;
Import java.awt.event.*;

Public class cardlayoutex extends Frame implements actionlistener {
   Button b1,b2,b3;
Cardlayout c;
   Panel p;
Cardlayoutex()
   {
Super("Card Layout");
Setsize(300,300);
Setvisible(true);
P=new Panel();
B1=new Button("First");
B2=new Button("Second");
B3=new Button("Third");

C=new cardlayout(40,30);
P.setlayout(c);
P.add("a",b1);
P.add("b",b2);
P.add("c",b3);
B1.addactionlistener(this);
B2.addactionlistener(this);
B3.addactionlistener(this);
Add(p);
   }

Public void actionperformed(actionevent e)
   {
C.next(p);
   }
Public static void main(String[] args) {
New cardlayoutex();
   }
}
```

OUTPUT:



14.3 LET US SUM UP

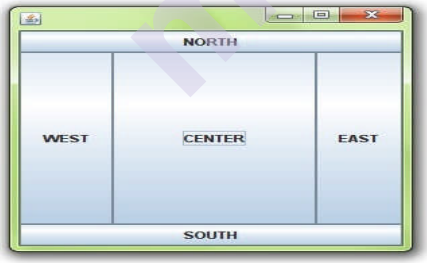
LayoutManager is an interface that is implemented by all the classes of layout managers. The Layout Managers are used to arrange components in a particular manner. Layout Manager is an interface that is implemented by all the classes of layout managers.

The default layout of frame is border layout. It is used to arrange component in five regions east,west,center,north and south

Constructor

1.BorderLayout()

2.BorderLayout(int horz,int vert)



14.4 LIST OF REFERENCES

1. Core Java 8 for Beginners, Vaishali Shah, Sharnam Shah SPD 1st 2015
2. Java: The Complete Reference Herbert Schildt McGraw Hill 9th 2014

S.Y. B.Sc. (IT)

SEMESTER - IV(CBCS)

CORE JAVA

SUBJECT CODE :USIT401

Prof. Suhas Pednekar
Vice-Chancellor,
University of Mumbai

Prof. Ravindra D. Kulkarni
Pro Vice-Chancellor,
University of Mumbai

Prof. Prakash Mahanwar
Director,
IDOL, University of Mumbai

Programme Co-ordinator : **Shri Mandar Bhanushe**
Head, Faculty of Science and Technology
IDOL, University of Mumbai, Mumbai

Course Co-ordinator : **Ms. Gouri Sawant**
Asst. Professor, B.Sc. IT,
IDOL, University of Mumbai, Mumbai

Editor : **Dr Asif Rampurawala**
Asst. Professor,
Vidyalankar college of Information Technology,
Wadala, Mumbai

Course Writers : **Ms. Pragati Ubale**
Asst. Professor,
Satish Pradhan Dnyanasadhana College, Thane

: **Mr Vinayak Pujari**
Asst. Professor,
I.C.S. College of Arts, Commerce and
Science, khed, Ratnagiri

: **Mr Tejas R. Jadhav**
Asst. Professor,
VPM's B.N.Bandodkar College of Science
(Autonomous), Thane.

: **Ms Pallavi Tawde**
Asst. Professor,
Vidyalankar college of Information Technology,
Wadala, Mumbai

December 2021, Print - I

Published by : Director
Institute of Distance and Open Learning ,
University of Mumbai,Vidyanagari, Mumbai -400 098.

DTP Composed : Mumbai University Press
Printed by Vidyanagari, Santacruz (E), Mumbai - 400 098

CONTENTS

14.5 CHAPTER END EXERCISES

1. Write a program to display a content using Flow Layout.
2. What is the role of layout manager? What is the default layout of Frame? Explain its working.
3. Write a program to display a registration page using Grid Layout.
4. Write a program to display a home page using Border Layout.
5. Write a program to display a about page using Card Layout.

❖❖❖❖