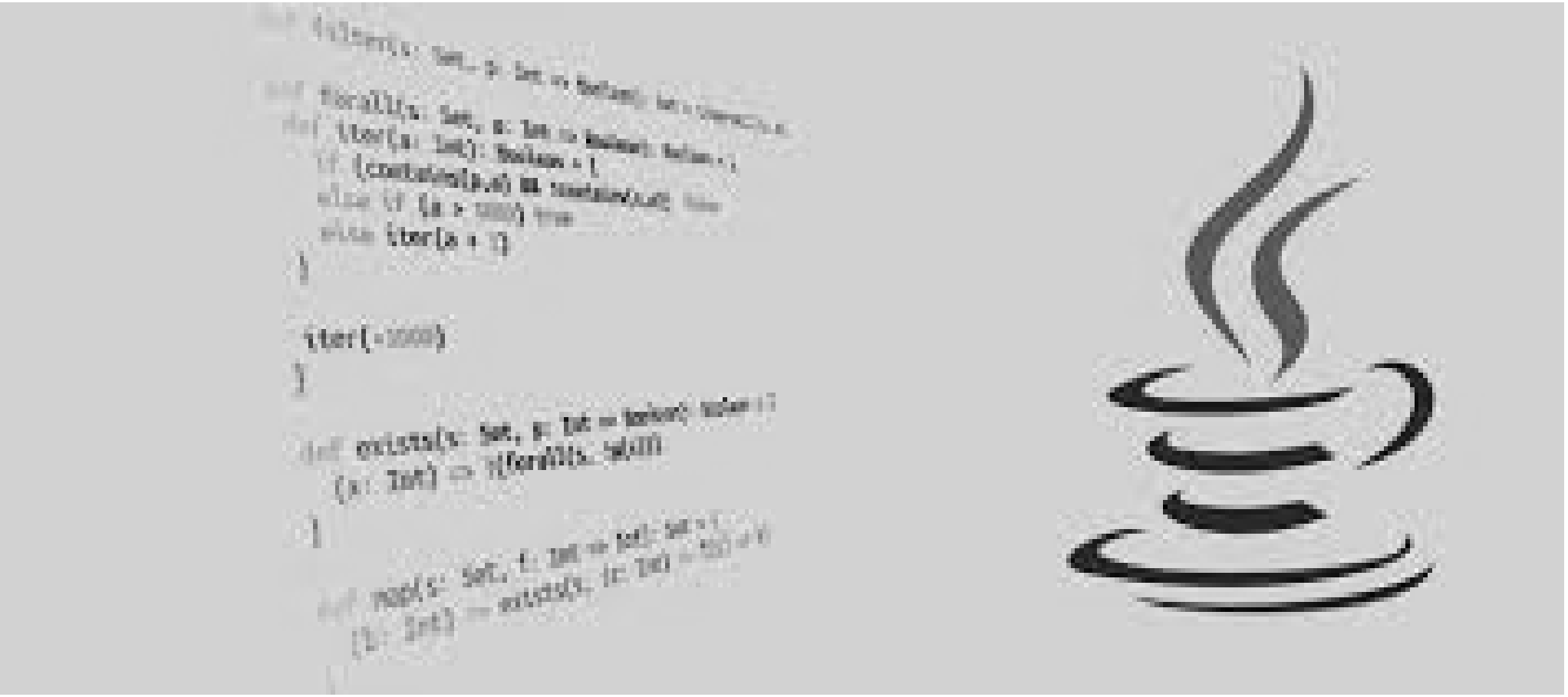
OOP with Java



DAYANANDA V N
Lecturer in Computer Science and Engg.
Acharya polytechnic, Bangalore

UNIT 1

INTRODUCTION OF JAVA

SYLLABUS

Fundamentals of Object Oriented Programming- Introduction, Object oriented Paradigm, Basic Concepts of OOP, Benefits of OOP, Applications of OOP.
Java Evolution - Java history, Java Features, How Java Differs from C and C++, Java and World Wide Web, Java Environment, Simple Java Program, An Application with Two Classes, Java Program Structure, Java Tokens, Java Statements, Implementing a Java Program, Java Virtual Machine, Command Line Arguments, Programming Style, Constants, Variables, Data Types, Scope of Variables, Symbolic Constants, Type Casting, Standard Default Values, Special Operators, Mathenfatical Functions, Labelled Loops (break & Continue) Operators and Expressions, Decision Making, Branching & Looping.

SYNOPSIS

Object oriented programming (OOP) is an approach to program organization and development, which attempts to eliminate some of the pitfalls of conventional programming methods by incorporating the best of structured programming features with several new concepts. It is a new way of organizing and developing programs and has nothing to do with any particular language. Languages that support OOP features include small talk, c, c++, Ada and pascal. C++ is an extension of 'C' language. C++ is basically a procedural language with object oriented extension. Java, a pure objected oriented language is one of the recent languages added to this list, the latest one being e#.

1. Give the definition of object oriented programming and how it is different from procedure oriented programming.

Ans. Object oriented programming is an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies such modules on demand.

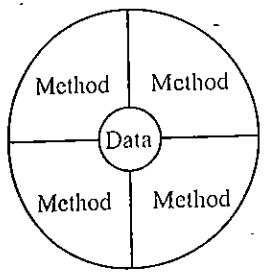
Difference between object oriented and procedure oriented programming.

	Object Oriented Programming	**Procedure Oriented Programming.**
1. Divided into	In OOP, program is divided into parts called objects.	In pop, program is divided into small parts called functions.
2. Importance	In OOP, importance is given to data rather than procedures or functions.	In pop importance given to data but functions.

3. Approach	OOP follows bottom up approach	Pop follows top down approach
4. Access Specifies	OOP has access specifiers named public, private, protected etc.	Pop does not have any access specifier.
5. Data moving	In OOP, objects can move and communicate with each other through member functions	In pop, Data can move freely from function to function in the system.
6. Expansion	OOP provides an easy way to add new data	To add new data and function in pop is not so easy.
7. Data hiding	OOP provides data hiding, so provides more security.	Pop does not have any proper way for hiding data, so it is less secure.
8. Overloading	In OOP, overloading is possible in the form of function overloading and operator overloading.	In pop, overloading is not possible.
Example	C++, Java, VB. NET	C, VB, FORTRAN, Pascal.

2. Explain the organization of data and methods in an object oriented programming.

Ans.



Object = Data + Methods.

OOP treats data as a critical element in the program development and does not allow it to flow freely around the system. It ties data more closely to the functions that operate on it and protects it from unintentional modification by other functions.

OOP allows us to decompose a problem into a number of entities called objects and then build data and functions around these entities. The combination of data and methods make an object.

The data of an object can be accessed only by the methods associated with that object. Methods of one object can access the methods of other objects.

3. List the unique advantages of an object oriented programming.

Ans. (i) Through inheritance, we can eliminate redundant code and extend the use of

existing classes.

(ii) The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.

(iii) It is easy to partition the work in a project based on objects.

(iv) It is possible to map objects in the problem domain to those objects in the program.

(v) It is possible to have multiple objects to coexist without any interference.

(vi) Software complexity can be easily managed.

(vii) We can build programs from the standard working modules that communicate with one another. This leads to saving of development time and higher productivity.

(viii) Message passing techniques for communication between objects is much simpler.

(ix) Object oriented systems can be easily upgraded from small to large systems.

(x) The data centered design approach enables us to capture more details of a model in an implementable form.

4. Define object and classes. Explain representation of an object with an example.

Ans. Object: Objects are the basic runtime entities. They may represent a person, a place, a bank account a table of data or any item that the program may handle.

Class: A class is a collection of objects of similar type classes are user defined data types and behaves like the built in types of programming language.

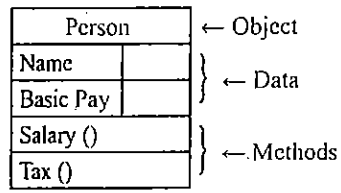
Representation of an object:

Object contains data and code to manipulate the data. A class may be thought of as a 'data type' and object as a 'variable' of that data type. Once a class has been defined, we can create any number of objects belonging to that class. Each object is associated with the data of type class with which they are created.

For example, If **fruit** has been defined as a class, then the statement

fruit mango *

Will create an object **mango** belonging to the class **fruit.**



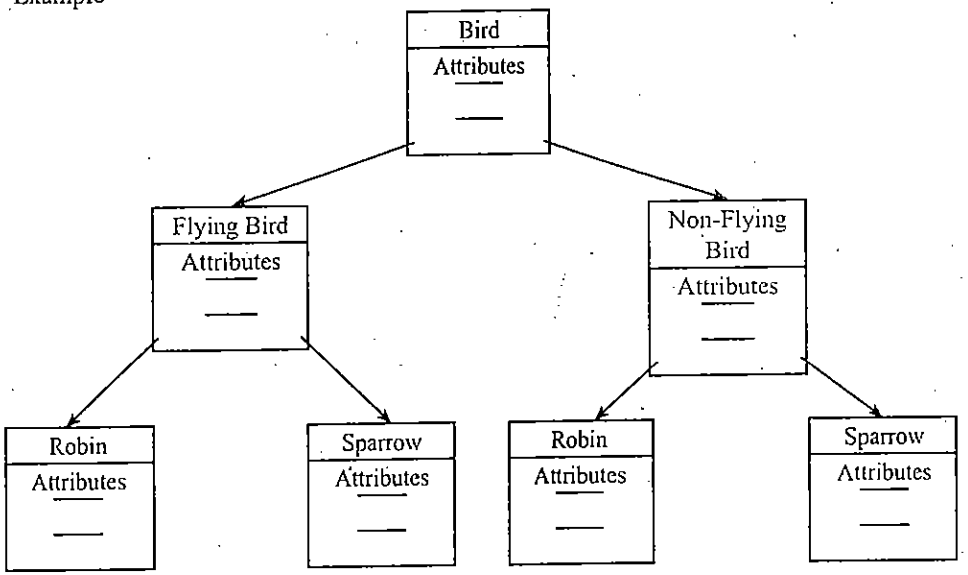
5. Define the following. (a) Data abstraction (b) Data encapsulation (c) Inheritance (d) Polymorphism.

Data abstraction: Abstraction refers to the act of representing essential feature without including the background details. Classes use the concept of abstraction and it is defined as a list of abstract attributes and methods that operate on these attributes.

Data Encapsulation: The wrapping up of data and methods into a single unit (called class) is known as encapsulation. Data encapsulation is the most striking feature of a class. The data is not accessible to the outside world and only those methods, which are wrapped in the class can access it. Theses methods provide the interface between the objects data and the program. This insulation of the data from direct access by the program is called **data hiding.**

Inheritance: It is the process by which objects of one class acquire the properties of objects of another class. Inheritance supports the concept of hierarchical classification.

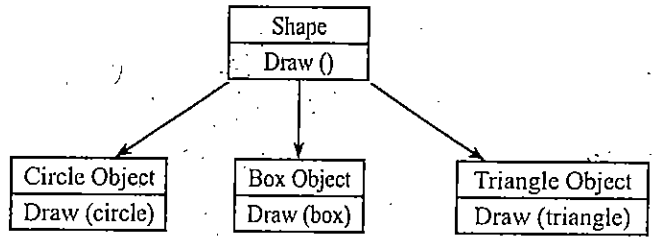
Example



Bird Robin is a part of the class flying bird, which is again a part of the class bird.

Polymorphism: It is the ability to take more than one form. Polymorphism plays an important role in allowing objects having different internal structures to share the same external interface.

Example



6. List the areas of application of OOPs technology.

Ans. (i) Real time systems.

(ii) Simulation and modelling.

(iii) Object oriented databases.

(iv) Hypertext.

(v) Neural networks and parallel programming.

(vi) Decision support and office automation system.

(vii) CIM/CAD system.

(viii) Artificial intelligence and expert systems.

7. Distinguish between objects and classes.

Object	Class
1. Object is an instance of a class.	1. Class is a template from which objects are created.
2. Object is a real time entity.	2. Class is a group of similar objects.
3. Object is a physical entity.	3. Class is a logical entity.
4. Object is created through new keyword. Ex: Student S1 = new student ( ) ;	4. Class is declared using class keyword. Ex: Class student { } .
5. Object is created many times as per requirement.	5. Class is declared once.
6. Object allocates memory when it is created	6. Class doesn't allocates memory when it is created.

8. Distinguish between Inheritance and polymorphism.

	Inheritance	Polymorphism
1. Basic	Inheritance is creating a new class using the properties of the already existing class.	Polymorphism is basically a common interface for multiple form.
2. Implementation	Inheritance is basically implemented on classes.	Polymorphism is basically implemented on functions or methods.
3. Use	Supports the concept of re usability in OOP and reduces the length of code.	Allows object to decide which form of the function to be invoked when, at compile time as well as run time.
4. Forms	It may be single, multiple, multilevel, hierarchical and hybrid inheritance.	It may be compile time polymorphism (overloading) and runtime polymorphism (overriding)

9. List the major differences between C and Java.

C	Java
1. It is purely conventional language.	1. It is purely object conventional language.
2. It includes structures and unions.	2. It includes class.
3. It supports pointer.	3. It does not support pointer.
4. It includes header file.	4. It does not include header file.
5. It does not support instance of operation.	5. It supports instance of operation.
6. It includes size of operator.	6. It does not supports size of operator.

10. List the difference between C++ and Java.

C++	Java
1. It is basically C with object oriented concepts.	1. It is pure object oriented language.
2. It supports operator overloading.	2. It does not supports operator overloading.
3. It supports multiple inheritance.	3. It does not supports multiple inheritance.
4. It supports global variables.	4. It does not supports global variables.
5. It uses destructor function.	5. It uses finalize function.
6. It includes header file.	6. It does not include header file.

11. Java is plat form independent language Justify.

Ans. The most significant contribution of Java over other languages is its portability. Java programs can be easily moved from one computer system to another, anywhere and anytime. Changes and upgrades in operating systems, processors and system resources will not force any changes in Java program.

Java ensures portability in two ways.

i. Java compiler generates byte code instructions that can be implemented on any machine.

ii. The size of the primitive data types are machine independent. Thus Java is a platform independent language.

12. Discuss how Java is more secured than other language.

Ans. An applet is a special kind of Java program that is designed to be transmit over the internet and automatically executed by a Java compatible browser. Note that every time you downloaded a normal program, you are at risk of getting virus infection. If the applet program are there, it is giving security to the program. Hence it is safe and secure.

13. List the features of Java.

Ans. The features of Java language are

(i) Simple and Powerful: The simplicity of Java is enhanced by its similarities of C and C++ because Java inherits most of the syntax of C and C++. Hence it becomes simple objected oriented language.

(ii) Object oriented language: It is a true object oriented language. Without the concept of class and object it is not possible to write even small simple programs.

(iii) Safe and secure: An applet is a special kind of Java program that is designed to be transmit over the internet and automatically executed by a Java compatible browser. Note that every time you download a normal program, you are at risk of getting virus infection. If applet program are there, it is giving security to the program. Hence it is safe and secure.

(iv) Robust: Java is a robust language because of 2 reasons like program failure and memory management. In this programs are divided into small objects so that memory problem is automatically eliminated and program failure also minimized.

(v) Platform independent and portable: The most significant contribution of Java over other languages is portability. i. e., Java program can be easily moved from one computer to another, anywhere and anytime. Changes and upgrades in operating system, processor and system resource will not force any changes in Java program.

(vi) Multi threaded: Multi threaded means multiple task executed simultaneously. Java was designed to meet the4 real world requirement to creating interactive network program. To accomplish this, Java supports multi threaded programming which allows you to do many things simultaneously.

(vii) Compiled and interpreted: Java is the only language using compiled and interpreted operations. In this first Java compiler translates Java source code into byte code. Byte code is not executable code. It is highly optimized intermediate output. Using this byte code, interpreter converts into machine code at the end.

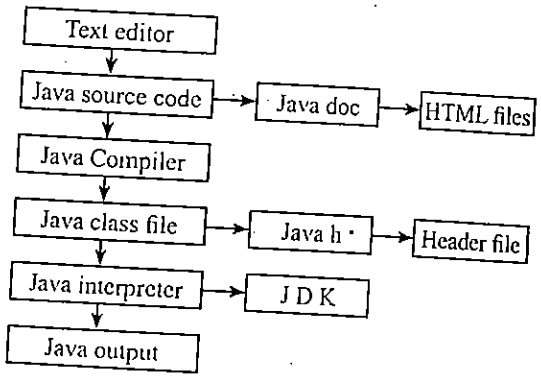
(viii) Distributed: Unlike the language like C and C++, Java is specially designed to work in networked environment Java has a large library classes for communicating the internets TCP/IP, Protocol, including http and FTP. Java code can manipulate resources via URL's.

(ix) Dynamic: Java is capable of dynamic linking with the different classes, methods and objects.

14. Explain Java runtime environment.

Ans. Java environment includes a large number of development tools, classes and methods. The development tools are the part of the system known as Java development kit (JDK) and hence class and methods are parts of the Java standard library (JSL) also known as application programming interface (API).

JDC comes with developing tools that are used for developing and running Java program and it is shown below.



(i) Applet viewer: It is used to run Java applets. The applet viewer creates a small window in which the applet can be viewed. It provides complete support for all applet function including multimedia capability.

(ii) Java: It is a Java interpreter which converts class files into Java output.

(iii) Java c: It is a Java compiler which translates Java source code into byte code and stored in class file.

(iv) Java doc: Basically Java doc reads through its file and creates html file that document your packages and classes including methods and data fields.

(v) Java h: It is the header file generation tool. Note that there is no header file in Java but converted as packages. To do certain task we need to import some packages.

(vi) JDB: It is a Java debugger tool used to monitor the execution of Java program in support of debugging and test activities.

15. Write a simple Java program and explain.

Ans.

```
Class sampleOne
{
{ public static void main (string args [ ])
  {
  System. out. println (' Java is better than c++');
   }
}
```

Class declaration:

The first line

Class sampleOne

declares a class, which is an object oriented construct. Class is a keyword and declares that a new class definition. **SampleOne** is a Java identifier that specifies the name of the class to be defined.

Opening brace: Every class definition in Java begins with an opening brace "{" and ends with a matching closing brace "}".

The main line:

The third line.

Public static void main (string args [ ]) defines a method named main. This is the starting point for the interpreter to begin the execution of the program. A Java application can have any number of classes but only one of them must include a main method to initiate

the execution.

The line contains a number of keywords - public, static and void.

Public - It is an access specifier that declares the main method as unprotected and therefore makes it accessible to all other classes.

Static - It declares this method as one that belongs to the entire class and not a part of any objects of the class. The main must always be declared as static since the interpreter uses this method before any objects are created.

Void - The type modifier states that the main method does not return any value.

The output line:

The only executable statement in the above program is,

System.out.println (" Java is better than c++"); This is similar to printf( ) statement of 'c' or c out << construct of 'c++'. The **println** method is a member of the **out** object which is static data member of **system** class. This line prints the string.

Java is better than c++.

The method println always appends a newline character to the end of the string. Every Java statement must end with a semicolon.

16. Write a simple Java program to illustrate the use of mathematical functions.

Ans.

```
/* Program to compute square root */
import Java. lang. Math;
class SquareRoot
{
  public static void main (string args [ ] )
{
double x = 5;
double y ;
y = Math. sqrt (x) ;
system. out. println ("y=" + y );
}
}
```


17. List the different sections of Java program structures.

Ans. The Java program may contain many classes of which only one class contain a main ( ) method definition. Class contains data members as well as method members that operates on data member. To write Java program we first define classes and then put them together. For this Java program may contain one or more sections as shown below.

Document Section	→	Suggested (Recommended)
Package Statement	→	Optional
Import Statement	→	Optional
Interface Statement	→	Optional
Class definition	→	Optional
Class containing a main () method	→	Essential

Document Section: The document section contains a set of comment lines giving the name of the program and other details like programmer name, machine used, environment used, OS used etc.

The documentation comment is of the form as shown below.

```
/* This is a
* comment line extended
* to as many lines
* as possible
 */
```

Package declaration: The package declaration must be the first statement in the file as executable statement. The package keyword is followed by the package name.

Ex: Package mypack;

Import Statements: This is used to access the classes and interfaces defined in the packages of Java standard library (JSL.

Ex: import Java. math;

Interface statements: An interface is like class but includes just method declaration. This is an optional section and used only when we wish to implement the multiple inheritance concept in Java.

Class definitions: Java program may contain multiple class definition. Classes are primary and essential requirements. The number of classes required may depend on complexity of the program.

Class containing main ( ) method: Every Java stand alone program required a main ( ) method as its starting point. Java program may contain only this part. The main ( ) method creates objects of various classes and establishes communication between them.

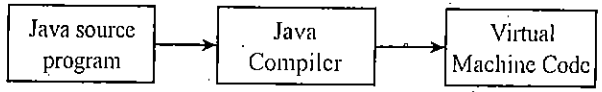
18. Write a Java program to illustrate an application with two classes.

Ans.

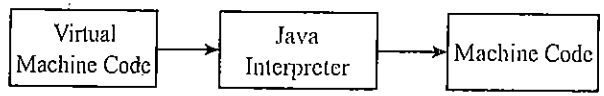
```
Class Room
{
float length;
float breadth;
void getdata (float a, float b)
{
length = a;
breadth = b;
}
}
class Roomarea
{
public static void main (string args [ ])
{
float area;
Room room1 = new Room ( );
room1. getdata (14, 10);
area = room1. length * room1. breadth ;
system. out. println ("Area = " + area);
}
}
```


19. Explain Java Virtual Machine.

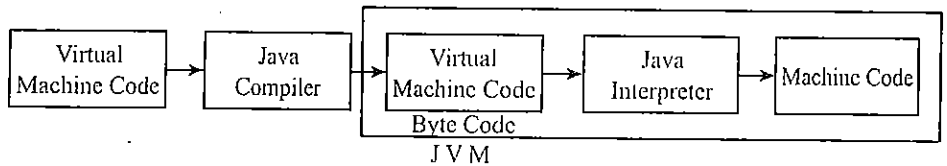
Ans. Java complier first translates the source code into an intermediate code known as byte code for machine that does not exist. This machine is called Java virtual machine that exists inside a computer memory. The following process is Java virtual machine code.



The byte code or virtual machine code is then interpreted by interpreter to generate machine language instructions that can be executed by any micro processor.



In general, Java is the two stage process like compile and interpreted, for that entire process is called Java virtual machine as shown below.



20. Discuss command line arguments in Java with an example program.

Ans. Command line arguments are parameters that are supplied to the application program at the time of invoking it for execution.

We can write Java programs that can receive and use the arguments provided in the command line.

The signature of the main ( ) method used in Java as

public static void main (string args [ ])

Here, args is declared as an array of strings, Any arguments provided in the command line are passed to the array args as its elements. Every individual arguments is indicated by index called 0, 1, 2.....

Ex: Consider the command line.

Java Test BASIC FORTRAN C++ Java.

The above command line contains 4 arguments. These are assigned to the array args as follows.

BASIC → args [0]

FORTRAN → args [1]

C++ → args [2]

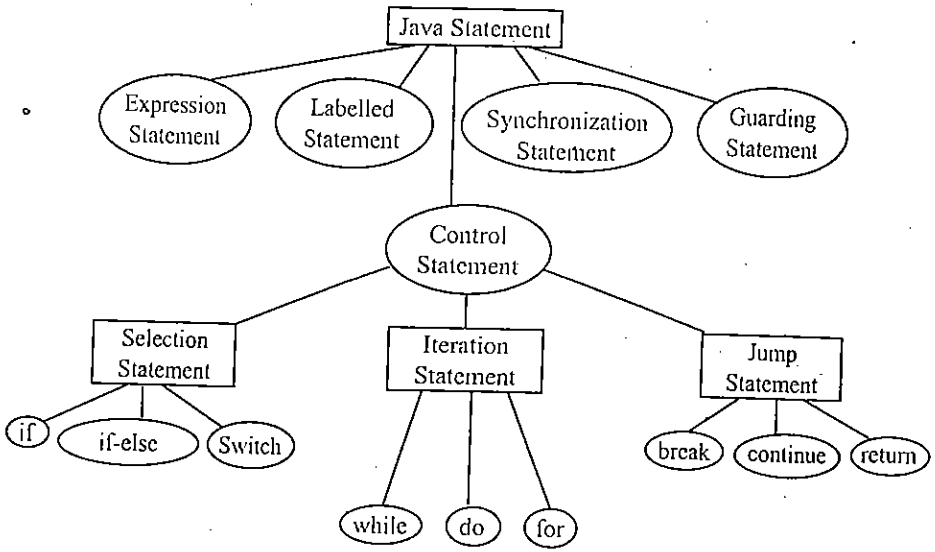
Java → args [3].

The number of arguments is obtained by the statement within the Java program is,

Count = args. length

21. List and explain Java statements.

Ans. A statement is an executable combination of tokens ending with a semicolon. Statements are usually executed in sequence in the order in which they appear. The different types of Java statements are



1. Expression statement: It changes the values of variables, call methods and create objects. Java has seven types of expression statements.

(a) Assignment.

(b) Pre increment.

(c) Pre decrement

(d) Post increment

(e) Post decrement.

(f) Method call.

(g) Allocation expression.

2. Labelled Statement: Any statement begin with a label . Labels must not be keywords. Already declared local labels or previously used labels are used in this module. In Java, these are used as arguments of jump statements.

3. Control statement: It is of 3 types.

(a) Selection statement: These select one of the several control flows. There are 3 types of selection statements in Java.

(i) if (ii) if-else (iii) switch

(b) Iteration statement: These specify how and when looping will takes place. There are 3 types of iteration statements.

(i) While (ii) do (iii) for

(c) Jump statements: These pass control to the beginning or end of the current block, or to a labelled statement. Such labels must be in the same block and **continue** labels must be on an iteration statement. The types of jump statements are

(i) break (ii) continue (iii) return

4. Synchronization statement: These are used for handling issues with multi threading.

5. Guarding statement: These are used for safe handling of code that may cause exceptions. These statements use the keywords try, catch and finally.

22. Describe in detail the steps involved in implementing a stand alone program.

Ans. Implementation of a Java application program involves the following steps.

* Creating the program.
* Compiling the program.
* Running the program.

Creating the program: We can create a program using any text editor.

Ex: Class Test

```
{
public static void main (string args [ ])
{
System. out println ("Welcome to the world of Java");
}
}
```

We must save this program in a file called test. Java ensuring that the file name contains the class name properly. This file is called the source file. All the Java source files will have the extension Java.

Compiling the program: To compile the program, we must run the Java complier Javac with the name of the source files on the command line as shown below.

Javac Test. Java

If everything is correct, the javac complier creates a file called Test. class containing the byte codes of the program. The compiler automatically names. The byte code file as.

< Class names.class>

Running the program: Java interpreter is used to run a stand alone program. At the command prompt, type

Java Test.

Now, the interpreter looks for the main method in the program and begins execution from there. When executed, the program displays.

Welcome to the world of Java.

23. Explain the contribution of Java to the world wide web with a figure. Illustrate how Java communicates with a web page.

Ans. WWW is an open ended information retrieval system designed to be used in the .nternet's distributed environment. For that Java makes use of hyper text mar up language (HTML) and web pages contain HTML tags that enables us to find, retrieve, manipulate and display documents world wide. Before Java, WWW was limited to the display of still

images and texts. The combination of Java and web pages has made it capable of support animation, graphics and games. With the support of Java, the support of web, we can run a Java program on other's computer accross the Internet.

Java communicate with a web page through a special tag called <APPLE T>. The figure shows the following communication steps.

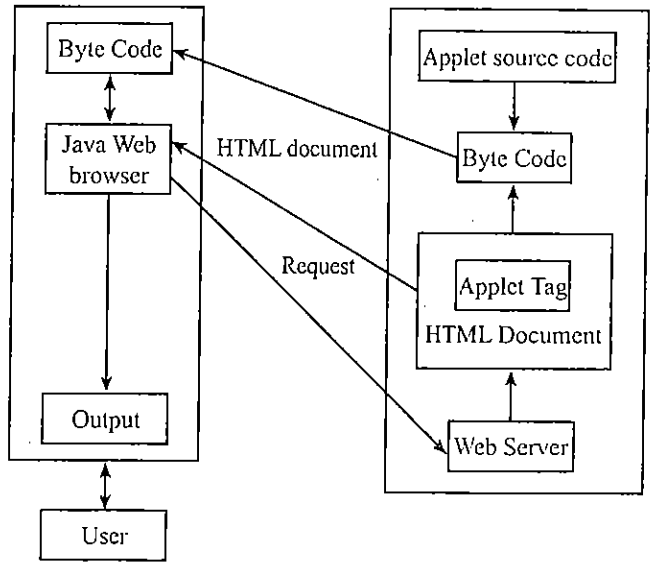
(i) The user sends a request for an HTML document to the remote computer's web server. Web server is a program that accepts a request, process the request and sends the required document.

(ii) The HTML document is returned to the user's browser. The document contains the APPLE T tag, which defines the applet.

(iii) The corresponding applet byte code is transferred to the user's computer.

(iv) The Java enabled browser on the user's computer interprets the byte codes and provides output.

(v) The user may have further interaction with the applet but with no further downloading from the provider's web server. This is because the byte code contains all the information necessary to interpret the applet.



24. With general syntax explain formatted input-output statements. Give example for each.

Ans. Formatted input Statement:

Read statement: The values are given to the variables interactively through the keyboard using read-line ( ) method.

Ex:import Java. io. DataInputstream;

```
Class Reading
{
Public static void main (string args [ ])
{
DataInputstream in = new DataInputstream (system. in);
int intNumber = 0;
float floatNumber = 0.of;
try
{
System.out.println ("Enter an integer:");
int Number = Integer. parse Int (in.read_line ( ));
System. out. println ("Enter a float number:");
float Number = Float. valueof (in. read line ( ). float value ();
}
Catch (Exception e) { }
System. out. println ("int Number = " + int Number);
System. out. println (" float Number = " float Number);
}
}
```

The read line ( ) method reads input from the keyboard as a string which is then converted to the corresponding data type using data type wrapper classes.

The keywords try and catch are used to handle errors that occurs during reading process.

Formatted output statement: Java supports two output methods that can be used to send the results to the source.

i. Print ( ) method: It sends the information into a buffer is not flushed until a newline character is sent. As a result, the print ( ) method prints output on one line until a newline character is encountered.

ii. Println ( ) method: It takes the information provided and displays it on a line followed by a line feed (carriage return)

Ex:

```
Class displaying
{
Public static void main (string args [ ])
{
System. out. println ("screen Display");
for (int i = 1; i< = 5; i++)
{
for (int j = 1; j< = i; j++)
{
System. out. print (" ");
System. out. print (i);
}
System. out. println ("\n");
}
System. out. println ("Screen Display Done");
}
}
```

Output:

Screen Display

1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

Screen Display Done.

25. Explain Dynamic binding and message passing.

Ans. Dynamic Binding: Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at runtime - It is associated with polymorphism and inheritance.

Message passing: The concept of message passing makes it easier to build systems that directly model or simulate this real world counterparts.

Message passing involves specifying the name of the object, name of the method ant the information to be sent.

Ex: Employee. Salary (name);

Here, Employee is the object, salary is the message and name is the parameter that contains information.

UNIT 2

CLASSES, OBJECTS AND METHODS; STRINGS AND STRING BUFFER CLASSES

SYLLABUS

Introduction, Defining a class, Fields Declaration Methods declaration, Creating objects, Accessing class members, Constructors, Methods overloading, static members, Nesting of methods, Inheritance: Extending a class, overriding methods, Final variables and methods, Final classes, Finalize methods, Abstract methods and classes, Methods with variable arguments, Visibility control. Strings and string Buffer classes - Strings, wrapper class, Enumerated types, Annotations.

SYNOPSIS

Java is a true object oriented language and therefore the structure of all Java programs is classes. Classes create objects and objects use methods to communicate between them. In Java, the data items are called **fields** and functions are called **methods**. Calling a specific method in an object is described as sending the object a message.

1. Define constructors. list its special properties.

Ans. Constructors are the special type of member function that enables an object to initialize itself when it is created. It is very simple and more concise to initialize an object.

The properties of constructors are

1. Constructors have the same name as that of the class.
2. Constructors are executed when an object is declared.
3. Constructors have neither return value nor void.
4. The main function of the constructor is to initialize objects and allocate memory to objects.
5. It supports constructor overloading.
6. It can support for default arguments.

2. Define object. Explain object creation from a class.

Ans. Object: Objects are the basic runtime entities in an object oriented system.

Creating objects: An object in Java is essentially a block of memory that contains space to store all the instance variables. Creating an object is also referred as instantiating an object.

Objects in Java are created using the new operator. The new operator creates an object of the specified class and returns a reference to that object.

Ex: Rectangle rect 1; // declare the object

rect 1 = new Rectangle ( ); // instantiate the object.

The **first** statement declares a variable to hold the object reference. The **second** statement assigns the object reference to the variable. The variable rect 1 is now an object of the **Rectangle** class.

Action	Statement	Result
Declare Instantiate	Rectangle rect 1; rect 1 = new Rectangle ( );	null rect 1 rect 1 Rectangle Object

Both statements can be combined into one as shown below.

Rectangle rect 1 = new Rectangle ( ) ;

Where rect 1 is called an object an object of that class. Similarly we can also create multiple objects for same class.

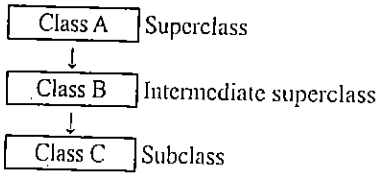
Ex: Rectangle rect 1 = new Rectangle ( );

Rectangle rect 2 = new Rectangle ( );

3. Define multilevel inheritance with an example.

Ans. It is a process of deriving a sub class from previously derived class is known as multilevel inheritance.

Ex:



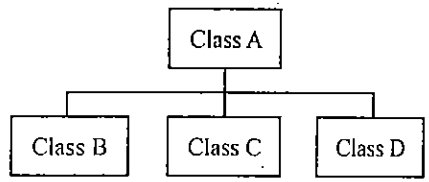
Class A serves as base class for the derived class B which is inturn serves as bass class for the derived class C.

A derived class with multilevel base classes is declared as follows.

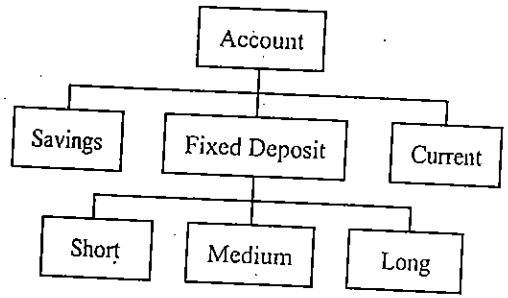
```
Class A
{
--------
--------
}
Class B extends A // Firstlevel.
{
--------
--------
}
Class C extends B // Secondlevel.
{
--------
--------
}
```

4. Define hierarchical inheritance with an example.

Ans. The process of deriving one or more subclass from a single base class is called hierarchical inheritance.

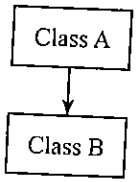


Ex:



5. Describe the syntax of single inheritance in Java.

Ans. The process of deriving a subclass from only one super class is called single inheritance.




```
Ex:   Class faculty
      {
         float salary = 30000;
      }
         Class science extends faculty
      {
         float bonus = 2000;
      Public static void main (string args [ ])
      {
         Science obj = new science ( );
         System. out. println ("Salary is:" + obj. salary );
         System. out. println ("Bonus is:" + obj. bonus);
         }
      }
```


Output:

Salary is : 30000.0

Bonus is : 2000.0

6. Define subclass constructor and the use of keyword super.

Ans. A subclass constructor is used to construct the instance variables of both the subclass and the superclass. The subclass constructor uses the keyword **super** to invoke the constructor method of the superclass. The keyword super is used subject to the following conditions.

(i) Super may only be used within a subclass constructor method.

(ii) The call to the superclass constructor must appear as the first statement within the subclass constructor.

(iii) The parameters in the super call must match the order and type of the instance variable declared in the superclass.

Super key can be used in the following ways.

(i) Base class fields.

(ii) Base class methods.

(iii) Base class Constructor.

```
Ex:   Class A
      {
         int    a;
      A ( )
      {
         a = 100;
      }
      Void show ( )
      {
        System. out. println ("a=" + a);
        }
      }
      Class B
      {
```

```
        int b;
        B ( )
        {
      b = 200;
      super. show ( );
      system. out. println ("b =" + b);
       }
      }
       class method test
       {
       Public static void main (string args [ ])
       {
        B   b = new  B ( );
    }
}
```

7. Define final classes and finalize methods.

Ans. Final classes: A class that cannot be sub classed is called a final class. This is achieved in Java using the keyword **final.**

```
final class Aclass
{
   -------
   -------
}
final class Bclass extends someclass
{
   -------
   -------
}
```

Any attempt to inherit these classes will cause an error and the compiler will not allow it. Declaring a class **final** prevents any unwanted extensions to the class.

Finalize Methods: Constructor method is used to initialize an object when it is declared. This process is known as initialization. Similarly Java supports a concept called finalization which is just opposite to initialization. It is similar to destructor in C++. The finalize method is simply **finalize** ( ) and it can be added to any class. Java this method automatically.

Whenever it counts to reclaim the space for that object. The finalize method should explicitly define the tasks to be performed. The syntax is

```
Void finalize ()
{
-------
-------
}
```

8. Define abstract methods and classes.

Ans. Making a method final to ensure that the method is not redefined in a subclass i.e., the method can never be subclassed. Java indicate that a method must always be redefined in a subclass, thus making overriding compulsory. This is done by using the modifier keyword **abstract** in the method definition.

Ex:

```
abstract class shape
{
----
----
abstract void draw( );
-----
-----
}
```

When a class contains one or more abstract methods, it should also be declared abstract as shown above. While using abstract classes, we must satisfy the following conditions.

(i) We cannot use abstract classes to instantiate objects directly. For example,

Shape S = new shape ( );

is illegal because shape is an abstract class.

(ii) The abstract methods of an abstract class must be defined in its subclass.

(iii) We cannot declare abstract constructors or abstract static methods.

9. Write the visibility control of Java classes.

Ans. Sometimes it may be necessary to restrict access to certain variables and methods from outside the class. To do so, we use access specifies or visibility specifiers or visibility modifiers. Java provides three types of visibility modifiers: Public, private and protected.

Public Access: The variables or methods defined as public makes it visible to all other classes outside the class.

Ex: Public int number;

Public void sum ( )

{

}

A variable or method declared as public has the widest possible visibility and accessible everywhere.

Friendly Access or default: Friendly access makes the fields visible only in the same package but not in other packages. We have not used public modifier, yet they were still accessible in other classes in the program.

Protected Access: The **protected** modifier makes the fields visible not only to all classes and subclass in the same package but also to subclasses in other packages.

Private Access: They are accessible only with their own class. They cannot be inherited by subclasses and therefore not accessible in subclasses. A method declared as **private** behaves like a method declared as **final**.

Access Location	Access modifier	public	protected	friendly (default)	private protected	private
Same class		Yes	Yes	Yes	Yes	Yes
Subclass in same package		Yes	Yes	Yes	Yes	No
Other classes is same package		Yes	Yes	Yes	No	No
Subclass in other packages		Yes	No		Yes	No
Nor- subclasses in other packages		Yes	No	No	No	No

Private Protected Access: A field can be declared with two keywords Private and Protected together like.

Private Protected int Codenumber;

This gives visibility level in between protected access and private access. It makes the fields visible in all subclasses regardless of what package are in. These fields are not accessible by other classes in same package.

10. Compare Arrays and Vectors.

Arrays	Vectors
(i) It is homogeneous.	(i) It is heterogeneous.
(ii) Array can store only primitive data types.	(ii) Vectors can store primitive data types as well as objects.
(iii) Array is not synchronized.	(iii) Vector is synchronized.
(iv) The size of the array is declared in advance.	(iv) We do not declare the size of the vector.
(v) Once declared cannot grow in size.	(v) Vectors can always grow in size by adding more number of elements than the declared size.

11. Explain any five string methods.

Ans. The string class provides many useful method to perform operation on sequence of character value. Some commonly used methods are.

Sl. No	Method Name	Description.
1.	S2 = S1. to Lower Case;	Converts the string S1 to all lower case letters.
2.	S2 = S1 to Upper Case;	Converts the string S1 to all upper case.
3.	S2 = S1. trim ( );	Remove white spaces at the beginning and end of the string S1.
4.	S1. equals (S2)	Returns 'true' if S1 is equal to S2.
5.	S1. length ( )	Gives the length of string S1.
6.	S1. Char At (n)	Gives the nth character in S1.
7.	S1. Compare To (S2)	Returns positive if S1>S2.
8.	S1. Concat (S2)	Returns negative if S1<S2
9.	S2 = S1. replace (X;, 'Y');	Returns zero if S1 = S2. Concatenates S1 and S2
10.	S1. index of ('X')	Replaces all appearance of X with Y Gives the position of first occurrence of 'X' in the string S1

12. Explain any five string buffer methods.

Sl No.	String method name	Description.
1.	S1. append (S2)	Appends the string S2 to S1 at the end.
2.	S1. reverse ( )	Reverses the current string.
3.	S1. SetcharAt (n.'X')	Modifies the nth character to X.
4.	S1. Setlength (n)	Sets the length of the string S1 to 'n'. If n<S1. length ( ), S1 is truncated. If n> S1. length ( ),zeros are added to S1.
5.	S1. insert (n, S2)	Inserts the string S2 at the position n of the string S1.

13. Differentiate between interfaces and classes.

Ans.

Interfaces	Classes
1. The members of an interface are always declared as constant i.e., their values are final.	1. The members of the class can be constant or variables.
2. It supports multiple inheritance.	2. It does not supports multiple inheritance.
3. It can use only the public access specifier.	3. It can use access specifiers like public, protected or private.
4. It does not supports constructor declaration.	4. It supports constructor declaration.
5. It can not be instantiated.	5. It can be instantiated.
6. Anything declared in the interface is used by class.	6. In this fields and methods are not used by interface itself.

14. Explain the use of wrapper class in Java.

Ans. Primitive data types can be converted into object types by using the wrapper classes contained in the Java.lang package. Each primitive data types has its own wrapper class in Java. The below table shows the list of wrapper class for each primitive data type

Sl No.	Primitive type	Wrapper class
1.	boolean	Boolean
2.	char	Character
3.	double	Double
4.	float	Float
5.	int	Integer
6.	long	Long.

The wrapper classes have number of methods for handling primitive data type to objects. They are listed below.

Sl. No	Constructor Calling	Conversion Action
1.	Integer IntVal = new Integer (i);	Primitive integer to integer object.
2.	Float FloatVal = new Float (f);	Primitive float to float object.
3	Double DoubleVal = new Double (d);	Primitive double to double object.
4.	Long LongVal = new Long (l);	Primitive long to long object.
5.	Character CharVal = new Character (i);	Primitive char to char object
6.	boolean Val = new Boolean (i);	Primitive boolean to boolean object.

The below table shows converting object numbers to primitive numbers using type value ( ) method.

Sl No.	Method Calling	Conversion Action
1.	int i = IntVal. in value ( );	Object to primitive integer
2.	float f = FloatVal. float Value ( );	Object to primitive float
3.	long l = LongVal. long Value ( );	Object to primitive long.
4.	double = DoubleVal. double Value ( );	Object to primitive double.

15. Explain class definition with fields and methods declaration.

Ans. Defining a class: A class is a user defined data type with a template that serves to define its properties. Once the class type has been defined, we can create variables within the class. The general syntax for class declaration is

```
Class classname [extends superclassname]
{
    [ fields declaration;]
    [ methods declaration;]
}
```

Everything inside the square bracket is optional. This means that the following would be a valid class definition when you want to use base class or superclass.

```
Class classname.
{
    [fields declaration;]
    [methods declaration;]
}
```

There is no semicolon (;) after flower brackets.

Class name uses any valid Java identifier. Usually class contains two important definition.

(i) Field definition.

(ii) Method definition.

Field Definition: Data is encapsulated in a class by placing data inside the body of the class definition. These variables are called instance variables because they are created whenever an object of the class is instantiated. We can also call these variables as local variable.

Ex:
```
Class Area
{
    int length, breadth;   // instance variables.
}
```

These variables are only declared and therefore no storage has been created in memory. Therefore instance variables are also known as **member variables**.

Method Declaration: A class with only data fields has no life. The objects created by such a class can not respond to any messages. Therefore add methods that are necessary for manipulating the data contained in the class. Methods are declared inside the body of the class but immediately after the declaration of instance variables.

The general form of method declaration is

```
type methodname (parameter - list)
{
    method - body;
}
```

Method declaration has 4 basic parts.

(i) The name of the method (method name)

(ii) The type of the value the method returns (type)

(iii) A list of parameters (parameter - list)

(iv) The body of the method.

Where,

The **method name** is a valid identifier.

The **type** specifies the type of value the method would return. It can be simple data type such as int as well as any class type. It can be void type, if the method does not return any value.

The **parameter list** is always enclosed in parenthesis. It contains variable names and types of all values.

The **body** actually describes the operations to be performed on the data.

Ex:
```
Class Area
{
   int length ;
   int width ;
  void getdata (int x, int y)     // Method declaration
  {
   length = x ;
   width = y ;
}
}
```

16. With an example, Explain accessing of class members.

Ans. Once the objects are created, each containing its own set of variables, we should assign values to those variables in order to use them in our program. All the variables must be assigned before they are used. Since we are outside the class, we can not access the instance variables and methods directly. To do this, we must use the object with. (dot) operator where dot operator is called direct member access operator. The general syntax is,

```
Object name. Variable name = value ;
Object name. method name (parameter list);
```

Where Object name is the name of the object.

Variable name is the name of the instance variable inside the object.

Method name is the name of the method.

Parameter list is the list of actual values separated by comma.

The instance variables of Rectangle class may be accessed and assigned values as shown below.

```
rect 1. length = 15;
rect 1. width = 10;
rect 2. length = 20;
rect 2. width = 12;
```

The two objects rect 1 and rect 2 store different values as shown below.

	rect1
rect1.length	15
rect1.width	10

	rect1
rect2.length	20
rect2.width	12

This is one way of assigning values to the variables in the objects. Another way to assign values to the instance variables with the help of object name, method name and dot operator.

Rectangle rect 1 = new Rectangle ( );

rect 1.getdata (15, 10);

This code creates rect 1 as object and then passes the values 15 and 10 for the methods defined in that class.

Ex:

```
Class Rectangle
{
    int length, width ;
    void getdata (int x, int y)
    {
        length = x;
        width = y ;
    }
  int rect area ( )
  {
     int area = length x width;
     return (area);
  }
}
class Rect area
{
    public static void main (string args [ ])
{
    int area 1, area 2;
```



```
    Rectangle rect 1 = new rectangle ( ) ;
    Rectangle rect 2 = new Rectangle ( ) ;
    rect 1. length = 15 ;
    rect 1. width = 10 ;
    area 1 = rect 1. length * rect 1. width ;
    rect 2. getdata (20, 12);
    area 2 = rect 2. rect Area ( );
    System. out. println ("Area 1 = " + area 1);
    System. out. println (" Area 2 = " + area 2);
  }
}
```

Output

Area 1 = 150

Area 2 = 240

17. Write a program to illustrate constructor

Ans.

```
Class Rectangle
{
   int length, breadth;
  Rectangle ( )
  {
    length = 0 ;
    breadth = 0 ‘
  }
 Rectangle (int l, int b)
 {
  length = l;
  breadth = b;
  }
  void calculate_area ( )
```

```
{
    int area ;
    area = length * breadth;
System. out println ("Area of rectangle = " + area)
}
}
    Class Area
{
  Public static void main (string args [ ])
{
    Rectangle r1 = new Rectangle ( );
    Rectangle r2 = new Rectangle (200, 100);
        r1. Calculate_area ( );
        r2. Calculate_area ( );
    }
    }
```

Output:

Area of rectangle = 0

Area of rectangle = 20000

18. Discus the process of method overloading.

Ans. The methods that have the same name, different parameter list are called method overloading. It is used when the objects are required to perform similar task but using different input parameters.

To create an overload method, we have to provide several different method definitions in the class, all with same name but with different parameter lists, All these methods varies only by number or type of parameter. Each parameter list should be unique.

Ex:

```
Class Sum
{
float first, second;
void getdata (float a, float b)
{
```



```
        first = a ;
        second = b;
    system. out. println (a + b);
    }
    void getdata ( )
    {
    first = second = 10 ;
    system. out. println (a + b);
}
void getdata (int a, int b)
{
    first = a ;
    second = b ;
system. out. println (a + b) ;
}
}
Class Area
{
    public static void main (string args [ ] )
    {
        sum S1 = new sum ( );
        sum S2 = new sum ( );
        sum S3 = new Sum ( );
        S1. getdata ( ) ;
        S2. getdata (10, 20) ;
        S3. getdata (6. 5, 7. 5) ;
    }
    }
```

19. Write a program to illustrate method overloading.

Ans. Refer Q18. (Program).

20. Write a program to sort 'n' elements of an array.

Ans.

```
Class sorting
{
    Public static void main (string args [ ])
{
    int number [ ] = {55, 40, 80, 65, 71} ;
    int n = number. length ;
    system. out. println ("Given list : ");
    for (int i = 0; i<n; i++)
{
    System. out. println (" " + number [i] ) ;
}
    system. out. println ("\n");
    for (int i=0; i<n; i++)
    {
        for (int j = i+1; j < n; j++)
    {
        if (number [i] < number [j])
        {
            int temp = number [i] ;
            number [i] = number [j];
            number [j] = temp ;
        }
    }
}
System. out. println (" sorted list:");
for (int i = 0; i<n; i++)
{
    system. out. println (" " + number [i] );
}
    system. out println (" ");
    }
}
```

Output:

Given list: 55 40 80 65 71

Sorted list : 80 71 65 55 40.

21. Write a program to illustrate vectors.

Ans.

```
    import Java util. *;
    class langvector
{
    public static void main (string args [ ] )
    {
    vector list = new vector ( );
    int length = args. length ;
    for (int i = 0; i< length ; i++)
{
    list. addElement (args [i]);
}
    list. insert Element At ("C0B0L", 2);
    int size = list. size ( );
    String list Array [ ] = new string [size];
    list. copy Into (List Array);
    System. out. println ("List of languages");
    for (int i=0; i<size; i++)
    {
        system. out. println (List Array [i]);
    }
}
```

```
}
```

Command line input and output are.

C:\Java\Prog>Java LangVector Ada BASIC C++ FORTRAN JAVA.

List of languages.

Ada

BASIC

COBOL

C++

FORTRAN

Java.

22. Write a program to illustrate wrapper classes.

Ans.

```
import Java. io. *;
class Test
{
Public static void main (string args [ ])
{
system. out. println ("\n creating objects");
Integer i = new integer (25);
float f = new float (15, 25 f);
character c = new character ('n') ;
Double d = new Double (35, 25d);
System. out. println ("object i = " + i + "\n    f = " +f + "\n
                        c = " + c+"\n     d= "+d"\n");
int a = i. int value ( );
float b = l.float value ( );
char c = c. char value ( );
double d = d. double value ( );
System. out. println ("\t a = " + a+"\t b = "+b+"\t c= " + c+ " \t  d = " + d + "\n");
system. out println (" Converting number to string \n");
```



```
String S1 = Integer. to string (a);
String S2 = float. to string (b);
String S3 = character. to string (c);
String S4 = double to string (d);
System. out. println ("S1 = " +S1 +"\t   S2 = " + S2 + " \t S3 = " + S + " \t  S4 = " +
S4 +"\n");
System. out. println ("\n converting strings to numberic" );
}
}
```

23. Compare overloading and overriding methods.

Method Overloading	Method Overriding
(i) This method have same name and different parameter list.	(i) This method have same name and same parameter list.
(ii) Overloading happens at compile time.	(ii) Overriding happens at runtime.
(iii) Static methods can be overloaded.	(iii) Static methods cannot be overridden.
(iv) Static binding is being used for overloaded method.	(iv) Dynamic binding is being used for overriding method.
(v) Overloading is being done in the same class.	(v) Overriding base and child classes are required.

24. Define static member write a program to illustrate static members.

Ans. A class basically contains two sections, one declares variables as fields and other declares methods. These variables and methods together called instance variable and instance method. This is because every time the class is instantiated a new copy of each of them is created. They are accessed using the objects only. We want to define a member that is common to all the objects and accessed without using particular objects i.e., the member belongs to the class as a whole rather than objects created from the class. Such members can be defined as

```
Static int count;
Static int max (int x, int y);
```

The members that are declared by using the keyword static are called static members.

Program:

```
Class static demo
{
```

```
        int a'
        static int b;
    }
    static demo ( )
    {
        a = 10;
        b = 20;
    }
    void display (void)
    {
    system. out println ("a = " + a+ "b = " + b);
    }
    static void show ( )
    {
        system. out. println (" b = " + b);
    }
    class execute
{
    public static void main (string args [ ])
    {
    static demo S = new static demo ( );
    S. display ( );
static demo.show ( );
}
}
```


UNIT 3

INTERFACE: MULTIPLE INHERITANCE

SYLLABUS

Introduction, Defining Interfaces, Extending Interfaces, Implementing Interfaces Accessing Interface Variables.

SYNOPSIS

Introduction:

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods. Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviours that a class implements.

An interface in Java is a blueprint of a class. It has variables and methods, variables as final static and methods as abstract only. An interface is a reference type, to a class, that can contain only constants and method signatures. There are no method bodies. This means that interfaces do not specify any code to implement these methods and data fields contain only constants. Therefore, it is the responsibility of the class that implements an interface to define the code for implementation of these methods.

An interface is similar to a class in the following ways.

1. An interface can contain any number of methods.

2. An interface is written in a file with a. **Java** extension, with the name of the interface matching the name of the file.

3. The byte code of an interface appears in a .**class** file.

4. Interfaces appear in packages, and their corresponding byte code file must be in a

directory structure that matches the package name.

However, an interface is different from a class in several ways, including

1. You cannot instantiate an interface.

2. An interface does not contain any constructors.

3. All of the methods in an interface are abstract.

4. An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.

5. An interface is not extended by a class; it is implemented by a class.

6. An interface can extend multiple interfaces.

The constant declaration syntax is

Static final type constant _ name=value;

Method declaration will contain only a list of methods without any body statement.

return _ type method _ name (parameter _ list);

interface interface name

{

constant declarations;

method signatures;

}

Defining interface:

The interface keyword is used to declare an interface. The general syntax for defining an interface:

Interfaces have the following properties

1. An interface is implicitly abstract. You do not need to use the abstract keyword while declaring an interface.

2. Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.

3. Methods in an interface are implicitly public.

Example:

interface Product

{

statistic final int num =1000;

static final String name="Java"

void show();

}

Note that the code for the method is not included in the interface and the method declaration simply ends with the semicolon. The class that implements the interface must define the code for the method.

There are **mainly three** reasons to use interface. They are

1. It is used to achieve fully abstraction.

2. By interface, we can support the functionality of multiple inheritances.

3. It can be used to achieve loose coupling.

What are the differences between abstract methods and interfaces?

Sl. No	Class	Interface
1.	It can have abstract and non-abstract methods	It can have only abstract methods
2.	It doesn't support multiple inheritances.	It supports multiple inheritances.
3.	It can have final, non-final, static and non-static variables.	It has only static and final variables.
4.	It can have static methods, main method and constructor	It can not have static methods, main method or constructor
5.	It can provide the implementation of interface.	It can not provide the implementation of abstract class
6.	The abstract keyword is used to declare abstract class	The interface keyword is used to declare interface.

Extending interfaces:

An interface can extend other interfaces, just a class can extend or subclass another class. However, whereas a class can extend only one other class, an interface can extend any number of interfaces. An interface can extend interfaces using the keyword extends as shown below:

```
interface interface1 extends interface2
{
constant declarations;
method signatures;
}
interface Product
{
stastic final int num =1000;
static final String name="Java"
}
interface extends  Product
{
void show();
{
}
}
```

The new sub interface interface1 will inherit members of the super interface interface2 in the manner similar to classes. We can put all constants in one interface and all methods in the other. This enables us to use the constants in classes where the methods are not required. This is how Java implements the concept of header files, as it does not support header file inclusion. Usually, we can put all the constants in one interface and the methods in the other. This will enable us to use the constants in classes where the methods are not required.

Extending Multiple Interfaces:

A Java class can only extend one parent class. Multiple inheritances are not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface. That means, interfaces are allowed to extend other interfaces, sub-interfaces cannot define the methods declared in the super interfaces. The following demonstrates the extending of the interfaces. Sub-interfaces are still interfaces, not classes. Instead, it is the responsibility of any class that implements the derived interface to define all the methods. Note that, when an interface extends two or more interfaces, they are separated by commas.

Note: It is important to remember that an interface cannot extend classes. This violates the rule that interface can have only abstract methods and constants.

```
interface Product
{ statics final int num =1000;
}
interface Method extends  Product
{ void show();
}
interface ProMet extends product, Method
{ ……..
……..
}
```

Example: Program to extend an interface

```
/* Write a program to demonstrate using extending an interface
*/
import Java. lang.*;
interface Constants
{
double pi=3.14159;
double radius=6.5;
```

```
}
interface Method extends Constants
{
void compute( );
}

class Circle implements Method
{
public void compute()
{
double area, peri;
area = pi*radius*radius;
peri =2*pi*radius;
System. out. println ("Area="+area);
System. out. println ("Perimeter="+peri);
}
}
class Extend_ Interface
{
 public static void main(String args[ ])
{
Circle c=new Circle ();
c. compute ();
}
}
```


Implementing interfaces:

An interface can be used as super class, where all the members can be inherited by classes. The following example illustrates how to implement interface.

```
class class name implements interface name
 {
  Body of a class;
 }
```

When a class implements more than one interface, they are separated by commas. The following example illustrates how to implement interface.

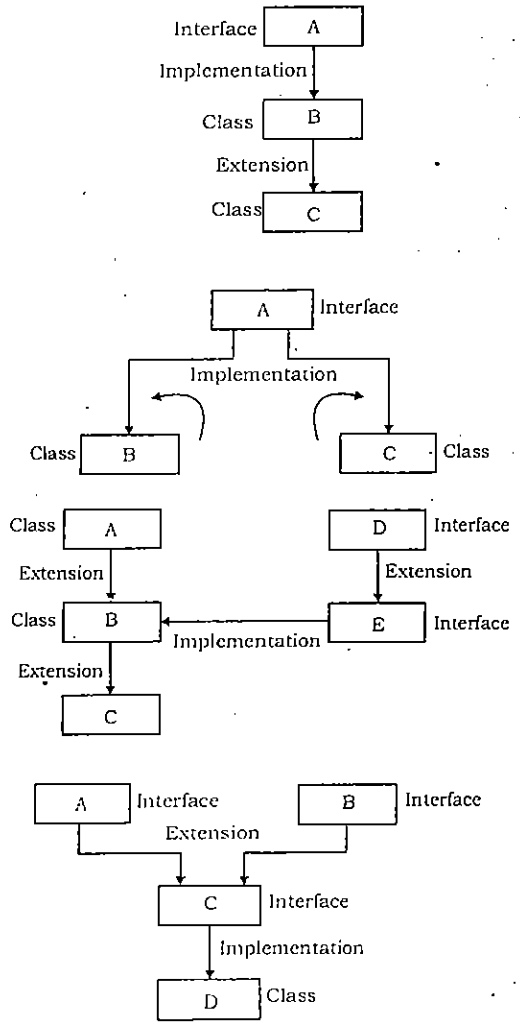
Example: Program to implement an interface.

Write a program to demonstrate using implementing interface

```
interface Area
{
    final static float PI=3.14F;
    float compute (float X, float Y);
}
 class Rectangle implements Area
 {
  public float compute (float X, float Y)
  {
   return ( X*Y);
  }
}
class Imple_Interface
{
public static void main (String args [ ])
{
Rectangle rect=new Rectangle ( );
```

```
System.out.println ("Area of Rectangle is: " +rect.compute (10.5f, 20.7f));
}
}
```

The various forms of interface implementation are as follows:



To implement the concept of multiple inheritance using interfaces? Explain with an example.

Java does not support multiple inheritance. It provides a method called interface to implement multiple interface. An interface is basically a kind of class. It contains methods and variables. The general form of interface definition is:

Programs to implement multiple inheritances

Program1:

```
interface A
{
   int x=100;
  void display _ A();
   }
interface B
{
   int y=200;
      void display _ B();
   }
   class C implements A,B
{
      public void display _ A()
{
   System. out. println ("x="+x);
}
     public void display _ B()
{
System. out. println ("y="+y);
}
     }
  class Impl _ inface
```

```
    {
     public static void main(String args[])
{
   C obj = new  C();
   obj. display _ A();
  obj. display _ B();
    }
    }
```

Program2:

Ans.
```
class Employee
{
Int empno;
String name;
String designation;
Void getemp (int eno, String n, string design)
{
empno = eno;
name=n;
designation=design;
}
Void putemp()
{
System out. println ("Employee Number="+empno);
System. out. println ("Employee Name="+name);
System. out. println ("designation="+designation);
}
}
class salary extends Employee
{
```



```
Int basic, da, hra, gross;
Void getsal (int b)
{
basic=b;
da = (int)(basic*1.07+0.5);
hra = (int)(basic*0.09+0.5);
gross=basic + da + hra;
}
Void putsal()
{
System. out. println("Salary is ="+gross);
}
}
Interface Overtime
{
Int hours=5;
Int rate=50;
Void computeot ( );
}
class Resultant extends Salary implements Overtime
{
Int total, ot;
public void computeot ()
{
ot=hours*rate;
}
void display()
{
computeot()
```

```
total=gross+ot;
putemp();
putsal();
System. out. println ("Overitme Salary="+ot);
System. out. println ("Total salary="+total);
}
}
Class Multipleinherit
{
public static void main(String a[])
{
Resultant r= new Resultant();
r.getemp (20,"Ramesh  BN","HOD");
r.getsal (11975);
r. display ();
}
}
```

Program3: Program to implement multiple inheritance.

```
class Student
 {
  int regno;
  void get Number( int n)
  {
   regno=n;
  }
  void put Number( )
  {
   System.out.println("Rollno: " +regno);
  }
```



```
  }

  class Test extends Student
  {
   int marks;
   void getMarks(int x)
   {
    marks=x;
   }
   void putMarks( )
   {
    System.out.println("marks= " +marks);
   }
  }


interface Sports
 {
  int sportwt=8;
  void putWt( );
 }
class Results extends Test implements Sports
 {
  int total;
  public void putWt( )
  {
   System.out.println("Sport weight= " +sportwt);   }
 }
class Multiple
```

```
{
public static void main ( String args [ ])
{
Results student=new Results( );
student.getNumber(1234);
student.getMarks(90);
student.putNumber( );
student.putMarks( );
student.putWt( );
}
}
```

5 MARK QUESTIONS AND ANSWERS

1. Differentiate between interfaces and classes.

Ans.

Sl. No	Class	Interface
1.	It can have abstract and non-abstract methods	It can have only abstract methods
2.	It doesn't support multiple inheritances.	It supports multiple inheritances.
3.	It can have final, non-final, static and non-static variables.	It has only static and final variables.
4.	It can have static methods, main method and constructor	It can not have static methods, main method or constructor
5.	It can provide the implementation of interface.	It can not provide the implementation of abstract class
6.	The abstract keyword is used to declare abstract class	The interface keyword is used to declare interface.

2. Write the general syntax of creating an interface and explain

Ans. The interface keyword is used to declare an interface. The general syntax for defining an interface:

```
interface interfacename
{
```



```
constant declarations;
method signatures;
}
```

Interfaces have the following properties

4. An interface is implicitly abstract. You do not need to use the abstract keyword while declaring an interface.

5. Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.

6. Methods in an interface are implicitly public.

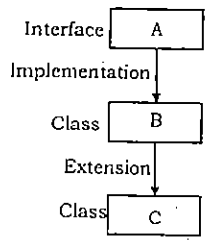
Example:

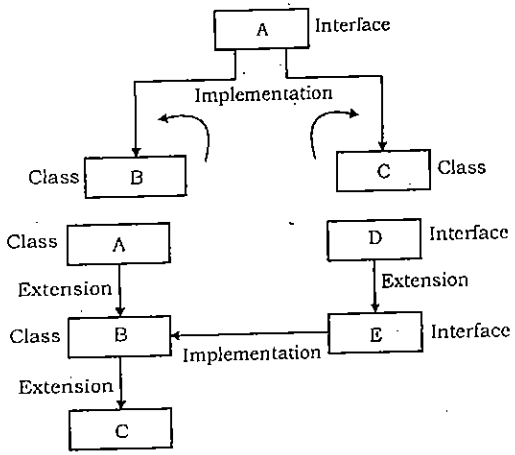
```
interface Product
{
stastic final int num=1000;
static final String name="Java"
void show();
}
```

Note that the code for the method is not included in the interface and the method declaration simply ends with the semicolon. The class that implements the interface must define the code for the method.

3. Explain the various forms of interface implementation

Ans. The various forms of interface implementation are as follows:





4. Explain how to access interface variables with an example.

Ans. Note that the code for the method is not included in the interface and the method declaration simply ends with the semicolon. The class that implements the interface must define the code for the method.

```
import java.lang.*;
interface Constants
{
double pi=3.14159;
double radius=6.5;
void compute( );
}
class Circle implements Constants
{
public void compute()
{
double area, peri;
area=pi*radius*radius;
peri=2*pi*radius;
```



```
System. out. println ("Area="+area);
System. out. println ("Perimeter="+peri);
}
}
class Extend _ Interface
{
public static void main(String args[])
{
Circle c=new Circle ();
c. compute ();
}
}
```

5. Write the similarities between class and interfaces.

Ans. The similarities between class and interface are

1. The syntax is similar to except the keyword class or interface.
2. Both contain the fields and methods.
3. Both can be extended

10 MARK QUESTIONS AND ANSWERS

1. Explain how to extend interfaces with an example.

Ans. An interface can extend other interfaces, just a class can extend or subclass another class. However, whereas a class can extend only one other class, an interface can extend any number of interfaces. An interface can extend interfaces using the keyword extends as shown below:

```
interface interface1 extends interface2
{
constant declarations;
method signatures;
}
```

The new sub interface interface1 will inherit members of the super interface interface2 in the manner similar to classes. We can put all constants in one interface and all methods in the other. This enables us to use the constants in classes where the methods are not required. This is how java implements the concept of header files, as it does not support header file inclusion.

Usually, we can put all the constants in one interface and the methods in the other. This will enable us to use the constants in classes where the methods are not required.

```
interface Product
{
stastic final int num=1000;
static final String name="Java"
}
interface extends  Product
 {
 void show();
 {
 }
 }
```

Example:

```
import java.lang.*;
interface Constants
 {
 double pi=3.14159;
 double radius=6.5;
 }
```



```
interface Method extends Constants
{
void compute( );
}
class Circle implements Method
{
public void compute()
{
double area, peri;
area=pi*radius*radius;
peri = 2*pi*radius;
System. out. println ("Area="+area);
System. out. println ("Perimeter="+peri);
}
}
class Extend _ Interface
{
 public static void main(String args[])
{
Circle c=new Circle ();
c. compute();
}
}
```

2. With an example explain how to support multiple inheritance

Ans. To implement the concept of multiple inheritance using interfaces? Explain with an example.

Java does not support multiple inheritance. It provides a method called interface to implement multiple interface. An interface is basically a kind of class. It contains methods and variables. The general form of interface definition is:

Programs to implement multiple inheritances

```
interface A
{
   int x=100;
   void display _ A( );
    }
 interface B
{
   int y=200;
      void display _ B( );
    }
   class C implements A,B
 {
       public void display _ A( )
 {
   System. out. println ("x="+x);
 }
      public void display _ B()
 {
 System. out.println ("y="+y);
 }
}
```



```
class Impl _ inface
 {
     public static void main(String args[])
{
   C obj = new  C();
   obj. display _ A();
 obj. display _ B ();
   }
   }
```

3. Write a program to implement interfaces.

Ans. An interface can be used as super class, where all the members can be inherited by classes. The following example illustrates how to implement interface.

```
class classname implements interfacename
 {
 Body of a class;
 }
```

When a class implements more than one interface, they are separated by commas. The following example illustrates how to implement interface.

Example: Program to implement an interface.

```
interface Area
{
   final static float PI=3.14F;
   float compute (float X, float Y);
}
 class Rectangle implements Area
{
```

```
public float compute (float X, float Y)
{
 return ( X*Y);
 }
}
class Imple _ Interface
{
public static void main (String args [ ])
{
Rectangle rect=new Rectangle ( );
System. out. println ("Area of Rectangle is: " + rect.compute (10.5f, 20.7f));
}
 }
```


UNIT 4

PACKAGES: PUTTING CLASSES TOGETHER

SYLLABUS

Introduction, Java API Packages, Using System Packages, Naming Conventions, Creating Packages, Accessing a Package, Using a Package, Adding a Class to a Package, Hiding Classes, Static Import.

SYNOPSIS

Introduction:

The main feature of OOP is its ability to reuse the code already created. One way of achieving this is by extending the classes and implementing the interfaces. This is limited to reusing the classes within a program. What if we need to use classes from other programs without physically copying them into a program under development? This can be achieved in Java by using what is known as packages, a concept similar to **"class libraries"** in other languages.

Advantages of packages:

It is a way of grouping a variety of classes, interfaces and sub-packages together. The grouping is usually done according to functionality. In fact, packages act as **"containers"** for classes. By organizing our classes into packages we achieve the following benefits.

1. The classes contained in the packages of other programs can be easily reused.
2. In packages, classes can be unique compared with classes in other packages. That is, two classes in two different packages can have the same name. This may be referred by their fully qualified name, comprising the package name and the class name.
3. Packages provide a way to **"hide"** classes thus preventing other programs or packages from accessing classes that are meant for internal use only.
4. Java package provides access protection.
5. Java package removes naming collision.

Types of packages:

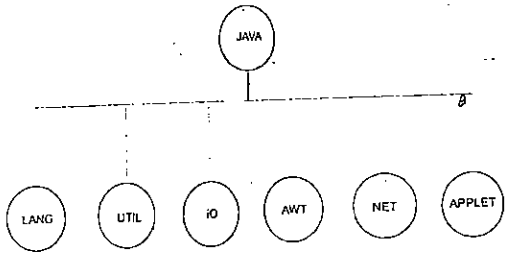
There are two types of packages one for the internal representation classes for handling our data and use existing class libraries for designing user interfaces. The types are

1. API (Application Programming Interface) packages.

2. User-defined packages.

Java API packages:

Java API provides a large number of classes grouped into different packages according to functionality. Most of the time we use the packages available with the java. The following figure shows the classes that belonging to each package. The packages are lang, awt, javax, swing, net, io, util etc.



Sl. No	Package name	Contents
1.	Java. lang	Language support classes. These are classes that java compiler itself uses and therefore they are automatically imported.
2.	Java. util	Language utility classes such as vectors, hash tables, random numbers, data, etc.
3	Java. io	Input/output support classes. They provide facilities for the input and output of data.
4	Java.awt	Set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on.
5	Java.net	Classes for networking. They include classes for communicating with local computers as well as internet servers.
6	Java.applet	Classes for creating and implementing applets.

Using system packages:

The packages are organized in a hierarchical structure as shown below. This shows that the package named java contains the package awt, which in turn contains various classes required for implementing graphical user interface.

There are two ways of accessing the classes stored in a package. The general syntax is

import firstpackage.secondpackage.Myclass;

1. Using the fully qualified class name: In this approach use the fully qualified class name in the import statement.

import Java. awt. Color;

2. Using all the members of a package: Sometimes it might be necessary to use more than one class of a single package in a program. Instead of using a separate import statement using fully qualified names as follows:

import Java. awt. Color;

import Java. awt. Font;

import Java. awt. imsge;

OR

import Java. awt.*;

Here, package name may denote a single package or a hierarchy of packages. The star (*) indicates that the all classes contained in package are imported to source code. Multiple import statements are allowed.

Imports the class Myclass and allows accessing its member's directly using class name.

How to create packages:

We know that, the java system packages are organized and used. To create our own packages, we must first declare the name of the package using the package keyword followed by a package name. This must be the first statement in a java source file. The general syntax is

```
package mypackage;   // package declaration
public class Myclass   //class definition
{
// body of class
}
```

Here, my package is a package name is considered as part of this package. The above definition must be now saved as MY class. Java in a directory named my package (a subdirectory created within working directory. When the source file compiled Java will create a .class file in the same

directory. Notice that the .class file must be located in a directory that has the same name as the package, and this directory must be the subdirectory of the directory where classes that will import the packages are located.

The steps involved in creating a package.

Packages are used to group a variety of classes and/or interfaces together so that they can be used in another program. The following are the steps involved in creating a package.

1. Declare the package at the beginning of the file by using the following syntax

package packagename;

2. Define the class and Interface that are part of package and declare it Public.

3. Create a subdirectory under the directory where the main source files are stored.

4. Store the listing as the class name. Java file in the sub directory created.

5. Compile the file. This creates .class file in the sub directory.

How to access user created packages?

The approaches used to access API packages can be used to access user created packages: fully qualified class name consisting of packagename and class name separated by dots or import statement. We use import statement when there are many references to a particular package or the package name is too long.

The general form of import statement is

import package1[.package2][.package3].....[packagen].class name;

To hide some of the classes in the package from accessing from outside the package?

When we import a package using asterisk (*) all public classes are imported. However if you want to "not import" some classes, that is you want to hide these classes from accessing outside of the package such classes should be declared **"non public".**

Example:

```
Package mypack1;
Public class A
{
Body of A
}
```



```
Class B //not public hidden
{
//Body of B
}
```

Here, the class B which is not public is hidden from outside of the package mypack1.

Consider the below program segment which imports mypack1.

```
import mypack1.*
.....
A a =new A(); //ok
B b =new B( ); // error
```

You are making attempt to create an object of B class. But class B is not accessible outside the package. Therefore compiler generates an error message.

static Import

The static import feature of Java 5 facilitates the java programmer to access any static member of a class directly. There is no need to qualify it by the class name.

Advantage of static import:

Less coding is required if you have access any static member of a class oftenly.

Disadvantage of static import:

If you overuse the static import feature, it makes the program unreadable and unmaintainable.

Simple Example of static import

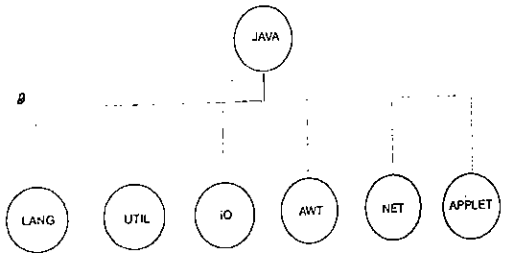
```
import static java. lang. System.*;
class StaticImport Example
{
  public static void main (String args[])
{
     out. println ("Hello"); //Now no need of System. out
  out. println ("Java");
  }
}
```

5 MARK QUESTIONS AND ANSWERS

1. Define a package. List Java API packages

Ans. It is a way of grouping a variety of classes, interfaces and sub-packages together. The grouping is usually done according to functionality.

Java API provides a large number of classes grouped into different packages according to functionality. Most of the time we use the packages available with the java. The following figure shows the classes that belonging to each package. The packages are lang, awt, javax, swing, net, io, util etc.



2. How to create and implement a package

Ans. The steps involved in creating a package.

Packages are used to group a variety of classes and/or interfaces together so that they can be used in another program. The following are the steps involved in creating a package.

1. Declare the package at the beginning of the file by using the following syntax **package packagename;**
2. Define the class and Interface that are part of package and declare it Public.
3. Create a subdirectory under the directory where the main source files are stored.
4. Store the listing as the classname.java file in the sub directory created.
5. Compile the file. This creates .class file in the sub directory.

3. Explain the naming convention of a package with an example

Ans. By using standard Java naming conventions, you make your code easier to read for yourself and for other programmers. Readability of Java program is very important. It indicates that less time is spent to figure out what the code does. Package name should be in lower case letter e.g. java, lang, sql, util etc

4. Explain how to access packages with an example

Ans. The approaches used to access API packages can be used to access user created packages: fully qualified class name consisting of packagename and class name separated by dots or import statement. We use import statement when there are many references to a particular package or the package name is too long.

The general form of import statement is

import package1[.package2][.package3].....[packagen].class name;

Example:

```
import Java. awt. Color;
import Java. awt. Font;
import Java. awt. imsge;
OR
import Java. awt.*;
```

5. Explain how to add a class to a package with an example

Ans. We know that, the java system packages are organized and used. To create our own packages, we must first declare the name of the package using the package keyword followed by a package name. This must be the first statement in a java source file. The general syntax is

```
package mypackage;   // package declaration
public class Myclass   //class definition
{
// body of class
}
```

Here, mypackage is a package name is considered as part of this package. The above definition must be now saved as MYclass.Java in a directory named mypackage (a subdirectory created within working directory. When the source file compiled Java will create a .class file in the same directory. Notice that the .class file must be located in a directory that has the same name as the package, and this directory must be the subdirectory of the directory where classes that will import the packages are located.

6. Discuss the various levels of access protection available for packages

Ans.

Access location ↓ / Access modifier →	Public	Protected	Friendly (default)	Private Protected	Private
Same class	Yes	Yes	Yes	Yes	Yes
Subclass in same package	Yes	Yes	Yes	Yes	No
other classes in same package	Yes	Yes	Yes	No	No
Subclass in other packages	Yes	Yes	No	Yes	No
Non-subclasses in other packages	Yes	No	No	No	No

7. Explain how to hide classes in a package

Ans. When we import a package using asterisk (*) all public classes are imported. However if you want to "not import" some classes, that is you want to hide these classes from accessing outside of the package such classes should be declared "non public".

Example:

```
Package mypack1;
Public class A
{
Body of A
}
Class B //not public hidden
{
//Body of B
}
```

Here, the class B which is not public is hidden from outside of the package mypack1.

Consider the below program segment which imports mypack1.

You are making attempt to create an object of B class. But class B is not accessible outside the package. Therefore compiler generates an error message.

8. Explain static import and how is it useful.

Ans. The static import feature of Java 5 facilitates the java programmer to access any static member of a class directly. There is no need to qualify it by the class name.

Advantage of static import:

Less coding is required if you have access any static member of a class oftenly.

Disadvantage of static import:

If you overuse the static import feature, it makes the program unreadable and unmaintainable.

Simple Example of static import

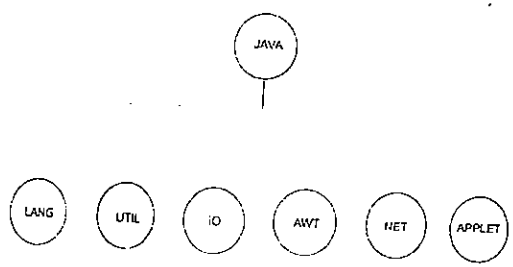
```
import static Java. lang. System.*;
class Static Import Example
{
 public static void main (String args [ ])
{
  out. println ("Hello"); //Now no need of System. out
  out. println ("Java");
  }
}
```

10 MARKS QUESTIONS AND ANSWERS

1. Explain Java API packages.

Ans. Java API packages:

Java API provides a large number of classes grouped into different packages according to functionality. Most of the time we use the packages available with the java. The following figure shows the classes that belonging to each package. The packages are lang, awt, javax, swing, net, io, util etc.



Sl. No	Package name	Contents
1.	Java. lang	Language support classes. These are classes that java compiler itself uses and therefore they are automatically imported.
2.	Java. util	Language utility classes such as vectors, hash tables, random numbers, data, etc.
3	Java. io	Input/output support classes. They provide facilities for the input and output of data.
4	Java. awt	Set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on.
5	Java. net	Classes for networking. They include classes for communicating with local computers as well as internet servers.
6	Java. applet	Classes for creating and implementing applets.

2. Write a program to add a class to a package.

Ans. We know that, the Java system packages are organized and used. To create our own packages, we must first declare the name of the package using the package keyword followed by a package name. This must be the first statement in a Java source file. The general syntax is

```
package mypackage;   // package declaration
public class Myclass   //class definition
{
// body of class
}
```

Here, mypackage is a package name is considered as part of this package. The above definition must be now saved as MYclass. Java in a directory named mypackage (a subdirectory created within working directory. When the source file compiled Java will create a .class file in the same directory. Notice that the .class file must be located in a directory that has the same name as the package, and this directory must be the subdirectory of the directory where classes that will import the packages are located.

3. Write a Package program to demonstrate basic arithmetic operators

Ans. Note: First create a directory by name "Arithmetic" and move to that directory, create the following file and compile the file in the same directory.


```
package arithmetic;
public class MyMath
{
public int add(int x,int y)
{
return x+y;
}
public int sub(int x,int y)
{
return x-y;
}
public int mul(int x,int y)
{
return x*y;
}
public double div(int x,int y)
{
return (double)x/y;
}
public int mod(int x,int y)
{
return x%y;
}
}
```

Note: Move to parent directory, write the following file and execute it.

```
import arithmetic.*;
class Test
{
public static void main(String as[])
{
MyMath m=new MyMath();
```

```
System. out. println (m.add(8,5));
System. out. println( m. sub(8,5));
System. out. println (m. mul(8,5));
System. out. println (m.div(8,5));
System. out. println (m.mod(8,5));
}
}
```

4. Write a program to use inbuilt packages to calculate square root of a number

Ans.

```
package com. tutorialspoint;
import Java.lang.*;
public class MathDemo {
  public static void main(String[] args) {
  // get two double numbers numbers
  double x = 9;
  double y = 25;
  // print the square root of these doubles
  System. out. println ("Math. sqrt (" + x + ")=" + Math. sqrt(x));
  System. out. println ("Math. sqrt(" + y + ")=" + Math. sqrt(y));
  }
}
```


UNIT 5

MULTI THREADED PROGRAMMING

SYLLABUS

Introduction, Creating Threads, Extending the Thread Class, Stopping and Blocking a Thread, Life Cycle of a Thread, Using Thread Methods, Thread Exceptions, Thread Priority, Synchronization, Implementing the 'Runnable' Interface, Inter-thread Communication.

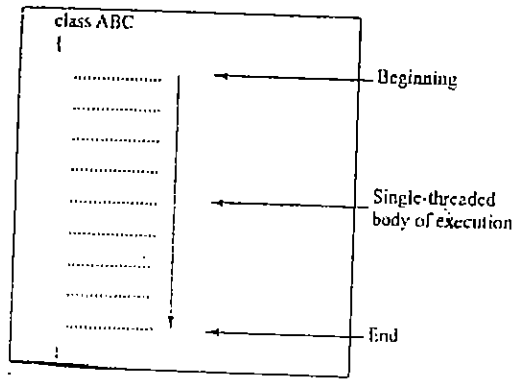
SYNOPSIS

Introduction:

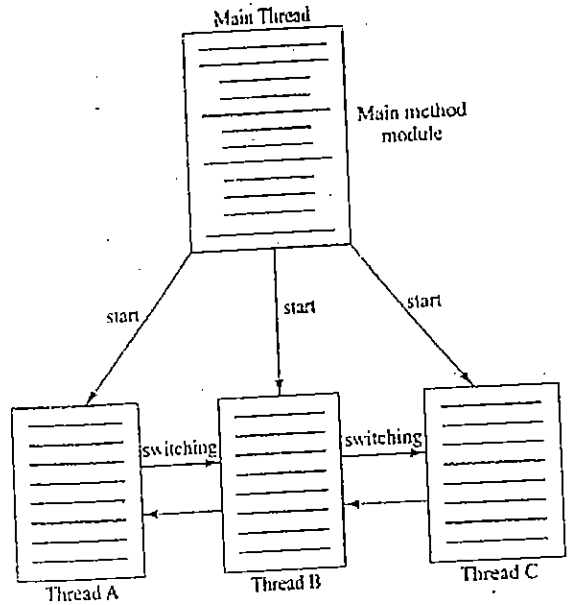
Normally all programs will have a single point of control. The statements present in the programs are executed in a sequential manner. Such a program or process is called a heavy weight process. Sometimes it might be required to divide a process into several sub processes each having a separate point of control meaning the execution of one sub-process does not depend on the other sub-processes. In other words, the java subprograms of a main application program and share the same memory space, they are known as lightweight threads or lightweight processes.

What is a thread? Explain the way of creating thread using thread class.

A thread is a sequence of code statements executing within the context of a process. Threads cannot execute on its own; they require the overhead of a parent process to run. It is also called single threaded.



A program having multiple independent point of control is called multi-thread.



Threads are more useful programming tools for two main reasons. First, they enable programs to do multiple things at a time. This is useful for such activities as letting a user to do something while something else is happening in the back ground.

Threads are extensively used in java-enabled browsers such as hot-java. These browsers can download a file to the local computer.

Sl. No	Multi-threading	Multi-tasking
1.	It is a programming concept in which a program or a process is divided into two or more subprograms or threads that are executed at the same time in parallel.	It is an operating system concept in which multiple tasks are performed simultaneously.
2.	It supports execution of multiple parts of a single program simultaneously.	It supports execution of multiple programs simultaneously.

3.	The processor has to switch between different parts or threads of a program.	The processor has to switch between different programs or processes.
4.	It is highly efficient.	It is less efficient.
5.	A thread is the smallest unit in multi-threading.	A program or process is the smallest unit in a multitasking environment.
6.	It helps in developing efficient programs.	It helps in developing efficient operating systems.

Creating threads:

Creating threads in Java is simple. Threads are implemented in the form of objects that contain a method called run (. The run () method is the heart and soul of any thread. It makes up the entire body of a thread and is the only method in which the thread's behavior can be implemented. A typical run appears like

```
public void run ( )
{
.....
.....
}
```

The run () method should be invoked by an object of the concerned thread. This can be achieved by creating the thread and initiating with the help of another thread method called start.

A new thread can be created in two ways.

1. By creating a thread class: Define a class that extends Thread class and override its run () method with the code required by the thread.

```
class thread _ name extends Thread
{
Body of thread
}
```

2. By converting a class to a thread: define a class that implements Runnable interface. The Runnable interface has only one method, run (), that is to be defined in the method with the code to be executed by the thread.

Extending the thread class:

A thread is similar to a program that has a single flow of control. We can create thread that extends Thread class and override its run ( ) method with the code required by the thread.

It uses the following steps.

1. Declare the class which extends Thread class.
2. Implement the run ( ) method.
3. Create a thread object and call the start ( ) method to initiate the thread execution.

Starting a new thread:

To actually create and run an instance of our thread class, we must write,

class _ name thread _ object = new class _ name(); // Thread is in born state.

thread _ object. start(); // invokes a run method. It moves from born state to running state.

The main method dies at the end of its main method. However, before it dies, it creates and starts all the three threads.

new class _ name(.start();

Stopping and blocking a thread:

Whenever we want to stop a thread from running further, then we can make use of stop ( ) method.

Thread object name. stop( );

This statement causes the thread to move to the dead state.

A Thread will also move to the dead state automatically when it reaches the end of its method.

A Thread can also be temporarily suspended or blocked from entering into runnable and subsequently running state by using the following thread methods.

sleep ( ); // Blocked for a specified time

suspend ( ); // Blocked until further orders

wait ( ); // Blocked until certain condition occurs.

These methods cause the thread to go into blocked state. The thread will return into the runnable state when the specified time is elapsed in the case of sleep( ) method. The resume ( ) method is invoked in the case of suspend( ) method and notify( ) method is called in case of wait() method.

Consider the following example.

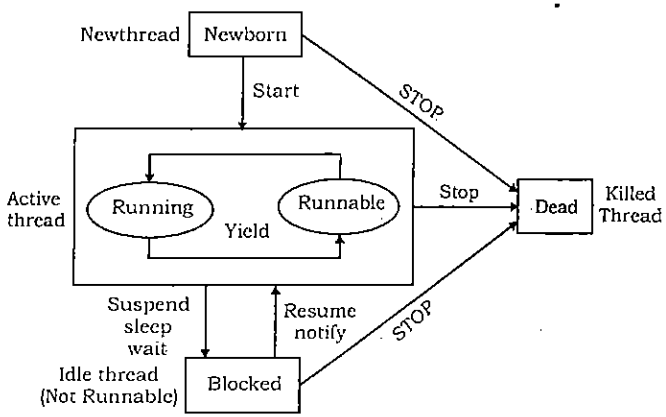

```
class A extends Thread
{
   public void run( )
    {
for(int i=1;i<=5;i++)
          {
            System. out. println ("From Thread A : i="+i);
          }
            System. out. println ("Exit from Thread A");
      }
}

class B extends Thread
{
   public void run( )
    {
for(int j=1;j<=5;j++)
          {
            System. out. println ("From Thread B : j="+j);
          }
            System. out. println ("Exit from Thread B");
        }
}

class C extends Thread
{
   public void run( )
    {
for(int k=1;k<=5;k++)
          {
            System. out. println ("From Thread C : k="+k);
          }
```

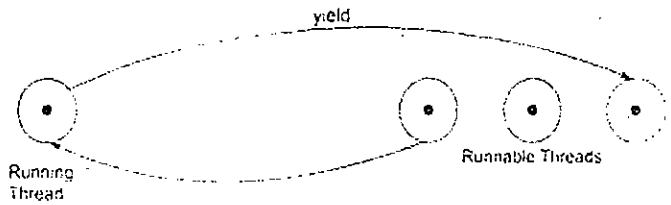
```
        System. out. println ("Exit from Thread C");
    }
}
```

Life cycle of a thread: A thread is a single flow of execution with in a program. During the life time of a thread, it may enter many states which as follows.



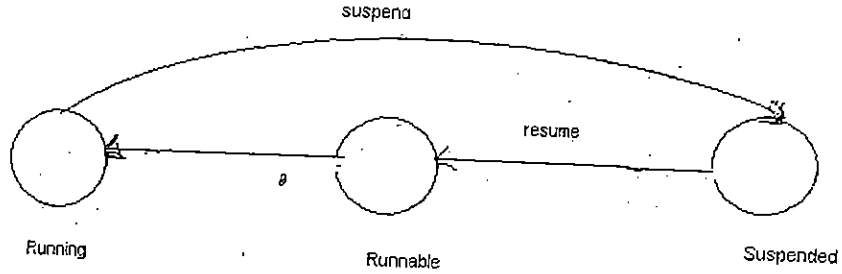
1. New born state: When we create a thread object, the thread is born and is said to be new born state. At this state, we can make the thread running by using start ( ) method or we can kill the thread by using stop ( ) method.

2. Runnable state: When a thread is ready for execution and is waiting for the CPU, then the thread is said to be in Runnable state. When we want a thread to relinquish control to another thread, then we can make use of yield ( ) method is shown below.

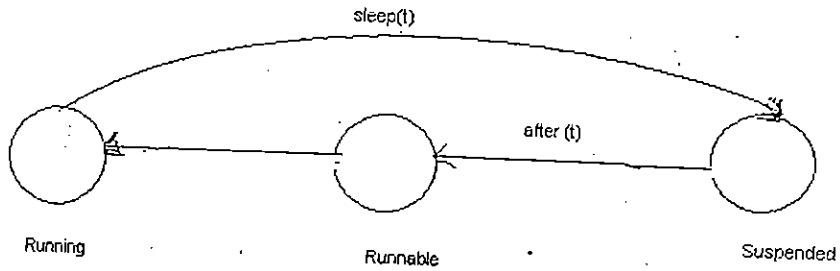


3. Running state: When a thread is being executed by CPU, then the thread is said to be running state. A thread may relinquish control when it finishes execution or preempted by higher priority thread. A running thread may relinquish its control in one of the following situations.

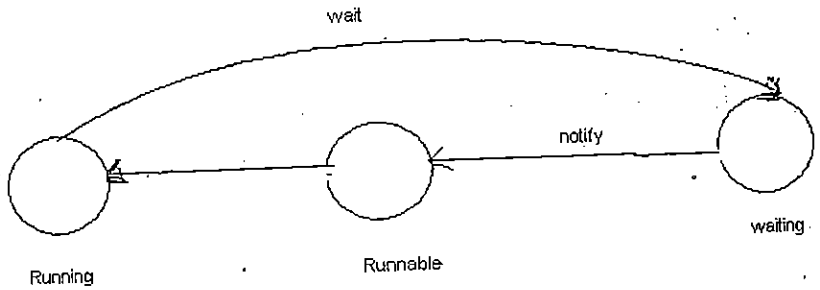
a. It has been suspended using suspend () method. A suspended thread can be reviewed bt using the resume () method. These approaches useful when we want to to suspend a thread for some time due to certain reasons, but do not want to kill it.



b. We can put the thread to sleep for a specific amount of time period using the method sleep(time) where time is in milliseconds. This means that the thread is out of the queue during this time period.



c. It has been told to wait until some event occurs. This is done using the wait () method. The thread can be scheduled to run using the notify () method.



4. Blocked state: A thread can be blocked by using suspend ( ), sleep ( ) and wait ( ) method. It prevents the thread from entering into the runnable state and thus running state. When we want to block the thread for certain reason then we make use of suspend ( ) method. We can revoke the suspend ( ) method by using resume ( ) method. When we want to block the thread until some event occurs then we make use of wait ( ) method by using notify ( ) method.

When we want to block the thread for specified time period then we make use of sleep ( ) method. The Thread will reenter into runnable state as soon as the time period is elapsed.

5. Dead state: When a thread finishes its execution, it enters into Dead state. We can also kill the thread by using stop ( ) method.

Using thread methods:

Thread methods can be used to control the behavior of thread. The below program illustrates the use of **yield (), sleep () and stop ()** methods.

```
class A extends Thread
{
   public void run( )
    {
for(int i=1;i<=5;i++)
          {
            If (i==1) yield();
System. out. println ("From Thread A  : i="+i);
          }
            System. out. println ("Exit from Thread A");
       }
```



```
}
class B extends Thread
{
   public void run( )
    {
for(int j=1;j<=5;j++)
          {
            System. out. println ("From Thread B  : j="+j);
If (j==3) stop();
          }
            System. out. println ("Exit from Thread B");
     }
  }
class C extends Thread
{
   public void run( )
    {
for(int k=1;k<=5;k++)
          {
            System. out. println ("From Thread C  : k="+k);
          if (k==1)
    try
    {   sleep (1000);
    }
catch (Exception e)
          {
          }
}   System. out. println ("Exit from Thread C");
     }
}
catch (Exception e)
          {
```

```
            }
}   System. out. println ("Exit from Thread C");
      }
}
class Thread Test
{
     public static void main (String args[ ] )
    {
  System. out. println ("Start a thread A");
  new A(.start();
System. out. println("Start a thread B");
 new B(.start();
System. out. println("Start a thread C");
   }
}
```

Thread priorities to threads with example

In Java, we can assign a priority to each thread. By assigning a priority, we can schedule the order for running. We can set the priority of a thread by using the setPriority ( ) method which is as follows:

Thread Name. Set Priority (int Number);

The int Number is an integer value to which the thread's priority is set.The Thread class defines several priority constants which is as follows:

MIN _ PRIORITY = 1

NORM _ PRIORITY = 2

MAX _ PRIORITY = 10

The int Number can assume one of these constants or any value between 1 to 10.The default setting is NORM _ PRIORITY.

The Following example illustrates use of priority in threads:

```
class A extends Thread
{
public void run( )
```



```
{
System. out. println(" Thread A started");
for (int i=1; i<=5; i++)
{
System. out. println("From thread A");
}
}
}
class B extends Thread
{
public void run( )
{
System. out. println(" Thread B started");
for (int j=1; j<=5; j++)
{
System. out. println("From thread B");
}
}
}
class C extends Thread
{
public void run( )
{
System. out. println(" Thread B started");
for (int k=1; k<=5;k++)
{
System. out. println("From thread C");
}
}
}
class Thread priority
{
public static void main( String args [ ])
```

```
{
A thread A=new A( );
B thread B =new B( );
C thread C=new C( );
thread A. set Priority (Thread. MIN _ PRIORITY);
thread B . set Priority(Thread. MAX _ PRIORITY);
thread A . start( );
thread B . start( );
thread C. start();
System. out. println("End of Main thread");
}
 }
```

Whenever multiple threads are ready for execution, the Java chooses the highest priority thread and executes it. When Java is currently executing Low priority thread and if another thread of higher priority comes along, the currently running thread will be preempted and moves that thread into runnable state.

Always we want the main thread to finish first. One approach is to call sleep () method within main (), with a long enough delay to ensure that all child threads terminate prior to main thread. However, this is not a satisfactory solution, and it also raises a question: How can one thread know when another thread has ended.

Two ways exist to determine whether a thread has finished. First you can call isAlive () on the thread. This method is defined by Thread, and its general form is shown here

Final Boolean is Alive ();

The is Alive () method returns true if the thread upon which it is called is still running. It returns false otherwise.

While is alive () is occasionally useful, the method that you will more commonly use to wait for a thread to finish is join ();

Syntax: **public final void join (long millisecond);**

The join () method makes a thread to wait for the thread to finish on which join() is called. To wait for a thread to terminate, invoke one of the join() methods on its thread object.

Synchronization? When to use it?

In multithreading, we may see there are more than one thread that try to access the same resources at the same time. For example, one thread may try to read a record from a file while another thread is still writing to the same file.

Such situation will bring incorrect result. Java can overcome this problem by using a technique known as synchronization. Java provides a keyword **'synchronised'** that helps to solve the problem.

When we declare a method synchronized, Java creates a "monitor" and hands it over to the thread that calls the method first time. As long as the thread holds the monitor, no other thread can enter the synchronized section of code.

```
synchronized void display ( )
{
.....
.....// code here is synchronized
.....
}
```

Whenever a thread has completed its work of using synchronized method, it will hand over the monitor to the next thread that is ready to use the same resource.

```
Class thread _ name implements Runnable
{
.....
}
```

Inter–thread communication:

Java provides a very efficient way through which multiple-threads can communicate with each other. This way reduces the CPU idle time. i.e., a process where a thread is paused running in its critical region and another thread is allowed to enter in the same critical section to be executed. This technique is known s inter-thread communication which is implemented by some methods. These methods are defined in **"Java.lang"** package and can only be called within synchronized code shown as:

1. **Notify ( ):** It wakes up the first thread that called wait ( ) in the same object.

final void notify ()

2. Notify All( ): It wakes up all the thread that called wait( ) on the same object. The highest priority thread will run first.

final void notify all ()

3. **wait ( ):** It indicates the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls method Notify or NotifyAll(.

final void wait ()

Creating thread using Runnable interface:

We know that threads can be created using runnable threads. It maks use of runnable interface to implement threads.

The Runnable interface declares the run method () that is required for implementing threads in our programs. To do this, we must perform the following steps.

1. Declare the class as implementing the Runnable interface.

2. Implement the run () method.

3. Create a thread by defining an object that is instantiated from this "Runnable" class as the target of the thread.

4. Call the thread's start() method to run the thread.

Consider the program for Runnable interface

```
class A implements Runnable
{
   public void run( )
    {
for(int i=1;i<=10;i++)
          {
           System. out. println ("From Thread A  : i="+i);
          }
            System. out. println ("Exit from Thread A");
   }
}
```



```
class Thread Run
{
    public static void main (String args[ ] )
    {
A runnable =new A();
Thread thread A= new Thread(runnable);
threadA. start();
System. out. println("End of the main Thread");
}
}
```

5 MARK QUESTIONS AND ANSWERS

1. Define thread.

Ans. A thread is a sequence of code statements executing within the context of a process. Threads cannot execute on its own; they require the overhead of a parent process to run. It is also called single threaded.

Threads are more useful programming tools for two main reasons. First, they enable programs to do multiple things at a time. This is useful for such activities as letting a user to do something while something else is happening in the back ground.

2. Differentiate between multi threading and multitasking

Ans. The differences between multi threading and multitasking are listed below.

Sl. No	Multi-threading	Multi-tasking
1.	It is a programming concept in which a program or a process is divided into two or more subprograms or threads that are executed at the same time in parallel.	It is an operating system concept in which multiple tasks are performed simultaneously.
2.	It supports execution of multiple parts of a single program simultaneously.	It supports execution of multiple programs simultaneously.
3.	The processor has to switch between different parts or threads of a program.	The processor has to switch between different programs or processes.

4.	It is highly efficient.	It is less efficient.
5.	A thread is the smallest unit in multi-threading.	A program or process is the smallest unit in a multitasking environment.
6.	It helps in developing efficient programs.	It helps in developing efficient operating systems.

3. Explain how to create thread by extending Thread class with an example

Ans. Creating threads in java is simple. Threads are implemented in the form of objects that contain a method called run (. The run () method is the heart and soul of any thread. It makes up the entire body of a thread and is the only method in which the thread's behavior can be implemented. A typical run appears like

The run () method should be invoked by an object of the concerned thread. This can be achieved by creating the thread and initiating with the help of another thread method called start.

A new thread can be created in two ways.

Define a class that extends Thread class and override its run () method with the code required by the thread.

```
class thread_name extends Thread
{
Body of thread
}
```

4. Explain how to stop and block a thread

Ans. Stopping and blocking a thread:

Whenever we want to stop a thread from running further, then we can make use of stop ( ) method.

Thread object name. stop( );

This statement causes the thread to move to the dead state.

A Thread will also move to the dead state automatically when it reaches the end of its method.

A Thread can also be temporarily **suspended or blocked** from entering into runnable and subsequently running state by using the following thread methods.

sleep ( ); // Blocked for a specified time

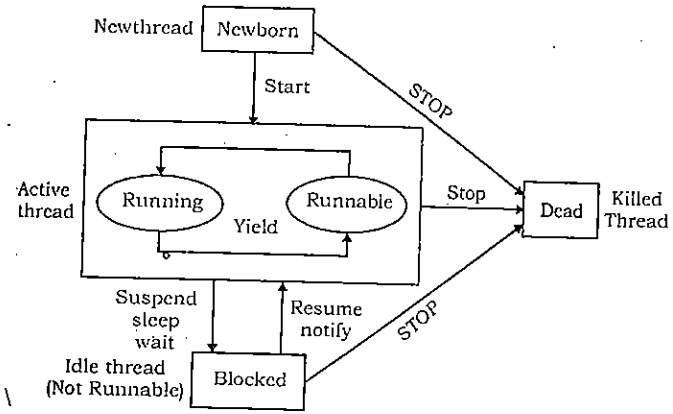
suspend ( ); // Blocked until further orders

wait ( ); // Blocked until certain condition occurs.

These methods cause the thread to go into blocked state. The thread will return into the runnable state when the specified time is elapsed in the case of sleep( ) method. The resume ( ) method is invoked in the case of suspend ( ) method and notify ( ) method is called in case of wait ( ) method.

5. Describe complete life cycle of a thread

Ans. A thread is a single flow of execution with in a program. During the life time of a thread, it may enter many states which as follows.



1. New born state: When we create a thread object, the thread is born and is said to be new born state. At this state, we can make the thread running by using start ( ) method or we can kill the thread by using stop ( ) method.

2. Runnable state: When a thread is ready for execution and is waiting for the CPU, then the thread is said to be in Runnable state. When we want a thread to relinquish control to another thread, then we can make use of yield ( ) method is shown below.

3. Running state: When a thread is being executed by CPU, then the thread is said to be running state. A thread may relinquish control when it finishes execution or preempted by higher priority thread. A running thread may relinquish its control in one of the following situations.

4. Blocked state: A thread can be blocked by using suspend ( ), sleep ( ) and wait ( ) method. It prevents the thread from entering into the runnable state and thus running state. When we want to block the thread for certain reason then we make use of suspend ( ) method. We can revoke the

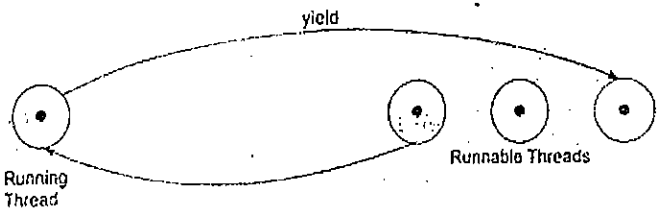
suspend ( ) method by using resume ( ) method. When we want to block the thread until some event occurs then we make use of wait ( ) method by using notify ( ) method.

When we want to block the thread for specified time period then we make use of sleep ( ) method. The Thread will reenter into runnable state as soon as the time period is elapsed.

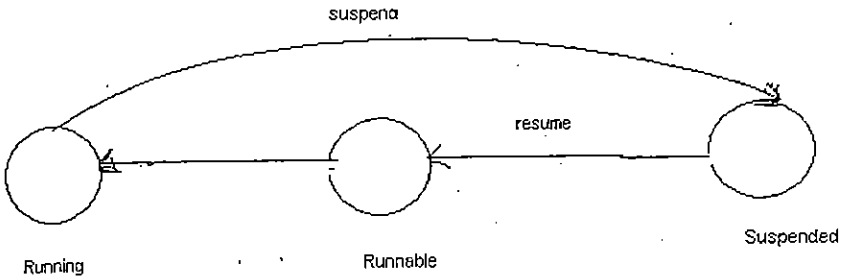
5. Dead state: When a thread finishes its execution, it enters into Dead state. We can also kill the thread by using stop ( ) method.

6. Explain yield(),sleep() and stop() methods of a thread

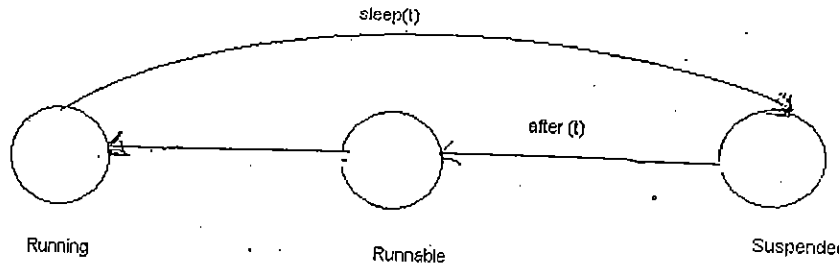
Ans. When we want a thread to relinquish control to another thread, then we can make use of yield () method is shown below.



It has been suspended using suspend () method. A suspended thread can be reviewed bt using the resume () method. These approaches useful when we want to to suspend a thread for some time due to certain reasons, but do not want to kill it.



We can put the thread to sleep for a specific amount of time period using the method sleep(time) where time is in milliseconds. This means that the thread is out of the queue during this time period.



7. How do we set priorities for threads?

Ans. In Java, we can assign a priority to each thread. By assigning a priority, we can schedule the order for running. We can set the priority of a thread by using the setPriority ( ) method which is as follows:

Thread Name. Set Priority (int Number);

The int Number is an integer value to which the thread's priority is set. The Thread class defines several priority constants which is as follows:

MIN _ PRIORITY = 1

NORM _ PRIORITY = 2

MAX _ PRIORITY = 10

The int Number can assume one of these constants or any value between 1 to 10.The default setting is NORM _ PRIORITY.

8. How to create a Runnable thread?

Ans. We know that threads can be created using runnable threads. It maks use of runnable interface to implement threads. The Runnable interface declares the run method () that is required for implementing threads in our programs. To do this, we must perform the following steps.

1. Declare the class as implementing the Runnable interface.

2. Implement the run () method.

3. Create a thread by defining an object that is instantiated from this "Runnable" class as the target of the thread.

4. Call the thread's start() method to run the thread.

Consider the program for Runnable interface

9. Define synchronization? When do we use it

Ans. In multithreading, we may see there are more than one thread that try to access the same resources at the same time. For example, one thread may try to read a record from a file while another thread is still writing to the same file.

Such situation will bring incorrect result. Java can overcome this problem by using a technique known as synchronization. Java provides a keyword 'synchronised' that helps to solve the problem.

When we declare a method synchronized, java creates a "monitor" and hands it over to the thread that calls the method first time. As long as the thread holds the monitor, no other thread can enter the synchronized section of code.

```
synchronized void display ( )
{
.....
.....// code here is synchronized
.....
}
```

Whenever a thread has completed its work of using synchronized method, it will hand over the monitor to the next thread that is ready to use the same resource.

```
Class thread _ name implements Runnable
{
.....
}
```

10. Differentiate between suspending and stopping a thread

Ans.

Sl. No	suspending	stopping
1.	A suspend () method will move the thread to blocked state.	A stop () method moves the thread to dead state.

2.	A Thread will also move to the dead state automatically when it reaches the end of its method.	Once it moves to the dead state, it will not come to runnable or run state.
3.	It is blocked until further orders.	It is totally dead state.

10 MARK QUESTIONS AND ANSWERS

1. Explain the different methods of creating threads

Ans. Creating threads in java is simple. Threads are implemented in the form of objects that contain a method called run (. The run () method is the heart and soul of any thread. It makes up the entire body of a thread and is the only method in which the thread's behavior can be implemented. A typical run appears like

```
public void run ( )
{
.....
.....
}
```

The run ( ) method should be invoked by an object of the concerned thread. This can be achieved by creating the thread and initiating with the help of another thread method called start.

A new thread can be created in two ways.

a. By creating a thread class: Define a class that extends Thread class and override its run () method with the code required by the thread.

```
class thread _ name extends Thread
{
Body of thread
}
```

b. By converting a class to a thread: define a class that implements Runnable interface. The Runnable interface has only one method, run (), that is to be defined in the method with the code to be executed by the thread.

2. Write a program to create a threads using a thread class

Ans. A thread can be created using a thread class as shown below.

```
class A extends Thread
{
   public void run( )
    {
for(int i=1;i<=5;i++)
            {
              System. out. println ("From Thread A  : i="+i);
            }
              System. out. println ("Exit from Thread A");
       }
}
class Threadext
{
public static void main (String args[ ] )
   {
new A(.start();
   }
}
```

3. Explain thread creation by implementing runnable interface with an example

Ans. The Runnable interface declares the run method () that is required for implementing threads in our programs. To do this, we must perform the following steps.

1. Declare the class as implementing the Runnable interface.
2. Implement the run () method.
3. Create a thread by defining an object that is instantiated from this "Runnable" class as the target of the thread.
4. Call the thread's start () method to run the thread.

Consider the program for Runnable interface

```
class A implements Runnable
{
```



```
   public void run( )
    {
for(int i=1;i<=10;i++)
            {
             System. out. println ("From Thread A  : i="+i);
            }
              System. out. println ("Exit from Thread A");
   }
}
  class ThreadRun
{
    public static void main (String args[ ] )
   {
A runnable =new A();
Thread threadA= new Thread(runnable);
threadA.start();
System.out.println("End of the main Thread");
}
}
```

4. Write a note on inter- thread communication

Ans. Java provides a very efficient way through which multiple-threads can communicate with each other. This way reduces the CPU idle time. i.e., a process where a thread is paused running in its critical region and another thread is allowed to enter in the same critical section to be executed. This technique is known s Interthread communication which is implemented by some methods. These methods are defined in **"Java. lang"** package and can only be called within synchronized code shown as:

1. **Notify ( ):** It wakes up the first thread that called wait ( ) in the same object.

final void notify ()

2. **NotifyAll( ):** It wakes up all the thread that called wait( ) on the same object. The highest priority thread will run first.

final void notifyall ()

3. wait ( ): It indicates the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls method Notify or Notify All(.

final void wait ()

5. Write program to create thread by extending thread class.

Ans. Consider an example to create thread by extending thread class.

```
class A extends Thread
{
   public void run( )
    {
for(int i=1;i<=5;i++)
            {
              System. out. println ("From Thread A  : i="+i);
            }
              System. out. println ("Exit from Thread A");
      }
}
class B extends Thread
{
   public void run( )
    {
for(int j=1;j<=5;j++)
            {
              System. out. println ("From Thread B  : j="+j);
             }
              System. out. println ("Exit from Thread B");
            }
}
class C extends Thread
{
   public void run( )
    {
for(int k=1;k<=5;k++)
```



```
            {
              System. out. println ("From Thread C  : k="+k);
            }
          System. out. println ("Exit from Thread C");
        }
    }
class Threadext
{
 public static void main (String args[ ] )
   {
new A(.start();
new B(.start();
       new C(.start();
  }
}
```

6. Write a program to set priorities for threads.

Ans.

```
class A extends Thread
{
public void run( )
{
System. out. println(" Thread A started");
for (int i=1; i<=5; i++)
{
System. out. println("From thread A");
}
}
}
class B extends Thread
{
public void run( )
{
System. out. println (" Thread B started");
```

```
for (int j=1; j<=5; j++)
{
System. out. println ("From thread B");
}
 }
}
class C extends Thread
{
public void run( )
{
System. out. println(" Thread B started");
for (int k=1; k<=5;k++)
{
System. out. println("From thread C");
}
 }
}
class Threadpriority
{
public static void main( String args [ ])
{
A thread A=new A( );
B thread B=new B( );
C thread C=new C( );
thread A. set Priority(Thread. MIN _ PRIORITY);
thread B. set Priority (Thread. MAX _ PRIORITY);
thread A. start( );
thread B. start( );
thread C. start();
System. out. println("End of Main thread");
}
}
```


UNIT 6

MANAGING ERRORS AND EXCEPTIONS

SYLLABUS

Introduction, Types of Errors, Exceptions, Syntax of Exception Handling Code, Multiple Catch Statements, Using Finally Statement, Throwing Our Own Exceptions.

SYNOPSIS

Introduction:

It is common to make mistakes while developing as well as typing a program. A mistake might lead to an error causing to program to produce unexpected results. Errors are the wrongs that can make a program go wrong.

A program may produce an incorrect output or may terminate the execution of the program abruptly or even may cause the system to crash. It is therefore important to detect and manage properly all the possible error conditions in the program so that the program will not terminate or crash during execution.

What are the types of errors in Java program?

Errors are the mistakes that can make the program to go wrong. An error may produce an incorrect output or may terminate the execution of the program abruptly or even may cause the system to crash.

An error may be classified as follows:

i. Compile-Time Errors: When the program is compiled, the java compiler will detect and displays all the syntax errors. Such errors are called compile Time errors. When the compiler displays an error. It will not create .class file.

```
class Sample
{
public Static void main (String args [ ])
{
```

```
System. out. println ("Hello")
}
}
```

In above program, the semicolon (;) is missing at the end of point statement. The java compiler displays an appropriate message to the user.

The compile-Time error could be of the following:

Most of the compile-time errors are due to typing mistakes. Typographical errors are hard to find.

1. Missing semicolons.
2. Missing brackets in the class methods.
3. Misspelling of identifiers and keywords.
4. Missing double quotes in strings.
5. Use of undeclared variables.
6. Incompatible types in assignments / initializations.
7. Bad references of objects.
8. Use of = in place of == operator, etc.

ii. Run-Time Errors: When the program is successfully compiled, a .class file is created. Then java interpreter interprets the class file. During running, the program may produce wrong results due to wrong logic. Such errors are called runtime errors.

Some of common runtime errors are:

1. Dividing an integer by zero.
2. Accessing an element that is out of the bounds of an array.
3. Converting invalid string into number.
4. Accessing array elements out of bound.
5. Attempting to use a Negative size for an array and others.
6. Converting invalid string to a number.
7. Trying to illegally change the state of a thread.
8. Attempting to use a negative size for an array.


```
class RunError
{
  public static void main (String args[ ])
  {
    int a=10;
    int b=5;
    int c=5;
      int x =a / (b-c);
    System. out. println (" The value of x is" +x);
  }
}
```

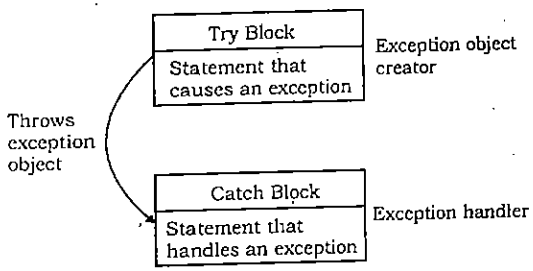
What is an exception? List the Java common exception types and causes.

An exception is a condition that is caused by a run time error in the program. When the java interpreter encounters an error such as dividing an integer by zero, it creates an exception object and throws it. If the exception object is not caught and handled properly the interpreter will display an error message and will terminate the program. If we want the program to continue with the execution of the remaining code, then we should try to catch the exception object thrown by the error condition and then display an appropriate message for taking corrective actions. This task is known as exception handling.

The try block contains one or more statements that are likely to cause an error. When an error has generated, Java interpreter creates an exception object and throws which is caught by catch block. The catch block contains one or more statement that are necessary to process the exception. The following programs illustrate how to handle arithmetic exception. The general syntax is shown below.

```
try
{
//statement(s) to monitor errors
}
catch (Exceptiontype1 e)
{
```

```
block of statements to handle exceptiontype1;
}
catch (Exceptiontype1 e)
{
block of statements to handle exceptiontype1;
}
Finally
{
block of statements to handle exception type;
}
```



Explain the constructs used in exception handling in Java.

If the exception object is not caught and handled properly, the interpreter will display an error message and will terminate the program. If we want the program to continue with the execution of the remaining code, then we should try to catch the exception object thrown by the error condition and then display an appropriate message to take corrective action. This task is known as Exception handling.

The Exception handling provides the following mechanism.

1. Find the problem **(Hit the exception)**
2. Inform that an error has occurred **(Throw the exception)**
3. Receive the error information **(Catch the exception)**
4. Take corrective action **(Handle the exception)**

Common Java Exceptions

Exception Type	Cause of exception
Arithmetic Exception	Caused by math errors such as division by zero
ArrayIndexOutOfBound Exception	Caused by bad array indexes
Array Store Exception	Caused when a program tries to store the wrong type of data in an array.
File Not Found Exception	Caused by an attempt to access a nonexistent file
IO Exception	Caused by general I/O failures
Null Pointer Exception	Caused by referencing a null object
Number Format Exception	Caused when the conversion between strings and numbers fails
Out Of Memory Exception	Caused when there is not enough memory to allocate an object
Security Exception	Caused when an applet tries to perform an action not allowed by the browsers security settings
Stack Over Flow Exception	Caused when the system runs out of stack space
StringIndex Out OF Bounds Exception	Caused when a program attempts to access a nonexistent character position in a string

Example1:

```
class Excet
{
public static void main( String args[ ])
{
 int a=10;
 int b=5;
 int c=5;
 int x,y;
 try  {
 x= a/(b-c); // Exception is generated
```

```
}
catch (Arithmetic Exception e)
{
System. out. println ("Division by zero");
}
y= a/ (b+c);
System. out. println ("y=" +y);
}
      }
```

Example 2:

```
class Exce Bound
{
public static void main( String args[ ])
{
  int a[]={1,2,3,4};
 try
 {
   System. out. println("a[0]=:"+a[10]);
 }
 catch (Array Index Out Of Bounds Exception e)
 {
 System. out. println(" Exception information is as follows...\n"+e);
 System. out. println(" You are trying to access a location which is not Present");
 }
}
}
```

What is the use of multiple catch block?

In some cases, more than one exception could be raised by a single piece of code. To handle this type of situation, you can specify two or more catch clauses, each catching a different type of exception. When an exception is thrown, each catch statement is inspected in order, and the first one of whose type matches that of the exception generated is executed. After one catch statement

executed, the others are bypassed, and execution continues after the try/catch block. The general form of exception handling code with multiple catch blocks is:

```
try
{
//statement(s) to monitor errors
}
catch (Exceptiontype1 e)
{
//statements to handle exceptiontype1
}
catch (Exceptiontype2 e)
{
//statements to handle exceptiontype2
}
catch (Exception type N e)
{
//statements to handle exception type N
}
```

You can catch all types of exceptions by setting up you catch block for exceptions of type Exception, like this

catch (Exception e) ;

Note that Java does not require any processing of the exception at all. We can simply have a catch statement with an empty block to avoid program termination. For example,

```
catch (Exception e)
{}
```

or catch (Exception e);

What is the use of finally block? When is it used?

In addition to try and catch blocks, Java also enables you to specify a finally block that will execute, regardless of any exceptions that may have occurred. This block can be used to handle an exception that is not caught by any of the catch statement. It can be used to handle any exception

generated within a try block. It may be added immediately after the try block or after the last catch block as shown below:

```
try
{
Statements(s)
}
finally
{
//statements
}
```

The finally block is guaranteed to execute regardless of whether or not an exception is thrown. Therefore, it can be used to perform operations such as closing files and releasing system resources.

Throwing our own exceptions:

There may be times when we would like to throw our own exceptions. We can do this by using the keyword throw as follows.

Ex:

```
throw new Throwable _ subclass;
throw new Arithmetic Exception();
throw new Number Format Exception();
```

Program: Exception handling using user-defined exception

```
import Java. util.*;
class MyException extends Exception
{
 MyException (String inform)
 {
  super(inform);
 }
}
```



```
class Test Except
{
 public static void main( String args[ ])
{
 int x,y;
Scanner s =new Scanner(System.in);
System.out.println("Enter two numbers");
x=s.nextInt();
y=s.nextInt();
try {
if(x<=y)
 {
throw new My Exception (" Negative or Zero Result ");
}
else
{
System. out. println("The difference (x-y)=:"+(x-y));
}
}
catch (MyException e)
{
System.out.println(e);
}
} }-
```

Using exceptions for Debugging:

The exception handling mechanism can be used to hide errors from rest of the program. It is possible that the programmers may misuse this technique for hiding errors rather than debugging

the code. Exception handling mechanism may be effectively used to locate the type and place of errors. Once we identify the errors, we must try to find out why errors occur before we cover them with the exception handlers.

5 MARK QUESTIONS AND ANSWERS

1. Define exception and explain its purpose

Ans. An exception is a condition that is caused by a run time error in the program. When the java interpreter encounters an error such as dividing an integer by zero, it creates an exception object and throws it. If the exception object is not caught and handled properly the interpreter will display an error message and will terminate the program. If we want the program to continue with the execution of the remaining code, then we should try to catch the exception object thrown by the error condition and then display an appropriate message for taking corrective actions.

2. Explain types of errors with suitable examples

Ans. Errors are the mistakes that can make the program to go wrong. An error may produce an incorrect output or may terminate the execution of the program abruptly or even may cause the system to crash.

Errors are classified into two categories, namely

i. Compile-Time Errors: When the program is compiled, the java compiler will detect and displays all the syntax errors. Such errors are called compile Time errors. When the compiler displays an error. It will not create .class file.

```
class Sample
{
public Static void main (String args [ ])
{
System. out. println ("Hello")
}
}
```

In above program, the semicolon (;) is missing at the end of point statement. The java compiler displays an appropriate message to the user.

ii. **Run-Time Errors:** When the program is successfully compiled, a .class file is created. Then Java interpreter interprets the class file. During running, the program may produce wrong results

due to wrong logic. Such errors are called runtime errors.

```
class RunError
{
  public static void main (String args[ ])
  {
   int a=10;
    int b=5;
    int c=5;
      int x =a / (b-c);
    System. out. println (" The value of x is" +x);
  }
}
```

3. List the compile time errors

Ans. The most of the compile-time errors are due to typing mistakes. Typographical errors are hard to find. Such common errors are listed below.

1. Missing semicolons.
2. Missing brackets in the class methods.
3. Misspelling of identifiers and keywords.
4. Missing double quotes in strings.
5. Use of undeclared variables.
6. Incompatible types in assignments / initializations.
7. Bad references of objects.
8. Use of = in place of == operator, etc.

4. List the run time errors

Ans. The most common runtime errors are:

1. Dividing an integer by zero.
2. Accessing an element that is out of the bounds of an array.
3. Converting invalid string into number.
4. Accessing array elements out of bound.

5. Attempting to use a Negative size for an array and others.

6. Converting invalid string to a number.

7. Trying to illegally change the state of a thread.

8. Attempting to use a negative size for an array.

5. List the Java exceptions

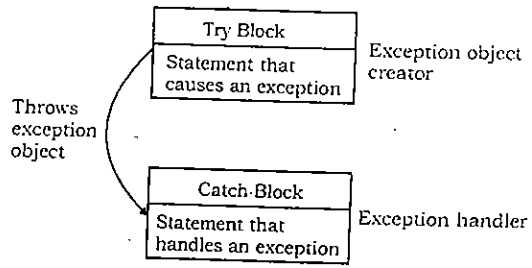
Ans. The common Java exceptions are listed in the following table.

Exception Type	Cause of exception
Arithmetic Exception	Caused by math errors such as division by zero
ArrayIndexOutOfBound Exception	Caused by bad array indexes
Array Store Exception	Caused when a program tries to store the wrong type of data in an array
File Not Found Exception	Caused by an attempt to access a nonexistent file
IO Exception	Caused by general I/O failures
Null Pointer Exception	Caused by referencing a null object
Number Formal Exception	Caused when the conversion between strings and numbers fails
Out Of Memory Exception	Caused when there is not enough memory to allocate an object
Security Exception	Caused when an applet tries to perform an action not allowed by the browsers security settings
Stack Over Flow Exception	Caused when the system runs out of stack space
StringIndex Out OF Bounds Exception	Caused when a program attempts to access a nonexistent character position in a string

6. Explain the syntax of exception handling.

Ans. The figure shown above illustrates exception handling mechanism. The try block contains one or more statements that are likely to cause an error. When an error has generated, Java interpreter creates an exception object and throws which is caught by catch block. The catch block contains one or more statement that are necessary to process the exception. The following programs illustrate how to handle arithmetic exception. The general syntax is


```
try
{
//statement(s) to monitor errors
}
catch (Exceptiontype1 e)
{
block of statements to handle exceptiontype1;
}
catch (Exceptiontype1 e)
{
block of statements to handle exceptiontype1;
}
Finally
{
block of statements to handle exception type;
}
```



7. Explain multiple catch blocks with an example.

Ans. In some cases, more than one exception could be raised by a single piece of code. To handle this type of situation, you can specify two or more catch clauses, each catching a different type of exception. When an exception is thrown, each catch statement is inspected in order, and the first one of whose type matches that of the exception generated is executed. After one catch statement executed, the others are bypassed, and execution continues after the try/catch block. The general form of exception handling code with multiple catch block is:

```
try
{
//statement(s) to monitor errors
}
catch (Exceptiontype1 e)
{
//statements to handle exceptiontype1
}
catch (Exceptiontype2 e)
{
//statements to handle exceptiontype2
}
catch (Exception type N e)
{
//statements to handle exception type N
}
```

You can catch all types of exceptions by setting up you catch block for exceptions of type Exception, like this **catch (Exception e) ;**

Note that Java does not require any processing of the exception at all. We can simply have a catch statement with an empty block to avoid program termination. For example,

```
catch (Exception e)
{ }
or catch (Exception e);
```

8. How many catch blocks can be used with one try block., explain

Ans. Sometimes more than one exception could be raised by a single piece of code. To handle this type of situation, you can specify two or more catch clauses, each catching a different type of exception. When an exception is thrown, each catch statement is inspected in order and the first one whose type matches that of the exception is executed. After one catch statement executes, the others are bypassed and execution continues after the try and catch block. In general many catch blocks can be used.

9. Create a try block that is likely to generate three types of exception and then incorporate necessary catch block to catch and handle them appropriately

Ans.

```
class TestMultiple
{
    public static void main(String args[])
{
     try
{
      int a[]=new int[5];
      a[5]=30/0;
 }
     catch(Arithmetic Exception e)
     {
System. out. println("Task 1 is completed");
     }
     catch(Array Index Out Of Bounds Exception e)
     {
       System. out. println("Task 2 completed");
     }
     catch(Exception e)
     {
       System. out. println("Common task completed");
     }
        System. out. println("Rest of the code...");
 }
}
```

10. Explain the finally block. When and how it is used with a suitable example

Ans. In some cases, more than one exception could be raised by a single piece of code. To handle this type of situation, you can specify two or more catch clauses, each catching a different type of exception. When an exception is thrown, each catch statement is inspected in order, and the first one of whose type matches that of the exception generated is executed. After one catch statement executed, the others are bypassed, and execution continues after the try/catch block. The general form of exception handling code with multiple catch blocks is:

You can catch all types of exceptions by setting up you catch block for exceptions of type Exception, like this

catch (Exception e) ;

Note that Java does not require any processing of the exception at all. We can simply have a catch statement with an empty block to avoid program termination. For example,

catch (Exception e)

{ }

or catch (Exception e);

11. Explain how exception handling mechanism can be used for debugging a program

Ans. The exception handling mechanism can be used to hide errors from rest of the program. It is possible that the programmers may misuse this technique for hiding errors rather than debugging the code. Exception handling mechanism may be effectively used to locate the type and place of errors. Once we identify the errors, we must try to find out why errors occur before we cover them with the exception handlers.

12. Define an exception called "No Match Exception"that is thrown when a string is not equal to " India". Write a program that uses this exception

Ans.

```
import Java. io.*;
class No Match Exception extends Exception
{
public static void main(String args[])
{
   String s = new String();
```



```
   DataInputstream in = new DataInputstream (System. in);
   try
{
     System. out. println("Enter the String:");
     s = in.read Line();
   }
   catch(IO Exception nme)
   {}
   try{
     if(s. equals("India"))
     {
       throw new No Match Exception();
     }
   }
   catch(No Match Exception nmc)
 {
     System. out. println("No Match Exception Caught: " +nme);
   }
 }
}
```

13. Explain how to throw our own exceptions

Ans. There may be times when we would like to throw our own exceptions. We can do this by using the keyword throw as follows.

Ex: throw new Throwable _ subclass;

or

throw new Arithmetic Exception();

throw new Number Format Exception();

14. Write a program to implement " Throwing our own exceptions"

Ans. Exception handling using user-defined exception

```
import Java. util.*;
class My Exception extends Exception
{
 My Exception (String inform)
 {
   super(inform);
  }
}
class Test Except
{
 public static void main( String args[ ])
  {
 int x,y;
Scanner s =new Scanner(System.in);
System.out.println("Enter two numbers");
x=s. nextInt();
y=s.nextInt();
try
{
if(x<=y)
 {
throw new MyException (" Negative or Zero Result ");
}
else
{
System. out. println("The difference (x-y)=:"+(x-y));
}
}
```



```
catch (My Exception e)
{
System. out. println(e);
}
 }
}
```

10 MARK QUESTIONS AND ANSWERS

1. Write a program to illustrate multiple catch blocks

Ans.

```
class Mul Catch
{
public static void main( String args[ ])
{

int arr[]={1,2,3,4};
try
  {
   System. out. println("a[0]=:"+arr[10]);
  }
  catch (Array Index Out Of Bounds Exception e)
  {
   System. out. println(" Exception information is as follows...\n"+e);
  System. out. println(" You are trying to access a location which is not Present");
 }
int a=10;
int b=5;
int c=5;
int x,y;
try
{
```

```
x= a/(b-c); // Exception is generated
}
catch (Arithmetic Exception e)
{
System. out. println ("Division by zero");
}
y= a/(b+c);
System. out. println ("y=" +y);

}
}
```

2. Write a program to illustrate nested try statement.

Ans.

```
/* An example of nested try statements */
class NestTry
{
 public static void main(String args[])
 {
  try
   {
   int a=args. length;
  int b=42/a;
  System. out. println("a="+a);
   try
    {
    if (a==-1)
    a=a/(a-a);
    if (a==2)
```



```
    {
    int c[]={1};
    c[42]=99;
    }
   }catch (Array Index Out Of Bounds Exception e)
  {
 System. out. println("Array index out of bounds"+e);
  }
 }
 catch (Arithmetic Exception e)
 {
 System. out. println("Divide by zero error"+e);
 }
 }
 }
```

3. Write a program for throwing your own exception

Ans.

```
import Java. util.*;
class My Exception extends Exception
{
 My Exception (String inform)
 {
   super(inform);
  }
}
class Test Except {
 public static void main( String args[ ])
 {
 int x,y;
```

```
Scanner s =new Scanner(System.in);
System. out. println("Enter two numbers");
x=s. nextInt();
y=s. nextInt();
try {
if(x<=y)
 {
throw new My Exception (" Negative or Zero Result ");
}
else
{
System. out. println("The difference (x-y)=:"+(x-y));
}
 }
catch (My Exception e)
{
System. out. println(e);
}
}
 }
```


QUESTION PAPERS

DTE SUPER MODEL QUESTION PAPER (WITH ANSWERS)

IV- Semester Diploma in Computer science & Engineering

OOP with Java Programming

Time: **3 Hours** Max Marks: **100**

PART-A

Answer any SIX questions. Each carries 5 marks. **5 x 6 = 30 Marks**

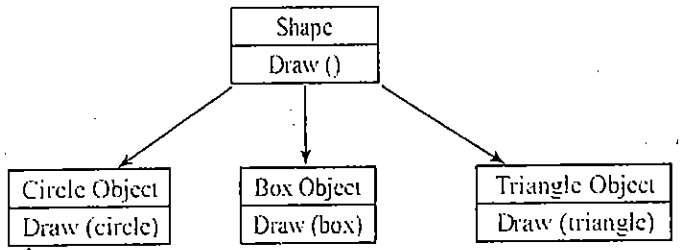
1. List any five major differences between C++ and java.

Ans. Refer Q10 of Unit –I of 5 mark question

2. Distinguish between Inheritance &Polymorphism.

Ans. The mechanism of deriving a new class from an old one is called inheritance. The old class is known as the base class or super class or parent class and the new one is called the sub class or derived class or child class.

Polymorphism is defied as the ability to take different forms. It has two types such as static polymorphism and dynamic polymorphism.



2. Define constructors. List its special properties.

Ans. Refer Q1. of Unit –II of 5 mark question.

3. Compare arrays and vectors.

Ans. Refer Q10. of Unit –II of 5 mark question.

4. Explain how to add a class to a package with an example.

Ans. We know that, the java system packages are organized and used. To create our own packages, we must first declare the name of the package using the package keyword followed by a package name. This must be the first statement in a java source file. The general syntax is

```
package mypackage;          // package declaration
public class Myclass        //class definition
{
// body of class
}
```

Here, mypackage is a package name is considered as part of this package. The above definition must be now saved as MYclass.Java in a directory named mypackage (a subdirectory created within working directory). When the source file compiled Java will create a .class file in the same directory. Notice that the .class file must be located in a directory that has the same name as the package, and this directory must be the subdirectory of the directory where classes that will import the packages are located.

5. Explain the various forms of interface implementation.

Ans. Refer Q3. of Unit –III of 5 mark question.

6. Explain yield (), sleep () and stop () methods of a thread.

Ans. Refer Q6. of Unit –V of 5 mark question.

7. List the java exceptions.

Ans. Refer Q5. of Unit –VI of 5 mark question.

8. Illustrate with an example nested try statement.

Ans. Refer Q2. of Unit –VI of 10 mark question.

PART-B

Answer any SEVEN full questions each carries 10 marks. 10 x 7 = 70 Marks

1. xplain the features of Java.

Ans. Refer Q13. of Unit –I question.

2. Explain class definition with fields and method declaration.

Ans. Refer Q15. of Unit –II question.

3. Define static member. Write a program to illustrate static members.

Ans. A class basically contains two sections. One declares variables and other declares methods. These variables and methods are called instance variables and instance methods. This is because every time the class is instantiated, a new copy of each of them is created. They are accessed using the objects.

We want to define a member that is common-to all the objects and accessed without using a particular object. That is, the member belongs to the class as a whole rather than the objects created from the class. Such members can be defined as follows.

```
static int count;
static int max (int x, int y);
```

The members that are declared static are called static members. Since the members are associated with the class itself rather than individual objects, the static variables and static members are often referred to as class variables and class methods in order to distinguish from their counterparts, instance variables and instance methods.

Static variables are used when we want to have a variable common to all instances of a class. One of the most common examples is to have a variable that could keep a count of how many objects of a class have been created. Remember, java creates only one copy for a static variable which can be used even if the class is never actually instantiated.

Similarly, static methods can be called without using the objects. They are also available for use by other classes. Methods that are of general utility but do not directly affect an instance of that class are usually declared as class methods. Java class libraries contain a large number of class methods.

```
class Sample
{
  int a;
 static int b;
 Sample()
 {
 a=10;
   b=20;
  }
 void display (void)
 {
System.out.println("a= :"+a + "b= : "+b);
 }
static void show(void)
{
 System.out.println(" b= :"+b);
}
}
```

Example:

```
class StatApplication
 {
  public void static main (string args[])
  {
   Sample s=new Sample();
    s.display();
     Sample.show();
 }
  }
```

Note that the static methods are called using class names. In fact, no objects have been created for use. Static methods have several restrictions.

1. They can only call other static methods.
2. They can only access static data.
3. They cannot refer to this or super in any way.

4. Define inheritance. Explain different forms of inheritance.

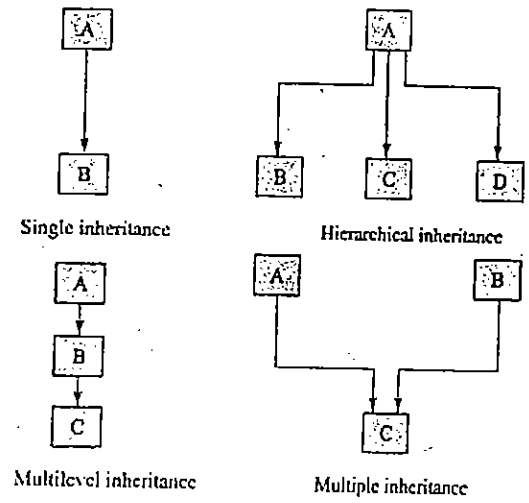
Ans. The mechanism of deriving a new class from an old one is called inheritance. The old class is known as the base class or super class or parent class and the new one is called the sub class or derived class or child class. The inheritance allows subclasses to inherit all the variables and methods of their parent classes. Inheritance may take different forms.

1. Single inheritance: A derived class with only one class is called single inheritance.

2. Multi-level inheritance: It is a process of deriving a class from previously derived class called multi-level inheritance.

3. Hierarchical inheritance: More than one derived classes from a single base class is called hierarchical inheritance.

4. Multiple inheritance: It is the process of deriving a class from a single base class is called multiple inheritance.



Forms of inheritance

5. Write a program to implement interfaces

Ans. Refer Q3. of Unit –III of 10 mark question.

6. Explain Java API packages

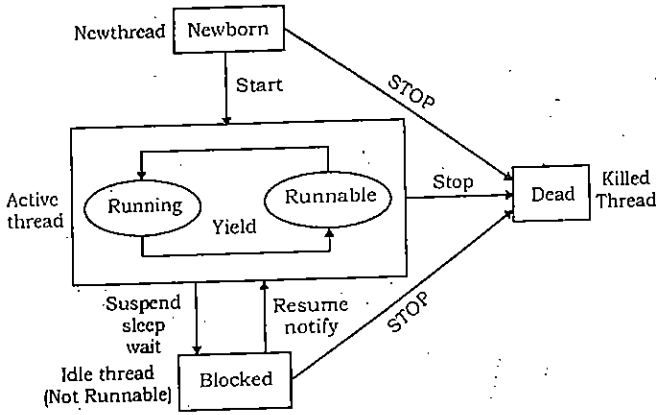
Ans. Refer Q1. of Unit –IV of 10 mark question.

7. Write a Package program to demonstrate basic arithmetic operators

Ans. Refer Q3. of Unit –IV of 10 mark question.

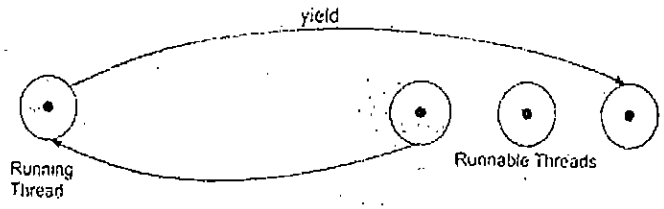
8. Explain the life cycle of thread.

Ans. A thread is a single flow of execution with in a program. During the life time of a thread, it may enter many states which as follows.



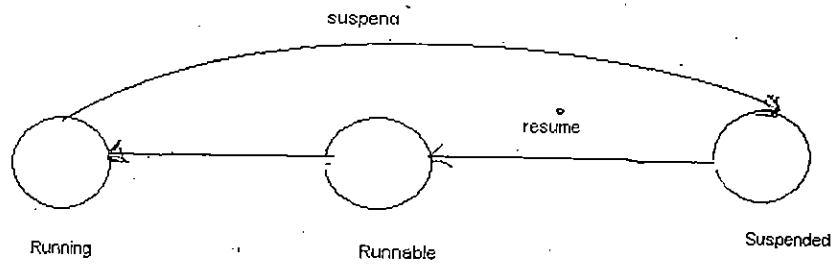
1. New born state: When we create a thread object, the thread is born and is said to be new born state. At this state, we can make the thread running by using start ( ) method or we can kill the thread by using stop ( ) method.

2. Runnable state: When a thread is ready for execution and is waiting for the CPU, then the thread is said to be in Runnable state. When we want a thread to relinquish control to another thread, then we can make use of yield ( ) method is shown below.

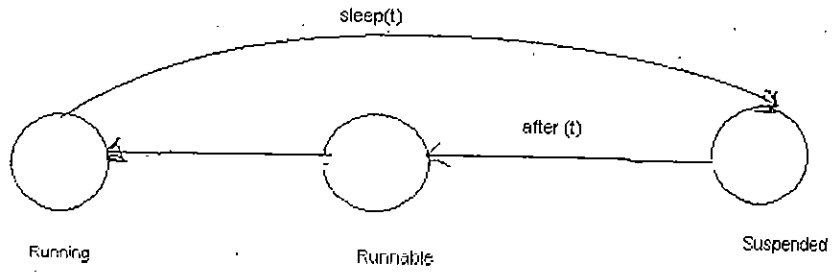


3. Running state: When a thread is being executed by CPU, then the thread is said to be running state. A thread may relinquish control when it finishes execution or preempted by higher priority thread. A running thread may relinquish its control in one of the following situations.

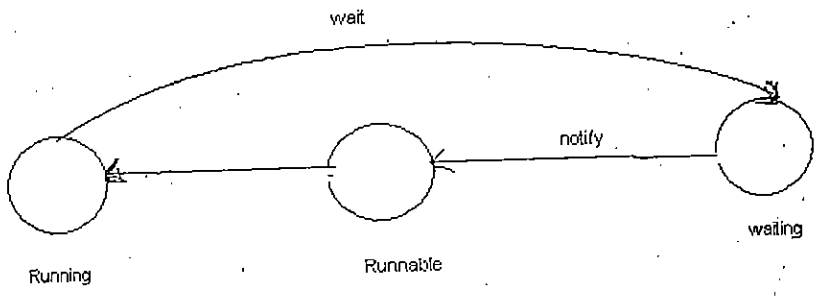
a. It has been suspended using suspend () method. A suspended thread can be reviewed bt using the resume () method. These approaches useful when we want to to suspend a thread for some time due to certain reasons, but do not want to kill it.



b. We can put the thread to sleep for a specific amount of time period using the method sleep(time) where time is in milliseconds. This means that the thread is out of the queue during this time period.



c. It has been told to wait until some event occurs. This is done using the wait () method. The thread can be scheduled to run using the notify () method.



4. Blocked state: A thread can be blocked by using suspend ( ), sleep ( ) and wait (.) method. It prevents the thread from entering into the runnable state and thus running state. When we want to block the thread for certain reason then we make use of suspend ( ) method. We can revoke the suspend ( ) method by using **resume** ( ) method. When we want to block the thread until some event occurs then we make use of wait ( ) method by using notify ( ) method.

When we want to block the thread for specified time period then we make use of sleep ( ) method. The Thread will reenter into runnable state as soon as the time period is elapsed.

5. Dead state: When a thread finishes its execution, it enters into Dead state. We can also kill the thread by using stop ( ) method.

9. Explain with an example thread creation by implementing runnable interface.

Ans. class A implements Runnable

```
{
   public void run( )
    {
       for(int i=1;i<=10;i++)
           {
```



```
                System.out.println ("From Thread A  : i="+i);
            }
                System.out.println ("Exit from Thread A");
    }
}

  class ThreadRun
       {
    public static void main (String args[ ] )
    {
        A runnable =new A();
        Thread threadA= new Thread(runnable);
        threadA.start();
        System.out.println("End of the main Thread");
}
}
```

10. Write a program for throwing your own exception.

Ans. Refer Q3. of Unit –VI of 10 mark question.

SUPER MODEL QUESTION PAPER (WITH ANSWERS)

IV- Semester Diploma in Computer science & Engineering

OOP with Java Programming

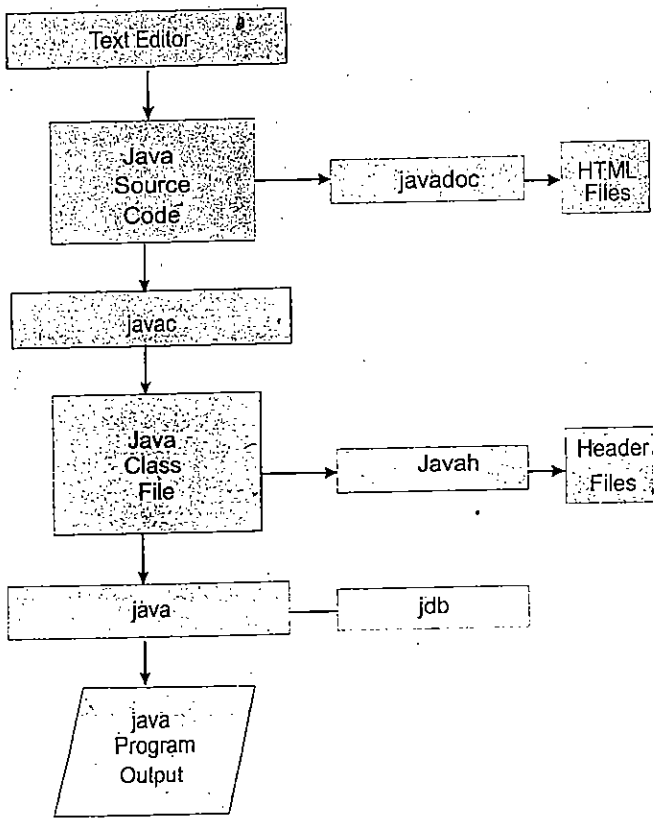
Time: **3 Hours** Max Marks: **100**

PART-A

Answer any SIX questions. Each carries 5 marks. **5 x 6 = 30 Marks**

1. Explain java run time environment.

Ans. Java development kit comes with a collection of tools that are used for developing and running java programs. They include:



1. Applet viewer: used to run java applets (without using a java compatible browser) the applet viewer creates a window in which the applet can be viewed. It provides complete support for all applet functions, including networking and multimedia capabilities

2. Java: java interpreter, which runs applets and applications by reading and interpreting code files.

3. Javac: the java compiler which translates java source code to byte code files.

4. Javadoc: Basically javadoc reads through your source code file and creates HTML file that document your packages and classes, including methods and data fields.

5. Javah: the header file generation tool, javah is used to produce the C files required to develop native methods. It produces both header and source files. The header files contain c language definitions that map to java class definitions. The source files contain the stubs for C functions that map to class methods.

6. Javap: the javap tool is java's disassembler, which is a program that can take a byte code (CLASS) file and change it into a description of the original source code

7. JDB: the java debugger is used to monitor the execution of java programs in support of debugging and test activities. The java debugger jdb enables you to debug your java classes. The java debugger is a command line debugger. You can use the jdb to set breakpoints, inspect objects and variables, and monitor threads.

2. Define the following. (a) Data abstraction, (b) Data encapsulation.

Ans. (a) Data abstraction refers to implementing the data without its background details. It provides the data hiding feature in java.

(b) The wrapping of data fields and methods together as a single unit is called data encapsulation. It provides a way of gathering fields and its corresponding methods together as a single unit based on its behavior.

3. Compare and contrast overloading and overriding methods.

Ans.

Sl. No	Method overloading	Method overriding
1.	The methods have the same name, different parameter lists and different definitions are called the method overloading.	An object to respond to the same method but have different behavior when that method is called. A method in the subclass that has the same name, same arguments and the same return type as a method in the super class. This is known as method overriding.
2.	A family of same method can be created with different arguments.	A family of same method can be created with same arguments.
3.	Static binding	Dynamic binding.

4. Define final and class and finalize () methods.

Ans. A constructor method is used to initialize an object when it is declared. This process known as initialization. Similarly, java supports a concept called finalization, which is just opposite to initialization. We know that java run-time is an automatic garbage collection system. It automatically frees up the memory resources used by the objects. The garbage collector cannot free these resources. In order to free these resources we must use a finilizer method. This is similar to destructor in C++.

The finalizer method is simply finalize () and can be added to any class. Java calls that method whenever it is about to reclaim the space for that object. The finalize method should explicitly defines the tasks to be performed.

```
protected void finalize ()
{
.....
Abstract void draw ();
.....
}
```

5. Define package. List java API packages.

Ans. Refer Q1. of Unit –IV of 5 mark question.

6. Differentiate between class and interface.

Ans.

Sl. No	Class	Interface
1.	It can have abstract and non-abstract methods	It can have only abstract methods
2.	It doesn't support multiple inheritances.	It supports multiple inheritances.
3.	It can have final, non-final, static and non-static variables.	It has only static and final variables.
4.	It can have static methods, main method and constructor	It can not have static methods, main method or constructor
5.	It can provide the implementation of interface.	It can not provide the implementation of abstract class
6.	The abstract keyword is used to declare abstract class	The interface keyword is used to declare interface.

7. Explain how to stop and block a thread.

Ans. Refer Q4. of Unit –V of 5 mark question.

8. List the compile time errors.

Ans. The compile-Time error could be of the following:

Most of the compile-time errors are due to typing mistakes. Typographical errors are hard to find.

1. Missing semicolons.
2. Missing brackets in the class methods.
3. Misspelling of identifiers and keywords.
4. Missing double quotes in strings.
5. Use of undeclared variables.
6. Incompatible types in assignments / initializations.
7. Bad references of objects.
8. Use of = in place of == operator, etc.

9. Explain multiple catch statements with an example.

Ans. In some cases, more than one exception could be raised by a single piece of code. To handle this type of situation, you can specify two or more catch clauses, each catching a different type of

exception. When an exception is thrown, each catch statement is inspected in order, and the first one of whose type matches that of the exception generated is executed. After one catch statement executed, the others are bypassed, and execution continues after the try/catch block. The general form of exception handling code with multiple catch block is:

You can catch all types of exceptions by setting up you catch block for exceptions of type Exception, like this **catch (Exception e);**

Note that java does not require any processing of the exception at all. We can simply have a catch statement with an empty block to avoid program termination. For example,

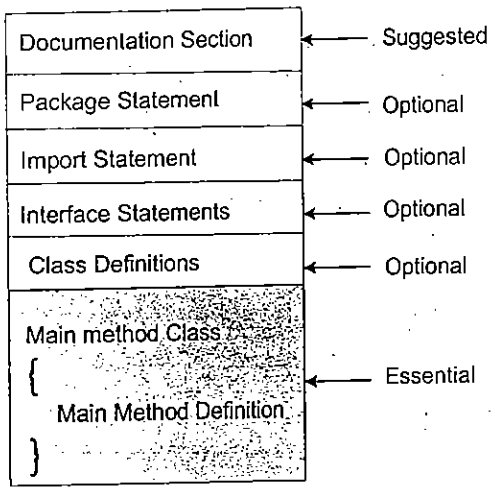
catch (Exception e) {} or catch (Exception e);

PART-B

Answer any SEVEN full questions each carries 10 marks. **10 x 7 = 70 Marks**

1. Explain the java program structure with a diagram

Ans. A java program may contain many classes of which only one class contain a main () method definition. Classes contain data members and methods that operate on the data members. To write java program we first define classes and then put them together. Java program may contain one or more sections as shown in the figure below:



1. Documentation section: The documentation section comprises a set of comment lines giving the name of the program the other details, which the programmer would like to refer to a later

time. Java supports three types of comments line, block, and documentation. The first two, line and block are used to add notes to your code that other developers can read as they are viewing your source code.

The documentation comment is

```
/* This is
* a java program documentation
* comment and ends in the next line.
*/
```

2. Package declaration: If included the package declaration must be the first statement in the file. The package keyword is followed by a package name.

3. Import statements: Import statements are used to access the classes and/or interface defined in packages of java standard library (JSL).

4. Interface statement: An interface is like a class but includes just method declarations, not their definitions. This is an optional section and is used only when we wish to implement the multiple inheritance concept of java

5. Class definition: A java program may contain multiple class definitions. Classes are the primary and essential elements of a java program. These classes are used to map the real world objects. The number of classes mainly depends on the complexity of a program.

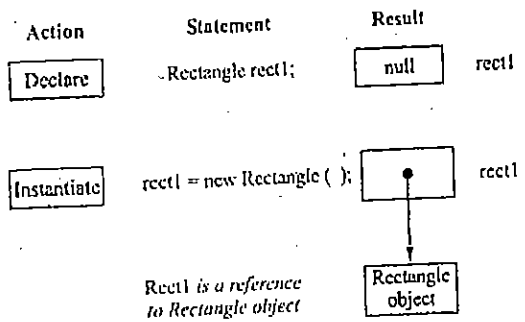
6. Class containing main () method: Since every java standalone application requires a main ( ) method as its starting point this class is the essential part of the java program. A simple java program may contain only this part. The main method creates objects of various classes and establishes the communication between them.

2. Discuss how to create objects in java.

Ans. An object in java essentially a block of memory that contains space to store all the instance variables. Creating an object is also referred to as instantiating an object.

Object in java are created using the new operator. The new operator creates an object of the specified class and returns a reference to that object.

```
Rectangle rect1;      // declare an object
rect1=new Rectangle ();
```



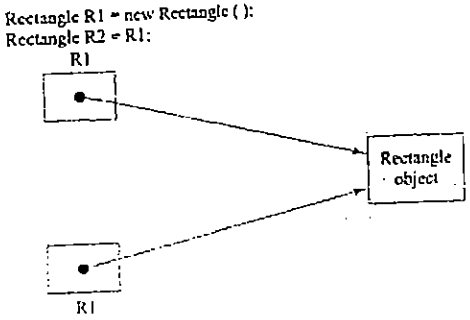
- Both statements can be combined as shown below. In this method Rectangle acts as a default constructor. We can also create number of objects of Rectangle.

```
Rectangle rect1= new Rectangle ();

Rectangle rect1= new Rectangle ();

Rectangle rect2= new Rectangle ();
```

It is important to understand that each object has its own copy of the instance variables of its class. This means that any changes to the variables of one object have no effect on the variables of another. It is also possible to create two or more references to the same object.



Both R1 and R2 refer to the same object.

3. Write a program to illustrate the method overriding.

Ans. An object to respond to the same method but have different behavior when that method is called. That means, we should override the method defined in the super class. This is possible by defining a method in the subclass that has the same name, same arguments and the same return type as a method in the super class. Then, when that method is called, the method defined in the subclass is invoked and executed instead of the one in the super class. This is known as method overriding.

```
class Super
{
int x;
Super(int x)
{
this.x=x;
}
void display()
{
System.out.println("Super of x="+x);
}
}
class Sub
{
int y;
Super (int x, int y)
{
Super(x);
this.y=y;
}
void display()
{
```

```
System.out.println("Super of x="+x);
System.out.println("Sub of y="+y);
}
}
class OverrideTest
{
Public static void main (String args[])
{
Sub s= new Sub(100,200);
s.display();
}
}
```

4. Write a program to illustrate wrapper classes.

Ans. Refer Q22. of Unit –II question

5. With an example explain how to support multiple inheritance.

Ans. Java does not support multiple inheritance. It provides a method called interface to implement multiple interface. An interface is basically a kind of class. It contains methods and variables. The general form of interface definition is:

```
interface A
{
        int x=100;
       void display_A();
        }

 interface B
{
        int y=200;
       void display_B();
       }
```



```
class C implements A,B
        {
      public void display_A()
        {
          System.out.println("x="+x);
        }
     public void display_B()
        {
        System.out.println("y="+y);
        }
       }
class Impl_inface
 {
    public static void main(String args[])
        {
        C obj = new  C();
        obj.display_A();
        obj.display_B ();
        }
  }
```

6. Write a program to use built in packages to calculate square root of a number.

Ans.

```
import java.lang.*;
public class MathDemo {
  public static void main(String[] args) {
  // get two double numbers numbers
  double x = 9;
  double y = 25;
```

```
// print the square root of these doubles
System.out.println("Math.sqrt(" + x + ")=" + Math.sqrt(x));
System.out.println("Math.sqrt(" + y + ")=" + Math.sqrt(y));
 }
}
```

7. Write a program to create threads by extending the thread class.

Ans. Refer Q2. of Unit –V of 10 mark question.

8. Write a note on inter-thread communication.

Ans. Refer Q4. of Unit –V of 10 mark question.

9. Write a program to set priorities for threads.

Ans. Refer Q6. of Unit –V of 10 mark question.

10. Write a program to implement multiple catch blocks.

Ans. Refer Q1. of Unit –VI of 10 mark question.

SUPER MODEL PRACTICE QUESTION PAPER (WITH ANSWERS)

IV- Semester Diploma in Computer science & Engineering

OOP with Java Programming

Time: **3 Hours** Max Marks: **100**

PART-A

Answer any SIX questions. Each carries 5 marks. **5 x 6 = 30 Marks**

1. Distinguished between objects and classes.
2. List any five advantages of OOP.
3. Write visibility controls used in java.
4. Explain any five string methods.
5. Write the general syntax of creating an interface and explain
6. Explain static import and how is it useful.
7. Define synchronization? When do we use it?
8. Define exception and explain its purpose.
9. List the run time errors.

PART-B

Answer any SEVEN full questions each carries 10 marks. **10 x 7 = 70 Marks**

1. List and explain java statements.
2. Discus object creation in java.
3. With an example explain accessing of class members.
4. Write a program to sort N elements of an array.
5. Explain how to extend interfaces with an example.
6. Explain java API packages.
7. Explain the different methods of creating threads.
8. Write a program to implement yield (), sleep () and stop () methods.
9. Write a program to illustrate nested try statement.
10. Write a program to for throwing your own exception.