III SEM CSE

ANNA UNIVERSITY EXAMIATIONS

OBJECT ORIENTED PROGRAMMING

16 MARKS WITH ANSWERS

UNIT I

1.Explain OOP Principles.

OOP is defined as object oriented programming.

Basic Concepts of OOP

- Class
- Object
- Method
- Inheritance
- Data Abstraction
- Data Encapsulation
- Polymorphism
- Message Passing

Defining the Class:

A class is defined by the user is data type with the template that helps in defining the properties. Once the class type has been defined we can create the variables of that type using declarations that are similar to the basic type declarations. In java instances of the classes which are actual objects

Eg:

```
classclassname [extends superclassname]
{
[fields declarations;]
[methods declaration;]
}
```

Field Declaration

Data is encapsulated in a class by placing data fields inside the body of the class definition. These variables are called as instance variables.

```
Class Rectangle
{
int length;
int width;
}
```

Method Declaration

A Class with only data fields has no life, we must add methods for manipulating the data contained in the class. Methods are declared inside the class immediate after the instance variables declaration.

Eg:

```
class Rectangle
{
int length; //instance variables
int width;
Void ge
tData(int x, int y) // Method Declartion
```

```
{
Length =x;
Width = y;
}
}
```

Creating the objects:

An object in java essentially a block of memory, which contains space to store all the instance variables. Creating an object is also referred as instantiating an object. Object in java are created using the new operator.

Eg:

```
Rectangle rec1;
// Declare the object
Rec1 = new Rectangle //instantiate the object
```

The above statements can also be combined as follows

```
Rectangle rec1 = new Rectangle;
```

Methods:

Methods are similar to functions or procedures that are available in other programming languages.

Difference B/w methods and functions

Difference b/w method and function is method declared inside class, function can be declared anywhere inside are outside class

Writing methods in java

if we had to repeatedly output a header such as:

```
System.out.println("GKMCET");
System.out.println ("Allapakkam");
System.out.println ("Meppedu Road");
```

We could put it all in a method like this:

```
public
static void printHeader()
{
System.out.println("GKMCET");
System.out.println("Allapakkam");
System.out.println("Meppedu Road");
}
```

Inheritance:

Inheritance is the process of forming a new class from an existing class or base class. The base class is also known as parent class or super class, the new class that is formed is called derived class.

Derived class is also known as a child class or sub class. Inheritance helps in reducing the overall code size(reusability) of the program, which is an important concept in object oriented programming.

Data Abstraction:

Data abstraction increases the power of programming language by creating user defined data types. Data abstraction also represents the needed information in the program without presenting the details.

Data Encapsulation:

Data encapsulation combines data and functions into a single unit called class. When using Data Encapsulation, data is not accessed directly; it is only accessible through the

functions present inside the class. Data encapsulation enables the important concept of data hiding possible.

Polymorphism:

Polymorphism allows routines to use variables of different types at different times. An operator or function can be given different meanings or functions. The ability of objects to respond differently to the same message or function call.

Message passing:

A message passing system provides primitives for sending and receiving messages using objects. These primitives may by either synchronous or asynchronous or both.

2. Explain the features of java programming language ?

Simple :

Java is Easy to write and more readable

Java has a concise, cohesive set of features that makes it easy to learn and use.

Secure :

Java program cannot harm other system thus making it secure.

Java provides a secure means of creating Internet applications.

Java provides secure way to access web applications.

Portable :

Java programs can execute in any environment for which there is a Java run-time system.(JVM)

Java programs can be run on any platform (Linux,Window,Mac)

Java programs can be transferred over world wide web (e.g applets)

Object-oriented :

Java programming is object-oriented programming language.

Like C++ java provides most of the object oriented features.

Java is pure OOP. Language. (while C++ is semi object oriented)

Robust :

Java encourages error-free programming by being strictly typed and performing run-time checks.

Multithreaded :

Java provides integrated support for multithreaded programming.

Architecture-neutral :

Java is not tied to a specific machine or operating system architecture.

Machine Independent i.e Java is independent of hardware .

Interpreted :

Java supports cross-platform code through the use of Java bytecode.

Bytecode can be interpreted on any platform by JVM.

High performance :

Bytecodes are highly optimized.

JVM can executed them much faster .

Distributed :

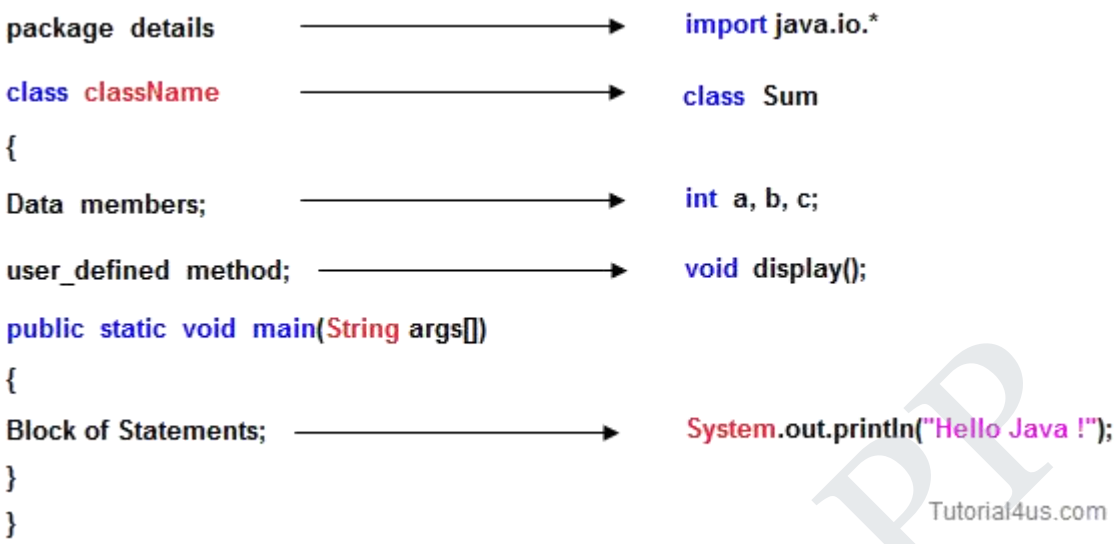
Java was designed with the distributed environment.

Java can be transmit,run over internet.

Dynamic :

Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time.

3. Describe the Structure of Java Program ?



A **package** is a collection of classes, interfaces and sub-packages. A sub package contains collection of classes, interfaces and sub-sub packages etc. java.lang.*; package is imported by default and this package is known as default package.

Class is keyword used for developing user defined data type and every java program must start with a concept of class.

"ClassName" represent a java valid variable name treated as a name of the class each and every class name in java is treated as user-defined data type.

Data member represents either instance or static they will be selected based on the name of the class.

User-defined methods represents either instance or static they are meant for performing the operations either once or each and every time.

Each and every java program starts execution from the main() method. And hence main() method is known as program driver.

Since main() method of java is not returning any value and hence its return type must be void.

Since main() method of java executes only once throughout the java program execution and hence its nature must be static.

Since main() method must be accessed by every java programmer and hence whose access specifier must be public.

Each and every main() method of java must take array of objects of String.

Block of statements represents set of executable statements which are in term calling user-defined methods are containing business-logic.

The file naming conversion in the java programming is that which-ever class is containing main() method, that class name must be given as a file name with an extension .java.

4. Explain the concept of class in java?

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

fields

methods

constructors

blocks

nested class and interface

Syntax to declare a class:

class <class_name>
{
field;
method;
}

Instance variable in Java

A variable which is created inside the class but outside the method, is known as instance variable. Instance variable doesn't get memory at compile time. It gets memory at run time when object(instance) is created. That is why, it is known as instance variable.

Method in Java

In java, a method is like function i.e. used to expose behavior of an object.

Advantage of Method

Code Reusability

Code Optimization

new keyword in Java

The new keyword is used to allocate memory at run time. All objects get memory in Heap memory area.

```
class Student{
int id;//field or data member or instance variable
String name;
public static void main(String args[]){
Student s1=new Student();//creating an object of Student
System.out.println(s1.id);//accessing member through reference variable
System.out.println(s1.name);
}
}
```

5. Explain the concept of objects and how to create objects in java ?

An entity that has state and behavior is known as an object

An object has three characteristics:

state: represents data (value) of an object.

behavior: represents the behavior (functionality) of an object

identity: Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.

Object is an instance of a class. Class is a template or blueprint from which objects

are created. So object is the instance(result) of a class.

Object Definitions:

Object is *a real world entity*.

Object is *a run time entity*.

Object is *an entity which has state and behavior*.

Object is *an instance of a class*.

Creating Objects

Using new keyword is the most basic way to create an object. This is the most Common way to create an object in java.

```
public classNewKeywordExample
{
String name = "hello";
public static void main(String[] args)
        {
        NewKeywordExampleobj = new NewKeywordExample();
        System.out.println(obj.name);
        }
}
```

6. Explain Constructors with examples.

Constructor:

Constructors are the method. It is used to give initial values to instance variables of the objects. Itis automatically called at the time of object creation.

Syntax :

```
Constructor name(arguments)
{
Statements;
Initial
values;
}
```

Rules:

A constructor has the same name as the class.
A class can have more than one constructor.
A constructor can take zero ,one ,or more parameters.
A constructor has no return value.
A construc
tor is always called with the new operator.
Example program:

```
Class complex:
{
Float rp;
Float ip;
Complex(float x,float y) //Defining Constructor
{
Rp=x;
Ip=y;
```

```
}
Void print()
{
System.out.println("Real part"=rp);
System.out.println("Imaginary part"+ip);
}
Void sum(complex a,complex b)
{
Rp=a.rp+b.rp;
Ip=a.ip=b.ip;
}
}
Class demo
{
Public static void main(str
ingargs[]);
{
Complex a,b,c;
a=new complex(5,6);
//Calling Constructor
b=new complex(8,9);
c=new complex();
c.sum(a,b);
c.print();
}
}
```

7. Explain Packages in detail.

Packages

Packages are java's way of grouping a variety of classes and/or interfaces together. Packages are container for the classes.

- It is the header file in c++.
- It is stored in a hierarichical manner.
- If we want to use the packages in a class, we want to import it.

The two types of packages are:

- System package.
- User defined package.

Uses of Packages:

Packages reduce the complexity of the software because a large number of classes can be grouped into a limited number of packages.

We can create classes with same name in different packages.Using packages we can hide classes.

We may like to use many of the classes contained in a package.it can be achieved by

Import packagename .classname

OR

Import packagename.*

Creating the package

To create our own packages
packagefirstpackage;// package declaration
public class FirstClass //class definition
{.....
(body of class)
.....}
The file is saved as FirstClass.java,located at firstpackage directory. when it is compiled .class file will be created in same ditectory.

Access a package

The import statement can be used to search a list of packages for a particular class. The general form of import statement for searching class is a as follows.

Import package1 [.package2] [.package3].classname;
Using the package
package package1 ;
public class classA
{
public void displayA()
{
System.out.println("class A");
}
}

Adding a class to package

Define the class and make it public
Place the package statement
Package P1
Before the class definition as follows
Package p1;
Public class B
Package name
{
// body of B
}
Example:
Package college
Class student
{
intregno;
String name;
Student(intr,stringna);
{
Regno=r
Name=na
}
Public void print()
{
System.out.println("Regno"+regno);
System.out.println("Name"+name);
}
}

```
Package Course
Class engineering
{
Intregno;
String branch;
String year;
Engineering(int r, string br, string yr)
{
Regno=br;
Branch=br;
Year=yr;
}
public void print()
{
Systems.out.println("Regno"+regno);
Systems.out.println("
Branch"+branch);
Systems.out.println("Year"+year);
}
}
import college.*;
import course.*;
Class demo
{
Public static void main(string args[])
{
Student sl=new Engineering(1,"CSE","III
Year");
Sl.print();
El.print();
}
}
```

8. What is Array?.How to declare array?Discuss the methods under Array Class.

An array is a data structure that stores a collection of values of the same type. You access each individual value through an integer index.

Array Name [index]

Integer constant, variable, or expression

For Instance we can definean array name salary to represent a set of salaries of a group of employees. A particular value is indicated by writing a number called index in brackets after the array name.

salary [10]

it represents the salary of the 10^{th} employee.

Types of arrays

One dimensional arrays

Two dimensional arrays

One Dimensional arrays

A list of items can be given one variable name using only one subscript and such a variable is called single - subscripted of one dimensional array. The subscript can also start from 0. ie x[0]. If we want to represent a set of five numbers, say (35,40,20,57,19) by an array variable number, then we have to create the variable number as follows

int number [ ] = new int [5];

The value to the array elements can be assigned as follows

Number

[0] =35;

Number [1] =40;

Number [2] =20;

Number [3] =57;

Number [4] =19;

This would cause the array number to store the values shown as follows;

Creating an array

Declaring the array

Creating memory locations

Putting values into the memory locations.

Array in java can be declared in two forms

Form 1

typearrayname [ ];

Form 2 type [ ] arrayname;

Creation of arrays

arrayname = new type [size];

Eg;

number = new int [5] ;

average = new float[10];

it is alsopossible to combine declaration and creation.

int number [ ] = new int [5];

Initialization of arrays

The final step is to put values into the array created. This process is known as initialization using the array subscripts as shown below.

arrayname[subscript] = value ;

Eg

number[0] = 15;

we can also initialize by following way

typearrayname [ ] = { list of values }

Array Length

All array store the allocated size in an variable named length. We can obtain the length of array a using a.length

Eg:

intasize = a.length;

Two Dimensional arrary:

Usage :

IntmyArray [ ] [ ];

myArray= new int [3] [4];

OR

initmyArray [ ] [ ] = new int [3][4]

This creates a table that can store 12 integer values, four across and three down.

Strings:

Series of characters represents a string, and easiest way to represent the string is array

Eg:

Char charArray [ ] = new char [2] ;

charArray[0] = 'j' ;

charArray[1] = 'a' ;

String can be declared and created in the following way

stringstringname;

stringname = new string ("string");

Operations on string

int m = stringname.length() //will return the length of the string

string city = "New" + "Delhi"// will return New Delhi

9. Explain in detail how to create methods in java ?

A Java method is a collection of statements that are grouped together to perform an operation

Creating Method

Considering the following example to explain the syntax of a method

Syntax

```
public static intmethodName(int a, int b) {
// body
}
```

Here,

public static − modifier

int − return type

methodName − name of the method

a, b − formal parameters

int a, int b − list of parameters

Method definition consists of a method header and a method body.

Syntax

```
modifierreturnTypenameOfMethod (Parameter List) {
// method body
}
```

The syntax shown above includes −

modifier − It defines the access type of the method and it is optional to use.

returnType − Method may return a value.

nameOfMethod − This is the method name. The method signature consists of the method name and the parameter list.

Parameter List − The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.

method body − The method body defines what the method does with the statements.

Method Calling

For using a method, it should be called. There are two ways in which a method is called i.e., method returns a value or returning nothing (no return value).

When a program invokes a method, the program control gets transferred to the called method. This called method then returns control to the caller in two conditions, when the return statement is executed.it reaches the method ending closing brace

Following is the example to demonstrate how to define a method and how to call it

Example

```
publicclassExampleMinNumber{
publicstaticvoid main(String[]args){
int a =11;
int b =6;
int c =minFunction(a, b);
System.out.println("Minimum Value = "+ c);
}
/** returns the minimum of two numbers */
publicstaticintminFunction(int n1,int n2){
int min;
if(n1 > n2)
```

```
min= n2;
else
min= n1;
return min;
}
}
```

10. Explain static variable and static methods in java?

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

The static can be:

variable (also known as class variable)

method (also known as class method)

block

nested class

---

1)Java static variable

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

The static variable can be used to refer the common property of all objects (that is not unique for each object)

The static variable gets memory only once in class area at the time of class loading.

Advantage of static variable

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

Example of static variable

```
class Student8{
int rollno;
```

```
String name;
static String college ="ITS";
Student8(int r,String n){
rollno = r;
name = n;
}
void display ()
{
System.out.println(rollno+" "+name+" "+college);
}
public static void main(String args[]){
Student8 s1 = new Student8(111,"Karan");
Student8 s2 = new Student8(222,"Aryan");
s1.display();
s2.display();
}
}
```

2) Java static method

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

A static method belongs to the class rather than object of a class.

A static method can be invoked without the need for creating an instance of a class.

static method can access static data member and can change the value of it.

Example of static method

//Program of changing the common property of all objects(static field).

```
class Student9{
int rollno;
String name;
static String college = "ITS";

static void change(){
college = "BBDIT";
}
Student9(int r, String n){
rollno = r;
```

```
name = n;
}
void display (){System.out.println(rollno+" "+name+" "+college);}
public static void main(String args[]){
Student9.change();

Student9 s1 = new Student9 (111,"K");
Student9 s2 = new Student9 (222,"A");
Student9 s3 = new Student9 (333,"S");

s1.display();
s2.display();
s3.display();
}
}
```

11. Explain the different categories of operators in java ?

Operator in java is a symbol that is used to perform operations.

There are many types of operators in java which are given below:

Unary Operator,

Arithmetic Operator,

Shift Operator,

Relational Operator,

Bitwise Operator,

Logical Operator,

Ternary Operator and

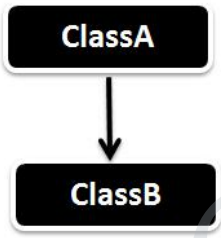
Assignment Operator.

UNIT –II

1.ExplainSingle Inheritance in Java

Single Inheritance is the simple inheritance of all, When a class extends another class(Only one class) then we call it as **Single inheritance.**

The below diagram represents the single inheritance in java where **Class B** extends only one class **Class A.**



Class B will be the **Sub class** and **Class A** will be one and only **Super class**.

Single Inheritance Example

```
publicclassClassA
{
publicvoiddispA()
{
System.out.println("disp() method of ClassA");
}
}
publicclassClassBextendsClassA
{
publicvoiddispB()
{
System.out.println("disp() method of ClassB");
}
publicstaticvoid main(Stringargs[])
{
//Assigning ClassB object to ClassB reference
ClassB b =newClassB();
//call dispA() method of ClassA
```

```
b.dispA();
//call dispB() method of ClassB
b.dispB();
}
}
```

2. How will you achieve multiple inheritance in java ?

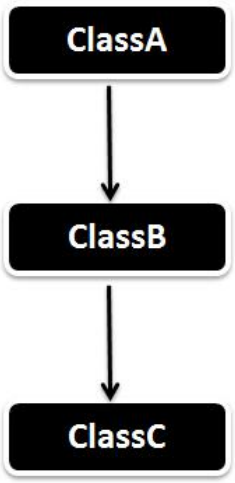
Multiple Inheritance is nothing but **one** class **extending more than** one class.

Multiple Inheritance is basically not supported by many **Object Oriented Programming** languages such as **Java, Small Talk, C# etc.. (C++ Supports Multiple Inheritance).**

As the **Child** class has to manage the dependency of more than one **Parent** class. But you can achieve multiple inheritance in Java using **Interfaces**.

3.Explain Multilevel Inheritance in java ?

In **Multilevel Inheritance** a derived class will be **inheriting a parent class** and as well as the derived class **act as the parent class** to other class. As seen in the below diagram. **ClassB**inherits the property of **ClassA** and again **ClassB** act as a parent for **ClassC**. In Short **ClassA**parent for **ClassB** and **ClassB** parent for **ClassC**.



MultiLevel Inheritance Example

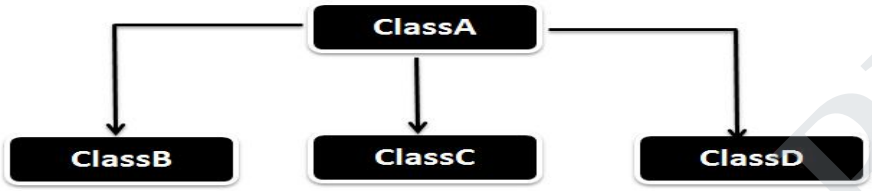
```
public class ClassA
{
public void dispA()
{
System.out.println("disp() method of ClassA");
}
}
public class ClassB extends ClassA
{
public void dispB()
{
System.out.println("disp() method of ClassB");
}
}
public class ClassC extends ClassB
{
public void dispC()
{
System.out.println("disp() method of ClassC");
}
public static void main(String args[])
{
//Assigning ClassC object to ClassC reference
ClassC c = new ClassC();
//call dispA() method of ClassA
c.dispA();
//call dispB() method of ClassB
c.dispB();
//call dispC() method of ClassC
c.dispC();
}
```

}

4.Explain Hierarchical Inheritance in java ?

In **Hierarchical inheritance** one parent class will be inherited by **many** sub classes. As per the below example **ClassA** will be inherited by **ClassB, ClassC** and **ClassD**.

ClassA will be acting as a parent class for **ClassB, ClassC** and **ClassD**.



Hierarchical Inheritance Example

```
public class ClassA
{
public void dispA()
{
System.out.println("disp() method of ClassA");
}
}
public class ClassB extends ClassA
{
public void dispB()
{
System.out.println("disp() method of ClassB");
}
}
public class ClassC extends ClassA
{
public void dispC()
{
System.out.println("disp() method of ClassC");
}
}
public class ClassD extends ClassA
{
public void dispD()
{
System.out.println("disp() method of ClassD");
}
}
```

```
public class HierarchicalInheritanceTest
{
public static void main(String args[])
{
//Assigning ClassB object to ClassB reference
ClassB b = new ClassB();
//call dispB() method of ClassB
b.dispB();
//call dispA() method of ClassA
b.dispA();


//Assigning ClassC object to ClassC reference
ClassC c = new ClassC();
//call dispC() method of ClassC
c.dispC();
//call dispA() method of ClassA
c.dispA();

//Assigning ClassD object to ClassD reference
ClassD d = new ClassD();
//call dispD() method of ClassD
d.dispD();
//call dispA() method of ClassA
d.dispA();
}
}
```

5. Explain abstract class in java with example?

A class that is declared using "**abstract**" keyword is known as abstract class. It can have abstract methods(methods without body) as well as concrete methods (regular methods with body). A normal class(non-abstract class) cannot have abstract methods.

Abstract class declaration

An abstract class outlines the methods but not necessarily implements all the methods.

```
//Declaration using abstract keyword
abstract class A{
//This is abstract method
abstract void myMethod();

//This is concrete method with body
voidanotherMethod(){
//Does something
}
```

}
A class derived from the abstract class must implement all those methods that are declared as abstract in the parent class.

Abstract class cannot be instantiated which means you cannot create the object of it. To use this class, you need to create another class that extends this this class and provides the implementation of abstract methods

If a child does not implement all the abstract methods of abstract parent class, then the child class must need to be declared abstract as well.

Example of Abstract class and method

```
abstract class MyClass{
public void disp(){
System.out.println("Concrete method of parent class");
}
abstract public void disp2();
}

class Demo extends MyClass{
public void disp2()
{
System.out.println("overriding abstract method");
}
public static void main(String args[]){
Demo obj = new Demo();
obj.disp2();
}
}
```

6. How will you define interface in java?

Using the keyword interface, you can fully abstract a class' interface from its implementation. That is, using interface, you can specify what a class must do, but not how it does it.

Interfaces are syntactically similar to classes, but they lack instance variables, and their methods are declared without any body. In practice, this means that you can define interfaces that don't make assumptions about how they are implemented.

Once it is defined, any number of classes can implement an interface. Also, one class can implement any number of interfaces.

To implement an interface, a class must create the complete set of methods defined by the interface. However, each class is free to determine the details of its own implementation.

By providing the interface keyword, Java allows you to fully utilize the "one interface, multiple methods" aspect of polymorphism.

Interfaces are designed to support dynamic method resolution at run time. Normally, in order for a method to be called from one class to another, both classes need to be present at compile time so the Java compiler can check to ensure that the method signatures are compatible

Defining an Interface

An interface is defined much like a class. This is a simplified general form of an interface:

access interface name { return-type method-name1(parameter-list);

return-type method-name2(parameter-list);

type final-varname1 = value;

type final-varname2 = value;

return-type method-nameN(parameter-list);

type final-varnameN = value; }

When no access modifier is included, then default access results, and the interface is only available to other members of the package in which it is declared. When it is declared as public, the interface can be used by any other code

7. Explain with example about implementation of interface ?

Once an interface has been defined, one or more classes can implement that interface.

To implement an interface, include the implements clause in a class definition, and then create the methods defined by the interface.

The general form of a class that includes the implements clause looks like this:

classclassname [extends superclass] [implements interface [,interface...]]

{ // class-body }

If a class implements more than one interface, the interfaces are separated with a comma.

If a class implements two interfaces that declare the same method, then the same method will be used by clients of either interface.

The methods that implement an interface must be declared public. Also, the type signature of the implementing method must match exactly the type signature specified in the interface definition

```
class Client implements Callback {

public void callback(int p)

{ System.out.println("callback called with " + p); }

}
```

8.Explain with example how Interfaces can be extended ?

One interface can inherit another by use of the keyword extends.

The syntax is the same as for inheriting classes.

When a class implements an interface that inherits another interface, it must provide

implementations for all methods defined within the interface inheritance chain.

EXAMPLE

```
interface A {

void meth1();

void meth2();

}

interface B extends A

{ void meth3(); }

MyClass implements B

{

public void meth1()

{

System.out.println("Implement meth1().");

}

public void meth2() { System.out.println("Implement meth2()."); }
```

```
public void meth3() { System.out.println("Implement meth3()."); } }

classIFExtend {

public static void main(String arg[]) {

MyClassob = new MyClass();

ob.meth1();

ob.meth2();

ob.meth3(); }

}
```

9. Explain Final Method and final class in Java with Example

Sometimes you may want to prevent a subclass from overriding a method in your class. To do this, simply add the keyword final at the start of the method declaration in a super class. Any attempt to override a final method will result in a compiler error

```
class base
{
    final void display()
    {
     System.out.println("Base method called");
     }
}

class Derived extends Base
  {
      void display() //cannot override
    {
     System.out.println("Base method called");
    }
  }
   class finalMethod
 {
       public static void main(String[] args)
    {
        Derived d =new Derived();
        d.display();
    }
 }
```

On compiling the above example, it will display an error

Final class

A **final class** can not be inherited/extended.

Java program to demonstrate example of final class.

```
1       importjava.util.*;
2
3       final class Base
4       {
5       public void displayMsg()
6       {
7       System.out.println("I'm displayMsg() in Base class.");
8       }
9       }
10
11      public class FinalClassExample extends Base
12      {
13      public void displayMsg1()
14      {
15      System.out.println("I'm displayMsg1() in Final class.");
16      }
17
18      public static void main(String []s)
19      {
20      FinalClassExample FCE=new FinalClassExample();
21      FCE.displayMsg();
22      FCE.displayMsg1();
23      }
24      }
```

10. Explain Object Cloning in Java with example ?

The **object cloning** is a way to create exact copy of an object. The clone() method of Object class is used to clone an object.

The **java.lang.Cloneable interface** must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates **CloneNotSupportedException**.

The **clone() method** is defined in the Object class. Syntax of the clone() method is as follows

Example of clone() method (Object cloning)

```
class Student18 implements Cloneable
{
int rollno;
String name;
Student18(int rollno,String name){
this.rollno=rollno;
```

```
this.name=name;
}

public Object clone()throws CloneNotSupportedException
{
return super.clone();
}

public static void main(String args[]){
try{
Student18 s1=new Student18(101,"amit");
Student18 s2=(Student18)s1.clone();
System.out.println(s1.rollno+" "+s1.name);
System.out.println(s2.rollno+" "+s2.name);
}catch(CloneNotSupportedException c){}
}
}
```

11.Explain Inner class in java with example ?

A non-static class that is created inside a class but outside a method is called member inner class.

Syntax:

```
class Outer{
//code
class Inner{
//code
}
}
```

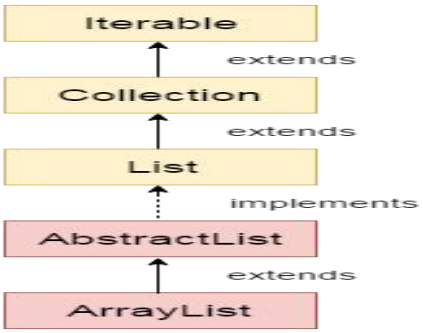
Java Member inner class example

In this example, we are creating msg() method in member inner class that is accessing the private data member of outer class.

```
class TestMemberOuter1{
private int data=30;
class Inner{
void msg(){System.out.println("data is "+data);}
}
```

```
public static void main(String args[]){
TestMemberOuter1 obj=new TestMemberOuter1();
TestMemberOuter1.Inner in=obj.new Inner();
in.msg();
}
}
```

12. Explain the methods in array list with example ?



Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.

The important points about Java ArrayList class are:

Java ArrayList class can contain duplicate elements.

Java ArrayList class maintains insertion order.

Java ArrayList class is non synchronized.

Java ArrayList allows random access because array works at the index basis.

In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

Hierarchy of ArrayList class

As shown in above diagram, Java ArrayList class extends AbstractList class which implements List interface. The List interface extends Collection and Iterable interfaces in hierarchical order.

ArrayList class declaration

Let's see the declaration for java.util.ArrayList class.

public class ArrayList<E> **extends** AbstractList<E> **implements** List<E>, Random Access, Cloneable, Serializable

Constructors of Java ArrayList

Constructor	Description
ArrayList()	It is used to build an empty array list.
ArrayList(Collection c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

Methods of Java ArrayList

Method	Description
void add(int index, Object element)	It is used to insert the specified element at the specified position index in a list.
booleanaddAll(Collection c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
void clear()	It is used to remove all of the elements from this list.
intlastIndexOf(Object o)	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.

Object[] toArray(Object[] a)	It is used to return an array containing all of the elements in this list in the correct order.
boolean add(Object o)	It is used to append the specified element to the end of a list.
booleanaddAll(int index, Collection c)	It is used to insert all of the elements in the specified collection into this list, starting at the specified position.
Object clone()	It is used to return a shallow copy of an ArrayList.
intindexOf(Object o)	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
void trimToSize()	It is used to trim the capacity of this ArrayList instance to be the list's current size.

Java ArrayList Example

```
import java.util.*;
class TestCollection1{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();//Creating arraylist
  list.add("Ravi");//Adding object in arraylist
  list.add("Vijay");
  list.add("Ravi");
  list.add("Ajay");
  //Traversing list through Iterator
  Iterator itr=list.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

13. Explain String Class and its methods with examples ?

String is a sequence of characters, for e.g. “Hello” is a string of 5 characters. In java,
string is an immutable object which means it is constant and can cannot be changed once it has been created.

Creating a String

There are two ways to create a String in Java

1. String literal
2. Using new keyword

String literal

In java, Strings can be created like this: Assigning a String literal to a String instance:

```
String str1 ="Welcome";
String str2 ="Welcome";
```

Using New Keyword

```
String str1 = new String("Welcome");
String str2 = new String("Welcome");
```

In this case compiler would create two different object in memory having the same string.

A Simple Java String Example

```
public class Example{
public static void main(String args[]){
        //creating a string by java string literal
        String str = "Beginnersbook";
        chararrch[]={'h','e','l','l','o'};
        //converting char array arrch[] to string str2
        String str2 = new String(arrch);

        //creating another java string str3 by using new keyword
        String str3 = new String("Java String Example");

        //Displaying all the three strings
        System.out.println(str);
        System.out.println(str2);
        System.out.println(str3);
  }
}
```

Java String Methods

Here are the list of the methods available in the Java String class.

1. charcharAt(int index): It returns the character at the specified index. Specified index value should be between 0 to length() -1 both inclusive. It throws IndexOutOfBoundsException if index<0||>= length of String.
2. boolean equals(Object obj): Compares the string with the specified string and returns true if both matches else false.
3. booleanequalsIgnoreCase(String string): It works same as equals method but it doesn't consider the case while comparing strings. It does a case insensitive comparison.
4. intcompareTo(String string): This method compares the two strings based on the Unicode value of each character in the strings.
5. intcompareToIgnoreCase(String string): Same as CompareTo method however it ignores the case during comparison.
6. booleanstartsWith(String prefix, int offset): It checks whether the substring (starting from the specified offset index) is having the specified prefix or not.
7. booleanstartsWith(String prefix): It tests whether the string is having specified prefix, if yes then it returns true else false.
8. booleanendsWith(String suffix): Checks whether the string ends with the specified suffix.
9. inthashCode(): It returns the hash code of the string.
10. intindexOf(intch): Returns the index of first occurrence of the specified character ch in the string.
11. intindexOf(intch, intfromIndex): Same as indexOf method however it starts searching in the string from the specified fromIndex.
12. intlastIndexOf(intch): It returns the last occurrence of the character ch in the string.
13. intlastIndexOf(intch, intfromIndex): Same as lastIndexOf(intch) method, it starts search from fromIndex.
14. intindexOf(String str): This method returns the index of first occurrence of specified substring str.
15. intlastindexOf(String str): Returns the index of last occurrence of string str.
16. String substring(intbeginIndex): It returns the substring of the string. The substring starts with the character at the specified index.
17. String substring(intbeginIndex, intendIndex): Returns the substring. The substring starts with character at beginIndex and ends with the character at endIndex.
18. String concat(String str): Concatenates the specified string "str" at the end of the string.
19. String replace(char oldChar, char newChar): It returns the new updated string after changing all the occurrences of oldChar with the newChar.
20. boolean contains(CharSequence s): It checks whether the string contains the specified sequence of char values. If yes then it returns true else false. It throws NullPointerExceptionof 's' is null.
21. String toUpperCase(Locale locale): Converts the string to upper case string using the rules defined by specified locale.
22. String toUpperCase(): Equivalent to toUpperCase(Locale.getDefault()).

23. public String intern(): This method searches the specified string in the memory pool and if it is found then it returns the reference of it, else it allocates the memory space to the specified string and assign the reference to it.
24. publicbooleanisEmpty(): This method returns true if the given string has 0 length. If the length of the specified Java String is non-zero then it returns false.
25. public static String join(): This method joins the given strings using the specified delimiter and returns the concatenated Java String
26. String replaceFirst(String regex, String replacement): It replaces the first occurrence of substring that fits the given regular expression "regex" with the specified replacement string.
27. String replaceAll(String regex, String replacement): It replaces all the occurrences of substrings that fits the regular expression regex with the replacement string.
28. String[] split(String regex, int limit): It splits the string and returns the array of substrings that matches the given regular expression. limit is a result threshold here.
29. String[] split(String regex): Same as split(String regex, int limit) method however it does not have any threshold limit.
30. String toLowerCase(Locale locale): It converts the string to lower case string using the rules defined by given locale.
31. public static String format(): This method returns a formatted java String
32. String toLowerCase(): Equivalent to toLowerCase(Locale. getDefault()).
33. String trim(): Returns the substring after omitting leading and trailing white spaces from the original string.
34. char[] toCharArray(): Converts the string to a character array.
35. static String copyValueOf(char[] data): It returns a string that contains the characters of the specified character array.
36. static String copyValueOf(char[] data, int offset, int count): Same as above method with two extra arguments – initial offset of subarray and length of subarray.
37. voidgetChars(intsrcBegin, intsrcEnd, char[] dest, intdestBegin): It copies the characters of **src**array to the **dest** array. Only the specified range is being copied(srcBegin to srcEnd) to the destsubarray(starting fromdestBegin).
38. static String valueOf(): This method returns a string representation of passed arguments such as int, long, float, double, char and char array.
39. booleancontentEquals(StringBuffersb): It compares the string to the specified string buffer.
40. booleanregionMatches(intsrcoffset, String dest, intdestoffset, intlen): It compares the substring of input to the substring of specified string.
41. booleanregionMatches(booleanignoreCase, int srcoffset, String dest, intdestoffset, intlen): Another variation of regionMatches method with the extra boolean argument to specify whether the comparison is case sensitive or case insensitive.
42. byte[] getBytes(String charsetName): It converts the String into sequence of bytes using the specified charset encoding and returns the array of resulted bytes.
43. byte[] getBytes(): This method is similar to the above method it just uses the default charset encoding for converting the string into sequence of bytes.
44. int length(): It returns the length of a String.
45. boolean matches(String regex): It checks whether the String is matching with the specified regular expression regex.

46. intcodePointAt(int index):It is similar to the charAt method however it returns the Unicode code point value of specified index rather than the character itself.

UNIT III

1.Explain how exception is handled in java?

Exception is an error event that can happen during the execution of a program and disrupts its normal flow

Java being an object oriented programming language, whenever an error occurs while executing a statement, creates an **exception object** and then the normal flow of the program halts.

When the exception occurs in a method, the process of creating the exception object and handing it over to runtime environment is called **"throwing the exception"**..

Once runtime receives the exception object, it tries to find the handler for the exception. Exception Handler is the block of code that can process the exception object

The handler is said to be **"catching the exception"**. If there are no appropriate exception handler found then program terminates printing information about the exception.

Java Exception handling is a framework that is used to handle runtime errors only, compile time errors are not handled by exception handling in java.

Java provides specific keywords for exception handling purposes.

1. **throw** – We know that if any exception occurs, an exception object is getting created and then Java runtime starts processing to handle them. Sometime we might want to generate exception explicitly in our code, for example in a user authentication program we should throw exception to client if the password is null.**throw** keyword is used to throw exception to the runtime to handle it.
2. **throws** – When we are throwing any exception in a method and not handling it, then we need to use **throws** keyword in method signature to let caller program know the exceptions that might be thrown by the method. The caller method might handle these exceptions or propagate it to it's caller method using throws keyword. We can provide multiple exceptions in the throws clause and it can be used with main() method also.
3. **try-catch** – We use try-catch block for exception handling in our code. try is the start of the block and catch is at the end of try block to handle the exceptions. We

can have multiple catch blocks with a try and try-catch block can be nested also. catch block requires a parameter that should be of type Exception.

4. **finally** – finally block is optional and can be used only with try-catch block. Since exception halts the process of execution, we might have some resources open that will not get closed, so we can use finally block. finally block gets executed always, whether exception occurred or not.

```
import java.io.*;

publicclassExcepTest{


publicstaticvoid main(Stringargs[]){

try{

int a[]=newint[2];

System.out.println("Access element three :"+ a[3]);

}catch(ArrayIndexOutOfBoundsException e){

System.out.println("Exception thrown  :"+ e);

}

System.out.println("Out of the block");

}

}
```

Catching Exceptions

A method catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following −

Syntax

```
try {
   // Protected code
} catch (ExceptionName e1) {
   // Catch block
}
```

The code which is prone to exceptions is placed in the try block. When an exception occurs, that exception occurred is handled by catch block associated with it. Every try block should be immediately followed either by a catch block or finally block.

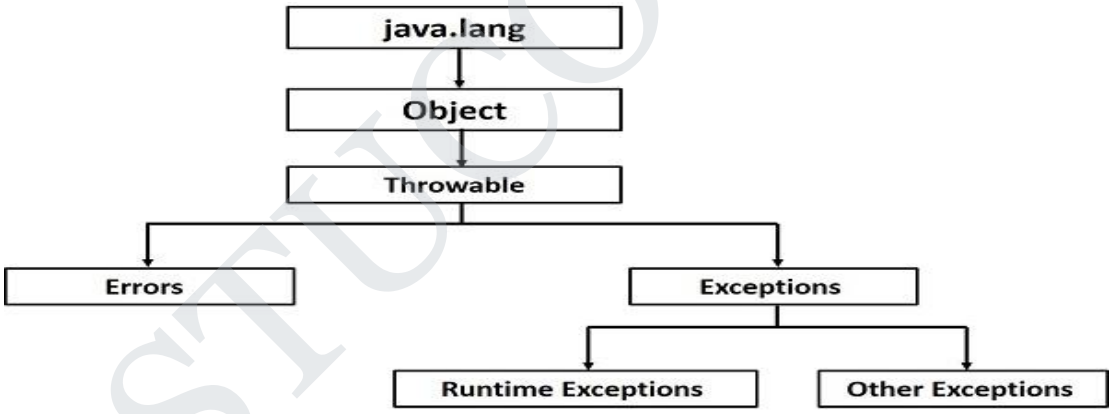
A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in protected code, the catch block (or blocks) that follows the try is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block much as an argument is passed into a method parameter.

2. Explain exception hierarchy in java?

All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class. Other than the exception class there is another subclass called Error which is derived from the Throwable class.

Errors are abnormal conditions that happen in case of severe failures, these are not handled by the Java programs. Errors are generated to indicate errors generated by the runtime environment. Normally, programs cannot recover from errors.

The Exception class has two main subclasses: IOException class and RuntimeException Class.



Exceptions Methods

Following is the list of important methods available in the Throwable class.

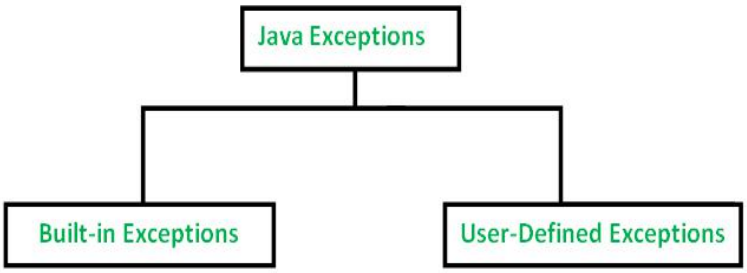
Sr.No.	Method & Description
1	**public String getMessage()** Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor.

2	**public ThrowablegetCause()** Returns the cause of the exception as represented by a Throwable object.
3	**public String toString()** Returns the name of the class concatenated with the result of getMessage().
4	**public void printStackTrace()** Prints the result of toString() along with the stack trace to System.err, the error output stream.
5	**public StackTraceElement [] getStackTrace()** Returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack, and the last element in the array represents the method at the bottom of the call stack.
6	**public ThrowablefillInStackTrace()** Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace.

3. Write a java program to handle division by zero ?

```
classJavaException {
public static void main(String args[]){
try{
int d = 0;
int n =20;
int fraction = n/d;
 }
catch(ArithmeticException e){
System.out.println("In the catch clock due to Exception = "+e);
}
finally{
            System.out.println("Inside the finally block");
}
}
}
```

4.Describe the types of Exception in Java with Examples ?



Built-in exceptions are the exceptions which are available in Java libraries. These exceptions are suitable to explain certain error situations.

Below is the list of important built-in exceptions in Java.

1. **ArithmeticException**
   It is thrown when an exceptional condition has occurred in an arithmetic operation.
2. **ArrayIndexOutOfBoundException**
   It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.
3. **ClassNotFoundException**
   This Exception is raised when we try to access a class whose definition is not found
4. **FileNotFoundException**
   This Exception is raised when a file is not accessible or does not open.
5. **IOException**
   It is thrown when an input-output operation failed or interrupted
6. **InterruptedException**
   It is thrown when a thread is waiting , sleeping , or doing some processing , and it is interrupted.
7. **NoSuchFieldException**
   It is thrown when a class does not contain the field (or variable) specified
8. **NoSuchMethodException**
   It is thrown when accessing a method which is not found.
9. **NullPointerException**
   This exception is raised when referring to the members of a null object. Null represents nothing
10. **NumberFormatException**
    This exception is raised when a method could not convert a string into a numeric format.
11. **RuntimeException**
    This represents any exception which occurs during runtime.
12. **StringIndexOutOfBoundsException**
    It is thrown by String class methods to indicate that an index is either negative than the size of the string

Examples of Built-in Exception:

- **Arithmetic exception**

```
// Java program to demonstrate ArithmeticException
classArithmeticException_Demo
{
```

```
        publicstaticvoidmain(String args[])
        {
           try{
              inta = 30, b = 0;
              intc = a/b;  // cannot divide by zero
              System.out.println ("Result = "+ c);
           }
           catch(ArithmeticException e) {
              System.out.println ("Can't divide a number by 0");
           }
        }
     }
```

- **ArrayIndexOutOfBounds Exception**

```
classArrayIndexOutOfBound_Demo
    {
    publicstaticvoidmain(String args[])
     {
    try{
       inta[] = newint[5];
       a[6] = 9; // accessing 7th element in an array of
                 // size 5
    }
    catch(ArrayIndexOutOfBoundsException e){
       System.out.println ("Array Index is Out Of Bounds");
    }
    }
    }
```

5.How to create User Defined Exception in Java?

User Defined Exception or custom exception is creating your own exception class and throws that exception using 'throw' keyword. This can be done by extending the class Exception.

Example of User defined exception in Java

```
classMyExceptionextendsException{
String str1;
MyException(String str2)
{
        str1=str2;
}
publicStringtoString(){
        return("MyException Occurred: "+str1);
}
}

classExample1{
publicstaticvoid main(Stringargs[]){
```

```
        try{
                System.out.println("Starting of try block");
                // I'm throwing the custom exception using throw
                thrownewMyException("This is My error Message");
        }
        catch(MyExceptionexp){
                System.out.println("Catch Block");
                System.out.println(exp);
        }
}
}
```

6. Explain byte streams and character streams in java ?

The java.io package contains nearly every class you might ever need to perform input and output (I/O) in Java. All these streams represent an input source and an output destination.

Stream

A stream can be defined as a sequence of data. There are two kinds of Streams −

- **InPutStream** − The InputStream is used to read data from a source.
- **OutPutStream** − The OutputStream is used for writing data to a destination.

Byte Streams

Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are, **FileInputStream** and **FileOutputStream**. Following is an example which makes use of these two classes to copy an input file into an output file −

Example

```
import java.io.*;

publicclassCopyFile{

publicstaticvoid main(Stringargs[])throwsIOException{

FileInputStreamin=null;

FileOutputStreamout=null;

try{
```

```
in=newFileInputStream("input.txt");
out=newFileOutputStream("output.txt");
int c;
while((c =in.read())!=-1){
out.write(c);
}
}finally{
if(in!=null){
in.close();
}
if(out!=null){
out.close();
}
}
}
```

Character Streams

Java **Byte** streams are used to perform input and output of 8-bit bytes, whereas Java **Character** streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are, **FileReader** and **FileWriter**.

Example

```
import java.io.*;
public class CopyFile {
public static void main(String args[]) throws IOException {
FileReader in = null;
FileWriter out = null;
try {
```

```
in = new FileReader("input.txt");
out = new FileWriter("output.txt");
int c;
while ((c = in.read()) != -1) {
out.write(c);
        }
     }
finally {
if (in != null) {
in.close();
        }
if (out != null) {
out.close();
        }
     }
  }
}
```

7. Write a program to get the input from the user using InputStreamReader?

```
import java.io.*;
public class ReadConsole {
public static void main(String args[]) throws IOException {
InputStreamReadercin = null;
try {
cin = new InputStreamReader(System.in);
System.out.println("Enter characters, 'q' to quit.");
char c;
```

```
do {
        c = (char) cin.read();
System.out.print(c);
      } while(c != 'q');
}finally {
if (cin != null) {
cin.close();
 }
```

```
importjava.io.BufferedReader;
importjava.io.IOException;
importjava.io.InputStreamReader;
publicclassTest
{
   publicstaticvoidmain(String[] args) throwsIOException
   {
      //Enter data using BufferReader
      BufferedReader reader = newBufferedReader(newInputStreamReader(System.in));
       // Reading data using readLine
      String name = reader.readLine();
      // Printing the read line
      System.out.println(name);
   }
}
}
 }
}
```

8.Explain the different ways to read and write to console in java ?

Ways to read input from console in Java

1.Using Buffered Reader Class

2.Using Scanner Class

```
   Importjava.util.Scanner;
   ClassGetInputFromUser
{
   Publicstaticvoidmain(String args[])
```

```
    {
       // Using Scanner for Getting Input from User
       Scanner in = newScanner(System.in);

       String s = in.nextLine();
       System.out.println("You entered string "+s);

       inta = in.nextInt();
       System.out.println("You entered integer "+a);

       Floatb = in.nextFloat();
       System.out.println("You entered float "+b);
    }
}
```

3. Using Console Class

```
PublicclassSample
{
   Publicstaticvoidmain(String[] args)
    {
       // Using Console to input data from user
       String name = System.console().readLine();
        System.out.println(name);
    }
}
```

Java Write to Console Output

The PrintStream is an output stream derived from the OutputStream, it also implements the low-level method write(). Thus, write() can be used to write to the console. The simplest form of write() defined by the PrintStream is shown below :
void write(int*byteval*)

```
classWriteConsoleOutput
 {
public static void main(String args[])
    {

int y;
 y = 'X';
System.out.write(y);
System.out.write('\n');
}
 }
```

9. Write a program to perform reading nad writing from a file ?

Writing data in a file

```
class Test
{
public static void main( String args[])
 {
FileOutputStreamfo=new FileOutputStream("prog.txt");
String s1="Welcome to Codesdope";
byte b1[]=s1.getBytes();        //converting string into byte array
fo.write(b1);
fo.close();
 }
}
```

Here, byte b1[ ]=s1.getBytes(); is converting string(character array) into byte array.

Then by writing fo.write(b1);, we are writing the data in a file named prog.txt because fo is the object of the FileOutputStream class.

Reading data from a file

```
class Test1
{
public static void main( String args[])
 {
FileInputStream fi=new FileInputStream("prog.txt");
int n=0;
while((n=fi.read())!=-1){
System.out.println((char)n);
}
fin.close();
```

```
}

}
```

10. Explain stacktrace in java ?

When an exception is thrown in your program, you can find the exact statement in your program that caused the exception to occur by examining the lines that are displayed right after the line that indicates which exception was encountered.

These lines of information that are displayed when an exception occurs is known as *Stack trace.* It lists the different methods that the exception passed through before your program was completely aborted. Each line in the stack trace contains not only the method name and the corresponding classname but also the name of the source file that contains the class and the line number where the exception occurred.

```
public class ModifiedStackTrace
{
    static void B()
    {
     int x = 12 ;
     int y = 0 ;
     int z =x/y;
    System.out.println("z="+z);
    }
    static void A()
    {
       B();
    }
     public static void main(String[] args)
     {
          try
          {
              A();
          }
           catch(ArithmeticException e)
          {
              System.out.println("Integer Division by 0 not Possible,Try Again\n");
              System.out.println("Stack Trace Information...\n");
              e.printStackTrace();
          }
     }
}
```

UNIT IV

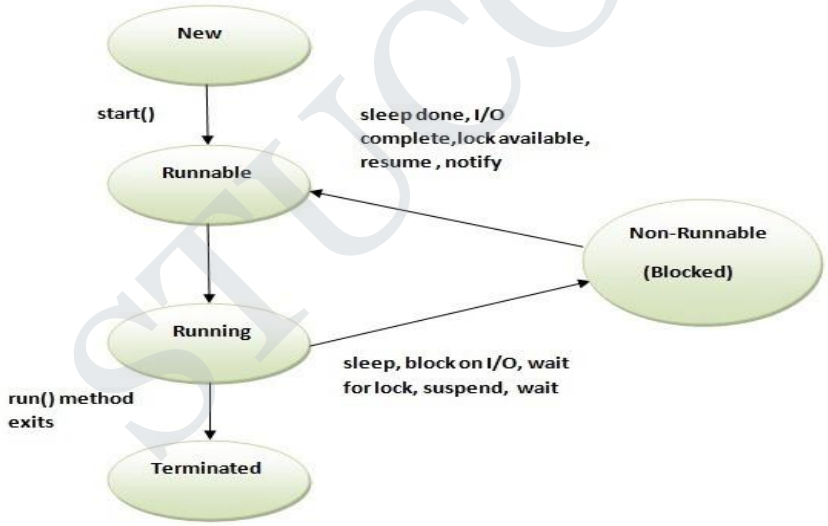
1.Explain the life cycle of thread in Java?

Thread States

A thread can be in one of the five states.

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

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



1) New

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

2) Runnable

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

3) Running

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

4) Non-Runnable (Blocked)

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

5) Terminated

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

2. Explain how to create thread by using runnable interface?

The easiest way to create a thread is to create a class that implements the runnable interface. After implementing runnable interface, the class needs to implement the **run()** method, which is of form,

publicvoidrun()

- run() method introduces a concurrent thread into your program. This thread will end when run() method terminates.
- You must specify the code that your thread will execute inside run() method.
- run() method can call other methods, can use other classes and declare variables just like any other normal method.

To call the **run()** method, **start()** method is used. On calling start(), a new stack is provided to the thread and run() method is called to introduce the new thread into the program.

Note: If you are implementing Runnable interface in your class, then you need to explicitly create a Thread class object and need to pass the Runnable interface implemented class object as a parameter in its constructor.

To make a class runnable, we can implement java.lang.Runnable interface and provide implementation in public void run() method.

To use this class as Thread, we need to create a Thread object by passing object of this runnable class and then call start() method to execute the run() method in a separate thread.

Java thread example by implementing Runnable interface

```
public class HeavyWorkRunnable implements Runnable {

public void run() {
System.out.println("Doing heavy processing - START
      "+Thread.currentThread().getName());
try {
Thread.sleep(1000);
          //Get database connection, delete unused data from DB
doDBProcessing();
          } catch (InterruptedException e) {
e.printStackTrace();
          }
System.out.println("Doing heavy processing - END
        "+Thread.currentThread().getName());
   }
private void doDBProcessing() throws InterruptedException {
Thread.sleep(5000);
   }
}
```

3. How to Create Threads in Java by Extending Thread Class?

The first way to create a thread is to create a subclass of the Thread class. This class must override the run( ) method as discussed above and it may override the other methods too. Then the class that needs the thread can create an object of the class that extends thread class

```
class PrintString1
{
          public static void main(String args[])
          {
                    StringThread1 t = new StringThread1 ("Java",50);
                    t.start ( );
          }
}

class StringThread1 extends Thread
{
          private String str;
```

```
        privateintnum;

        StringThread1 (String s, int n)
        {
                str=new String (s);
                num=n;
        }

        public void run ( )
        {
                for (int i=1; i<=num; i++)
                        System.out.print(str + " ");
        }
}
```

4.Explain how to achieve Synchronization in java?

At times when more than one thread try to access a shared resource, we need to ensure that resource will be used by only one thread at a time.

The process by which this is achieved is called **synchronization**. The synchronization keyword in java creates a block of code referred to as critical section.

General Syntax :

```
synchronized(object)
{
//statement to be synchronized
}
```

Every Java object with a critical section of code gets a lock associated with the object. To enter critical section a thread need to obtain the corresponding object's lock.

If we do not use syncronization, and let two or more threads access a shared resource at the same time, it will lead to distorted results.

Consider an example, Suppose we have two different threads **T1** and **T2**, T1 starts execution and save certain values in a file *temporary.txt* which will be used to calculate some result when T1 returns. Meanwhile, T2 starts and before T1 returns, T2 change the values saved by T1 in the file temporary.txt

Now obviously T1 will return wrong result.

To prevent such problems, synchronization was introduced. With synchronization in above case, once T1 starts using *temporary.txt* file, this file will be **locked**(LOCK mode), and no other thread will be able to access or modify it until T1 returns.

Using Synchronized Methods

Using Synchronized methods is a way to accomplish synchronization. But lets first see what happens when we do not use synchronization in our program.

Using Synchronized block

If you have to synchronize access to an object of a class or you only want a part of a method to be synchronized to an object then you can use synchronized block for it.

```
classFirst
{
publicvoiddisplay(String msg)
{
System.out.print("["+msg);
try
{
Thread.sleep(1000);
}
catch(InterruptedException e)
{
e.printStackTrace();
}
System.out.println("]");
}
}
classSecondextendsThread
{
 String msg;
 First fobj;
 Second (First fp,Stringstr)
{
fobj=fp;
msg=str;
start();
}
publicvoidrun()
{
synchronized(fobj)//Synchronized block
{
fobj.display(msg);
}
```

```
}
}
publicclassSyncro
{
publicstaticvoid main (String[]args)
{
  First fnew=newFirst ();
  Second ss=newSecond (fnew,"welcome");
  Second ss1=newSecond(fnew,"new");
  Second ss2 =newSecond (fnew,"programmer");
}}
```

5. Explain interthread communication in Java?

Inter-thread communication or **Co-operation** is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of **Object class**:

- wait()
- notify()
- notifyAll()

1) wait() method

Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

Method	Description
public final void wait()throws InterruptedException	waits until object is notified.
public final void wait(long	waits for the specified amount of

timeout)throws InterruptedException	time.

2) notify() method

Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. Syntax:

public final void notify()

3) notifyAll() method

Wakes up all threads that are waiting on this object's monitor. Syntax:

public final void notifyAll()

Understanding the process of inter-thread communication

The point to point explanation of the above diagram is as follows:

1. Threads enter to acquire lock.
2. Lock is acquired by on thread.
3. Now thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits.
4. If you call notify() or notifyAll() method, thread moves to the notified state (runnable state).
5. Now thread is available to acquire lock.
6. After completion of the task, thread releases the lock and exits the monitor state of the object.

Example of inter thread communication in java

```
class Customer{
int amount=10000;

synchronized void withdraw(int amount){
System.out.println("going to withdraw...");

if(this.amount<amount){
System.out.println("Less balance; waiting for deposit...");
try{wait();}catch(Exception e){}
}
```

```
this.amount-=amount;
System.out.println("withdraw completed...");
}

synchronized void deposit(int amount){
System.out.println("going to deposit...");
this.amount+=amount;
System.out.println("deposit completed... ");
notify();
}
}
class Test{
public static void main(String args[]){
final Customer c=new Customer();
new Thread(){
public void run(){c.withdraw(15000);}
}.start();
new Thread(){
public void run(){c.deposit(10000);}
}.start();
}}
```

6. Explain Daemon thread in Java with example ?

Daemon thread is a low priority thread (in context of JVM) that runs in background to perform tasks such as garbage collection (gc) etc., they do not prevent the JVM from exiting (even if the daemon thread itself is running) when all the user threads (non-daemon threads) finish their execution.
JVM terminates itself when all user threads (non-daemon threads) finish their execution, JVM does not care whether Daemon thread is running or not, if JVM finds running daemon thread (upon completion of user threads), it terminates the thread and after that shutdown itself.

Properties of Daemon threads:

1. A newly created thread inherits the daemon status of its parent. That's the reason all threads created inside main method (child threads of main thread) are non-daemon by default, because main thread is non-daemon. However you can make a user thread to Daemon by using **setDaemon() method** of thread class.

When the JVM starts, it creates a thread called "Main". Your program will run on this thread, unless you create additional threads yourself. The first thing the "Main" thread does is to look for your static void main (String args[]) method and invoke it. That is the entry-point to your program. If you create additional threads in the main method those threads would be the child threads of main thread.

2. **Methods of Thread class that are related to Daemon threads:**

   public void setDaemon(boolean status): This method is used for making a user thread to Daemon thread or vice versa. For example if I have a user thread t then t.setDaemon(true) would make it Daemon thread. On the other hand if I have a Daemon thread td then by calling td.setDaemon(false) would make it normal thread(user thread/non-daemon thread).
   publicbooleanisDaemon(): This method is used for checking the status of a thread. It returns true if the thread is Daemon else it returns false.

3. setDaemon() method can only be called before starting the thread. This method would throw IllegalThreadStateException if you call this method after Thread.start() method.

DAEMON THREAD EXAMPLES

Example 1: DaemonThreadExample1.java
This example is to demonstrate the usage of setDaemon() and isDaemon() method.

```
publicclassDaemonThreadExample1extendsThread{
publicvoid run(){

        // Checking whether the thread is Daemon or not
        if(Thread.currentThread().isDaemon()){
        System.out.println("Daemon thread executing");
        }
        else{
        System.out.println("user(normal) thread executing");
}
}
publicstaticvoid main(String[]args){
        DaemonThreadExample1 t1=newDaemonThreadExample1();
        DaemonThreadExample1 t2=newDaemonThreadExample1();

t1.setDaemon(true);

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

Example 2: DaemonThreadEx2.java
If you call the setDaemon() method after starting the thread (start() method), it would

throw IllegalThreadStateException. This clearly means that you can call setDaemon() method only before starting a thread.

```
public class DaemonThreadEx2 extends Thread {

public void run(){
        System.out.println("Thread is running");
  }

public static void main(String[] args){
         DaemonThreadEx2 t1=new DaemonThreadEx2();
t1.start();
         // It will throw IllegalThreadStateException
        t1.setDaemon(true);
  }
}
```

Difference between Daemon threads and non-Daemon thread (user thread)

The main difference between Daemon thread and user threads is that the JVM does not wait for Daemon thread before exiting while it waits for user threads, it does not exit until unless all the user threads finish their execution.

7.Explain the concept of ThreadGroup in Java with example ?

Java provides a convenient way to group multiple threads in a single object. In such way, we can suspend, resume or interrupt group of threads by a single method call.

Java thread group is implemented by *java.lang.ThreadGroup* class.

Constructors of ThreadGroup class

There are only two constructors of ThreadGroup class.

No.	Constructor	Description
1)	ThreadGroup(String name)	creates a thread group with given name.
2)	ThreadGroup(ThreadGroup parent, String name)	creates a thread group with given parent group and name.

Important methods of ThreadGroup class

There are many methods in ThreadGroup class. A list of important methods are given below.

No.	Method	Description
1)	intactiveCount()	returns no. of threads running in current group.
2)	intactiveGroupCount()	returns a no. of active group in this thread group.
3)	void destroy()	destroys this thread group and all its sub groups.
4)	String getName()	returns the name of this group.
5)	ThreadGroupgetParent()	returns the parent of this group.
6)	void interrupt()	interrupts all threads of this group.
7)	void list()	prints information of this group to standard console.

THREADGROUP EXAMPLE

```
public class ThreadGroupDemo implements Runnable{
   public void run() {
       System.out.println(Thread.currentThread().getName());
   }
  public static void main(String[] args) {
     ThreadGroupDemo runnable = new ThreadGroupDemo();
  ThreadGroup tg1 = new ThreadGroup("Parent ThreadGroup");
       Thread t1 = new Thread(tg1, runnable,"one");
       t1.start();
      Thread t2 = new Thread(tg1, runnable,"two");
     t2.start();
       Thread t3 = new Thread(tg1, runnable,"three");
       t3.start();
       System.out.println("Thread Group Name: "+tg1.getName());
       tg1.list();
      }
```

}

8. Explain Generic Programming in Java?

Generics in Java are similar to templates in C++. The idea is to allow type (Integer, String and user defined types) to be a parameter to methods, classes and interfaces.

For example, classes like HashSet, ArrayList, HashMap, etc use generics very well. We can use them for any type.

Generic Class

we use <> to specify parameter types in generic class creation. To create objects of generic class, we use following syntax.

```
// To create an instance of generic class
BaseType<Type>obj = new BaseType<Type>()
```

A Simple Java program to show working of user defined Generic classes

```
// We use <> to specify Parameter type
classTest<T>
{
   // An object of type T is declared
   T obj;
   Test(T obj) {  this.obj = obj;  }  // constructor
   publicT getObject()  { returnthis.obj; }
}

classMain
{
   publicstaticvoidmain (String[] args)
   {
      // instance of Integer type
      Test <Integer>iObj = newTest<Integer>(15);
      System.out.println(iObj.getObject());

      // instance of String type
      Test <String>sObj =newTest<String>("G");
      System.out.println(sObj.getObject());
   }
}
```

Advantages of Generics

Programs that uses Generics has got many benefits over non-generic code.

1. Code Reuse: We can write a method/class/interface once and use for any type we want.
2. Type Safety: Generics make errors to appear compile time than at run time (It's always better to know problems in your code at compile time rather than making your code fail at run time).
3. Individual Type Casting is not needed.
4. Implementing generic algorithms: By using generics, we can implement algorithms that work on different types of objects and at the same they are type safe too.

9. Explain concept of Generic class with example program?

A generic class declaration looks like a non-generic class declaration, except that the class name is followed by a type parameter section.

As with generic methods, the type parameter section of a generic class can have one or more type parameters separated by commas. These classes are known as parameterized classes or parameterized types because they accept one or more parameters.

Example

Following example illustrates how we can define a generic class −

```
publicclassBox<T>{
private T t;
publicvoid add(T t){
this.t = t;
}
public T get(){
return t;
}
publicstaticvoid main(String[]args){
Box<Integer>integerBox=newBox<Integer>();
Box<String>stringBox=newBox<String>();
integerBox.add(newInteger(10));
stringBox.add(newString("Hello World"));
System.out.printf("Integer Value :%d\n\n",integerBox.get());
System.out.printf("String Value :%s\n",stringBox.get());
}
}
```

10. Explain Generic Methods in java with example program?

Java **Generic** methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods, or with a single class declaration, a set of related types, respectively.

Generics also provide compile-time type safety that allows programmers to catch invalid types at compile time.

Generic Methods

You can write a single generic method declaration that can be called with arguments of different types. Based on the types of the arguments passed to the generic method, the compiler handles each method call appropriately. Following are the rules to define Generic Methods −

- All generic method declarations have a type parameter section delimited by angle brackets (< and >) that precedes the method's return type (< E > in the next example).
- Each type parameter section contains one or more type parameters separated by commas. A type parameter, also known as a type variable, is an identifier that specifies a generic type name.
- The type parameters can be used to declare the return type and act as placeholders for the types of the arguments passed to the generic method, which are known as actual type arguments.
- A generic method's body is declared like that of any other method. Note that type parameters can represent only reference types, not primitive types (like int, double and char).

Example

Following example illustrates how we can print an array of different type using a single Generic method public class GenericMethodTest {

```
// generic method printArray

public static < E > void printArray( E[] inputArray ) {

    // Display array elements

for(E element : inputArray) {

System.out.printf("%s ", element);
```

```
    }
System.out.println();
  }
public static void main(String args[]) {
Integer[] intArray = { 1, 2, 3, 4, 5 };
Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };
System.out.println("Array integerArray contains:");
printArray(intArray);   // pass an Integer array
System.out.println("\nArraydoubleArray contains:");
printArray(doubleArray);   // pass a Double array
System.out.println("\nArraycharacterArray contains:");
printArray(charArray);   // pass a Character array
  }}
```

11.Give Examples for implementing bounded types?

Example on how to implement bounded types (extend superclass) with Generics

```
class Bound<T extends A>
{
        private T objRef;
        public Bound(T obj){
      this.objRef = obj;
   }
     public void doRunTest(){
    this.objRef.displayClass();
   }
}
class A
{
```

```
    public void displayClass()
    {
        System.out.println("Inside super class A");
    }
}
 class B extends A
{
    public void displayClass()
    {
        System.out.println("Inside sub class B");
    }
}

class C extends A
{
    public void displayClass()
    {
        System.out.println("Inside sub class C");
    }
}

public class BoundedClass
{
    public static void main(String a[])
    {

        // Creating object of sub class C and
        // passing it to Bound as a type parameter.
        Bound<C>bec = new Bound<C>(new C());
        bec.doRunTest();

        // Creating object of sub class B and
        // passing it to Bound as a type parameter.
        Bound<B>beb = new Bound<B>(new B());
        beb.doRunTest();

        // similarly passing super class A
        Bound<A>bea = new Bound<A>(new A());
        bea.doRunTest();

    }
}
```

MULTIPLE BOUNDS

Bounded type parameters can be used with methods as well as classes and interfaces.

Java Generics supports multiple bounds also, i.e . In this case A can be an interface or class. If A is class then B and C should be interfaces. We can't have more than one class in multiple bounds.

Syntax

<T extends **superClassName&Interface**>

```
class Bound<T extends A & B>
{

   private T objRef;

   public Bound(T obj){
      this.objRef = obj;
   }

   public void doRunTest(){
      this.objRef.displayClass();
   }
}

interface B
{
   public void displayClass();
}

class A implements B
{
   public void displayClass()
   {
      System.out.println("Inside super class A");
   }
}

public class BoundedClass
{
   public static void main(String a[])
   {
      //Creating object of sub class A and
      //passing it to Bound as a type parameter.
      Bound<A>bea = new Bound<A>(new A());
      bea.doRunTest();

   }
}
```

UNIT V

1. Explain the methods in Graphics class with example?

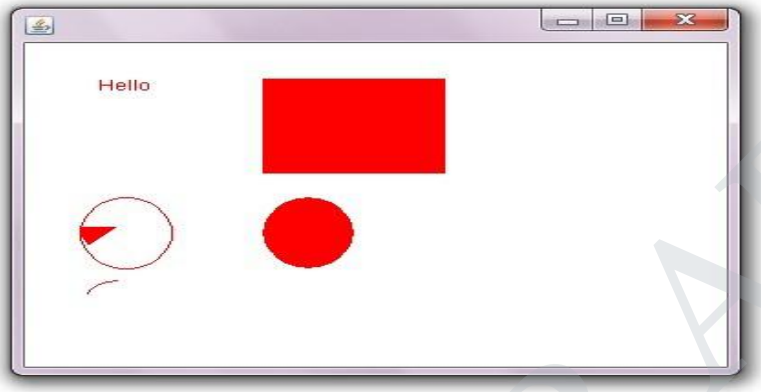
Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int**

startAngle, int arcAngle): is used to fill a circular or elliptical arc.

10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Example of displaying graphics in swing




```
import java.awt.*;
import javax.swing.JFrame;

public class DisplayGraphics extends Canvas{

  public void paint(Graphics g) {
    g.drawString("Hello",40,40);
    setBackground(Color.WHITE);
    g.fillRect(130, 30,100, 80);
    g.drawOval(30,130,50, 60);
    setForeground(Color.RED);
    g.fillOval(130,130,50, 60);
    g.drawArc(30, 200, 40,50,90,60);
    g.fillArc(30, 130, 40,50,180,40);

  }
    public static void main(String[] args) {
    DisplayGraphics m=new DisplayGraphics();
    JFrame f=new JFrame();
    f.add(m);
    f.setSize(400,400);
    //f.setLayout(null);
    f.setVisible(true);
  }
```

```
}
```

2. Explain Java Action Listener Interface with example?

The Java ActionListener is notified whenever you click on the button or menu item. It is notified against ActionEvent. The ActionListener interface is found in java.awt.event package. It has only one method: actionPerformed().

actionPerformed() method

The actionPerformed() method is invoked automatically whenever you click on the registered component.

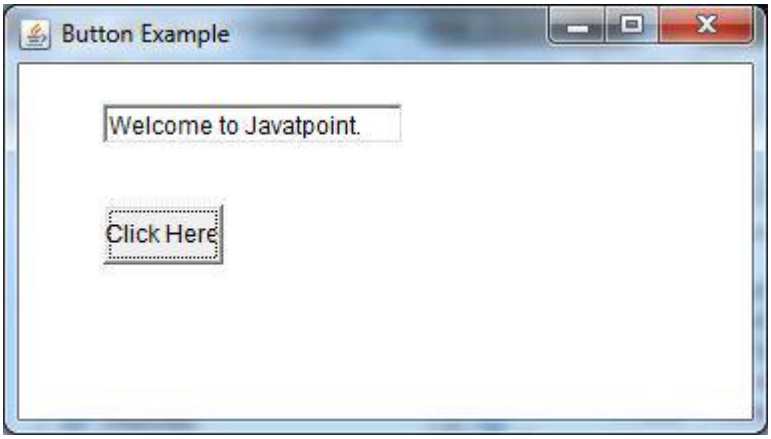
public abstract void actionPerformed(ActionEvent e);

Java ActionListener Example: On Button click

```
import java.awt.*;
import java.awt.event.*;
public class ActionListenerExample {
public static void main(String[] args) {
    Frame f=new Frame("ActionListener Example");
    final TextField tf=new TextField();
    tf.setBounds(50,50, 150,20);
    Button b=new Button("Click Here");
    b.setBounds(50,100,60,30);

    b.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        tf.setText("Welcome to Javatpoint.");
    }
    });
    f.add(b);f.add(tf);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

Output:



3. Write a program to display colors in a frame?

```
import java.awt.Graphics;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.image.BufferedImage;
import javax.swing.JComponent;
import javax.swing.JFrame;
 public class Main extends JComponent {
 BufferedImage image;
 public void initialize() {
 int width = getSize().width;
 int height = getSize().height;
  int[] data = new int[width * height];
  int index = 0;
  for (int i = 0; i < height; i++) {
  int red = (i * 255) / (height - 1);
  for (int j = 0; j < width; j++) {
  int green = (j * 255) / (width - 1);
  int blue = 128;
```

```
  data[index++] = (red<<16) | (green<<8) | blue;
    }
    }
image = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
image.setRGB(0, 0, width, height, data, 0, width);
}
public void paint(Graphics g) {
if (image == null)
initialize();
g.drawImage(image, 0, 0, this);
 }
public static void main(String[] args) {
JFrame f = new JFrame("Display Colours");
f.getContentPane().add(new Main());
f.setSize(300, 300);
f.setLocation(100, 100);
f.addWindowListener(new WindowAdapter() {
public void windowClosing(WindowEvent e) {
System.exit(0);
}
};
f.setVisible(true);
}
}
```

4. **Explain with example how to display image using swing?**

For displaying image, we can use the method drawImage() of Graphics class.

Syntax of drawImage() method:

public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer) is used to draw the specified image.

Example of displaying image in swing

```
import java.awt.*;
import javax.swing.JFrame;

public class MyCanvas extends Canvas{

   public void paint(Graphics g) {

      Toolkit t=Toolkit.getDefaultToolkit();
      Image i=t.getImage("p3.gif");
      g.drawImage(i, 120,100,this);

   }
      public static void main(String[] args) {
      MyCanvas m=new MyCanvas();
      JFrame f=new JFrame();
      f.add(m);
      f.setSize(400,400);
      f.setVisible(true);
    } }
```

5. Explain Java Adapter Classes with example ?

Java adapter classes *provide the default implementation of listener interfaces*. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it *saves code*.

The Adapter classes with their corresponding listener interfaces are given below.

java.awt.event Adapter classes

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener

MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

Java WindowAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class AdapterExample{
    Frame f;
    AdapterExample(){
        f=new Frame("Window Adapter");
            f.addWindowListener(new WindowAdapter(){
           public void windowClosing(WindowEvent e) {
             f.dispose();
           }
        });
         f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
public static void main(String[] args) {
```

```
    new AdapterExample();
}
}
```

Java KeyAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class KeyAdapterExample extends KeyAdapter{
    Label l;
    TextArea area;
    Frame f;
    KeyAdapterExample(){
        f=new Frame("Key Adapter");
        l=new Label();
        l.setBounds(20,50,200,20);
        area=new TextArea();
        area.setBounds(20,80,300, 300);
        area.addKeyListener(this);

        f.add(l);f.add(area);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public void keyReleased(KeyEvent e) {
        String text=area.getText();
        String words[]=text.split("\\s");
        l.setText("Words: "+words.length+" Characters:"+text.length());
```

```
    }
      public static void main(String[] args) {
        new KeyAdapterExample();
    }
}
```

6. Explain flow layout with example?

In Java swing, Layout manager is used to position all its components, with setting properties, such as the size, the shape and the arrangement. Different layout managers could have varies different settings on its components.

FlowLayout

The FlowLayout arranges the components in a directional flow, either from left to right or from right to left. Normally all components are set to one row, according to the order of different components. If all components can not be fit into one row, it will start a new row and fit the rest in.

To construct a FlowLayout, three options could be chosen:

- FlowLayout(): construct a new FlowLayout object with center alignment and horizontal and vertical gap to be default size of 5 pixels.
- FlowLayout(int align): construct similar object with different settings on alignment
- FlowLayout(int align, int hgap, int vgap): construct similar object with different settings on alignment and gaps between components.

For the constructor with the alignment settings, the possible values could be: LEFT, RIGHT, CENTER, LEADING and TRAILING.

```
import java.awt.*;

import java.awt.event.*;

import javax.swing.*;

public class SwingLayoutDemo {

   private JFrame mainFrame;

   private JLabel headerLabel;

   private JLabel statusLabel;

   private JPanel controlPanel;
```

```
private JLabel msglabel;
public SwingLayoutDemo(){
   prepareGUI();
}
public static void main(String[] args){
   SwingLayoutDemo swingLayoutDemo = new SwingLayoutDemo();
   swingLayoutDemo.showFlowLayoutDemo();
}
private void prepareGUI(){
   mainFrame = new JFrame("Java SWING Examples");
   mainFrame.setSize(400,400);
   mainFrame.setLayout(new GridLayout(3, 1));
   headerLabel = new JLabel("",JLabel.CENTER );
   statusLabel = new JLabel("",JLabel.CENTER);
   statusLabel.setSize(350,100);
    mainFrame.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent windowEvent){
        System.exit(0);
      }
   });
   controlPanel = new JPanel();
   controlPanel.setLayout(new FlowLayout());
   mainFrame.add(headerLabel);
   mainFrame.add(controlPanel);
   mainFrame.add(statusLabel);
```

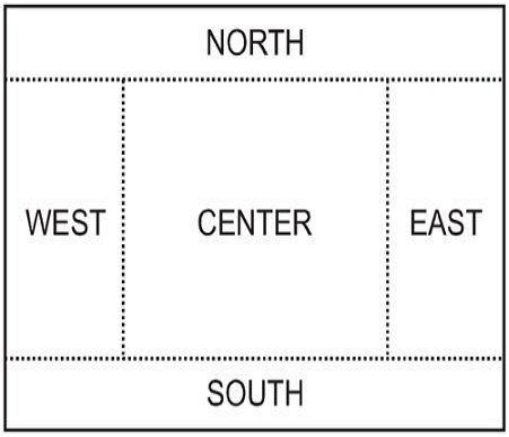
```
    mainFrame.setVisible(true);
  }
   private void showFlowLayoutDemo(){
    headerLabel.setText("Layout in action: FlowLayout");
    JPanel panel = new JPanel();
    panel.setBackground(Color.darkGray);
    panel.setSize(200,200);
    FlowLayout layout = new FlowLayout();
    layout.setHgap(10);
    layout.setVgap(10);
    panel.setLayout(layout);
    panel.add(new JButton("OK"));
    panel.add(new JButton("Cancel"));
    controlPanel.add(panel);
    mainFrame.setVisible(true);
  }
}
```

7.Explain Border Layout with example ?

The BorderLayout manager is one that divides a container into five regions distinct : north (upper region) , south (lower region) , west (left region) , east (right region) and center (central region) .

For defining the BorderLayout, use the following syntax: -
name container.setLayout (new BorderLayout (horizontal - spacing , vertical - spacing) spacing - horizontal and vertical - spacing

Distribution Manager BorderLayout divides the container JPanel into five zones, one for each component, as can be seen in Figure.




```
import javax.swing.*;
import java.awt.*;
public class BorderLayoutJavaExample extends JFrame
 {
    private JButton NrthBtn = new JButton("North Button");
    private JButton SthBtn = new JButton("South Button");
    private JButton EstBtn = new JButton("East Button");
    private JButton WstBtn = new JButton("West Button");
    private JButton CntrBtn = new JButton("Center Button");
    public BorderLayoutJavaExample()
    {
     setLayout(new BorderLayout());
     add(NrthBtn,BorderLayout.NORTH);
     add(SthBtn, BorderLayout.SOUTH);
     add(EstBtn, BorderLayout.EAST);
     add(WstBtn, BorderLayout.WEST);
     add(CntrBtn, BorderLayout.CENTER);
     setVisible(true);
     setSize(400, 150);
    }
     public static void main(String[] args)
      {
        BorderLayoutJavaExample frame = new BorderLayoutJavaExample();
      }
 }
```

8. Explain Swing Components and Containers with examples ?

A component is an independent visual control. Swing Framework contains a large set of components which provide rich functionalities and allow high level of customization. They all are derived from JComponent class. All these components are lightweight components. This class provides some common functionality like pluggable look and feel, support for accessibility, drag and drop, layout, etc.

A container holds a group of components. It provides a space where a component can be managed and displayed. Containers are of two types:

Top level Containers

It inherits Component and Container of AWT.

It cannot be contained within other containers.

Heavyweight.

Example: JFrame, JDialog, JApplet

Lightweight Containers

It inherits JComponent class.

It is a general purpose container.

It can be used to organize related components together.

JButton

JButton class provides functionality of a button. JButton class has three constuctors,

JButton(Icon ic)

JButton(String str)

JButton(String str, Icon ic)

It allows a button to be created using icon, a string or both. JButton supports ActionEvent. When a button is pressed an ActionEvent is generated.

Example using JButton

```
import javax.swing.*;

import java.awt.event.*;

import java.awt.*;

public class testswing extends JFrame

{

 testswing()

 {

  JButton bt1 = new JButton("Yes");              //Creating a Yes Button.
```

```
 JButton bt2 = new JButton("No");              //Creating a No Button.
 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)    //setting close operation.
 setLayout(new FlowLayout());                  //setting layout using FlowLayout object
 setSize(400, 400);                        //setting size of Jframe
 add(bt1);               //adding Yes button to frame.
 add(bt2);               //adding No button to frame.
 setVisible(true);
}
 public static void main(String[] args)
 {
  new testswing();
 }
}
```

JTextField

JTextField is used for taking input of single line of text. It is most widely used text component. It has three constructors,

JTextField(int cols)

JTextField(String str, int cols)

JTextField(String str)

cols represent the number of columns in text field.

Example using JTextField

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class MyTextField extends JFrame
```

```
{
 public MyTextField()
 {
  JTextField jtf = new JTextField(20);          //creating JTextField.
  add(jtf);                                     //adding JTextField to frame.
  setLayout(new FlowLayout());
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setSize(400, 400);
  setVisible(true);
 }
 public static void main(String[] args)
 {
  new MyTextField();
 }
}
```

JCheckBox

JCheckBox class is used to create checkboxes in frame. Following is constructor for JCheckBox,

JCheckBox(String str)

Example using JCheckBox

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class Test extends JFrame
{
 public Test()
```

```
{
 JCheckBox jcb = new JCheckBox("yes");   //creating JCheckBox.
 add(jcb);                               //adding JCheckBox to frame.
 jcb = new JCheckBox("no");              //creating JCheckBox.
 add(jcb);                               //adding JCheckBox to frame.
 jcb = new JCheckBox("maybe");           //creating JCheckBox.
 add(jcb);                               //adding JCheckBox to frame.
 setLayout(new FlowLayout());
 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 setSize(400, 400);
 setVisible(true);
}
public static void main(String[] args)
{
 new Test();
}
}
```

JRadioButton

Radio button is a group of related button in which only one can be selected. JRadioButton class is used to create a radio button in Frames. Following is the constructor for JRadioButton,

JRadioButton(String str)

Example using JRadioButton

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
```

```
public class Test extends JFrame
{
 public Test()
 {
  JRadioButton jcb = new JRadioButton("A");          //creating JRadioButton.
  add(jcb);                                          //adding JRadioButton to frame.
  jcb = new JRadioButton("B");                       //creating JRadioButton.
  add(jcb);                                          //adding JRadioButton to frame.
  jcb = new JRadioButton("C");                       //creating JRadioButton.
  add(jcb);                                          //adding JRadioButton to frame.
  jcb = new JRadioButton("none");
  add(jcb);
  setLayout(new FlowLayout());
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setSize(400, 400);
  setVisible(true);
 }
 public static void main(String[] args)
 {
  new Test();
 }
}
```

JComboBox

Combo box is a combination of text fields and drop-down list.JComboBox component is used to create a combo box in Swing. Following is the constructor for JComboBox,

JComboBox(String arr[])

Example using JComboBox

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class Test extends JFrame
{
 String name[] = {"Abhi","Adam","Alex","Ashkay"};  //list of name.
 public Test()
 {
  JComboBox jc = new JComboBox(name);  //initialzing combo box with list of name.
  add(jc);                                    //adding JComboBox to frame.
  setLayout(new FlowLayout());
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setSize(400, 400);
  setVisible(true);
 }
 public static void main(String[] args)
 {
  new Test();
 }
}
```

9. Write a program to change background color of a frame (Using Action Event)

```
import java.awt.*;  //importing awt package
import javax.swing.*;  //importing swing package
import java.awt.event.*;  //importing event package
```

```
//For an event to occur upon clicking the button, ActionListener interface should be
implemented

class StColor extends JFrame implements ActionListener{

JFrame frame;

JPanel panel;

JButton b1,b2,b3,b4,b5;

 StColor(){

 frame = new JFrame("COLORS");

 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

 panel = new JPanel();

 panel.setSize(100, 50);

 b1 = new JButton("BLUE");  //Creating a button named BLUE

 b1.addActionListener(this);  //Registering the button with the listener

 b2 = new JButton("RED");  //Creating a button named RED

 b2.addActionListener(this);  //Registering the button with the listener

b3 = new JButton("CYAN");//Creating a button named CYAN

 b3.addActionListener(this);//Registering the button with the listener

   b4 = new JButton("PINK");  //Creating a button named PINK

  b4.addActionListener(this);  //Registering the button with the listener

   b5 = new JButton("MAGENTA");  //Creating a button named MAGENTA

   b5.addActionListener(this);  //Registering the button with the listener

   //Adding buttons to the Panel

   panel.add(b1);

   panel.add(b2);

   panel.add(b3);

   panel.add(b4);
```

```
    panel.add(b5);
    frame.getContentPane().add(panel);  //adding panel to the frame
    frame.setSize(500,300);
    frame.setVisible(true);
    frame.setLayout(new FlowLayout());
}
    public void actionPerformed(ActionEvent e) {
        Object see = e.getSource();
        if(see ==(b1)){  //Checking if the object returned is of button1
        frame.getContentPane().setBackground(java.awt.Color.blue);        }
        if(see == b2){  //Checking if the object returned is of button2
       frame.getContentPane().setBackground(java.awt.Color.red);          }
        if(see == b3){  //Checking if the object returned is of button3
        frame.getContentPane().setBackground(java.awt.Color.cyan        }
        if(see == b4){  //Checking if the object returned is of button4
       frame.getContentPane().setBackground(java.awt.Color.pink);          }
        if(see == b5){  //Checking if the object returned is of button5
        frame.getContentPane().setBackground(java.awt.Color.magenta);  //changing the
panel color to magenta
        }
    }
}
class Test {
    public static void main(String[] args) {
        StColor o = new StColor();
    }
```

}

10. How will you Create Scrollbar in java swing?

A scrollbar consists of a rectangular tab called *slider* or *thumb* located between two *arrow buttons*. Two arrow buttons control the position of the slider by increasing or decreasing a number of units, one unit by default.

The area between the slider and the arrow buttons is known as paging area. If user clicks on the paging area, the slider will move one block, normally 10 units.

The slider's position of scrollbar can be changed by:

- Dragging the slider up and down or left and right.
- Pushing on either of two arrow buttons.
- Clicking the paging area.
- Create a scrollbar in swing, you use JScrollBar class. You can create either vertical or horizontal scrollbar.
- Here are the JScrollBar's constructors.

Constructors	Descriptions
JScrollBar()	Creates a vertical scrollbar.
JScrollBar(int orientation)	Creates a scrollbar with a given orientation.
JScrollBar(int orientation, int value, int extent, int min, int max)	Creates a scrollbar with a given orientation and initialize the following scrollbar's properties: value, extent, minimum, and maximum.

Example of using JScrollBar

```
package jscrollbardemo;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Main {

    public static void main(String[] args) {
        final JFrame frame = new JFrame("JScrollbar Demo");
        final JLabel label = new JLabel( );

        JScrollBar hbar=new JScrollBar(JScrollBar.HORIZONTAL, 30, 20, 0, 500);
        JScrollBar vbar=new JScrollBar(JScrollBar.VERTICAL, 30, 40, 0, 500);

        class MyAdjustmentListener implements AdjustmentListener {
            public void adjustmentValueChanged(AdjustmentEvent e) {
```

```
            label.setText("Slider's position is " + e.getValue());
            frame.repaint();
        }
    }

    hbar.addAdjustmentListener(new MyAdjustmentListener( ));
    vbar.addAdjustmentListener(new MyAdjustmentListener( ));

    frame.setLayout(new BorderLayout( ));
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(300,200);
    frame.getContentPane().add(label);
    frame.getContentPane().add(hbar, BorderLayout.SOUTH);
    frame.getContentPane().add(vbar, BorderLayout.EAST);
    frame.getContentPane().add(label, BorderLayout.CENTER);
    frame.setVisible(true);
  }
}
```