Week 7

//Java Program to illustrate how to declare, instantiate, initialize and traverse the Java array.

class Testarray

{

public static void main(String args[])

{

int a[]=new int[5];//declaration and instantiation

a[0]=10;//initialization

a[1]=20;

a[2]=70;

a[3]=40;

a[4]=50;

//traversing array

for(int i=0;i<a.length;i++)

{

System.out.println(a[i]);

}

}

}

//Java Program to print the array elements using for-each loop

class Testarray1

{

public static void main(String args[])

{

int arr[]={33,3,4,5}; //printing array using for-each loop

for(int i:arr)

System.out.println(i);

}

}

//Java Program to illustrate the use of multidimensional array

class Testarray3

{

public static void main(String args[])

{

//declaring and initializing 2D array

int arr[][]={{1,2,3},{2,4,5},{4,4,5}};

//printing 2D array

for(int i=0;i<3;i++)

for(int j=0;j<3;j++)

System.out.println(arr[i][j] + “”);

System.out.println();

}

}

**StringExample.java public**

class StringExample

{

public static void main(String args[])

{

String s1="java";//creating string by Java string literal

char ch[]={'s','t','r','i','n','g','s'};

String s2=new String(ch);//converting char array to string

String s3=new String("example”) ;//creating Java string by new keyword

System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

}

}

**Testimmutablestring.java**

class Testimmutablestring

{

public static void main(String args[])

{

String s="Sachin"; s.concat(" Tendulkar");

//concat() method appends the string at the end

System.out.println(s);//will print Sachin because strings are immutable objects

}

}

**Testimmutablestring1.java**

class Testimmutablestring1

{

public static void main(String args[])

{

String s="Sachin"; s=s.concat(" Tendulkar");

System.out.println(s);

}

}

What is a mutable String? A String that can be modified or changed is known as mutable String. StringBuffer and StringBuilder classes are used for creating mutable strings.

class StringBufferExample

{

public static void main(String args[])

{

StringBuffer sb=new StringBuffer("Hello ");

sb.append("Java");//now original string is changed

System.out.println(sb);//prints Hello Java

}

}

Week 8

Design a Employee Class and demonstrate inheritance using Programmer class

##### **Java Inheritance Example**

class employee

{

float salary=40000;

}

class Programmer extends Employee

{

int bonus=10000;

public static void main(String args[])

{

Programmer p=new Programmer();

System.out.println("Programmer salary is:"+p.salary); System.out.println("Bonus of Programmer is:"+p.bonus);

}

}

**ex:-**Demonstrate use of Single Inheritance.

//Dog class inherits the Animal class,

class Animal

{

void eat()

{

System.out.println("eating...");}

}

class Dog extends Animal

{

void bark()

{

System.out.println("barking...");}

}

class TestInheritance

{

public static void main(String args[])

{

Dog d=new Dog();

d.bark();

d.eat();

}

}

Design a cMultilevel Inheritance

**Example:-**BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

class Animal

{

void eat()

{

System.out.println("eating...");}

}

class Dog extends Animal

{

void bark()

{

System.out.println("barking...");

}

}

class BabyDog extends Dog

{

void weep()

{

System.out.println("weeping...");}

}

class TestInheritance2

{

public static void main(String args[])

{

BabyDog d=new BabyDog(); d.weep();

d.bark();

d.eat();

}

}

**Develop a program to demonstrate hierarchical inheritance**

// Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

class Animal

{

void eat()

{

System.out.println("eating...");}

}

class Dog extends Animal

{

void bark()

{

System.out.println("barking...");}

}

class Cat extends Animal

{

void meow()

{

System.out.println("meowing...");

}

}

class TestInheritance3

{

public static void main(String args[])

{

Cat c=new Cat();

c.meow();

c.eat();

//c.bark();//C.T.Error

}

}

Ex: Program to illustrate Open Closed Principle

class Cuboid

{

    public double length;

    public double breadth;

    public double height;

}

class Application

{

  public double get\_total\_volume(Cuboid[] geo\_objects)

    {

        double vol\_sum = 0;

  for (Cuboid geo\_obj : geo\_objects)

{

          vol\_sum += geo\_obj.length \* geo\_obj.breadth

                       \* geo\_obj.height;

        }

  return vol\_sum;

    }

}

public class GFG {

    public static void main(String args[])

    {

     Cuboid cb1 = new Cuboid();

 cb1.length = 5;

        cb1.breadth = 10;

        cb1.height = 15;

   Cuboid cb2 = new Cuboid();

        cb2.length = 2;

        cb2.breadth = 4;

        cb2.height = 6;

Cuboid cb3 = new Cuboid();

 cb3.length = 3;

        cb3.breadth = 12;

        cb3.height = 15;

  Cuboid[] c\_arr = new Cuboid[3];

        c\_arr[0] = cb1;

        c\_arr[1] = cb2;

        c\_arr[2] = cb3;

        Application app = new Application();

        double vol = app.get\_total\_volume(c\_arr);

       System.out.println("The total volume is " + vol);

    }

}

Week 9

Develop a java program to demonstrate Method overloading in polymorphism

// Java Program for Method overloading

// By using Different Types of Arguments

// Class 1 class Helper

{

// Method with 2 integer parameters

static int Multiply(int a, int b)

{

// Returns product of integer numbers

return a \* b;

}

// Method 2 // With same name but with 2 double parameters

static double Multiply(double a, double b)

{

// Returns product of double numbers

return a \* b;

}

}

// Class 2

// Main class

class GFG

{

// Main driver method public static void main(String[] args)

{

// Calling method by passing

// input as in arguments

System.out.println(Helper.Multiply(2, 4));

System.out.println(Helper.Multiply(5.5, 6.3));

}

}

// Java program for Method Overloading

// by Using Different Numbers of Arguments

// Class 1

// Helper class

class Helper

{

// Method 1

// Multiplication of 2 numbers

static int Multiply(int a, int b)

{

// Return product return a \* b;

}

// Method 2 //

// Multiplication of 3 numbers

static int Multiply(int a, int b, int c)

{

// Return product return a \* b \* c;

}

}

// Class 2

// Main class class GFG

{

// Main driver method

public static void main(String[] args)

{

// Calling method by passing

// input as in arguments

System.out.println(Helper.Multiply(2, 4));

System.out.println(Helper.Multiply(2, 7, 3));

}

}

**Java Runtime Polymorphism: BIKE**

**class** Bike

{

**void** run()

{

System.out.println("running");

}

}

**class** Splendor **extends** Bike

{

**void** run()

{

System.out.println ("running safely with 60km");

}

**public** **static** **void** main(String args[])

{

Bike b = **new** Splendor (); //upcasting

b.run();

}

}

Output:

**running safe running safely with 60km**

**Java Runtime Polymorphism: Shape**

**class** Shape

{

**void** draw()

{System.out.println("drawing...");}

}

**class** Rectangle **extends** Shape

{

**void** draw()

{System.out.println("drawing rectangle...");}

}

**class** Circle **extends** Shape

{

**void** draw()

{System.out.println("drawing circle...");}

}

**class** Triangle **extends** Shape{

**void** draw()

{System.out.println("drawing triangle...");}

}

**class** TestPolymorphism2

{

**public** **static** **void** main(String args[])

{

Shape s;

s=**new** Rectangle();

s.draw();

s=**new** Circle();

s.draw();

s=**new** Triangle();

s.draw();

}

}

OUTPUT: drawing rectangle... drawing circle... drawing triangle...

Week 10

Design a Bike abstract class that contains only one abstract method run.

Provide its implementation through Honda class.

**abstract** **class** Bike{

**abstract** **void** run();

}

**class** Honda4 **extends** Bike

{

**void** run()

{

System.out.println("running safely");

}

**public** **static** **void** main(String args[])

{

 Bike obj = **new** Honda4();

 obj.run();

}

}

Design a abstract class Bank which has method for rate of interest. Demonstrate the use of this abstract class

***TestBank.java***

**abstract** **class** Bank{

**abstract** **int** getRateOfInterest();

}

**class** SBI **extends** Bank{

**int** getRateOfInterest(){**return** 7;}

}

**class** PNB **extends** Bank{

**int** getRateOfInterest(){**return** 8;}

}

**class** TestBank{

**public** **static** **void** main(String args[]){

Bank b;

b=**new** SBI();

System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");

b=**new** PNB();

System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");

}

}

### Develop Abstract class having constructor, data member and methods

//Example of an abstract class that has abstract and non-abstract methods

**abstract** **class** Bike{

   Bike(){System.out.println("bike is created");}

**abstract** **void** run();

**void** changeGear(){System.out.println("gear changed");}

 }

//Creating a Child class which inherits Abstract class

**class** Honda **extends** Bike{

**void** run(){System.out.println("running safely..");}

 }

//Creating a Test class which calls abstract and non-abstract methods

**class** TestAbstraction2{

**public** **static** **void** main(String args[]){

  Bike obj = **new** Honda();

  obj.run();

  obj.changeGear();

 }

}

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

**interface** printable

{

**void** print();

}

**class** A6 **implements** printable

{

**public** **void** print()

{

System.out.println("Hello");

}

**public** **static** **void** main(String args[])

{

A6 obj = **new** A6();

obj.print();

 }

}

Develop simple Interface with class

**interface** printable{

**void** print();

}

**class** A6 **implements** printable{

**public** **void** print(){System.out.println("Hello");}

**public** **static** **void** main(String args[]){

A6 obj = **new** A6();

obj.print();

 }  }

Develop Drawable Interface using classes like Rectangle, Circle

//Interface declaration: by first user

**interface** Drawable{

**void** draw();

}

//Implementation: by second user

**class** Rectangle **implements** Drawable{

**public** **void** draw(){System.out.println("drawing rectangle");}

}

**class** Circle **implements** Drawable{

**public** **void** draw(){System.out.println("drawing circle");}

}

//Using interface: by third user

**class** TestInterface1{

**public** **static** **void** main(String args[]){

Drawable d=**new** Circle();//In real scenario, object is provided by method e.g. getDrawable()

d.draw();

}}

Develop a interface of bank

*File: TestInterface2.java*

**interface** Bank{

**float** rateOfInterest();

}

**class** SBI **implements** Bank{

**public** **float** rateOfInterest(){**return** 9.15f;}

}

**class** PNB **implements** Bank{

**public** **float** rateOfInterest(){**return** 9.7f;}

}

**class** TestInterface2{

**public** **static** **void** main(String[] args){

Bank b=**new** SBI();

System.out.println("ROI: "+b.rateOfInterest());

}}

Multiple inheritance interface

**interface** Printable{

**void** print();

}

**interface** Showable{

**void** show();

}

**class** A7 **implements** Printable,Showable{

**public** **void** print(){System.out.println("Hello");}

**public** **void** show(){System.out.println("Welcome");}

**public** **static** **void** main(String args[]){

A7 obj = **new** A7();

obj.print();

obj.show();

 }

}

DEVELOP A INHERITANCE INTERFACE

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

**interface** Printable{

**void** print();

}

**interface** Showable **extends** Printable{

**void** show();

}

**class** TestInterface4 **implements** Showable{

**public** **void** print(){System.out.println("Hello");}

**public** **void** show(){System.out.println("Welcome");}

**public** **static** **void** main(String args[]){

TestInterface4 obj = **new** TestInterface4();

obj.print();

obj.show();

 }

}

Example of abstract class and interface in Java

simple example where we are using interface and abstract class both.

//Creating interface that has 4 methods

**interface** A{

**void** a();//bydefault, public and abstract

**void** b();

**void** c();

**void** d();

}

//Creating abstract class that provides the implementation of one method of A interface

**abstract** **class** B **implements** A{

**public** **void** c(){System.out.println("I am C");}

}

//Creating subclass of abstract class, now we need to provide the implementation of rest of the methods

**class** M **extends** B{

**public** **void** a(){System.out.println("I am a");}

**public** **void** b(){System.out.println("I am b");}

**public** **void** d(){System.out.println("I am d");}

}

//Creating a test class that calls the methods of A interface

**class** Test5{

**public** **static** **void** main(String args[]){

A a=**new** M();

a.a();

a.b();

a.c();

a.d();

}}