QB-25:-/*Write a java program to implement the multiple inheritance concepts for calculating area of circle and square*/

```
// Interface for shapes that have area

interface AreaCalculatable
{
    double calculateArea();
}

// Circle class implementing AreaCalculatable interface

class Circle implements AreaCalculatable
{
    private double radius;

    public Circle(double radius)
    {
        this.radius = radius;
    }
    public double calculateArea()
    {
        return Math.PI * radius * radius;
    }
}

// Square class implementing AreaCalculatable interface

class Square implements AreaCalculatable
{
    private double side;

    public Square(double side)
    {
        this.side = side;
    }

    public double calculateArea()
    {
        return side * side;
    }
}
```

```
class AreaCalculator
{
    public static void main(String[] args)
    {
        // Create instances of Circle and Square
        Circle circle = new Circle(5);
        Square square = new Square(4);

        // Calculate and print areas
       System.out.println("Area of Circle: " + circle.calculateArea());
       System.out.println("Area of Square: " + square.calculateArea());
    }
}
```

QB-26:-

Write a program that illustrates interface inheritance. Interface P is extended by P1 And P2. Interface P12 inherits from both P1 and P2.Each interface declares one constant and one method. Class Q implements P12.Instantiate Q and invokes each of its methods. Each method displays one of the constants

```
// Define the interface P
interface P
{
    int CONSTANT_P = 10;
    void methodP();
}

// Define the interface P1 extending P
interface P1 extends P
{
    int CONSTANT_P1 = 20;
    void methodP1();
}

// Define the interface P2 extending P
interface P2 extends P
{
    int CONSTANT_P2 = 30;
    void methodP2();
}

// Define the interface P12 inheriting from both P1 and P2
interface P12 extends P1, P2
```

```
{
    int CONSTANT_P12 = 40;
    void methodP12();
}

// Define class Q implementing P12
class Q implements P12
{
    public void methodP()
    {
     System.out.println("Method P invoked with constant: " + CONSTANT_P);
    }
    public void methodP1()
    {
    System.out.println("Method P invoked with constant: " + CONSTANT_P1);
    }
    public void methodP2()
    {
    System.out.println("Method P invoked with constant: " + CONSTANT_P2);
    }
    public void methodP12()
    {
  System.out.println("Method P12 invoked with constant:"+CONSTANT_P12);
    }
}

 class QB_26
 {
    public static void main(String[] args)
     {
        // Instantiate Q
        Q q = new Q();
        // Invoke methods
        q.methodP();
        q.methodP1();
        q.methodP2();
        q.methodP12();
    }
}
```

QB-27:-/*Create interface Univerity with method takeExam(). Create Interface State with method getName(). Create interface StateUniversity that extends University and State. Create print() inside StateUniversity. Create class TestState that implements StateUniversity

interface. and Create a class withmain method that calls all methods of TestState class */

```
// Interface University

interface University
{
    void takeExam();
}

// Interface State

interface State
{
    String getName();
}

// Interface StateUniversity extending both University and State

interface StateUniversity extends University, State
{
    void print();
}

// Class TestState implementing StateUniversity interface

class TestState implements StateUniversity
{

    public void takeExam()
    {
        System.out.println("Taking exam...");
    }
    public String getName()
    {
        return "State University";
    }
    public void print()
    {
        System.out.println("State University's name: " + getName());
    }
}
 class QB_27
 {
    public static void main(String[] args)
```

```
    {
        TestState testState = new TestState();

        // Calling methods of TestState class
        testState.takeExam();
        testState.print();
    }
}
```

QB-34:-The Transport interface declares a deliver() method. The abstract class Animal is the superclass of the Tiger, Camel, Deer and Donkey classes. The Transport interface is implemented by the Camel and Donkey classes. Write a test program that initialize an array of four Animal objects. If the object implements the Transport interface, the deliver() method is invoked.

```
// Transport interface
interface Transport
{
    void deliver();
}

// Abstract class Animal
abstract class Animal
{
    // Abstract method
    abstract void sound();
}

// Tiger class extending Animal
class Tiger extends Animal
{
    void sound()
    {
        System.out.println("Tiger roars!");
    }
}

// Camel class extending Animal and implementing Transport
class Camel extends Animal implements Transport
{
    void sound()
    {
        System.out.println("Camel grunts!");
```

```
    }

    public void deliver()
    {
        System.out.println("Camel delivers goods.");
    }
}

// Deer class extending Animal
class Deer extends Animal
{
    void sound()
    {
        System.out.println("Deer barks!");
    }
}

// Donkey class extending Animal and implementing Transport
class Donkey extends Animal implements Transport
{
    void sound()
    {
        System.out.println("Donkey brays!");
    }

    public void deliver()
    {
        System.out.println("Donkey carries loads.");
    }
}

class QB_34
{
    public static void main(String[] args)
    {
        // Initialize an array of Animal objects
 Animal[] animals = {new Tiger(), new Camel(), new Deer(), new Donkey()};

         // Iterate over the array and check if the object implements
Transport interface
        // If it does, invoke the deliver() method
        for (Animal animal : animals)
         {
            if (animal instanceof Transport)
            {
```

```
                ((Transport)animal).deliver();
            }
        }
    }
}
```

QB-35:-

Create an interface named Polygon. It has a default method getSides() and an abstract method getArea().create two classes Rectangle and Square that implement Polygon.The Rectangle class provides the implementation of the getArea() method and overrides the getSides() method. the Square class only provides the implementation of the getArea() method.while calling the getSides() method using the Rectangle object, the overridden method is called.However, in the case of the Square object, the default method is called.

```
// Define the interface Polygon
interface Polygon
{
    // Abstract method
    double getArea();

    // Default method
    default int getSides()
    {
        return 4; // Default number of sides for a polygon
    }
}

// Rectangle class implementing Polygon
class Rectangle implements Polygon
{
    double length;
    double width;

    // Constructor
    Rectangle(double length, double width)
    {
        this.length = length;
        this.width = width;
    }

    // Override getArea method
    public double getArea()
```

```
    {
        return length * width;
    }

    // Override getSides method

    public int getSides()
    {
        return Polygon.super.getSides(); // Rectangle has 4 sides
    }
}

// Square class implementing Polygon
class Square implements Polygon
{
    double side;

    // Constructor
    Square(double side)
    {
        this.side = side;
    }

    // Override getArea method

    public double getArea()
    {
System.out.println("Number of sides of Square : " + Polygon.super.getSides());
        return side * side;
    }
}
class QB_35
{
    public static void main(String[] args)
    {

        Rectangle rectangle = new Rectangle(5, 4);

   System.out.println("Number of sides of Rectangle: " + rectangle.getSides());
        System.out.println("Area of Rectangle: " + rectangle.getArea());


        Square square = new Square(5);

        System.out.println("Area of square : " + square.getArea());
```

```
    }
}
```

QB-43:-
Q1) Create an abstract class pen with methods write () and refill () as abstract methods
Q2) Use the pen class from Q1 to create a concrete class fountain pen with additionalmethod change Nib ()
Q3) Create a class monkey with jump () and bite() methods Create a class human Which inherits this monkey class and implements basicanimal interface with eat() and sleepmethods
Q4) Create a class telephone with (), lift () and disconnected () methods as abstract methods create another class smart telephone and demonstrate polymorphism
Q5) Demonstrate polymorphism using using monkey class from Q3
Q6) Create an interface TVremote and use it to inherit another interface smart TVremote
Q7) Create a class TV which implements TVremote interface from Q6

```
// Q1) Create an abstract class Pen
abstract class Pen
{
    // Abstract methods
    abstract void write();
    abstract void refill();
}

// Q2) Create a concrete class FountainPen extending Pen
class FountainPen extends Pen
{
    // Implementation of abstract methods

    void write()
    {
        System.out.println("Writing with Fountain Pen");
    }
    void refill()
    {
        System.out.println("Refilling Fountain Pen");
    }
    // Additional method
```

```
    void changeNib()
    {
        System.out.println("Changing nib of Fountain Pen");
    }
}

// Q3) Create a class Monkey with jump() and bite() methods
class Monkey
{
    void jump()
    {
        System.out.println("Monkey jumps");
    }
    void bite()
    {
        System.out.println("Monkey bites");
    }
}

// Create an interface BasicAnimal
interface BasicAnimal
{
    void eat();
    void sleep();
}

// Q3) Create a class Human which inherits Monkey and implements
BasicAnimal interface
class Human extends Monkey implements BasicAnimal
{
    // Implementation of BasicAnimal interface methods

    public void eat()
    {
        System.out.println("Human eats");
    }
    public void sleep()
    {
        System.out.println("Human sleeps");
    }
}

// Q4) Create an abstract class Telephone
abstract class Telephone
{
```

```
    // Abstract methods
    abstract void dial();
    abstract void lift();
    abstract void disconnect();
}

// Q4) Create another class SmartTelephone inheriting from Telephone
class SmartTelephone extends Telephone
{
    // Implementation of abstract methods
    void dial()
    {
        System.out.println("Dialing a number on Smart Telephone");
    }
    void lift()
    {
        System.out.println("Lifting the receiver of Smart Telephone");
    }
    void disconnect()
    {
        System.out.println("Disconnecting call on Smart Telephone");
    }

    // Additional method
    void browseInternet()
    {
        System.out.println("Browsing the internet on Smart Telephone");
    }
}

// Q5) Demonstrate polymorphism using Monkey class
class PolymorphismDemo
{
    void performActions(Monkey monkey)
    {
        monkey.jump(); // Polymorphic method call
        monkey.bite(); // Polymorphic method call
    }
}

// Q6) Create an interface TVRemote
interface TVRemote
{
    void turnOn();
    void turnOff();
```

```
    void changeChannel(int channel);
}

// Create another interface SmartTVRemote inheriting TVRemote
interface SmartTVRemote extends TVRemote
 {
    void browseInternet();
}

// Q7) Create a class TV which implements TVRemote interface
class TV implements TVRemote
{
    // Implementation of TVRemote interface methods
    public void turnOn()
    {
        System.out.println("TV is turned on");
    }
    public void turnOff()
    {
        System.out.println("TV is turned off");
    }
    public void changeChannel(int channel)
    {
        System.out.println("Changing channel to " + channel);
    }
}
class QB_43
{
    public static void main(String[] args)
    {
        // Q2) Create an instance of FountainPen
        FountainPen fountainPen = new FountainPen();
        fountainPen.write();
        fountainPen.refill();
        fountainPen.changeNib();

         // Q3) Create an instance of Human and demonstrate inherited
methods
        Human human = new Human();
        human.jump();
        human.bite();
        human.eat();
        human.sleep();

        // Q5) Demonstrate polymorphism using Monkey class
```

```
        PolymorphismDemo demo = new PolymorphismDemo();
        Monkey monkey = new Monkey();
        demo.performActions(monkey);

       // Q7) Create an instance of TV and demonstrate TVRemote interface
methods
        TV tv = new TV();
        tv.turnOn();
        tv.changeChannel(5);
        tv.turnOff();

         // Q4) Create an instance of SmartTelephone and demonstrate
polymorphism
        Telephone telephone = new SmartTelephone();
        telephone.dial();
        telephone.lift();
        telephone.disconnect();

         // We can't directly call browseInternet method as it's not
available in Telephone class.
        // However, we can create a SmartTelephone reference to access
browseInternet method
        SmartTelephone smartTelephone = (SmartTelephone) telephone;
        smartTelephone.browseInternet();
    }
}
```