Exercise 01:

Declare an interface called “MyFirstInterface”. Decalre integer type variable called “x”. Declare an abstract method called “display()”.

1. Try to declare the variable with/without public static final keywords. Is there any difference between these two approaches? Why?
2. Declare the abstract method with/without abstract keyword. Is there any difference between these two approaches? Why?
3. Implement this into a class called “IntefaceImplemented” . Override all the abstract methods. Try to change the value of x inside this method and print the value of x. Is it possible for you to change x? why?

**public interface MyFirstInterface{**

**int x=11;**

**void display();**

**}**

**public interface MyFirstInterface{**

**void display();**

**}**

**There is not difference between declaring the variable x with or without.**

**Because the public static final keywords in an interface, In Java interfaces, all variables are implicitly public, static, and final, whether explicitly declare them as such or not.**

**So, both approaches are equivalent, and the variable x will be treated as a public, static, and final variable in the MyFirstInterface.**

**......................................................................................**

**public interface MyInterface {**

**void method1();**

**abstract void method2();**

**}**

**There is no difference between declaring the abstract method display() with or without the abstract keyword in an interface. All methods defined in an interface are implicitly abstract. So, both approaches are equivalent, and the display() method in the MyFirstInterface will be treated as an abstract method.**

**...........................................................**

**public class InterfaceImplemented implements MyfirstInterface{**

**@Override**

**public viod display(){**

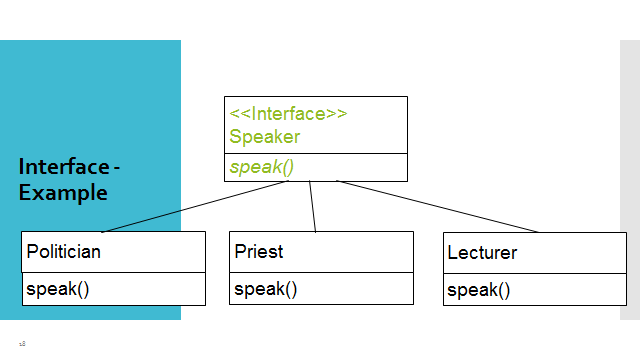
**x=11;**

**System.out.println("value of X :"+x);**

**In the InterfaceImplemented class, when we try to change the value of x inside the display() method, it will result in a compilation error. This is because the variable x in the interface is implicitly final, and final variables cannot be reassigned new values once they are initialized. So, trying to modify the value of x inside the implementing class will lead to a compile-time error.**

Exercise 02:

Develop a code base for the following scenario. Recall what we have done at the lecture…



**package com.mycompany.practical;**

**public interface Speaker {**

**public void speak();**

**}**

**.........................................................................**

**package com.mycompany.practical;**

**public class politician implements Speaker {**

**@Override**

**public void speak(){**

**System.out.println("Politician'speaking:" );**

**}}**

**.................................................................................**

**package com.mycompany.practical;**

**public class priest implements Speaker{**

**@Override**

**public class speak(){**

**System.out.println("Priest speaking");{**

**}}**

**.........................................................................**

**package com.mycompany.practical;**

**public class lecuter implements Speaker{**

**@Override**

**public class speak(){**

**System.out.println("Lecture speaking");{**

**}}}**

**............................................................................**

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

**priest p= new priest();**

**p.speak();**

**politician po=new politician();**

**po.speak();**

**lecturerr le= new lecturerr();**

**le.speak();**

**}**

**}**

**OUTPUT=====> Politician'speaking:**

**Priest speaking:**

**Lecture speaking:**

**Exercise 03:**

**Try following code. What is the outcome? Why?**

**Class 01: Class 02:**

**final class Student { class Undergraduate extends Student{}**

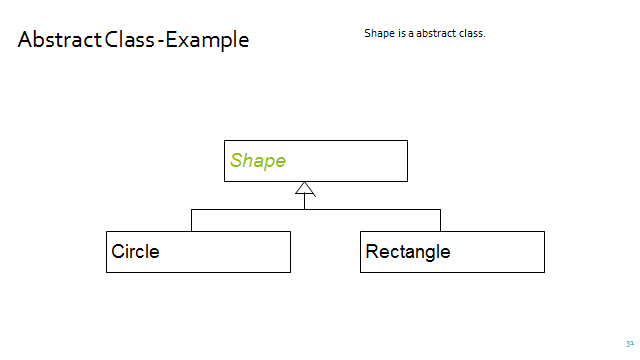
**final int marks = 100;**

**final void display();**

**}**

Exercise 04:

Develop a code base for the following scenario. Shape class contains an abstract method called “calculateArea” and non-abstract method called “display”. Try to pass required values at the instantiation. Recall what we have done at the lecture…



**Student class:**

**That class is mark as final. That means can not change any things**

**It has a final instance variable marks. It set up as 100 and cannot be changed once initialized.**

**It has a final method display(), which means it cannot be overridden by any subclass.**

**Undergraduate class:**

**Undergraduate class extend to the Student class. but since Student is marked as final, it cannot be subclassed.**

**The code will not compile.student class use final keyword,therefore it cannot be overriden**

**in any subclass.The error message will be similar to "cannot inherit from undergraduate.**

**<....................................Question No 04......................................>**

**public abstract class Shape {**

**public abstract double calculateArea();**

**public void display() {**

**System.out.println("Area: " + calculateArea());**

**}}**

**...............................................................**

**public class Circle extends Shape {**

**private double radius;**

**public Circle(double radius) {**

**this.radius = radius;**

**}**

**@Override**

**public double calculateArea() {**

**return 3.143 \* radius \* radius;**

**}}**

**............................................................**

**public class Rectangle extends Shape {**

**private double length;**

**private double width;**

**public Rectangle(double length, double width) {**

**this.length = length;**

**this.width = width;**

**}**

**@Override**

**public double calculateArea() {**

**return length \* width;**

**}}**

**................................................................**

**public class TestShape {**

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

**Circle circle = new Circle(5.0);**

**circle.display();**

**Rectangle rectangle = new Rectangle(4.0, 6.0);**

**rectangle.display();**

**}}**

**OUTPUT======> circle**

**Area :78.575**

**Rectangle**

**Area: 24.0**