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?
4. **interface MyFirstInterface {**

**int x = 10; // Implicitly public, static, and final**

**}**

**The variable x in the interface is automatically public, static, and final. You can access it using the interface name like MyFirstInterface.x.**

**Adding public static final keywords won't make any difference because they are already implied.**

1. **interface MyFirstInterface {**

**void display(); // Implicitly abstract**

**}**

**The method display() in the interface is automatically abstract. Adding the abstract keyword won't make any difference because it's already implied.**

**class InterfaceImplemented implements MyFirstInterface {**

**@Override**

**public void display() {**

**// You can't change the value of 'x' here**

**// 'x' is implicitly 'final' and attempting to modify it will result in a compilation error**

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

**}**

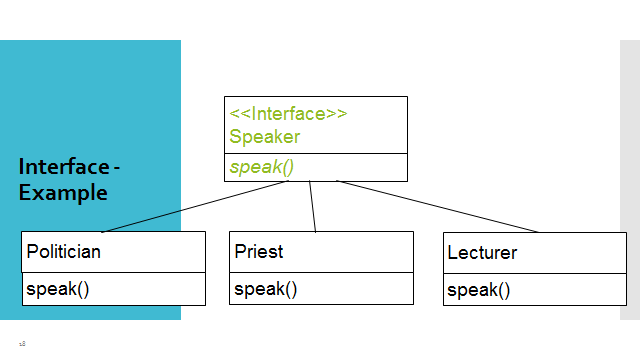
**}**

**Regarding changing the value of x inside the implementing class's method, you can't do that. The variable x declared in the interface is implicitly final. It's a constant, and Java does not allow you to change the value of a constant variable.**

**If you try to change the value of x inside the display() method, you will get a compilation error like "Cannot assign a value to final variable 'x'".**

Exercise 02:

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



**interface Speaker {**

**void speak();**

**}**

**class Politician implements Speaker {**

**public void speak() {**

**System.out.println("I am a politician and I speak to persuade people.");**

**}**

**}**

**class Priest implements Speaker {**

**public void speak() {**

**System.out.println("I am a priest and I speak to give guidance and comfort.");**

**}**

**}**

**class Lecturer implements Speaker {**

**public void speak() {**

**System.out.println("I am a lecturer and I speak to teach people.");**

**}**

**}**

**public class Main {**

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

**Speaker politician = new Politician();**

**politician.speak();**

**Speaker priest = new Priest();**

**priest.speak();**

**Speaker lecturer = new Lecturer();**

**lecturer.speak();**

**}**

**}**

**Exercise 03:**

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

**Class 01: Class 02:**

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

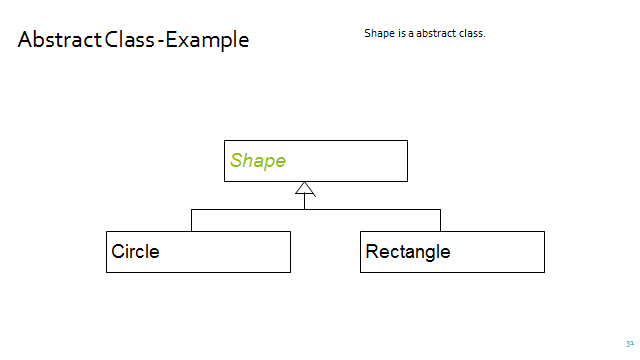
**final int marks = 100;**

**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…



**abstract class Shape {**

**abstract double calculateArea();**

**public void display() {**

**System.out.println("I am a shape");**

**}**

**}**

**class Square extends Shape {**

**private double side;**

**public Square(double side) {**

**this.side = side;**

**}**

**@Override**

**double calculateArea() {**

**return side \* side;**

**}**

**}**

**class Circle extends Shape {**

**private double radius;**

**public Circle(double radius) {**

**this.radius = radius;**

**}**

**@Override**

**double calculateArea() {**

**return 3.14 \* radius \* radius;**

**}**

**}**

**public class Main {**

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

**Square square = new Square(5);**

**System.out.println("The area of the square is: " + square.calculateArea());**

**Circle circle = new Circle(2);**

**System.out.println("The area of the circle is: " + circle.calculateArea());**

**}**

**}**