Practical 05: Inheritance & Abstract Classes

Exercise 01:

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

```
// MyFirstInterface.java
public interface MyFirstInterface {
   int x = 0; // Integer type variable declaration (implicitly public, static, and final)
   // Abstract method declaration
   void display();
}
```

1. Try to declare the variable with/without public static final keywords. Is there any difference between these two approaches? Why?

   ```
   // MyFirstInterface.java
   public interface MyFirstInterface {
      int x = 0; // Declared without any access modifiers (implicitly public, static, and final)
   }
   ```

2. Declare the abstract method with/without abstract keyword. Is there any difference between these two approaches? Why?

   ```
   // Approach 1: Declaring abstract method with 'abstract' keyword
   public interface MyFirstInterface {
      abstract void display();
   }

   // Approach 2: Declaring abstract method without 'abstract' keyword
   public interface MyFirstInterface {
      void display();
   }
   ```

   In conclusion, whether you use the abstract keyword or not when declaring methods in an interface, there is no difference in how the code behaves.

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?

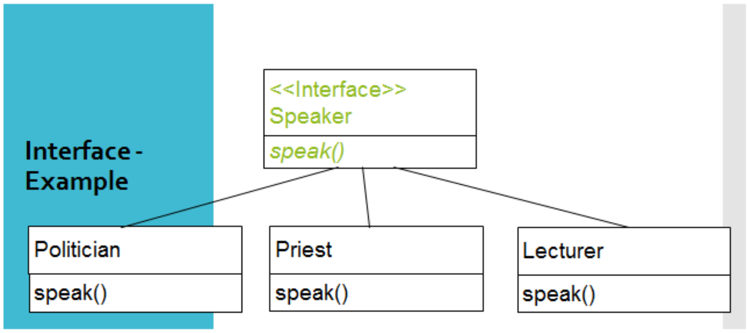
   ```
   public class InterfaceImplemented implements MyFirstInterface {
      int x = 5; // Non-static variable x in the implementing class
   ```

```
    @Override
    public void display() {
      x = 10; // Changing the value of x
      System.out.println("Inside display() method - x: " + x);
    }

    public static void main(String[] args) {
      InterfaceImplemented obj = new InterfaceImplemented();
      obj.display(); // Output: Inside display() method - x: 10
      System.out.println("After calling display() method - x: " + obj.x); // Output: After calling
  display() method - x: 10
    }
  }
```

Exercise 02:

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



```
// Interface for speakers
interface Speaker {
  void speak();
}
// Class for politicians
class Politician implements Speaker {
  @Override
  public void speak() {
    System.out.println("I am a politician, and I am speaking.");
  }
}
// Class for priests
class Priest implements Speaker {
  @Override
  public void speak() {
    System.out.println("I am a priest, and I am speaking.");
  }
}
// Class for lecturers
class Lecturer implements Speaker {
  @Override
  public void speak() {
    System.out.println("I am a lecturer, and I am speaking.");
  }
}
// Main method
public class Main {
  public static void main(String[] args) {
    // Create a politician
    Politician politician = new Politician();
    // Create a priest
    Priest priest = new Priest();
    // Create a lecturer
    Lecturer lecturer = new Lecturer();
    // Make the politician speak
    politician.speak();
    // Make the priest speak
    priest.speak();
    // Make the lecturer speak
    lecturer.speak();
  }
}
```

<u>Exercise 03:</u>

Try following code. What is the outcome? Why?

Class 01:

```
final class Student {

        final int marks = 100;

        final void display();

}
```

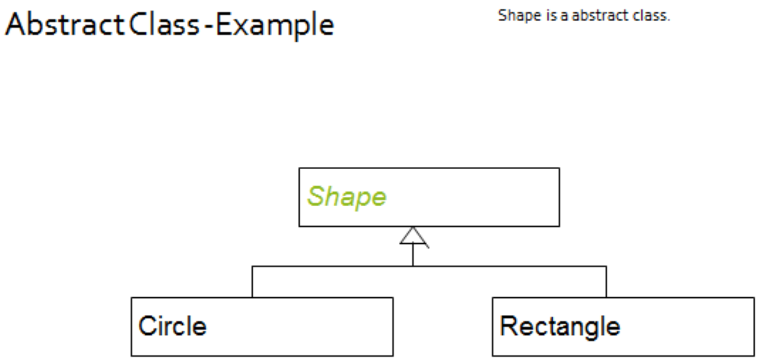
Class 02:

```
class Undergraduate extends Student{}
```

The outcome of this code will be a compilation error due to the attempt to extend a final class (Student) with the Undergraduate class. The final modifier prevents inheritance and subclassing of the Student class, so it cannot be used as a superclass for other classes.

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-Example

Shape is a abstract class.

```
import java.lang.Math;
abstract class Shape {
  protected String name;
  protected double length;
  protected double width;

  public Shape(String name, double length, double width) {
    this.name = name;
    this.length = length;
    this.width = width;
  }

  abstract double calculateArea();

  public void display() {
    System.out.println("The area of " + name + " is " + calculateArea());
  }
}

class Rectangle extends Shape {
  public Rectangle(String name, double length, double width) {
    super(name, length, width);
  }

  @Override
  double calculateArea() {
    return length * width;
  }
}

class Circle extends Shape {

  public Circle(String name, double radius) {
    super(name, radius, radius);
  }

  @Override
  double calculateArea() {
    return Math.PI * radius * radius;
  }
}

public class Main {

  public static void main(String[] args) {
    Rectangle rectangle = new Rectangle("Rectangle", 10, 20);
    rectangle.display();

    Circle circle = new Circle("Circle", 10);
    circle.display();
  }
}
```