OOP using JAVA – Practical 05

ID – 28865

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?

**• When you declare a variable in an interface without any access modifiers (e.g., public, private, protected), it is implicitly considered as public static final. So, whether you explicitly mention the keywords or not, the variable is still public, static, and final.**

**• The public keyword allows the variable to be accessed from anywhere.**

**• The static keyword makes the variable shared among all instances of the class implementing the interface.**

**• The final keyword makes the variable constant, meaning its value cannot be changed**

**once assigned. Therefore, there is no difference between declaring the variable with/without public static final keywords in an interface**

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

**• In an interface, all methods are implicitly abstract, even if you don't explicitly use the abstract keyword. So, whether you include the abstract keyword or not, the method is still abstract.**

**• The abstract keyword indicates that the method does not have a body and needs to be implemented by the class implementing the interface. Therefore, there is no difference between declaring the abstract method with/without the abstract keyword in an interface**

1. 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?

**• To implement the interface, you would create a class, let's say "InterfaceImplemented," and use the "implements' keyword to implement the "MyFirstInterface."**

**In the "Interfacelmplemented" class, you would need to override the abstract method "display0" from the interface. However, you cannot override the value of the interface variable "x" because it is declared as final in the interface.**

**The final keyword makes the variable constant, and its value cannot be changed once assigned. So, any attempt to modify the value of "x" will result in a compilation error. • Here's an example implementation of the "Interface implemented" class**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**package com.company.textinterface;**

**public interface MyFirstInterface {**

**int x = 10;**

**void display();**

**}**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**package com.company.textinterface;**

**public class InterfaceImplemented implements MyFirstInterface {**

**public void display(){**

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

**}**

**}**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**package com.company.textinterface;**

**public class TextInterface {**

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

**InterfaceImplemented I = new InterfaceImplemented();**

**I.display();**

**}**

**}**

Exercise 03:

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 Students{}

**final class Student{**

**final int marks = 100;**

**final void display(){**

**System.out.println(“Hellow Tharuka”);**

**}**

**Class Undergraduate extends Students{}**

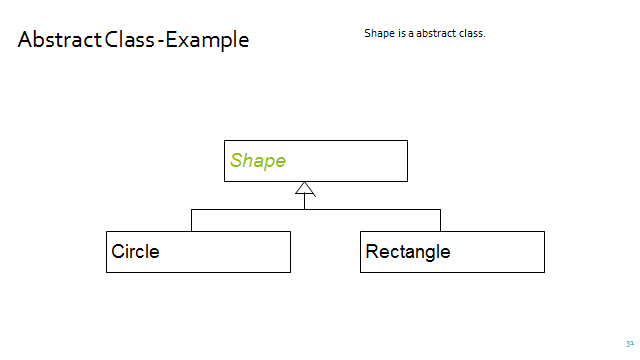
**}**

The “Student” class is declared as “final”, which means it can not be subclassed. It also has a final instance variable “marks” initialized to 100 and a final method “display()” which print the value of “marks”. The final keyword ensures that these members can not be overridden or modified in any subclasses.

The “Undergraduate” class extends the “Student” class. However, since the “Student" class is declared as “final” , it can not be subclassed. Therefore code will produce a compilation error.

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…

  
package com.company.mainshape;

public abstract class Shape {

abstract double calculateArea();

public void display(){

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

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

package com.company.mainshape;

public class Rectangle extends Shape{

private double length,width;

public Rectangle(double length,double width){

this.length = length;

this.width = width;

}

double calculateArea(){

return length \* width;

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

package com.company.mainshape;

public class Circle extends Shape {

private double radius;

public Circle(double radius){

this.radius = radius;

}

double calculateArea(){

return Math.PI \* radius \*radius;

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

package com.company.mainshape;

public class MainShape {

public static void main(String[] args) {

Rectangle R1 = new Rectangle(6.00,9.00);

R1.calculateArea();

R1.display();

Circle C1 = new Circle(7.00);

C1.calculateArea();

C1.display();

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

package com.company.mainshape;

public class MainShape {

public static void main(String[] args) {

Shape S1 = new Rectangle(6.00,9.00);

S1.calculateArea();

S1.display();

Shape S2 = new Circle(7.00);

S2.calculateArea();

S2.display();

}

}