Name : W. A. C. Fernando
Student ID : 26545

Practical 05

1.

i. Interface variables are necessarily public, static, and final. As a result, they do not need to be specified explicitly using these phrases. Both methods will get the same outcomes. Adding the terms public static final to the variable definition, on the other hand, has no effect in this circumstance.

ii. All interface methods are implicitly abstract. As a result, they do not need to be stated explicitly using the abstract keyword. Both methods will get the same outcomes. Adding the abstract keyword directly to the method specification, on the other hand, has no effect in this circumstance.

iii. The interface variable x is implicitly final, which means that once assigned, its value cannot be changed. Even if you may access x via the show() function of the implementing class, any effort to alter its value will result in a compilation error. As a result, it is not feasible to change the value of x within the implementing class.

2.

```
package com.mycompany.speakmain;

public interface Speaker {
   void speak();
}

public class Politician implements Speaker{

   @Override
   public void speak()
   {
      System.out.println("Politician speaking...");
   }
}

public class Priest implements Speaker{

   @Override
   public void speak()
   {
      System.out.println("Priest speaking...");
   }
}

public class Lecturer implements Speaker{
    static Lecturer lecturer;

    @Override
    public void speak()
    {
        System.out.println("Lecturer speaking...");
    }
}

package com.mycompany.speakmain;

import static com.mycompany.speakmain.Lecturer.lecturer;

public class SpeakMain {

    public static void main(String[] args) {
        Politician politician=new Politician();
        politician.speak();

        Priest priest=new Priest();
        priest.speak();

        Lecturer.lecturer=new Lecturer();
        lecturer.speak();
    }
}
```

3. Class 01's Student class is designated final, which implies it cannot be inherited by any other class. Furthermore, the show() function is tagged as final, which indicates that it cannot be overridden by a subclass.

Class 02 attempts to inherit from the Student class, but it cannot be extended since Student is marked as final. As a result, a compilation error will occur.

The final keyword is used to restrict class inheritance as well as method and variable overrides. A class or method that has been marked final cannot be subclassed or overridden.

To remedy the compilation issue, we may either delete the final keyword from the Student class or remove the inheritance attempt in the Undergraduate class, depending on our planned design.

4.
```
package com.mycompany.shapemain;

abstract class Shape {
    abstract double calculateArea();

    void display()
    {
        System.out.println("Displaying shape...");
    }
}

public class Circle extends Shape{
    private double radius;

    public Circle(double radius)
    {
        this.radius=radius;
    }

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

public class Rectangle extends Shape{
    private double length,width;

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

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

public class ShapeMain {

    public static void main(String[] args) {
        Circle circle=new Circle(5.0);
        System.out.println("Area of a circle: "+circle.calculateArea());
        circle.display();

        Rectangle rectangle=new Rectangle(3,4);
        System.out.println("Area of a rectangle: "+rectangle.calculateArea());
        circle.display();
    }
}
```