**Exercise 1**

1. An interface's specified variables are immediately viewed as public, static, and final. Therefore, using the terms public, static, and final directly with respect to interface variables is discouraged by the Java compiler as being redundant.

If you try to declare the variable with these keywords inside of an interface, a compilation error will happen.

// This is NOT allowed in an interface

public interface MyFirstInterface {

public static final int x = 10; // Error: Illegal modifier for the interface field x; only public, static & final are permitted

}

1. A method is automatically assumed to be abstract when declared inside of an interface. Because of this, the Java compiler considers it redundant and disallowed to explicitly use the abstract keyword with interface methods.

There will be a compilation issue if you try to declare the method with the abstract keyword inside of an interface.

// This is NOT allowed in an interface

public interface MyFirstInterface {

abstract void display(); // Error: Illegal modifier for the interface method display(); only public & abstract are permitted

}

1. A constant variable x (implicitly static, final, and public) and an abstract method display() are both included in the MyFirstInterface.

MyFirstInterface interface is implemented by the InterfaceImplemented class, which also offers a display() function implementation.

We are unable to alter the value of x inside the display() method since it is marked as final in the interface. Once they are initialized, final variables cannot be given new values. A compilation error will occur if you try to modify the value of x inside the display() method.

It's crucial to remember that interface variables are implicitly static, final, and public, and that interface methods are inherently abstract and public. There will be compilation issues if these keywords are used explicitly with interface members. Final variables cannot be changed after initialization, regardless of whether they are contained within a method of the implementing class.

**MyFirstinterface.java**

public interface MyFirstInterface {

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

void display(); // Implicitly abstract

}

**InterfaceImplemented.java**

public class InterfaceImplemented implements MyFirstInterface {

@Override

public void display() {

// Attempting to change the value of x inside the method

// This will cause a compilation error since x is final

// x = 20; // Error: cannot assign a value to final variable x

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

}

}

**Exercise 3**

final class Student {

final int marks = 100;

final void display();

}

class Undergraduate extends Student {}

Class 01:Student

This class is deemed to have finished. A final class cannot be subclassed, which prevents you from deriving new classes from it.

Marks is a final instance variable for the class, which means that once it has been given a value, it cannot be modified.

Another final method for the class is display(). There is no way to override a final method in a subclass.

class 02: Undergraduate,

The Student class has been declared final, therefore this class's attempts to extend it are unsuccessful.

Student cannot be subclassed because it is a final class.