Exercise 01:

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

public interface MyFirstInterface {

int x = 0;

void display();

}

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

\\\\\\\\ with public static final:

public interface MyFirstInterface {

public static final int x = 0;

void display();

}

\\\\\\\\ without public static final:

public interface MyFirstInterface {

int x = 0;

void display();

}

In Java interfaces, variables are implicitly public, static, and final. It means that both of the above declarations are equivalent. Regardless of whether you explicitly write `public static final` or just declare the variable without those keywords, the compiler treats them the same way.

The interface variables are public because they are meant to be accessed from any class that implements the interface.

The variables are static because they belong to the interface itself rather than any specific instance of a class that implements the interface. Therefore, you can access these variables using the interface name, like MyFirstInterface.x.

The variables are final because, in an interface, you can only define constants (values that cannot be changed). Once you assign a value to the variable, you cannot modify it later.

So, in summary, using `public static final` explicitly in the interface variable declaration is redundant, and Java allows you to omit those keywords. The compiler treats them as constants by default.

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

\\\\\\\\ without the abstract keyword

public interface MyFirstInterface {

void display();

}

\\\\\\\\ with the abstract keyword

public interface MyFirstInterface {

abstract void display();

}

In both cases, the `display()` method is abstract, and any class that implements the `MyFirstInterface` interface must provide an implementation for this method. An abstract method is a method without a method body, and its implementation is provided by the implementing class.

So, using the `abstract` keyword in an interface method declaration is optional, but it's common practice to omit it, as it is implicitly understood that all methods declared in an interface are abstract by default.

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?

public class InterfaceImplemented implements MyFirstInterface {

@Override

public void display() {

// Implementation of the display() method

System.out.println("Inside display() method.");

// Attempting to change the value of x

x = 10; // This will result in a compilation error

}

public static void main(String[] args) {

InterfaceImplemented obj = new InterfaceImplemented();

obj.display();

}

}

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();

}

The provided code will not compile successfully due to an error in the ‘Student’ class definition.

The issue lies in the ‘Student’ class's method declaration