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

Main class

package com.mycompany.interface2;

public class Interface2

{

public static void main(String[] args)

{

IntefaceImplemented i1=new IntefaceImplemented();

i1.display();

}

}

Interface

package com.mycompany.interface2;

public interface MyFirstInterface

{

public static final int x=10;

abstract void display();

}

IntefaceImplemented

package com.mycompany.interface2;

public class IntefaceImplemented implements MyFirstInterface

{

public void display()

{

System.out.println("x="+x);

}

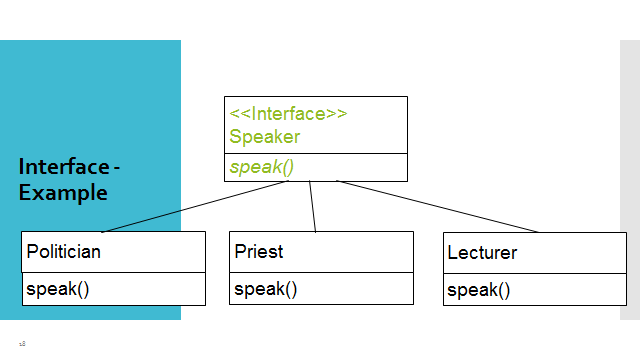
}

Answers

1. There is no need to use public static final keyword. Because in interface all variables are implicitly public, static, and final. So, when you declare a variable inside an interface, it is by default treated as public static final. Therefore, adding public static final explicitly won't make any difference; it's redundant.
2. When you declare a method inside an interface, it is implicitly abstract, even if you don't use the abstract keyword. So, adding abstract explicitly is also redundant. All interface methods are abstract by default, meaning they have no implementation in the interface itself, and the implementing classes must provide the method's implementation.
3. when trying to change the value of x inside the display() method, a compilation error will occur because x is declared as a constant (public static final). Constants cannot be reassigned a new value once they are initialized

**Exercise 02:**

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

****

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();

}

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…

