Practical 06: Inheritance & Abstract Classes

Exercise 01:

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

```
interface MyFirstInterface {

   int x = 10;

   void display();

}
```

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

No, there is no difference between declaring the variable x with or without the public static final keywords. In an interface, all variables are implicitly public, static, and final, so the two approaches are equivalent.

```
interface MyFirstInterface {

   public static final int x = 10;

   void display();

}
```

```
interface MyFirstInterface {

   int x = 10;

   void display();

}
```

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

No, there is no difference between declaring the display method with or without the abstract keyword. In an interface, all methods are implicitly abstract and public, so the two approaches are equivalent.

```
interface MyFirstInterface {

   int x = 10;

   abstract void display();

}
```

```
interface MyFirstInterface {

   int x = 10;

   void display();

}
```

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?

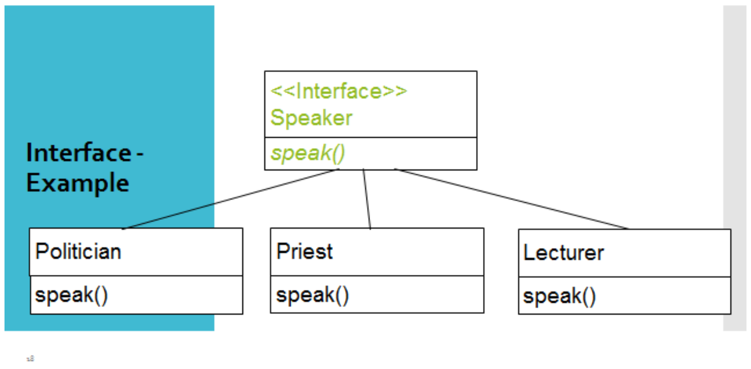
No, it is not possible to change the value of x in the display method of the InterfaceImplemented class because x is a final constant and cannot be reassigned.

```
class InterfaceImplemented implements MyFirstInterface {

   public void display() {

      // x = 20; // This line would cause a compile-time error

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

   }

}
```

Practical 06: Inheritance & Abstract Classes

Exercise 02:

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



```
interface Speaker {

   void speak();

}


class Politician implements Speaker {

   @Override

   public void speak() {

      System.out.println("I am a politician and I am speaking.");

   }

}


class Priest implements Speaker {

   @Override

   public void speak() {

      System.out.println("I am a priest and I am speaking.");
```

```
    }
}

class Lecturer implements Speaker {
    @Override
    public void speak() {
        System.out.println("I am a lecturer and I am speaking.");
    }
}

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

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

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