OOAD Project 2

Michael Metz, Sam Goto

Version 9, 2022

Table of Contents

Part One

1. What are three ways modern Java interfaces differ from a traditional OO interface design that describes only function signatures and return types to be implemented? Provide a Java code example of each.

   Java 8 introduced default method and static method along with lamda expressions.

   - *(Note: Interface Segregation Principle)*
   - *(Note: Single Responsibility Principle)*
   - **Default Method** - Allows its implementation classes to invoke the method without implementing it in the class, and it can be overridden.
   - **Static Method** - Allows its implementation classes to invoke the method directly from the interface, however it cannot be overridden.
   - **Lamda expresions** - Allows to define functional interfaces in line and it can be called without implementing the interface.

   Defaut and Static methods Example

```
interface Animal {
   // Default method: This can be invoked in implementing classes, and can be
overridden
   default void makeSound() {
      System.out.println("###");
   }

   // Static method: This method cannot be overridden
   static void eat() {
      System.out.println("Yum yum...");
   }
}
```

   Lamda expressions example

```
// Lamda expressions
interface functionalInterface {
   void speak(String name);
}

class Example {
   public static void main(String[] args) {
      // define the functional interface
      functionalInterface example = (String name)->System.out.println("Hi, I am " +
name);
      // call the function
      example.speak("Sam");
   }
}
```

2. Describe the differences and relationship between abstraction and encapsulation. Provide a Java code example that illustrates the difference.

   The main difference between abstraction and encapsulation is that abstraction is used to only show relevant information to hide implementation details whereas encapsulation is used to hide and protect data by limiting access to some specific data usually using getters and setters. The relationship between abstraction and encapsulation is that encapsulation supports abstraction as it helps hide specific information.

   Code Example

```
// --------------Abstraction---------------
abstract class Computer {
   ...
   // enforces the child classes to have the Boot method
   public void Boot() {
      // Do stuff to boot up the computer
   }
   ...
}

public class MacBook extends Computer {
   // extending the Computer abstract class, thus the Boot method can be called
   Boot();
}



// -------------Encapsulation--------------
public class Student() {
   // encapsulate the name, thus it'll only be accesible from the getter and setter
defined below
   private String name;

   /**
   * Constructor
   * @param name
   */
   Student(String name) {
      this.name = name;
   }



   /**
   *
   * @return the name
   */
   public String getName() {
      return name;
   }
   /**
   *  sets the name
   *  @param name
   */
   public void setName(String name) {
      this.name = name;
   }
}
```