

Federal Urdu University of Arts, Sciences & Technology, Islamabad

Object Oriented Programming LAB

Assignment 1: OOP LAB (JAVA)

Mr. Muhammad Waseem Malik
1/23/2022

Contents

Key Dates and Details	2
1. Overview:	3
1.1. Encapsulation.....	3
1.2. Encapsulation Program Example	3
1.3. Inheritance	4
1.4. Inheritance Program Example.....	4
1.5. Polymorphism	4
1.6. Polymorphism Program Example.....	5
1.7. Abstract Classes and Methods.....	5
1.8. Abstract Program Example	6
1.9. Interface Program Example	6
1.10. Interface Program Example	7
2. Assignment.....	7
2.1. Task 1 (10)	7
2.2. Task 2 (5)	8
2.3. Task 3 (10)	8
2.4. Task 4 (5):	9

Key Dates and Details

Subject:	OOP LAB
Class:	BSCS 2 nd C
Assessment Type:	Assignment Part 1
Marks:	30
Assessment Weighting:	15%
Submission Method:	Google Class Room
Date Set:	20-JAN-2022
Submission Date:	11:59pm, 23-JAN-2022
Provisional Feedback Release Date:	02:00pm, 25-JAN-2022

1. Overview:

Java is an object oriented language and some concepts may be new. Take breaks when needed, and go over the examples as many times as needed.

1.1. Encapsulation

Encapsulation is one of the four fundamental OOP Concepts. The other three are inheritance, polymorphism and abstraction. Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding.

To achieve encapsulation in Java –

- Declare the variables of a class as private.
- Provide public setter and getter methods to modify and view the variables values.

1.2. Encapsulation Program Example

//File name EncapExample

```
public class EncapExample {
    private String name;
    private int age;

    public int getAge() {
        return age;
    }
    public String getName() {
        return name;
    }
    public void setAge( int newAge) {
        age = newAge;
    }

    public void setName(String newName) {
        name = newName;
    }
}
```

//File Name MainClass

```
public class MainClass{

    public static void main(String args[]) {
        EncapExample en = new EncapExample ();
        en.setName("Asim");
        en.setAge(20);
        System.out.print("Name: " + en.getName() + "Age: " + en.getAge());
    }
}
```

1.3. Inheritance

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class). extends Keyword **extends** is the keyword used to inherit the properties of a class.

Following is the syntax of extends keyword.

```
class Super {
    .....
    .....
}
class Sub extends Super {
    .....
    .....
}
```

1.4. Inheritance Program Example

```
class Vehicle {
    protected String brand = "Ford";
    public void honk() {
        System.out.println("Tuut, tuut!");
    }
}

class Car extends Vehicle {
    private String modelName = "Mustang";
    public static void main(String[] args) {

        Car myCar = new Car();

        myCar.honk();
        System.out.println(myCar.brand + " " + myCar.modelName);
    }
}
```

1.5. Polymorphism

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object. Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance. Like we specified in the previous chapter; Inheritance lets us inherit attributes and methods from another class.

Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways. For example, think of a superclass called `Animal` that has a method called `animalSound()`.

Subclasses of `Animals` could be `Pigs`, `Cats`, `Dogs`, `Birds` - And they also have their own implementation of an animal sound (the pig oinks, and the cat meows, etc.):

1.6. Polymorphism Program Example

```
public class Animal {
    public void animalSound() {
        System.out.println("The animal makes a sound");
    }
}

public class Pig extends Animal {
    public void animalSound() {
        System.out.println("The pig says: wee wee");
    }
}

class Dog extends Animal {
    public void animalSound() {
        System.out.println("The dog says: bow wow");
    }
}

class Main {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();
        Animal myPig = new Pig();
        Animal myDog = new Dog();
        myAnimal.animalSound();
        myPig.animalSound();
        myDog.animalSound();
    }
}
```

1.7. Abstract Classes and Methods

Data abstraction is the process of hiding certain details and showing only essential information to the user. Abstraction can be achieved with either abstract classes or interfaces (which you will learn more about in the next section).

The `abstract` keyword is a non-access modifier, used for classes and methods:

- Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

An abstract class can have both abstract and regular methods:

1.8. Abstract Program Example

```
abstract class Animal {
    // Abstract method (does not have a body)
    public abstract void animalSound();
    // Regular method
    public void sleep() {
        System.out.println("Zzz");
    }
}

class Pig extends Animal {
    public void animalSound() {
        System.out.println("The pig says: wee wee");
    }
}

public class Main {
    public static void main(String[] args) {
        Pig myPig = new Pig(); // Create a Pig object
        myPig.animalSound();
        myPig.sleep();
    }
}
```

1.9. Interface Program Example

An interface is a completely "abstract class" that is used to group related methods with empty bodies:

```
interface Animal {
    public void animalSound();
    public void run();
}
```

To access the interface methods, the interface must be "implemented" (kinda like inherited) by another class with the implements keyword (instead of extends). The body of the interface method is provided by the "implement" class:

1.10. Interface Program Example

```
interface Animal {
    public void animalSound(); // interface method (does not have a body)
    public void sleep(); // interface method (does not have a body)
}

// "implements" the Animal interface
class Pig implements Animal {
    public void animalSound() {
        // The body of animalSound() is provided here
        System.out.println("The pig says: wee wee");
    }
    public void sleep() {
        // The body of sleep() is provided here
        System.out.println("Zzz");
    }
}

public class Main {
    public static void main(String[] args) {
        Pig myPig = new Pig(); // Create a Pig object
        myPig.animalSound();
        myPig.sleep();
    }
}
```

2. Assignment

An overview of Java Programming concept is given in the previous sections. Understand these few examples well and solve the following questions.

2.1. Task 1 (10)

Create an Encapsulated class **Account** with **accountTitle**, **balance** as **data member**. Create one parameterized **constructor** to set data members value and two parameterized methods **withdraw** and **deposit** balance (hint: set method). In withdraw method subtract the amount from balance and in the deposit method add the amount in balance. Create another method with name of **currentBalance** (hint: get method) to display accountTitle and balance.

In the **Main** class create two objects of account class and pass the different values. First call the currentBalance method to view the detail, then calls the deposit method to add amount and again call the currentBalance method. In the next step call the withdraw method and call again the currentBalance method. Repeat the same activity with second object and run the program.

2.2. Task 2 (5)

Create a **SavingsAccount** class. Use a static data member **annualInterestRate** to store the annual interest rate for each of the savers. Each member of the class contains a private data member **savingsBalance** indicating the amount the saver currently has on deposit. Provide member function **calculateMonthlyInterest** that calculates the monthly interest by multiplying the balance by **annualInterestRate** divided by 12; this interest should be added to **savingsBalance**. Provide a static member function **modifyInterestRate** that sets the static **annualInterestRate** to a new value. Write a driver program to test class **SavingsAccount**. Instantiate two different objects of class **SavingsAccount**, **saver1** and **saver2**, with balances of \$2000.00 and \$3000.00, respectively. Set the **annualInterestRate** to 3 percent. Then calculate the monthly interest and print the new balances for each of the savers. Then set the **annualInterestRate** to 4 percent, calculate the next month's interest and print the new balances for each of the savers.

2.3. Task 3 (10)

Create a class named Pizza that stores information about a single pizza. It should contain the following:

- Private instance variables to store the size of the pizza (either small, medium, or large), the number of cheese toppings, the number of pepperoni toppings, and the number of ham toppings.
- Constructor(s) that set all of the instance variables
- Public methods to get and set the instance variables.
- A public method named `calcCost()` that returns a double that is the cost of the pizza.
- Pizza cost is determined by:
 - Small: \$10 + \$2 per topping
 - Medium: \$12 + \$2 per topping
 - Large: \$14 + \$2 per topping
- Public method named `getDescription()` that returns a String containing the pizza size, quantity of each topping.

Write test code to create several pizzas and output their descriptions. For example, a large pizza with one cheese, one pepperoni and two ham toppings should cost a total of \$22. Now Create a **PizzaOrder** class that allows up to three pizzas to be saved in an order. Each pizza saved should be a **Pizza** object. Create a method `calcTotal()` that returns the cost of order.

In the runner order two pizzas and return the total cost.

2.4. Task 4 (5):

An abstract class has a constructor which prints "This is constructor of abstract class", an abstract method named 'a_method' and a non-abstract method which prints "This is a normal method of abstract class". A class 'SubClass' inherits the abstract class and has a method named 'a_method' which prints "This is abstract method". Now create an object of 'SubClass' and call the abstract method and the non-abstract method. (Analyse the result)

****Good Luck****