ClassObjIntro

Monday, April 27, 2020 12:41 PM

Java Class Model

The purpose of this lab is to introduce the basics of writing object-oriented code in Java. The lab may appear simplistic in nature, but it is important that the *concepts* of object-oriented coding are grasped correctly, as they will provide a foundation for the rest of the course.

Exercise 1 – Creating a Class

In this exercise, write the `Person` class with a class attribute of type `Address` along with class attributes that store relevant information about the person. It is important that you note that this class follows the concept of *encapsulation*, which is a fundamental part of object-oriented design.

Skeleton classes named `Person` and `Address` have been provided. The `Person` class will model and store basic characteristics of a person. Address data will be handled by a class named `Address`.

Follow the accessibility rules-of-thumb (attributes are private and methods are public). Details include:
A class called `Address`:

- `Address` has 3 attributes: street, city and zip, all of type `String`
- `Address` has a 3-argument constructor to initialize the attributes
- There are 3 `get` methods to retrieve the attributes, e.g., `getStreet()`

The class `Person`:

- `Person` has 3 attributes: name of type `String`, age of type `int` and address of type `Address`
- A 3-argument constructor, to initialize the name, age and address attributes
- A `setName(String name)` and a `getName()` method to set and get the name attribute
- A `setAge(int age)` and a `getAge()` method to set and get the age attribute
- A `toString()` method that returns a textual representation of the class. The `toString()` method is inherited from the `Object` class. We will override it here so that a person is appropriately presented as a string. Note that this method is invoked automatically when the class is used in a string context, e.g., concatenation or as an argument to `System.out.println()`.

Exercise 2 – Test the Person Class

All programs need a starting point. For standalone applications, this is the `main` method. Each class can have a `main` method that can be used as a unit tester, even if it is not the starting point for the entire application.

You could also write a class that functions as a test driver. The purpose of this exercise is to write a driver that starts up the program and tests the `Person` class created in the previous exercise.

1. Write a `main` method for the `Person` class that creates an instance of `Person` and an `Address` by invoking the non-default constructors.

2. Test the class by causing the `toString()` method to be invoked.
   Hint: pass the instance as an argument to `System.out.println()`.

Exercise 3 - Extend the Person Class

All fully-functional object-oriented languages support inheritance. In this exercise, inherit from the `Person` class to create two more classes that are slightly more unique and have different functionality from one another.

1. You are provided two starting files with which to make two subclasses of `Person`: `Employee` and `Contractor`.

2. `Employee` class details:

   a) A `salary` attribute of type `double`.

   b) Write a constructor that takes all the arguments required to construct the `Person` class as well as one to set `salary`.

   c) Add `setSalary(double salary)` and `getSalary()` methods.

3. `Contractor` class details:

   a) Two attributes, `permanent` of type boolean that stores if the contractor is a permanent contractor or not and an `hourlyRate` attribute of type `double`.

   b) Write a constructor that takes all the arguments required to construct the `Person` class, as well as the two for this class.

   c) Add `setHourlyRate(double salary)` and `getHourlyRate()` methods.

4. Override the `toString()` method of the `Person` class in both of the sub-classes to print out the extra details created in each sub-class.

   Hint: Reuse existing functionality; do not duplicate code. Use the `super` keyword to call the `superclass` constructor as well as the superclass `toString()` method.