**LAB #2 – Java Methods**

**Due Date: Week 4 (demo of your code in class) – 5 February, 2017**

**Value: 5%**

**Maximum points: 15 points**

**Purpose:** The purpose of this Lab assignment is to:

* Practice the use of instance methods in Java classes
* Practice the use of static methods in Java classes
* Practice with arrays, switches, loops, etc…

**References:** Read Chapter 4, 5, 6, 7.

This material provides the necessary information you need to complete the exercises.

Be sure to read the following general instructions carefully:

- This lab should be completed individually by all the students.

- You will have to demonstrate your solution in a scheduled lab session and submitting the code **through dropbox link on eCentennial**.

You must name your Eclipse project according to the following rule:

**YourFullName\_COMP228\_sectionNumber\_LabNumber**

Example: **JohSmith\_COMP228\_006\_Lab2**

Each exercise should be placed in a separate package named *exercise1*, *exercise2*, etc.

Submit your assignment in a **zip file** that is named according to the following rule:

**YourLastName\_COMP228\_sectionNumber\_LabNumber.zip**

Example: **JohSmith\_COMP228\_006\_Lab2.zip**

Apply the naming conventions for variables, methods, classes, and packages:

- *variable names* start with a *lowercase* character

- *classes* start with an *uppercase* character

- **packages** use only *lowercase* characters

- *methods* start with a *lowercase* character

**Exercise 1: (7 points)**

Working with ArrayList, composition, loops.

Write a Java application using some code from previous lab 1.

Copy your Patient class.

Create Address class. See UML below.

Create Clinic class. Clinic class has Address object and ArrayList of Patients. A default constructor should instantiate the ArrayList of Patients. See UML below.

Create Driver class:

* **createPatient**() method should receive all necessary variables to create a new patient. Then it should create a new patient and return it back. Use this method to create 5 new patients and store them in the ArrayList in the Clinic object.
* **printPatients()** method should receive ArrayList of Patients and print the all patients in console view (system.out…). You will have to use some kind of loop structure to iterate though all patients in the list.

**Exercise 2: (5 points)**

Working with switch, SecureRandom, static variables, inner class. Make sure you do not instantiate an object of a class when you call static members of the class.

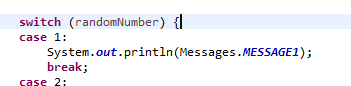
Create Driver class. See UML below.

Create MagicBall class. See UML below. Create a **final inner class** called **Messages** in the MagicBall. Messages class should contain final static messages to display to the user. Messages are below.

In the driver class create object of the magic ball. Show description of the ball to the user by calling **toString**() method. Then call method **pickUpTheBall**(). This method should display greeting message. Then call **shakeTheBall**() method. This method should create a random number and based on that random number it should display a message to the user. You should have a **switch** statement in this method. Switch example is below.

Messages:

1. Ask again later
2. It is certain
3. Yes, definitely
4. Cannot predict now
5. Don't count on it
6. My reply is no



**Exercise 3: (3 points)**

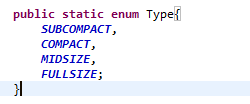
Working with enum, method overloading.

Create Car class. Create enum type. See extract of code for the enum below. Create three instance variables: Type type (hint: the enum you created), String model, BigDecimal price. Provide all getters and setters. Provide **toString**() method that should return all the fields in a string. Provide three methods **setPrice**(). Using overloading technique set different price for the car:

1. If Driver class provides only price then set the price.
2. If Driver class sends two arguments price and discount then subtract discount amount from the price.
3. If Driver class sends three arguments price, discount and coupon then the method should subtract discount and coupon amounts from the price.

Create a Driver class. Create three methods in the driver class **createCar**(). The methods should be different in terms of what arguments are passed to them to call the three different setPrice() methods on the car object. Pass required arguments to the method to create a car. The return type of this method should be Car.

Write unit tests to test **setPrice**() methods. There should be three tests to test all three methods.



**Evaluation:**

|  |  |
| --- | --- |
| **Functionality** |  |
| Correct implementation of classes (instance variable declarations, constructors, getter and setter methods, etc.) | 40% |
| Correct implementation of driver classes (declaring and creating objects, calling their methods, interacting with user, displaying results) | 40% |
| Comments, correct naming of variables, methods, classes, etc. | 5% |
| **Friendly input/output** | 15% |
| **Total** | 100% |