ENSF 593/594 – Principles of Software Development

Fall 2019

Lab Assignment #2: Introduction to Classes and Objects

Due Dates
Submit electronically on D2L **before 11:59 PM on Monday September 30th**

The objectives of this lab are to:

1. practice with classes and objects
2. Learn about working with multiple objects
3. Practice with arrays of objects
4. Have a deeper understanding of how java collections like ArrayLists work

<u>The following rules apply to this lab and all other lab assignments in future:</u>

1. Before submitting your lab reports, take a moment to make sure that you are handing in all the material that is required. If you forget to hand something in, that is your fault; you can't use `I forgot' as an excuse to hand in parts of the assignment late.
2. **<u>20% marks</u>** will be deducted from the assignments handed in up to **<u>24 hours</u>** after each due date. It means if your mark is X out of Y, you will only gain 0.8 times X. There will be no credit for assignments turned in later than 24 hours after the due dates; they will be returned unmarked.

Total lab marks: 60

Exercise - 1: Flight Simulator (20 Marks)

Read this First: Why OOP?

While the real-world systems are a collection of objects working together, software systems and in particular engineering applications will be easier designed and developed, if they resemble a real-world system. For example, a 'flight simulator application' will be easier designed and developed, if its components resemble an actual cockpit equipped with standard flight instruments such as altimeter, attitude indicator, airspeed indicator, magnetic compass, heading indicator, etc.

Source:
http://en.wikipedia.org/wiki/File:Six_flight_instruments.JPG

In this example the cockpit is an object that contains many smaller objects. In flight simulator software not necessarily all the objects need to be tangible objects. As an example, when simulating a flight from departure point to a destination point, you may want to consider two objects: one represents the departure point and the other one represents the destination point. These points have some properties: longitude, latitude, that can be considered as x and y coordinates of a point on a Cartesian plane.
The concept of modularity in software engineering stems from hardware industry. This concept makes the process of building and maintenance of hardware easier. For example, if your cell phone needs more memory, you can simply unplug the existing memory card and replace it with a new one. In the software industry the same concept can be employed to make the simulation of aircraft cockpits easier. Each instrument can be encapsulated as separate object. If the simulation of any of these instruments needs to be changed, you must only focus on that particular instrument (object) and the interfaces that make the communication of this object possible with the other object (i.e. its member methods). In summary, OOP is a better programming style, particularly in complex systems, because makes it easier to develop, easier to maintain, easier to debug, and easier to reuse.

Task 1.1 – Writing Class Point

You will define and use a class that represents a point on a Cartesian plane.

Write the definition of class Point with the following data members, member functions (i.e. instance methods), and constructors:

- Three private data members:
  - A double data member that represents the x coordinate of a point on a Cartesian plane.
  - A double data member that represents the y coordinate of a point on a Cartesian plane.
  - A string data member that represents the label or a name for a point on a Cartesian plane.
- Two constructors:
  - A default constructor (a constructor with no arguments). Its implementation detail is explained later.
  - A constructor that receives two arguments of type double and one argument of type string.
- Several instance methods:
  - Access methods to allow data members x and y coordinates of a point, and its label to be retrieved or modified (i.e. getters and setters)
  - Method called toString() with no arguments. This method should return the information about a point as a string.
    Hint: Lookup how toString() works in java

Task 1.2 – Writing Class Time

Class Time simply makes an abstraction of time (hours, minutes, and seconds), and provides some methods to be able to retrieve, modify, or reset the value of time. (You can reuse some/all your code from class Clock in lab 1)

Class Time must have:

- Three integer data members called: hours, minutes, seconds.
- A default constructor that initializes the above-mentioned data members to zero.
- A second constructor that receives an integer argument, called totalSeconds that represents the total number of seconds in a time period, and converts this number to seconds, minutes, and hours. For example, if the value of this argument is 7832, it should assign the values: 2 to hours, 10 to minutes, and 32 to seconds.
- And, the following instance methods:
  - Method called increment that receives an integer argument, called dt (in seconds), and adds the value of dt to the current time. For example, if time is: 14:50:40, and dt is 20 seconds, the new value of time will be: 14:51:00. Note that dt can be any number, for example 300 seconds,

- Getter and setter methods for hour, minutes and seconds data members

Note that the value of hours can be unlimited, but the values of minutes and seconds cannot be more than 59.

Task 1.3 – Object Communication

Read this First:
Applications written in object-oriented style normally use several objects of different classes, and these classes need to communicate to implement some tasks. As an example, an object can use the other objects as its parts, or an object can call the other objects member functions to use their services (We will discuss much more about the relationships between objects later in the course). In this part of the lab you are going to create an excessively simplified abstraction of an aircraft's cockpit.

Here are only a few lines of the screen output of a sample run of the program, if every function works as expected:

```
Flight number CYC 1888 is ready for departure ...
From: Calgary Airport(50, 50)
To: Toronto Airport(2000, 2000)
Please Press ENTER to Take Off...

Flight Number: CYC 1888
Current Location: On The Sky at(210.00, 210.00)
Traveled Time from Source: 00:25:00
Current Average Speed: 543.06 miles/hour
Estimated Remaining Travel Time: 4:39:41
-------------------------------------------
Flight Number: CYC 1888
Current Location: On The Sky at(381.00, 381.00)
Traveled Time from Source: 00:41:40
Current Average Speed: 674.07 miles/hour
Estimated Remaining Travel Time: 3:23:48
-------------------------------------------
Flight Number: CYC 1888
Current Location: On The Sky at(543.00, 543.00)
Traveled Time from Source: 01:06:40
Current Average Speed: 627.49 miles/hour
Estimated Remaining Travel Time: 3:17:1
-------------------------------------------
```

What to Do:

Download Cockpit.java and RunCockpit.java from D2L. Read its content. Your job is to complete the definition of Cockpit.java so that the entire program runs. You don't have to modify RunCockpit.java.

Note about sleep in main: Command sleep puts a few seconds delay into the execution of the program, allowing you to simulate the passage of time, making the appearance of the program more realistic.

What to submit:

Submit all your .java files: Point.java, Time.java, Cockpit.java, RunCockpit.java. Also, create and submit javadocs for this project.

Exercise - 2: Constructing Your Own ArrayList Class (20 Marks)

In this exercise, you will write a class that works like a simple version of ArrayList. The main difference between your class and the existing ArrayList class in Java, is that your class will only construct lists of only one type (e.g. integers), whereas the ArrayList in Java can be used to create a list of any type. We will be able to create such a class in the week to come!

Task 2.1 – IntArrayList

Create a class called IntArrayList using the following partial description:

```
public class IntArrayList {

      private int [] array;


      public IntArrayList (int size){

            if (size > 0)
                  array = new int [size];
      }
      public IntArrayList () {}


      public void addElement (int x) {

            //code missing
      }
       //more methods here
}
```

Note: you must have two constructors same as above.

The main method for this class would look like:

```
public static void main (String [] args) {

            IntArrayList ar = new IntArrayList ();

            ar.addElement (2);
            ar.addElement (4);
            ar.addElement (6);
            ar.addElement (8);

           ar.insertAt (2, 5); //inserting number 5 at index 2
           ar.removeAt (1);    //remove the element at index 1
           ar.resize (2);       //the new size of the array will be 2 elements
           ar.printElements();    //This should print all elements in the array
      }
```

Your job is to complete this class by writing the definition for the instance methods: `insertAt, removeAt, resize, and printElements`

Task 2.2 – ObjectArrayList

Create a class called ObjectArrayList.java that works just like the one you made for task 1.2. However, make the array to be of type Point. You can use the class Point that made for Exercise 1 for this question.

You will then have an arraylist of objects of type Point. You will then have to implement all methods including the constructors and the instance methods `insertAt, removeAt, resize, and printElements` for this class.

Note: in Task 2.1, when inserting for example, you would insert an integer. Here, when you insert, you must insert an object of type Point.

What to Hand in: Submit your .java files for tasks 2.1 and 2.2.

Exercise - 3: Simple Full Stack Development (20 Marks)

This exercise will be discussed and defined during the class on Thursday September 19th.

How to submit: Include all your files for ALL exercises in one folder, zip your folder and upload it in D2L before the deadline.