Computer Science COMP-2120 - Fall 2020

Assignment 1

Due: End of Sunday, October 11, 2020

Submission Instruction: (10 points)

****

- Submit your assignment on Blackboard as a **single zip file**. **(2 points)**
- The zip file should be named as **Assignment1_*StudentId*.zip**, in which replace *StudentId* with your university student number. **(2 points)**
- The zip file should include only Java files, i.e. files with the extension name ".java" **(2 points)**
- DO NOT include any compiled class file. If you intentionally or unintentionally submit compiled files, you will receive an email from myself or a TA and you have 12 hours from the time of sending the email to submit your java files. **If you do not submit the zip file including the Java source files before that 12 hours, then you will receive zero for the assignment**.
- For every problem, **follow the name of the files as instructed**. **(2 points)**
- **If you follow all the above instructions properly, then you will receive 2 bonus points.**

****

Note: For each problem, you must create the Java file(s) with the exact names that are provided at the end of each problem.

Problem 1. **(10 points)**

Goal: Practice arithmetic operations in Java.

Write a Java program, in which declare and initialize required variables with some arbitrary values, and compute the following mathematical operations:

$$s = s_0 * v_0 - \frac{1}{2} g * t^3 \qquad G = \frac{3}{4}\pi^2 \frac{a^3}{p^{\frac{1}{2}}(m_1 - m_2)}$$

Java file name: Problem1.java

Problem 2. **(8 points)**

Goal: Practice arithmetic operation and the meaning and order of the mathematical operations in Java.

Write a Java program and have the following arithmetic operations and show the results in separate lines.

```
double x = -3.5;
double y = 1.9;
int n = 23;
int m = 14;

a.  x - n / y + x + (n * y)
b.  n / m + n % m
```

```
c.  n % 2 + m % 3
d.  (m + n) / 3.0
e.  (n - m) / 3
f.  (n – x) / 3
g.  1 - (1 - (1 - n))
h.  m % 10 + (m – (n % 10))
```

Java file name: Problem2.java

Problem 3. **(12 points)**

Goal: Practice how to use Java API by creating objects from standard Java classes in our Java code.

Browse the Java API Documentation on Internet and try to find two classes, one for creating two-dimensional lines, and one for creating two-dimensional ellipses. Then, write a Java program, to do the following tasks:

1- Create a line with the following initial values: from the point (3, 12) to the point (17, 31)
2- Change the end point of the line to (19, 13).
3- Create an ellipse with the following initial values:
   Height=25, Width=60,
   The x coordinate of the upper-left corner of the framing rectangle of this ellipse = 4
   The y coordinate of the upper-left corner of the framing rectangle of this ellipse = 13

After each of the following steps, test your program by printing the actual and expected results.

Note*: In this example, we just want to create objects and not drawing them on the screen.

Java file name: Problem3.java

Problem 4. **(20 points)**

Goal: Practice how to create user-defined classes in Java.

A microwave control panel has five buttons: 1) one for increasing the time by 30 seconds, 2) one for switching between power levels, Low, Medium, and High, 3) a stop button, 4) a reset button, and 5) a start button. Implement a class that simulates the microwave, with a method for each button. The method for start button should print a message "Cooking for … seconds at level …" and changes the current status of the microwave. The method for stop button should print a message "Cooking stopped." and changes the current status of the microwave. Based on the above description, you must figure out the required properties of the microwave control panel. At the end, write a separate Java program as the tester to test your microwave class with different values.

Note*: There is no graphical task in this problem.

Java file names: Microwave.java and TestMicrowave.java

Problem 5. **(20 points)**

Goal: Practice how to create user-defined classes in Java.

Enhance the `BankAccount` class and see how abstraction and encapsulation enable evolutionary changes to software:

- Begin with a simple enhancement: charging a fee for every deposit and withdrawal. Supply a

mechanism for setting the fee and modify the `deposit` and `withdraw` methods so that the fee is levied. The initial transaction fee is 50 cents, and this can be changed later on. Test your class and check that the fee is computed correctly.

- Now make a more complex change. The bank will allow a fixed number of free transactions (deposits and withdrawals) every month, and charge for transactions exceeding the free allotment. The charge is not levied immediately but at the end of the month. The initial value for the number of free transactions is 10.
  Supply a new method `deductMonthlyCharge` to the `BankAccount` class that deducts the monthly charge and resets the transaction count.
  (Hint: Use `Math.max(actual transaction count , free transaction count)` in your computation.)
- Write a tester Java program that verifies that the fees are calculated correctly over several months.

Java file names: BankAccount.java and TestBankAccount.java

Problem 6. **(20 points)**
Goal: Practice how to create user-defined classes in Java.
Design a Java class for **Person** with the following behaviours:

- ***talk***, in which a person can talk a sentence, if s/he is at least two-year old.
- ***eat***, in which a person can eat something if s/he is hungry. If the person eats something, then s/he becomes full (not hungry).
- ***needFood***, in which the person's status will become hungry if s/he is currently full.
- ***walk***, in which a person will walk a specific distance. If a person walks more than four kilometers, then s/he becomes tired and is not able to walk anymore.
- ***sleep***, in which a person will sleep if s/he is awake. If a person sleeps, then s/he is no tired anymore, her/him walking distance will reset, and s/he can walk again. Note that if a person is sleeping, s/he can't walk.
- ***awake***, in which the person is awaken if s/he is currently sleeping.
- ***grow***, in which the person's age will increase by one year. If a person reaches to 55, then his ability to walk will decrease by half every five years.

  You need to indicate all the instance variables (properties) a give person should have to be able to handle the above list of behaviors and store the current states of a given person. For instance, every person has name, age, current distance of walking, sleeping status, walking distance capability, etc.

  After designing the Person class, implement it as a Java class. Then write a tester Java program to test that class by creating some Person instance objects and do some actions for each of them. To make sure that your tester program tests all the functionalities of a given person, you have to call every method of the class at least once.

  Assume that every person can have one friend, which is another person. Therefore, two other behaviors for a person could be ***getFriend*** and ***changeFriend***. Modify the Person class that you have designed to be able to handle these two behaviours as well. Note that you need to have another instance variable as well.

Java file names: Person.java and TestPerson.java