Western University
Department of Computer Science
CS 1027 (Computer Science Fundamentals II)
Assignment 1

Professor: Ali Bou Nassif, Ph.D., P.Eng.

Instructions: Read carefully before you start.

This assignment should be done individually. The deadline is Thursday July 2, 2015 at 11:54 pm. Please refer to the course outline (**Assignments Submissions and Late Assignments Policy Section)** for the late penalty. Each student must submit a zip file (zip and NOT rar) that compresses the main folder of his/her assignment (check the last paragraph of the assignment for more details). All files should work under Eclipse. It is very important that you comment your code thoroughly. Marks will be deducted if the code is not commented properly. The overall mark is 120 (20 bonus marks)

Question 1: 60 points

The goal of this question is to demonstrate UML diagrams, Classes, Objects, Inheritance Polymorphism.

Consider the following requirements:

a. Interface ***People***

- Methods
    - **convertGrade()**

b. Class ***Student*** implements ***People***
The class ***Student*** is not abstract (this means you have to override the method **convertGrade**).
The **convertGrade()** method is implemented as the following:
If grade is >=90, output is A+
If grade is >=80, output is A
If grade is >=70, output is B
If grade is >=60, output is C
If grade <60, output is F

- Attributes
    - **ID** of type integer (int)
    - **name** of type String
    - **grade** of type integer
- Methods
    - Constructor(int, String, int) of 3 parameters to initialize the instance variables
    - **toString()** that returns the name and ID of person objects:
      Example: name is Joe ID is 1000

c. Class ***Undergrad*** inherits from class Student

- Attributes
  - **gpa** of type float
- Methods
  - ***Constructor (float, int, String, int).*** The constructor will initialize the gpa and call the Student's constructor to initialize the ID, name and grade
  - **print()** that calls **toString()** method to print the **ID**, **name** of an **Undergrad** object followed by printing the **gpa**
    Example: name is Laila ID is 5000 gpa is 3.7

d. Class ***Grad*** inherits from class **Student**

- Variables
  - ***supervisor*** of type String
- Methods
  - **Constructor (String, int, String, int)** that has a similar function to Undergrad's constructor
  - **convertGrade()** that overrides the **convertGrade()** method based on the following rules:

If grade is >=80, output is A
If grade is >=70, output is B
If grade <70, output is F

Note:

a. The visibility of all attributes in the super class is protected
b. The visibility of all attributes in the subclasses is private
c. The visibility of all methods in all classes is public
d. Each private variable must have two public methods; getter and setter

1- Construct the UML class diagram based on the above requirements. If you do not know any tool to construct the class diagram, you can download NClass from http://nclass.sourceforge.net/downloads.html. NClass is free and very easy to use. Submit the actual class diagram (image) and not the source code of the diagram. Check the paragraph called "How to use NClass" below for more details. Use NClass to generate java code for you (you will see 4 files, one for the interface and 3 for the classes)

2- Write a program in Java to implement the above requirements.
   a. Complete the implementation of all methods in all files generated from NClass
   b. Write a new class (new file) called StudentDemo.java that contains the main method. In this class, you need to:
      i. Create an **Undergrad** object called **u** of **gpa** 3.8, **ID** 100, **name** "ali" and **grade** 95
      ii. Print the details of the object **u** (call the **print** method)
      iii. Change the **ID** of **u** to 500
      iv. Print again the details of the object **u**
      v. Call the method (and print the results) **convertGrade()** of **u**
      vi. Crate a **Grad** object called **g** of supervisor "mike" **ID** 200, **name** "sami" and **grade** 95
      vii. Change the **supervisor** to "tim"

viii. Print the string representation of the object **g**
ix. Call the method (and print the results) **convertGrade()** of **g**
x. Create a variable **s** of Student type and refers it to Undergrad object **u** (demonstrating polymorphism)
xi. Call and print the method **convertGrade()** of **s**
xii. Have **s** refer it to **Grad** object **g** (demonstrating polymorphism)
xiii. Call and print the method **convertGrade()** of **s**

How to use NClass:

a. Download and install NClass from http://nclass.sourceforge.net/downloads.html
b. Run NClass
c. File -> New -> Project
d. File -> New -> Java Diagram
e. Diagram -> New -> (choose your diagram). You can also choose the shortcuts in the menu bar
f. Make sure you use "Generalization" to represent inheritance and "Realization" to represent implementation of an interface
g. When you finish the diagram, go to Diagram -> Generate Code -> Choose Java as a language, delete the import list (java.io.* and java.util.*). Also, uncheck the box "Fill methods with 'Not Implemented' exceptions
h. Press on the Generate button and choose your destination.
i. You will notice that 4 files are created.

Question 2 (60 points)

The goal of this question is to demonstrate infix to postfix conversion using stack. In class, we showed how to evaluate a postfix notation. Please refer to the lectures to learn more about infix and postfix notations.

The algorithm for converting infix to postfix is as follows:

1. Scan the Infix string from left to right.
2. Initialize an empty stack.
3. If the scanned character is an operand, add it to the Postfix string. If the scanned character is an operator and if the stack is empty, Push the character to the stack.
4. If the scanned character is an operator and the stack is not empty, compare the precedence of the character with the element on top of the stack(topStack). If topStack has higher precedence over the scanned character, Pop the stack, else Push the scanned character to stack. Repeat this step as long as stack is not empty and topStack has precedence over the character.
5. If the scanned character is a left brace '(', Push the character to the stack
6. If the scanned character is a right brace
    a. While the topStack is not left brace AND stack is not empty
        i. Add the topStack to the Postfix string
        ii. Pop the stack
    b. Pop the stack // when you exit the while loop

7. Repeat this step until all the characters are scanned.
8. After all characters are scanned, you have to add any character that the stack may have to the Postfix string. If the stack is not empty, add topStack to Postfix string and Pop the stack. Repeat this step as long as stack is not empty.
9. Return the Postfix string.

Write a file called InfixtoPostix.java to implement the above algorithm.

Example:

Let us see how the above algorithm will be implemented using an example.

Infix String: a+b*c-d

Initially the Stack is empty and our Postfix string has no characters. Now, the first character scanned is 'a'. 'a' is added to the Postfix string. The next character scanned is '+'. It being an operator, it is pushed to the stack.

Stack: +
Postfix String: a

Next character scanned is 'b' which will be placed in the Postfix string. Next character is '*' which is an operator. Now, the top element of the stack is '+' which has lower precedence than '*', so '*' will be pushed to the stack.

Stack: +*
Postfix String: ab

The next character is 'c' which is placed in the Postfix string. Next character scanned is '-'. The topmost character in the stack is '*' which has a higher precedence than '-'. Thus '*' will be popped out from the stack and added to the Postfix string. Even now the stack is not empty. Now the topmost element of the stack is '+' which has equal priority to '-'. So pop the '+' from the stack and add it to the Postfix string. The '-' will be pushed to the stack.

Stack: -
Postfix String: abc*+

Next character is 'd' which is added to Postfix string. Now all characters have been scanned so we must pop the remaining elements from the stack and add it to the Postfix string. At this stage we have only a '-' in the stack. It is popped out and added to the Postfix string. So, after all characters are scanned, this is how the stack and Postfix string will be:

Stack
Postfix String: abc*+d-

End result:
Infix String : a+b*c-d
Postfix String : abc*+d-

Deliverables:

1. A zip file of name ***your uwo user name***. This file should contain two folders named question1 and question2 (all lower case). The question1 folder should contain 5 files (People.java, Student.java, Undergrad.java, Grad.java and StudentDemo.java). The question2 folder should contain the file InfixtoPostfix.java
2. ReadMe.pdf file. The submission of this file is optional. However, if you feel that you would like to tell the TAs any useful information about your assignment, feel free to write it in this file. This file can be added to the main folder
3. Submit the zip file to OWL before the deadline