COSC 1047 – Winter 2017 – Assignment 2
Due: (see D2L)

This assignment is designed to give you some experience writing Java object classes using Aggregation and String processing. This assignment is done using Eclipse. Import the .zip file that is provided on D2L and when it is complete, export it again as a .zip. All submissions should be done through D2L. You can submit and resubmit as many times as you want up to the due date and time. If you have any problems submitting, email me – alangille@cs.laurentian.ca - in a timely manner. **Make sure your name, student number and a brief description of your program appear at the top of each file in comments. Also make sure that you use comments to clarify your code for marking.** ***Failure to do either of these tasks may result in a deduction of marks.***

Submit solutions to questions 1 and 2 only. But DO ALL questions to be prepared for the tests and exam. Solutions will be provided (via D2L) only for required questions.

If you work in partners remember to include both names in the source code. The person who is left out will receive zero.

Note: It is imperative that you import the a2-w17.zip file from D2L properly or you won't be able to work in the proper project environment. Although the screen shots are a bit outdate these instructions are correct:

http://agile.csc.ncsu.edu/SEMaterials/tutorials/import_export/

Grading:

For the submitted questions the grading is as follows:

Proper documentation, formatting, comments and Java Docs where requested:	5 marks
Program compiles properly and runs:	5 marks
Program runs properly for most of the examples and requirements of the question:	5 marks
Program runs properly for all of the examples and requirements of the question:	5 marks
Extra work where applicable:	5 marks

Small non-critical errors and suggestions for improving code will be given as comments and should be addressed in future assignments. Major errors and failure to follow the question guidelines will result in deductions per the scheme listed above.

Question 1: (Unicycle.java, UnicycleTester.java) (20 marks) In the project (question1 package) you are given object classes for Frames and Wheels. Your job is to create a new Object class Unicycle that has as data field a Frame object and a Wheel object. It also has a String data field which represents the customer for whom the Unicycle is being built. I suggest taking a few minutes to look over Frame and Wheel to see what they do. Your Unicycle class should have the following:

- A default, no argument constructor that creates a unicycle with your name as the customer and a frame/wheel combination of your choice.
- A fully-argumented constructor that takes in a wheel, frame and customer name.
- A copy constructor that creates a new Unicycle that has the same properties as the incoming argument Unicycle.
- Get and set methods for all of the data fields.
- An appropriate toString method.

The idea here is to practice deep-copy for your constructors, set methods and return methods. Wherever appropriate you must avoid using shallow copy. Write a hard-coded tester to make sure that your objects are working properly and that there are no "data leaks" (this is what happens when one object changes and those changes are reflected in another).

Bonus: Create an interactive tester class that collects from the user all of the information required to create their own custom Unicycle and allows them to modify if they choose to.

My hard-coded tester output:

```
Creating Unicycle using default constructor:
Unicycle[Aaron Langille, Frame[16, blue], Wheel[12, learner]]

Creating new frame and wheel for another unicycle:
Frame[18, green]
Wheel[16, mountain]

Creating new custom unicycle from the parts for Peter Parker (aka Spiderman):
Unicycle[Peter Parker, Frame[18, green], Wheel[16, mountain]]

Pete changed his mind and wants to alter the unicycle to be a red and blue learner:
Unicycle[Peter Parker, Frame[18, red and blue], Wheel[16, learner]]
```

2. (StringTools.java, StringToolsTester.java) Write a class of static methods that accomplish the following tasks. In all cases do not look for built-in methods that accomplish the task, use loops as well as String and Character methods to get the tasks done.

reverse(String s) : String
-take a String as an argument and return the String reversed. Eg. "Hello" would be returned as "olleH".

binaryToDecimal(String s) : int
-take a String as an argument and return the decimal equivalent as an integer. If any of the characters are neither zero nor one return null. Eg. "1000" would return 8. "10001" would return 17. "111111" would return 63. "111112" would return -1. "100001.1" would return -1.

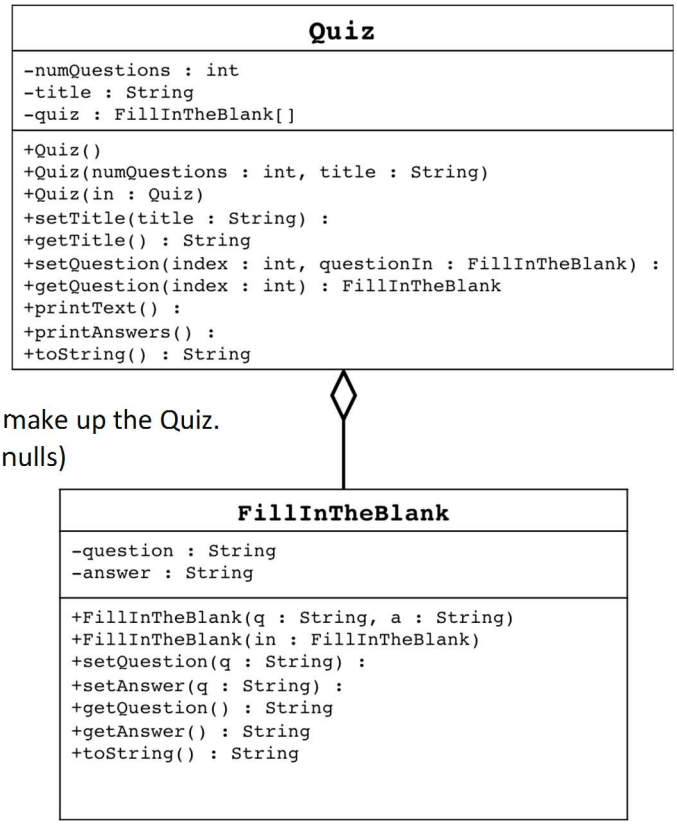
initials(String s) : String
-take a String as an argument and return a properly "initialized" and capitalized name. If the String does not contain exactly three names return null. Eg. "James tiBeriUs kiRK" would return "J. T. Kirk". "jean luc picard" would return "J. L. Picard". "AaroN LANGille" would return null.

mostFrequent(String s) : char
-take a String and return the Character that occurs most frequently. In the event of a tie, return the last character that was checked in the tie.

replaceSubString(String s1, String s2, String s3) : String
-take three Strings as arguments. The first String is the user's "text". The second String is a pattern to look for. Make all Strings lowercase in the method to facilitate search and replace. The third String is a pattern to replace the second String with. Eg: "the dog jumped over the fence" as the first String, "the" as the second and "that" as the third. Returned String is "that dog jumped over that fence".

Write a tester class that shows off all of your methods. Be sure that errors (input that results in null) are handled properly.

Question 3: (FillInTheBlank.java, Quiz.java, QuizTester.java) Consider the following aggregation relationship:



You are given the FillInTheBlank class which defines a fill-in-the-blank-type quiz question. Spend a few moments going over this class to understand how it works. A short tester class has also been provided to show you how it works. Add a copy constructor to this class that makes it easy to duplicate an existing question. Write JavaDocs for the FillInTheBlank class.

Next, write a Quiz class that is made up of FillInTheBlank questions. The UML of this class is given above.

A few hints:

- numQuestions simply keeps track of how many total questions make up the Quiz.
- The default constructor creates a quiz with 5 empty questions (nulls) and a title of your choosing.
- printTest() should display the title of the test and then all the test questions (with question numbers) and printAnswers() should display only the answers (with question numbers). IF one of the questions has not yet been answered (i.e., it's still null in the array) do not print anything. You can use an if statement to test for nulls. Try it out.
- setQuestion() and getQuestion() should make sure that the index numbers are appropriate and if not should do nothing.
- toString() simply prints the quiz title and the number of questions the quiz can hold.
- Be sure to use DEEP COPY and DEEP RETURN where appropriate.
- If anything else is vague or uncertain use the discussion forum! My Quiz class is 75 lines without comments and with liberal spacing and curly braces.
- Scanners should only appear in your tester class.
- Write JavaDocs for the Quiz class.

Finally write a quiz tester class that creates a simple 5 question quiz. Your tester should allow the user to create the questions. When the quiz has been created use the printTest() and printAnswers() methods to display the results.

No output is shown here. Be creative, be polite (to your users) and if you want to compare and contrast program output use the discussion forum.

Bonus: After you've done the work above in the tester, create a second quiz using your quiz copy constructor. For your second (copied) test, swap the first and last questions and for the middle question change the question AND the answer (do this by access the question's set methods, not by replacing the question). Finally, print the two tests and two answer sets to show that they are distinct and that the changes to one do not affect the other (i.e., your deep copy is working).