COSC 1047 – Winter 2017 – Assignment 1
Due: (see D2L)

This assignment is designed to give you some experience writing Java object classes and inheritance. 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 a1-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. Write a class that has the following methods. Use nested for loops for all of these array operations:

`public static void initialize(int[][] a, int min, int max)` : initialize the array by filling it with values between min and max.

`public static int[][] sum(int[][] a, int[][] b)` : compute the sum of two arrays and return a new array with those values. Remember, each value in array `a` gets added to the corresponding value in array `b`. Assume that the arrays are the same size.

`public static void reverse(int[][] a)` : reverse all the values in the array

`public static int[][] copy(int[][] a)` : create a new array that is a value-by-value copy of the argument array.

`public static void printArray(int[][] a)` : print the values stored in the array

In your `main()` method prompt the user to enter two integers that will be the rows and columns for two arrays. Use your methods to initialize both arrays, to compute the sum and store it in a new array, to create a copy of one of your arrays and to reverse one of them. Print out your results as you go along.

Sample output should appear on D2L before too long but get started!

Question 2. (**Polynomial.java and PolynomialTester.java**) Not all object classes represent "tangible" items. Write a class called Polynomial that represents and evaluates a polynomial expression. Recall that a polynomial function of x takes the form : $P(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1} + a_nx^n$ where coefficients a_i are floating-point numbers, the exponents of *x* are integers, and the largest exponent *n*—called the degree of the polynomial—is greater than or equal to zero. It sounds complicated but it might not be as bad as you think. Your class is defined by two data fields, one that represents the degree of the polynomial (the value of largest exponent *n*) and another that is a 1D array of coefficients *ai*. Thus if the value of n is 4 and the coefficients in the array are (in order) 7, 2, 5, 0, 1 the corresponding polynomial is $7 + 2x + 5x^2 + x^4$. Coefficients can be integers or doubles. Below is the UML diagram for this class:

Polynomial
-degree : int -coefficients : double[ ]
+Polynomial(deg : int) +Polynomial(p : Polynomial) +setCoefficient(index : int, value : double) : +getCoefficient(index : int) : double +evaluate(x : double) : double +toString() : String +equals(obj2 : Object) : boolean

Here are some extra hints to help you along:

- Polynomial(deg : int) : this is an argumented constructor that creates a new polynomial object with degree deg. The coefficients will all be 0 after the object is created.
- Polynomial(p: Polynomial) : this is a copy constructor. It should take an existing polynomial object as an argument and should create a new one with the same degree and coefficients.
- setCoefficient, getCoefficient should be straight forward (I think?!). Note: If the user enters invalid indices, do nothing (change no coefficient value, get no coefficient value).
- evaluate(x : double) : this method returns the computed value of the polynomial given the value of argument x. If we use our previous example polynomial $7 + 2x + 5x^2 + x^4$, the value of this polynomial for a value of x = 3 would be: $7 + 2(3) + 5(3)^2 + (3)^4 = 139$. You'll need a loop to access all of the coefficients.
- equals(obj2): compares to Polynomials (or a polynomial and another object) for equality. Two polynomials are considered equal IFF they have the same degree and all the same coefficients (in the same order).
- toString(): displays a polynomial to the console (see sample output on next page). Use a loop to constructor your
- Write JavaDocs for the Polynomial class (not the tester).

REMEMBER – your tester class will do the input and output work; it will make use of your object. There should be NO Scanners in the Polynomial class. Keep your methods abstract and generalized – data comes in through arguments and out through return statements.

See next page for tester class.

Write an application class called PolynomialTester that accomplishes the following:

- Prompt the user to enter the degree of the desired polynomial. Call your constructor and create the basic polynomial. At least make sure that the degree is positive and greater than zero (i.e., input validation).
- Prompt the user to enter all the necessary coefficients. Use a loop and your setCoefficient() method to set the coefficients for your polynomial.
- Display the "completed" polynomial.
- Prompt the user to enter in a value for x (double or int are both fine). Evaluate the polynomial for that value of x and display the result.
- Use a sentinel loop to re-prompt the user to enter other values of x and to keep computing the evaluated polynomial until they enter the string "quit".
- Before quitting, create a copy of the polynomial using your copy constructor. Show that they are equal using your equals() method and by printing them both out to the screen. Then allow the user to change one coefficient of their choice and show that they are different (use equals() and display to the screen). **This is not shown in the sample output below**.
- Write proper Java docs for your Polynomial.java source file and write normal comments for your tester.

```
Enter polynomial degree: 4
Coeff for degree 0: 7
Coeff for degree 1: 2
Coeff for degree 2: 5
Coeff for degree 3: 0
Coeff for degree 4: 1
Polynomial: 7.0 + (2.0)x^1 + (5.0)x^2 + (1.0)x^4
Enter a value of x for which to evaluate the polynomial: 3
For x = 3, polynomial = 139.0
Enter a value of x for which to evaluate the polynomial: -3.1
For x = -3.1, polynomial = 141.2021
Enter a value of x for which to evaluate the polynomial: 9.26
For x = 9.26, polynomial = 7806.9089057599995
Enter a value of x for which to evaluate the polynomial: quit
```

Question 3: Write a hero class (`Hero.java`) that could be used in a role playing game. A hero has a name, an amount of health, an amount of mana (magic power) and can carry a limited number of items to help him or her on their quest. Your class should have data fields as follows:

name – self-explanatory?
health – an integer number between 100 (full health) and 0 (expired/deceased)
max_items – the maximum number of items that the hero can carry at one time
items – a list (ArrayList of Strings?) of items currently in the hero's possession. Think of this as the hero's backpack of holding. Once this value is set, it cannot be changed.

The class should define the following methods:

-Constructor (argumented): This constructor should allow for two arguments – a name and a number of items. The initial health value for a hero should be 100. A new hero should not be carrying any items but should have an empty list ready to accept items later on. If the maximum items is greater than 10 or less than 0 throw an IllegalArgumentException (look back in the book if you need a reminder).

-Constructor (default): This constructor should allow for zero arguments. The initial health value 100. The hero's name should be "Anonymous" and they should have a limit of 2 items in their backpack.

-inventory() : This method takes no arguments and returns (not simply print) a String representation of the items that the hero is currently carrying. If the hero is carrying nothing then a message should be displayed.

-take() : This method should take one argument - the name of an item that the hero is adding to their inventory. The hero can only "take" an item if they are carrying less than their maximum number of items. If the hero is at their maximum carrying capacity, print an appropriate message.

-drop() : This method should take a single argument – the name of an object the hero would like to remove from their inventory. If the user is not currently carrying the requested item then print an appropriate message, otherwise remove the item from inventory.

-takeDamage() : This method takes a single integer argument and subtracts that amount from the hero's health. It the resulting hero health would be less than zero, set the health to zero.

-heal() : This method takes a single integer argument and adds that amount from the hero's health. It the resulting hero health would be greater than 100, set the health to 100.

-toString() : write a proper toString method that will display the current state of a hero.

Write a tester class (`HeroTester.java`) that tests your hero class. Create a hero object using the argumented constructor and create another one the default constructor and print their initial state. Test your take(), drop(), inventory(), takeDamage() and heal() methods on one of your heroes to show that all of the methods work as advertised above. Hard-code names and objects from your own imagination (be creative but not insulting or derogatory). Show your heroes' final state before exiting the program. Use my output as a rough guide.

There are a couple of things to note about this output. 1) There are no loops in my runner script and all of the arguments were **hard-coded** so there is **no** user input. 2) All of the bold text is generated by the hero class, **not** by the tester meaning all non-bold text is generated by the tester (simply System.out statements), not by the hero class.

```
-------Creating heroes--------
Creating a hero with no name.
Displaying hero: Hero: Anonymous, health: 100, can carry: 2 items, current inventory: Your hero is unburdened by
worldly possessions.]
Creating a hero with a name.
Displaying hero: Hero: Brave Sir Robin, health: 100, can carry: 3 items, current inventory: Your hero is
unburdened by worldly possessions.]

-------Testing methods--------
Handing Sir Robin a sword.
Displaying inventory: [sword, ]
Handing Sir Robin a spoon.
Displaying inventory: [sword, spoon, ]
Handing Sir Robin a cape of good fortune.
Displaying inventory: [sword, spoon, cape of good fortune, ]
Handing Sir Robin a tomato:
Your hero is overburdened and cannot carry any more...
Displaying inventory: [sword, spoon, cape of good fortune, ]
Brave Sir Robin drops his spoon:
Displaying inventory: [sword, cape of good fortune, ]
Brave Sir Robin drops his watch:
Your hero is not carrying that item - watch
Displaying inventory: [sword, cape of good fortune, ]
Brave Sir Robin drops his sword:
Displaying inventory: [cape of good fortune, ]
Brave Sir Robin drops his cape of good fortune:
Displaying inventory: Your hero is unburdened by worldly possessions.]
Brave Sir Robin drops his bravery.
Your hero is not carrying that item - bravery
Displaying inventory: Your hero is unburdened by worldly possessions.]
Handing Sir Robin a pointy hat of COSC success:
Displaying inventory: [pointy hat of COSC success, ]
Brave Sir Robin is so busy worrying about his inventory he doesn't notice the danger and takes 50 damage from a
shrubery!
Displaying hero: Hero: Brave Sir Robin, health: 50, can carry: 3 items, current inventory: [pointy hat of COSC
success, ]
Brave Sir Robin takes a deap breath and recovers 5 health.
Displaying hero: Hero: Brave Sir Robin, health: 55, can carry: 3 items, current inventory: [pointy hat of COSC
success, ]
The shrubery regroups and deals 119 damage to poor Brave Sir Robin.
Displaying hero: Hero: Brave Sir Robin, health: 0, can carry: 3 items, current inventory: [pointy hat of COSC
success, ]
Brave Sir Robinn calls on the power of all the past Avatars, enters the Avatar state and heals for 128 points.
Displaying hero: Hero: Brave Sir Robin, health: 100, can carry: 3 items, current inventory: [pointy hat of COSC
success, ]
Remember, a hero can't have -ve health nor health > 100.

-------Final State--------
Displaying hero: Hero: Anonymous, health: 100, can carry: 2 items, current inventory: Your hero is unburdened by
worldly possessions.]
Displaying hero: Hero: Brave Sir Robin, health: 100, can carry: 3 items, current inventory: [pointy hat of COSC
success, ]
```

This would be so much better if a user could interactively create and maintain a character of their own. Create a tester program that allows a user to create a new hero object. Create a menu that allows them to manipulate the character as needed. The menu options should correspond to the methods that can be called. Be sure to display your hero after each change that is made. Loop the menu until the user decides they are done.

P.S. It should be pretty easy to load from and save to file instead of having to re-create the character each time. You don't have to do this but for those of you that are interested, give it a try.