ASSIGNMENT 4

COMP-202, Winter 2017, All Sections

Due: Wednesday, March 22^{nd}, 11:59pm

Please read the entire PDF before starting. You must do this assignment individually.

Question 1:	15 points
Question 2:	45 points
Question 2:	40 points
	100 points total

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment, in some cases through automated tests. While these tests will never be used to determine your entire grade, they speed up the process significantly, which allows the TAs to provide better feedback and not waste time on administrative details. Plus, if the TA is in a good mood while he or she is grading, then that increases the chance of them giving out partial marks. :)

Up to 30% can be removed for bad indentation of your code as well as omitting comments, coding structure, or missing files. Marks will be removed as well if the class and method names are not respected.

To get full marks, you must:

- Follow all directions below
- Make sure that your code compiles
  - Non-compiling code will receive a very low mark
- Write your name and student name is written as a comment in all .java files you hand in
- Indent your code properly
- Name your variables appropriately
  - The purpose of each variable should be obvious from the name
- Comment your work
  - A comment every line is not needed, but there should be enough comments to fully understand your program

Part 1 (0 points): Warm-up

Do **NOT** *submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of Part 1, then we suggest that you consult the TAs during their office hours; they can help you and work with you through the warm-up questions. You are responsible for knowing all of the material in these questions.*

Warm-up Question 1 (0 points)

Write a method `longestSubArray` that takes as input an array of integer arrays (i.e. a multi-dimensional array) and returns the *length* of the longest sub-array. For example, if the input is: `int[][] arr= {{1,2,1},{8,6}};` then `longestSubArray` should return 3.

Warm-up Question 2 (0 points)

Write a method `subArraySame` that takes as input an array of integer arrays (i.e. a multi-dimensional array) and checks if all of the numbers in each 'sub-array' are the same. For example, if the input is: `int[][] arr= {{1,1,1},{6,6}};` then `subArraySame` should return true and if the input is: `int[][] arr= {{1,6,1},{6,6}};` then `subArraySame` should return false.

Warm-up Question 3 (0 points)

Write a method `largestAverage` that takes as input an array of double arrays (i.e. a multi-dimensional array) and returns the double array with the largest *average* value. For example, if the input is: `double[][] arr= {{1.5,2.3,5.7},{12.5,-50.25}};` then `largestAverage` should return the array $\{1.5, 2.3, 5.7\}$ (as the average value is 3.17).

Warm-up Question 4 (0 points)

Write a class describing a `Cat` object. A cat has the following `attributes`: a name (String), a breed (String), an age (int) and a mood (String). The mood of a cat can be one of the following: `sleepy, hungry, angry, happy, crazy`. The cat `constructor` takes as input a String and sets that value to be the breed. The `Cat` class also contains a method called `talk()`. This method takes no input and returns nothing. Depending on the mood of the cat, it prints something different. If the cat's mood is `sleepy`, it prints *meow*. If the mood is `hungry`, it prints *RAWR!*. If the cat is `angry`, it prints *hssssss*. If the cat is `happy` it prints *purrrr*. If the cat is `crazy`, it prints a String of 10 gibberish characters (e.g. raseagafqa).

The cat `attributes` are all **private**. Each one has a corresponding **public** method called `getAttributeName()` (ie: `getName(), getMood()`, etc.) which returns the value of the `attribute`. All but the `breed` also have a **public** method called `setAttributeName()` which takes as input a value of the type of the attribute and sets the attribute to that value. Be sure that only valid mood sets are permitted. (ie, a cat's mood can only be one of five things). There is no setBreed() method because the breed of a cat is set at birth and cannot change.

Test your class in another file which contains only a main method. Test all methods to make sure they work as expected.

Warm-up Question 5 (0 points)

Using the `Cat` type defined in the previous question, create a `Cat[]` of size 5. Create 5 `Cat` objects and put them all into the array. Then use a loop to have all the `Cat` objects `meow`.

Warm-up Question 6 (0 points)

Write a class `Vector`. A `Vector` should consist of three `private` properties of type double: x, y, and z. You should add to your class a constructor which takes as input 3 doubles. These doubles should be assigned to x, y, and z. You should then write methods `getX()`, `getY()`, `getZ()`, `setX()`, `setY()`, and `setZ()` which allow you to get and set the values of the vector. Should this method be static or non-static?

Warm-up Question 7 (0 points)

Add to your `Vector` class a method `calculateMagnitude` which returns a double representing the magnitude of the vector. Should this method be static or non-static? The magnitude can be computed

by taking:

$$magnitude = \sqrt{x^2 + y^2 + z^2}$$

Warm-up Question 8 (0 points)

Write a method `scalarMultiply` which takes as input a `double[]`, and a `double scale`, and returns `void`. The method should modify the input array by multiplying each value in the array by `scale`. Should this method be static or non-static?

Warm-up Question 9 (0 points)

Write a method `deleteElement` which takes as input an `int[]` and an `int target` and deletes all occurrences of `target` from the array. By "delete" we mean create a new array (of smaller size) which has the same values as the old array but without any occurrences of `target`. The method should return the new `int[]`. Question to consider: Why is it that we have to return an array and can't simply change the input parameter array like in the previous question? Should these methods be static or non-static?

Warm-up Question 10 (0 points)

Write the same method, except this time it should take as input a `String[]` and a `String`. What is different about this than the previous method? (Hint: Remember that `String` is a reference type.)

Part 2

The questions in this part of the assignment will be graded.

Question 1: Exam Grading (15 points)

For this question, you will use multi-dimensional arrays to write a program that grades a multiple choice exam. Your code for this question should go in a file `ExamGrading.java`

We *strongly recommend* that you complete the first three warm-up questions before starting this problem.

Write a method `gradeAllStudents` that takes as input a two dimensional `char` array containing student responses to a multiple choice exam, and a one dimensional `char` array containing the solutions to each question. Each element in the 2d array is an array representing one student's responses to the entire exam. This method returns a `double[]` representing the grades for each student as percentages. Assume that all questions are worth 1 point. You may also assume that all `char` values are upper-case letters, and valid exam responses.

If, at any point, the number of responses for a student does not match the number of questions on the exam (the number of solutions), your method should `throw` an `IllegalArgumentException` that contains a message letting the user know which student (by index) caused the error, as well as what the two conflicting lengths were.

Below is an example. Suppose we have a multiple choice exam with 6 questions and written by 4 students:

```
char[][] responses = {{'C', 'A', 'B', 'B', 'C', 'A'}, {'A', 'A', 'B', 'B', 'B', 'B'},
                      {'C', 'B', 'A', 'B', 'C', 'A'}, {'A', 'B', 'A', 'B', 'B', 'B'}};
char[] solutions = {'C', 'A', 'B', 'B', 'C', 'C'};
```

The output of `gradeAllStudents` would look like:

```
[83.33333333333334, 50.0, 50.0, 16.666666666666664]
```

Question 2: Cheating in Exams (45 points)

For this question, you will again use multi-dimensional arrays, and you will write several methods to build a program that flags students for potential cheating on multiple choice exams. Your code for this question should go in *the same file* as the previous question: `ExamGrading.java`

(a) Write a method `numWrongSimilar` that takes as input two `char` arrays representing the answers of two different students, as well as a `char` array of solutions. This method returns the number of *wrong* answers that the students had the same. This method should `throw` an `IllegalArgumentException` if the students have responses that differ in length from each-other, or from the solutions.

For example, suppose the answer arrays are:

```
{'A','B','C','D','C','A'}
{'A','B','D','B','B','A'}
```

and the solutions are:

```
{'C','B','C','D','A','A'}
```

Then the method would return 1, since the only question where they both put *the same* wrong answer is the first one.

(b) Write a method `numMatches` that takes as input a `char[][]` representing all students' responses, a `char[]` representing the solutions, an `int` representing the `index` of a particular student, and another `int` representing the *similarity threshold* (minimum number of wrong answers that have to be the same in order for it to be flagged as suspicious).

This method compares the answers of the student at `index` with all other students. We will obtain the number of students who share wrong answers by repeatedly calling the `numWrongSimilar` method. However, only those students with a number of shared wrong answers greater than or equal to the similarity threshold will be counted. The method returns the number of students whose wrong answers are too similar to those of student `index`.

You *must* use the `numWrongSimilar` method that you wrote previously in writing this method.

For example, suppose we have the following three student responses (we have named the students for clarity in the example. You are not expected to name your students):

```
{'A','B','C','D','B','A'} (Student 0: Samwise)
{'C','B','D','D','B','B'} (Student 1: Frodo)
{'C','B','D','D','C','B'} (Student 2: Gandalf)
```

and the solutions are:

```
{'C','B','C','D','A','A'}
```

If we call the method with `index` 1 and similarity threshold `2`, the method would compare the number of *incorrect* responses that Frodo shares with Samwise (one), and the number of incorrect responses that Frodo shares with Gandalf (two). Since the similarity threshold is `2`, we would return 1, since there is only one student (Gandalf) who has at least `similarity threshold` wrong answers as Frodo.

(c) Finally, write a method `findSimilarAnswers` that takes as input three things:

- An `int` representing the minimum number of answers that need to be flagged in order to count as suspiciously similar.
- A `char[][]` representing the responses of all students.
- A `char[]` representing the solutions.

This method returns an `int[][]` where each index represents a student, and each sub-array lists the indices of all students who have similar wrong answers to the student at that index. Note that this array might be jagged (not rectangular), but that it should be symmetric (if student 3 matches with student 7, then student 7 should match with student 3).

In order to do this, you will need to use the `numMatches` method to determine the length of each

sub-array. Then, you need to use the `numWrongSimilar` method to determine which student indices belong in that sub-array. Be careful not to match a student with themselves!

Hint: In order to quickly display the contents of arrays, use the `Arrays.toString` and `Arrays.deepToString` methods.

Below is an example. Suppose we have a multiple choice exam with 6 questions and written by 4 students:

```
char[][] responses = {{'C', 'A', 'B', 'B', 'C', 'A'}, {'A', 'A', 'B', 'B', 'B', 'B'},
                      {'C', 'B', 'A', 'B', 'C', 'A'}, {'A', 'B', 'A', 'B', 'B', 'B'}};
char[] solutions = {'C', 'A', 'B', 'B', 'C', 'C'};
```

If we run `findSimilarAnswers` with a similarity threshold of 3, we would see:

```
[[], [3], [], [1]]
```

Students 1 and 3 had at least half of their exams containing wrong answers that were the same (the first, last and second to last questions).
If we run `findSimilarAnswers` with a similarity threshold of 5, we would see:

```
[[], [], [], []]
```

Since no pairs of students had at least 5 of their exam questions containing the same wrong answers.
If we run `findSimilarAnswers` with a similarity threshold of 1, we would see:

```
[[2], [3], [0, 3], [1, 2]]
```

Student 0 has only one wrong answer, and it is the same as one of student 2's wrong answers. Student 1 and student 3 share three wrong answers. In addition to sharing one wrong answer with student 0, student 2 shares two wrong answers with student 3.

Question 3: Wool Farming (40 points)

For this question, you will write several classes, and then create and use instances of those classes in order to simulate a sheep farm.

(a) (7 points) Write a class `Dog`. A Dog has the following *private* attributes:

- A `String` name
- A `String` breed

The dog class also contains the following *public methods*:

- A constructor that takes as input the name and breed of the dog
- `getName` which returns the name of the dog.
- A `herd` method that returns the number of sheep the dog can herd. This will depend on the type of dog doing the herding. All dogs of the Collie breed can herd 20 sheep, while all Shepherds can herd 25 sheep. Kelpies and Teruvens can each herd up to 30 sheep. All other breeds of dogs can herd 10 sheep.

  For example, a `Dog` of breed *White Shepherd* or *belgian shepherd*, or just *shepherd* would be able to herd 25 sheep.

  Hint: Think about you can use the `toLowerCase()` and `contains()` methods to do this check in a case-insensitive manner.

(b) (8 points) Write a class `Sheep`. A Sheep has the following *private* attributes:

- A `String` name
- An `int` age
- A `boolean` hasWool

- A `static Random` numberGenerator

Initialize the `Random` numberGenerator with seed 123. The Sheep class should also have the following *public* methods:

- `getName()`: returns the name of a Sheep
- `getAge()`: returns the age of a Sheep
- A constructor that takes as input the name and age of a Sheep. It should set `hasWool` to be `true`
- `shear()`: This method returns a random double between 6 and 10 (inclusive), representing the amount of wool (in pounds) taken from the sheep. You can only shear a sheep if it presently `hasWool`. This method sets `hasWool` to false after a sheep is sheared. This method should return 0 if the sheep has no wool on it. Use the `static Random` numberGenerator and `nextDouble` in generating the random number. (Do *not* use Math.random()). Note that by using the Random class, you will get the same sequence of 'random' numbers every time you run your code.

(c) (15 points) Write a class `Farm`. This class has three *private* attributes:

- An array of `Sheep`
- A `Dog`
- A `String` name

It also has the following public methods:

- A constructor that takes as input the name of the farm, a `Dog` who will be responsible for herding the sheep, and an array of sheep. It initializes the `Sheep` array and *copies* the Sheep references from the input array to the Farm array. Throw an `IllegalArgumentException` if there are more Sheep than the Dog can herd.
- `getName` that returns the name of the farm.
- `getNumSheep` that returns the number of sheep on the farm.
- `printFarm` that prints the name of the farm, the name of the dog on the farm, as well as the names and ages of all of the sheep.
- `getWool` it returns (in pounds) the amount of wool obtained from shearing all sheep on the farm.

(d) (10 points) In the provided class `WoolFactory` write a main method.
Here, you will write a main method that creates a farm where the name of the farm, the name and breed of dog, and number of sheep are chosen by the user via a Scanner object.

In order to facilitate generating your Sheep array, we have provided methods to generate random Sheep ages, and to select Sheep names randomly from a provided array of potential Sheep names (you are welcome to use your own names or modify this code).

Then, shear all of the sheep on the farm. Display the number of sheep on the farm, and then use the printFarm() method to give them complete farm information, and then print how much money they made. Sheep's wool sells for $1.45 per pound.

Sample interaction and output is shown below:

```
> run WoolFactory
What is the name of your farm? Quinn
What is the name of your dog? Rover
What kind of dog is Rover? Collie
And how many sheep do you have? 12
The farm has 12 sheep.
Farm: Quinn Dog: Rover
Cynric 3
Aethelred 7
Baldred 6
Eardwulf 5
Baldred 6
Ine 10
Cuthred 3
Eadberht 8
Cenred 1
Edward VIII 5
Aethelheard 3
Cuthred 7
We just sheared 101.90973005990251lbs of wool for a value of $147.76910858685864
> run WoolFactory
What is the name of your farm? Quinn
What is the name of your dog? Charlie
What kind of dog is Charlie? Pug
And how many sheep do you have? 27
java.lang.IllegalArgumentException: Maximum number of sheep for this dog is 10. Received 27 sheep
      at Farm.<init>(Farm.java:11)
      at WoolFactory.main(WoolFactory.java:45)
      at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
      at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
      at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
      at java.lang.reflect.Method.invoke(Unknown Source)
      at edu.rice.cs.drjava.model.compiler.JavacCompiler.runCommand(JavacCompiler.java:272)
```

What To Submit

You have to submit one zip file that contains all your files to myCourses - Assignment 4. If you do not know how to zip files, please enquire that information from any search engine or friends. Google might be your best friend with this, and for a lot of different little problems as well.

These files should all be inside your zip. Do not submit any other files, especially .class files.

`WoolFactory.java`
`Sheep.java`
`Dog.java`
`Farm.java`
`ExamGrading.java`
`Confession.txt` (optional) In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your problem, it may lead the TA to give you more partial credit. On the other hand, it also may lead the TA to notice something that otherwise they would not.