# ASSIGNMENT 3
# 2D Arrays and Objects

COMP-202, Fall 2016

Due: Wednesday, November 16th, 2016 (23:55)

You must do this assignment individually and, unless otherwise specified, you should follow all the instructions. Graders have the discretion to deduct up to 15% of the value of this assignment for deviations from the general instructions and regulations.

| | |
|---|---|
| Part 1: | 0 points |
| Part 2, Question 1: | 60 points |
| Part 2, Question 2: | 20 points |
| Part 2, Question 3: | 20 points |
| | 100 points total |

## 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.*

**Warm-up Question 1** (0 points)

Write a method that takes as input an array of integer arrays (i.e. a multidimensional array) and checks if all of the numbers in each 'sub-array' are the same. For example, if the input is:

$$\{\{1, 1, 1\}, \{6, 6\}\}$$

then in should return true and if the input is:

$$\{\{1, 6, 1\}, \{6, 6\}\}$$

it should return false.

**Warm-up Question 2** (0 points)

Write a method that takes as input an array of double arrays (i.e. a multidimensional array) and returns the double array with the largest *average* value. For example, if the input is:

$$\{\{1.5, 2.3, 5.7\}, \{12.5, -50.25\}\}$$

then it should return the array $\{1.5, 2.3, 5.7\}$ (average value is 3.17).

**Warm-up Question 3** (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 (enum Mood). 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 4** (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 5** (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 6** (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

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

**Warm-up Question 7** (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 8** (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 this method be static or non-static?

**Warm-up Question 9** (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.*

In this assignment, a 75% non-compilation penalty will apply. This means that if your code does not compile, you will receive a maximum of 25% for the question. If you are having trouble getting your code to compile, please contact your instructor or one of the TAs or consult each other on the discussion boards.

# Birthday Paradox (60 points)

**Question 1: Birthday Paradox** (60 points)

In this question, you will write code to run an experiment testing the `Birthday Paradox`. The birthday paradox is the name of a seemingly paradoxical probabilistic fact: if more than 23 people are in the same room, we expect (mathematically) that two of them have the same birthday. See `https://en.wikipedia.org/wiki/Birthday_problem` for details.

You will run your experiment by writing several methods inside of a class called `BirthdayParadox`. The general idea is we assume all birthdays are equally likely. You will create randomly filled arrays of integers where each integer is meant to represent a specific day. (e.g. 0 could mean January 1st, 1 could mean January 2nd, etc.) You will then generate arrays of various sizes and measure the frequency of duplicate numbers.

The methods to write are as follows:

1. Write a method `generateArray` that takes as input two integer values: `size` and `range`. This method generates and returns an array of length `size` containing random integers between 0 and `range-1` (including both 0 and `range-1`). Note that each random number should be generated independently (you can use `Math.random()`). (5 points)

2. Write a method `generateAllData` that takes as input three integer values: `iterations`, `size`, and `range`. It calls the method `generateArray` a total of `iterations` times in order to generate and return a two dimensional array, in which each sub-array is a random array created by the `generateArray` method. (5 points)

3. Write a method `countElement` that takes as input a two dimensional array of ints as well as an integer `element`, and returns the number of times that `element` occurs in the `int[][]`. You may assume that all of the 'sub-arrays' have the same length. (10 points)

4. Write a method `maxDay` that takes as input a two dimensional array of integers and returns the mode, which is the element that occurs the most frequently inside of the `int[][]`. You may assume that all of the 'sub-arrays' have the same length. Additionally, your method can assume that all of the values in the input array are between 0 (included) and 365 (excluded). If there is a tie, pick whichever one you like.
   **Hint:** This can be done using many calls to the `countElement` method. (10 points)

5. Write a method `hasDuplicates` that takes as input an array of ints and returns whether or not there are any duplicates in the input array. Your method can assume that all of the values in the input array are between 0 (included) and 365 (excluded).
   **Hint:** This can be done in a few lines by putting the input array into an array of arrays so that you can use the `maxDay` and `countElement` methods. (10 points)

6. Write a method `runExperiment` that takes as input an integer corresponding to the size of each of the sub-arrays and uses the `generateAllData` and the `hasDuplicates` methods to determine which fraction of the random arrays contains duplicate elements. To do so, this method should call `generateAllData` with `iterations=200` and `range=365`. This method then returns the computed fractional value, and will `throw` an `IllegalArgumentException` if the input size is smaller than 1. (10 points)

As an example if `size` is 5, your method should call `generateAllData` to produce 200 arrays of `size` 5, all with values between 0 (inclusive) and 365 (exclusive). You should return the proportion of times that there is a duplicate element in those 200 arrays. An example computation is below with `size` 5. Note that we have only used 7 instead of 200 for `iterations` for the sake of space.

Suppose after calling `generateAllData` you have the following results returned:

```
{0 4 4 2 1},
{0 1 2 3 4},
{0 364 364 2 3},
{1 2 3 4 5},
{1 2 2 1 100},
{123 321 101 3 3},
{1 8 19 88 14}
```

Notice that the 1st, 3rd, 5th and 6th arrays contain duplicates. The others do not. Thus the method should return $\frac{4}{7}$.

7. In your main method, print a table of ratios obtained by repeatedly calling `runExperiment(int size)` for values of `size` from 1 to 100. Sample output is below (in your assignment it should go up to 100). Since there is some randomness in the results, your numbers may differ slightly. However, you must print a table like the below. (10 points)

```
> run BirthdayParadox
1 0.0
2 0.005
3 0.01
4 0.03
5 0.015
6 0.035
7 0.035
8 0.045
9 0.085
10 0.085
11 0.145
12 0.22
13 0.24
14 0.2
15 0.26
16 0.26
...
```

# Weather (40 points)

In this part of the assignment, you will define your own class and then write methods to use it.

**Question 2: Defining a new type: WeatherEntry Class** (20 points)

Define a class `WeatherEntry` that has the following private properties: `double temperatureInCelsius`, `boolean isSunny`. This class also has a public `constructor` that takes values for *both* of its attributes as input.

Additionally, your class has three public non-static methods. You may not add any other public methods besides these three and the above constructor. You may, however, add additional *private* methods. Finally, you must not add any additional properties.

1. `getTemperatureCelsius`. This method takes no input and returns the temperature in Celsius.

2. `isGoodWeather`. This method takes no input and returns whether or not the day is a nice day. A day in Montreal is nice if it is above -30 degrees Celsius and is sunny.

3. `display`. This method takes as input a boolean value `isCelsius` indicating whether the user would like to see the information in Celsius or Fahrenheit. It prints the temperature in the requested unit, whether or not it is sunny out, and if it is a nice day. For example:
   `It is 55 degrees Fahrenheit and is sunny.  It is a good day.`

**Question 3: Using your new type** (20 points)

You will now write methods in a different class, `WeatherUtilities`. Some of these methods will use Objects of type `WeatherEntry`.

1. Write a `static` method `countGoodDays` that takes as input a double array of temperatures in Celsius, and a boolean array of the sunniness status of days. Index i in both arrays stores the information for the $i^{th}$ day.
   This method counts how many days meet the `isGoodWeather` criteria from the previous question and returns that value. You *cannot* use any method from the `WeatherEntry` class when implementing this method.
   If the lengths of the input arrays are not the same, you must `throw` an `IllegalArgumentException`.

2. Now write a `static` method `countGoodDays` that takes as input an array of `WeatherEntry` objects. It uses the `isGoodWeather` method from the `WeatherEntry` class to count and then return the number of good days in the input array. You may assume that none of the `WeatherEntry` objects are set to null. You may also assume that the input array itself is not null.

3. In your main method, you will create an array of `n` WeatherEntry objects where `n` is specified by the user via *command line arguments*. It then uses a `Scanner` to obtain the temperature and sunny status of each day, one at a time, uses those values to create `WeatherEntry` objects and then places those values into the array.
   *After filling the array*, it counts how many good days are in the array, and displays the result. In addition, it finds and prints the highest and lowest temperatures. Sample output from this step is below:

   ```
   There were 10 nice days.
   The highest temperature was 30 degrees Celsius and the lowest was -20.
   ```

   Note that in order to get full marks for this question, you are required to have at least two loops. It is recommended (although not required) that you write `private` helper methods to compute max, min, and to count the number of good days.

## What To Submit

You have to submit one zip file called `Assignment3_YourName.zip` with all your files in it to MyCourses under Assignment 3.

These files must all be inside your zip. Double check to ensure you have included the `.java` source files rather than the .class files. Also, please ensure after submitting that you have received an email receipt from mycourses confirming that your submission was successful. If you have not received an email, you need to double check your submission.

```
BirthdayParadox.java
WeatherEntry.java
WeatherUtilities.java
```