ASSIGNMENT 4

COMP-202, Winter 2018, All Sections

Due: Tuesday, March 27th, 11:59pm

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

Question 1:	40 points
Question 2:	60 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 ID 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

As usual, do **NOT** *submit this part, as it will not be graded.*

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 *hsssss.* 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 `talk`.

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: Sort and Find (40 points)

For this question, you need to create a class called `SortAndFind`. Inside such class, you will write several methods to help you sort a 2-dimensional array of integers both row-wise and column-wise. You will also write a method that implements an algorithm to find an element inside the array. Throughout this question, we call a two-dimensional array a matrix, if all its sub-arrays have the same length.

To get full marks, you must write the following methods. Note that you are free to write any additional method if they help in the design or readability of your code.

(a) (5 points) **1a. A method to generate an array of random integers**

Write a method called `generateRandomMatrix` that takes as input two integers `m` and `n` indicating the dimensions of a two-dimensional array. The method must create an `m` by `n` array of integers. It must then populate all its elements with random integers between 0 (included) and 50 (excluded). To do so, create an object of type `Random` (remember that to use `Random` you should add the appropriate import statement at the beginning of your file). To make your program easier to debug (and grade), you must provide a *seed* equal to 123 for the Random object.

For example, with the seed of 123, `generateRandomMatrix(3,4)` returns the following array:

`[[32, 0, 26, 39], [45, 7, 34, 37], [35, 3, 39, 26]]`

The order in which you see the numbers might vary depending on how you initialized the elements of the array.

(b) (2 points) **A method to display all the element of a matrix**

Write a method called `displayMatrix` that takes as input a two-dimensional array of integers. The method should display all the elements of the array, one subarray per line. Separate each element using the tab character (`'\t'`).

Example, if you call `displayMatrix` with the array `[[32, 0, 26], [39, 45, 7], [34, 37, 35]]` as input, the following must be displayed:

```
32  0   26
39  45  7
34  37  35
```

If you call `displayMatrix(generateRandomMatrix(6,8))`, then the following must be displayed:

```
32  0   26  39  45  7   34  37
35  3   39  26  22  15  37  49
20  35  4   16  22  37  36  0
28  3   8   42  21  44  11  22
30  33  2   25  3   34  35  36
48  42  43  0   14  36  43  0
```

(c) (9 points) **A method to sort a 1-dimensional array**

Write a method called `sortOneRow` that takes as input a one dimensional array of integers. The method should not return any value. The method should sort the array in an increasing order. To do so, you **must** implement the following algorithm (called selection sort):

- Idea: consider the array as if it was divided into two parts, one sorted (on the left) and the other unsorted (on the right). Note that at the beginning the sorted part is empty. To implement this idea, you need to keep track of where the sorted part ends and where the unsorted part begins. For instance, you can keep track of the first index of the unsorted part of the array. Such index at the beginning will be 0.

- Procedure:
  - Find where the smallest element in the unsorted part of the array is positioned.
  - Swap that element with the element in the initial position of the unsorted part of the array.
  - Update the index indicating where the unsorted part of the array begins.

  Repeat the above steps until the entire array is sorted.

Consider the following simple array:

```
int[] x = {5, 3, 1, 4, 2};
```

Let's apply the procedure described above step by step:

1. At the beginning, the sorted part of the array is empty, while the unsorted part goes from the element at index 0 to the element at index 4.
   - First let's find the smallest element in the unsorted part of the array. The elements in the unsorted part of the array are $\{5, 3, 1, 4, 2\}$, therefore the smallest is 1.
   - Then, let's swap the element 1 with the element in the initial position of the unsorted part of the array. The initial position is at index 0, therefore we swap 1 with the element 5 and we obtain the following array $\{1, 3, 5, 4, 2\}$.
   - Now we can increase by 1 the index indicating were the unsorted part of the array begins. Initially this index was 0, now it becomes 1.
2. We need to iterate through the same steps a second time. Now the sorted part of the array is $\{1\}$, while the unsorted part is $\{3, 5, 4, 2\}$ (it begins with the element at index 1).
   - First, we find the smallest element in $\{3, 5, 4, 2\}$, which is 2.
   - Then, we swap the 2 with the first element in the unsorted part of the array (the 3). After the swap the array looks as follows: $\{1, 2, 5, 4, 3\}$.
   - Update the index indicating the initial position of the unsorted array from 1 to 2.
3. After you repeat the above steps for a third time, the array will be $\{1, 2, 3, 4, 5\}$ and the index indicating the initial position of the unsorted array will be 3.
4. When the index indicating the initial position of the unsorted part of the array is equal to the index of the last element of the array you can stop repeating the above steps and you'll be left with a sorted array. In this specific example, we would need one last iteration to make sure that the array is actually sorted.

Consider the following array:

```
int[] x = {28, 3, 8, 42, 21, 44, 11, 22};
```

Then, after the call `sortOneRow(x)` is executed, the array `x` will be as follows:

`{3, 8, 11, 21, 22, 28, 42, 44}`.

(d) (9 points) **A method to sort one column of a 2-dimensional array**

Write a method called `sortOneColumn` that takes as input a two-dimensional array of integers and an int indicating which column should be sorted. The method should not return any value. Implement the algorithm described above to sort the specified column. Be careful, in this method you need to work directly with a two-dimensional array. You can assume that the array received as input represents a matrix (i.e. all its sub-arrays have the same length).

Consider the following array:

`int[][] x = {{32, 0, 26}, {39, 45, 7}, {34, 37, 35}, {3, 39, 26}, {22, 15, 37}};`

Then, to sort the second column of the array we can make the following method call `sortOneColumn(x,1)`. After the call is executed, the array `x` will be as follows:

`{{32, 0, 26}, {39, 15, 7}, {34, 37, 35}, {3, 39, 26}, {22, 45, 37}}`

(e) (5 points) **A method to sort a 2-dimensional array**

Write a method called `sortMatrix` that takes as input a two-dimensional array of integers and does not return any value. Once again, you can assume that the array received as input represents a matrix. The method should sort the matrix first by sorting all its rows, and then by sorting all its columns. Consider the following array:

`int[][] x = {{32, 0, 26}, {39, 45, 7}, {34, 37, 35}, {3, 39, 26}, {22, 15, 37}};`

After the method call `sortMatrix(x)` is executed, the array will be as follows:

`{{0, 22, 32}, {3, 26, 37}, {7, 26, 37}, {15, 35, 39}, {34, 39, 45}}`

(f) (10 points) **A method to find an element in a 2-dimensional array**

Write a method called `findElement` that takes as input a two-dimensional array of integers and an int `n`. Assume that the array received as input is a matrix and it has been sorted. Then the method must find the position of the element `n` inside the matrix. The method returns an array of integers with two elements. If `n` is in the matrix than the elements correspond to the row and the column of `n`, otherwise the elements are both equal to -1. To find the element `n` inside the matrix, implement the following algorithm:

1. Start by looking at the bottom left element of the matrix.
2. Compare the element with the input.
3. If they are equal, then create/initialize the array with the correct postion and return it.
4. If `n` is smaller than the element you are looking at, then move up in the matrix by one position.
5. If `n` is greater than the element you are looking at, then move to the right in the matrix by one position.
6. Go back to step 2, until you either find the element or you get out of bounds.

If the element if not in the matrix, the method returns the array {`-1, -1,`}.

Consider the following array

`int[][] x = {{0, 22, 32}, {3, 26, 37}, {7, 26, 37}, {15, 35, 39}, {34, 39, 45}}`

Then:

- `findElement(x, 15)` returns {`3,0`} since the element 15 is in the fourth row, first column of the matrix. The method will start by comparing 15 with the bottom left element, in this case 34. Since 15 is smaller than 34, the method will move up by one position in the matrix and compare 15 with 15. The two values are equal, therefore the method can return the position of 15 inside the matrix.
- `findElement(x, 18)` returns {`-1, -1`} since there is no element equal to 18 inside the array. The method will start by comparing 18 with 34. Since 18 is smaller than 34, the method will move up by one position in the matrix and compare 18 with 15. Since 18 is greater than 15, the method will now move one position to the right inside the matrix and compare 18 with 35. Since 18 is smaller than 35, the method will move up and compare 18 to 26. Since 18 is smaller than 26, the method will keep moving up and it compares 18 to 26. It will then move up again and compare 18 to 22. 18 is smaller than 22 and therefore the method will move up by one

position once again. At this point, the matrix has ended and so the method can conclude that 18 is not in the matrix and return {`-1, -1`}.

- Finally, note that following the above procedure, `findElement(x, 26)` returns {`2, 1`} since the 26 in third row, second column, is the first 26 the algorithm will find.

(g) (0 points) **Set up the main method (optional)**

Inside the `main` method of the class `SortAndFind` do the following:

- Generate a random two-dimensional array of the desired dimensions using `generateRandomMatrix`.
- Display the array using `displayMatrix`.
- Sort the array using `sortMatrix`.
- Display the array again using `displayMatrix`.
- Test your `findElement` method by looking for different elements inside the array.

For example, when you run the class `SortAndFind` generating a 6 by 8 two-dimensional array you should see the following output:

```
> run SortAndFind
32  0   26  39  45  7   34  37
35  3   39  26  22  15  37  49
20  35  4   16  22  37  36  0
28  3   8   42  21  44  11  22
30  33  2   25  3   34  35  36
48  42  43  0   14  36  43  0

0   0   11  20  22  28  35  36
0   3   14  21  22  34  36  37
0   4   16  26  33  35  39  44
2   7   22  30  34  37  39  45
3   8   25  32  35  37  42  48
3   15  26  36  42  43  43  49

The element 14 is in position [1, 2]

The element 26 is in position [5, 2]

The element 5 is in position [-1, -1]
```

Note, the print statements were added in the main method.

Question 2: Bridge Bidding (60 points)

For this question, you will write several classes, and then create and use instances of those classes in order to code part of the card game Bridge. Bridge is a 4 players game that uses a standard deck of 52 cards. At the beginning, all the cards are dealt to the players, and each player receives 13 cards. The game of Bridge has two main parts: the *Bidding* and the *Play*. In the bidding phase, a player must first evaluate one's hand of card in order to figure out how good the hand is. To do this, players use a basic point-count system which is a combination of different points. In this assignment, you will do the following:

- Write a class to define a new type `Card` which is meant to represent a card in a game such as bridge, poker, or blackjack.

- Write a class to define a new type `Deck` which is meant to represent a deck of 52 cards.
- Write a `BridgeUtilities` class to store a set of methods needed to implement a basic point-count system for the game Bridge.
- Write a `Game` class where you'll be coding the first part of the Bidding phase in a game of Bridge.

Note:

- In addition to the required methods listed below, you are free to add as many other `private` methods as you want (no additional `public` method is allowed).
- It is up to you to figure out whether a method should be `static` or not based on its description.

(a) (12 points) Write a class `Card`. A Card has the following `private` attributes:

- A `String` suit.
- An `int` value.

There are four possible options for suits: hearts, spades, clubs, and diamonds. There are 13 possible options for the value: ace, two, three, four, five, six, seven, eight, nine, ten, jack, queen, and king. For this assignment, we will represent the value using an `int` which has a value between 1 and 13. The suit will be represented using a `String`.

The Card class also contains the following `public` methods:

- A constructor that takes as input an `int` for the value and a `String` for the suit, and initializes the attributes accordingly. The inputs are considered to be valid if they create a card of value between 1 and 13 (both included) and of suit equal to either *hearts*, *spades*, *clubs*, or *diamonds* (please ignore capitalization). If the input are invalid, then the constructor should throw an `IllegalArgumentException` explaining that no card of such type can be created.
- `getSuit()` which returns the suit of the card.
- `getValue()` which returns the value of the card.
- `print()` which takes nothing as input and prints the content of the card. For example "3 of spades" or "13 of clubs". Note that it is not necessary to print the words such as "king" or "queen". You can leave them as 13 or 12 if you want to.

(b) (18 points) Write a class `Deck`. A Deck has the following `private` attribute:

- An array of `Card`s.

The Deck class also contains the following `public` methods:

- A constructor that takes nothing as input and initializes the attribute with a `Card[]` of size 52. The constructor should also initialize such array with all 52 possible cards. To get full marks you must use at least 1 loop to accomplish this (that is, you cannot write 52 total statements to assign all possible values). Hint: Create a `String` array of size 4 with the suit values, and then use nested for loops to initialize the cards.
- `shuffle()` which takes no input and returns no value. This method should shuffle the array of `Card`s of *this* `Deck`. To do so, do the following:
  - create an object of type `Random` (remember that to use `Random` you should add the appropriate import statement at the beginning of your file). To make your program easier to debug (and grade), you must provide a *seed* for the Random object. Use a seed equal to 123.
  - Then create a loop that iterates 1000 times.
  - Inside the loop, generate 2 random integers between 0 and 51 (both included) and swap the values at those positions in the array.

For example, suppose the array of `Card`s of *this* `Deck` is the following:

```
[AH, 2H, 3H, 4H, 5H, 6H, 7H, 8H, 9H, 10H, JH, QH, KH, AS, 2S, 3S, 4S, 5S,
6S, 7S, 8S, 9S, 10S, JS, QS, KS, AC, 2C, 3C, 4C, 5C, 6C, 7C, 8C, 9C, 10C,
JC, QC, KC, AD, 2D, 3D, 4D, 5D, 6D, 7D, 8D, 9D, 10D, JD, QD, KD]
```

Where the first symbol denotes the value of the card and the second the suit (C stands for *clubs*, H for *hearts*, D for *diamonds*, and S for *spades*). Then, after the method `shuffle()` is appropriately called on such `Deck`, the array of `Card`s will look as follows:

```
[AD, 6D, AS, QH, 7H, 5D, 5S, KS, 9D, AH, 4D, 6H, 4H, 2H, 9S, 10D, JH, JD,
KD, 8H, 3C, 7S, 10C, 8S, JC, 10H, QS, 4C, 3H, 9H, 7D, 8C, KH, JS, 3S, 7C,
6C, 3D, 9C, 6S, QC, 10S, 5H, 5C, 4S, 2D, KC, 8D, 2S, 2C, QD, AC]
```

- `dealHand()` which takes as input two integers. The first one, call it `n`, indicates how many cards should be dealt, the second one indicates the player's number. The idea is that the first player should be dealt the first `n` cards in *this* Deck, the second player the next `n` cards, and so on. The method should return an array of `Card`s representing the hand dealt to the player. If there are not enough cards to deal (for instance we cannot deal a hand of 12 cards to a fifth player, since the first four players already took 48 cards, and only 4 cards are left in the deck), then the method should throw an `IllegalArgumentException` explaining that there are not enough cards in the deck.

(c) (20 points) Write a class `BridgeUtilities`. This class is not meant to represent a new data type. You will write this class simply to group together methods that are needed to implement a basic point-count system for the game Bridge. As such, this class has no attributes.

To evaluate the points associated to a hand of cards, we will be using a combination of high card points (HCP) and distributional points, as follows.

High Card Points. The top four cards in each suit are given point values relative to their rank as follows:

- Every Ace is worth 4 points.
- Every King is worth 3 points.
- Every Queen is worth 2 points.
- Every Jack is worth 1 point.

All other cards are worth 0. Remember that the values associated to an ace, king, queen, and jack are equal to 1, 13, 12, and 11 respectively.

Distributional Points. In addition to HCP, another factor that determines the strength of the a bridge hand is the length of each suit. Since there are 13 cards and 4 suits, a "balanced" hand would have 4 cards in one suit and 3 cards in the other three suits. In bridge, a "balanced" hand is a very weak hand. Thus, if the hand is "unbalanced" it is considered to be good. For this reason, the hand is worth 1 extra point for every card more than 4 in each suit. For example, if the hand has 5 cards of the same suit one extra point is given, if it has 6 cards of the same suit 2 extra points are given, if it has 7 cards of the same suit 3 extra points are given, and so on.

For example, suppose you were dealt the following hand:

`1C, 2C, JC, KC, 9C, 3H, 4H, 5H, 6H, 7H, KH, 1D, 1S`

Then, the evaluation of your hand would go as follows

- High Cards Points

3 aces (1C, 1D, 1S) = 3 * 4 = 12 points
2 kings (KC, KH) = 2 * 3 = 6 points
0 queens = 0 points
1 jack (JC) = 1 * 1 = 1 point

- Distributional Points

  5 clubs = 1 more than 4 = 1 point
  6 hearts = 2 more than 4 = 2 points
  1 diamonds = less than or equal to 4 = 0 points
  1 spades = less than or equal to 4 = 0 points

For a total of 12 + 6 + 1 + 1 + 2 = 22 points.

Inside the `BridgeUtilities` class write the following methods:

- A `private` constructor that takes no inputs and does nothing. By adding this constructor we make sure that no objects of type BridgeUtilities can be created from outside this class.
- A `private` method `countValue` that takes as input a `Card[]` and an `int` representing a specific card value. The method should return how many `Card`s in the array have the specified value.
- A `private` method `countSuit` that takes as input a `Card[]` and a `String` representing a specific card suit. The method should return how many `Card`s in the array have the specified suit. Note: make sure to ignore capitalization.
- A `public` method `countPoints` that takes as input a `Card[]` and returns the number of points that the hand is worth. The method should compute the points using the schema described above: HCP + Distributional points. To get full marks, you need to use the `countValue` and `countSuit` methods.

(d) (10 points) Write a class `Game`. This class is where the `main` method should go. Inside the `main` method do the following:

1. Create a `Deck`.
2. Shuffle the `Deck`.
3. Deal four hands of `Cards` (one per player). Each hand should contain 13 cards.
4. Compute how many points each hand is worth.
5. For each player, display the cards inside their hand and how many points their hand is worth.

For example, when you run the class `Game` you should see the following output:

```
> run Game
Player 1 was dealt the following hand of cards:
1 of Diamonds , 6 of Diamonds , 1 of Spades , 12 of Hearts , 7 of Hearts , 5
of Diamonds , 5 of Spades , 13 of Spades , 9 of Diamonds , 1 of Hearts , 4
of Diamonds , 6 of Hearts , 4 of Hearts
Their hand is worth 19 points

Player 2 was dealt the following hand of cards:
2 of Hearts , 9 of Spades , 10 of Diamonds , 11 of Hearts , 11 of Diamonds ,
13 of Diamonds , 8 of Hearts , 3 of Clubs , 7 of Spades , 10 of Clubs , 8 of
Spades , 11 of Clubs , 10 of Hearts
Their hand is worth 6 points

Player 3 was dealt the following hand of cards:
12 of Spades , 4 of Clubs , 3 of Hearts , 9 of Hearts , 7 of Diamonds , 8 of
Clubs , 13 of Hearts , 11 of Spades , 3 of Spades , 7 of Clubs , 6 of Clubs
, 3 of Diamonds , 9 of Clubs
Their hand is worth 7 points

Player 4 was dealt the following hand of cards:
6 of Spades , 12 of Clubs , 10 of Spades , 5 of Hearts , 5 of Clubs , 4 of
Spades , 2 of Diamonds , 13 of Clubs , 8 of Diamonds , 2 of Spades , 2 of
Clubs , 12 of Diamonds , 1 of Clubs
Their hand is worth 12 points
```

Note that if you are testing your program without a seed for the `Random` object from the `Deck` class, then the output should be different.

What To Submit

Please put all your files in a folder called Assignment4. Zip the folder (please DO NOT rar it) and submit it in MyCourses. Inside your zipped folder there must be the files listed below. **Do not submit any other files, especially .class files**.

`SortAndFind.java`
`Card.java`
`Deck.java`
`BridgeUtilities.java`
`Game.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.