ASSIGNMENT 3
Objects

COMP-202, Fall 2015

Due: Friday, November 20th, 2015 (23:59)

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:	40 points
Part 2, Question 2:	10 points
Part 2, Question 3:	40 points
Part 2, Question 4:	10 points
	100 points total

Free pass usage

If you wish to use your "free pass" (see the course webpage for more details), you are required to email the TA who is marking your assignment *before* the deadline. Please see the table below, which is referenced by the *last* name of the student, to determine your TA. Note that this is also the person whom you should see if you have questions or other complains about your marks.

TA Name	**Email Address**	**Student Last Name Range**
Stephanie Laflamme	stephanie.laflamme@mail.mcgill.ca	A-BAN
Teng Long	teng.long@mail.mcgill.ca	BAO-BUDD
Andrew Holliday	andrew.holliday@mail.mcgill.ca	BUDE-CHR
Carlos Gonzalez Oliver	carlos.gonzalezoliver@mail.mcgill.ca	CHS-DOW
Hua Qun (Robin) Yan	huaqun.yan@mail.mcgill.ca	DOX-GAG
Feras Abu Talib	ta_feras@hotmail.com	GAH - HARB
David Bourque	david.bourque@mail.mcgill.ca	HARC - JON
Paul Husek	paul.husek@mail.mcgill.ca	JOO - KOR
Babak Samari	babak@cim.mcgill.ca	KOS - LI, H
Seyyed Mozafari	sh.mozafari@mail.mcgill.ca	LI, I - MA
Egor Katkov	egor.katkov@mail.mcgill.ca	MAC - MUR
Chenghui Zhou	chenghui.zhou@mail.mcgill.ca	MUS - PARE
Jonathan Campbell	jonathan.campbell@mail.mcgill.ca	PARF - RENT
Neeth Kunnath	neeth.kunnath@mail.mcgill.ca	RENU - SHIM
Ayush Jain	ayush.jain@mail.mcgill.ca@mail.mcgill.ca	SHIN - TANG
Tzu-Yang (Ben) Yu	tzu-yang.yu@mail.mcgill.ca	TANN - WANG, Q
Mohammad Patoary	mohammad.patoary@mail.mcgill.ca	WANG, T - YANI
Xiaozhong Chen	xiaozhong.chen@mail.mcgill.ca	YAY- Z (end)

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 an input an array of integer arrays 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 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 cat `constructor` takes as input a String and sets that value to be the breed. The mood of a cat can be one of the following: `sleepy, hungry, angry, happy, crazy`. 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 random String of between 10 and 25 characters (letters).

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)
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.

Warm-up Question 5 (0 points)
Add to your `Vector` class a method calculateMagnitude() which returns a double representing the magnitude of the vector. The magnitude can be computed by taking

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

Warm-up Question 6 (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`. Question to consider: Would this approach work if we had a `double` as input instead of a `double[]`?

Warm-up Question 7 (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. 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?

Warm-up Question 8 (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.

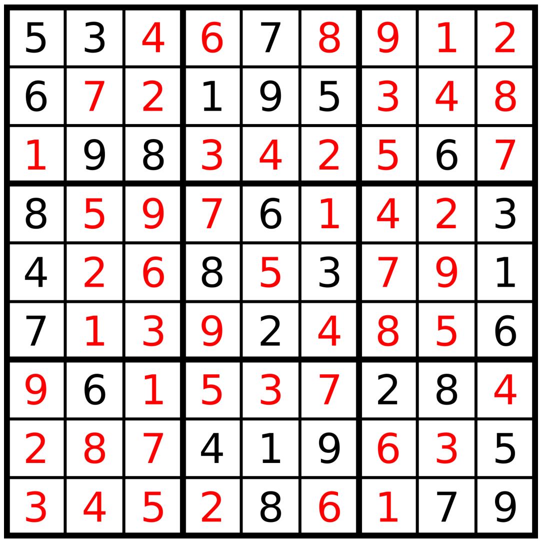
Beginning in this assignment, a 75% non-compilation penalty will apply. This means that if your code does not compile, you will receive a maximum of 25% for the question. If you are having trouble getting your code to compile, please contact your instructor or one of the TAs or consult each other on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

Question 1: Sudoku (40 points)

For this question, you will write a method called `isSudoku` to verify whether or not a two dimensional integer array represents a valid solution to a Sudoku puzzle. This method will return `true` if the solution is valid and `false` otherwise. This method, and all other methods for this question, should be written in a file called Sudoku.java.

A sample of a completed sudoku puzzle is shown in Figure 1.

You **cannot** use any methods from the `Arrays` class in solving this problem. You also **cannot** use hash tables or any other data structures not covered in class.



5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Figure 1: A Completed Sudoku Puzzle

In Sudoku, we need to fill in a 9×9 grid with digits 1-9 such that:

- Each column contains all of the digits exactly once
- Each row contains all of the digits exactly once
- Each of the 9 3×3 sub-grids must also contain all of the digits exactly once

In solving this problem, you are required to write the following 'helper' methods:

- A method, `sort`, that takes as input an integer array and returns a new array whose values are the same as the input array, but are in increasing order. Be sure to not change any of the values in the original array when you do this!
- A method, `uniqueEntries`, that takes as input a single array of integers and checks for duplicate values in the array. It returns `true` if all of the values in the array are unique and `false` otherwise. Note that this is much easier to do if you use the `sort` method from the previous question.
- A method, `getColumn`, that takes as input an array of integer arrays as well as an index, j and returns an integer array corresponding to the j^{th} *column* in the array of arrays.
- A method, `flatten`, that takes as input a square ($n \times n$) array of integer arrays and returns a single array of length $n \times n$. For example, if the original array is:

  $$\{\{1,2,3\},\{4,5,6\},\{7,8,9\}\}$$

  then the array returned by this method is:

  $$\{1,2,3,4,5,6,7,8,9\}$$

- A method, `subGrid`, that takes as input a square ($n \times n$) array of integer arrays, two indices, i and j, and a size m. This method will return an array of integer arrays of size $m \times m$ corresponding to the section in the original array that starts at the coordinates (i, j). For example, if the input array is:

  $$\{\{1,2,3\},\{4,5,6\},\{7,8,9\}\}$$

  and the size is 2, and the coordinates are (1,1), then the array returned by this method is:

  $$\{\{5,6\},\{8,9\}\}$$

Note that these methods do not check if the entries are valid (in the range 1-9), nor do they check the length of the input arrays. Your `isSudoku()` method should make these checks *before* calling any of the above methods. If desired, you can write additional helper methods as well.

A Deck of Cards

Background Information: Enums in Java

The code for this assignment will use something called an *enum*. An *enum* in Java is a user defined type that can hold a fixed set of values. In some ways, it is similar to a `boolean` in that a `boolean` can take on a fixed set of values (`true` or `false`). When you define an enum, you specify which values are acceptable. For example, one might wish to create a type `Day` which can represent one of seven values representing the days of the week. (The alternative would be to use an `int` to store a number representing the day of the week via a code. For example, you might choose to store the value of 0 for Sunday, 1 for Monday, 2 for Tuesday, etc. However, this solution is not clean because a)the compiler won't check that you didn't use an invalid value–for example a negative number–in your int and b)the code does not appear very clean since the type is `int` but in reality it does not represent a number.)

To define an enum, you simply list the set of values that it can take on. For example:

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}
```

defines a new type `Day` that can hold one of seven values.

There are several ways you can use the above enum:

1. To declare a variable of type `Day`, you would write, for example, `Day favourite;`
2. To use a literal value with an enum, you must write the enum name, then a dot, and then the literal value. For example: `Day.MONDAY`. You can use this value like any other value of the enum's type:

   ```
   Day currentDay = Day.MONDAY;
   //......
   if (currentDay == Day.MONDAY) {
       System.out.println("Ugh! I hate Mondays!");
   }
   ```

3. To go through every single value that an enum can take on, you can use a "foreach" loop. We have not seen the "foreach" loop in class, but the syntax is similar to a regular for loop. The body of the for loop will execute for every possible value of the term on the right side of the `:` Inside the loop, the variable declared on the left side of the `:` will take on each subsequent value. For example, to print every day of the week, you would write the following:

   ```
   for (Day d : Day.values())
   {
        System.out.println(d);
   }
   ```

   The above for each loop will first execute with `d` taking on the first value from `Day.values()` (SUNDAY), then the second, and so on.

4. Lastly, in some cases you might want to compare enums based on their order in the list for the enum definition. In this case, you can use a method defined on an enum called `ordinal` to compare the two. The method ordinal will return a number representing the order the value was listed in the enum definition. For example, to check between two `Day` variables `d1` and `d2` which comes earlier in the week, one could write:

   ```
   if (d1.ordinal() < d2.ordinal())
   {

   }
   ```

For more information about enums, see `http://docs.oracle.com/javase/tutorial/java/javaOO/enum.html` and `http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Enum.html`.

The assignment will use two enums, `Suit` and `Value` to represent the notion of a `Card`. These are defined for you and posted on the course webpage in the files `Suit.java` and `Value.java`. **For the code to work correctly you MUST keep these enum definitions in their own files. That is the definition of Suit must be in a file Suit.java and the definition of Value must be in a file Value.java.**

Question 2: Card class (10 points)

In this question, you will define a type `Card` that represents a playing card in Java. Remember that one of the goals of defining this new type is to allow you to do things such as create an array of `Card`s, which will be much easier to maintain than an array of values and an array of suits.

A `Card` object should have the following two private properties:

- private Suit suit : This is an enum of type Suit representing what suit (e.g. Hearts, Clubs, Spades, or Diamonds) the `Card` is supposed to represent.
- private Value value : This is an enum of type Value representing what value (e.g. ace, two, three,....jack, queen, king) the `Card` is supposed to represent.

A `Card` also has the following non static public methods:

- `public Card(Suit theSuit, Value theValue)` : This constructor should initialize the suit and value properties of the `Card` object. You may assume that `theSuit` and `theValue` are within acceptable ranges.
- `public Suit getSuit()` : This method should return the value of the `suit` property.
- `public Value getValue()` : This method should return the value of the `value` property.
- `public String toString()` : This method should return a `String` form of the `Card`. You may do this however you like, but your toString() method must produce a String that includes both the Suit and the Value of the Card. For example, "ACE of SPADES" or "AS" or "1S" are all acceptable options.

Question 3: CardPile class (40 points)

In this question, you will write a class `CardPile`. The point of this class is to define an object that will allow you to more easily manipulate a pile of cards. Remember that with a regular array, it is difficult to add an element to an array, since you can't resize an array after it has been created. With a `CardPile` object, we will be able to add and remove cards more easily. (Behind the scenes, we will still be dealing with an array, but the messy aspects are hidden and thus there will be an illusion of simplicity.)

A `CardPile` has two private properties:

- `private Card[] cards` : This is an array which always is of size 52 (you can make a class constant for this if you like) and is designed to store the entire deck. Typically, many of the spots will be null. This is done so that you can add and remove `Card`s to and from the `CardPile` without having to create a new array each time.
- `private int numCards` : This is an integer that represents the number of **non-null** values inside the array `cards`. This value should be initialized to be 0 when your `CardPile` is first created and then incremented or decremented every time you add or remove a `Card`.

As you write the methods below, you will need to ensure that array `cards` will stay the same size (52). The elements in the array from index 0 up to but not including `numCards` should be non null `Card`s and the elements from `numCards` until index 51 should all be set to be null.

You should have the following public methods (non static unless otherwise specified)

- `public CardPile()` : This is a constructor that will initialize the `cards` array to be an array of size 52. It also initializes the value of `numCards` to 0.
- A method `void addToBottom(Card c)`. This method should find the smallest index `i` for which the value of `cards[i]` is equal to `null` and place `c` in that position. The method will then update the counter `numCards`. (Hint: You may find this counter useful to help find the first null spot in the array!)
- A method `boolean isEmpty()` which returns a boolean representing whether the card pile is empty.
- A method `get(int i)` which takes as input an `int` and returns the `Card` object at that specific location. Note: If *i* is less than `cards.length` but greater than `numCards`, it is acceptable to either return null or for your code to produce a runtime error (since the index is out of bounds).
- A method `Card remove(int i)`. This method will remove the element located at index `i` from the private property `cards` and return its value. Note that after calling `remove` your array `cards` **must**

not have any "holes" in it. That is to say, none of the elements from position 0 to `numCards-1` should be null. Also note that you will need to update `numCards` after this method is called.

Hint: To do this, you should first store into a variable of type `Card` the `Card` at index i of `cards`. Next, you should *shift* all values at indices greater than `i` in the array one to the left, so that what had previously been at index `i+1` is now at index `i`, and what had previously been at index `i+2` is now at index `i+1`, etc. Then return the removed `Card`. For example, if `cards` contains {a, b, c, d, null} and `i is 1`, after the method is called, `cards` should end up with the contents of {a,c,d,null,null} and *b* is returned.

- A method `int find(Suit s, Value v)` which takes as input a `Suit` enum and a `Value` enum and returns an `int` representing the index of where the card with specified `Suit` and `Value` can be found in `cards`. If no such card exists, the method should return -1. (Note that -1 is chosen since it cannot be a valid array index in Java.)
- A method `String toString()`. This method should return a `String` representation of the `CardPile`. It should do this by calling the `toString` method on each particular `Card` and putting a number as well as a space between them. Note that when you use `String` concatenation (i.e. with the + sign), automatically the toString method is called with objects. For example, if the CardPile contains the Ace of Spades, Two of Hearts, and King of Clubs, the `toString()` method should produce the `String 0.AS 1.TH 2.KC` (an extra space at the end is OK). (If your `toString()` in the `Card` class is implemented slightly differently your results might look a bit different.)
- The last method you write will be a `static` method. This method will be called `makeFullDeck` and it creates a deck of size 52 which is filled in with all possible cards. It then shuffles the deck by calling the `shuffle()` method provided in the UtilityCode.java file on this `CardPile` and then returns the shuffled deck. **Update: To be clear, you need to return the `CardPile` object and NOT the `Card[]`.**

Question 4: A very simple game (10 points)

The purpose of this question is mainly designed to let you test your Card and CardPile classes. As you write this method, you may discover bugs in your other classes. Make sure to fix them!

In the next assignment, we will be working with the CardPile class again.

Write a program in which you write the logic for a very simple card game. First construct a full `CardPile` of cards (with all 52 cards). Then, via a command line argument (like we used on assignment 1), accept as input the number of players in the game. You may assume this number is at least 2.

Create `args[0] CardPile` objects and store them into a `CardPile` array.

Using the methods you wrote (`makeFullDeck`, `addToBottom`, `isEmpty`, `find`, and `remove` primarily, but perhaps others), "deal" the cards out, one at a time, to the `CardPile` objects so that each player gets roughly the same number of Cards. It is possible at the end that some players will have one extra card. (For example, if there are 52 cards and 5 players, then some players will have 10 cards and others 11 cards.)

The winner of the game is the player with the Ace of Spades. Write code to determine who has that card and print the winner's number to the screen. (This will help test your `find()` method.)

What To Submit

You have to submit one zip file with all your files in it to MyCourses under Assignment 3. If you do not know how to zip files, please ask any search engine or friends. Google might be your best friend with this, and a lot of different little problems as well.

These files should all be inside your zip.

`Card.java`
`CardPile.java`
`Sudoku.java`
`CardGame.java`
`confession.txt (optional)` - You should write in this file any information that you think is useful for the TA to mark the assignment. This should include things you were not sure of as well as parts of your code that you don't think it will work. Of course, like a confession, this will draw the TA's attention to the part of your code that doesn't work, but he/she will probably be more lenient than if he/she has to spend a lot of time looking for your error. It demonstrates that even though you couldn't solve the problem, you understand roughly what is going on.