CSCI 1101 - 2017
Assignment 3

Complete the assigned tasks and submit a ZIP file containing your finished source code (these are the .JAVA files) on Brightspace.

Your assignment **must be your own work**. You may ask questions of your instructor, course TAs, or Learning Centre TAs. If you discuss the ideas in this assignment with your classmates, **it is your responsibility to make sure you are not plagiarizing** – take no notes during the discussion, wait 30 minutes, and whatever you remember is safe to be used in your own code.

Submission Deadlines (firm):
Due: Sunday July 23, by 11:59pm (midnight)
Late submission (10% penalty): Monday July 24, by 11:59pm (midnight)

Submissions outside of these given deadlines are **not** accepted.

Before submitting, check that:

1. Your code is properly formatted, includes reasonable comments (including the header comment), and is properly tested for the given problem.
2. Your submission ZIP file contains your **source code** .JAVA files, not your compiled .CLASS files.
3. Your code **compiles** and can be run – even if your program does not do everything perfectly, make sure it compiles before you submit.

Header Comments

Your code should now include header comments for **all** of your class (.java) files. The comment should include the lab/assignment number, the course (CSCI 1101), the name of your program and a short description of the entire class, the date, your name and Banner ID, and a declaration that matches the first page of this document (e.g., whether you received help). See the example below for what a header comment should include:

```
/*Lab1, Question 1 CSCI 1101
   Student.java holds information about a student at Dalhousie in CSCI1101 and
   their grades
   June 29, 2015
   John Smith B00112345
   This is entirely my own work. */

public class Student {
//rest of Code
```

If applicable, your demo class should then also have a similar header:

```
/*Lab1, Question 1  - demo class CSCI 1101
   StudentDemo.java is a demo program for the Student class. It creates student
   objects, and compares different students.
   June 29, 2015
   John Smith B00112345
   I received help with creating Student objects from my TA but the rest is my
   own work. */

public class StudentDemo {
//rest of Code
```

Question 1
You will write an implementation of a simple card game called War.

Details of the game's rules can be found here:
http://www.bicyclecards.com/how-to-play/war/

This game uses a "standard" deck of playing cards, and involves two players. Each player is dealt half of a randomly shuffled deck (so 26 cards each). During each turn, both players simultaneously flip the top card of their deck. The player who flips the highest value card wins the turn, and "captures" both cards by adding them to the bottom of their deck.

If both players flip over cards with the same value, then a "war" begins: each player takes the next two cards from their deck, flipping over one and leaving the second face-down. The flipped cards are compared, and the highest value of *these* cards determines the winner, who now takes the "war pile" (all six cards). If these are also the same value, then the war continues; both players continue drawing two cards from their deck and flipping only the first one to compare values. The ultimate winner takes all cards in the "war pile". After a war, play resumes as normal by flipping and comparing only one card each at a time.

Your game should use three class: Card, Deck, and Demo.

Card

- A Card object is made up of a suit (Hearts, Diamonds, Clubs, or Spades) and a rank
    - The ranks, in order from lowest to highest, are: 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack. Queen, King, Ace
    - So nothing beats an Ace, and a 2 doesn't beat anything
- The Card class should internally track which values are valid for both the suit and for the rank, so that another class can discover the valid suit/rank values even before creating a new Card.
- There should be a comparison method that determines which of two Cards has the higher rank (note that suit is irrelevant when comparing rank; the Ace of Spades is "worth" as much as an Ace of Hearts)
- There should be a toString method that gives a reasonable representation for the Card
    - You may find it easier to assign numerical values *internally* for each of the ranks Jack, Queen, King, and Ace, but you will still want these String names used in the toString output

Deck

- A Deck object stores a collection of Cards in an ArrayList. Write methods that are related to an Object that should store a deck of cards, including a way to create the deck and populate the ArrayList with one copy of each valid Card suit/rank combination, appropriate get/set methods, and (importantly) a shuffle method.
    - Your simulated Deck of playing Cards should be populated with one of each type of card (a unique suit/rank pairing); valid combinations should be determined by accessing the Card class itself
    - The shuffle method should reorder the cards in the internal ArrayList into a random order. There are many ways to achieve this; you should research a simple approach for randomizing the order of the Cards.

Demo

- The Demo should ask for two player names, create a full Deck of Cards, and shuffle the deck. This should then be "dealt" as two hands to the two players (giving each player their own pile of 26 cards).

You may decide on your own method of storing each player's pile of Cards; using an ArrayList is one approach.

- Play the game by automatically processing each turn (no further player input is required) in sequence
    - The game is over when one player has all the Cards in their pile, and this player is then the winner
- At the end of the game, ask the players if they would like to play again (with a Yes/No prompt) and if so, start a new game with a newly shuffled Deck.

You may use any other methods or classes not defined here, if it makes your task easier. **HOWEVER**: you are solely responsible for documenting these so that their purpose is clear and you receive full marks!

Sample output is provided below. Your program does *not* need to look identical, but it *should* provide the same level of information.

```
Welcome to WAR! Let's Play!

Enter the first player's name: BOB
Enter the second player's name: SUE

BOB                                  #Cards           SUE                              #Cards           Winner
2  Clubs                             26               Jack of Hearts                   26               SUE (wins 1 card)
9  Clubs                             25               Ace of Clubs                     27               SUE (wins 1 card)
King Spades                          24               Jack of Spades                   28               BOB (wins 1 card)
6  Spades                            25               6 of Diamonds                    27               WAR
**********************************************WAR*********************************************

4  Spades                            25               6 of Hearts                      27               SUE (wins 3 cards)

********************************************END WAR*******************************************
9  Hearts                            22               10 of Diamonds                   30               SUE (wins 1 card)

… <many more lines> …


5 Clubs                              1                Queen of Spades                  51               SUE (wins 1 card)

SUE Wins! Congratulations
Play again (y/n)? n
```