CS 4773
Object Oriented Systems
Fall 2018

Assignment 2: War
Due: Sunday Oct. 7th by midnight

DESCRIPTION:
In this assignment, you will design and build an object-oriented implementation of 3 variations of the card game, War.

The basic rules for the game are located here:
`http://www.bicyclecards.com/how-to-play/war/`

These are the rules for one variation of War. Your project must handle 3 variations of War:

a. placing won cards on the bottom of the player's hand (you will need to have a maximum # of iterations because this game variation can last a ridiculously long time). Winner is player with most cards in hand at end of game.
b. placing won cards in a separate points pile. Winner is player with most cards in points pile at end of game.
c. 3-player war placing won cards in a separate points pile. The rules for 3-person war are located here: `https://en.wikipedia.org/wiki/War_(card_game)`. Winner is player with most cards in points pile at end of game.

***** IMPORTANT: YOUR PROGRAM MUST IMPLEMENT AT LEAST 4 OF THE GRASP AND/OR SOLID PRINCIPLES WE HAVE COVERED!** Include in your project a README.TXT file that lists and explains which classes are implementations of which principles.

Your game implementations must log gameplay to the console. All 3 variations should allow for the possibility of a tie and your output should indicate if a tie occurs.

Example 1 using a full deck:

```
Bob plays EIGHT of DIAMONDS as up card
Sue plays EIGHT of HEARTS as up card
War!
Bob plays FOUR of CLUBS as up card
Sue plays TWO of HEARTS as up card
Bob wins the round
Score is Bob 4, Sue 0
Bob plays TWO of SPADES as up card
Sue plays SIX of HEARTS as up card
Sue wins the round
Score is Bob 4, Sue 2
```

```
…
Score is Bob 20, Sue 32
Sue wins!
```

Example 2 using a deck with 2 cards of the same rank:

```
Bob plays EIGHT of HEARTS as up card
Sue plays EIGHT of CLUBS as up card
War!
Tie game!
```

Your code must follow all of the readability rules you used in Assignment 1, AS WELL AS:

a. your Java project must have AT LEAST 8 .java files (including Enums but excluding JUnit files)
b. no .java file may be more than 150 lines long (including blank lines and comments)
c. no .java file may be fewer than 3 lines long (including blank lines and comments)
d. error handling can only be accomplished using unchecked exceptions

You are required to create 10 JUnit test cases that test game variation outcomes WITH A FIXED DECK. This means that your card deck implementation must allow for the traditional 52 cards with a shuffle, as well as manually assigning it a set of known cards (e.g., a 2 card deck, a 3 card deck, a 4 card deck, etc.). If you know the order of the cards in the deck, then you know the expected outcome of the unit tests.

SUGGESTED APPROACH:

1. break out the functional requirements and data attributes
2. create a UML class diagram where you assign some of the functions and attributes for just a few objects and draw relationships between the objects. start simple: with the cards and the deck
3. implement what you designed in #2
4. create some unit tests for what you did in #3
5. repeat #s 2 through 4 until you have a testable first variation of War
6. now design, build, and test a second variation but be sure to use polymorphism and share code with the first variation
7. repeat until done

DELIVERABLES:

1. Name your Eclipse project CS4773Assignment2
2. Put a PNG or JPG copy of your UML diagram in your Eclipse project and call it assignment2UML.png or .jpg
3. If you worked on a team, be sure to include a file called TEAM.TXT that has ALL team member names and the total time in hours that it took you to complete this

assignment.

4. Export the entire project to a zip file called assignment2.zip and submit the resulting zip file to Blackboard.

LATE POLICY:

-10 pts for every day late

RUBRIC

30 pts	All 3 variations of War execute and output required log messages
30 pts	Object-oriented design including use of 4 GRASP and/or SOLID principles, and satisfies # of java files and file size requirements and unchecked exceptions for error handling
20 pts	Code is readable
10 pts	UML class diagram
10 pts	At least 10 JUnit test cases and all pass