CS 4773
Homework Assignment 2: War
Due: in class, Tuesday, October 4

Note: You may work with one partner on this assignment.

You may submit to Blackboard as many times as you wish up to the deadline. However, all Blackboard submissions for your team must be from the same team member.

DESCRIPTION
In this assignment, you will design and build an object-oriented implementation of 3 variations of the card game, War. The basic rules are available at: https://www.pagat.com/war/war.html

Your project must handle these 3 variations of War:

1) The two-player version initially described at https://www.pagat.com/war/war.html with one change. The game ends as soon as one of the following conditions is met: a) one player has won all of the cards, or b) a maximum number of rounds has been played. The winner is the player with the most cards at the end of the game. If a player does not have enough cards to complete a war, use the first listed possibility for ending the game.

2) A different version for two players. Cards that are won are placed in a separate points pile. Cards are not recycled. The game ends after the players use the cards that were initially dealt to them. The winner is the player with the most cards in his/her points pile at the end of the game. If the players do not have enough cards to complete a war, that round is a draw and they keep their cards.

3) The three-player war game, but with one change. Cards that are won are placed in a separate points pile. The winner is the player with most cards his/her in points pile at end of game. If players do not have enough cards to complete a war, use the first listed possibility for ending the game.

Game output is written to standard output. For each round, indicate the cards played by each player as well as which player won the round, and all players' scores. For variation 1, the players' scores are the number of cards in their hands. For variations 2 and 3, the scores are the number of cards in their points piles. For example:

```
Player 1 plays THREE of SPADES
Player 2 plays JACK of DIAMONDS
Player 2 wins the round
Player 1 has a score of 8
Player 2 has a score of 6
Player 1 plays SEVEN of HEARTS
Player 2 plays SEVEN of CLUBS
*** WAR! ***
Player 1 plays TEN of CLUBS
Player 2 plays FIVE of HEARTS
Player 1 winds the round
Player 1 has a score of 14
Player 2 has a score of 6
```

At the end of the game, print the name of the winner or if it is a tie, then declare that the result is a tie.

For this project, you must:

1. Create a domain model for the War card game. Your Java implementation must then have at least four classes that correspond to four different important conceptual classes of your domain model. The 3 variations of War count as a single conceptual class not 3.

2. Have two or more packages with at least one package that does not import classes from any of the project's other packages (reducing coupling)

3. Submit clean, readable code just as in the first assignment. Follow the clean coding practices from homework assignment 1 with the exception that no .java file may be longer than **150** lines including blank lines and comments. Error handling must be done with unchecked exceptions. In Java, unchecked exceptions are instances of the RuntimeException class (or a subclass of RuntimeException).

4. Implement a total of at least 4 of the SOLID and/or GRASP principles. Include a README.txt file in your project that lists and explains which classes are implementations of which principles.

5. Produce a Java class diagram (as a .png or .jpg file) of all your classes in their respective packages and the dependencies between them.

6. Include 10 Junit tests that exercise your game logic without using the user interface. The tests should use pre-arranged decks of cards so that there is an expected outcome to a game. These pre-arranged decks can be of any size, so you can have simple games such as where each player has 1 card. Your implementation of a deck of cards should support this as well as producing random shuffles on a standard deck of cards. These Junit tests must execute and be passed when Maven builds your package.

Suggested approach:

1. Create the domain model by examining the rules and perhaps playing a game of War.
2. Break out the functional requirements and data objects.
3. 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.
4. Implement what you designed in step 3.
5. Test what you did in step 4.
6. Repeat steps 3 through 5 until you have completed the first variation of War.
7. Design, build, and test the second and third variations making sure to use polymorphism and share code with the first variation.

DELIVERABLES

1. Make this assignment its own Maven project called cs4773-hw2
2. Put a copy of your domain model (as a png or jpg file) in your project and call it hw2DM
3. Put a copy of your Java class diagram (as a png or jpg file) in your project and call it hw2CD
4. Zip up the following in a file called hw2.zip
    a) Your src directory
    b) Your hw2DM file
    c) Your hw2CD file
    d) Your README file
    e) Your pom file
5. If you worked on a team, be sure that your name is included in the comments area on Blackboard when your work is submitted
6. Print your source code listing created by the DirToPDF tool

RUBRIC

-15 pts	Not handing in a listing created by the DirToPDF tool
20 pts	All variations of War execute correctly and gameplay is sent to standard output
20 pts	A domain model describing the game of War where at least four important conceptual classes are represented by different Java classes
20 pts	At least one package is decoupled from the rest of the project
10 pts	Design uses 4 GRASP and/or SOLID principles
10 pts	Code follows clean code principles
10 pts	A Java class diagram showing all classes of your project
10 pts	At least 10 Junit test cases and all are passed