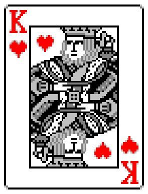
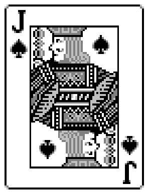
CMSC 202 — Fall 2013



WAR! A Two Player Card Game



Assigned	Thursday, October 3th
Program Due	Thursday, October 17rd by 8:00am
Weight	7%
Updates	

Objectives

To gain experience in the following areas.

- Using composition in an object-oriented design
- Interfacing with pre-written code
- Using enumerated types
- Writing unit tests

To continue to gain experience in the following areas.

- Working with the object-oriented paradigm
- Documenting classes
- Documenting methods

Project Description

Perhaps the simplest card game in the known universe is the kid's game of "War". In this simple game, a standard 52-card deck is shuffled and cards are dealt to each of the two players, alternating between the players, who place each card on the top of their pile. In each turn, the players both show the face of their top card. The player with the higher ranking card wins his opponent's card and puts both cards on the bottom of his pile. If the two cards are of the same rank, WAR breaks out. Each player places three (3) cards face down while yelling "I DE-CLARE" as loud as possible, then turn over the next (4th) card while shouting "WAR!!". The player with the higher ranking 4th card wins the two (2) cards that originally tied, all six (6) cards that were face down, plus the two (2) cards that determined the winner of the WAR -- ten (10) cards in all. Of course, if both players' 4th cards are the same rank, there's another immediate WAR. Play continues until one player has all the cards.

In this project, you will implement classes necessary to play a variation of WAR. In this simplified variation, the two players will turn over all cards in their piles just once. At the end of the game, the player with the most cards will be declared the winner.

For each turn, each player's name, the card he/she shows, and the result of the turn should be displayed. At the end of the game, the number of cards won by each player, the number of WARs won by each player, and the total number of WARs during the game should be displayed.

Sample output from an actual run of this project is shown below.

```
Welcome to WAR!!
Please enter player 1's name: Bob
Please enter player 2's name: Mary
Please enter the RNG seed for shuffling: 123456

Turn  1
-------
Bob shows three of Spades
Mary shows three of Hearts
WAR!!



Turn  2
-------
Bob shows four of Spades
Mary shows jack of Diamonds
Mary wins 10 cards



Turn  3
-------
Bob shows eight of Spades
Mary shows eight of Hearts
WAR!!



....



Turn 19
```

```
-----
Bob shows five of Diamonds
Mary shows jack of Spades
Mary wins 2 cards


Turn 20
-----
Bob shows three of Diamonds
Mary shows ace of Hearts
Mary wins 2 cards


Game Over!!
There were 2 wars
Bob won 22 cards and 1 war(s)
Mary won 30 cards and 1 war(s)
Winner: Mary
```

Project Specification Details

You will be given `Project2.java` that has `main( )` already written. You will also be given two files, `Rank.java` and `Suit.java` that contain enumerated data types for card ranks and suits, respectively. You will be expected to write the following classes.

- Player - Represents a single player of the game
- Game - Plays the game
- CardPile - Represents the cards in a single player's pile
- Deck - Represents the full 52-card deck
- Card - Represents a single card

  WAR uses a single standard 52-card deck. There are four (4) suits (hearts, diamonds, clubs, and spades). In each suit there are thirteen (13) ranks. In order from highest to lowest the ranks are ace, king, queen, jack, 10, 9, 8, 7, 6, 5, 4, 3, and 2. Suits are irrelevant in WAR.

  So that all students play the same game (i.e. the cards are dealt in the same order), it is necessary that the deck of cards be initialized appropriately before shuffling. Initialize the deck of cards in this order starting ***from the top of the deck*** -- ***Ks, Qs, ... 2s, As, Kh, Qh, ... 2h, Ah, Kd, Qd, ... 2d, Ad, Kc, Qc, ... 2c, Ac*** . Looping through the arrays returned by `Suit.values( )` and `Rank.values( )` when creating Cards to add to the deck will result in the desired ordering.

  The algorithm below, which shuffles an array of integers, must be adapted and implemented to shuffle the cards to start a new game. This algorithm is known as the Fisher-Yates algorithm, was first designed for computers by Richard Durstenfeld, and popularized by Donald Knuth.

1. A[0], A[1]... A[n-1] is an array of integers
2. Let N = n-1
3. Pick a random index, k, between 0 and N (inclusive).
4. Swap the values of A[k] and A[N].
5. Decrease N by one.
6. Repeat from step (c) until N is less than 1.

To choose a random index (step 3), use the `nextInt( )` method of the random number generator class `Random` found in the Java API. If you seed the random number generator with the value 123456, the shuffled deck and player cards should be as follows and the result of playing the game should match the sample output above.

After shuffling, the first 10 cards ***at the top of the deck are*** 3d, Ah, 5d, Js, Td, Qs, 6c, 4d, Kc, and 9h.
After dealing the cards, player 1's first 5 cards starting from ***the top of his/her pile*** are 3s, Ac, Qh, 7c, and 4s. Player 2's first 5 cards starting from ***the top of his/her pile*** are 3h, 3c, 2c, 8c, and Jd.

The three classes that are provided for your use are located here.

- Project2.java
- Suit.java
- Rank.java

You can simply cut and paste these classes into classes in your `proj2` package.

Requirements, Tips, and Hints

1. The package for this project must be named `proj2`.
2. You must implement `main` in your Deck class to perform unit testing. For this project, you only need to demonstrate unit testing for the Deck class. However, be thorough!
3. All classes that you write must have a class header comment, including the class's invariants.
4. All methods of all classes that you write must have a method header comment according to the CMSC 202 coding standards.
5. In theory, a WAR may break out when the players don't have enough cards to complete the WAR as described above. This will of course depend on the RNG seed you use for shuffling the deck. This situation WILL NOT arise when using the seed value 123456 and we gurantee that it will not happen when your project is graded, so there's no need to design or code for this case.
6. When finished with your project, transfer all of the .java files to the GL system. Compile the code, run i and test it thoroughly.

7. Assuming that your deck is implemented as an array of Cards, you may find coding easier if the highest array index is considered the "top" of the deck.

Grading

See the course website for a description of how your project will be graded.

Project Submission

1. **submit** all .java files you create. DO NOT submit Project2.java, Suit.java, Rank.java.
2. **submitls** to verify that the files you submitted are in the remote directory.

The order in which the files are listed doesn't matter. However, you must make sure that all files necessary t compile and run your project are submitted. You need not submit all files at the same time. You may resubm your files as often as you like, but only the last submittal will be graded and will be used to determine if your project is late. For more information, see the projects page on the course website.
More complete documentation for submit and related commands can be found here.
Remember -- if you make any change to your program, no matter how insignificant it may seem, you should recompile and retest your program before submitting it. Even the smallest typo can cause compiler errors ar a reduction in your grade.
Avoid unpleasant surprises!