CS/COE 0445 Spring 2015 Assignment 1

Online: Wednesday, January 14, 2015
Due: All source files plus a completed Assignment Information Sheet zipped into a single .zip file and submitted properly to the submission site by **11:59PM on Wednesday, January 28, 2015** (Note: see the submission information page for submission details)
Late Due Date: 11:59PM on Friday, January 30, 2015

Purpose: To refresh your Java programming skills and to emphasize the object-oriented programming approach used in Java. Specifically, you will work with control structures, class-building, interfaces and generics to **create** and **utilize** a simple array-based data structure.

Goal 1: To design and implement a simple class MultiDS that will act as a simple data structure for accessing Java Objects. Your MultiDS class will primarily implement 2 interfaces – PrimQ<T> and Reorder. The details of these interfaces are explained in the files PrimQ.java and Reorder.java. **Read these files over very carefully before implementing your MultiDS class.**

Goal 2: To utilize your MultiDS class by implementing a simple version of the card game "War". In this case your program will be a client using MultiDS and the details of the MultiDS implementation will be abstracted out.

Details 1: For the details on the functionality of your MultiDS class, carefully read over the files PrimQ.java, Reorder.java and Assig1A.java provided on the Web site. You must use these files as specified and **cannot remove/alter any of the code that is already written in them**. There are different ways of implementing the PrimQ<T> and Reorder interface methods, some of which are more efficient than others. Try to think of the best way of implementing these methods in this assignment, but the most important thing at this point is getting them to work. I recommend a LOT of pencil and paper work before actually starting to write your code. Later we will discuss the relative merits of different implementations.

After you have finished your coding of MultiDS, the **Assig1A.java** file should compile and run correctly, and should give output identical to the output shown in the sample executions (except for the segments where the data is shuffled, since it will be pseudo-random in that case).

Details 2: War is a card game played often by children that has many variations. You will implement the simple version as described below:

- Initially shuffle a 52-card standard deck of cards
- Deal the cards out completely to two players, alternating cards to each player
- Do the following until one player is out of cards or until a set number of rounds have completed:
    - Each player plays the top card in his / her hand
    - If one player's card beats the other (has a higher rank – suits don't matter), that player puts both cards into his discard pile
    - If the players tie, it is a WAR, so do the following until the WAR is resolved:
        - Each player plays a card without comparing
        - Each player plays one more card and compares in the same way as above
        - The winner of the WAR takes all 6 played cards (initially compared cards, uncompared cards, second compared cards)
        - If the WAR cards also yield a tie, repeat the process (one uncompared card, one compared card) until there is a winner

The following rules also apply to the game:

- If at any point in the game a player's hand is empty, the player should move the cards in his discard pile into his hand and shuffle them.
- If at any point in the game (even in the middle of a round) a player has no cards remaining in either his /

her hand or his / her discard pile, he / she has lost the game.

- If the both players have cards remaining after a set number of rounds (can vary), the player with the most cards (hand + discard pile) is the winner. In this case, it is possible for a tie to occur, with each player having 26 cards.

Implementation Requirements:

- Your initial card deck, the player's hands, the player's discard piles and the cards "in play" must all be stored in MultiDS<Card> objects.
- The Card class must be used as provided and cannot be changed.
- The maximum number of rounds for the game must be read in from the command line

You must submit in a single .zip file (minimally) the following 6 complete, working source files for full credit:

PrimQ.java Reorder.java
Assig1A.java Card.java

the **above four files** are given to you and must not be altered in any way.

MultiDS.java War.java

the **above two files** must be created so that they work as described. If you create any additional files, be sure to include those as well.

The idea from your submission is that your TA can unzip your .zip file, then compile and run both of the main programs (Assig1A.java and War.java) **from the command line** WITHOUT ANY additional files or changes, so be sure to test it thoroughly before submitting it. If you cannot get the programs working as given, clearly indicate any changes you made and clearly indicate why (ex: "I could not get the reverse() method to work, so I eliminated code that used it") on your Assignment Information Sheet. You will lose some credit for not getting it to work properly, but getting the main programs to work with modifications is better than not getting them to work at all. See the CS 0445 Web site for an Assignment Information Sheet template – you do not have to use this template but your sheet should contain the same information. **Note: If you use an IDE such as NetBeans to develop your programs, make sure they will compile and run on the command line before submitting – this may require some modifications to your program (such as removing some package information).**

Hints / Notes:

- See program A1Help.java for some help with the Card class and using the MultiDS class with Card objects.
- See file A1Out.txt to see how your output for Assig1A should look. As noted, your output when running Assig1A.java should be identical to this with the exception of the order of the values after being shuffled.
- See file WarOut.txt for an example run of my War.java program. Your War program output does not have to look exactly like this but the functionality should be the same.
- Your MultiDS class will need to allow for a variable number of objects, up to some maximum number (set by the initial size). You can implement this by having an integer instance variable (ex: "count") that indicates how many slots in the MultiDS are filled. For example, you could have a MultiDS with a capacity of 52 (for your hands) for a hand with anywhere from 0 up to 52 Cards in it. In this case, the array length would be 52 but the "count" would be some value between 0 and 52. When implementing MultiDS be careful to use the "count" value rather than the array length when doing operations such as the toString() method.