CSC 130
Programming Project #3

Due Date: by *11:55pm* Tuesday, March 24, 2015

Objectives:

- To practice creating and manipulating an array of objects
- To practice creating and using a linked data structure
- To practice creating and using multiple classes
- To practice creating and using programmer-defined exception classes
- To practice testing programmer-defined classes

The Problem:

The purpose of this project is to add functionality to the Deck class and to define a DiscardPile class so that together they can be used by a greater variety of card games. The Card class will also be modified so that Jokers can be created. Both the Deck and DiscardPile classes will have programmer-defined exception classes associated with them. The application class will be responsible for fully testing these classes. Your ability to show that the constructors/methods defined within these classes work in all situations will constitute a portion of your final grade. You do not need to test constructors/methods that were provided, but not modified.

Specifications:

You ***must*** use the method names specified below as well as the project and class names specified. All classes will be part of a project called ***proj3sp15*** (recall that the project name should be all lower-case letters). Including the classes in a single project will make it easier for you to submit everything together as a .zip file.

A partial implementation of the **Card** class will be provided. The following behaviors should be modified. You should not add/modify any other methods to/in this class. Be sure to modify the Javadoc comments as necessary.

- The createJoker method should be added as a static method and is responsible for creating and returning a Joker. A Joker is ranked higher than any of the other cards (value = 14) and should have "Joker" stored as its suit.
- The toString method should be modified so that it returns the string "joker" accordingly for a Joker.

In addition to the methods that are already defined, the **Deck** class will be modified or enhanced to provide the following behaviors. It should *not* be defined using generics and *is* a subclass of Object (i.e. it does not extend another class).

- A **parameterized constructor** should be added and is responsible for initializing a multi-deck Deck object that may contain jokers. The number of decks and the number of jokers are specified by the user.
- The **shuffleDeck** method should be modified so it no longer assumes that the deck is full. It should shuffle only the cards currently stored in the deck.

- The **dealCard** method should be modified so that a DeckException is thrown when the deck is empty. The exception should contain an appropriate message.
- The **cut** method is to simulate the cutting of a deck. You can generate a random number in the appropriate range and use this number to determine the place where the deck should be cut. If the deck is empty, a DeckException should be thrown and should contain an appropriate message.
- The **putCardsInDeck** method will accept a reference to a Card array. The cards in the array are to be removed and stored in the deck, leaving the array empty (be sure each element of the array is set to null.) The cards removed from the array should be stored at the *bottom* of the deck. A DeckException should be thrown if putting cards back into the deck causes its size to be greater than the maximum size of the deck.
- The **isEmpty** method determines if the deck is empty.
- The **toString** method should be modified so that it returns a string similar to the one below:

  Card 1: Ace of heart **// In particular, you should have the label**
  Card 2: 10 of diamonds // **You should display the topmost card first**
  Etc.

- Be sure to modify the JavaDoc comment at the top so that it correctly describes this new, enhanced deck class and includes your name as the author.
- Modify the JavaDoc comment for all methods that have been modified.
- Include JavaDoc comments for all new methods.

The **DiscardPile** class should be defined as follows.

- The DiscardPile class will define a single, *private* instance variable. It will store the address of the linked list that stores the cards in the discard pile.
- The **default constructor** is responsible for creating an empty pile. Initially, it does not contain any cards.
- The **addCard** method will accept a card to be stored on top of the discard pile.
- The **removeCard** method is responsible for removing and returning the top-most card from the discard pile. A DiscardPileException should be thrown if the discard pile is empty and should contain an appropriate message.
- The **removeCards** method is responsible for removing and returning, in a Card array, the number of cards specified by the user. A DiscardPileException should be thrown if the discard pile does not contain the number of cards the user wishes to remove. The exception should contain an appropriate message.
- The **size** method will return the number of cards currently in the discard pile.
- The **isEmpty** method determines if the discard pile is empty.
- The **toString** method will return a reference to a String object that contains the cards, from top to bottom, currently stored in the discard pile. The string, when printed, should be similar to the below:

  Card 1: Ace of hearts **// In particular, you should have the label**
  Card 2: 10 of diamonds

Etc…

- Be sure to include a JavaDoc comment at the top and JavaDoc comments for all constructors/methods.

The **Node** class will *not* be defined using generics. The decision not to use generics will allow you to focus on the manipulation of the linked list itself without the additional syntax.

The **DeckException** class will include only a parameterized constructor. The constructor accepts the message to be stored in the object. An object of type DeckException is *unchecked*.

The **DiscardPileException** class will include only a parameterized constructor. The constructor accepts the message to be stored in the object. An object of type DiscardPileException is *unchecked*.

The application class, **Proj3Ap**p, will be used to test the methods in these classes. Your ability to show that you have fully tested all of the new/modified methods will constitute a portion of your grade for this assignment. You should test all possible situations, for example, calling dealCard/shuffleDeck/cut on an empty deck, calling dealCard/shuffleDeck/cut on a deck that is not full or one that contains only a few cards, calling removeCard(s) on a discard pile that does not contain enough cards, and calling putCardsInDeck on a deck that would cause the deck to become larger than its maximum size, for example. These are only some examples, be careful to consider others while testing your code.

You can begin by creating the Card and Deck classes, then copying the code for these from Blackboard into the appropriate class. Recall that a package name should be specified when creating classes and that it should have the same name as the project. It is imperative that you follow these guidelines when creating the project and class files so that my test code can be utilized without modifications.

Citation Policy:

This assignment is to be completed individually (not with your pair programming partner). If you receive substantial help from another person you must give them credit for their contribution in a comment. Failure to do so is considered plagiarism.

General Requirements:

- You should spend some time making your I/O as neat as possible. The overall appearance of the I/O will be graded.
- Neatness Counts!!! Proper indentation is required.
- Be sure to select meaningful variable names.
- All classes should have a comment at the beginning that includes a title, a description, and your name as the author. Include plenty of comments in the application class. All constructors/methods must be preceded by a JavaDoc comment. **Documentation will be considered when grading this assignment!**
- Recall you are required to submit a program that is syntax free and producing some output, even if the output is incorrect. Programs which contain syntax errors or do not run will not be graded.

Extra Credit:

If you complete the requirements as stated above early, you can receive extra credit for creating and playing a card game that uses the deck and the discard pile classes. You should speak with me prior to doing this to receive approval for the game you would like to implement. You will not receive extra credit for an assignment that is submitted late.

Submit:

- Create a .zip file to upload to Blackboard:
    - Right-click the project folder (you should be at the highest-level folder in the project hierarchy), choose Send To | Compressed (zipped) Folder
- Upload the .zip file to the Project #3 assignment in Blackboard by 11:55pm Tuesday, March 24th.

Late Assignments:

Assignments received by 11:55pm on Wednesday, March 25th will receive a maximum of 90 points.
Assignments received by 11:55pm on Thursday, March 26th will receive a maximum of 80 points.
Assignments will not be accepted after this date/time for any reason.