Assignment – card game (linking and sorting)

comp310 Object oriented data structures

# Topic(s)

|  |  |
| --- | --- |
|  | Topic |
| Primary | Linked lists |
| Sorting |
| As needed | Iterators |
| Recursion |

# Objectives

* Understand Linked Lists and sorting concepts and the syntax specific to implementing those concepts in Java.

# instructions

1. **Select a partner to work with (strongly recommended to generate more ideas during design).**
2. **Read ALL instructions** so that you get a big-picture view of the assignment before you start.
3. **Ask** clarification **questions** if any of the instructions don’t make sense *(you are probably not the only one with that question).*
4. **Create your classes in the** order given. *(Analogy: you must have a solid foundation before you can build a house).* Yes, every class listed must be included.

**Note:** In order to allow creative freedom, some of the **details** of selecting instance variables and methods for classes are left to the student. However, instructor will provide **direction and assistance** if they are needed.

## Card

|  |
| --- |
| Card |
| face  suit  next, previous : Card  ? |
| // Methods |

**Write a Card class to represent a playing card**. Include instance variables previous and next of type Card to hold references to the cards before and after the card in a chain of cards *(Role of Card similar to Node).* As usual, provide constructors, accessor and mutator methods for all instance variables, and a toString method for displaying the card *(will help with testing).*

**Test your Card class before you proceed! *(This will usually mean “include a main method in your class, with object creation and method calls.”). For example, your main might look like the following (warning: if you copy and paste the following watch out for the quotes – they’re often different than the quotes in program editors):***

public static void main(String[] args) {

Card[] card = new Card[2]; // array of cards

card[0] = new Card(“Nine”,”Hearts”);

card[1] = new Card(“Nine”,”Diamonds”);

card[0].setNext(card[1]);

card[1].setPrev(card[0]);

System.out.println(card[0].getNext());

System.out.println(card[1].getPrev());

**. . . // best to test all methods**

## pile of cards

|  |
| --- |
| Pile |
| topCard : Card  bottomCard : Card  nCards : int  ? |
| // Methods  Pile of cards |

**Write a Pile class to represent a general pile of cards.** Your cards will be stored as a doubly-linked list (see illustration below), so include instance variables for the top and bottom cards in the pile as well as the number of cards. As usual, provide constructors, accessor and mutator methods for all instance variables, and a toString method

Include methods representing **operations** on piles of cards, such as

* Shuffling and Sorting
* Removing and adding cards (to the top, bottom, etc,..). Note that when you add and remove the next and previous references (in Card class) must be kept up-to-date

topCard

previous

next

bottomCard

* Searching
* Splitting
* Grouping by face or suit.
* Other?

Other methods may be added later if you determine that you need them.

**Test your Pile class methods before you proceed.**

## deck and hand

**Extend classes** Deck and Hand **from** Pile and add any instance variables and behavior (methods) specific to each. For example, Deck would have a deal( ) method. As usual, provide constructors, accessor and mutator methods for all instance variables, and a toString method. Once you write a card game class you may determine that these classes need additional instance variables and methods.

**Test your Deck and Hand classes before you proceed!**

## player

**Write a Player class.** Include instance variables related to players, such as name, hand (instance of your Hand class), and score. Include methods related to players. These may also be added later as you write your Game class.

**Test your Player class before you proceed.**

## game

**Determine a multiple-player card came that you’d like to implement. It MUST BE complex enough to require objects from your Player, Card, Deck and Hand classes and operations provided in your classes, such as shuffling, sorting, searching, dealing, determine win,…**

**Write a Game class.** Include instance variables related to your type of game, such as an array of type Player *(your Player class above) and a* Deck of cards *(your Deck class above).* Depending on the game that you are implementing, the methods that you have might include initialize() to set up for a new game, and play() to run the game. You have creative freedom to develop additional classes if they would make your implementation cleaner and/or more elegant.

**Have fun playing your card game! ☺**

# Good programming practiceS

**Classes**

* Make instance variables private
* Make methods public and include *(almost always)* 
  + Constructors
  + Accessor(s) and Mutator(s)
  + toString
  + main *method to test before you proceed. Note: Put very little code here – try to add as much class-related behavior into class methods as possible.*

# Analysis and Summary

Add a journal entry that answers the following questions **(Note: Each student must write their own summary, even if you worked in pairs on the problems).**

1. Was this the best way to structure classes for a card game? Please explain.
2. Describe how you approached and solved the problem.
3. If you worked in pairs:
   1. How did you “divide up” the work so that each student still met the objectives for the assignment (i.e., learned, understood and applied the concepts).
   2. What was your contribution?
   3. How did you coordinate code changes/testing?
   4. Other observations about working with a partner?
4. Where did you have trouble? How did you move forward? What topics still confuse you?
5. What did you learn from this assignment? *(Please be specific)*
6. Any recommendations for improving this assignment in the future?

# Submitting your work

1. Make sure that your **name(s)** are in **all** of your project files.
2. Make a compressed file (.zip, .rar, etc.,) for your project and include **all files** *(both summaries if you worked in pairs)*
3. In **Blackboard**
   1. **Attach** your compressed file
   2. **Submit**