University of Illinois @ Urbana-Champaign

CI 489: Data Structures for CS Teachers

Application 1: Card Game

Objectives and Overview

Application assignments are intended to be an opportunity to take a step back use some of the data structures you have been using. It is often easy to get so bogged down in the details of creating and implementing data structures and not have the opportunity to use them in performing useful tasks. In this exercise you will be using a linked list similar to the implementation of the one you implemented last week to implement a simple card games.
This game, which is described in greater detail in later sections allows the user to, on each turn, select a card from the deck and guess if that card is higher or lower than their previous card. If their guess is correct they get a point and are allowed to continue. If they are wrong, the game terminates and their score is the number of previously correct guesses.
You are given a greater deal of autonomy in how you implement this assignment in that, as long as the shuffle method shuffles the deck, and the game generally behaves as described you can consider this correct. Do be careful though, this higher degree of autonomy means that you will need to be more strategic in how you approach testing your code. **It is highly suggested that you read over this document multiple times in order to determine how the game is to be played and to map out an approach you might take to implementing it in Java.**

Structures and Specifications

For this assignment you are provided the `LinkedList.java` and `Card.java` classes and those should remain unmodified. The only class you will need to implement is the one in `CardGame.java`. That being said, understanding how the former two classes are implemented is fundamental to this assignment and a useful exercise in and of itself. Do take some time and care to read over the brief descriptions below as well as their source code prior to moving onto the rest of the assignment.

LinkedList.java

The class provided in this file is a working implementation of the class you implemented in the linked list implementation assignment. The following are the methods included in that class you might find useful and what they do:

1. `public ListNode<E> getNodeAtPosition(int pos)`: Gets and returns a node at a given index but does not remove it.
2. `public void addToFront(E e)`: Adds a new node with some data (e) to the front of the list.
3. `public void addToEnd(E data)`: Adds a new node with some data (e) to the end of the list.
4. `public void addNodeAtPos(int pos, E data)`: Adds a new node with some data (e) at an index (pos) in the list.
5. `public ListNode<E> removeFromFront()`: Removes and returns the node at the front of the list.
6. `public ListNode<E> removeFromEnd()`: Removes and returns the node from the end of the list.
7. `public ListNode<E> removeNodeAtPos(int pos)`: Removes and returns a node from a given index.

Card.java

As you will note, the card class is provided for you and has the following attributes:

1. `public static final List<String> SUITES`: This is a static attribute which is a list containing all of the possible suites available.
2. `public static final List<String> CARD_VALUES`: This is a static attribute which is a list containing all of the values of cards avaliable.
3. `private final String cardSuite`: This is a *nonstatic* attribute containing the suite of a specific instance of a card.
4. `private final String cardValue`: This is a *nonstatic* attribute containing the value of a specific instance of a card.

Beyond these attributes, we have the following methods available:

1. `Card(String cardSuite, String cardValue)`: This is the card constructor. it takes a suite and a value and initializes the respective class attributes with those that are passed in as parameters.
2. `public String getCardSuite()` and `public String getCardValue()`: These are the getter methods. Recall that since the attributes are private we must manually control access to them through getters and setters. Here, once a card is created we don't want to change the attributes of the card. As such, we only define getters.
3. `public String toString()`: This prints out a string representation of the card that contains the value and suite of the card.

CardGame.java

This will be the only file you will modify for the assignment. In doing so you will implement the following methods.

1. `public static void shuffle(List<Card> deck)`: This method takes a single deck of cards as a parameter and should shuffle the deck. The method by which you shuffle the deck is left up to you. Details on how the shuffle method are present the shuffle section below.
2. `public static guessHigherOrLower()`: This is our main game method which will facilitate the playing of the game. It is also described in detail in the section below
3. `public static void main(String[] args)`: The main method will be responsible for the following things: 1) Initializing our deck to have all cards/suites and performing an initial shuffle, 2) asking the user if they want to play the game in a loop, and 3) breaking out of the loop if they don't or calling the game method if they do.

shuffle

The shuffle method should randomly reorganize the cards in some manner. The `Random` class will be useful for this method. In particular the `nextInt(x)` method, where x is the upper bound on the positive, random integer this method call will provide. You may want to consult the Java docs to see this class, it's methods, and how they behave.

The manner in which shuffling occurs is left up to you and will be deemed as successful as long as, with each call, the cards in the deck are randomly reorganized. The following is one method by which you might accomplish this:

1. Get a random index from the list (i.e., `rand.nextInt(deck.size())`).
2. Remove the node at that index and get the card stored in the `data` attribute.
3. Add that card back at a new random index.
4. Repeat the above steps some number of times that is sufficient to really mess up the deck.

To test this method it might be useful to print out the contents of the list before and after shuffling in order to ensure that it is working correctly. You might also use the debugger to step through the code and investigate the contents of the list before and after shuffling.

guessHigherOrLower

This method controls the game. You are given the following in the starter code:

- A scanner to read input from the user.
- A score variable to keep track of the score.
- A card variable that gets the card we are currently comparing against. It draws a card initially from the end of the list.
- A print statement that prints out the card that we are starting with.

What you are responsible for from there on out is what will become the body of the while loop. The basics flow of a game is as follows:

1. Begin by asking the user what they would like to do with that turn. Would they like to shuffle or get a card?
2. If they select shuffle then...
   (a) Shuffle the deck and then continue to the next turn (i.e., next iteration).
3. If they say get a card...
   (a) Ask them if they want to get from the top or the bottom.
   (b) Call either the `removeFromFront()` or `removeFromEnd()` methods with respect to their previous response and store the `Card` that gets returned.
   (c) Ask the user if they guess that that card is higher or lower than the previous card.
   (d) Then....
      i. If the user guessed high and the value of the card that was just drawn is higher then tell them the new card was higher, what the new card was, and then increment their score.
      ii. Else if the user guessed low and the value of the card that was just drawn is lower then tell them the new card was higher, what the new card was, and then increment their score.
      iii. If they did not guess correctly then report that they got it wrong, their final score, then break out of the loop.
4. If they type something else...
   (a) Report that the command was invalid

The trickiest part of this process is comparing the values of cards. However, there are a variety of approaches you could take. Here are a few suggestions to get your started:

1. You could get the index of each card's value in the `CARD_VALUES` array and compare those indices (e.g, `Card.CARD_VALUES.indexOf(card.getCardValue())`).
2. You could construct your own additional Java method that allows for the comparison of card values.

main

The main method consists of an initial game prompt and a while loop that facilitates the playing of multiple rounds of the game. It should have the following parts:

1. On each iteration of the loop a new deck should be initialized with all card value/card suite pairs. For this, you will want a nested for loop that iterates over all of the `Card.CARD_VALUES` and `Card.SUITES`, creates a new card at each iteration with the pairing for that iteration, and adds that card to the deck.
2. Shuffle the deck after it is finished being filled.
3. Call `guessHigherOrLower()` and pass it the newly created deck of cards.
4. Ask the user if they want to continue after that round of the game has been played. If they type q the loop should break, otherwise it should continue.

Example Output

```
--Guess Higher Or Lower: A Card Game--
This is a simple card game. In a given move you can
choose to shuffle the deck or select a card. If you select a card
it will be selected from the top of the deck and compared to the
previously selected card. If the user guessed that the card is low
and the card has a lower value or if you guessed high and the card
had a higher value then your score will be incremented. If you do not
then the game will end and your final score is reported.

Your first card is: ACE OF HEARTS. Let the game begin!


Enter one of the following commands: shuffle, get card.
Your command: asdf
Invalid command.

Enter one of the following commands: shuffle, get card.
Your command: get card
Select from top or bottom: top
Guess high or low: high
Correct! The card, SIX OF DIAMONDS, was higher!

Enter one of the following commands: shuffle, get card.
Your command: shuffle

Enter one of the following commands: shuffle, get card.
Your command: get card
Select from top or bottom: top
Guess high or low: low
Correct! The card, TWO OF CLUBS, was lower!

Enter one of the following commands: shuffle, get card.
Your command: get card
Select from top or bottom: top
Guess high or low: high
Correct! The card, NINE OF DIAMONDS, was higher!

Enter one of the following commands: shuffle, get card.
Your command: get card
Select from top or bottom: top
Guess high or low: high
Incorrect, the card was EIGHT OF SPADES . Final score was : 3
Press q to quite or any other key to continue:
```