COSC 222: Data Structures

Lab 1 – Lists & Linked Lists (20 marks + 2 bonus marks)

Lists are a very important fundamental data structure. Not only are they one of the most commonly used data structures on their own, they are also used in modified formats to implement many of the other data structures we will see later in this course. This lab will give you practice using lists to organize and store data, as well as creating a list object which functions as a Double Linked List.

Remember to *always* add detailed comments to your code, as this helps your TA to understand your thoughts as they mark your answers and enables them to give better feedback if you need help.

Question 1 (List Basics): Book Lists [5 + 2 marks]

Kiana, Zill, and Bily are friends. They love to read books of different categories. They each have a list of books. Write a Java program that organizes their book lists.

You have been given two starter files for this question: **Books.java** and **BookLists.java**. Only edit the methods in BookLists.java which are marked "`/* YOUR CODE HERE */`", according to the instructions below. Do not change the method parameters or return types.

Part A [1 mark]:

Complete the method `printList()` which prints out the given list in bullet format and with a line of asterisks at the end. For example, Kianas's list should look like this:

```
Kiana's List:
- Beloved, written by Toni Morrison (Category: Classics, Price: 35.05)
- The Water Dancer, written by Ta-Nehisi Coates (Category: Fantasy, Price: 34.0)
- Ninth House, written by Leigh Bardugo (Category: Fantasy, Price: 27.44)
- Olive Again, written by Elizabeth Strout (Category: Fiction, Price: 31.97)
- Carrie, written by Stephen King (Category: Horror, Price: 50.0)
***
```

Part B [1 mark]:

Complete the method `combineLists()` to combine all three of the friends' lists into one master list. Add each book of the three lists to this list (hint: do not add each item individually; there are methods available to merge elements from multiple collections: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html. There should be no more than 5 lines of code within this method).

Part C [1 mark]:

The friends decide they all don't want the book "Little Women". Complete the *void* method `deleteByName()` so that it removes any elements in the list which have the given name. This

method should directly change the contents of `allList`. Check that the output under `No more "Little Women" in the list:` shows all of the original items *except* the `Little Women`.

Part D [1 mark]:
Complete the method `countUnique()` to return the number (int) of unique book names in `allList`. For example, There are a total of 15 entries where the book named "Carrie" has 3 separate entries and "Olive Again" has two separate entries. You need to find out the number of unique books on the list.

Part E [1 mark]:
Complete the method `findBookByAuthor()` to return a new ArrayList containing all of the books from the list which match the given author name. Note that this should not change the contents contained in `allList` and a book name can be repeated more than once here.

```
Books by Elizabeth Strout:
- Olive Again, written by Elizabeth Strout (Category: Fiction, Price: 31.97)
- Olive Again, written by Elizabeth Strout (Category: Fiction, Price: 31.97)
- Amy and Isabelle, written by Elizabeth Strout (Category: Fiction, Price: 21.2)
***
```

Part F [2 marks (BONUS)]:
Complete the method `combineDuplicates()` to combine all books that are listed more than once to be listed just one time. If you run the program with your code on it, it should show the following output:

```
BONUS - Combined duplicates:
- Beloved, written by Toni Morrison (Category: Classics, Price: 35.05)
- The Water Dancer, written by Ta-Nehisi Coates (Category: Fantasy, Price: 34.0)
- Ninth House, written by Leigh Bardugo (Category: Fantasy, Price: 27.44)
- Olive Again, written by Elizabeth Strout (Category: Fiction, Price: 31.97)
- Carrie, written by Stephen King (Category: Horror, Price: 50.0)
- The Savior, written by J.R. Ward (Category: Romance, Price: 14.89)
- The Help, written by Kathryn Stockett (Category: Fiction, Price: 23.6)
- Circe, written by Madeline Miller (Category: Fantasy, Price: 23.51)
- Amy and Isabelle, written by Elizabeth Strout (Category: Fiction, Price: 21.2)
```

Submit only **BookLists.java** (not **Books.java**) with the updated methods.

Question 2 (ArrayList): Card Shuffle [6 marks]
Write a program to create and shuffle a deck of cards. Begin with the file **CardShuffle.java** and complete the following three small methods: `makeDeck()`, `shuffle()`, and `deal()`.

Part A [2 marks]:

Complete the method `makeDeck()` which should return a new ArrayList (`deck`) containing all the cards. Note: Use the final String arrays `SUITS` and `RANKS` to make the deck of cards. If you use them properly, you will make a deck of cards (e.g. `Ace of Spades, Ace of Hearts, Ace of Clubs, Ace of Diamonds, 2 of Spades, 2 of Hearts, etc.`).

Part B [2 marks]:

Complete the method `shuffle()`, which will rearrange the elements of `deck` into a random order. There is a method called `Collections.shuffle()` which will shuffle an array list. However, you must use the following technique: Choose a random card position from `0` to `deck.size()-1`. Delete the card in that position and put it in position `0` instead. From now on, leave that card alone. Choose a random card position from `1` to `deck.size()-1`. Delete the card in that position. Put it in position `1`. And so on for positions `2..deck.size()-2`. (There's no need to select a random card to go into position `deck.size()-1` because there's only one card left at that point anyway.)

Part C [2 marks]:

Complete the method `deal()`, which will deal the indicated number of hands, with the indicated number of cards in each hand, from the top of the deck. The cards that are dealt should be removed from the deck. It should return an *array* of `ArrayLists`, where each `ArrayList` contains the cards in one hand. It should deal in the normal way, with consecutive cards going into different hands, in a circular manner, as shown in the example below.

Note that `numHands * numCards` should always be less than or equal to sixteen. When the file is run, it should produce random output like the following.

```
The new deck is [Ace of Spades, Ace of Hearts, Ace of Clubs, Ace of Diamonds, 2 of
Spades, 2 of Hearts, 2 of Clubs, 2 of Diamonds, 3 of Spades, 3 of Hearts, 3 of Clubs,
3 of Diamonds, 4 of Spades, 4 of Hearts, 4 of Clubs, 4 of Diamonds]
The shuffled deck is [2 of Clubs, 4 of Clubs, 2 of Hearts, 3 of Spades, 3 of Diamonds,
4 of Diamonds, 3 of Hearts, 2 of Spades, 4 of Spades, 4 of Hearts, Ace of Diamonds,
2 of Diamonds, Ace of Clubs, Ace of Spades, 3 of Clubs, Ace of Hearts]
How many hands should be dealt? 3
How many cards in each hand? 5
The hands are:
Hand 1: [2 of Clubs, 3 of Spades, 3 of Hearts, 4 of Hearts, Ace of Clubs]
Hand 2: [4 of Clubs, 3 of Diamonds, 2 of Spades, Ace of Diamonds, Ace of Spades]
Hand 3: [2 of Hearts, 4 of Diamonds, 4 of Spades, 2 of Diamonds, 3 of Clubs]
The remaining deck: [Ace of Hearts]
```

Submit the **CardShuffle.java** file with all three updated methods.

Question 3 (LinkedList): Delivery Route Double-Linked List [9 marks]

For this question, you will create a double-linked list to represent a delivery driver's route. Write a program that implements a double-linked list to store and organize the route. Note that you will find a similar example in the class lecture.

Part A [1 mark]:

Create a class (**Stop.java**) which models a stop on the driver's route. This class should have:

- Four private instance variables: `address` (String), `orderNumber` (int), `next` (Stop), and `previous` (Stop).
- A constructor which will initialize all four of these variables, however it will take `address` and `orderNumber` as parameters and `next` and `previous` should be `null`. These references should be changed when they are added to the double linked list.
- All of the possible getter/setter methods (e.g., `getAddress`, `setAddress`, `getOrderNumber`, `setOrderNumber`, `getPrevious`, `setPrevious`, `getNext`, `setNext`).
- A `toString()` method which returns a String showing the address and order number in the format "`[address (#order)]`" (i.e. "`[123 Sesame St. (#456)]`")

Part B [1 mark]:

Create a class (**Route.java**) which models the list. This class will be similar to the `java.util.LinkedList` class in that you will create a list of Stops, each of which contain a reference to the next. However, this class will be a double linked list, which means every Stop will also have a link to the previous Stop. This class should have:

- Two private instance variables (`start` and `end`) containing a reference to the first and last objects in the list (in this case, Stop objects) or if the list is empty, both referring to `null`
- A constructor to create an empty list
- A `toString()` method which returns a String showing the list in order (see format below). Return "`No stops on route`" if the route is empty.

Also add the following methods to the **Route.java** class.

Part C [2 marks]:

`void addStart(String address, int orderNumber)`: Create a Stop with the given address and order number and add it to the beginning of the list.

Part D [2 marks] :

`void addEnd(String address, int orderNumber)`: Create a Stop with the given address and order number and add it to the end of the list. You will get zero marks for this question if you traverse the list to find the end element.

Part E [3 marks]:

`void insert(String address, int orderNumber)`: Create a Stop with the given address and order number, and insert it *in the correct location in the list* so that the list remains sorted by `address` in alphabetical order. Note that as the addresses all start with a number, they will be sorted by increasing value of the first digit. You should use the String method `str1.compareToIgnoreCase(str2)` to determine the correct position of the incoming element. Note that this command returns an int: 0 means the two strings are equal, a negative

value means `str1` is less than `str2` (i.e. `str1` comes alphabetically before `str2`), and a positive value means `str1` is greater than `str2` (i.e. `str1` comes alphabetically after `str2`).

Test your methods by running the provided file **RouteTest.java**. If the above methods have been successfully completed, you should see the following output. (Note that if any methods were not named the same as mentioned above, you may receive errors.)

Sample Output:

```
addEnd:
[123 Sesame St (#100)]
[221B Baker St (#101)]
[124 Conch St (#102)]
[322 Maple St (#103)]
[485 Maple Dr (#104)]
[698 Candlewood Ln (#105)]
[1640 Riverside Dr (#106)]
[84 Beacon St (#107)]
[320 Fowler St (#108)]
[711 Maple St (#109)]
[4 Privet Dr (#110)]
[129 W 81 St (#111)]
[1407 Graymalkin Ln (#112)]

addStart:
[1407 Graymalkin Ln (#112)]
[129 W 81 St (#111)]
[4 Privet Dr (#110)]
[711 Maple St (#109)]
[320 Fowler St (#108)]
[84 Beacon St (#107)]
[1640 Riverside Dr (#106)]
[698 Candlewood Ln (#105)]
[485 Maple Dr (#104)]
[322 Maple St (#103)]
[124 Conch St (#102)]
[221B Baker St (#101)]
[123 Sesame St (#100)]

insert:
[123 Sesame St (#100)]
[124 Conch St (#102)]
[129 W 81 St (#111)]
[1407 Graymalkin Ln (#112)]
[1640 Riverside Dr (#106)]
[221B Baker St (#101)]
[320 Fowler St (#108)]
[322 Maple St (#103)]
[4 Privet Dr (#110)]
[485 Maple Dr (#104)]
[698 Candlewood Ln (#105)]
[711 Maple St (#109)]
[84 Beacon St (#107)]
```

Submit your Java files **Stop.java** and **Route.java**.

Submission Instructions:

- Create a folder called "Lab1_<*student_ID* >" where <*student_ID* > is your student number/ID (e.g. **Lab1_12345678**). Only include the mentioned java files in the folder. **DO NOT** include any other files (e.g., .class, .java~ or any other files)
    - For Question 1, include your **BookLists.java** file.
    - For Question 2, include your **CardShuffle.java** file.
    - For Question 3, include your **Stop.java** and **Route.java** files.
- Make a **zip file** of the folder and upload it to Canvas.
- To be eligible to earn full marks, your Java programs **must compile and run** upon download, without requiring any modifications.
- These labs are your chance to learn the material for the exams. **Code your labs independently.**