COMP202 SPRING 2016

PROGRAMMING PROJECT 4

Due Date: **Wednesday, April 27 at 23:59**

Submission Location: F Drive

Corresponding Teaching Assistant: **Devris Isler**

Corresponding TA's Office Hours: Monday, 11:00-12:00

Extra Office Hours: ***Monday, April 25, 10:00-11:00 / Tuesday, April 26, 11:00-13:00 / Wednesday, April 27, 12:00-13:30***

1 DESCRIPTION

In the Project 4, you will get familiar with implementation of sorting algorithms in Java. Your project **must** get command from console for calling the methods that asked to implement in this project. You are required to apply two different sorting algorithms which are ***Merge-Sort*** and ***Quicksort*** in order to sort a deck of cards in a specific manner. You are asked to implement Merge Sort and Quicksort. Suppose there is a deck of unsorted/mixed cards. Before specifying the rules of sorting the cards, realize the card properties below;

1. A card has two enumerated types which are ***Suit*** and ***Rank.***
2. The suit is defined as: $Suit = \{"S","H","D","C"\}$ where ***S*** represents ***Spade***, ***H*** represents ***Heart***, ***D*** represents ***Diamond***, and ***C*** represents ***Club***.
3. Rank is a number is in the range of $[1,13]$.

You are given a deck of cards and asked to sort the card by the following rules;

1. The suit of the card matters, and the priority of the suit specified as ;

$$\boldsymbol{Priority_{Of_{Spade}}} > \boldsymbol{Priority_{Of_{Heart}}} > \boldsymbol{Priority_{Of_{Club}}} > \boldsymbol{Priority_{Of_{Diamond}}}$$

2. The cards in the deck should be sorted based on their suits and ranks. After applying the sorting algorithm to the deck, cards having same suit should be grouped (the priority of the suit is important, and the cards should be in an order such that the cards having the highest priority should be in the head of the deck and lowest one at the back), and each suit groups' cards shall be sorted based on their ranks in ascending order. Before starting the implementation, take a moment and understand the procedure of sorting the deck, and scratch on a paper to visualize the merge sort and quicksort on the deck.

For the implementation, you are required to have following essential classes (you cannot add more classes);

1. ***Card*** class that basically have the properties of a card. Suit and rank are private variables. You can think the card as a node.
2. ***Deck*** class consists of a number of cards. The deck is represented by a Linked List (preferable Circular Linked List, and you are not allowed to use arrays, LinkedList). Each node in the linked list represents a card.
   Deck class should have following methods (you can add more methods in order to satisfy the requirements for this project; however, following methods shall be implemented)
   Methods: $\boldsymbol{createDeck}(), \boldsymbol{mergeSort}(), \boldsymbol{quicksort}(), \boldsymbol{printDeck}()$ (you are free to decide the parameters of the methods, and you can add helper methods as you like). Description of the methods and used commands for calling the methods from console are shown in Table 1.

Methods	*Description*	*Commands*
$createDeck()$	Creates a deck based on the given a file (that is $input.txt$ file for sample but the name of the file may differ even though the file's extension is still .txt) which consist of cards. The method reads the given file, and inserts the records into the deck in given order in the file.	createDeck*filename.txt
$mergeSort()$	Sorts the deck, by using **Merge Sort** algorithm, in given specific sorting manner	mergeSort
$quickSort()$	Sorts the deck, by using **Quicksort** algorithm, in a given specific sorting manner	quickSort
$printDeck()$	Visualizes the deck created by using $input.txt$ file.	printDeck

Table 1: Description of the methods and command used for calling the method from the console.

Note: Commands are not case sensitive. For example, mergeSort and Mergesort are the same commands.

3. ***LinkedList*** class is a collection of nodes where each node represents a card. (Since you implemented linked list in the Project 2, it will be easier for you. Just **be aware** that each card has two properties which are very important for sorting.)
4. ***Main*** class where you test your program.

2 INPUT & OUTPUT

2.1 INPUT FILE FORMAT

Your project should read a file (when the user types $createDeck$ command on the console, and the $input.txt$ file is given in the project folder as a sample) in order to create the deck. Each line in $input.txt$ file, which represents a card, given in a format as $< \boldsymbol{Suit} >< \boldsymbol{Single\_Space} >< \boldsymbol{Rank} >$ (see Figure 1), and also realize that there is **no** space between the lines.

```
H 1
S 5
S 3
D 9
D 3
H 7
C 10
C 6
```

Figure 1: A sample of the input.txt

2.2 CREATING THE DECK

After the $createDeck()$ method is called by the user typing $createDeck * filename.txt$ command on the console, the output of the $createDeck()$ method shall be as in the Figure 2.

```
The file has been read without any error and the deck has been successfully created.
```

Figure 2: The expected output after calling the createDeck() method

2.3 VISUALIZATION OF THE DECK

When $printDeck()$ method is called (the method will be called after the $createDeck()$ method called), the output of the method should be displayed on the console as in the Figure 3. The format of the method's output is as in the following;

$$Suit_{Of_{Card_1}}|Rank_{Of_{Card_1}} -> Suit_{Of_{Card_2}}|Rank_{Of_{Card_2}} -> \cdots -> Suit_{Of_{TheLastCard}}|Rank_{Of_{TheLastCard}}$$

$$H|1 -> S|5 -> S|3 -> D|9 -> D|3 -> H|7 -> C|10 -> C|6$$

Figure 3: A sample of the deck created based on input.txt

2.4 MERGE SORT

For sorting the deck created based on $input.txt$ by using merge-sort algorithm, you should show each iteration/step in the merge-sort when a $mergeSort()$ method is called. The expected output on the console is shown in Figure 4. (Add a statement/joke/wise saying that you like or make up in the beginning of the output. It can be about Data Structures and Algorithms.)

H|1 is the leftmost card when the division is finished. In other words, it is enough to display only the first card when division is accomplished.

```
What a Beautiful Merging Project!

Initial Unsorted Deck:

H|1-> S|5-> S|3-> D|9-> D|3-> H|7-> C|10-> C|6

Merge Sort Started:

H|1
S|5-> H|1
S|5-> H|1    S|3
S|5-> H|1    S|3-> D|9
S|3-> S|5-> H|1-> D|9
S|3-> S|5-> H|1-> D|9    D|3
S|3-> S|5-> H|1-> D|9    H|7-> D|3    C|10
S|3-> S|5-> H|1-> D|9    H|7-> D|3    C|6-> C|10
S|3-> S|5-> H|1-> D|9    H|7-> C|6-> C|10-> D|3
S|3-> S|5-> H|1-> H|7-> C|6-> C|10->D|3-> D|9

Merge Sort is finished..
```

The first partition (**H|1 ->S|5**) is sorted. **S|3** will be displayed and then it will merge with **D|9**. After the merging, **S|3->D|9** is displayed. However, **S|3->D|9** and **S|5->H|1** are not yet merged.

This is the first merging between two cards which are **H|1** and **S|5**. **S|5** should be replaced before the **H|1**, so **H|1** and **S|5** should be swapped.

S|3->D|9 and **S|5->H|1** are merged, and the result of merging is **S|3-> S|5-> H1-> D|9**. In addition to that, **D|3** which is the first card of the second half of the deck in partitioning.

Figure 4: The expected output when the mergeSort() method is called

In merge sort, the deck is partitioned until only one card is left. Then, the cards are merged from left to right (from head of the linked list to end of the linked list). First card (**H|1**) in the head is displayed, then the result of merging the first card with the second card (**S|5**) is displayed. Later on, the third card (**S|3**) is displayed next to the first two merged cards (which are **S|5-> H|1** in the Figure 4), then merging result of fourth card (**D|9**) and the third card (**S|3**) is displayed next to them (**S|5-> H|1 S|3->D|9**). After that, these merged cards are merged to each other (**S|3-> S|5-> H|1-> D|9**) and the fifth card (**D|3**) is displayed next to them (**S|3-> S|5-> H|1-> D|9 D|3**). The procedure continues like this until the all cards are merged meaning that the deck is sorted.

2.5 QUICKSORT

For sorting the deck created based on $input.txt$ file by using quicksort algorithm, you should show each partitioning and pivot, which is chosen the last card of the deck, in the quicksort when a $quickSort()$ method is called. Please realize that first left side of the partitioning is shown and then the right side of the partitioning is shown. This is for the sake of simplicity of the implementation. The expected output of the $quickSort()$ method on the console is shown in the Figure 5. (Add a statement/joke/wise saying that you like or make up in the beginning of the output. It can be about Data Structures and Algorithms.)

```
"If the automobile had followed the same development cycle as the computer,
a Rolls-Royce would today cost $100,
get a million miles per gallon,
and explode once a year,
killing everyone inside." (Robert X. Cringely)

Initial Unsorted Deck:

H|1-> S|5-> S|3-> D|9-> D|3-> H|7-> C|10-> C|6

QuickSort Started:

Pivot C|6:
H|1-> S|5-> S|3-> H|7-> C|6-> D|9-> C|10-> D|3

Pivot H|7:
H|1-> S|5-> S|3-> H|7-> C|6-> D|9-> C|10-> D|3

Pivot S|3:
S|3-> S|5-> H|1-> H|7-> C|6-> D|9-> C|10-> D|3

Pivot H|1:
S|3-> S|5-> H|1-> H|7-> C|6-> D|9-> C|10-> D|3

Pivot D|3:
S|3-> S|5-> H|1-> H|7-> C|6-> C|10-> D|3-> D|9


QuickSort is finished
```

C|6 is chosen as the pivot, and the deck is partitioned based on the pivot (**C|6**). And the result of the deck is displayed

H|7 is chosen as the pivot, and the left partition of the deck (which is **H|1->S|5-> S|3->H|7**) is partitioned based on the pivot. The result of the deck is displayed (**H|1-> S|5->S|3-> H|7-> C|6->D|9->C|10->D|3**). After the left side of the deck, created by partitioning based on the pivot **C|6**, is sorted, then same displaying logic is applied to the right of the deck (which is **D|9-> C|10-> D|3**).

Figure 5: The expected output when the quickSort() method is called

3 IMPORTANT!!

- Do the project **individually**. Otherwise, you risk being penalized of cheating.
- Your at-home submissions will **be checked** against plagiarism
- In two cases you will **NOT** be allowed to attend the in-lab part and your grades will be zero:
  - If you submit your projects (at-home part) **after** the due time.
  - If you **plagiarize** in at-home part

- You **must** first understand the how the *quicksort* and *merge sort* algorithms work. You are **strongly** recommended to write the pseudocode of the solution that you define to solve the given problem in this project.
- You **are not** allowed to use any java library that substitutes the parts you are asked to implement and solve.
- You can come to corresponding TA's office and ask if you are having trouble with debugging rather than sending your code via email and ask what is wrong with your code.
- Your code **must** be able to read $filename.txt$ located in your project's root directory
- Please **do not zip your files** and follow the submission rules.
- Upload it through Novell Login to the F Drive/COURSES/UGRADS/COMP202/PROJECT/YOUR_NAME
- Please consider the following format for your projects submission:

  - $Project4$ should be only the name for your folder.

- Do not upload two versions of your project under the same name, instead keep a counter and add it to the end of your file:
  - First version: Project4
    Second version: Project4_V2
    ith version: Project4_Vi

***Only the latest version of your submissions will be checked.*

***Except the emergency situation, please try to upload JUST one and the final version*

Good Luck!
Have fun implementing the project ☺