CS 2336 – PROJECT 3 – GotG2 Ticket Reservation System (Maintenance Project 2)

Pseudocode Due: 2/26 by 11:59 PM

Project Due: 3/22 by 11:59 PM

Submission and Grading:

- All project deliverables are to be submitted in eLearning.
- The pseudocode should be submitted as a Word or PDF document and is not accepted late.
- Zip all of the source files into a single zip file for submission.
- Projects submitted after the due date are subject to the late penalties described in the syllabus.
- Programs must compile and run with JDK 8.
- Each submitted program will be graded with the rubric provided in eLearning as well as a set of test cases. These test cases will be posted in eLearning after the due date. Each student is responsible for developing sample test cases to ensure the program works as expected.
- **Type your name and netID in the comments at the top of all files submitted.**
- **Your main class file must be named Main.java**
- **The linked list classes must be in a package named LinkList**
- **Zip the contents of the src subdirectory into a single zip file (not .rar, not .tar). *Do not zip the src directory, only its contents.***

Objectives: Create and manipulate a linked list in Java. Utilize inheritance and classes to create a basic data structure.

Problem: In preparation for the release of Guardians of the Galaxy 2, you have been hired by the owner of a small movie theater to develop the backend for an online ticket reservation system. Patrons will be able to reserve seats in one of three auditoriums. Once the patron has selected an auditorium, the program should display the current seating arrangement and allow the patron to select seats. A report should be generated at the end of the program to specify for each individual auditorium and overall for all auditoriums how many seats were sold/unsold and how much money was earned.

Changes from project 1 are displayed in blue.

Classes

- Base Node (Abstract)
  - Members
    - Row (integer)
    - Seat (integer)
  - Methods
    - Overloaded constructor
    - Mutators
    - Accessors
- DoubleLinkNode (derived)
  - Members
    - Next (DoubleLinkNode pointer)

        - Prev (DoubleLinkNode pointer)
    - Methods
        - Overloaded constructor
        - Mutators
        - Accessors
- Linked List (separate class, same package)
    - Members
        - Head (DoubleLinkNode pointer)
        - Tail (DoubleLinkNode pointer)
    - Methods
        - Overloaded constructor
            - Takes DoubleLinkNode parameter for head
            - Assign tail to proper node
        - Mutators
        - Accessors
        - Other methods that are necessary to interact with a linked list
            - Remember methods should be generic enough to be used on a linked list regardless of the problem.
    - **All functions that modify the linked list must be of your own design rather than using built-in functions of Java libraries**

Details

- **To avoid potential errors when grading, do not create multiple Scanner objects for System.in. If the Scanner object is needed in multiple functions, please pass the Scanner object into the function.**
- The seating arrangement for each auditorium will be stored in separate files. These files will be named *A1.txt, A2.txt* and *A3.txt* for auditorium 1, 2 and 3 respectively.
- Each line in the file will represent a row in the auditorium. The number of rows in each auditorium is unknown to you.
- The number of seats in each row of a **single** auditorium will be the same. For example, if the first line of the file has 15 seats, then every subsequent row in the theater will also have 15 seats. This does not mean that each auditorium has the same number of seats in each row. One auditorium may have 15 seats per row and another may have 20 seats.
- Empty seats are represented by a pound sign (#).
- Reserved seats are represented by a period (.).
- Each file should be read into memory and stored in a linked list
    - For each auditorium, you will have 2 linked lists – one for reserved seats and one for available seats
    - When available seats are reserved, the nodes from the available list should be moved into the reserved list
    - Both lists should be sorted by row and seat number – all row 1 seats grouped together in order by seat, then row 2, etc.
- If the user is reserving seats, ask the user for the number of tickets, the row and seat number of the first seat. If that seat and all seats to the right of it (up to the number of tickets desired) are available, move

the nodes from the available linked list into the reserved linked list. After seats are reserved, please display a confirmation of this on the screen.
    - If the desired seats are not available, offer the user the best available seats that meet their criteria **in the entire auditorium**. The best available seats are the seats closest to the middle of the auditorium.
        - Think of the distance between 2 points
        - For simplicity, use the distance between the first seat of the section and the center.
        - In the event of a tie for distance, the row closest to the middle of the auditorium should be selected.
        - You may develop this piece any way that you see fit. Please try to be as efficient as possible with the memory.
    - State to the user what the best available seats are and then ask if they would like those seats.
    - If the user declines the best available seats, return the user to the main menu.
    - If the user accepts the best available seats, reserve them and display a conformation to the screen.
- Tickets can only be reserved the day of the screening and all screenings are at 7 PM that night. There is no need to worry about multiple screenings or reserving seats on a specific day.
- All tickets are $7 regardless of patron age or auditorium.
- At the end of the program, overwrite the original files with the new information. Use a recursive function to write the data back to the file. **NOTE IN THE COMMENTS AT THE TOP OF YOUR MAIN SOURCE FILE THE LINE NUMBERS OF THE RECURSIVE FUNCTION CALL AND THE FUNCTION DEFINITION.**

User Interface and Input: Present a user-friendly menu system for the user to reserve seats.

```
1. Reserve Seats
2. View Auditorium
3. Exit
```

Although in reality the user would likely only make one purchase, for testing purposes, assume the user will repeat the ticket buying process until they decide to quit.

After the user has selected a non-exit option, present the user with a menu to select the auditorium.

```
1. Auditorium 1
2. Auditorium 2
3. Auditorium 3
```

Once the auditorium has been selected, display the current seating availability for that auditorium. An example seating chart is provided below for an auditorium with 5 rows and 20 seats per row.

```
  12345678901234567890
1 ...##..#####........
2 ########....####..##
3 .........##.........
4 #.#.#.#.#.#.#.#.#.#.
5 ########.#####.#####
```

The seats are numbered sequentially from left to right and only the ones digit is displayed above each column to make it easier to display the chart. It is understood that the second set of digits from 1-0 are for the numbers 11-20 in the above example.

After the user has selected the auditorium and the seating chart has been displayed, prompt the user for the following information in the order below:

- Row number
- Starting seat number
- Number of tickets

Assume that the user wants to reserve sequential seats to the right of the first seat entered.

If the desired seats are not available, offer the user the best available seats that meet their criteria in the **entire auditorium** (see details above). Prompt the user to enter a **Y** to reserve the best available or **N** to refuse the best available. Once the selection has been processed, return to the main menu.

When prompting the user for input, expect anything. Do not assume any input will be valid. If you ask for a number, the user could put in a floating point number, symbols or maybe even the Gettysburg Address (or at least a sentence or two). Make sure that your program handles proper validation. If invalid input is entered, loop until valid input is received. Consider using exception handling for that task (but it is not required)

Output: At the end of the program, write the current status of each auditorium to the proper file. Also display a formatted report to the console. The report should consist of 4 columns:

- Column 1 – labels
    - Auditorium 1
    - Auditorium 2
    - Auditorium 3
    - Total
- Column 2 - number of seats reserved for each label
- Column 3 - the number of open seats for each label
- Column 4 - the total of the ticket sales for each label