Project 3

Ticket Reservation System

KEY ITEMS: Key items are marked in red. Failure to include or complete key items will incur additional deductions as noted beside the item.

Submission and Grading:

- The pseudocode will be submitted in eLearning as a PDF document and is not accepted late.
- All project source code will be submitted in Zybooks.
- Projects submitted after the due date are subject to the late penalties described in the syllabus.

• Type your name and netID in the comments at the top of all files submitted. (-5 points)

Objectives:

- Create and manipulate a linked list of linked lists in Java.
- Utilize classes to create a basic data structure.

Specification

Problem: The owner of a small theater in an isolated town in Montana has hired you to develop the backend for an online ticket reservation system. to handle the customer demand. Your program will allow attendees to select seats and display the current seating arrangement. At the end of the program, a report will be generated for the owner indicating how many seats were sold/unsold and how much money was earned. Given what you know from the previous version (Project 1), your task is to update the system with a more dynamic solution.

Pseudocode: Your pseudocode should describe the following items

- Auditorium constructor
- Input validation
- How to implement Best available seats
- A list of at least 10 test cases you will check during testing
    - Specific input is not necessary
    - Describe what you are testing

Zybooks Information:

- You will have multiple source files
    - `Main.java`
    - `Auditorium.java`
    - `Node.java`
    - `Seat.java`
- Core implementation has unlimited submissions
    - This will help you make sure the basic actions of your program are working properly in the environment
- Final submission is limited to 10 submissions

Core Implementation:

- Create an auditorium object
- Read the file into the object
- Determine if seats are available
    - Reserve them if available
    - Determine if number of seats requested is possible in a row
- Write the report
- Write the auditorium to a file

Classes

- Seat – Seat.java
    - Members
        - Row (integer)
        - Seat (character)
        - Ticket type (character)
    - Methods
        - Default constructor
        - Overloaded constructor
        - Mutators
        - Accessors
- Node – Node.java
    - **Generic class (-5 otherwise)**
    - Members
        - Next (Node pointer)
        - Down (Node pointer)
        - Previous(Node pointer)
        - Payload (generic)
    - Methods

        - Default constructor
        - Overloaded constructor
        - Mutators
        - Accessors

    {class Node is generic, no references to Seat class }

- Auditorium – Auditorium.java
    - Members
        - First (Node pointer)
            - Acts as head pointer of linked list
    - Methods
        - Constructor
            - Builds auditorium grid and fills in each node with a seat object based on file input
        - Mutator
        - Accessor
    - **generic node is used, Auditorium will need to be generic as well**
        - **No references to Seat class**

Additional methods may be added to any class as long as the methods are specific to the functionality of the class. All methods must be universal, this means that the method could be used (and makes sense to use) in any program that uses the class.

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.**
- **Do not create a package**
- The seating arrangement for each auditorium will be stored in a file.
    - Prompt the user for the filename at the beginning of the program
- Each line in the file will represent a row in the auditorium. The number of rows in the auditorium is unknown to you.
- The number of seats in each row of the auditorium will be the same.
- No row will have more than 26 seats.
- The auditorium will be held in an auditorium object which will use a grid of nodes connected by pointers
  (-20 points if not)
    - Create a grid of connected nodes that replicates the auditorium displayed in the file
    - Each row is a linked list

    - The first node in each linked list will connect to the first node of the linked list below it (through the down pointer)
    - Mark each seat with a row and seat number
    - Mark what type of ticket was sold for that seat
- Empty seats are represented by a period (.).
- Reserved seats are represented by a letter (A, C or S) in the file
    - This will be used to create reports
    - A =adult
    - C = child
    - S = senior
- Reserved seats will be represented by a hashtag (#) on the screen
    - The user does not need to know what type of ticket was sold, just that a seat is reserved.
- There is no need to worry about multiple screenings or reserving seats on a specific day.
- Ticket prices are as follows:
    - Adult - $10
    - Child - $5
    - Senior - $7.50
- Use exception handling to gracefully handle invalid user input (-5 points if not)

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

```
   1. Reserve Seats
   2.  Exit
```

Loop the menu until the user decides to quit. Imagine this is for a ticket kiosk in the lobby of the theater.
If the user wants to reserve seats, 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.

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

The rows are numbered, and the seats are identified in each row by a letter of the alphabet

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

- Row number
- Starting seat letter
- Number of adult tickets
- Number of child tickets
- Number of senior tickets

Assume that the user wants to reserve sequential seats to the right of the first seat entered. Adult tickets will be reserved first, followed by child and then senior. All seats must be open for a reservation to be processed.

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. Use the Euclidean distance formula. Euclidean distance between 2 points (x_1, y_1) and (x_2, y_2) is given by the formula:
$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$
- Use the distance between the center of the selection and the center of the auditorium. Keep the centers as double numbers to get correct distance.
- In the event of a tie for distance, the row closest to the middle of the auditorium should be selected.
- In the event of a tie for closest row, use the row with the smaller number.
- In the event of a tie within a row, use the smaller seat number.

State to the user what the best available seats are and then ask if they would like those seats. Prompt the user to enter a **Y** to reserve the best available or **N** to refuse the best available. 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 confirmation to the screen. Once the selection has been processed, return to the main menu**.** If there are no alternate seats available, display **`no seats available`** to the user instead of a prompt and 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 a sentence. Make sure that the user selection does not go out of bounds of the auditorium. If invalid input is entered, loop until valid input is received.

User Interface Workflow: Please do not add extra prompts since this will cause a mismatch in the input which will either force the program to throw an exception or cause the program to perform an unintended operation.

- `Prompt for filename`
- `Display main menu`
- `Prompt for input`

If user is reserving tickets

- `Prompt for row`
  - Validate – loop until valid
  - Valid row = row number listed in auditorium display
- `Prompt for starting seat`
  - Validate – loop until valid
  - Valid seat = seat letter listed in auditorium display
- `Prompt for number of adult tickets`
  - Validate – loop until valid
  - Valid ticket number = number >= 0
- `Prompt for number of child tickets`
  - Validate – loop until valid
  - Valid ticket number = number >= 0
- `Prompt for number of senior tickets`
  - Validate – loop until valid
  - Valid ticket number = number >= 0

If selected seats available complete the booking. If user selected seats unavailable:

Display best available

- `Prompt user to reserve (Y/N)`

If options Y is selected complete the booking

- Return to main menu

Loop to top of workflow until user selects exit

Output: Display the best available seats in the following format:

```
<row><starting seat> - <row><ending seat>
```

Example: 3D – 3F

At the end of the program, write the current status of the auditorium to `A1.txt`. Remember to write the auditorium reservations using A, C and S to identify the type of ticket sold. Also, display a report to the console using the following format:

```
Total Seats: <value>
Total Tickets: <value>
Adult Tickets: <value>
Child Tickets: <value>
Senior Tickets: <value>
Total Sales: <value>
```

Total amount of money collected for tickets in the whole auditorium. Not just the ones purchased using the program.

All values except total ticket sales will be an integer value. Total ticket sales will be a floating-point value rounded to 2 decimal places and formatted with a dollar sign before the first digit of the number.