Data Structures and Algorithms 1

CA Exercise 1 – Theatre Booking System

The objective of this CA exercise is to create a theatre booking system in Java that makes heavy use of custom-built internal data structures. The system should help the user manage a single theatre by allowing them to:

- Add a new show to the system (e.g. Phantom of the Opera, The Lion King, etc.). The following information should be stored for each show: title, running time (minutes), start date, end date, and ticket prices (for stalls, circle and balcony).
- Add a new performance to the system. A performance is for a show for a given date. The following information should be stored for each performance: date and time (either evening or matinee). A maximum of 2 performances (i.e. both matinee and evening) can run in a single day.
- Add a new customer to the system. A customer is a theatre goer/patron, who buys/books tickets. The following information should be stored for each customer: name, email address, and phone number.
- Add a booking to the system. A booking can be for one or more seats for a specific performance. The seats might be continuous/together, or scattered. The following information should be stored for each booking: customer details, booking id, performance and seat details. The booking facility should not allow a seat to be double-booked for any performance. A graphical theatre map (of some sort) could be used to show seat availability for performances to help with bookings.
    - The system should offer a facility to search for and choose suitable contiguous seats for a booking (i.e. if the customer is trying to book 3 seats in the stalls the system tries to find 3 seats together in the stalls). Note that a booking can still include a variable number of seats scattered throughout the theatre.
- View facilities to view details of all shows, performances, bookings made, total receipts, etc. The theatre map could be used again to show occupancy for performances.
- Cancel/delete facilities. The user should be able to cancel bookings (full or partial), performances, and shows. The user should also be able to delete customers. All cancellations/deletions should be appropriately cascading (e.g. cancelling a performance will also remove all bookings for that performance; deleting a customer will also remove all bookings made by that customer; etc.)
- Reset facility. Clears all bookings, performances and shows.
- Save and load the entire system data to support persistence between executions.
    - This can be done using any suitable file format (e.g. CSV, XML, binary, etc.). There is no need to use any database system beyond this.
- Other appropriate facilities to manage the theatre as you see fit.

The (small) theatre to manage has 94 seats in total arranged as 24 in the Balcony (3 rows of 8), 30 in the Circle (3 rows of 10), and 40 in the Stalls (4 rows of 10). The theatre layout, with seat numbering, is as follows:

	B17	B18	B19	B20	B21	B22	B23	B24		
	B9	B10	B11	B12	B13	B14	B15	B16		Balcony
	B1	B2	B3	B4	B5	B6	B7	B8		
C21	C22	C23	C24	C25	C26	C27	C28	C29	C30	
C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	Circle
C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	
S31	S32	S33	S34	S35	S36	S37	S38	S39	S40	
S21	S22	S23	S24	S25	S26	S27	S28	S29	S30	Stalls
S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	
S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	

This is an arbitrary size and layout but I want you to stick to it exactly, as it poses a couple of small challenges for you to work out regarding bookings and display.

Notes

- This is an individual CA exercise, and students should work by themselves to complete it.
- You will have to demonstrate this CA exercise in the lab sessions and you will be interviewed on various aspects of it. You are expected to be able to answer all questions on any code you are forwarding as your own.
- This CA exercise is worth 35% of your overall module mark.
- You should implement a suitable JavaFX graphical user interface to interact with your system.
    - The GUI does not have to be particularly fancy, but should nevertheless be functional.
- You should implement one or more JUnit classes to systematically test your code as you develop it. Exactly what you test for is up to you, but make sure that you demonstrate a test-driven approach to your development.
- It is important to note that you cannot use any existing Java collections or data structures classes (e.g. ArrayList, LinkedList, or any other class that implements the Collection interface or any of its children – if in doubt, ask me!). You also cannot use regular arrays directly (except for seat allocations, as noted below). You essentially have to implement the required data structures and algorithms from scratch and from first principles (in line with the module learning outcomes). This is deliberate.
    - You will have to create numerous custom ADTs such as Show, Performance, Customer, Booking, etc. (your classes/class names may differ).
    - Regular arrays can be used, if desired, for seat allocation purposes. However, beyond that, you should only use custom-built linked lists for storing the primary system data.
    - You should also avoid simply storing data using the JavaFX components themselves. The JavaFX components can of course be used to display data/information as required, but the main data should be stored separate from the JavaFX components.

Indicative Marking Scheme

- Appropriate custom ADTs = 10%
- Create/add facilities (show, performance, booking, customer) = 16%
- Delete/cancel facilities (show, performance, booking (full or partial), customer) = 10%
- Seat search/selection facility (contiguous and/or available seats, etc.) = 12%
- Graphical theatre map (used for bookings, and to show availability/occupancy) = 10%
- View all (shows/performances/bookings/receipts/etc.) = 10%
- Reset facility = 5%
- Save/load facility = 8%
- Appropriate JavaFX user interface = 8%
- JUnit testing (implement a minimum of 5 useful unit tests) = 5%
- General (commenting, style, logical approach, completeness, etc.) = 6%