Ticket Service

A simple ticket service that facilitates the discovery, temporary hold, and final reservation of seats within a high-demand performance.

Design

The current project works with one screen, one movie and one show at a time. However, the project is designed in such a way that it can be extended to accommodate multiple shows, multiple movies and multiple screens.

The project has the following entities. They can also represent tables if/when a Database is used.

Setup Entities for Theatre Admin

1. **Screen** – A domain class which represents a screen (venue) in the theatre. The members of this class are
    a. Screen Id – Autogenerated (acts like a primary key). Starts from 1000.
    b. Screen Name – Another unique identifier to the screen which can be modified by the theatre admin (unlike the screen id).
    c. Capacity – No. of seats
    d. Screen Size – The diagonal measurement of the physical screen in inches. This field is used to differentiate IMAX screens from normal ones.
    e. Status (isActive) - A Boolean variable which denotes whether the screen is active at present or not. New show can be created (i.e. a movie can be associated with this screen) only when the screen is active.
2. **Movie** – A domain class which represents a movie which is showing in the theatre. It has the following fields
    a. Movie Id - Autogenerated (acts like a primary key). Starts from 1000.
    b. Movie Title – The name of the movie
    c. Movie Language – Language in which the movie is played
    d. Status (isActive) - A Boolean variable which denotes whether the movie is active at present or not. New show can be created only when the movie is active.
3. **Show** – A domain class which represents a show (of a movie in a screen at specific time). The member variables are
    a. Show Id - Autogenerated (acts like a primary key). Starts from 1000.
    b. Movie Id – Id of the movie which is being shown.
    c. Screen Id – Id of the screen where the movie is being shown.
    d. Show Time – Time of the show
    e. Status (isActive) - A Boolean variable which denotes whether the show is active at present or not. A customer can book tickets for a show only when it is active.

Transaction Entity

4. **Booking** – A domain class which represents a booking for a show. It has the following fields
    a. Seats Info – The specific seats which are reserved for the booking and related info
    b. Customer Email – Unique Identifier for a customer
    c. Confirmation Code – A unique string which confirms and identifies the booking

The other transactional entities which are present in the project are the following. These cannot be represented as tables.

5. **Seats** – An entity which maintains and manages the seats in a show. It contains the following
    a. Show Id – The id of the show for which the seats are managed
    b. Seat Availability – The current number of vacant seats which are neither held not reserved by a customer
    c. Seat Array – Array representation of the seating
    d. Row Availability – No. of seats which are vacant in each row
6. **SeatHold** – An entity identifying the specific seats held by a customer and related information. Its members are
    a. SeatHold Id – Autogenerated. Starts from 1000.
    b. No. of Seats – Required number of seats to be held for a customer
    c. Hold Start Time – The current time when the object is created (used to discard the hold when it expires)
    d. Customer Email – Unique Identifier for a customer
    e. Seats – Array of seat numbers that are allocated for the hold

The project has the following services

1. **TicketService** – An interface which has the declaration of the three major functionalities of this project (discovery, temporary hold, and final reservation of seats).
2. **BookingService** – Implementation of the TicketService Interface. The public methods of this service are
    a. numSeatsAvailable() – This method returns the current number of vacant seats that are neither held not reserved by a customer. It discards the expired holds before calculating the available seat count.
    b. findAndHoldSeats() – This method finds the best available seats for a customer. It follows the logic, with is described in the Implementation section, to identify the best possible seats. It returns a SeatHold object with all the required information about the hold.
    c. reserveSeats() – This method reserves the held seats and confirms the booking. It makes sure that the hold is not expired before proceeding with the reservation. It returns a confirmation code by which the customer can uniquely identify the booking.

The following classes are used for the initial setup. These are written with respect to the current requirements. They should be modified/replaced if/when the requirement changes.

1. **TheatreSetup** – This is a setup which must be created before starting the booking process for a show. It takes care of creating the Screen, Movie and Show objects. All the constants such as the

no. of rows and columns in a screen, the screen capacity, the screen size, the movie details, etc. are specified in here.

2. **Theatre** – This is the main class of the project. It uses a separate Thread class for executing each booking request. The main method creates a TheatreSetup object before creating threads for ticket booking.

Implementation

The best seats are the ones farthest from the screen. So, the seats are filled from the top row first column and descends accordingly.

Consider the below seating array. Initially all seats are vacant.

	1	2	3	4	5
E	TRUE	TRUE	TRUE	TRUE	TRUE
D	TRUE	TRUE	TRUE	TRUE	TRUE
C	TRUE	TRUE	TRUE	TRUE	TRUE
B	TRUE	TRUE	TRUE	TRUE	TRUE
A	TRUE	TRUE	TRUE	TRUE	TRUE

----------SCREEN----------

Customer A requests a hold for 4 seats. Since all seats are available, seats E1, E2, E3, E4 are held for him. He takes 15 seconds to complete the reservation.

At the same time, Customer B requests a hold for 2 seats. Since only one seat is available in the top (E) row, the next best seats D1, D2 are held for him. He does not wait and finalizes the reservation immediately.

Simultaneously, Customer C request a hold for 5 seats. The best available seats for him are C1, C2, C3, C4, C5. He waits for 35 seconds before proceeding with the reservation.

Now, the seat array looks like this

	1	2	3	4	5
E	FALSE	FALSE	FALSE	FALSE	TRUE
D	FALSE	FALSE	TRUE	TRUE	TRUE
C	FALSE	FALSE	FALSE	FALSE	FALSE
B	TRUE	TRUE	TRUE	TRUE	TRUE
A	TRUE	TRUE	TRUE	TRUE	TRUE

----------SCREEN----------

After 5 seconds, Customer D requests a hold of 18 seats. However, the number of available vacant seats are 14, so his request could not be processed.

After another 5 seconds, Customer D again requests a hold for 13 seats. Since there are no 13 seats available in the same row, he is allotted seats E5, D3, D4, D5, B1, B2, B3, B4, B5, A1, A2, A3, A4 along multiple rows.

	1	2	3	4	5
E	FALSE	FALSE	FALSE	FALSE	FALSE
D	FALSE	FALSE	FALSE	FALSE	FALSE
C	FALSE	FALSE	FALSE	FALSE	FALSE
B	FALSE	FALSE	FALSE	FALSE	FALSE
A	FALSE	FALSE	FALSE	FALSE	TRUE

----------SCREEN----------

Now, Customer A proceeds with the reservation. It has been 15 seconds since his hold was created. Since the hold is valid for 15 more seconds, his reservation is confirmed.

When Customer C tries to finalize his reservation, his request did not go through since his hold expired 5 seconds back.

So now, the seat array looks like

	1	2	3	4	5
E	FALSE	FALSE	FALSE	FALSE	FALSE
D	FALSE	FALSE	FALSE	FALSE	FALSE
C	TRUE	TRUE	TRUE	TRUE	TRUE
B	FALSE	FALSE	FALSE	FALSE	FALSE
A	FALSE	FALSE	FALSE	FALSE	TRUE

----------SCREEN----------

After few seconds, C again tries to hold 5 seats and this time he immediately proceeds with the reservation without waiting. So, seats C1, C2, C3, C4, C5 were held and reserved for him.

Finally, the seating looks like this.

	1	2	3	4	5
E	FALSE	FALSE	FALSE	FALSE	FALSE
D	FALSE	FALSE	FALSE	FALSE	FALSE
C	FALSE	FALSE	FALSE	FALSE	FALSE
B	FALSE	FALSE	FALSE	FALSE	FALSE
A	FALSE	FALSE	FALSE	FALSE	TRUE

----------SCREEN----------

No more customer tries to reserve the tickets before the show time and hence the booking for this particular show has completed.

Algorithm for holding and reserving the seats

Find best available seats for a customer

Inputs:

- No. of seats required (numSeats)
- Duration the customer waits to reserve the seats after hold requests (waitTime)
- Customer Email (email)

Output:

- Confirmation code of the booking if the reservation is confirmed. Null otherwise.

Algorithm:

I. Get the current number of vacant seats (availability) which are neither held nor reserved, after discarding the expired holds.
II. If availability is less than numSeats, return null.
III. Else, proceed to find the best possible seats.
    a. Loop through each row and get the number of vacant seats in that row (rowCount).
        i. If rowCount is greater or equal to numSeats, return the required number of continuous vacant seats.
        ii. If no such row was found, whose rowCount is greater or equal to numSeats, return required number of seats in multiple rows.
IV. Create a SeatHold object and assign the matched best seats to it.
V. Wait for waitTime milliseconds.
VI. To proceed with the reservation, first check if the SeatHold object has expired or not.
    a. If so, return null.
    b. Else create a new booking with the SeatHold details.
    c. Return the booking confirmation number.