Submission of ReserveTicket for Mahmoud Abunasser

Program design

The program is developed using Java Netbeans. The use of Java Gradle was not accomplished, this can be added upon your request. The program is designed in 5 classes. It implements two categories of Venue by utilizing OOP principles available in Java. The advanced Venue is RectVenue where the location of seats is taken into consideration when a reservation is requested. The more basic category only keeps track of the capacity and availability of the venue. The design nicely splits the functionalities into the base and sub classes.

Classes

1- ReserveTicket

The class that contain the static main method. It has a loop that creates the command line menu that calls all functionalities.

2- Venue

The class that implements the TicketService; The interface indicated in the instruction sheet. This is the basic class of the application that keeps track of seat holds and reservations. It also keep track of the capacity of the venue. Note that the signature for reserveSeats is changed, the new signature make more sense with the design of classes in my solution.

This class implements a simple venue where the positions of seats are not indicated -- a more elaborate solution is given in a subclass. It keeps track of how many seats are available, the seats that are reserved and the seats that are on hold.

The functionality of cleaning up expired seats is implemented in the Venue class. Upon the creation of a hold, set a timer for it to timeout (number of millisecond to timeout is an attribute of seatHold, a static data member). When the timer is called, the hold is flagged for deletion but will not deleted right away. The actual deletion will take place upon counting the number of holds (which also invoked when showing the available holds). This mechanism prevents the deletion of holds when the user is about to reserve the hold. Consider the following scenario: A user makes a hold for a customer, then go to the menu to finalize the reservation. This menu item lists of available holds and let the user pick the one. The customer asks some questions before the user make the selection, this causes the hold to time out while the selection of the hold is possible. The user is expected to be frustrated if the hold that seem to be present is actually deleted. My implementation is expected to enhance the user experience by keeping the hold until the final selection is made.

This class keeps track of which seatHold is still on hold and which seatHold is reserved by maintaining two lists of seathold. Another valid approach might store the status of a seatHold in the SeatHold class and keep track of one list for both holds and reservations. I opted for the design I have because the other design might result with a less efficient solution to find holds or present holds mixed with reservations to the user.

Making a final reservation is implemented in reserveSeat. This involve prompting the user with available holds, allowing the user to select one, and implement the hold for the selected hold. I was hesitant to implement prompting the user for input in this class. It might desired to keep prompts for all user inputs in the main function. My implementation missed that point, but I believe it achieved a simpler solution.

I achieved a more user friendly solution by allowing the user to reserve his last hold by entering zero for the hold ID to be reserved. This is a simplification of frequently occurring workflow and expected to enhance the user experience and reduce possibilities for error.

3- RectVenue

A subclass of the of the Venue class that allow the selection of best seats for holds as they come. It facilitates a mathematical definition of best seat. A seat is better if it is closer to the middle point in the front line of the stage. The distance between the stage and the front seat, width and depth of seats are attributes of the RectVenue. They are assigned default values upon the creation of the object. These values can be parametrized to allow a more dynamic design of the RectVenue, this is left for future improvements.

The thoughtful design of RectVenue and its base class allowed for the addition of significant functionality while it contains only one function to be overridden. The constructor initializes the setup of the seats in the venue. The drawback of this design is having the number of seats stored in addition to the number of rows and columns; a data storage redundancy which could become a potential source for bugs in future. This problem can be solved by introducing another subclass of Venue (say SimpleVenu) that contains the number of seats as a member variable and in the Venue class as abstract. The Venue class will seize to hold the data member: seatCount. Methods referencing seatCount will turn into abstract methods.

4- Seat

This is a helper class for RectVenue. It is used to store information about the seats in the venue. In case of having a RectVenue, the reservations and holds will keep track of seats by using this class. This class also allow for sorting the seats by their bestness using: Collections.sort.

5- SeatHold

A class defining the attributes and behaviour of holds and reservations. Contain static methods to allow for the entry and validation of email addresses. The validation functionality is not to be considered as complete by any means. A library might be used to enhance the validation of email entry. A real life solution is expected to pull the email from already existing source, such as the database of customers.

It stores two static data members:The Id of the next hold to come and the time for a hold to be kept before it timeout. They are static because they are are shared among all objects.

It also store the hold information and whether the hold had timed out or not. The deletion of a timed out hold will not take place when a user can potentially make a reservation. The design guideline was: give a user the chance to make a reservation of a timed out hold if the user can select that hold.

It also has the option to store the list of seats in the hold. This is mainly utilized when the venue is of type RectVenue.

Potential improvements

- Contribute the closeness of sets in a seathold to their rank. The current solution does not guarantee sets to be next to each other. For example: a venue of 10*10, first reservation of 20 seats, and second reservation of 3 seats. The second reservation will have dispersed seats, they are the best next available seats. Our second customer is not expected to be satisfied by his/her reservation. A better solution might involve allowing the customer to pick from the set of best clusters of seats, letting the user to change clusters might also be a good idea.
- The mechanism to keep the hold while the user is making a selection works fine for single user. More testing and potentially modifying the design might be necessary If the system is being accessed by multiple users concurrently.

Test Cases

Here are some examples of test cases that I implemented.

Test case	**response**	**Successful**
Venue of one seat and make one reservation holding single seat.	Done	Yes
Enter a venue of one row and one column and reserve the only seat	Done	Yes
Try to hold more seats.	Venue fully booked	Yes
Restart application		
Trying to hold zero seats	Not a valid entry	Yes
Trying to hold negative number of seats	Not a valid entry	Yes
Restart application		

Entering number of rows of the venue of value zero or less	Not a valid entry	Yes
Entering number of columns of the venue of value zero or less	Not a valid entry	Yes
Enter venue dimension of 20*20		
Make a hold of 19 seats. To check if best seats are assigned	The hold is given seats from the second row	Yes
Make a hold and let time out	Timed out successfully within the specified time, the hold cannot be reserved.	Yes
Make a hold and go to reserve, let timeout before selecting	Timed out successfully and the reservation can be completed because the user can select it	Yes
Make three seat holds and view the holds. Select the holds one at a time to test the functionality of selecting last hold by entering 0	Done. Difficult to do before they timeout. Might need to increase SeatHold.timeOutMillisconds	Yes
Repeat the point above many times (make section of hold based on the last hold and based on previous hold in different combinations)	Done	Yes
Make the last hold of one more than the available seats	Not available	Yes
Make the last hold of exactly the number of available seats	Done	Yes