COMP 360, Assignment 5

In this assignment, you will explore some fun and exciting Java classes and ways of using them. You will write a program to manage the reservations for a performance at a theater (*a la* Ticketmaster). To keep things simple, there is only one performance, and the theater has a single row of seats that are numbered with increasing integers starting from zero.

Read this entire document before you start working so you can get a feel for how the classes fit together.

Getting Started

Make a new NetBeans project (or Eclipse, or your IDE of choice).
In NetBeans, Choose File, New Project, Java Application, Next, fill in a project name, uncheck "create main class", and choose "Finish."
Download the zip file from the class webpage containing the starter code. Unzip those files and copy them into your NetBean project's "src" directory. The files should then appear in NetBeans.

Classes to Edit

- The `Reservation` class manages a single reservation in the theater. A reservation should store the name of the person who made the reservation, as well as what seats in the theater are reserved for that person. Each reservation's seats are always consecutive.

  Decide how you will represent the information a Reservation has to know about and add appropriate private fields to the class to hold the information (see below for suggestions).

  Add the following methods to the Reservation class:

  - `public Reservation(String name, int startSeat, int howMany)`: Constructor. Creates a new Reservation for the person with the given name, starting in the seat specified, and containing "howMany" consecutive seats.
  - `public String getName()`: Returns the name of the person who made this reservation.
  - `public int getFirstSeat()`: Returns the first seat number in the reservation.
  - `public int getLastSeat()`: Returns the last seat number in the reservation.
  - `public int getNumberOfSeats()`: Returns the number of seats in the reservation.
  - `public String toString()`: Returns a textual representation of this reservation. Include the name of the reservation, the starting seat, and the ending seat.

  Implementation suggestions: This class should not be particularly complicated. You need three fields: a String and two integers.

- The `Theater` class manages reservations for the performance. You should decide what data structure to use to store the reservations.

  Add the following methods to the Theater class:

  - `public Theater(int size)`: Constructor. Creates a new Theater with the number of seats specified. The seats are labeled 0 to size−1.

- `public Reservation makeReservation(String personName, int howManySeats)`: Attempts to reserve a block of "howManySeats" consecutive seats for the person specified. If the theater has a block of empty seats of the appropriate size, a new `Reservation` object is created and returned. If there are no blocks of seats available that are large enough to accommodate this request, this method returns `null`.

  A person cannot request specific seats; seats are allocated starting at seat 0 and increasing from there. Within a reservation, all the seats must be next to each other; that is, if a person asks for four seats, you should allocate them the first available group of four *consecutive* seats. So if the theater has ten seats and all are available, a person requesting four seats would be allocated seats 0–3. The next person requesting four seats would be given seats 4–7. The next person requesting four seats would be denied a reservation.

  You may assume that no person will ever make more than a single reservation (all the names in the reservations will be unique).

- `public boolean cancelReservation(String personName)`: Removes the reservation for this person from the theater and returns `true`. If the person specified has no reservation, this request is ignored and the method returns `false`.

- `public Reservation lookupReservation(String personName)`: Returns the reservation for this person, assuming there is one. If there is not, returns `null`.

Implementation suggestions: There are lots of ways to store the reservations. You need a technique that enables two things: (1) fast lookup by name, and (2) a fast way to know what seats are available and which are not (so you can quickly find blocks of empty seats).

- You do not need to make any changes to the `Ticketmaster` class. This class holds the `main` method for the project, as well as additional code to read commands from the keyboard and process them. You should take a look at the code and make sure you understand how it works, but you shouldn't need to change anything here.

  The code should compile and run "out of the box," even without changes on your part. Many of the methods return placeholder values (like null or false) that you can remove when you start implementing those methods.

Sample Run

```
Enter theater size: 20

Enter command: reserve kirlin 6
Made reservation: [Reservation for kirlin: seats 0 through 5]
Theater now looks like: [xxxxxx--------------]

Enter command: reserve sanders 4
Made reservation: [Reservation for sanders: seats 6 through 9]
Theater now looks like: [xxxxxxxxxx----------]

Enter command: cancel kirlin
Reservation canceled.
Theater now looks like: [------xxxx----------]

Enter command: reserve seaton 3
Made reservation: [Reservation for seaton: seats 0 through 2]
Theater now looks like: [xxx---xxxx----------]

Enter command: reserve mouron 12
Could not make reservation.

Enter command: reserve gottlieb 7
Made reservation: [Reservation for gottlieb: seats 10 through 16]
Theater now looks like: [xxx---xxxxxxxxxx---]

Enter command: lookup sanders
[Reservation for sanders: seats 6 through 9]

Enter command: cancel sanders
Reservation canceled.
Theater now looks like: [xxx-------xxxxxxx---]

Enter command: reserve mouron 12
Could not make reservation.

Enter command: cancel gottlieb
Reservation canceled.
Theater now looks like: [xxx-----------------]

Enter command: reserve mouron 12
Made reservation: [Reservation for mouron: seats 3 through 14]
Theater now looks like: [xxxxxxxxxxxxxxx-----]

Enter command: end
```

Assessment

Solutions should be:

- Correct
- In good style, including indentation and line breaks
- Written using features discussed in class.

Turn-in Instructions

- Upload `Ticketmaster.java`, `Theater.java`, and `Reservation.java` to Moodle before the project deadline.