Fall 2026 COP 3330 Program #8: Ticketing System with Exceptions

For this assignment, you'll modify either your solution or the posted solution for Program #3. For convenience, the original write-up for program #3 is included at the end of this assignment.

There are two main tasks for this assignment edit:

Task #1

In the original assignment minor error-checking was implemented using the tools available with the syntax taught to that point in time:

1. Invalid menu selections were screened out and users were reprompted.
2. Users who entered invalid regions were prompted
3. Negative numbers of seats weren't sold.

Now that you can create your own exception, modify the program by creating three new exception classes for each of these items:

InvalidMenuSelectionException
InvalidRegionException
NegativeSeatException

and then **catching** these exceptions in your program so that it behaves similarly to program 3.

Task #2

If the user accidentally enters a non-integer for any item, catch the InputMismatchException and handle it in a way that you see fit.

Input Specification

For any option, the user might accidentally mis-type and enter a non-integer value. These issues should be handled by catching the InputMismatchException and dealing with it in a way that you see fit.

Once an integer is entered for a choice, it may not be in range in three specific ways as previously described. Create exceptions for these three ways and catch those exceptions in the appropriate place and handle them as you see fit.

Output Specification

Please follow the output specification of the original assignment when possible. For any Exceptions that are caught, create output (reprompts, etc.) as you see fit to handle the situation.

Deliverables

Please submit a single zip file of a directory containing all of your files. for the assignment.
At a minimum there should be the 3 exception files and a file called ticketing_v3.java. Make sure to include appropriate comments and use good programming style. **Clearly indicate what code was added to the program 3 solution you modified.**

Original Program 3 Description

In the last part of the previous assignment, we printed out a "simulated" seating chart. Since we hadn't learned arrays yet, we couldn't store a record of the seating chart in our program. Now that arrays have been taught, we can do a program which manages a seating chart!

To make life a bit easier, we'll have a fixed seating chart. We'll make our seating chart with 26 rows, each with 60 seats. The rows will be labeled A – Z, from front to back and the seat numbers will go 1 to 60 from left to right.

We'll split the seats into 6 regions and the cost of every seat within a region will be the same.

Here are the regions and the cost of the seats in that region:

Region	Seats in Region	Cost/Seat in Region
1	Rows A – M, Seats 1 – 20	$200
2	Rows A – M, Seats 21 – 40	$300
3	Rows A – M, Seats 41 – 60	$200
4	Rows N – Z, Seats 1 – 20	$100
5	Rows N – Z, Seats 21 – 40	$150
6	Rows N – Z, Seats 41 – 60	$100

Whenever a user wants to buy a seat, you'll ask them which region (1 – 6) they want to buy a seat, and how many seats they want to buy.

If there are enough open seats in the region, automatically select the seats in order by row (from earlier letter row to later letter row) sorted by seat number (lower seat numbers first). You should print out to the user which seats they are receiving as well as total cost of the purchase.

If there are not enough open seats in the region, then you inform the user that this is the case and do not perform a transaction.

The user will also have an option to view the whole seating map, showing which seats are taken and which are still open.

The third option the user will have is to view the total number of tickets sold so far as well as the total revenue.

When the user quits the program, print out a final report with the final seating chart, the number of tickets sold, and the total revenue collected from the ticket sales.

Required Static Methods (other than main)

To make this a bit easier for you, and also so the graders aren't dealing with programs that are poorly designed, several static methods will be required with their behavior will be specified below. You may write MORE static methods that are listed here, but these are the bare minimum requirements:

```
// Returns a character array with 26 rows and 60 columns
// representing the seating chart for the concert. The
// array should be initiated so that all cells store 'O'.
public static char[][] makeEmptyChart();

// Asks the user to enter a menu choice and continues to prompt
// the user until they enter a valid choice and returns that
// choice. (Choices are 1 = buy, 2 = print chart,
// 3 = list tickets sold, total revenue, 4 = quit.
public static int getMenuChoice(Scanner stdin);

// Asks the user for the region in which they want to buy
// tickets and continues to prompt the user until they enter
// a valid choice and returns that choice (1 to 6)
public static int getRegion(Scanner stdin);

// Prints out the current seating chart for the concert.
// Please follow the format shown on the sample output.
public static void printChart(char[][] seats);

// Returns the total cost of buying numTickets number of tickets
// in region number region using the seating chart seats. If
// too few tickets are available in the region, 0 is returned
// and no tickets are issued. Otherwise, the method lists each
// ticket sold (follow sample output for format) and returns the
// total cost of all of the tickets.
public static int buySeats(char[][] seats, int region, int
numTickets);

// Returns the number of open seats in the seating chart seats
// in region, region. region is guaranteed to be in between 1
// and 6.
public static int numOpenSeats(char[][] seats, int region);
```

Required Constants

For full credit, you must use these constants in all relevant places throughout your code instead of the corresponding integer literals:

```
final public static int TOTAL_ROWS = 26;
final public static int TOTAL_COLS = 60;
final public static int REGION_ROWS = 13;
final public static int REGION_COLS = 20;

final public static int[] COST = {200, 300, 200, 100, 150, 100};
```

The first two specify the dimensions of the whole seating chart.
The last two specify the dimensions of each region.

Note that for full credit, you'll have to use these in such a way that if the values of TOTAL_ROWS and TOTAL_COLS were to change to be different multiples of REGION_ROWS and REGION_COLS, that your code would work perfectly without any other changes. (The exception of course is the A – Z printing, since that doesn't extend past 26...)

Input Specification

For the menu choice, the user will always enter values that fit into the integer data type and will eventually enter an integer in between 1 and 4, inclusive. (Your menu method will screen out all invalid choices.)

For option #1 of the menu, the user will enter values that fit into the integer data type for region and number of tickets and will eventually enter an integer in between 1 and 6 for region. (The method for reading in the region will screen out all invalid entries.

Output Specification

Please follow the output samples given, but here is a rough description of the format for output for each menu option:

For option #1, please first ask the user for the region in which they want to buy tickets. Once they enter a valid region (after possibly several attempts), ask the user how many tickets they want to buy in that region. Then, go ahead and execute buying the seats. If the user wanted too many tickets (not enough available in the region), then output the following message:

```
Sorry the transaction couldn't be made.
Not enough available seats.
```

If the user wanted a negative number of tickets, output:

```
Sorry you can't buy a negative number of seats.
```

If the user's request can be accommodated, then output the following header line:

```
Here are the tickets you are receiving:
```

Follow this with the list of tickets, with exactly 20 tickets per line (except possibly the last line which might have fewer). Each ticket should be listed as a row followed by a number with no space in between the row and number, but a space after the number. For example, if the tickets purchased were A21, A22 and A23, the output, with the header, would be:

```
Here are the tickets you are receiving:
A21 A22 A23
```

Add a blank line after the last line with output (so one line with no characters).

For option #2, please print the chart according to the following format. Here is a full empty chart:

```
Here is the current seating chart:

  12345678901234567890 12345678901234567890 12345678901234567890

A OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
B OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
C OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
D OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
E OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
F OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
G OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
H OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
I OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
J OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
K OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
L OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
M OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO

N OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
O OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
P OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
Q OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
R OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
S OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
T OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
U OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
V OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
W OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
X OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
Y OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
Z OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOOOOOOOO
```

The only difference for what should be printed is that the 'X' character should be in each location where a seat was purchased.

For option #3, please print two lines with the following format:

```
Current tickets sold = X
Current revenue = $Y
```

where X is the number of tickets sold and Y is an integer representing the current revenue in dollars.

Finally for option #4, print a message with the following format:

```
Thank you for using Ticketing System V1
Total tickets sold = X
Total revenue = $Y
Here is the final seating chart:
```

with the final seating chart printed below using the format previously provided.