

Fall 2026 COP 3330 Program #4: Ticketing System Version 2

In this assignment you will mostly duplicate the functionality of program 3, but with an object-oriented design. Instead of interacting with a user with prompts, your program will read input from standard input (without being prompted) and process similar ticketing requests as program 3.

Assignment Requirements

The input will be guaranteed to follow a format (values entered will conform to the desired bounds), so less error checking will be required for this version. In addition, the main method in the class Ticketing_V2.java will be given to you, so for the assignment all you will have to do is fill in the methods for two separate classes:

Ticket
Inventory

To test your program, you'll have to run it on the command line, piping the input to it and redirecting the output to a file. Then you can check to see if the output matches the desired output.

Here is what you type on the command prompt to do this:

```
java Ticketing_V2 < test_input.txt > myoutput.txt
```

This will run the program assuming that the contents of test_input.txt are typed in as standard input and then print the output to the file myoutput.txt. On a Mac, you might have to type "." before you type java.

Ticket Class

Here are the instance variables for the class:

```
private char row;  
private int seatNum;  
private int faceValueCost;
```

The row will be a letter from A to Z, the seat a non-negative integer and the faceValueCost will be a positive integer.

Here are the methods you'll need to fill in:

```
// Creates a Ticket object for row thisRow, seat number thisSeat  
// with an face value cost of faceValueCost.  
public Ticket(char thisRow, int thisSeat, int initCost)  
  
// Returns the face value cost of this ticket.  
public int getFaceValue()  
  
// Returns a string representation of this ticket.  
public String toString()
```

Inventory Class

Here are the instance variables for the class:

```
private Ticket[][] seats;  
private boolean[][] sold;  
private int numColSections;  
private int numRegions;  
private int numSold;  
private int totalRevenue;
```

seats is a 2D array of Ticket with 26 rows and $20n$ columns, where n is the number of sections by column. There is one entry in this array for every seat in the Inventory object.

sold is the same size as seats, and will be set to true if a ticket for that seat is sold and false otherwise.

numColSections will be set to the value of n described previously.

numRegions will be twice numColSections, essentially the total number of regions the Inventory is subdivided into.

numSold will equal the total number of tickets sold in this Inventory object.

totalRevenue will equal the total revenue for the tickets sold in this Inventory object.

Here are the methods to fill in for this class:

```
// Pre-condition: The dimension of prices is 2 by  $n$ , corresponding  
// to an arena with 26 rows and  $20n$  seats, with sections numbered  
// 1 to  $2n$ . The first  $n$  sections are rows A-M, and the last  $n$   
// sections are rows N-Z. prices[i][j] is the initial cost of every  
// ticket in section  $n*i+j+1$ .
```

```
// Post-condition: Initializes the inventory to be 26 rows and  $20n$   
// columns of tickets, set to the price indicated in the prices  
// array, and setting no tickets as sold.
```

```
public Inventory(int[][] prices)
```

```
// Returns the current # of total tickets sold in the inventory.  
public int getNumSold()
```

```
// Returns the total revenue collected from original sales.  
public int getTotalRevenue()
```

```

// Returns the number of available seats in the section with number
// secNum.
// Note: if an invalid section number is given, 0 should be
// returned automatically.
public int numAvailable(int secNum)

// Returns true iff we can buy numTickets tickets from region
// number region. Note: Should return false if numTickets < 0.
public boolean canBuy(int region, int numTickets)

// Pre-condition: Should only be called if numTickets number of
// tickets can be bought from the region with the region number
// region.

// Post-condition: Executes the purchase, marking numTickets
// tickets (in the order described in program 3) as sold in section
// with the number secNum. For each ticket sold, a single line
// should be printed with the String representation of a Ticket
// object. Use the sample input/output to determine this
// representation.
public void buySeats(int secNum, int numTickets)

// Returns a String representation of the current object.
// Note: This is the full string that gets printed for option #2,
// including the header "Here is the current seating chart:"
public String toString()

```

Samples Program Runs

Three test input/output samples have been provided. If you fill in the Ticket class and Inventory class properly, then when you run the posted Ticketing_V2.java, you should get the exact output that is provided.

Deliverables

Please submit two .java files:

- 1) Ticket.java
- 2) Inventory.java

The TAs will download your files and put them in the same directory as the posted Ticketing_V2.java file and then run the application to grade your work.

Please make sure to include internal comments in both files.