

## Question 2

Not yet answered

Marked out of 25.00

v6 (latest)

### Part 2: Java

You are going to make a system that represents the seats in a venue, such as a theatre or a cinema, and a process for booking seats at events that are held in that venue. The tasks will get harder as you progress, and build on each other, so try to ensure that you complete each stage before moving on.

We have provided initial versions of **Seat.java** and **SeatType.java** which you should use to complete the tasks. You can add any additional fields, methods, or modifiers to these classes as needed, and can also write and submit any additional classes that are needed, but please **do not change the provided packages, class names, or method names**.

Here are the initial versions of the two files -- you should download these and use them to create your solution.

- [SeatType.java](#)
- [Seat.java](#)

All classes you create should be in the **boxOffice** package.

#### Java Task 2a: Representing individual seats (8 Marks)

For our purposes, a seat has four core properties:

- The **row**, which must be a single character representing a capital letter A through Z
- The **seat number**, which must be a positive integer
- The **seat type**, which is represented by the provided **SeatType** enumerated type
- The **availability**, which is a Boolean value indicating whether the seat has been reserved

1. Write code to represent individual seats using the above properties. The full details of the implementation are up to you, but you must use the provided **Seat.java** file as a starting point for representing the core properties. A seat should be initially available when it is created, while the other properties should be set in the constructor. **[4 marks]**
2. Be sure that your constructor validates that the values of all core properties are valid: that is, the row and seat number are in range. If an attempt is made to create a seat object with invalid values, your constructor should throw an **IllegalArgumentException** with a descriptive message. **[2 marks]**
3. Include getter methods for all core properties, as well as a setter method for the **availability** property only. **[1 mark]**

#### Java Task 2b: Representing a venue (8 Marks)

A **venue** is a location such as a theatre which has a number of rows, each of which has a number of seats. The row letters always start at A, and the seat numbers always start at 1. Note that the number and distribution of seats may be different in each row. There can be at most 26 rows.

For example, the following might be the seat configuration in a very small venue:

Row/Seat	1	2	3	4	5	6
A	Standard	Deluxe	Deluxe	Standard		
B	Deluxe	Deluxe	Deluxe			
C	Standard	Standard	Standard	Standard	Standard	Standard

A venue configuration can be given by a string of the following format:

- First line: number of rows
- Remaining lines: seat configuration in each row, separated by spaces

For example, the above venue would be specified by the following string:

```
3
S D D S
D D D
S S S S S S
```

In Java, you could specify this string like this: "3\nS D D S\nD D D\nS S S S S S"

1. Create a class called **Venue** to represent the seats in a venue as described above, using your **Seat** class from Task 2a. You can use whatever internal representation you choose. Your class should have one constructor which should take a string like above and create the necessary **Seat** objects. You can assume that the string is well-formed and do not need to do any error checking. **[4 marks]**
2. Add a method to the **Venue** class with the following signature:  
**public Seat getSeat (char row, int seatNum)**  
This method should return the **Seat** object corresponding to the given row and seat number. If the row or seat number is invalid, this method should throw an **IllegalArgumentException** with a descriptive message. **[2 marks]**
3. Add a method to the **Venue** class with the following signature:  
**public void printDetails()**  
This method should print out the details of all seats in all rows of the venue in a human-readable form. The details are up to you, but simply using **toString()** on the internal representation will not be sufficient for full marks. **[2 marks]**

## Java Task 2c: Representing an event (8 Marks)

An **Event** is a particular show that takes place at a venue. The details of an event include:

- The venue where the event will take place
- A price structure for tickets – that is, the price in pounds of each ticket type. For example, this might be:
  - Standard tickets: £10
  - Deluxe tickets: £20





An event also allows seats to be **reserved** – a customer can request one or more seats of a given type (Standard or Deluxe). If a customer is booking more than one seat, the seats must be adjacent – that is, the seats must have consecutive numbers and be in the same row. Once a seat has been reserved by a customer, it cannot be reserved by another unless the seat is **returned**.


1. Create an **Event** class to represent the details of an event, using the classes from the previous tasks. You can assume that all ticket prices can be represented as integers. **[2 marks]**
2. Add a method **reserveSeats** to the **Event** class with the following signature:  
**public int reserveSeats (int numSeats, SeatType seatType)**  
This method should attempt to find the indicated number of adjacent seats of the given type that are available. If enough seats are found, they should all be reserved, and the total price of the seats returned. If enough seats are not found, no seats should be reserved and the method should return -1. **[4 marks]**
3. Add a method **returnSeat** to the **Event** class with the following signature:  
**public void returnSeat (char row, intSeatNum)**  
If the indicated seat is reserved, it should be made available again. If the indicated seat is already available, or if the row and seat number are invalid, this method should throw an **IllegalArgumentException** with appropriate information. **[2 marks]**

### Java submission

Be sure that your classes are named as above, and that all classes are in the **boxOffice** package. You must submit all the .java files you wrote, including the final versions of **Seat.java** and **SeatType.java**.

Maximum size for new files: 100 MB

 [Files](#)   

  
You can drag and drop files here to add them.

Accepted file types

Text file .java