### **2023 Java Exam Question: Venue and Event Management**

#### **1. Seat Class (Task 2a)**

java

Copy code

package boxOffice;

public class Seat {

private char row;

private int seatNumber;

private SeatType seatType;

private boolean isAvailable;

// Constructor

public Seat(char row, int seatNumber, SeatType seatType) {

if (row < 'A' || row > 'Z') {

throw new IllegalArgumentException("Row must be between A and Z");

}

if (seatNumber <= 0) {

throw new IllegalArgumentException("Seat number must be a positive integer");

}

this.row = row;

this.seatNumber = seatNumber;

this.seatType = seatType;

this.isAvailable = true;

}

// Getters

public char getRow() {

return row;

}

public int getSeatNumber() {

return seatNumber;

}

public SeatType getSeatType() {

return seatType;

}

public boolean isAvailable() {

return isAvailable;

}

// Setter for availability

public void setAvailable(boolean isAvailable) {

this.isAvailable = isAvailable;

}

@Override

public String toString() {

return String.format("Seat[row=%c, number=%d, type=%s, available=%s]",

row, seatNumber, seatType, isAvailable);

}

}

#### **2. Venue Class (Task 2b)**

java

Copy code

package boxOffice;

import java.util.ArrayList;

import java.util.List;

public class Venue {

private List<List<Seat>> seats;

// Constructor

public Venue(String configuration) {

seats = new ArrayList<>();

String[] lines = configuration.split("\n");

int numberOfRows = Integer.parseInt(lines[0]);

for (int i = 1; i <= numberOfRows; i++) {

List<Seat> rowSeats = new ArrayList<>();

String[] seatTypes = lines[i].split(" ");

char row = (char) ('A' + i - 1);

for (int j = 0; j < seatTypes.length; j++) {

SeatType type = seatTypes[j].equals("S") ? SeatType.STANDARD : SeatType.DELUXE;

rowSeats.add(new Seat(row, j + 1, type));

}

seats.add(rowSeats);

}

}

public Seat getSeat(char row, int seatNum) {

int rowIndex = row - 'A';

if (rowIndex < 0 || rowIndex >= seats.size() || seatNum <= 0 || seatNum > seats.get(rowIndex).size()) {

throw new IllegalArgumentException("Invalid row or seat number");

}

return seats.get(rowIndex).get(seatNum - 1);

}

public void printDetails() {

for (List<Seat> row : seats) {

for (Seat seat : row) {

System.out.println(seat);

}

}

}

}

#### **3. Event Class (Task 2c)**

java

Copy code

package boxOffice;

import java.util.ArrayList;

import java.util.List;

public class Event {

private Venue venue;

private int standardPrice;

private int deluxePrice;

public Event(Venue venue, int standardPrice, int deluxePrice) {

this.venue = venue;

this.standardPrice = standardPrice;

this.deluxePrice = deluxePrice;

}

public int reserveSeats(int numSeats, SeatType seatType) {

for (List<Seat> row : venue.seats) {

List<Seat> availableSeats = new ArrayList<>();

for (Seat seat : row) {

if (seat.isAvailable() && seat.getSeatType() == seatType) {

availableSeats.add(seat);

if (availableSeats.size() == numSeats) {

int totalPrice = seatType == SeatType.STANDARD

? standardPrice \* numSeats

: deluxePrice \* numSeats;

for (Seat reservedSeat : availableSeats) {

reservedSeat.setAvailable(false);

}

return totalPrice;

}

} else {

availableSeats.clear();

}

}

}

return -1;

}

public void returnSeat(char row, int seatNum) {

Seat seat = venue.getSeat(row, seatNum);

if (!seat.isAvailable()) {

seat.setAvailable(true);

} else {

throw new IllegalArgumentException("Seat is already available");

}

}

}

### **2022 Java Exam Question: Graph Representation**

#### **1. Node Class (Task 2a)**

java

Copy code

package graphs;

import java.util.ArrayList;

import java.util.List;

public class Node {

private int label;

private List<Node> neighbours;

public Node(int label) {

this.label = label;

this.neighbours = new ArrayList<>();

}

public void addNeighbour(Node node) {

neighbours.add(node);

}

public List<Node> getNeighbours() {

return neighbours;

}

public int getLabel() {

return label;

}

@Override

public String toString() {

return String.valueOf(label);

}

@Override

public boolean equals(Object obj) {

if (this == obj) return true;

if (!(obj instanceof Node)) return false;

Node other = (Node) obj;

return this.label == other.label;

}

@Override

public int hashCode() {

return Integer.hashCode(label);

}

}

#### **2. GraphParser Class (Task 2b)**

java

Copy code

package graphs;

public class GraphParser {

public static Node[] parseGraph(String spec) {

String[] lines = spec.split("\n");

int numNodes = Integer.parseInt(lines[0]);

Node[] nodes = new Node[numNodes];

for (int i = 0; i < numNodes; i++) {

nodes[i] = new Node(i + 1);

}

for (int i = 2; i < lines.length; i++) {

String[] edge = lines[i].split(" ");

int from = Integer.parseInt(edge[0]) - 1;

int to = Integer.parseInt(edge[1]) - 1;

nodes[from].addNeighbour(nodes[to]);

}

return nodes;

}

}

#### **3. Edge Class (Task 2c)**

java

Copy code

package graphs;

public class Edge {

private Node start;

private Node end;

public Edge(Node start, Node end) {

this.start = start;

this.end = end;

}

@Override

public String toString() {

return "(" + start + ", " + end + ")";

}

@Override

public boolean equals(Object obj) {

if (this == obj) return true;

if (!(obj instanceof Edge)) return false;

Edge other = (Edge) obj;

return start.equals(other.start) && end.equals(other.end);

}

@Override

public int hashCode() {

return start.hashCode() + end.hashCode();

}

}

#### **4. GraphExplorer Class (Task 2d)**

java

Copy code

package graphs;

import java.util.HashSet;

import java.util.Set;

public class GraphExplorer {

public static Set<Edge> listEdges(Node[] nodes) {

Set<Edge> edges = new HashSet<>();

for (Node node : nodes) {

for (Node neighbour : node.getNeighbours()) {

edges.add(new Edge(node, neighbour));

}

}

return edges;

}

}