## Saveetha School of Engineering

**Saveetha Institute of Medical and Technical Sciences Department of Computer Science Engineering**

#### Assignment on topic: Unit 2

##### CO2: [Develop object-oriented programs using](https://docs.google.com/document/d/10f_xjkVwmQZAh87hr4x56QfoXzI4cEdQdbaNibtDYBo/edit#heading%3Dh.pi90ampbgfqv) Lists, Collection and File I/O.

How would you design the booking bus ticket class and implement the described functionality for this online bus booking system? Include the code and expected output based on the given scenario.

#### Scenario:

You are a software developer working for a bus company called **CityTransit**. CityTransit operates multiple bus routes across different cities, providing regular and reliable transportation services for commuters.

The company aims to implement an **online bus reservation system** that allows passengers to:

* View available bus tickets for different routes.
* Check seat availability on each bus.
* Book seats for their desired routes.
* Cancel bookings and re-book tickets if needed.
* Prevent double-booking of seats on the same bus.

**Tasks:**

#### Define the BusTicket class:

* + Create attributes to store the bus name, route, departureTime, price, seat number, and seat availability.

#### 2. Implement Methods in the BusTicket Class:

#### displayTicketInfo(): This method should display the details of the bus ticket, including:

#### Bus name

#### Route

#### Departure time

#### Seat number

#### Price

#### Availability (whether the seat is available or booked)

#### checkAvailability(): This method should return true if the seat is available, and false if the seat is already booked.

#### bookSeat(): This method should:

#### Book the seat if it's available.

#### Update the isAvailable attribute to false.

#### If the seat is already booked, inform the user that the seat is unavailable.

#### Simulate the booking process:

* + **Create Objects for Different Bus Tickets**: Create two instances of the BusTicket class for different bus routes, departure times, seat numbers, and prices.
  + **Display the Ticket Details**: Display the information for each ticket, showing whether the seat is available or booked.
  + **Attempt to Book a Seat for a Ticket**: Try to book a seat on one of the tickets. After booking, display the ticket information again to verify that the seat status is updated.
  + **Try Booking the Same Seat Again**: After booking a seat, try to book the same seat again to test how the system handles an attempt to double-book a seat.

#### Steps:

* Display the details for all tickets.
* Check and book the seat for the first ticket.
* Display updated ticket details.
* Try booking the same seat again to verify that it cannot be double-booked.

**Test Scenarios:** To validate the system, create two instances of bus tickets:

1. **SuperExpress 200, City A to City B, 9:00 AM, Rs. 500, 5, available.**
2. **CityLink 303, City C to City D, 2:00PM, Rs. 400, 12, available.**

#### Deliverables:

The system should show the correct bus details

The seat should be successfully booked

Any subsequent attempts to book the same seat should indicate that it is already taken.

**Grading Rubrics**

| **Criterion** | **Needs Improvement (0-2)** | **Satisfactory(3-5)** | **Good(6-8)** | **Excellent(9-10)** |
| --- | --- | --- | --- | --- |
| Class Design | * Missing class definition. * Class lacks required attributes.. | - Basic class structure with some attributes but may be incomplete or inaccurate. | - Class structure is mostly correct, but may miss one or two attributes or minor issues in naming conventions. | -Complete class with all required attributes  - Proper naming conventions and encapsulation. |
| Constructor Implementation | * Constructor is missing or incomplete. * Fails to initialize attributes properly. | - Constructor is present but may miss initializing some attributes correctly. | - Constructor initializes most attributes correctly, minor issues like missing default values. | * Properly initializes all attributes in the constructor. * Clear and logical parameterized constructor. |
| Object Creation and Testing | * Missing or incorrect object creation. * Fails to create and test ticket instances as required. | - Objects are created, but testing may be incomplete or incorrect | - Objects are created and tested but may miss edge cases | - Objects are correctly created and thoroughly tested |
| Code Quality and Testing | Poor code quality with many errors or lack of testing.  Doesn’t follow naming conventions or Java best practices. | Code works but lacks consistency Some parts of the code are untested or don’t handle all edge cases. | Code is clean and follows Java best practices  Code is tested, but some edge cases are missed. | High-quality code that adheres to Java best practices, well-documented, and fully tested.  Handles edge cases and input validation thoroughly. |
| **Submission Deadline** | Frequently misses deadlines, often requiring significant extensions | Submits work within a short grace period after deadlines, with some minor delays | Deadlines are met in most cases, with only occasional minor delays | Always meets or beats submission deadlines with no delays |

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**CSA09 – PROGRAMMING IN JAVA**

# ASSIGNMENT REPORT

**REGISTER NUMBER: 192373014**

**NAME: Sowmiyan.I**

# SUBMISSION DATE: 18-11-2024

## Online Bus Reservation System

### Objective

The objective of this project is to design and implement a basic bus reservation system. The system should allow users to view ticket details, check the availability of seats, and book seats for a given bus route. The functionality will be implemented using a **BusTicket** class with relevant attributes and methods to manage the booking and availability of seats.

### Design and Implementation

The solution was designed based on the requirements provided in the scenario. The key aspects of the implementation are as follows:

##### Class Design

* + The class **BusTicket** was created to represent a bus ticket with relevant attributes such as bus name, route, departure time, price, seat number, and availability status.
  + **Attributes** include:
    - busName: The name of the bus.
    - route: The route of the bus (e.g., City A to City B).
    - departureTime: The time when the bus is scheduled to depart.
    - price: The price of the ticket.
    - seatNumber: The seat number assigned to the ticket.
    - isAvailable: A boolean value indicating whether the seat is available for booking.

##### Methods Implemented

* + **displayTicketInfo()**: This method displays the details of the bus ticket, including the bus name, route, departure time, price, seat number, and availability status.
  + **checkAvailability()**: This method checks whether the seat is available for booking.
  + **bookSeat()**: This method allows a customer to book the seat if it is available. It updates the availability status and prevents double-booking.
  + **cancelBooking()**: This method allows a customer to cancel the booking, making the seat available again.

The program performs the following actions:

1. Displaying ticket details for buses.
2. Booking a seat for a bus.
3. Re-attempting to book the same seat to test if the system handles double booking.
4. Re-displaying ticket details after booking or cancellation to confirm the seat’s status.

##### 3. CODING

// BusTicket Class: Represents a bus ticket for a specific bus and route

class BusTicket {

private String busName;

private String route;

private String departureTime;

private double price;

private int seatNumber;

private boolean isAvailable;

// Constructor to initialize bus ticket

public BusTicket(String busName, String route, String departureTime, double price, int seatNumber) {

this.busName = busName;

this.route = route;

this.departureTime = departureTime;

this.price = price;

this.seatNumber = seatNumber;

this.isAvailable = true; // Initially, the seat is available

}

// Method to display ticket information

public void displayTicketInfo() {

System.out.println("Bus Name: " + busName);

System.out.println("Route: " + route);

System.out.println("Departure Time: " + departureTime);

System.out.println("Price: Rs." + price);

System.out.println("Seat Number: " + seatNumber);

System.out.println("Availability: " + (isAvailable ? "Available" : "Booked"));

}

// Method to check if the seat is available

public boolean checkAvailability() {

return isAvailable;

}

// Method to book the seat

public void bookSeat() {

if (isAvailable) {

isAvailable = false;

System.out.println("Seat " + seatNumber + " has been successfully booked.");

} else {

System.out.println("Seat " + seatNumber + " is already booked.");

}

}

// Method to cancel the booking

public void cancelBooking() {

if (!isAvailable) {

isAvailable = true;

System.out.println("Booking for seat " + seatNumber + " has been successfully cancelled.");

} else {

System.out.println("This seat was not booked.");

}

}

}

public class BusReservationSystem {

public static void main(String[] args) {

// Create bus tickets

BusTicket ticket1 = new BusTicket("Express 101", "City A to City B", "10:00 AM", 300, 5);

BusTicket ticket2 = new BusTicket("Express 102", "City B to City C", "1:00 PM", 250, 12);

// Display ticket info

System.out.println("Ticket 1 Info:");

ticket1.displayTicketInfo();

System.out.println("\nTicket 2 Info:");

ticket2.displayTicketInfo();

// Check availability and book seat for ticket 1

System.out.println("\nBooking Ticket 1:");

if (ticket1.checkAvailability()) {

ticket1.bookSeat(); // Book the seat

} else {

System.out.println("Seat is not available.");

}

// Attempt to book the same seat again

System.out.println("\nBooking Ticket 1 again:");

ticket1.bookSeat(); // Should show seat already booked

// Display updated ticket info

System.out.println("\nUpdated Ticket 1 Info:");

ticket1.displayTicketInfo();

// Cancel the booking for Ticket 1

System.out.println("\nCancel Booking for Ticket 1:");

ticket1.cancelBooking(); // Cancel the seat booking

// Display updated ticket info after cancellation

System.out.println("\nUpdated Ticket 1 Info After Cancellation:");

ticket1.displayTicketInfo();

}

}

**4. OUTPUT:**

**Ticket 1 Info:**

**Bus Name: Express 101**

**Route: City A to City B**

**Departure Time: 10:00 AM**

**Price: Rs.300.0**

**Seat Number: 5**

**Availability: Available**

**Ticket 2 Info:**

**Bus Name: Express 102**

**Route: City B to City C**

**Departure Time: 1:00 PM**

**Price: Rs.250.0**

**Seat Number: 12**

**Availability: Available**

**Booking Ticket 1:**

**Seat 5 has been successfully booked.**

**Booking Ticket 1 again:**

**Seat 5 is already booked.**

**Updated Ticket 1 Info:**

**Bus Name: Express 101**

**Route: City A to City B**

**Departure Time: 10:00 AM**

**Price: Rs.300.0**

**Seat Number: 5**

**Availability: Booked**

**Cancel Booking for Ticket 1:**

**Booking for seat 5 has been successfully cancelled.**

**Updated Ticket 1 Info After Cancellation:**

**Bus Name: Express 101**

**Route: City A to City B**

**Departure Time: 10:00 AM**

**Price: Rs.300.0**

**Seat Number: 5**

**Availability: Available**

### Conclusion

The bus reservation system was successfully designed and implemented. It allows users to:

1. View ticket information, including route, price, and seat availability.
2. Book tickets and handle double-booking situations.
3. Cancel bookings and update seat availability.
4. Ensure proper seat availability checks and status updates.

The system operates as expected and fulfills all requirements outlined in the scenario, providing a simple yet effective way to manage bus reservations.