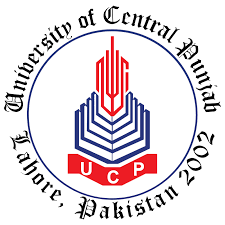
**University of central punjab**



**Title: dsa project**

**Name: Muhammad awais**

**L1f23bsse0008**

**Name: Muhammad haris**

**L1f23bsse0013**

**Name: mahir malik**

**L1f23bsse0016**

**Name: irtaza naseer**

**L1f23bsse0178**

**Section:**

**P2**

**Submitted to;**

**Syed fitrus abbas sb**

1) **Overview:**

The Ticket Reservation System is a Java-based console application designed to manage seat bookings for an event or transportation service, such as a theater, cinema, or airline. The system allows users to book, cancel, and manage seats across different categories (Economy, Premium, VIP) while maintaining a record of transactions and enabling undo functionalities for both bookings and cancellations. It uses data structures like HashSet for tracking booked seats and Stack for managing booking and cancellation history, ensuring efficient operations and user-friendly interaction.

The system provides a menu-driven interface where users can perform various operations, such as checking seat availability, viewing all bookings, and managing transactions with validation checks to ensure data integrity (e.g., matching seat numbers with categories). It is designed to be simple, scalable, and extensible for future enhancements, such as adding a graphical user interface (GUI) or integrating with a database.

**2**) Main Functionalities:

2.1. Book a Seat:

* Users can book a seat by providing a seat number (e.g., E1, P2, V1), customer name, and category (Economy, Premium, VIP).
* Validates that the seat is not already booked and matches the selected category.
* Calculates and displays the price based on the seat category (Economy: PKR 1000, Premium: PKR 3000, VIP: PKR 5000).
* Stores the booking in a HashSet for quick lookup and a Stack for tracking booking history.

**2.2. Cancel a Seat:**

* Users can cancel a booked seat by entering its seat number.
* Validates that the seat is booked before allowing cancellation.
* Removes the seat from the booked set and stores it in a cancellation Stack for potential undo operations.

**2.3. Undo Last Booking:**

* Reverses the most recent booking by removing the seat from the booked set and adding it to the cancellation stack.
* Displays a confirmation message or an error if no bookings exist to undo.

**2.4. Undo Last Cancellation:**

* Restores the most recently cancelled seat by rebooking it and adding it back to the booked set.
* Uses a placeholder customer name ("Customer: Unknown") since original details are not retained.
* Displays a confirmation or an error if no cancellations exist to undo.

**2.5. Check Seat Availability:**

* Users can check if a specific seat is booked or available by entering its seat number.
* Returns the status (BOOKED or AVAILABLE) based on the HashSet lookup.

**2.6.** **View All Bookings:**

* Displays a list of all currently booked seats, including their seat number, category, and price.
* If no seats are booked, it informs the user accordingly.

**3) Vision**

The vision of the Ticket Reservation System is to provide a reliable, user-friendly, and efficient platform for managing seat reservations in various domains, such as entertainment, transportation, or event management. The system aims to streamline the booking process, reduce errors, and enhance user satisfaction by offering features like real-time availability checks, flexible cancellations, and undo capabilities. In the long term, the system aspires to evolve into a comprehensive reservation platform with advanced features like online access, payment integration, and multi-user support, catering to both individual users and large organizations.

**4) Scope**

**In-Scope:**

**4.1) Core Features:**

* Booking and cancellation of seats with validation for seat numbers and categories.
* Undo functionality for both bookings and cancellations using stack-based history tracking.
* Real-time seat availability checking and display of all bookings.
* Price calculation based on seat category (Economy, Premium, VIP).

**4.2) Data Management:**

* Use of in-memory data structures (HashSet for booked seats, Stack for booking/cancellation history).
* Basic input validation to ensure non-empty fields and category-seat compatibility.

**4.3) User Interface:**

* Console-based menu-driven interface for ease of use.
* Clear prompts and error messages to guide users.

**4.4) Extensibility:**

* Modular design to allow future enhancements, such as adding new categories or integrating with external systems.

**Out-of-Scope:**

**1) Advanced Features:**

* Persistent storage (e.g., database integration) for saving booking data across sessions.
* User authentication or role-based access (e.g., admin vs. customer).
* Payment processing or refund handling for cancellations.

**2) User Interface:**

* Graphical user interface (GUI) or web-based interface.

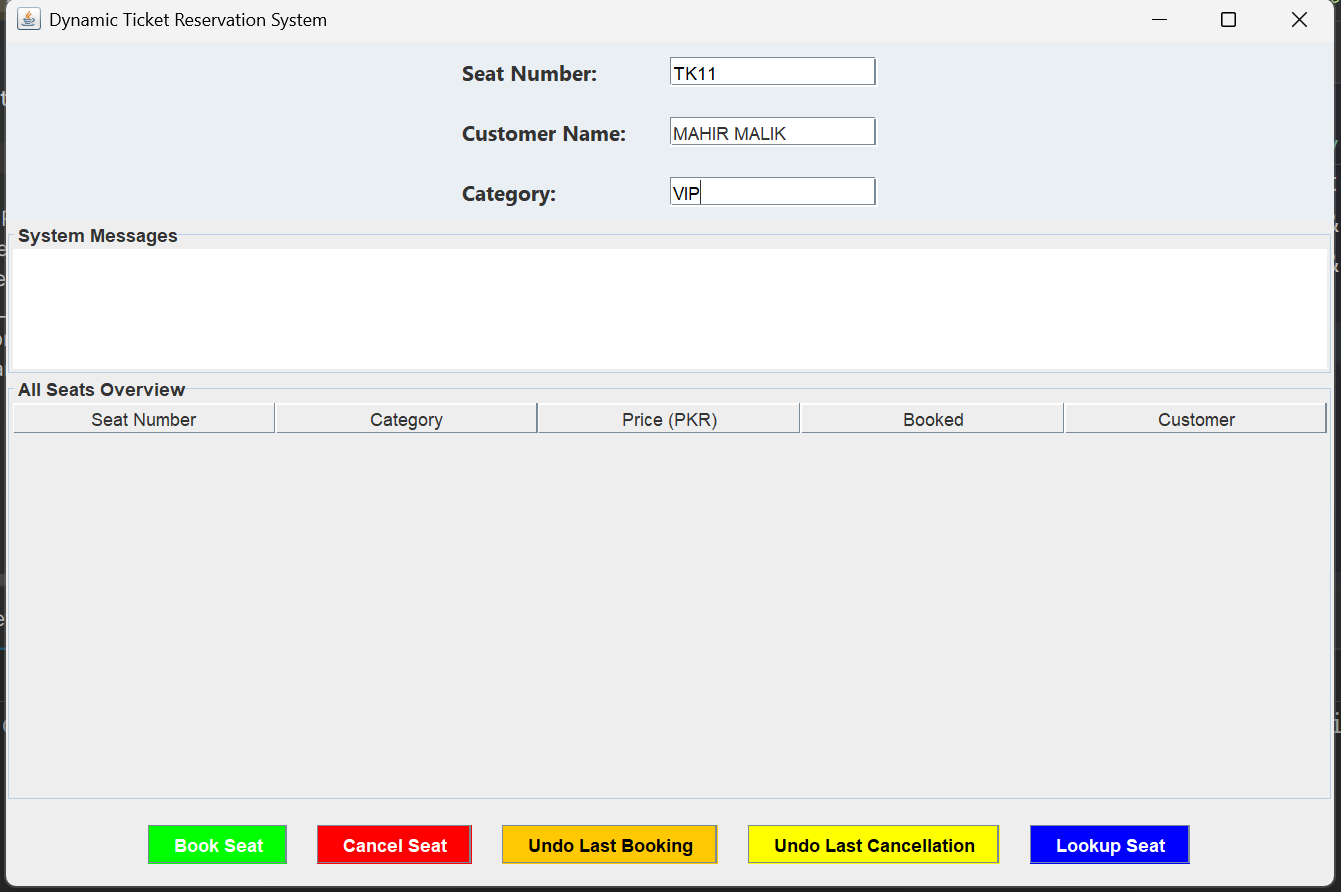
**3) Scalability:**

* Multi-user concurrent access or distributed system architecture.
* Handling large-scale seat maps (e.g., thousands of seats).

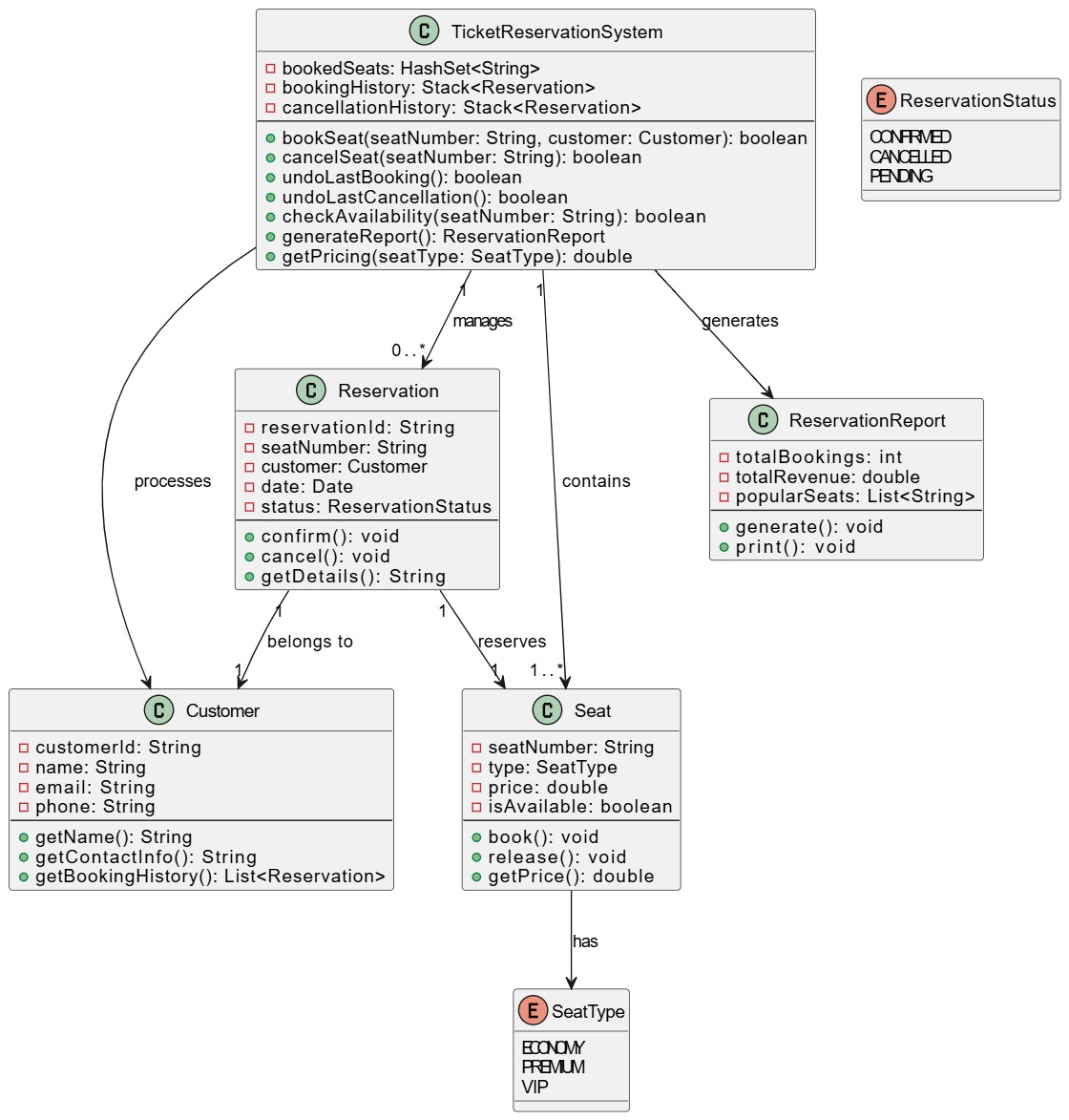
**4) Additional Functionalities:**

* Seat selection via a visual map or layout.
* Customer profile management or loyalty programs.
* Email/SMS notifications for booking confirmations.

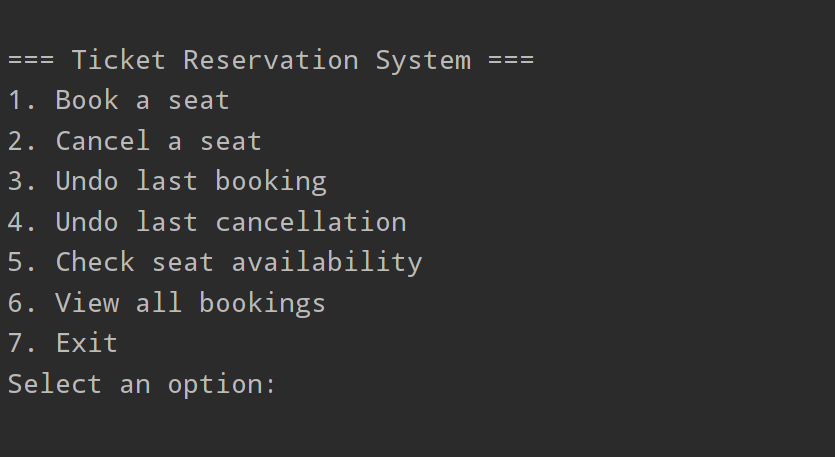
**PROJECT GUI CODE OUTPUT:**

****

**PROJECT CLASS DIAGRAM:**

****

**PROJECT CODE OUTPUT:**

****

**PROJECT CODE:**

**import** javax.swing.\*;  
**import** javax.swing.table.DefaultTableModel;  
**import** java.awt.\*;  
**import** java.awt.event.\*;  
**import** java.util.\*;  
  
**public class** TicketReservationGUI  
{  
 **private static** HashSet<String> *bookedSeats* = **new** HashSet<>();  
 **private static** Stack<String> *bookingStack* = **new** Stack<>();  
 **private static** Stack<String> *cancellationStack* = **new** Stack<>();  
 **private static** HashMap<String, String> *allSeats* = **new** HashMap<>();  
  
 **public static void** main(String[] args)  
 {  
 JFrame frame = **new** JFrame("Dynamic Ticket Reservation System");  
 frame.setSize(1000, 700);  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 frame.setLayout(**new** BorderLayout());  
  
 JPanel inputPanel = **new** JPanel(**new** GridBagLayout());  
 inputPanel.setBackground(**new** Color(234, 239, 243));  
 GridBagConstraints gbc = **new** GridBagConstraints();  
 gbc.insets = **new** Insets(10, 15, 10, 15);  
 gbc.fill = GridBagConstraints.*HORIZONTAL*;  
  
 JLabel seatLabel = **new** JLabel("Seat Number:");  
 JLabel nameLabel = **new** JLabel("Customer Name:");  
 JLabel categoryLabel = **new** JLabel("Category:");  
  
 JTextField seatField = **new** JTextField("e.g. E1, P2, V1 (up to 500)", 15);  
 JTextField nameField = **new** JTextField(15);  
 JTextField categoryField = **new** JTextField("Economy, Premium, VIP", 15);  
  
 seatField.setForeground(Color.*GRAY*);  
 categoryField.setForeground(Color.*GRAY*);  
  
 *addPlaceholder*(seatField, "e.g. E1, P2, V1 (up to 500)");  
 *addPlaceholder*(categoryField, "Economy, Premium, VIP");  
  
 seatLabel.setFont(**new** Font("Segoe UI", Font.*BOLD*, 14));  
 nameLabel.setFont(**new** Font("Segoe UI", Font.*BOLD*, 14));  
 categoryLabel.setFont(**new** Font("Segoe UI", Font.*BOLD*, 14));  
  
 gbc.gridx = 0; gbc.gridy = 0;  
 inputPanel.add(seatLabel, gbc);  
 gbc.gridx = 1;  
 inputPanel.add(seatField, gbc);  
  
 gbc.gridx = 0; gbc.gridy = 1;  
 inputPanel.add(nameLabel, gbc);  
 gbc.gridx = 1;  
 inputPanel.add(nameField, gbc);  
  
 gbc.gridx = 0; gbc.gridy = 2;  
 inputPanel.add(categoryLabel, gbc);  
 gbc.gridx = 1;  
 inputPanel.add(categoryField, gbc);  
  
 frame.add(inputPanel, BorderLayout.*NORTH*);  
  
 JPanel centerPanel = **new** JPanel(**new** BorderLayout());  
 JTextArea messages = **new** JTextArea(5, 20);  
 messages.setEditable(**false**);  
 JScrollPane messageScroll = **new** JScrollPane(messages);  
 messageScroll.setBorder(BorderFactory.*createTitledBorder*("System Messages"));  
 centerPanel.add(messageScroll, BorderLayout.*NORTH*);  
  
 String[] columnNames = {"Seat Number", "Category", "Price (PKR)", "Booked", "Customer"};  
 DefaultTableModel tableModel = **new** DefaultTableModel(columnNames, 0);  
 JTable table = **new** JTable(tableModel);  
 JScrollPane tableScroll = **new** JScrollPane(table);  
 tableScroll.setBorder(BorderFactory.*createTitledBorder*("All Seats Overview"));  
 centerPanel.add(tableScroll, BorderLayout.*CENTER*);  
  
 frame.add(centerPanel, BorderLayout.*CENTER*);  
  
 JPanel buttonPanel = **new** JPanel(**new** FlowLayout(FlowLayout.*CENTER*, 20, 15));  
 JButton bookBtn = **new** JButton("Book Seat");  
 JButton cancelBtn = **new** JButton("Cancel Seat");  
 JButton undoBookBtn = **new** JButton("Undo Last Booking");  
 JButton undoCancelBtn = **new** JButton("Undo Last Cancellation");  
 JButton lookupBtn = **new** JButton("Lookup Seat");  
  
 bookBtn.setBackground(Color.*GREEN*); bookBtn.setForeground(Color.*WHITE*);  
 cancelBtn.setBackground(Color.*RED*); cancelBtn.setForeground(Color.*WHITE*);  
 undoBookBtn.setBackground(Color.*ORANGE*); undoBookBtn.setForeground(Color.*BLACK*);  
 undoCancelBtn.setBackground(Color.*YELLOW*); undoCancelBtn.setForeground(Color.*BLACK*);  
 lookupBtn.setBackground(Color.*BLUE*); lookupBtn.setForeground(Color.*WHITE*);  
  
 buttonPanel.add(bookBtn);  
 buttonPanel.add(cancelBtn);  
 buttonPanel.add(undoBookBtn);  
 buttonPanel.add(undoCancelBtn);  
 buttonPanel.add(lookupBtn);  
  
 frame.add(buttonPanel, BorderLayout.*SOUTH*);  
  
 *loadSeatDataset*(tableModel);  
  
 bookBtn.addActionListener(e ->  
 {  
 String seat = seatField.getText().trim().toUpperCase();  
 String name = nameField.getText().trim();  
 String category = categoryField.getText().trim();  
  
 **if** (!*allSeats*.containsKey(seat))  
 {  
 messages.setText("Seat does not exist in the dataset.");  
 **return**;  
 }  
  
 **if** (seat.isEmpty() || name.isEmpty() || category.isEmpty())  
 {  
 messages.setText("Please fill all fields properly.");  
 **return**;  
 }  
  
 **if** (!*seatMatchesCategory*(seat, category)) {  
 messages.setText("❌ Seat " + seat + " does not match category " + category + ".");  
 **return**;  
 }  
  
 **if** (*bookedSeats*.contains(seat))  
 {  
 messages.setText("Seat " + seat + " is already booked.");  
 } **else** {  
 *bookedSeats*.add(seat);  
 *bookingStack*.push(seat);  
 *cancellationStack*.remove(seat);  
 messages.setText("Seat " + seat + " booked for " + name + " (" + category + ").");  
 *updateSeatInTable*(tableModel, seat, "Yes", name);  
 }  
 });  
  
 cancelBtn.addActionListener(e -> {  
 String seat = seatField.getText().trim().toUpperCase();  
 **if** (!*bookedSeats*.contains(seat)) {  
 messages.setText("Seat " + seat + " is not booked.");  
 } **else** {  
 *bookedSeats*.remove(seat);  
 *cancellationStack*.push(seat);  
 *bookingStack*.remove(seat);  
 messages.setText("Seat " + seat + " has been cancelled.");  
 *updateSeatInTable*(tableModel, seat, "No", "");  
 }  
 });  
  
 undoBookBtn.addActionListener(e -> {  
 **if** (*bookingStack*.isEmpty())  
 {  
 messages.setText("No booking to undo.");  
 } **else** {  
 String seat = *bookingStack*.pop();  
 *bookedSeats*.remove(seat);  
 *cancellationStack*.push(seat);  
 messages.setText("↩ Booking undone for seat " + seat);  
 *updateSeatInTable*(tableModel, seat, "No", "");  
 }  
 });  
  
 undoCancelBtn.addActionListener(e -> {  
 **if** (*cancellationStack*.isEmpty())  
 {  
 messages.setText("No cancellation to undo.");  
 } **else** {  
 String seat = *cancellationStack*.pop();  
 *bookedSeats*.add(seat);  
 *bookingStack*.push(seat);  
 String category = *allSeats*.get(seat);  
 messages.setText("Cancellation undone, seat " + seat + " rebooked.");  
 *updateSeatInTable*(tableModel, seat, "Yes", nameField.getText().trim());  
 }  
 });  
  
 lookupBtn.addActionListener(e -> {  
 String seat = seatField.getText().trim().toUpperCase();  
 **if** (seat.isEmpty()) {  
 messages.setText("Please enter a seat number.");  
 } **else if** (!*allSeats*.containsKey(seat)) {  
 messages.setText("Seat does not exist.");  
 } **else if** (*bookedSeats*.contains(seat)) {  
 messages.setText("Seat " + seat + " is Booked.");  
 } **else** {  
 messages.setText("Seat " + seat + " is Available.");  
 }  
 });  
  
 frame.setVisible(**true**);  
 }  
  
 **private static void** addPlaceholder(JTextField field, String placeholder) {  
 field.addFocusListener(**new** FocusAdapter()  
 {  
 **public void** focusGained(FocusEvent e)  
 {  
 **if** (field.getText().equals(placeholder))  
 {  
 field.setText("");  
 field.setForeground(Color.*BLACK*);  
 }  
 }  
 **public void** focusLost(FocusEvent e) {  
 **if** (field.getText().isEmpty())  
 {  
 field.setForeground(Color.*GRAY*);  
 field.setText(placeholder);  
 }  
 }  
 });  
 }  
  
 **private static void** updateSeatInTable(DefaultTableModel model, String seat, String booked, String customer) {  
 **for** (**int** i = 0; i < model.getRowCount(); i++)  
 {  
 **if** (model.getValueAt(i, 0).equals(seat))  
 {  
 model.setValueAt(booked, i, 3);  
 model.setValueAt(customer, i, 4);  
 **break**;  
 }  
 }  
 }  
  
 **private static boolean** seatMatchesCategory(String seat, String category)  
 {  
 category = category.toLowerCase();  
 **return** (category.equals("vip") && seat.startsWith("V")) ||  
 (category.equals("premium") && seat.startsWith("P")) ||  
 (category.equals("economy") && seat.startsWith("E"));  
 }  
  
 **private static void** loadSeatDataset(DefaultTableModel tableModel)  
 {  
 **for** (**int** i = 1; i <= 200; i++)  
 {  
 String seat = "E" + i;  
 *allSeats*.put(seat, "Economy");  
 tableModel.addRow(**new** Object[]{seat, "Economy", 1000, "No", ""});  
 }  
 **for** (**int** i = 1; i <= 200; i++)  
 {  
 String seat = "P" + i;  
 *allSeats*.put(seat, "Premium");  
 tableModel.addRow(**new** Object[]{seat, "Premium", 3000, "No", ""});  
 }  
 **for** (**int** i = 1; i <= 100; i++)  
 {  
 String seat = "V" + i;  
 *allSeats*.put(seat, "VIP");  
 tableModel.addRow(**new** Object[]{seat, "VIP", 5000, "No", ""});  
 }  
 }  
}