VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI 590018

Project Report on

"Railway Ticket Booking"

By

Pooja Basavaraj Kuruvinakoppa(1BM24CS203) Poorvi T(1BM24CS204)
Pradnya Bhosale (1BM24CS207) Pragathi M (1BM24CS208)

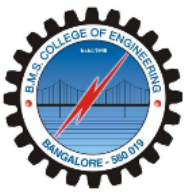
Under the Guidance of

Monisha H M

Assisstant Professor, Department of CSE

BMS College of Engineering

Work carried out at



Department of Computer Science and Engineering
BMS College of Engineering
(Autonomous college under VTU)
P.O. Box No.: 1908, Bull Temple Road, Bangalore-560 019

2025-2026

BMS COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the OOPS with JAVA project titled "**Railway Ticket Booking"** has been carried out by Pooja Basavaraj Kuruvinakoppa (1BM24CS203), Poorvi T (1BM24CS204), Pradnya Bhosale (1BM24CS207), Pragathi M (1BM24CS208) during the academic year 2025-2026.

Signature of the guide

Monisha H M

Assistant Professor,

Department of Computer Science and Engineering

BMS College of Engineering, Bangalore

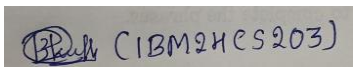
BMS COLLEGE OF ENGINEERING

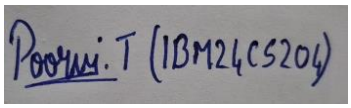
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

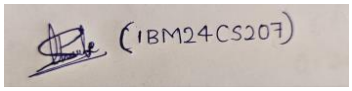
DECLARATION

We, Pooja Basavaraj Kuruvinakoppa (1BM24CS203), Poorvi T (1BM24CS204), Pradnya Bhosale (1BM24CS207), Pragathi M (1BM24CS208), students of 3rd Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this project work entitled "Railway Ticket Booking" has been carried out by us under the guidance of Monisha H M**,** Assisstant Professor, Department of CSE, BMS College of Engineering, Bangalore during the academic semester Sep-Dec 2025. We also declare that to the best of our knowledge and belief, the project reported here is not from part of any other report by any other students.

Signature of the Candidates







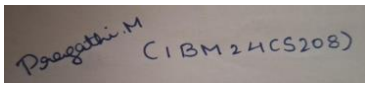


TABLE OF CONTENTS

I. Description Of the Project

1. OBJECTIVE OF THE PROJECT-

The objective of this project is to **design and implement a Railway Ticket Booking System using Java Swing** that simulates real-world railway reservation operations. The system allows users to:

- Select source and destination stations
- Choose different classes of coaches
- View seat availability
- Book tickets with passenger details
- Calculate ticket fare based on age and coach type
- Cancel booked tickets
- View booked tickets in a printable ticket format

The project aims to demonstrate the **effective application of Object-Oriented Programming (OOP) concepts**, GUI development using **Java Swing**, and **thread-safe booking mechanisms**.

2. INTRODUCTION-

Railway reservation systems are widely used real-time applications that require accuracy, efficiency, and proper handling of shared resources such as seat availability. Manual ticket booking systems are error-prone and inefficient, making automation essential.

This project presents a **Java-based Railway Ticket Booking System** with a **Graphical User Interface (GUI)** developed using **Swing and AWT components**. The system supports multiple coaches, dynamic seat allocation, passenger-based ticket generation, and cancellation functionality.

The application is designed to be **user-friendly, modular, and scalable**, making it suitable for understanding how real-world booking systems function while applying core Java programming concepts such as **inheritance, polymorphism, encapsulation, synchronization, collections, and event handling**.

3. DESIGN MODULES-

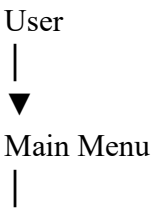
Tools Used-

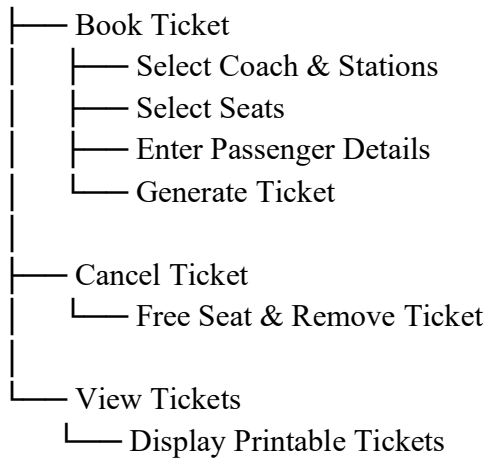
1. **Java Development Kit (JDK)**
    a. Used for compiling and running the Java program
    b. Provides core Java libraries used in the project
2. **Java Programming Language**
    a. Main language used to implement application logic
    b. Supports Object-Oriented Programming concepts such as inheritance and polymorphism
3. **Integrated Development Environment (IDE)**
    a. Tools such as **IntelliJ IDEA / Eclipse / NetBeans**
    b. Used for writing, debugging, and testing the code
4. **Java Swing & AWT Libraries**
    a. Used to design the graphical user interface
    b. Components like JFrame, JDialog, JButton, JLabel, etc.
5. **Java Collections Framework**
    a. ArrayList, HashMap, LinkedHashMap used for data storage and management
6. **Java Event Handling Mechanism**
    a. ActionListener used to handle button clicks and user interactions
7. **Java Concurrency Utilities**
    a. synchronized keyword used to ensure thread safety during ticket booking
8. **Operating System**
    a. Windows OS (or any OS supporting Java)

MODULES-

1. **Main Menu Module**
    a. Entry point of the application
    b. Provides navigation to booking, cancellation, and ticket viewing
2. **Booking Module**
    a. Coach selection
    b. Station selection
    c. Seat availability display
    d. Seat selection
3. **Passenger Module**
    a. Passenger name and age input
    b. Validation of passenger data
4. **Ticket Module**
    a. Ticket generation
    b. Ticket storage
    c. Fare calculation
5. **Cancellation Module**
    a. Ticket cancellation using seat ID
    b. Seat availability restoration
6. **View Tickets Module**
    a. Display of booked tickets
    b. Printable ticket-style layout

BLOCK DIAGRAM-





4. DETAILED DESCRIPTION OF MODULES-

Main Menu Module

Requirement & Functionality

- Acts as the central control of the application
- Displays options for booking, cancellation, viewing tickets, and resetting data

Java Techniques Used

- Swing components (JFrame, JButton, JLabel)

  JFrame: Provides the main window container for the graphical user interface of the application.

  JButton: Represents a clickable button that triggers specific actions when pressed by the user.

  JLabel: Used to display static text or images within the user interface.
- Event handling using ActionListener

  ActionListener: Handles user-generated events such as button clicks by executing predefined actions.
- Layout managers (BorderLayout, GridLayout)

BorderLayout: Arranges components in five regions—North, South, East, West, and Center—within a container.

GridLayout: Organizes components in a rectangular grid with equal-sized rows and columns.

Ticket Booking Module

Requirement & Functionality

- Allows user to select:
  - Coach type (1AC, 2AC, 3AC, GEN)
  - Source and destination stations
- Displays seat availability dynamically
- Supports booking of multiple seats at once

Java Techniques Used

- Swing components (JDialog, JComboBox, JCheckBox)

  JDialog: Used to create a secondary window for user interaction such as data entry or confirmation dialogs.

  JComboBox: Provides a drop-down list that allows users to select one option from multiple choices.

  JCheckBox: Allows users to select or deselect multiple independent options within the interface.
- Java Collections (Map, List)

  Map: Stores data as key–value pairs, enabling efficient retrieval and update of related information.

  List: Maintains an ordered collection of elements that allows duplicate values.
- Java Streams (stream(), filter(), collect())

  stream(): Converts a collection into a stream to perform functional-style operations on its elements.

  filter(): Selects elements from a stream that satisfy a given condition.

  collect(): Gathers processed stream elements into a collection such as a list or map.

- **Validation using conditional checks**
  Ensures correctness of user input by verifying conditions before processing data.

Passenger Details Module

Requirement & Functionality

- Collects passenger name and age
- Performs validation for empty input and invalid age
- Confirms booking per passenger

Java Techniques Used

- **Modal dialogs (JDialog)**
  A dialog window that blocks interaction with other windows until the user completes the required action.
- **Exception handling (NumberFormatException)**
  Handles runtime errors that occur when converting invalid input into numeric values.
- **Encapsulation of passenger data**
  Groups passenger-related information and behavior within a class to protect and manage data effectively.
- **Input validation logic**
  Verifies user input against defined rules to ensure accuracy and prevent invalid data entry.

Ticket Management Module

Requirement & Functionality

- Generates unique ticket IDs
- Stores ticket details
- Calculates fare based on:
  - Coach type
  - Passenger age (half fare for age ≤ 15)

Java Techniques Used

- **Encapsulation** using Ticket and BaseTicket classes
  Bundles ticket-related data and behavior within classes to ensure controlled access and better data management.
- **Java Date & Time (Date, SimpleDateFormat)**
  Used to store and format ticket booking date and time in a readable form.
- **Use of HashMap and ArrayList**
  Enables efficient storage, retrieval, and management of dynamic ticket and seat data.

OOP Design (Inheritance & Polymorphism)

Inheritance

- Ticket class extends abstract class BaseTicket

abstract class BaseTicket
class Ticket extends BaseTicket

Polymorphism

- Abstract method getTicketType() implemented in subclass
- Method called dynamically while displaying ticket details

Encapsulation

- Ticket-related data bundled inside classes
- Accessed through controlled methods

Ticket Cancellation Module

Requirement & Functionality

- Cancels ticket using seat ID
- Frees the seat for future booking

- Removes ticket from system

Java Techniques Used

- **Swing dialogs**
  Used to interact with the user through pop-up windows for input, messages, or confirmations.
- **Map lookups**
  Retrieve specific values efficiently using a unique key from a map collection.
- **Confirmation dialogs**
  Prompt the user to confirm or cancel an action before it is executed.
- **Collection modification**
  Updates data stored in collections by adding, removing, or changing elements dynamically.

Ticket Viewing (Printable Ticket) Module

Requirement & Functionality

- Displays all booked tickets
- Printable ticket-style layout
- Shows all ticket details clearly

Java Techniques Used

- **Custom painting using paintComponent()**
  Allows drawing customized graphical content on Swing components by overriding the default painting behavior.
- **Graphics2D, GradientPaint**
  Used to render advanced graphics and smooth color gradients for enhanced visual appearance.
- **Swing layouts (BoxLayout, JScrollPane)**

Used to render advanced graphics and smooth color gradients for enhanced visual appearance.

Synchronization & Thread Safety

Requirement & Functionality

- Prevents inconsistent seat booking
- Ensures data integrity when multiple operations occur

Java Techniques Used

- **Synchronization using synchronized (bookingLock)**:
  Ensures that only one thread can modify booking-related data at a time.
- **Thread-safe access to shared resources**
  Prevents data inconsistency by safely coordinating concurrent access to common data structures.

5. IMPLEMENTATION-

```
import javax.swing.*;
import javax.swing.border.EmptyBorder;
import java.awt.*;
import java.awt.event.*;
import java.text.SimpleDateFormat;
import java.util.*;
import java.util.List;
import java.util.stream.Collectors;

public class RailwaySystem1 {

   // Station list
```

```
    private static final String[] STATIONS = {"Bangalore", "Chennai", "Mumbai", "Hyderabad",
"Delhi", "Kolkata"};

    // Coach configuration
    private static final String COACH_1 = "1AC";
    private static final String COACH_2 = "2AC";
    private static final String COACH_3 = "3AC";
    private static final String COACH_GEN = "GEN";

    private static RailwaySystem1 instance;

    private static final Map<String, Integer> COACH_SEAT_COUNT = new
LinkedHashMap<>();
    private static final Map<String, Integer> COACH_PRICE = new LinkedHashMap<>();

    static {
        COACH_SEAT_COUNT.put(COACH_1, 10);
        COACH_SEAT_COUNT.put(COACH_2, 20);
        COACH_SEAT_COUNT.put(COACH_3, 30);
        COACH_SEAT_COUNT.put(COACH_GEN, 40);

        COACH_PRICE.put(COACH_1, 4000);
        COACH_PRICE.put(COACH_2, 2500);
        COACH_PRICE.put(COACH_3, 1000);
        COACH_PRICE.put(COACH_GEN, 500);
    }

    // Storage
    private final Map<String, Boolean> seatsAvailable = new HashMap<>();
    private final Map<String, Ticket> seatToTicket = new HashMap<>();
    private final List<Ticket> tickets = new ArrayList<>();
    private int ticketCounter = 1;
    private final Object bookingLock = new Object();

    // Main frame
    private JFrame mainFrame;

    public RailwaySystem1() {
        instance = this;
```

```
        initializeSeats();
        SwingUtilities.invokeLater(this::createAndShowGUI);
    }

    private void initializeSeats() {
        for (Map.Entry<String, Integer> e : COACH_SEAT_COUNT.entrySet()) {
            String coach = e.getKey();
            int count = e.getValue();
            for (int i = 1; i <= count; i++) {
                String seatId = (coach.equals(COACH_GEN) ? "GEN" : coach) + "-S" + i;
                seatsAvailable.put(seatId, true);
            }
        }
    }

    private void createAndShowGUI() {
        mainFrame = new JFrame("Railway Booking - Main Menu");
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainFrame.setSize(520, 300);
        mainFrame.setLocationRelativeTo(null);
        mainFrame.setLayout(new BorderLayout(10, 10));

        JLabel title = new JLabel("Railway Ticket Booking System", SwingConstants.CENTER);
        title.setFont(new Font("SansSerif", Font.BOLD, 20));
        title.setBorder(new EmptyBorder(10, 10, 10, 10));
        mainFrame.add(title, BorderLayout.NORTH);

        JPanel center = new JPanel();
        center.setLayout(new GridLayout(2, 2, 12, 12));
        center.setBorder(new EmptyBorder(10, 10, 10, 10));

        JButton bookBtn = new JButton("Book Tickets");
        JButton cancelBtn = new JButton("Cancel Ticket");
        JButton viewBtn = new JButton("View Tickets (Printable)");
        JButton resetBtn = new JButton("Reset All (DEV)");

        bookBtn.setFont(new Font("SansSerif", Font.PLAIN, 16));
        cancelBtn.setFont(new Font("SansSerif", Font.PLAIN, 16));
        viewBtn.setFont(new Font("SansSerif", Font.PLAIN, 16));
```

```
        resetBtn.setFont(new Font("SansSerif", Font.PLAIN, 12));

        center.add(bookBtn);
        center.add(cancelBtn);
        center.add(viewBtn);
        center.add(resetBtn);

        mainFrame.add(center, BorderLayout.CENTER);

        JLabel footer = new JLabel(
                "Coaches: 1AC(10), 2AC(20), 3AC(30), General(40) | Age<=15 => Half Fare",
                SwingConstants.CENTER);
        footer.setBorder(new EmptyBorder(0, 0, 10, 0));
        mainFrame.add(footer, BorderLayout.SOUTH);

        bookBtn.addActionListener(e -> SwingUtilities.invokeLater(this::openBookingWindow));
        cancelBtn.addActionListener(e -> SwingUtilities.invokeLater(this::openCancelWindow));
        viewBtn.addActionListener(e ->
SwingUtilities.invokeLater(this::openViewTicketsWindow));
        resetBtn.addActionListener(e -> {
            int confirm = JOptionPane.showConfirmDialog(mainFrame,
                    "Reset clears ALL bookings. Continue?", "Confirm Reset",
JOptionPane.YES_NO_OPTION);
            if (confirm == JOptionPane.YES_OPTION) {
                synchronized (bookingLock) {
                    seatsAvailable.keySet().forEach(k -> seatsAvailable.put(k, true));
                    seatToTicket.clear();
                    tickets.clear();
                    ticketCounter = 1;
                }
                JOptionPane.showMessageDialog(mainFrame, "All bookings cleared.", "Reset Done",
                        JOptionPane.INFORMATION_MESSAGE);
            }
        });

        mainFrame.setVisible(true);
    }

    // ---------- Booking Window ----------
```

```
    private void openBookingWindow() {
        JDialog bookDialog = new JDialog(mainFrame, "Book Tickets", true);
        bookDialog.setSize(900, 600);
        bookDialog.setLocationRelativeTo(mainFrame);
        bookDialog.setLayout(new BorderLayout(10, 10));

        // Top panel
        JPanel topPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 12, 12));
        topPanel.setBorder(new EmptyBorder(5, 5, 5, 5));
        topPanel.add(new JLabel("Coach:"));
        JComboBox<String> coachCombo = new
JComboBox<>(COACH_SEAT_COUNT.keySet().toArray(new String[0]));
        topPanel.add(coachCombo);

        topPanel.add(new JLabel("From:"));
        JComboBox<String> fromCombo = new JComboBox<>(STATIONS);
        topPanel.add(fromCombo);

        topPanel.add(new JLabel("To:"));
        JComboBox<String> toCombo = new JComboBox<>(STATIONS);
        toCombo.setSelectedIndex(1);
        topPanel.add(toCombo);

        JLabel availLabel = new JLabel("Available seats: 0");
        topPanel.add(availLabel);
        bookDialog.add(topPanel, BorderLayout.NORTH);

        // Center panel
        JPanel centerPanel = new JPanel(new BorderLayout(8, 8));
        centerPanel.setBorder(new EmptyBorder(10, 10, 10, 10));

        JPanel seatsContainer = new JPanel();
        seatsContainer.setLayout(new BorderLayout());
        JScrollPane seatsScroll = new JScrollPane(seatsContainer,
JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
                JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
        centerPanel.add(seatsScroll, BorderLayout.CENTER);

        JPanel rightPanel = new JPanel();
```

```
        rightPanel.setLayout(new BoxLayout(rightPanel, BoxLayout.Y_AXIS));
        rightPanel.setBorder(new EmptyBorder(10, 10, 10, 10));
        rightPanel.setPreferredSize(new Dimension(300, 300));

        JLabel instruction = new JLabel(
                "<html><b>Instructions:</b><br>1) Choose coach.<br>2) Manually select one or
more seats.<br>3) Click 'Proceed' to enter passenger details for each seat.<br>4) Each seat
becomes a separate ticket.</html>");
        instruction.setBorder(new EmptyBorder(0, 0, 10, 0));
        rightPanel.add(instruction);

        JButton proceedBtn = new JButton("Proceed (Enter passenger data & Book)");
        proceedBtn.setAlignmentX(Component.CENTER_ALIGNMENT);
        rightPanel.add(Box.createRigidArea(new Dimension(0, 10)));
        rightPanel.add(proceedBtn);
        rightPanel.add(Box.createRigidArea(new Dimension(0, 12)));

        JLabel priceNote = new JLabel("Prices: 1AC=4000 | 2AC=2500 | 3AC=1000 | GEN=500");
        priceNote.setFont(new Font("SansSerif", Font.PLAIN, 12));
        rightPanel.add(priceNote);
        centerPanel.add(rightPanel, BorderLayout.EAST);
        bookDialog.add(centerPanel, BorderLayout.CENTER);

        // Bottom panel
        JPanel bottom = new JPanel(new FlowLayout(FlowLayout.RIGHT));
        JButton closeBtn = new JButton("Close");
        bottom.add(closeBtn);
        bookDialog.add(bottom, BorderLayout.SOUTH);

        // Build seat checkboxes
        final Map<String, JCheckBox> currentSeatCheckboxes = new LinkedHashMap<>();
        Runnable buildSeatsUI = () -> {
            seatsContainer.removeAll();
            currentSeatCheckboxes.clear();
            String coach = (String) coachCombo.getSelectedItem();
            if (coach == null)
                return;
            int seatCount = COACH_SEAT_COUNT.getOrDefault(coach, 0);
            JPanel grid = new JPanel(new GridLayout(Math.max(3, (seatCount + 4) / 5), 5, 8, 8));
```

```
            grid.setBorder(new EmptyBorder(8, 8, 8, 8));
            int available = 0;
            for (int i = 1; i <= seatCount; i++) {
                String seatId = (coach.equals(COACH_GEN) ? "GEN" : coach) + "-S" + i;
                JCheckBox cb = new JCheckBox(seatId);
                boolean avail = seatsAvailable.getOrDefault(seatId, true);
                cb.setEnabled(avail);
                if (!avail)
                    cb.setText(seatId + " (Booked)");
                if (avail)
                    available++;
                currentSeatCheckboxes.put(seatId, cb);
                grid.add(cb);
            }
            seatsContainer.add(grid, BorderLayout.CENTER);
            seatsContainer.revalidate();
            seatsContainer.repaint();
            availLabel.setText("Available seats: " + available);
        };

        buildSeatsUI.run();
        coachCombo.addActionListener(e -> buildSeatsUI.run());

        proceedBtn.addActionListener(e -> {
            String coach = (String) coachCombo.getSelectedItem();
            String from = (String) fromCombo.getSelectedItem();
            String to = (String) toCombo.getSelectedItem();
            if (from == null || to == null || from.equals(to)) {
                JOptionPane.showMessageDialog(bookDialog, "Please select different 'From' and 'To'
stations.",
                        "Invalid Stations", JOptionPane.ERROR_MESSAGE);
                return;
            }
            List<String> selectedSeats = currentSeatCheckboxes.entrySet().stream()
                    .filter(entry -> entry.getValue().isSelected() && entry.getValue().isEnabled())
                    .map(Map.Entry::getKey).collect(Collectors.toList());
            if (selectedSeats.isEmpty()) {
                JOptionPane.showMessageDialog(bookDialog, "No seats selected. Please select seat(s)
to book.",
```

```
                "No Selection", JOptionPane.WARNING_MESSAGE);
            return;
        }
        double totalCost = 0;
        int bookedCount = 0;
        for (String seatId : selectedSeats) {
            PassengerDialog pd = new PassengerDialog(bookDialog, seatId);
            pd.setVisible(true);
            if (!pd.isConfirmed())
                continue;
            String pname = pd.getPassengerName();
            int age = pd.getPassengerAge();
            int basePrice = COACH_PRICE.getOrDefault(coach, 500);
            double price = age <= 15 ? basePrice / 2.0 : basePrice;
            String ticketId = generateTicketId();
            Ticket t = new Ticket(ticketId, pname, age, seatId, coach, from, to, price, new Date());
            synchronized (bookingLock) {
                tickets.add(t);
                seatToTicket.put(seatId, t);
                seatsAvailable.put(seatId, false);
            }

            JCheckBox cb = currentSeatCheckboxes.get(seatId);
            if (cb != null) {
                cb.setEnabled(false);
                cb.setSelected(false);
                cb.setText(seatId + " (Booked)");
            }
            totalCost += price;
            bookedCount++;
        }
        buildSeatsUI.run();
        if (bookedCount > 0) {
            JOptionPane.showMessageDialog(bookDialog, bookedCount + " ticket(s) booked.
Total: Rs. "
                    + String.format("%.2f", totalCost), "Booking Successful",
JOptionPane.INFORMATION_MESSAGE);
        } else {
            JOptionPane.showMessageDialog(bookDialog,
```

```
                "No tickets were booked (maybe you cancelled some passenger dialogs).",
                "Booking Cancelled", JOptionPane.INFORMATION_MESSAGE);
        }
    });

    closeBtn.addActionListener(e -> bookDialog.dispose());
    bookDialog.setVisible(true);
}

// ---------- Passenger Dialog ----------
private static class PassengerDialog extends JDialog {
    private boolean confirmed = false;
    private final JTextField nameField;
    private final JTextField ageField;

    PassengerDialog(Window parent, String seatId) {
        super(parent, "Passenger for " + seatId, ModalityType.APPLICATION_MODAL);
        setSize(360, 220);
        setLocationRelativeTo(parent);
        setLayout(null);

        JLabel info = new JLabel("Enter passenger details for " + seatId);
        info.setBounds(16, 10, 320, 24);
        add(info);

        JLabel nameLbl = new JLabel("Name:");
        nameLbl.setBounds(16, 48, 80, 24);
        add(nameLbl);
        nameField = new JTextField();
        nameField.setBounds(100, 48, 220, 24);
        add(nameField);

        JLabel ageLbl = new JLabel("Age:");
        ageLbl.setBounds(16, 88, 80, 24);
        add(ageLbl);
        ageField = new JTextField();
        ageField.setBounds(100, 88, 80, 24);
        add(ageField);
```

```
        JButton ok = new JButton("OK");
        ok.setBounds(60, 130, 100, 30);
        add(ok);
        JButton cancel = new JButton("Cancel");
        cancel.setBounds(190, 130, 100, 30);
        add(cancel);

        ok.addActionListener(e -> {
            String name = nameField.getText().trim();
            String ageText = ageField.getText().trim();
            if (name.isEmpty() || ageText.isEmpty()) {
                JOptionPane.showMessageDialog(this, "Please fill both name and age.", "Input
Required",
                        JOptionPane.WARNING_MESSAGE);
                return;
            }
            int age;
            try {
                age = Integer.parseInt(ageText);
                if (age < 0 || age > 120) {
                    JOptionPane.showMessageDialog(this, "Enter a valid age (0-120).", "Invalid
Age",
                            JOptionPane.WARNING_MESSAGE);
                    return;
                }
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(this, "Age must be a number.", "Invalid Input",
                        JOptionPane.WARNING_MESSAGE);
                return;
            }
            confirmed = true;
            setVisible(false);
        });

        cancel.addActionListener(e -> {
            confirmed = false;
            setVisible(false);
        });
    }
```

```
    boolean isConfirmed() {
        return confirmed;
    }

    String getPassengerName() {
        return nameField.getText().trim();
    }

    int getPassengerAge() {
        try {
            return Integer.parseInt(ageField.getText().trim());
        } catch (Exception e) {
            return 0;
        }
    }
}

// ---------- Cancel Ticket ----------
private void openCancelWindow() {
    JDialog cancelDialog = new JDialog(mainFrame, "Cancel Ticket", true);
    cancelDialog.setSize(420, 220);
    cancelDialog.setLocationRelativeTo(mainFrame);
    cancelDialog.setLayout(new BorderLayout(8, 8));
    cancelDialog.setResizable(false);

    JPanel main = new JPanel();
    main.setLayout(new BoxLayout(main, BoxLayout.Y_AXIS));
    main.setBorder(new EmptyBorder(12, 12, 12, 12));

    JLabel lbl = new JLabel("Enter seat ID to cancel (e.g., 1AC-S3 or GEN-S12):");
    lbl.setAlignmentX(Component.LEFT_ALIGNMENT);
    main.add(lbl);

    JTextField seatField = new JTextField();
    seatField.setMaximumSize(new Dimension(Integer.MAX_VALUE, 28));
    seatField.setAlignmentX(Component.LEFT_ALIGNMENT);
    main.add(Box.createRigidArea(new Dimension(0, 8)));
    main.add(seatField);
```

```
        main.add(Box.createRigidArea(new Dimension(0, 12)));

        JButton cancelBtn = new JButton("Cancel Ticket");
        cancelBtn.setAlignmentX(Component.CENTER_ALIGNMENT);
        main.add(cancelBtn);
        main.add(Box.createRigidArea(new Dimension(0, 8)));

        JLabel note = new JLabel("<html><i>Note:</i> Seat IDs are coach-prefixed. Example:
2AC-S5, GEN-S12. Use exact ID.</html>");
        note.setAlignmentX(Component.LEFT_ALIGNMENT);
        main.add(note);

        cancelDialog.add(main, BorderLayout.CENTER);

        cancelBtn.addActionListener(e -> {
            String seatInput = seatField.getText().trim().toUpperCase();
            if (seatInput.isEmpty()) {
                JOptionPane.showMessageDialog(cancelDialog, "Please enter a seat ID.", "Input
Required",
                        JOptionPane.WARNING_MESSAGE);
                return;
            }
            if (!seatsAvailable.containsKey(seatInput)) {
                JOptionPane.showMessageDialog(cancelDialog, "Seat ID not recognized: " +
seatInput, "Invalid Seat",
                        JOptionPane.ERROR_MESSAGE);
                return;
            }
            Ticket t = seatToTicket.get(seatInput);
            if (t == null) {
                JOptionPane.showMessageDialog(cancelDialog, "No booking found for " + seatInput,
"Not Found",
                        JOptionPane.ERROR_MESSAGE);
                return;
            }
            int conf = JOptionPane.showConfirmDialog(cancelDialog,
                    "Cancel ticket " + t.ticketId + " for " + t.name + " (Seat " + seatInput + ")?",
"Confirm Cancel",
                    JOptionPane.YES_NO_OPTION);
```

```
            if (conf != JOptionPane.YES_OPTION)
                return;

            synchronized (bookingLock) {
                tickets.remove(t);
                seatToTicket.remove(seatInput);
                seatsAvailable.put(seatInput, true);
            }

            JOptionPane.showMessageDialog(cancelDialog, "Ticket cancelled. Seat " + seatInput + "
is now available.",
                    "Cancelled", JOptionPane.INFORMATION_MESSAGE);
            seatField.setText("");
        });

        cancelDialog.setVisible(true);
    }

    private static abstract class BaseTicket {
        String ticketId;
        String name;
        int age;
        String seat;
        String coach;
        String from;
        String to;
        double price;
        Date bookedOn;

        BaseTicket(String ticketId, String name, int age, String seat, String coach,String from,
String to, double price, Date bookedOn) {
            this.ticketId = ticketId;
            this.name = name;
            this.age = age;
            this.seat = seat;
            this.coach = coach;
            this.from = from;
            this.to = to;
            this.price = price;
```

```
        this.bookedOn = bookedOn;
    }

    // Polymorphic method
    abstract String getTicketType();
}

// ---------- Ticket class ----------
private static class Ticket extends BaseTicket {

    Ticket(String ticketId, String name, int age,
        String seat, String coach,
        String from, String to,
        double price, Date bookedOn) {

        super(ticketId, name, age, seat, coach, from, to, price, bookedOn);
    }

    @Override
    String getTicketType() {
        return "Regular Ticket";
    }
}

// ---------- TicketPanel (with Ticket) ----------
private class TicketPanel extends JPanel {
    private Image bgImg;
    private final SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
    private Ticket ticket;

    TicketPanel(Ticket t) {
        this.ticket = t;
        try {
            bgImg = Toolkit.getDefaultToolkit().getImage("train_bg.png"); // optional background
image
        } catch (Exception ignored) {}
        setPreferredSize(new Dimension(2000, 180)); // broader width
        setMaximumSize(new Dimension(2000, 180));  // keep consistent width in BoxLayout
    }
```

```
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    if (ticket == null) return;

    Graphics2D g2 = (Graphics2D) g;

    // Yellow gradient block
    Color topColor = new Color(255, 245, 180);
    Color bottomColor = new Color(255, 255, 210);
    GradientPaint gp = new GradientPaint(0, 0, topColor, 0, getHeight(), bottomColor);
    g2.setPaint(gp);
    g2.fillRoundRect(5, 5, getWidth() - 10, getHeight() - 10, 20, 20);

    // Optional train image
    if (bgImg != null) {
        g2.setClip(new Rectangle(5, 5, getWidth() - 10, getHeight() - 10));
        g2.drawImage(bgImg, 5, 5, getWidth() - 10, getHeight() - 10, this);
        g2.setClip(null);
    }

    // Draw ticket details - left aligned with padding
    g2.setColor(Color.BLACK);
    g2.setFont(new Font(Font.MONOSPACED, Font.PLAIN, 12));
    int paddingX = 20; // horizontal padding from left
    int paddingY = 15; // vertical padding from top
    int lineSpacing = 12; // spacing between lines

    int y = paddingY;
    g2.drawString("Ticket ID : " + ticket.ticketId, paddingX, y);
    y += lineSpacing;
    g2.drawString("Type      : " + ticket.getTicketType(), paddingX, y);
    y += lineSpacing;
    g2.drawString("Name      : " + ticket.name, paddingX, y);
    y += lineSpacing;
    g2.drawString("Age       : " + ticket.age, paddingX, y);
    y += lineSpacing;
    g2.drawString("Coach     : " + ticket.coach, paddingX, y);
```

```
            y += lineSpacing;
            g2.drawString("Seat      : " + ticket.seat, paddingX, y);
            y += lineSpacing;
            g2.drawString("From      : " + ticket.from, paddingX, y);
            y += lineSpacing;
            g2.drawString("To        : " + ticket.to, paddingX, y);
            y += lineSpacing;
            g2.drawString("Price     : Rs. " + ticket.price, paddingX, y);
            y += lineSpacing;
            g2.drawString("Booked On : " + sdf.format(ticket.bookedOn), paddingX, y);
        }
    }

    // ---------- View Tickets ----------
    private void openViewTicketsWindow() {
        JDialog viewDialog = new JDialog(mainFrame, "View Tickets (Printable)", true);
        viewDialog.setSize(700, 600);
        viewDialog.setLocationRelativeTo(mainFrame);
        viewDialog.setLayout(new BorderLayout(8, 8));

        JPanel ticketsPanel = new JPanel();
        ticketsPanel.setLayout(new BoxLayout(ticketsPanel, BoxLayout.Y_AXIS));
        ticketsPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        loadTicketPanels(ticketsPanel);

        JScrollPane scroll = new JScrollPane(ticketsPanel);
        viewDialog.add(scroll, BorderLayout.CENTER);

        JButton refreshBtn = new JButton("Refresh");
        refreshBtn.addActionListener(e -> {
            ticketsPanel.removeAll();
            loadTicketPanels(ticketsPanel);
            ticketsPanel.revalidate();
            ticketsPanel.repaint();
        });

        JPanel top = new JPanel(new FlowLayout(FlowLayout.RIGHT));
        top.add(refreshBtn);
```

```
    viewDialog.add(top, BorderLayout.NORTH);

    viewDialog.setVisible(true);
}

private void loadTicketPanels(JPanel ticketsPanel) {
    synchronized (bookingLock) {
        for (Ticket t : tickets) {
            ticketsPanel.add(Box.createVerticalStrut(10));
            JLabel top = new JLabel("------------------------------------------------------------");
            top.setAlignmentX(Component.CENTER_ALIGNMENT);
            ticketsPanel.add(top);
            ticketsPanel.add(Box.createVerticalStrut(5));
            TicketPanel tp = new TicketPanel(t);
            tp.setPreferredSize(new Dimension(350, 130));
            tp.setMaximumSize(new Dimension(350, 130));
            tp.setAlignmentX(Component.CENTER_ALIGNMENT);
            ticketsPanel.add(tp);
            ticketsPanel.add(Box.createVerticalStrut(5));
            JLabel bottom = new JLabel("------------------------------------------------------------");
            bottom.setAlignmentX(Component.CENTER_ALIGNMENT);
            ticketsPanel.add(bottom);
            ticketsPanel.add(Box.createVerticalStrut(15));
        }
    }
}

// ---------- Helpers ----------
private String generateTicketId() {
    synchronized (bookingLock) {
        return String.format("T%03d", ticketCounter++);
    }
}

public static List<Ticket> ticketsStatic() {
    return instance.tickets;
}

public static void main(String[] args) {
```

```
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception ignored) {
        }
        new RailwaySystem1();
    }
}
```

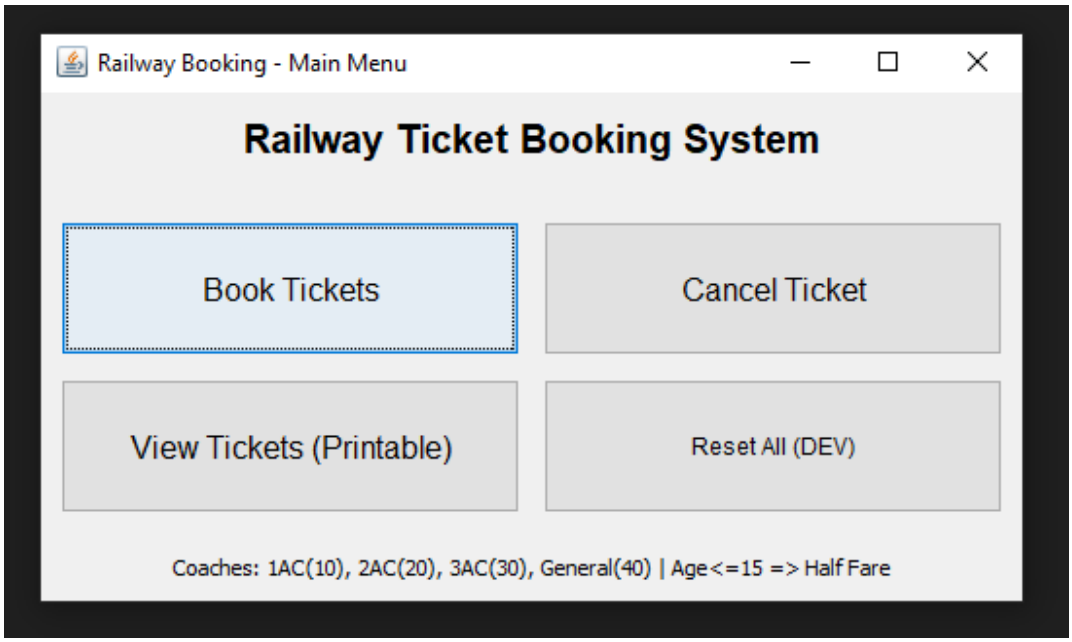
6. CONCLUSION-

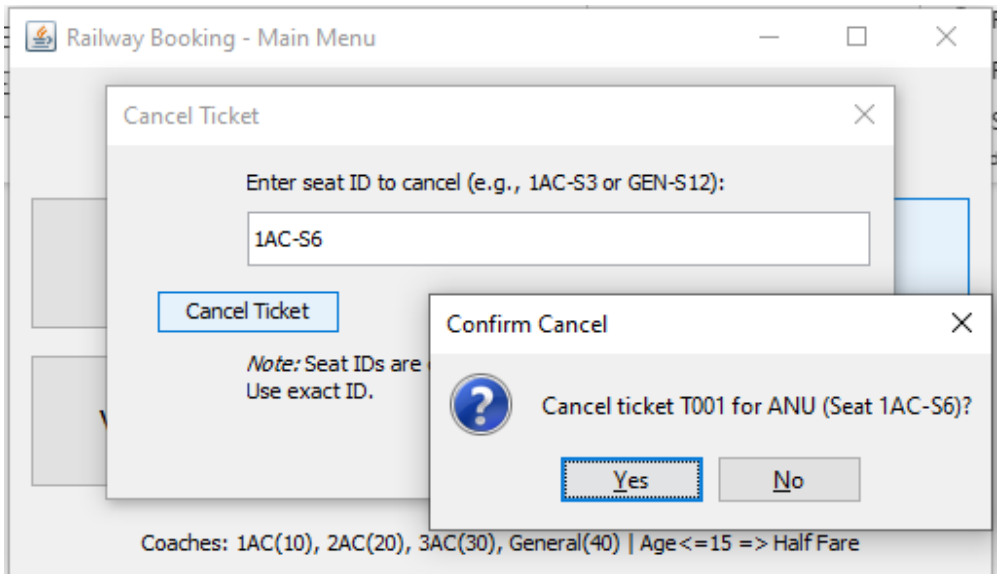
The Railway Ticket Booking System successfully demonstrates the application of **core Java programming concepts** in a real-world scenario. The project integrates **GUI development, object-oriented design, synchronization, collections, and exception handling** effectively.

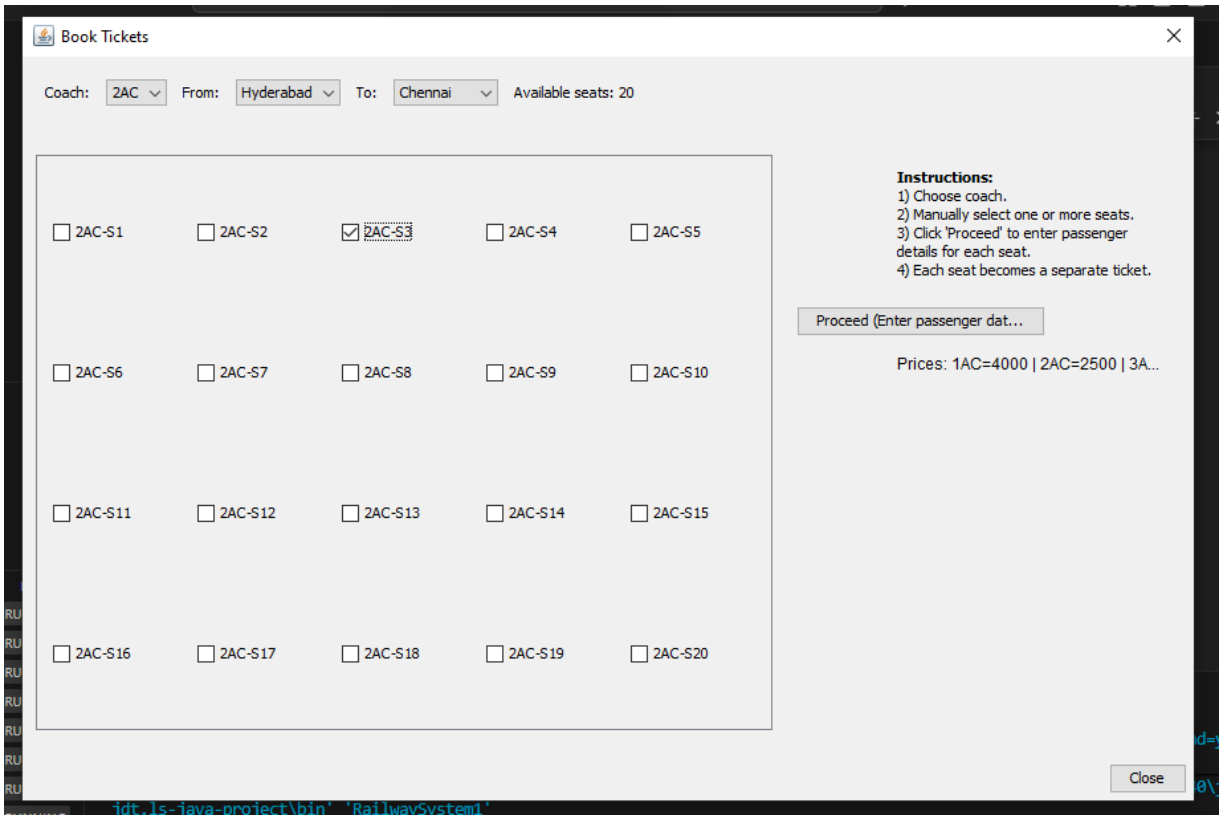
The modular structure of the system improves readability, maintainability, and scalability. By implementing inheritance and polymorphism through ticket classes, the design follows sound OOP principles. The use of synchronization ensures safe handling of shared data, making the system reliable.

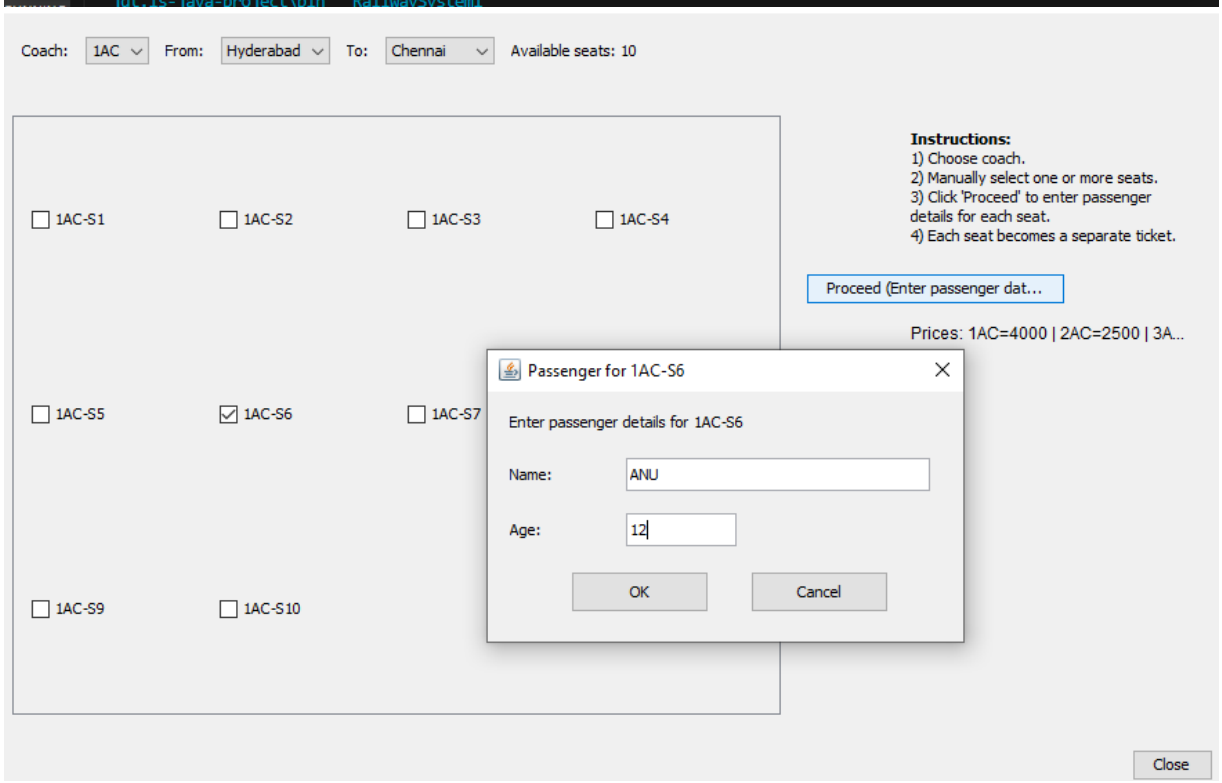
Overall, this project provides a strong foundation for understanding **Java-based application development** and serves as an effective academic and learning tool.

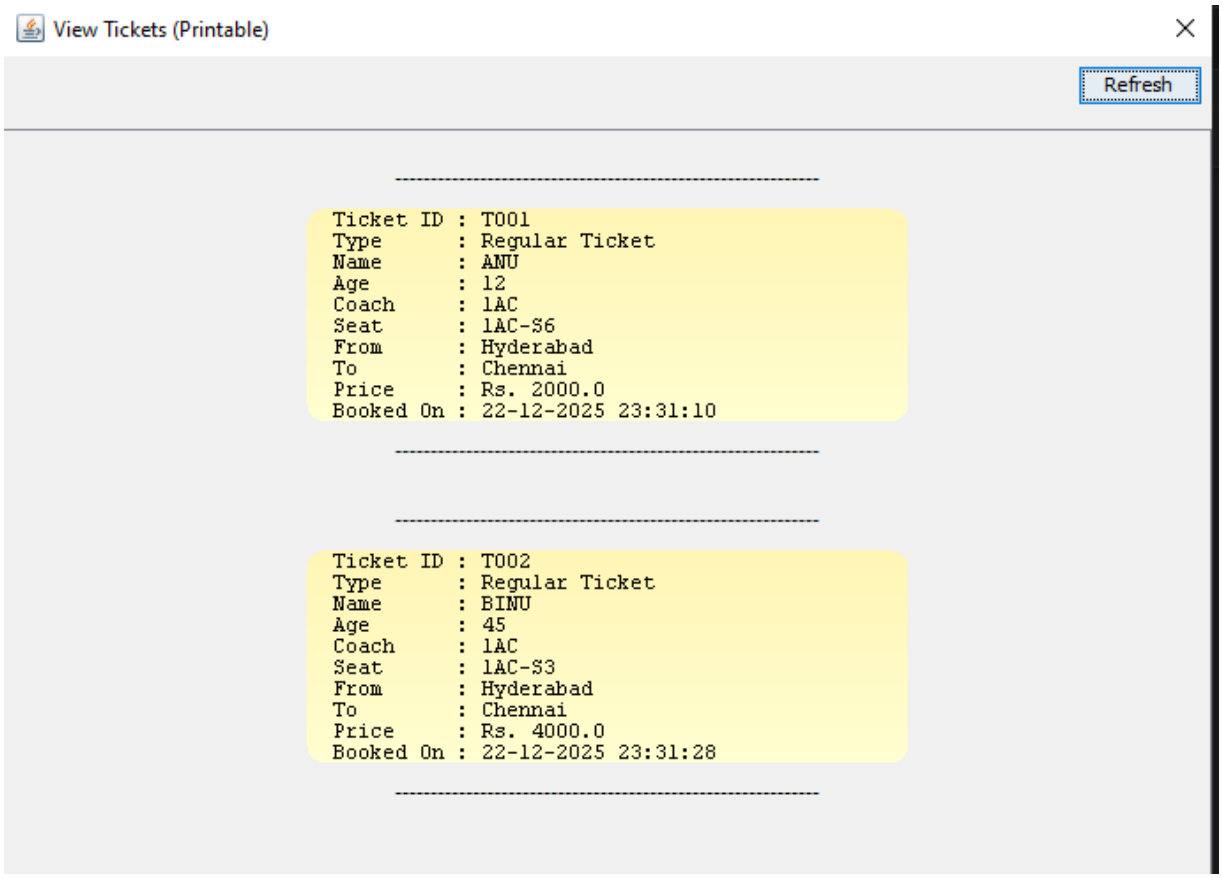
II. SCREENSHOTS

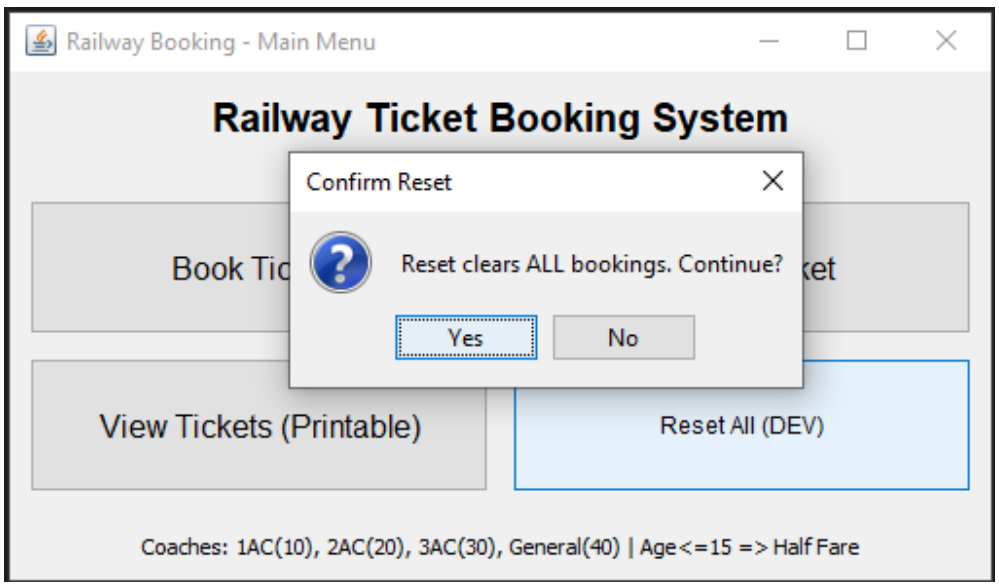

Railway Booking - Main Menu
Railway Ticket Booking System
Book Tickets
Cancel Ticket
View Tickets (Printable)
Reset All (DEV)
Coaches: 1AC(10), 2AC(20), 3AC(30), General(40) | Age<=15 => Half Fare


Railway Booking - Main Menu
Cancel Ticket
Enter seat ID to cancel (e.g., 1AC-S3 or GEN-S12):
1AC-S6
Cancel Ticket
Note: Seat IDs are
Use exact ID.
Confirm Cancel
Cancel ticket T001 for ANU (Seat 1AC-S6)?
Yes
No
Coaches: 1AC(10), 2AC(20), 3AC(30), General(40) | Age<=15 => Half Fare


Book Tickets
Coach: 2AC From: Hyderabad To: Chennai Available seats: 20
2AC-S1 2AC-S2 2AC-S3 2AC-S4 2AC-S5
2AC-S6 2AC-S7 2AC-S8 2AC-S9 2AC-S10
2AC-S11 2AC-S12 2AC-S13 2AC-S14 2AC-S15
2AC-S16 2AC-S17 2AC-S18 2AC-S19 2AC-S20
Instructions:
1) Choose coach.
2) Manually select one or more seats.
3) Click 'Proceed' to enter passenger details for each seat.
4) Each seat becomes a separate ticket.
Proceed (Enter passenger dat...
Prices: 1AC=4000 | 2AC=2500 | 3A...
Close


Coach: 1AC From: Hyderabad To: Chennai Available seats: 10
1AC-S1 1AC-S2 1AC-S3 1AC-S4
1AC-S5 1AC-S6 1AC-S7
1AC-S9 1AC-S10
Instructions:
1) Choose coach.
2) Manually select one or more seats.
3) Click 'Proceed' to enter passenger details for each seat.
4) Each seat becomes a separate ticket.
Proceed (Enter passenger dat...
Prices: 1AC=4000 | 2AC=2500 | 3A...
Passenger for 1AC-S6
Enter passenger details for 1AC-S6
Name: ANU
Age: 12
OK
Cancel
Close


View Tickets (Printable)
Refresh
Ticket ID : T001
Type : Regular Ticket
Name : ANU
Age : 12
Coach : 1AC
Seat : 1AC-S6
From : Hyderabad
To : Chennai
Price : Rs. 2000.0
Booked On : 22-12-2025 23:31:10
Ticket ID : T002
Type : Regular Ticket
Name : BINU
Age : 45
Coach : 1AC
Seat : 1AC-S3
From : Hyderabad
To : Chennai
Price : Rs. 4000.0
Booked On : 22-12-2025 23:31:28


Railway Booking - Main Menu
Railway Ticket Booking System
Confirm Reset
Reset clears ALL bookings. Continue?
Yes
No
View Tickets (Printable)
Reset All (DEV)
Coaches: 1AC(10), 2AC(20), 3AC(30), General(40) | Age<=15 => Half Fare

III. NEW LEARNINGS FROM THE PROJECT

Through the development of the Railway Ticket Booking System, several important technical and practical learnings were gained.

One of the major learnings was the **application of Object-Oriented Programming (OOP) concepts** in a real-world application. The project helped in understanding how **inheritance and abstraction** can be implemented using an abstract base class (BaseTicket) and extended by a concrete class (Ticket). The use of **polymorphism** through method overriding (getTicketType()) demonstrated how runtime behavior can be achieved in Java.

The project also provided hands-on experience in **Java Swing and AWT for GUI development**. Designing multiple windows using JFrame and JDialog, managing layouts, handling button events, and validating user input enhanced understanding of event-driven programming.

Another key learning was the use of **Java Collections Framework** such as HashMap, LinkedHashMap, and ArrayList to store and manage dynamic data like seat availability and booked tickets efficiently.

The implementation of **synchronization** using synchronized blocks introduced the importance of **thread safety** when multiple operations access shared resources. This helped in understanding how race conditions can be avoided in concurrent environments.

The project also improved skills in **exception handling and input validation**, especially while processing user input such as passenger age. Designing custom ticket layouts using **Graphics2D and custom painting** provided insight into advanced GUI rendering techniques.

Overall, the project enhanced problem-solving ability, logical thinking, and confidence in developing structured Java applications.

IV. FUTURE ENHANCEMENTS

Although the current system meets the core requirements, several enhancements can be implemented to improve functionality and scalability.

1. **Database Integration**
   The system can be enhanced by integrating a database such as MySQL or PostgreSQL to store ticket and passenger data permanently instead of using in-memory collections.

2. **User Authentication**
   Login and registration modules can be added to allow multiple users to book and manage their tickets securely.

3. **Multiple Ticket Types**
   Additional ticket types such as Tatkal, Senior Citizen, or Student tickets can be introduced to demonstrate stronger polymorphism.

4. **Online Payment Gateway**
   A payment module can be added to support digital transactions such as UPI, debit, or credit card payments.

5. **Real-Time Multi-User Support**
   The application can be extended into a client–server architecture to support simultaneous bookings by multiple users over a network.

6. **Enhanced GUI and Printing Support**
   Improved ticket designs, print/export options (PDF), and responsive layouts can be added.

7. **Seat Preference & Coach Visualization**
   Features like berth preference and graphical coach layouts can improve user experience.

V. REFERENCES

The information required to complete this project has been taken from the following -

- **Herbert Schildt**, *Java: The Complete Reference*, McGraw-Hill Education, 11th Edition.
- **Cay S. Horstmann**, *Core Java Volume I – Fundamentals*, Pearson Education, 11th Edition.
- **Oracle Corporation**, *Java Platform, Standard Edition Documentation*, Oracle Official Documentation.
- **GeeksforGeeks**, *Java Swing and OOP Concepts Reference Articles*.
- **TutorialsPoint**, *Java GUI Programming (Swing & AWT)*.