OBJECT ORIENTED PROGRAMMING IN JAVA

PROJECT REPORT

BY:

SYED TAHIR ALI
SRIVATHSAN S
R SKANTA SAMVARTAN

TRAIN RESERVATION SYSTEM

PROBLEM STATEMENT:

- The Train Reservation System aims to provide an efficient and user-friendly platform for booking and managing train tickets.
- The system addresses the challenges of manual ticket booking, providing a digital solution that automates the process of ticket reservation, fare calculation, cancellation and seat availability.

MOTIVATION FOR PROBLEM:

- The motivation for developing the Train Reservation System stems from the need to modernize and streamline the process of train ticket booking.
- Traditional manual methods are time-consuming, error-prone, and lack real-time information about seat availability. Passengers face various challenges and inconveniences that hinder the efficiency and user experience of the overall process.
- The system seeks to enhance user experience, reduce human errors, and provide accurate information to both users and railway authorities.

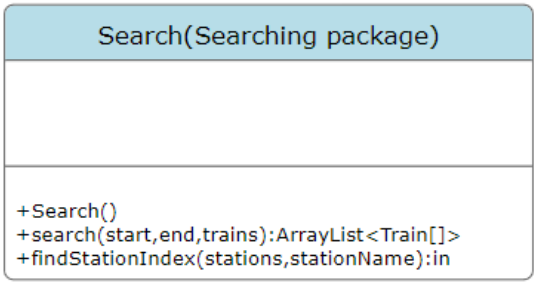
SCOPE:

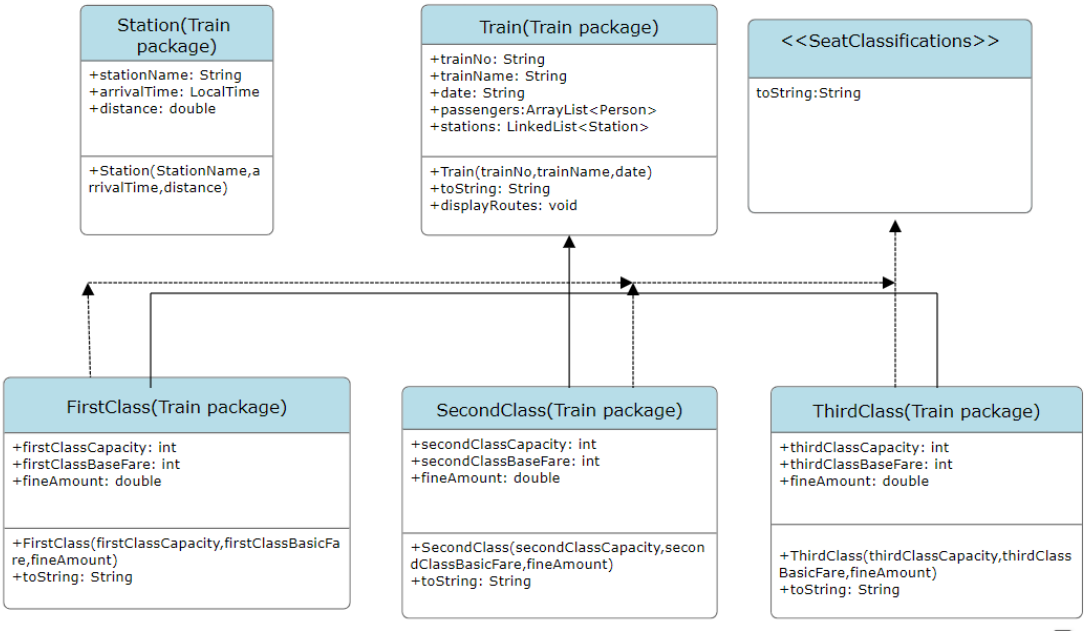
- Online ticket booking for various classes (First Class, Second Class, Third Class).
- Real-time seat availability status.
- Wallet-based payment system for user convenience.
- Ticket cancellation functionality.
- Storage of passenger and ticket information.

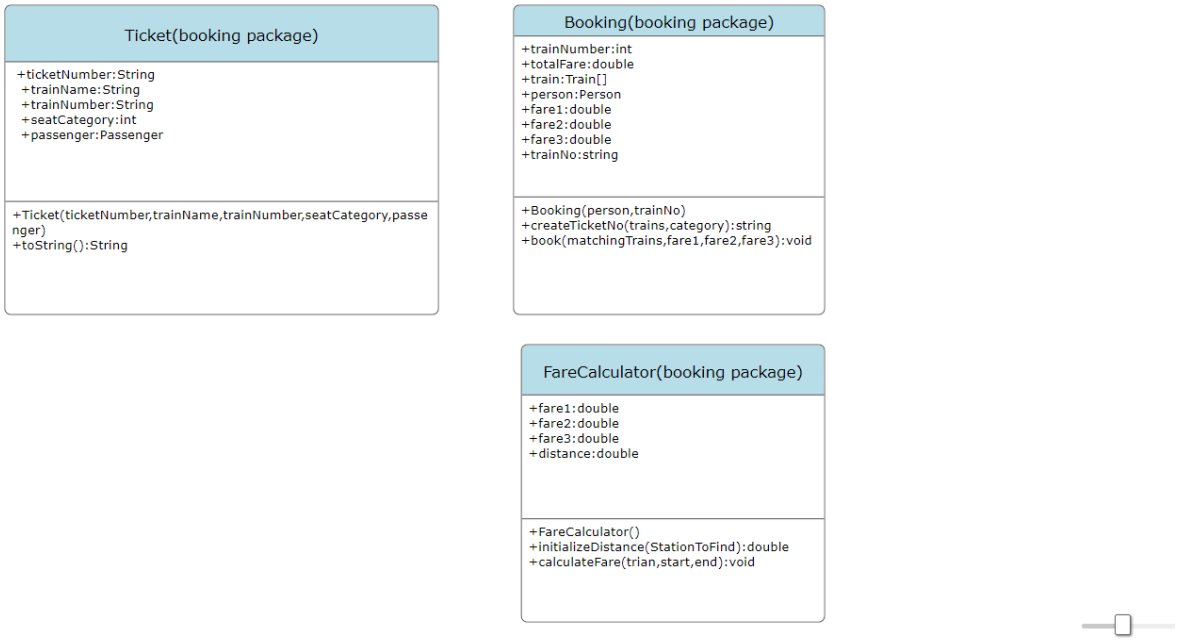
LIMITATION:

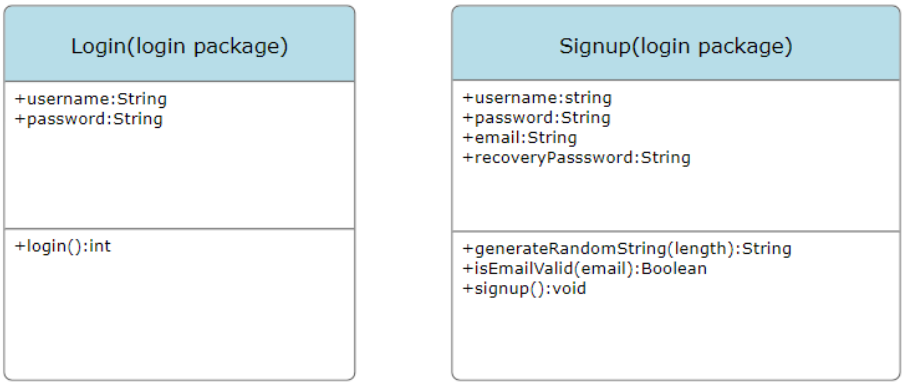
- Limited to a console-based interface.
- Does not incorporate user authentication for security.
- Minimal error handling in the current implementation.
- The system does not consider dynamic train schedules.

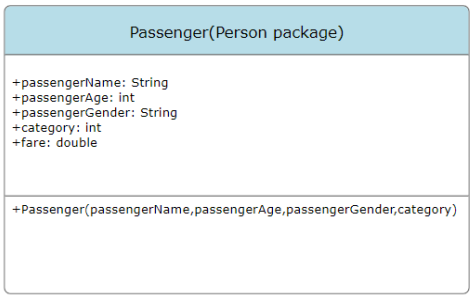
DESIGN OF THE SOLUTION (CLASS DIAGRAM):

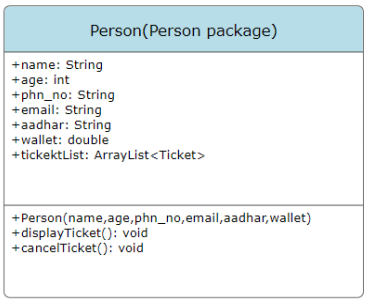


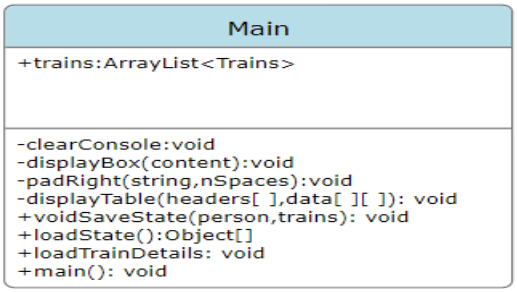



Ticket(booking package)
+ticketNumber:String
+trainName:String
+trainNumber:String
+seatCategory:int
+passenger:Passenger
+Ticket(ticketNumber,trainName,trainNumber,seatCategory,passenger)
+toString():String
Booking(booking package)
+trainNumber:int
+totalFare:double
+train:Train[]
+person:Person
+fare1:double
+fare2:double
+fare3:double
+trainNo:string
+Booking(person,trainNo)
+createTicketNo(trains,category):string
+book(matchingTrains,fare1,fare2,fare3):void
FareCalculator(booking package)
+fare1:double
+fare2:double
+fare3:double
+distance:double
+FareCalculator()
+initializeDistance(StationToFind):double
+calculateFare(trian,start,end):void

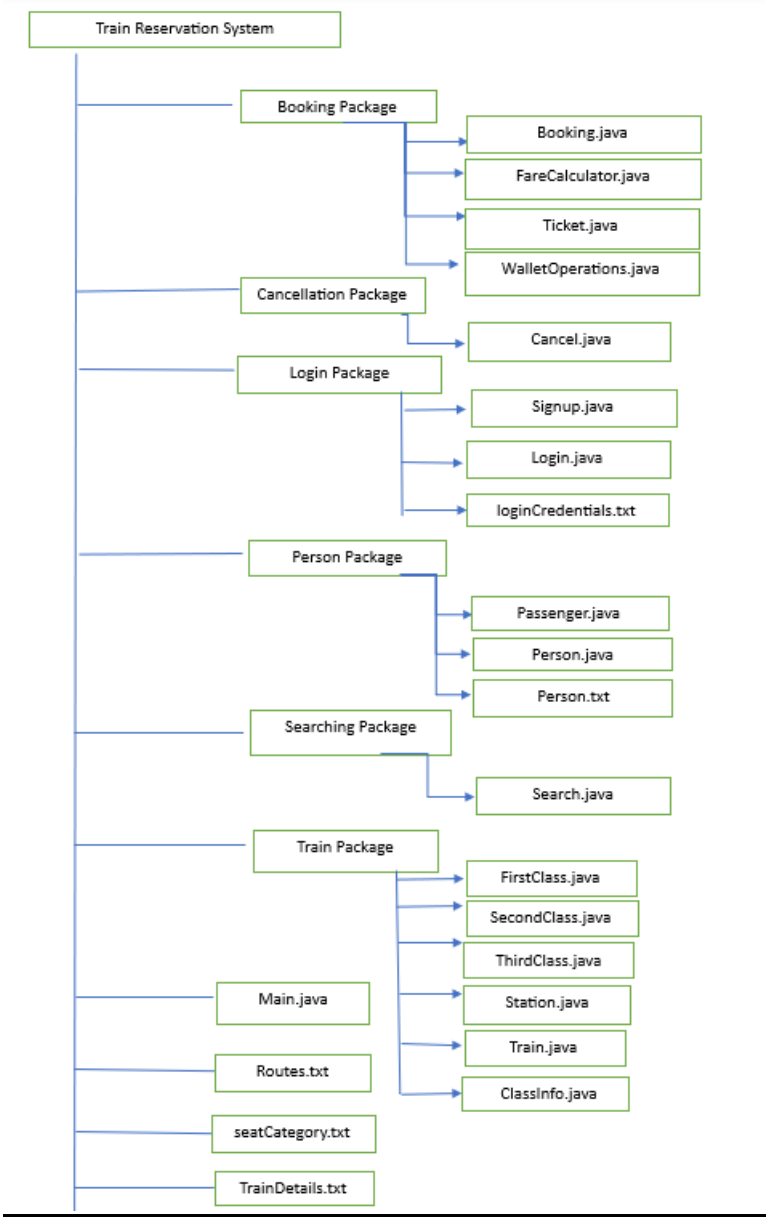

Login(login package)
+username:String
+password:String
+login():int
Signup(login package)
+username:string
+password:String
+email:String
+recoveryPasssword:String
+generateRandomString(length):String
+isEmailValid(email):Boolean
+signup():void


Passenger(Person package)
+passengerName: String
+passengerAge: int
+passengerGender: String
+category: int
+fare: double
+Passenger(passengerName,passengerAge,passengerGender,category)


Person(Person package)
+name: String
+age: int
+phn_no: String
+email: String
+aadhar: String
+wallet: double
+tickektList: ArrayList<Ticket>
+Person(name,age,phn_no,email,aadhar,wallet)
+displayTicket(): void
+cancelTicket(): void


Main
+trains:ArrayList<Trains>
-clearConsole:void
-displayBox(content):void
-padRight(string,nSpaces):void
-displayTable(headers[ ],data[ ][ ]): void
+voidSaveState(person,trains): void
+loadState():Object[]
+loadTrainDetails: void
+main(): void

MODULE SPLIT-UP:



- Class Login:
  - If the user already exist, then this module allows the user to do further booking process. The user can also reset their password through this module, if in case they want to change it or if they have forgotten their old password.
- Class Signup:
  - If the user is new to the system, then this module allows the user to create a new account and the system stores it and also the system provides a recovery code for the user , which can be used in situations where user forget their password.
- Class Booking:

- Booking module takes care of the whole booking process. This module get “from” and “to” locations from the user and display the trains available and handles the further booking process.

- Class FareCalculator:
  - This module calculates the ticket fair by considering the type of ticket the user wants to book(1st class or 2nd class or 3rd class ticket) ,their base fare which is set by the admin and the distance of the whole journey.
- Class Ticket:
  - This module generates the ticket for the current user.
- Interface WalletOperations:
  - WalletOperations Interface ensures that Payment is done through wallet only if sufficient balance is available in the wallet.
- Class Passenger:
  - Passenger Class accepts details of the passengers and stores it in the person object.
- Class Person:
  - This module checks if the person’s data is already available in person.txt file and facilitates further booking or cancellation process.
- Person.txt:
  - This files store the data of user such as name, wallet details.
- Class Train:
  - Reads the details of the train from TrainDetails.txt file and displays it to the user.
- Class Station:
  - Depending upon the stations that the user wants, this module explores the routes available in the routes.txt file.
- Interface ClassInfo:
  - This interface makes sure that capcity,baseFare and fineAmount is available for all the trains.
- Class FirstClass:
  - Stores the data of 1st class capacity, base fare of ticket and corresponding fine amount for cancellation of ticket.
- Class SecondClass:
  - Stores the data of 2nd class capacity, base fare of ticket and corresponding fine amount for cancellation of ticket.

- Class ThirdClass:
  - Stores the data of 3rd class capacity, base fare of ticket and corresponding fine amount for cancellation of ticket.
- Routes.txt:
  - This text file has the details of routes(distance from departure point,location of the station,time) throughout the journey of each train.
- SeatCategory.txt:
  - This text file has the details such as base fare,seats available,fine amount for each train.
- TrainDetails.txt:
  - This text file has the details of the train such as train number,train name, departure time.
- Main:
  - Main module integrates all the modules that were specified above and initiates the whole process of train ticket reservation.

IMPLEMENTATION (CODE):

//Package BookingPackage

//Booking.java

```
package bookingPackage;
import TrainPackage.*;
import java.util.*;
import PersonPackage.*;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.*;
```

```
public class Booking implements WalletOperations,Serializable{

 Scanner scan = new Scanner(System.in);
 private static final long serialVersionUID = 1L;
 public int trainNumber;
 public double totalFare = 0.0;
 public Train[] train;
 public Person person;
 public double fare1;
 public double fare2;
 public double fare3;
 public String trainNo;
 ArrayList<Train[]> Alltrains;
 ArrayList<Passenger> passengers = new ArrayList<Passenger>();

 public Booking(Person person,String trainNo,ArrayList<Train[]> Alltrains){
  this.person = person;
  this.trainNo = trainNo;
  this.Alltrains = Alltrains;
 }

 public String createTicketNo(Train[] trains, int category) {
    String categoryCode;

    if (category==1) {
       categoryCode = "A";
    } else if (category==2) {
       categoryCode = "B";
    } else{
       categoryCode = "C";
```

```
    }

    int capacity;
    if (category==1) {
       capacity = ((FirstClass) trains[1]).firstClassCapacity;
    } else if (category==2) {
       capacity = ((SecondClass) trains[2]).secondClassCapacity;
    } else {
       capacity = ((ThirdClass) trains[3]).thirdClassCapacity;
    }

    String ticketNo = trains[0].train_name + categoryCode +
String.valueOf(capacity);

    return ticketNo;
 }

  public static void writeSeatCategoryToFile(ArrayList<Train[]> trains, String
filePath) {
     try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {
        for (Train[] trainArray : trains) {
           Train train = trainArray[0];
           FirstClass firstClass = (FirstClass) trainArray[1];
           SecondClass secondClass = (SecondClass) trainArray[2];
           ThirdClass thirdClass = (ThirdClass) trainArray[3];

           String line = String.format("%s,%s,%s-%s,%s,%s-%s,%s,%s",
                 Integer.toString(firstClass.firstClassCapacity),
Integer.toString(firstClass.firstClassBaseFare), Double.toString(firstClass.fine_amt),
                       Integer.toString(secondClass.secondClassCapacity),
Integer.toString(secondClass.secondClassBaseFare),
Double.toString(secondClass.fine_amt),
```

```
                                Integer.toString(thirdClass.thirdClassCapacity),
Integer.toString(thirdClass.thirdClassBaseFare),
Double.toString(thirdClass.fine_amt));


            writer.write(line);
            writer.newLine();
        }
        System.out.println("Seat category file updated successfully.");
      }


      catch (IOException e) {
        System.err.println("Error writing to the seat category file: " + e.getMessage());
      }
  }


  public void reduceSeat(Passenger passenger) {


    if(passenger.category==1){
     ((FirstClass)train[1]).firstClassCapacity -= 1;
    }
    else if(passenger.category==2){
     ((SecondClass)train[2]).secondClassCapacity-=1;
    }
    else{
     ((ThirdClass)train[3]).thirdClassCapacity-=1;
     }


    }



  public void updateWalletAmount(double newAmount) {
```

```
    person.wallet = newAmount;
    updateCredentialsFile(); // Call a method to update the credentials file with the new wallet amount
 }

 private void updateCredentialsFile() {
    try {
       // Read all lines from the existing file
       File inputFile = new File("login/loginCredentials.txt");
       File tempFile = new File("login/loginCredentialsTemp.txt");

       BufferedReader reader = new BufferedReader(new FileReader(inputFile));
       BufferedWriter writer = new BufferedWriter(new FileWriter(tempFile));

       String line;
       while ((line = reader.readLine()) != null) {
          if (line.contains(person.getUsername())) {
             // Update the line with the new wallet amount
             String[] parts = line.split(",");
             parts[4] = String.valueOf(person.getWallet());
             line = String.join(",", parts);
          }
          writer.write(line + System.lineSeparator());
       }

       reader.close();
       writer.close();

       // Replace the existing file with the updated file
       inputFile.delete();
       tempFile.renameTo(inputFile);
```

```
    } catch (IOException e) {
        e.printStackTrace();
    }
  }


  //overidding the methods
  public boolean makePayment(double fare){
   if (this.person != null && this.person.wallet >= fare) {
      System.out.println("balance: " + this.person.wallet);
      person.wallet -= fare;
      System.out.println("Payment successful. Remaining balance: " +
this.person.wallet);
      updateWalletAmount(this.person.wallet);
      return true;
   }
   else {
      return false;
   }
  }

  public double checkBalance() {
     return person.wallet;
  }

  public void displayWalletStatus(){
   System.out.println("Balance Wallet Amount :"+this.person.wallet);
  }
```

```
  public void book(ArrayList<Train[]> matchingTrains,double fare1,double
fare2,double fare3) {

    int findCtrl =0;
    int paymentCtrl =0;

    for(Train[] tempObj:matchingTrains){

       if(tempObj[0].train_no.equals(trainNo)){
         train = tempObj;
         findCtrl =1;
         break;
       }
    }

    if(findCtrl ==1){



    System.out.println("Enter the no of tickets to be reserved : ");
    int noOfTickets = scan.nextInt();
     scan.nextLine();

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

       System.out.println("Enter the name of passenger : ");
       String name = scan.nextLine();

       System.out.println("Enter the age of passenger : ");
       int age = scan.nextInt();
       scan.nextLine();
```

```
        System.out.println("Enter the gender of passenger : ");
        String gender = scan.nextLine();

        System.out.println("Enter the category of passenger\n1.Enter 1 for First class\n2. Enter 2 for Second class\n3. Enter 3 for Third class");

        int category = scan.nextInt();
        scan.nextLine();

        int ctrl =0;

      switch (category) {
        case 1:
           ctrl = (((FirstClass) train[1]).firstClassCapacity > 0) ? 1 : 0;
           break;
        case 2:
           ctrl = (((SecondClass) train[2]).secondClassCapacity > 0) ? 1 : 0;
           break;
        case 3:
           ctrl = (((ThirdClass) train[3]).thirdClassCapacity > 0) ? 1 : 0;
           break;
        default:
           System.out.println("Invalid category");
           System.out.println("Re enter the details....");
           i--; // to fill the details of the same person;
      }
        if(ctrl ==1){   // we will be only adding in to the list if the seat availablity of that particular seatcategory is greater than 0;
        Passenger passenger = new Passenger(name,age,gender,category);
        passengers.add(passenger);
         }
```

```
    else{
      System.out.println("\n\n Sorry! Seat is not available \nTry Again Later\n\n");
    }
}

FareCalculator fareCalculator = new FareCalculator();

// Here we are calculating individual fair for the passengers.
for(Passenger passenger:passengers){
  if(passenger.category==1){
    passenger.fare = fare1;
    this.totalFare += fare1;
  }
  else if(passenger.category==2){
    passenger.fare = fare2;
    this.totalFare += fare2;
  }
  else{
    passenger.fare = fare3;
    this.totalFare += fare3;
  }
}

// Prompt the person to pay for the tickets..



System.out.println("Total fare : "+this.totalFare);
//displayWalletStatus();
```

```
    if(person.wallet>=this.totalFare){

      System.out.println("Money Available in wallet : "+person.wallet);
      System.out.println("Do you want to pay for the tickets using wallet ? (1 for yes /0 for no)");

      int choice=scan.nextInt();
      scan.nextLine();

      if(choice == 1 ){

        if (makePayment(this.totalFare)) {
            System.out.println("Payment successful");
            System.out.println("Ticket booked successfully");
            displayWalletStatus();
            paymentCtrl = 1;
        } else {
            System.out.println("Insufficient fund in wallet. Payment failed.");
        }
      }
      else{
        System.out.println("Rs."+this.totalFare+" paid successfully");
        paymentCtrl = 1;
      }
    }


    else{
        System.out.println("Rs."+this.totalFare+" paid successfully");
        paymentCtrl = 1;
```

```
      // we are just passing this...
    }

    //ticket generation after paymentCtrl=1
    if(paymentCtrl == 1){

      for(Passenger passenger:passengers){
        String ticketNo = createTicketNo(train,passenger.category);
        Ticket ticket = new
Ticket(ticketNo,train[0].train_name,train[0].train_no,passenger.category,passenger
);
        (person.ticketlist).add(ticket);

        reduceSeat(passenger);// we are reducing the seat occupied by that passenger.
      }

      train[0].passengers.add(person); //We are adding person object to the ArrayList
inside the Train Class to have the collection of entire ticket booked in the Train Class.

      writeSeatCategoryToFile(Alltrains, "seatCategory.txt"); // writing the seat
category to the file..);
    }
   }
   else{
    System.out.println("Invalid train number");
   }
  }

}
```

//FareCalculator.java

```
package bookingPackage;
import TrainPackage.FirstClass;
import TrainPackage.SecondClass;
import TrainPackage.ThirdClass;
import TrainPackage.Train;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.*;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Scanner;



public class FareCalculator implements Serializable{
  private static final long serialVersionUID = 1L;
  public double distance = 0.0;
  public double fare1 = 0.0;
  public double fare2 = 0.0;
  public double fare3 = 0.0;

  public FareCalculator() {}

  public double initializeDistance(String StationToFind) throws
FileNotFoundException {

    File routesFile = new File("routes.txt");
    Scanner scanner = new Scanner(routesFile);
```

```
  while (scanner.hasNextLine()) {

    String line = scanner.nextLine();
    String[] stationDetails = line.split(",");

    for (int i = 0; i < stationDetails.length; i++) {
       String stationDetail = stationDetails[i];
       String[] parts = stationDetail.split("-");
       String stationName = parts[0];
       double dist = Double.parseDouble(parts[2]);

       if (stationName.equals(StationToFind)) {
          return dist;
       }
    }
  }

  return 0.0;
}


public void calculateFare(Train[] train, String start, String end) {
  double startDistance=0.0;
  double endDistance=0.0;

  try {

    startDistance = initializeDistance(start);
    endDistance = initializeDistance(end);
```

```
    } catch (FileNotFoundException e) {

      System.out.println("Error: file not found.");

    }

    this.distance = endDistance - startDistance;

    this.fare1 = ((FirstClass) train[1]).firstClassBaseFare + (this.distance * 0.7);

    this.fare2 = ((SecondClass) train[2]).secondClassBaseFare + (this.distance * 0.7);

    this.fare3 = ((ThirdClass) train[3]).thirdClassBaseFare + (this.distance * 0.7);

  }
}
```

//Ticket.java

```
package bookingPackage;
import PersonPackage.*;
import java.io.Serializable;

public class Ticket implements Serializable {
  private static final long serialVersionUID = 1L;
  public String ticketNumber;
  public String trainName;
  public String trainNumber;
  public int seatCategory;
  Passenger passenger; // (var of type Passenger)
```

```
  public Ticket() {
  }

  public Ticket(String ticketNumber, String trainName,String trainNumber, int
seatCategory, Passenger passenger) {
    this.ticketNumber = ticketNumber;
    this.trainName = trainName;
    this.trainNumber = trainNumber;
    this.seatCategory = seatCategory;
    this.passenger = passenger;

  }

  public String toString(){
    return ("\nTicket Number: "+ticketNumber+"\nTrain Name: " +
trainName+"\nTrain Number: "+trainNumber+"\nSeat Category:
"+seatCategory+"\npassenger Name :"+passenger.passengerName+"\nPassenger
Age: "+passenger.passengerAge+"\nPassenger Gender:
"+passenger.passengerGender+"\nTicket Fare: "+passenger.fare);
  }

}
```

//WalletOperations.java

```
package bookingPackage;
import java.io.*;

public interface WalletOperations extends Serializable {
  static final long serialVersionUID = 1L;
  boolean makePayment(double fare);
  double checkBalance();
  void displayWalletStatus();
```

```
}
```

//Cancellation Package

//Cancel.java

```
package Cancellation;
import java.util.Scanner;
import java.util.*;
import PersonPackage.*;
import TrainPackage.*; // Import the TrainPackage to access the Train classes
import bookingPackage.*;

public class Cancel {
    Person person;
    Scanner scan;
    ArrayList<Train[]> trains;
     Train[] currTrain;



   public Cancel(Person person,ArrayList<Train[]> trains) {
      this.person = person;
      this.scan = new Scanner(System.in);
      this.trains = trains;

   }

   public void increaseSeat(Ticket ticket, Train[] train) {
      if (ticket.seatCategory == 1) {
         ((FirstClass) train[1]).firstClassCapacity += 1;
      } else if (ticket.seatCategory == 2) {
         ((SecondClass) train[2]).secondClassCapacity += 1;
```

```
    } else {
        ((ThirdClass) train[3]).thirdClassCapacity += 1;
    }
  }

public void cancel() {
    System.out.println("Do you want to cancel the booking? (1 for yes / 0 for no)");
    int cancelChoice;
    try {
        cancelChoice = scan.nextInt();
        scan.nextLine();  // Consume the newline character
    } catch (InputMismatchException e) {
        System.out.println("Invalid input. Please enter either 1 or 0.");
        scan.nextLine();  // Consume the invalid input
        return;
    }

    if (cancelChoice == 1) {

        System.out.println("Enter the train number to cancel the booking:");
        String trainNumber = scan.nextLine();

        for (Train[] train : trains) {
            if (train[0].train_no.equals(trainNumber)) {
                currTrain = train;  // Assigning the current train object to the train variable

                System.out.println("Enter the number of seats to cancel:");
                int no_of_seats;
                try {
                    no_of_seats = scan.nextInt();
                    scan.nextLine();  // Consume the newline character
```

```
            } catch (InputMismatchException e) {
                System.out.println("Invalid input. Please enter a valid number.");
                scan.nextLine();  // Consume the invalid input
                return;
            }

            for (int i = 0; i < no_of_seats; i++) {
                System.out.println("Enter the ticket number to cancel the seat:");
                String ticketNumber = scan.nextLine();

                for (Ticket ticket : person.ticketlist) {
                    if (ticket.ticketNumber.equals(ticketNumber)) {
                        System.out.println("Booking canceled successfully!");
                        System.out.println("\n");
                        increaseSeat(ticket, currTrain);
                        person.ticketlist.remove(ticket);
                        break;
                    }
                }

            }
            Booking.writeSeatCategoryToFile(trains,"seatCategory.txt" );


            return;
        }
    }

    System.out.println("Train not found.");

} else if (cancelChoice == 0) {
```

```
            System.out.println("Your booking has not been cancelled.");
        } else {
            System.out.println("Invalid choice. Please enter either 1 or 0.");
        }
    }
}
```

//Login package

//Login.java

```
package login;

import java.io.*;
import java.util.Scanner;

public class Login {
    public String userName;
    public String password;

    public int login() {
        try {
            File file = new File("login/loginCredentials.txt");
            Scanner scan = new Scanner(System.in);
            Scanner fileObj = new Scanner(file);

            System.out.println("**********************************");
            System.out.println("************ LOGIN PAGE **********");
            System.out.println("**********************************");

            System.out.print("\nEnter the username: ");
            userName = scan.nextLine();
```

```
        System.out.print("Enter the password: ");
        password = scan.nextLine();

        StringBuilder updatedContent = new StringBuilder();

        while (fileObj.hasNextLine()) {
            String line = fileObj.nextLine();
            String[] arr = line.split(",");

            if (userName.equals(arr[0]) && password.equals(arr[1])) {
                System.out.println("\n**********************************");
                System.out.println("***** You Logged in successfully *");
                System.out.println("**********************************");
                System.out.println();
                return 1;
            }

            if (userName.equals(arr[0])) {
                System.out.println("\nUser found but invalid password..");
                System.out.println("\nEnter 1 to reset password:");

                int reset = scan.nextInt();
                scan.nextLine();

                if (reset == 1) {
                    System.out.print("Enter the password reset code provided to you or Enter your old
Password: ");
                    String code = scan.nextLine();

                    if (code.equals(arr[3]) || code.equals(arr[1])) {
                        String newPassword;
```

```
            String newPassword1;

            do {
               System.out.print("Enter the New Password: ");
               newPassword = scan.nextLine();

               System.out.print("Reenter the New Password: ");
               newPassword1 = scan.nextLine();

               if (newPassword.equals(newPassword1)) {
                  System.out.println("**********************************");
                  System.out.println("***** Password reset successfully ****");
                  System.out.println("**********************************");
                  arr[1] = newPassword; // Update the password in the array
               } else {
                  System.out.println("Passwords do not match. Try again.");
               }
            } while (!newPassword.equals(newPassword1));
         }
      }

      updatedContent.append(String.join(",", arr)).append("\n");
   } else {
      updatedContent.append(line).append("\n");
   }
}

// Write the updated content back to the file
try (PrintWriter writer = new PrintWriter(file)) {
   writer.println(updatedContent.toString().trim());
}
```

```
            System.out.println("\n**********************************");
            System.out.println("***** Entered UserName or password is wrong ****");
            System.out.println("**********************************");
            return 0;
        } catch (FileNotFoundException e) {
            System.out.println("Problem in opening the file");
            return 0;
        } catch (IOException e) {
            System.out.println("Error reading or writing to the file");
            return 0;
        }
    }

    public String getUserName() {
        return userName;
    }
}
```

//Signup.java

```
package login;

import java.io.FileWriter;
import java.io.IOException;
import java.util.InputMismatchException;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

// Custom exception for invalid age format
```

```
class InvalidAgeException extends Exception {
    public InvalidAgeException(String message) {
        super(message);
    }
}

// Custom exception for invalid Aadhar format
class InvalidAadharException extends Exception {
    public InvalidAadharException(String message) {
        super(message);
    }
}

// Custom exception for invalid email format
class InvalidEmailException extends Exception {
    public InvalidEmailException(String message) {
        super(message);
    }
}

public class Signup {

    public String userName;
    public String password;
    public String email;
    public String recoveryPassword;
    public String phnNo, aadhar;
    public int age;
    public double wallet;

    public Signup() {
```

```
    }

    public Signup(String userName, String password, String email, String recoveryPassword) {
        this.userName = userName;
        this.password = password;
        this.email = email;
        this.recoveryPassword = recoveryPassword;
    }

    public String getUserName() {
        return userName;
    }

    public String getPassword() {
        return password;
    }

    public String getEmail() {
        return email;
    }

    public String getRecoveryPassword() {
        return recoveryPassword;
    }

    public String generateRandomString(int length) {
        String characters =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
        StringBuilder randomString = new StringBuilder();
        for (int i = 0; i < length; i++) {
            int index = (int) (Math.random() * characters.length());
```

```
            char randomChar = characters.charAt(index);
            randomString.append(randomChar);
        }
        return randomString.toString();
    }

    public boolean isValidAadhar(String aadhar) {
        return aadhar.matches("\\d{5}"); // Assuming Aadhar has 5 digits
    }

    public boolean isEmailValid(String email) {
        String emailRegex = "^[a-zA-Z0-9_+&*-]+(?:\\.[a-zA-Z0-9_+&*-]+)*@(?:[a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,7}$";
        Pattern pattern = Pattern.compile(emailRegex);
        Matcher matcher = pattern.matcher(email);
        return matcher.matches();
    }

    public void signup() {
        try {
            FileWriter file = new FileWriter("login/loginCredentials.txt", true);

            Scanner scan = new Scanner(System.in);
            System.out.println("\nEnter the username: ");
            userName = scan.nextLine();
            System.out.println("\nEnter the password: ");
            password = scan.nextLine();

            // Validate age
            do {
                System.out.println("\nEnter your age: ");
```

```
            try {
                age = scan.nextInt();
                if (age <= 0) {
                    throw new InvalidAgeException("Age must be a positive integer.");
                }
            } catch (InputMismatchException e) {
                throw new InvalidAgeException("Invalid age format. Please enter a valid age.");
            }
            scan.nextLine();
        } while (age <= 0);

        // Validate Aadhar
        do {
            System.out.println("\nEnter your Aadhar number: ");
            aadhar = scan.nextLine();
            if (!isValidAadhar(aadhar)) {
                throw new InvalidAadharException("Invalid Aadhar format(Must have 5 digits). Please
enter a valid Aadhar number.");
            }
        } while (!isValidAadhar(aadhar));

        // Validate email
        do {
            System.out.println("\nEnter the email: ");
            email = scan.nextLine();

            if (!isEmailValid(email)) {
                throw new InvalidEmailException("Invalid email format. Please provide a valid email.");
            }
        } while (!isEmailValid(email));
```

```
            System.out.println("\nEnter your phone number: ");
            phnNo = scan.nextLine();
            System.out.println("\nEnter your wallet amount: ");
            wallet = scan.nextDouble();
            scan.nextLine();

            recoveryPassword = generateRandomString(5);
            System.out.println("Please note down your account recovery code: " + recoveryPassword);
            System.out.println("\n\nPlease note down the above-generated password to reset your password later!");

            file.write(userName + "," + password + "," + email + "," + recoveryPassword + "," + wallet + "\n");
            scan.close();
            file.close();

            System.out.println("\nSignup successful.");

        } catch (IOException e) {
            System.out.println("Problem in opening the file");
        } catch (InvalidAgeException | InvalidAadharException | InvalidEmailException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }


}
```

//Package PersonPackage

//Person.java

```
package PersonPackage;
import java.util.*;
import bookingPackage.*;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.io.IOException;
import java.io.Serializable;
import java.io.EOFException;
import java.io.*;

public class Person implements Serializable{

  public String name;
  public int age;
  public String ph_no;
  public String email;
  public String aadhar;
  public ArrayList<Ticket> ticketlist = new ArrayList();

  public int ticketCount=ticketlist.size();
  public double wallet;

  public Person(){}

  public Person(String name,int age,String ph_no,String email,String aadhar,double
wallet){
     this.name = name;
     this.age = age;
```

```
    this.ph_no = ph_no;
    this.email = email;
    this.aadhar = aadhar;
    this.wallet = wallet;
    writeToFile();
  }

 public void writeToFile() {

    try (BufferedWriter writer = new BufferedWriter(new
FileWriter("PersonPackage/person.txt", true))) {
       String line = String.format("%s,%d,%s,%s,%s,%.2f,%s", // Added %s for
ticketNumber
            this.name, this.age, this.ph_no, this.email, this.aadhar, this.wallet,
getTicketNumbers());

       writer.write(line);
       writer.newLine();
       System.out.println("Person data written to file successfully.");
    }
     catch (IOException e) {
       e.printStackTrace();
    }
 }

 // New method to get comma-separated ticket numbers
 private String getTicketNumbers() {
    StringBuilder ticketNumbers = new StringBuilder();
    for (Ticket ticket : ticketlist) {
       ticketNumbers.append(ticket.ticketNumber).append(",");
    }
    return ticketNumbers.toString();
```

```
}

public void displayTicket() {
  for (Ticket ticket : ticketlist) {
    ticket.toString();
  }
} // displays details of all the tickets that the person has booked.



public void cancelTicket() {
  Scanner scan = new Scanner(System.in);
  String TicketNo;

  displayTicket();
  TicketNo = scan.nextLine();

} // in here call displayTicket() method first.Then get ticketNumber as input.
  // Then, using input we will identify corresponding ticket object(which has same
  // ticketNumber).Using ticket object, we will identify the category of seat.Then
  // calculate fine amount based on category.(display fine in wallet).

public static void writeListToFile(ArrayList<Person> persons) {
    try {
      FileOutputStream fos = new
FileOutputStream("PersonPackage/personData.ser");
      ObjectOutputStream oos = new ObjectOutputStream(fos);
      oos.writeObject(persons);
    } catch (IOException e) {
       e.printStackTrace();
    }
}
```

```
 // In Person class
 public String getUsername() {
     return this.name; // or whatever field represents the username in your Person class
 }
 public double getWallet(){
  return this.wallet;
 }

 public static ArrayList<Person> readListFromFile() {
    File file = new File("PersonPackage/personData.ser");

    if (!file.exists()) {
       System.out.println("File is empty. Returning empty list.");
       return new ArrayList<>();
    }

    try{
     FileInputStream fis = new FileInputStream(file);
       ObjectInputStream ois = new ObjectInputStream(fis);
       return (ArrayList<Person>) ois.readObject();
    } catch (FileNotFoundException e) {
       System.out.println("File not found. Returning empty list.");
    } catch (IOException | ClassNotFoundException e) {
       e.printStackTrace();
    }
    return new ArrayList<>();
 }
```

```
}
```

//Passenger.java

```
package PersonPackage;
import java.io.*;

public class Passenger extends Person implements Serializable{
  private static final long serialVersionUID = 1L;
  public String passengerName;
  public int passengerAge;
  public String passengerGender;
  public int category;
  public double fare;

  public Passenger(){}

  public Passenger(String passengerName, int passengerAge, String
passengerGender, int category) {
    this.passengerName = passengerName;
    this.passengerAge = passengerAge;
    this.passengerGender = passengerGender;
    this.category = category;

  }
}
```

//Package SearchingPackage

//Search.java

```
package SearchingPackage;
```

```
import TrainPackage.*;

import java.util.ArrayList;
import java.util.LinkedList;
import java.io.*;
public class Search implements Serializable
{
  private static final long serialVersionUID = 1L;
  public Search() {
  }

  public static ArrayList<Train[]> search(String start, String end, ArrayList<Train[]>
trains) {

     ArrayList<Train[]> matchingTrains = new ArrayList<>();

     for (Train[] train : trains){

        LinkedList<Station> stations = train[0].stations;

        int startIndex = findStationIndex(stations, start);
        int endIndex = findStationIndex(stations, end);

        if (startIndex != -1 && endIndex != -1 && startIndex < endIndex) {
           matchingTrains.add(train);
        }
     }
     return matchingTrains;
  }
```

```
    private static int findStationIndex(LinkedList<Station> stations, String
stationName) {
        for (int i = 0; i < stations.size(); i++) {
            if (stations.get(i).stationName.equals(stationName)) {
                return i;
            }
        }
        return -1; // Station not found in the route
    }
}
```

//Package TrainPackage

//ClassInfo.java

```
package TrainPackage;
import java.io.*;

public interface ClassInfo {

    int getCapacity();
    int getBaseFare();
    double getFineAmount();
}
```

//Station.java

```
package TrainPackage;

import java.time.*;
import java.time.format.DateTimeFormatter;
```

```
import java.io.Serializable;

public class Station implements Serializable{
  private static final long serialVersionUID = 1L;
  public String stationName;
  public LocalTime arrivalTime;
  public double distance;

  public Station(String stationName, String arrivalTime, double distance) {

    this.stationName = stationName;

    DateTimeFormatter formatObj = DateTimeFormatter.ofPattern("HH:mm");
    this.arrivalTime = LocalTime.parse(arrivalTime, formatObj);
    this.distance = distance;
  }
}
```

//Train.java

```
package TrainPackage;

import PersonPackage.*;
import java.time.*;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.LinkedList;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
```

```
import java.io.ObjectInputStream;
import java.io.IOException;
import java.io.Serializable;
import java.io.EOFException;
import java.io.*;



public class Train implements Serializable{
  private static final long serialVersionUID = 1L;
  public String train_no;
  public String train_name;
  public LocalDateTime dateTime;
  public LinkedList<Station> stations = new LinkedList<Station>();
  public ArrayList<Person> passengers = new ArrayList<Person>();

  public Train() {
  }

  public Train(String train_no, String train_name, String date) {

    this.train_no = train_no;
    this.train_name = train_name;
    DateTimeFormatter formatObj = DateTimeFormatter.ofPattern("dd-MM-yyyy
HH:mm:ss");
    this.dateTime= LocalDateTime.parse(date, formatObj);

  }

  public String toString() {
    return "Train No: " + train_no + " Train Name: " + train_name + " Date: " +
dateTime;
```

```
}

public void displayRoutes() {

  System.out.println("\nTRAIN SCHEDULE:");
  for (Station s : stations) {
    System.out.println(s.stationName + " " + s.arrivalTime );
  }

}

public static void writeListToFile(ArrayList<Train[]> trains)  throws Exception{
  try {
     FileOutputStream fos =new FileOutputStream("TrainPackage/trainData.ser");
     ObjectOutputStream oos = new ObjectOutputStream(fos);
     oos.writeObject(trains);
     oos.close();
     fos.close();
  }catch (FileNotFoundException e) {
       System.out.println("File not found. Returning empty list.");
  }
  catch(IOException e) {
     e.printStackTrace();
  }
}

public static ArrayList<Train[]> readListFromFile() throws Exception{
    try{
       FileInputStream fos =new FileInputStream("TrainPackage/trainData.ser");
       ObjectInputStream ois = new ObjectInputStream(fos);
       return (ArrayList<Train[]>) ois.readObject();
```

```
    } catch (FileNotFoundException e) {
        System.out.println("File not found. Returning empty list.");
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
    }
    return new ArrayList<>();
 }
}
```

//FirstClass.java

```
package TrainPackage;
import java.io.*;
public class FirstClass extends Train implements ClassInfo,Serializable{

 public int firstClassCapacity;
 public int firstClassBaseFare;
 public double fine_amt;
 private static final long serialVersionUID = 1L;
 public FirstClass() {
 }

 public FirstClass(int firstClassCapacity, int firstClassBaseFare, double fine_amt) {

  this.firstClassCapacity = firstClassCapacity;
  this.firstClassBaseFare = firstClassBaseFare;
  this.fine_amt = fine_amt;

 }
 public int getCapacity() {
    return firstClassCapacity;
 }
```

```
  public int getBaseFare() {
     return firstClassBaseFare;
  }

  public double getFineAmount() {
     return fine_amt;
  }
  public String toString() {
    return "FirstClassCapacity : " + firstClassCapacity + "FirstClassBaseFair: " +
firstClassBaseFare + "Fine_amt: " + fine_amt ;
  }
}
```

//SecondClass.java

```
package TrainPackage;
import java.io.*;

public class SecondClass extends Train implements ClassInfo,Serializable {

  public int secondClassCapacity;
  public int secondClassBaseFare;

  public double fine_amt;
  private static final long serialVersionUID = 1L;
  public SecondClass() {
  }

  public SecondClass(int secondClassCapacity, int secondClassBaseFare, double
fine_amt) {
```

```
    this.secondClassCapacity = secondClassCapacity;
    this.secondClassBaseFare = secondClassBaseFare;
    this.fine_amt = fine_amt;

  }
  public int getCapacity() {
     return secondClassCapacity;
  }

  public int getBaseFare() {
     return secondClassBaseFare;
  }

  public double getFineAmount() {
     return fine_amt;
  }
  public String toString() {
    return "SecondClassCapacity: " + secondClassCapacity + " SecondClassBaseFare "
+ secondClassBaseFare + "Fine_amt: " + fine_amt;
  }
}
```

//ThirdClass.java

```
package TrainPackage;
import java.io.*;

public class ThirdClass extends Train implements ClassInfo,Serializable {
  private static final long serialVersionUID = 1L;
  public int thirdClassCapacity;
  public int thirdClassBaseFare;
  // ArrayList<Tickets> thirdClassTickets = new ArrayList<Tickets>();
```

```
  public double fine_amt;


  public ThirdClass() {
  }


  public ThirdClass(int thirdClassCapacity, int thirdClassBaseFare, double fine_amt)
{
    this.thirdClassCapacity = thirdClassCapacity;
    this.thirdClassBaseFare = thirdClassBaseFare;
    this.fine_amt = fine_amt;
  }
  public int getCapacity() {
     return thirdClassCapacity;
  }


  public int getBaseFare() {
     return thirdClassBaseFare;
  }


  public double getFineAmount() {
     return fine_amt;
  }
  public String toString() {
    return " ThirdClassCapacity: " + thirdClassCapacity  + " ThirdClassBaseFair: " +
thirdClassBaseFare + " Fine_amt: " + fine_amt;
  }
}
```

//Main

```
import TrainPackage.*;
import login.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.EOFException;
import java.io.*;
import java.util.*;
import java.io.IOException;
import bookingPackage.*;
import SearchingPackage.*;
import PersonPackage.*;
import Cancellation.*;


public class new2 implements Serializable{

 //instance variables....
 private static Person currPer=null;

 private static ArrayList<Person> persons = new ArrayList<>();
 private static ArrayList<Train[]> trains = new ArrayList<>();



  private static void clearConsole() {
     try {
        final String os = System.getProperty("os.name");

        if (os.contains("Windows")) {
```

```
            new ProcessBuilder("cmd", "/c", "cls").inheritIO().start().waitFor();
        } else {
            System.out.print("\033[H\033[2J");
            System.out.flush();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
 }

private static void displayTicketBox(String content) {
   int width = content.length() + 4;
   System.out.println("+" + "-".repeat(width) + "+");
   System.out.println("| " + content + " |");
   System.out.println("+" + "-".repeat(width) + "+");
}

// Display tickets in a box
private static void displayTickets(ArrayList<Ticket> tickets) {
   for (Ticket ticket : tickets) {
      displayTicketBox(ticket.toString());
   }
}
// Display content in a box
   private static void displayBox(String content) {
      int width = content.length() + 4;
      System.out.println("+" + "-".repeat(width) + "+");
      System.out.println("| " + content + " |");
      System.out.println("+" + "-".repeat(width) + "+");
   }
```

```
// Pad a string to the right with spaces
private static String padRight(String s, int n) {
    return String.format("%-" + n + "s", s);
}


// Display data in a table format
private static void displayTable(String[] headers, String[][] data) {
    int[] maxLengths = new int[headers.length];

    // Calculate maximum lengths for each column
    for (int i = 0; i < headers.length; i++) {
        maxLengths[i] = headers[i].length();
        for (String[] row : data) {
            if (row[i].length() > maxLengths[i]) {
                maxLengths[i] = row[i].length();
            }
        }
    }

    // Print headers
    for (int i = 0; i < headers.length; i++) {
        System.out.print(padRight(headers[i], maxLengths[i] + 2) + "|");
    }
    System.out.println();
    // Print separator line
    for (int i = 0; i < headers.length; i++) {
        System.out.print("-".repeat(maxLengths[i] + 2) + "+");
    }
    System.out.println();
```

```
        // Print data rows
        for (String[] row : data) {
            for (int i = 0; i < headers.length; i++) {
                System.out.print(padRight(row[i], maxLengths[i] + 2) + "|");
            }
            System.out.println();  // Move this line to print a new line after each row
        }
    }


    public static void main(String[] args) {


        Scanner inp = new Scanner(System.in);
        int choice,loginCtrl = 0;



            clearConsole();
            System.out.println("Welcome to the Train Reservation System\n\n");
            System.out.println("1. Sign up\n\n2. Sign in\n\n3. Exit\n\n");
            System.out.println("Enter your choice :");
            choice = inp.nextInt();
            inp.nextLine();

            //persons = Person.readListFromFile();
            //trains = Train.readListFromFile();

        currPer = new Person("Srivathsan", 18
    ,"8939630347","srivathsan@gmail.com","12345678901",10000);
```

```
        switch (choice) {
            case 1:
                Signup signupObj = new Signup();
                signupObj.signup();
                // Add the new user to the persons ArrayList
                // String name,int age,String ph_no,String email,String aadhar,double
wallet


                persons.add(new Person(signupObj.userName,
signupObj.age,signupObj.phnNo,signupObj.email,signupObj.aadhar,signupObj.wall
et));


                //currPer = new Person(signupObj.userName,
signupObj.age,signupObj.phnNo,signupObj.email,signupObj.aadhar,signupObj.wall
et);


                // Set the current person to the signed-up user


            currPer = persons.get(persons.size() - 1);
            System.exit(0);
                break;


            case 2:
                Login loginObj = new Login();
                loginCtrl = loginObj.login();
                 // for (Person person_ : persons) {
                 //   if (person_.getUsername().equals(loginObj.userName)) {
                 //     currPer = person_;
                 //     System.out.println("Current person: " + currPer.getUsername());
                 //     break;
                 //   }
                 // }
```

```
            break;

        case 3:
            System.out.println("Thank you for using the system!");
            System.exit(0);

        default:
            System.out.println("Invalid input...\n\n");
      }


  if (loginCtrl == 1) {
      try {

          File fileObj = new File("trainDetails.txt");
          File routeObj = new File("routes.txt");
          File seatcategoryObj = new File("seatCategory.txt");

          Scanner scan = new Scanner(fileObj);
          Scanner route = new Scanner(routeObj);
          Scanner seatcategory = new Scanner(seatcategoryObj);

          while (scan.hasNextLine() && route.hasNextLine() &&
seatcategory.hasNextLine()) {
              Train[] traindetail = new Train[4];// Using this we will be storing train
obj ,firstclass obj , second and
                                  // third class obj...

              String line = scan.nextLine();
              String routes = route.nextLine();
```

```
String categoryDetails = seatcategory.nextLine();

String[] arr = line.split(",");// splitting the train related info's
String train_no = arr[0];
String train_name = arr[1];
String date = arr[2];

Train newObj = new Train(train_no, train_name, date);
traindetail[0] = newObj;// we are having the 0th element as the trainobj...

String[] arr2 = routes.split(",");// for splitting the train routes.

// after this we are further splitting the details like junction , arrival time
// and distance..
String[] arr3;

for (int i = 0; i < arr2.length; i++) {
   arr3 = arr2[i].split("-");
   (newObj.stations).add(new Station(arr3[0], arr3[1],
Double.parseDouble(arr3[2]))); // we are adding each
   // junction into the routes
   // of the respective train
   // obj.
}

// After this we are splitting the seat category details.
String[] arr4 = categoryDetails.split("-");

for (int i = 0; i < arr4.length; i++) {
   String[] dup = arr4[i].split(",");
```

```
                switch (i + 1) {
                    case 1:
                        traindetail[1] = new FirstClass(Integer.parseInt(dup[0]),
Integer.parseInt(dup[1]),
                                Double.parseDouble(dup[2]));
                        break;

                    case 2:
                        traindetail[2] = new SecondClass(Integer.parseInt(dup[0]),
                                Integer.parseInt(dup[1]), Double.parseDouble(dup[2]));
                        break;

                    case 3:
                        traindetail[3] = new ThirdClass(Integer.parseInt(dup[0]),
Integer.parseInt(dup[1]),
                                Double.parseDouble(dup[2]));
                        break;
                    default:
                        System.out.println("Wrong file format..");
                }
            }

            trains.add(traindetail); // Finally we are adding the train info with the
respective first,second and
            // third class info as a array into the ArrayList.


        }
        scan.close();
    } catch (FileNotFoundException e) {
        System.out.println("error occurred");
    }
```

```
    }
      // System.out.println("current person tickets : ");
       //currPer.displayTicket();
       //inp.nextLine();
       Scanner sc=new Scanner(System.in);
       System.out.println("Enter the From location :");
       String start = sc.nextLine();

       System.out.println("\n");

       System.out.println("Enter the To location :");
       String end = sc.nextLine();

       System.out.println("\n");

       ArrayList<Train[]> matchingTrains = Search.search(start, end, trains);

       if (!matchingTrains.isEmpty()) {
          FareCalculator fareCalculator = new FareCalculator();

          String[] headers = {"Train No", "Train Name", "Date", "First Class
Fare","Seats","Second Class Fare","Seats","Third Class Fare","Seats"};
          String[][] data = new String[matchingTrains.size()][headers.length];

          int index = 0;
          for (Train[] foundTrain : matchingTrains) {
             data[index][0] = foundTrain[0].train_no;
             data[index][1] = foundTrain[0].train_name;
             data[index][2] = foundTrain[0].dateTime.toString();
             fareCalculator.calculateFare(foundTrain, start, end);
```

```
                data[index][3] = String.valueOf(fareCalculator.fare1);
                data[index][4] = String.valueOf(((FirstClass)
foundTrain[1]).firstClassCapacity);
                data[index][5] = String.valueOf(fareCalculator.fare2);
                data[index][6] = String.valueOf(((SecondClass)
foundTrain[2]).secondClassCapacity);
                data[index][7] = String.valueOf(fareCalculator.fare3);
                data[index][8] = String.valueOf(((ThirdClass)
foundTrain[3]).thirdClassCapacity);
                index++;
            }

            displayTable(headers, data);
        }

        else {
            displayBox("No train found");
            System.exit(0);
        }

        System.out.println("\n");
        String trainNo = sc.nextLine();
        System.out.println("If you want to book a ticket enter 1 else enter 0");
        int bookCtrl = sc.nextInt();
        sc.nextLine();

        if (bookCtrl == 1) {
            System.out.println("\n");
            System.out.println("Enter the train number to book : ");
```

```
            trainNo = sc.nextLine();

            Booking booking = new Booking(currPer,trainNo,trains);//person



            FareCalculator fareFinal = new FareCalculator();// we are creating a new
object of fare calculator class to calculate the fare of the selected train



            for (Train[] tempObj : matchingTrains) {
               if (tempObj[0].train_no.equals(trainNo)) {
                  fareFinal.calculateFare(tempObj, start, end);
                  break;
               }
            }
            booking.book(matchingTrains, fareFinal.fare1, fareFinal.fare2,
fareFinal.fare3);


         } else {
            System.out.println("Thank you for using the system");
            System.exit(0);
         }


     for (Ticket ticket : currPer.ticketlist) {
        displayTicketBox(ticket.toString());
     }


    /* for (Train[] tempTrain : matchingTrains) {
        if (tempTrain[0].train_no.equals(trainNo)) {
           for (Person personObj : tempTrain[0].passengers) {
```

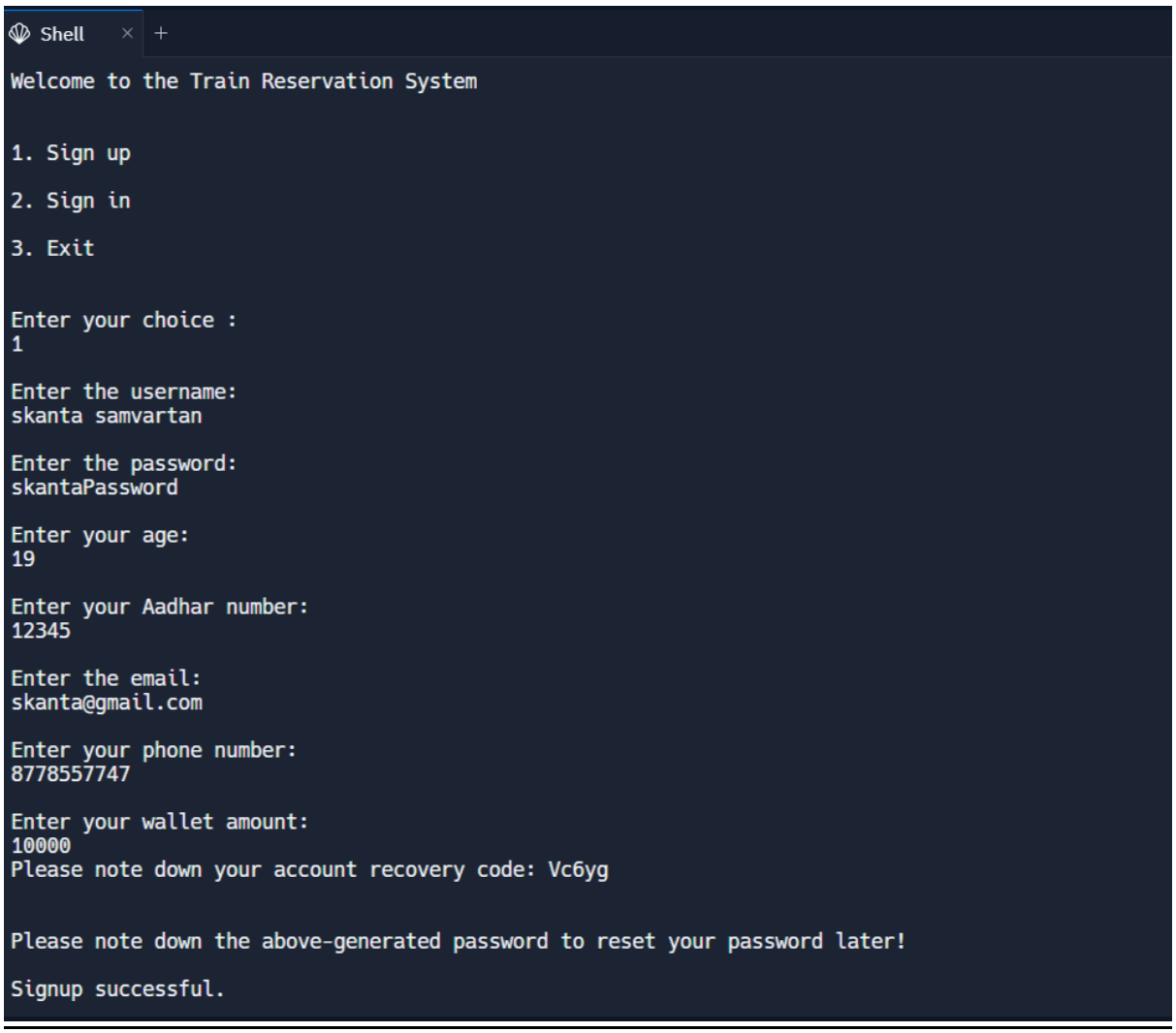
```
            for (Ticket ticket : personObj.ticketlist) {
                displayTickets(ticket.toString());
            }
        }
        break;  // Exit the loop after printing the tickets
    }
}*/

for (Train[] tempTrain : matchingTrains) {
if (tempTrain[0].train_no.equals(trainNo)) {
    for (Person personObj : tempTrain[0].passengers) {
        displayTickets(personObj.ticketlist);
    }
    break;  // Exit the loop after printing the tickets
}
}
Cancel cancelObj = new Cancel(currPer,trains);
cancelObj.cancel();

}
  }
```

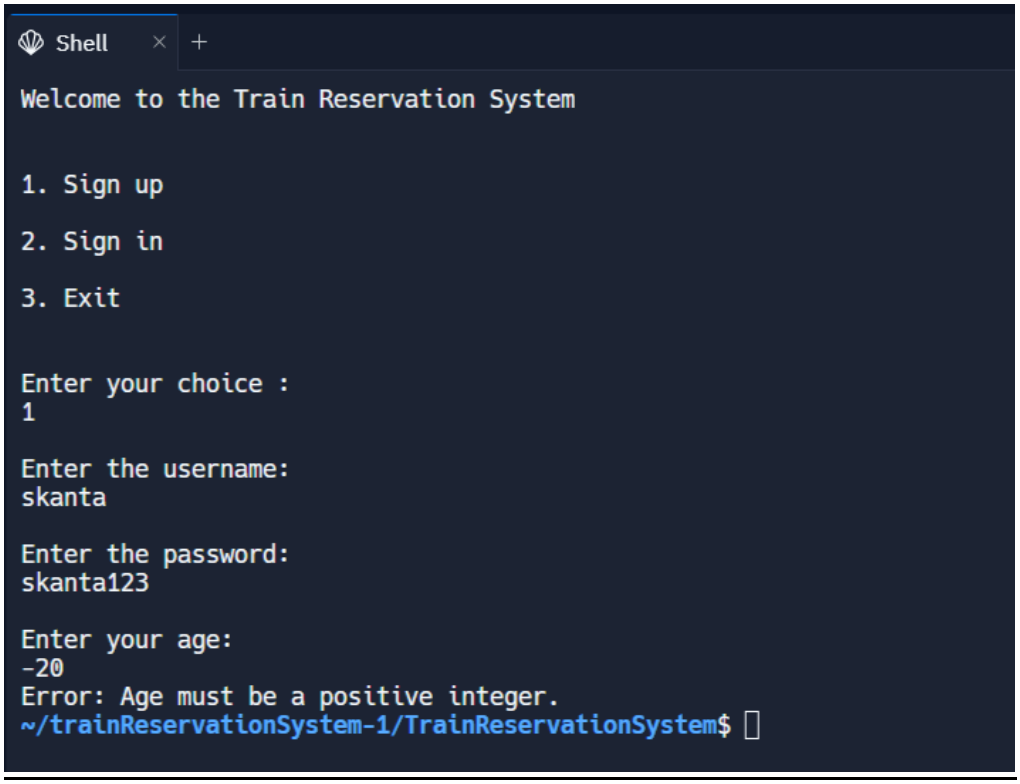
OUTPUT:

Output for Signup:

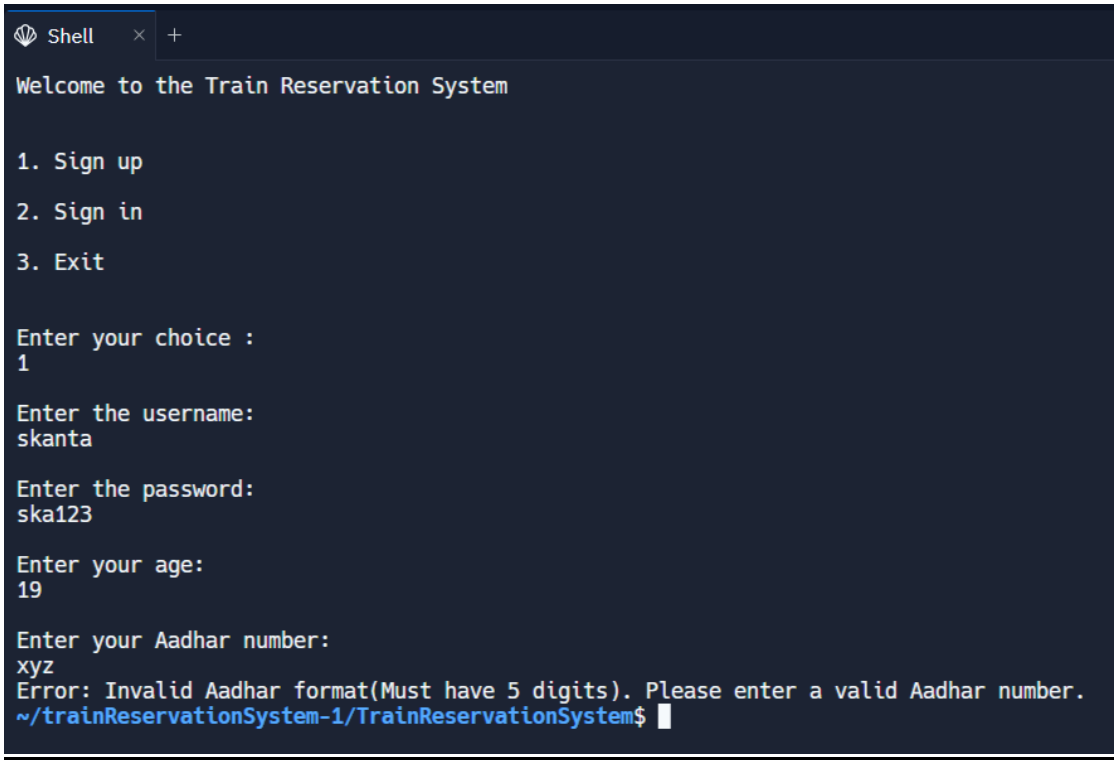


Output for few other test cases which handles some custom exceptions:

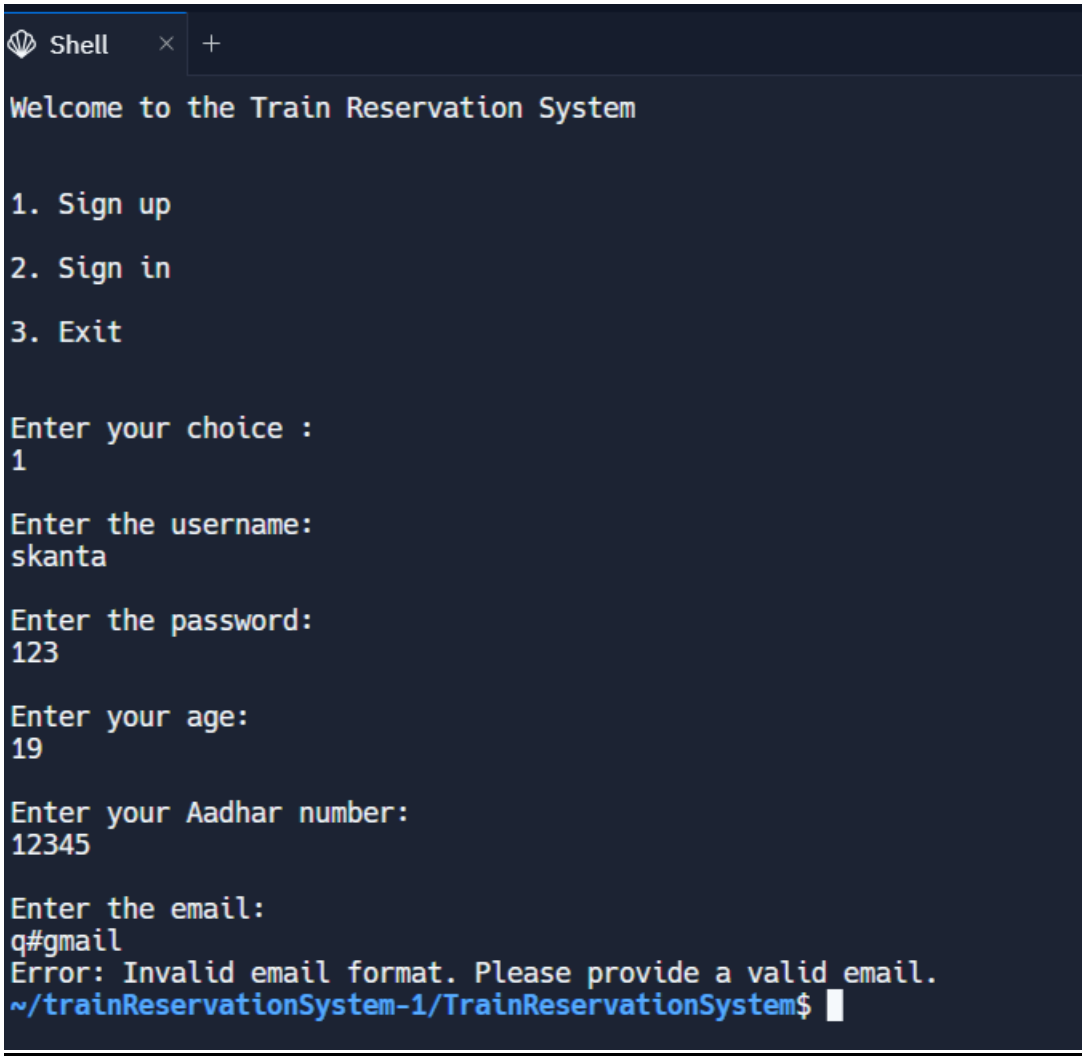
->Invalid age format:



->Invalid aadhar format:



->Invalid email format:



Output for login:

```
+-----------------------------------------------------------------------------------------------
Welcome to the Train Reservation System


1. Sign up

2. Sign in
Welcome to the Train Reservation System


1. Sign up

2. Sign in

3. Exit


Enter your choice :
2


**********************************
*********** LOGIN PAGE **********
**********************************

Enter the username: skanta samvartan
Enter the password: skantaPassword

**********************************
***** You Logged in successfully *
**********************************

Enter the From location :
chennai


Enter the To location :
salem
```

```
Train No  |Train Name           |Date                |First Class Fare  |Seats  |Second Class Fare   |Seats  |Third Class Fare    |Seats  |
----------+---------------------+--------------------+------------------+-------+--------------------+-------+--------------------+-------+
20643     |Vande Bharat         |2023-01-15T14:05:15 |733.8             |67     |433.79999999999995  |48     |353.79999999999995  |219    |
12623     |Trivandrum Express   |2023-01-15T19:45:10 |583.8             |25     |383.79999999999995  |29     |423.79999999999995  |239    |
22639     |Allapey Express      |2023-01-15T20:55:10 |683.8             |59     |433.79999999999995  |49     |393.79999999999995  |259    |
13351     |Azh  apuzha Express  |2023-01-15T23:15:10 |733.8             |100    |383.79999999999995  |29     |363.79999999999995  |250    |


If you want to book a ticket enter 1 else enter 0
1


Enter the train number to book :
20643
Enter the no of tickets to be reserved :
2


Enter the name of passenger :
sri
Enter the age of passenger :
18
Enter the gender of passenger :
male
Enter the category of passenger
1.Enter 1 for First class
2. Enter 2 for Second class
3. Enter 3 for Third class
1


Enter the name of passenger :
Hari
Enter the age of passenger :
19
Enter the gender of passenger :
male
```

```
male
Enter the category of passenger
1.Enter 1 for First class
2. Enter 2 for Second class
3. Enter 3 for Third class
2
Total fare : 1167.6


Money Available in wallet : 10000.0
Do you want to pay for the tickets using wallet ? (1 for yes /0 for no)
1
balance: 10000.0
Payment successful. Remaining balance: 8832.4
Payment successful
Ticket booked successfully


Balance Wallet Amount :8832.4
Seat category file updated successfully.
+-------------------------------------------------------------------------------------------------------------------
----------------+
|
Ticket Number: Vande BharatA67
Train Name: Vande Bharat
Train Number: 20643
Seat Category: 1
passenger Name :sri
Passenger Age: 18
Passenger Gender: male
Ticket Fare: 733.8 |
+-------------------------------------------------------------------------------------------------------------------
----------------+
+-------------------------------------------------------------------------------------------------------------------
------------------------------+
|
Ticket Number: Vande BharatB48
Train Name: Vande Bharat
Train Number: 20643
Seat Category: 2
```

```
Seat Category: 2
passenger Name :Hari
Passenger Age: 19
Passenger Gender: male
Ticket Fare: 433.79999999999995 |
+-------------------------------------------------------------------------------------------------------------------
------------------------------+
+-------------------------------------------------------------------------------------------------------------------
----------------+
|
Ticket Number: Vande BharatA67
Train Name: Vande Bharat
Train Number: 20643
Seat Category: 1
passenger Name :sri
Passenger Age: 18
Passenger Gender: male
Ticket Fare: 733.8 |
+-------------------------------------------------------------------------------------------------------------------
----------------+
+-------------------------------------------------------------------------------------------------------------------
------------------------------+
|
Ticket Number: Vande BharatB48
Train Name: Vande Bharat
Train Number: 20643
Seat Category: 2
passenger Name :Hari
Passenger Age: 19
Passenger Gender: male
Ticket Fare: 433.79999999999995 |
+-------------------------------------------------------------------------------------------------------------------
------------------------------+
Do you want to cancel the booking? (1 for yes / 0 for no)
1
Enter the train number to cancel the booking:
20643
Enter the number of seats to cancel:
2
Enter the ticket number to cancel the seat:
```

```
Enter the ticket number to cancel the seat:
Vande BharatB48
Booking canceled successfully!


Enter the ticket number to cancel the seat:
Vande BharatA67
Booking canceled successfully!
```

OOP FEATURES USED:

- ### Encapsulation
  - Encapsulation is a fundamental object-oriented programming (OOP) concept that involves bundling the data (attributes) and methods (behavior) that operate on the data within a single unit known as a class.
  - Purpose of Encapsulation:

  Data Hiding:

  - Encapsulation allows the hiding of the internal details and state of objects from the external world. In the Train Ticket Reservation System, this ensures that the internal representation of data, such as user credentials and ticket details, is not directly accessible outside the relevant classes.

  Modularity:

  - By encapsulating related attributes and methods within classes, the system becomes more modular. Each class represents a distinct entity, promoting ease of maintenance, readability, and updates.

  Enhanced Security:

  - Encapsulation contributes to improved security by restricting direct access to sensitive data. For example, user passwords and payment details are encapsulated within the appropriate classes, preventing unauthorized external access.

- Moreover since we use packages, we made use of 'protected' access specifier to access within the same package and to let no other packages access the data.

- Inheritance
  - In the Train Reservation System, the Train class has subclasses FirstClass, SecondClass, ThirdClass.
  - The train classes exhibit a logical hierarchy where specific types of trains (FirstClass, SecondClass, ThirdClass) are specialized versions of a more general train class (Train).
  - Base Class - Train:
    The Train class serves as the base class, containing shared attributes and methods applicable to all types of trains. It encapsulates general information about a train, including its name, number, date, list of stations and a list of passengers.
  - Derived Classes - FirstClass, SecondClass, ThirdClass:
    The specialized train classes (FirstClass, SecondClass, ThirdClass) inherit from the base Train class. Each derived class includes additional attributes specific to its type, such as firstClassCapacity, secondClassCapacity, and thirdClassCapacity, firstClassBaseFare, secondClassBaseFare, thirdClassBaseFare and their respective fine amounts.
- Polymorphism
  - In the Train Reservation System, we have made use of Polymorphism, through the means of method overriding and method overloading.
  - In Booking Package we have implemented WalletOperations interface which has makePayment() ,checkBalance(), displayWalletStatus() methods and these methods are overridden in Booking class.
- Exception Handling
  - The goal of exception handling is not just to catch the errors but also to provide some meaningful feedback to the user. In the Train ticket reservation system, we have made use of exception handling in various scenarios throughout the program and further we have the scope to add a lot more exceptions in the future. Few scenarios:

    i)Invalid Age Format:

To guarantee that users enter a valid age, the system checks for positive integer values. An `InvalidAgeException` is thrown if the age is not a positive integer or if an invalid format is detected during input.

ii) Invalid Aadhar Format:

For Aadhar number validation, the system expects a 5-digit format. The `InvalidAadharException` is triggered if the entered Aadhar number does not meet this criterion.

iii) Invalid Email Format:

Email validation is crucial for communication purposes. The system uses the `InvalidEmailException` to handle cases where users provide an email address that does not adhere to the standard format.

Throughout the program , file handling is sincerely taken care with the help of exception handling:

i)File I/O Issues:

During account creation, user information is stored in a file. The system is equipped to handle `IOExceptions` that may occur if there are problems opening or writing to the designated file.

ii) File Not Found Exception:

In the section where the program reads train details from files (trainDetails.txt, routes.txt, seatCategory.txt), there is a FileNotFoundException catch block that handles errors if the specified files are not found

iii) NullPointerException:

There is a risk of a NullPointerException when accessing objects that might not have been properly initialized. For example, if the trains ArrayList is not initialized, attempting to access it could lead to a NullPointerException.

iv) ArrayIndexOutOfBoundsException:

In the train details reading section, there might be an ArrayIndexOutOfBoundsException if the format of the input

files is incorrect. Ensure that the expected number of elements is present before accessing them.

- Packages
  - The motive of using packages in our project is to make use of the several benefits that it offers. Few such benefits are:
    1. Organizing and structuring
    2. Namespace Management
    3. Ease maintenance
    4. Code reusability
    5. Dependency management and enhanced readability
  - Organization of Packages in our software:
    - TrainReservationSystem
      - bookingPackage
        - Booking.java
        - FareCalculator.java
        - Ticket.java
        - WalletOperations.java
      - Cancellation
        - Cancel.java
      - Login
        - Login.java
        - Signup.java
        - LoginCredentials.txt
      - PersonPackage
        - Person.java
        - Passenger.java
        - Person.txt
      - SearchingPackage
        - Search.java
      - TrainPackage
        - ClassInfo.java
        - FirstClass.java
        - SecondClass.java
        - ThirdClass.java
        - Station.java
        - Train.java
      - Main.java
      - Routes.txt
      - SeatCategroy.txt
      - TrainDetails.txt

- Generics
  - Generics in Java provide a way to create classes, interfaces, and methods that can work with any data type, offering better type safety and code reuse.
  - We can add generics to our Program in the “booking” class Which ensures that the user can book tickets both using train name and train number.
    Here is the glimpse of it can be implemented:

```
public class Booking<T> implements WalletOperations, Serializable {
   // ...

   private T trainIdentifier;

   public Booking(Person person, T trainIdentifier, ArrayList<Train[]> Alltrains) {
      this.person = person;
      this.trainIdentifier = trainIdentifier;
      this.Alltrains = Alltrains;
   }

   // ... (rest of the class remains unchanged)

   public void book(ArrayList<Train[]> matchingTrains, double fare1, double fare2, double fare3) {
      int findCtrl = 0;
      int paymentCtrl = 0;

      for (Train[] tempObj : matchingTrains) {
         // Use the generic trainIdentifier to match against either train number or name
         if (tempObj[0].getTrainIdentifier().equals(trainIdentifier)) {
            train = tempObj;
            findCtrl = 1;
            break;
         }
      }
```

```
            // ... (rest of the method remains unchanged)
        }
    }
```

- Collections
  - ArrayList:
    The project extensively employs ArrayList from the Java Collections Framework to manage various data structures, including:
    i. ArrayList<Train[]> matchingTrains:
    This list stores all the filtered trains which consist of from and to location in their route.
    ii. ArrayList<Person> passengers:
    This list stores all the passengers. Each train object consists of this passengers list.
    iii. ArrayList<Ticket> ticketList:
    This list consists of all the tickets that a person has booked. Each person object consists of a ticketList.
  - LinkedList:
    The LinkedList data structure is utilized in the project, notably for:
    i. LinkedList<String> stationList:
    This list stores the names of stations, providing a flexible structure for managing station details.

FUTURE EXTENSION:

- Integration with Online Platforms:
  i. Enhancement: Integrate the system with online platforms to enable users to book and manage tickets through a web interface or mobile application.
  ii. Benefits:
    - Increased accessibility for users.
    - Real-time updates on train schedules and availability.
    - Integration with payment gateways for seamless transactions.

- User Authentication and Profiles:
    i. Enhancement: Implement user authentication and profile management features.
    ii. Benefits:
        - Personalized user experience.
        - Enhanced security for user data.

- Dynamic Fare Management:
    i. Enhancement: Implement a more sophisticated fare management system that considers factors like peak hours, seasons, and advanced booking.
    ii. Benefits:
        - Optimize pricing based on demand.
        - Incentivize off-peak travel.
        - Dynamic adjustment to market conditions.

- Seat Selection and Visualization:
    i. Enhancement: Provide users with the ability to choose their seats during the booking process, with a visual representation of the train layout.
    ii. Benefits:
        - Improved user experience.
        - Transparency in seat availability.
        - Customization options for passengers.

- Data Analytics and Reporting:
    i. Enhancement: Implement data analytics tools to analyze user behavior, popular routes, and booking trends.
    ii. Benefits:
        - Informed decision-making for route planning.
        - Marketing strategies based on user preferences.
        - Predictive maintenance for trains.