Phase 1

Flight Reservation System

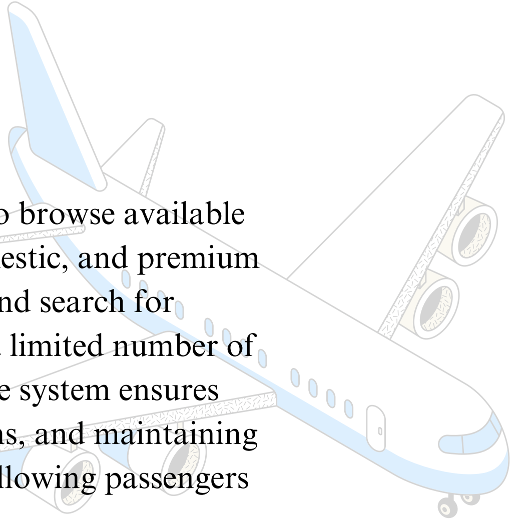
Group members:

Lama Almubarak (Leader)
Tala Alqahatni
Aljoharh Aldaej

The Division of work:

- Lama: Airline class and Reservation class, **created Admin Frame (GUI) , fixed runtime/compilation errors in main**
- Tala: Super class (Flight) and all Sub classes (DomesticFlight, BusinessDomesticFlight, InternationalFlight, PremiumInternationalFlight, and UML of report), **created exception class, Node class, and edited old methods in Airline class to suit node**
- Aljoharh: Main class (TestAirline) , **created Customer frame (GUI)**

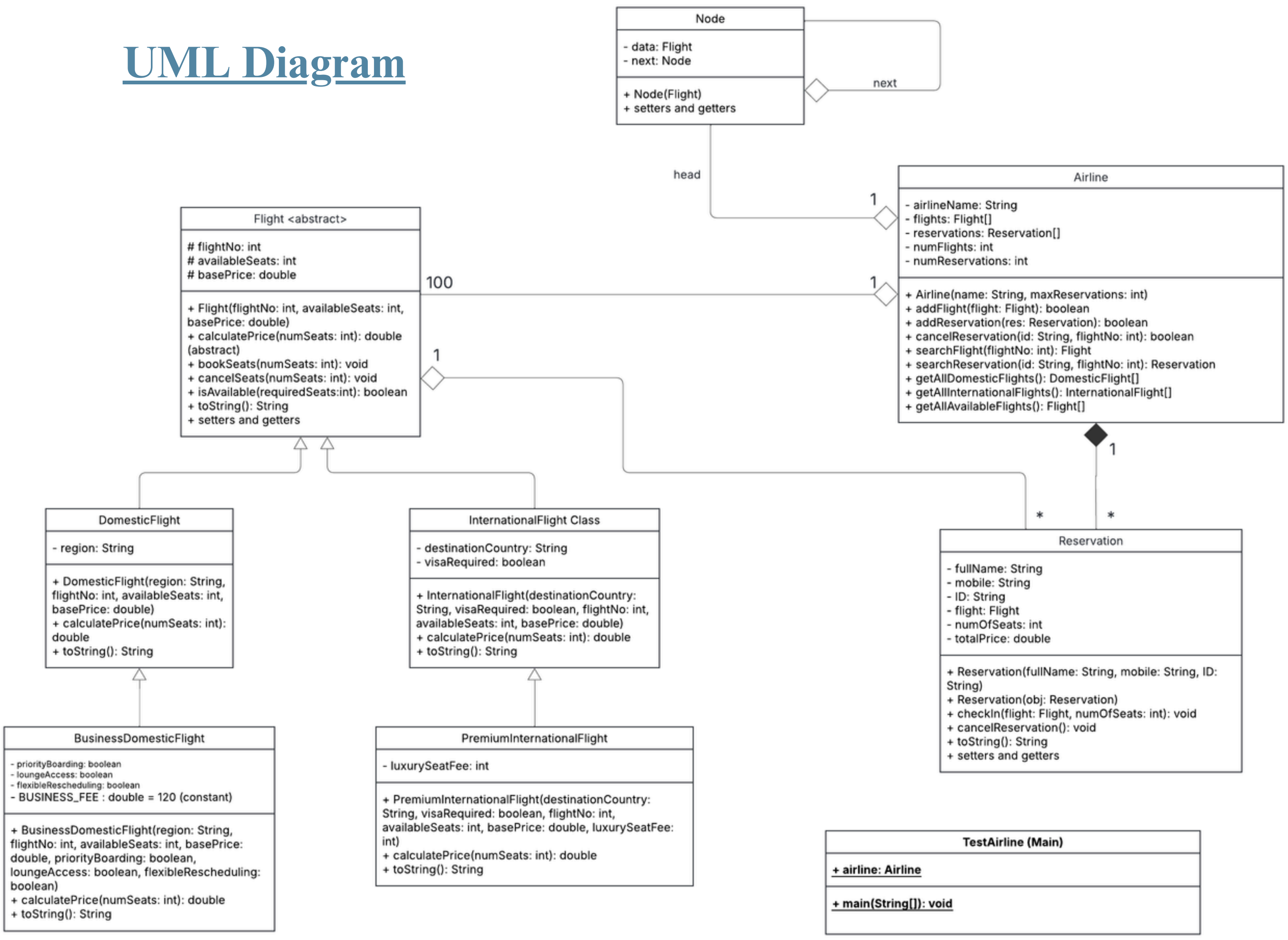
Introduction



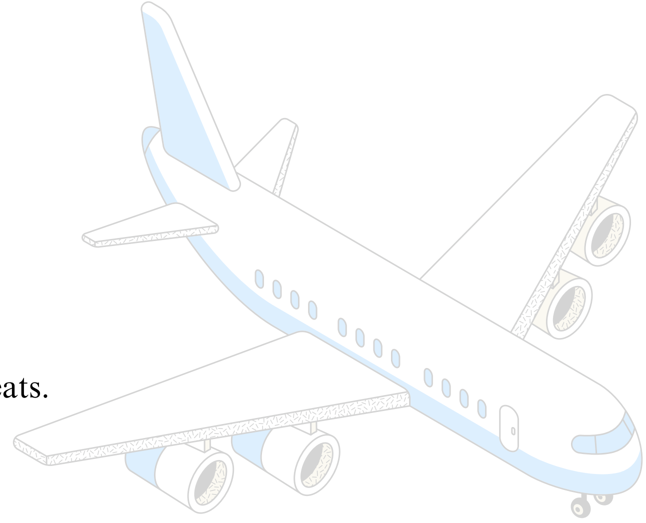
The program is an airline flight reservation system that allows users to browse available flights based on type, including domestic, international, business domestic, and premium international flights. Users can make reservations, cancel bookings, and search for reservations using a customer ID and flight number. Each flight has a limited number of available seats, and pricing may vary based on the flight category. The system ensures smooth booking operations by managing flights, handling reservations, and maintaining availability. It provides a structured approach to airline operations, allowing passengers to check in, book seats, and modify reservations efficiently.

Changes Made:

UML Diagram


Node
- data: Flight
- next: Node
+ Node(Flight)
+ setters and getters
next
head
1
Airline
- airlineName: String
- flights: Flight[]
- reservations: Reservation[]
- numFlights: int
- numReservations: int
+ Airline(name: String, maxReservations: int)
+ addFlight(flight: Flight): boolean
+ addReservation(res: Reservation): boolean
+ cancelReservation(id: String, flightNo: int): boolean
+ searchFlight(flightNo: int): Flight
+ searchReservation(id: String, flightNo: int): Reservation
+ getAllDomesticFlights(): DomesticFlight[]
+ getAllInternationalFlights(): InternationalFlight[]
+ getAllAvailableFlights(): Flight[]
Flight <abstract>
# flightNo: int
# availableSeats: int
# basePrice: double
+ Flight(flightNo: int, availableSeats: int, basePrice: double)
+ calculatePrice(numSeats: int): double (abstract)
+ bookSeats(numSeats: int): void
+ cancelSeats(numSeats: int): void
+ isAvailable(requiredSeats:int): boolean
+ toString(): String
+ setters and getters
100
1
1
1
*
*
DomesticFlight
- region: String
+ DomesticFlight(region: String, flightNo: int, availableSeats: int, basePrice: double)
+ calculatePrice(numSeats: int): double
+ toString(): String
InternationalFlight Class
- destinationCountry: String
- visaRequired: boolean
+ InternationalFlight(destinationCountry: String, visaRequired: boolean, flightNo: int, availableSeats: int, basePrice: double)
+ calculatePrice(numSeats: int): double
+ toString(): String
Reservation
- fullName: String
- mobile: String
- ID: String
- flight: Flight
- numOfSeats: int
- totalPrice: double
+ Reservation(fullName: String, mobile: String, ID: String)
+ Reservation(obj: Reservation)
+ checkIn(flight: Flight, numOfSeats: int): void
+ cancelReservation(): void
+ toString(): String
+ setters and getters
BusinessDomesticFlight
- priorityBoarding: boolean
- loungeAccess: boolean
- flexibleRescheduling: boolean
- BUSINESS_FEE : double = 120 (constant)
+ BusinessDomesticFlight(region: String, flightNo: int, availableSeats: int, basePrice: double, priorityBoarding: boolean, loungeAccess: boolean, flexibleRescheduling: boolean)
+ calculatePrice(numSeats: int): double
+ toString(): String
PremiumInternationalFlight
- luxurySeatFee: int
+ PremiumInternationalFlight(destinationCountry: String, visaRequired: boolean, flightNo: int, availableSeats: int, basePrice: double, luxurySeatFee: int)
+ calculatePrice(numSeats: int): double
+ toString(): String
TestAirline (Main)
+ airline: Airline
+ main(String[]): void

Description :



Class Flight (Abstract Class):

Attributes:

• flightNo: an integer that represents the flight number.
• availableSeats: an integer that stores the number of available seats.
• basePrice: a double that represents the base price of the flight.

Methods:

• Flight(int flightNo, int availableSeats, double basePrice): Constructor that initializes flight details with the received parameters.
• calculatePrice(int numSeats): Abstract method to calculate the total price based on the number of seats.
• bookSeats(int numSeats): Reserves seats if available.
• cancelSeats(int numSeats): Cancels a specified number of seats.
• isAvailable(int requestedSeats): Checks if the required number of seats is available.
• toString(): Returns flight details as a string.

Class DomesticFlight (Extends Flight):

Attributes:

• region: String that represents the region of the flight.

Methods:

• DomesticFlight(String region, int flightNo, int availableSeats, double basePrice): Constructor to initialize a domestic flight with the received parameters.
• calculatePrice(int numSeats): Calculates price based on domestic flight pricing rules.
• toString(): Returns details of the domestic flight.

Class BusinessDomesticFlight (Extends DomesticFlight):

Attributes:

• priorityBoarding: a boolean that indicates if priority boarding is available.
• loungeAccess: a boolean that determines whether lounge access is included.
• flexibleRescheduling: a boolean that specifies if rescheduling is flexible.
• BUSINESS_FEE: A constant of type (double) that represents a fee for business-class flights.

Methods:

• BusinessDomesticFlight(region: String, flightNo: int, availableSeats: int, basePrice: double, priorityBoarding: boolean, loungeAccess: boolean, flexibleRescheduling: boolean): Constructor that initializes business flight attributes.
• calculatePrice(int numSeats): Computes price with additional business-class fees.
• toString(): Returns details of the business domestic flight.

Description :

Class InternationalFlight (Extends Flight):

Attributes:

• destinationCountry: a String that represents the destination country.
• visaRequired: a boolean that indicates if a visa is needed.

Methods:

• InternationalFlight(destinationCountry: String, visaRequired: boolean, flightNo: int, availableSeats: int, basePrice: double): Constructor that initializes international flight details based on received parameter.
• calculatePrice(int numSeats): Computes total price based on international rates.
• toString(): Returns details of the international flight.

Class PremiumInternationalFlight (Extends InternationalFlight):

Attributes:

• luxurySeatFee: an integer that represents additional fees for premium seating.

Methods:

• PremiumInternationalFlight(destinationCountry: String, visaRequired: boolean, flightNo: int, availableSeats: int, basePrice: double, luxurySeatFee: int): Constructor initializing premium flight details.
• calculatePrice(int numSeats): Computes price with luxury seat fees included.
• toString(): Returns details of the premium international flight.

Class Airline:

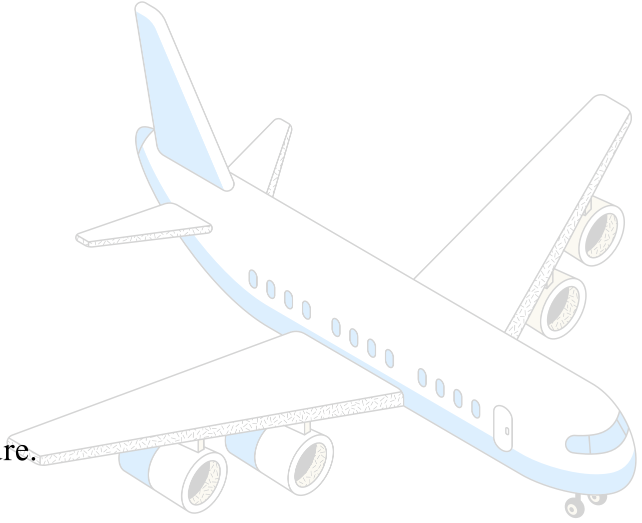
Attributes:

• airlineName: String that represents the name of the airline.
• reservations: Reservation[] - List of reservations.
• numFlights: an integer that represents number of available flights.
• numReservations: an integer that represents number of active reservations.

Methods:

• Airline(String name, int maxReservations): Constructor initializing the airline with the received parameters.
• addFlight(Flight f): Adds a flight to the airline.
• addReservation(Reservation res): Adds a reservation to the system.
• cancelReservation(String id, int flightNo): Cancels a reservation.
• searchFlight(int flightNo): Searches for a flight by number from received parameter.
• searchReservation(String id, int flightNo): Searches for a reservation.
• getAllDomesticFlights(): Returns all domestic flights.
• getAllInternationalFlights(): Returns all international flights.
• getAllAvailableFlights(): Returns flights with available seats.

Description :



Class Node (Implements Serializable):

Attributes:

• data: a Flight object that holds the data stored in this node.
• next: a Node reference pointing to the next node in the structure.

Methods:

• Node(obj: Flight): Constructor that initializes the node with a Flight object and sets the next reference to null.
• setNext(nextPtr: Node): Sets the reference to the next node.
• getNext(): Node: Returns the reference to the next node.
• setData(obj: Flight): Sets the Flight object stored in the node.
• getData(): Flight: Returns the Flight object stored in the node.

Class invalidMobileNum (Extends Exception):

Attributes:

Inherits attributes from the Exception class.

Methods:

• invalidMobileNum(s: String): Constructor that calls the superclass constructor with a custom error message describing the reason for the exception.

Interface InputOutputInterface:

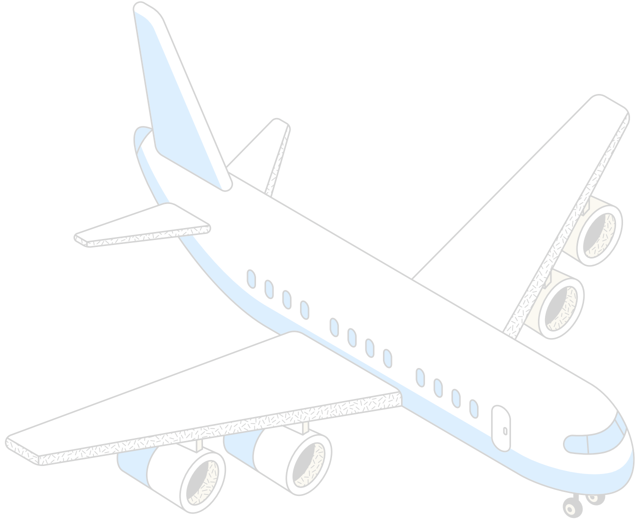
Attributes:

• fileOutput: a String constant that represents the filename where flight data is stored ("Flights.dat").

Methods:

• saveAllInformation(): Abstract method for saving all necessary information, likely to the file specified by fileOutput.
• readAllData(): Abstract method for reading data from the file and restoring it into the program.

Description :



Class Reservation:

Attributes:

• fullName: a string with Passenger's full name.
• mobile: a string of the contact number of the passenger.
• ID: a string of the passenger's ID
• flight: Flight object that references to the booked flight.
• numOfSeats: an integer that holds number of seats reserved.
• totalPrice: a double that represents the total cost of the reservation.

Methods:

• Reservation(String fullName, String mobile, String ID): Constructor to initialize reservation details with the received parameters.
• checkIn(Flight flight, int numOfSeats): Assigns the flight and number of seats.
• cancelReservation(): Cancels the reservation and releases seats.
• toString(): Returns reservation details as a string.

TestAirline Class:

Attributes:

• airline: An Airline object representing the airline, containing flights and reservations.

Methods:

- main(String[] args)

Creates different flights, adds them to the airline, and displays a menu for user interaction.

Code :

```
 package java2p;

public abstract class Flight {
  protected int flightNo;
  protected int availableSeats;
  protected double basePrice;

   public Flight() {};
     public Flight(int flightNo, int availableSeats, double basePrice) {
     this.flightNo = flightNo;
       this.availableSeats = availableSeats;
       this.basePrice = basePrice ;
       }

     public abstract double calculatePrice(int numSeats) ;

     public void bookSeats(int numSeats) {
      availableSeats -= numSeats;
      }

     public void cancelSeats(int numSeats) {
      availableSeats += numSeats;
      }

  public boolean isAvailable(int requestedSeats) {
      return availableSeats >= requestedSeats; // Returns true if seats are available,
otherwise false
  }

     public String toString() {
      return "Flight Number: " + flightNo + ", Available Seats: " + availableSeats;
      }

     //Setters and getters
  public int getFlightNo() {
   return flightNo;
  }

  public void setFlightNo(int flightNo) {
   this.flightNo = flightNo;
  }
 public int getAvailableSeats() {
 return availableSeats;
 }

 public void setAvailableSeats(int availableSeats) {
 this.availableSeats = availableSeats;
 }

 public double getbasePrice() {
 return basePrice;
 }

 public void setbasePrice(double basePrice) {
 this.basePrice = basePrice;
 }

}
------------------------------------------------------------------------------------------

package java2p;

public class DomesticFlight extends Flight{
   private String region;


   public DomesticFlight(String region, int flightNo, int availableSeats, double basePrice)
{
      super(flightNo, availableSeats, basePrice);
      this.region = region;
   }

   public double calculatePrice(int numSeats) {
      return basePrice * numSeats;
   }


   public String toString() {
      return super.toString() + ", Region: " + region;
   }

}
------------------------------------------------------------------------------------------
```

```
package java2p;

public class BusinessDomesticFlight extends DomesticFlight {
 private boolean priorityBoarding;
   private boolean loungeAccess;
   private boolean flexibleRescheduling;
   private static final double BUSINESS_FEE = 120; // Extra fee for business perks

   public BusinessDomesticFlight(String region, int flightNo, int availableSeats, double
basePrice,
                          boolean priorityBoarding, boolean loungeAccess, boolean
flexibleRescheduling) {
       super(region, flightNo, availableSeats, basePrice);
       this.priorityBoarding = priorityBoarding;
       this.loungeAccess = loungeAccess;
       this.flexibleRescheduling = flexibleRescheduling;
   }

   public double calculatePrice(int numSeats) {
       double total = super.calculatePrice(numSeats);

       // Business-Class Perks Increase Ticket Price
       if (priorityBoarding)
        total += 20 * numSeats;
       if (loungeAccess)
        total += 50 * numSeats;
       if (flexibleRescheduling)
        total += 80 * numSeats;

       return total + (BUSINESS_FEE * numSeats);
   }

   public String toString() {
       return super.toString() + ", Business Perks: " +
            (priorityBoarding ? "Priority Boarding, " : "") +
            (loungeAccess ? "Lounge Access, " : "") +
            (flexibleRescheduling ? "Flexible Rescheduling" : "Standard Ticket");
   }
}
```

```
package java2p;

public class InternationalFlight extends Flight {
   private String destinationCountry;
    private boolean visaRequired;

    public InternationalFlight(String destinationCountry, boolean visaRequired, int
flightNo, int availableSeats, double basePrice) {
        super(flightNo, availableSeats, basePrice);
        this.destinationCountry = destinationCountry;
        this.visaRequired = visaRequired;
    }

    public double calculatePrice(int numSeats) {
        double total = basePrice * numSeats;
        if (visaRequired) total += 100 * numSeats; // Visa processing fee per seat
        return total;
    }

    public String toString() {
        return super.toString() + ", Destination: " + destinationCountry + ", Visa Required:
" + visaRequired;
    }
}
-----------------------------------------------------------------------------------------
package java2p;

public class PremiumInternationalFlight extends InternationalFlight {
    private int luxurySeatFee;

    public PremiumInternationalFlight(String destinationCountry, boolean visaRequired,
int flightNo, int availableSeats, double basePrice, int luxurySeatFee) {
        super(destinationCountry, visaRequired, flightNo, availableSeats, basePrice);
        this.luxurySeatFee = luxurySeatFee;
    }

    public double calculatePrice(int numSeats) {
        return super.calculatePrice(numSeats) + (luxurySeatFee * numSeats);
    }

    public String toString() {
        return super.toString() + ", Luxury Seat Fee: $" + luxurySeatFee;
    }
}
```

```
package java2p;

public class Reservation {
   private String fullName;
   private String mobile;
   private String ID;
   private Flight flight;
   private int numOfSeats;
   private double totalPrice;

   public Reservation(String fullName, String mobile, String ID) {
      this.fullName = fullName;
      this.mobile = mobile;
      this.ID = ID;
   }

    //Copy constructor
   public Reservation(Reservation obj) {
      this.fullName = obj.fullName;
      this.mobile = obj.mobile;
      this.ID = obj.ID;
      this.numOfSeats = obj.numOfSeats;
      this.flight = obj.flight;
      this.totalPrice = obj.totalPrice;
   }

   // Check-in method to book a flight
   public void checkIn(Flight flight, int numOfSeats) {
      this.flight = flight;
      this.numOfSeats = numOfSeats;
      flight.setAvailableSeats(flight.getAvailableSeats() - numOfSeats);
      this.totalPrice = flight.calculatePrice(numOfSeats);
   }

   public void cancelReservation() {
      if (flight != null) {
         flight.setAvailableSeats(flight.getAvailableSeats() + numOfSeats);
         flight = null;
         numOfSeats = 0;
         totalPrice = 0;
         System.out.println("Reservation has been successfully canceled.");
      }
   }
```

```
@Override
public String toString() {
return "Full Name: " + fullName + "\nMobile: " + mobile + "\nID: " + ID +"\n" +(flight
!= null ? flight.toString() : "No Flight Booked") + "\nNumber of Seats: " + numOfSeats +
"\nTotal Price: " + totalPrice;
}

// Getters and Setters
public String getFullName() {
return fullName;
}

public void setFullName(String fullName) {
this.fullName = fullName;
}

public String getMobile() {
return mobile;
}

public void setMobile(String mobile) {
this.mobile = mobile;
}

public String getID() {
return ID;
}

public void setID(String ID) {
this.ID = ID;
}

public Flight getFlight() {
return flight;
}

public void setFlight(Flight flight) {
this.flight = flight;
}

public int getNumOfSeats() {
return numOfSeats;
}
```

```
 public void setNumOfSeats(int numOfSeats) {
 this.numOfSeats = numOfSeats;
 }

 public double getTotalPrice() {
 return totalPrice;
 }

 public void setTotalPrice(double totalPrice) {
 this.totalPrice = totalPrice;
 }

}
------------------------------------------------------------------------------------------

package java2p;

public class Airline {
   private String airlineName;
   private Flight[] flightList;
   private Reservation[] reservations;
   private int numFlights, numReservations;

   public Airline(String name, int maxReservations) {
      airlineName = name;
      flightList = new Flight[100];
      numFlights = 0;
      reservations = new Reservation[maxReservations];
      numReservations = 0;
   }

   public boolean addFlight(Flight flight) {
      if (numFlights < flightList.length) { // Aggregation
         flightList[numFlights] = flight;
         numFlights++;
         return true;
      }
      return false;
   }
 public boolean addReservation(Reservation res) {
 if (numReservations < reservations.length) { // Composition
 reservations[numReservations] = new Reservation(res);
 numReservations++;
 return true;
 }
 return false;
 }

 public boolean cancelReservation(String id, int flightNo) {
 for (int i = 0; i < numReservations; i++) {
 if (reservations[i].getID().equals(id) && reservations[i].getFlight().getFlightNo() ==
flightNo) {
 reservations[i].cancelReservation();
 reservations[i] = reservations[numReservations - 1];
 numReservations--;
 reservations[numReservations] = null;
 return true;
 }
 }
 return false;
 }

   public Reservation searchReservation(String id, int flightNo) {
       for (int i = 0; i < numReservations; i++) {
           if (reservations[i].getID().equals(id) && reservations[i].getFlight().getFlightNo()
== flightNo) {
               return reservations[i];
           }
       }
       return null;
   }

    public Flight searchFlight(int flightNo) {
       for (int i = 0; i < numFlights; i++) {
           if (flightList[i].getFlightNo() == flightNo) {
               return flightList[i];
           }
       }
       return null;
    }
 public DomesticFlight[] getAllDomesticFlights() {
 DomesticFlight[] list = new DomesticFlight[numFlights];
 int j = 0;
 for (int i = 0; i < numFlights; i++) {
 if (flightList[i] instanceof DomesticFlight) {
 list[j++] = (DomesticFlight) flightList[i];
 }
 }
 return list;
 }

 public InternationalFlight[] getAllInternationalFlights() {
 InternationalFlight[] list = new InternationalFlight[numFlights];
 int j = 0;
 for (int i = 0; i < numFlights; i++) {
 if (flightList[i] instanceof InternationalFlight) {
 list[j++] = (InternationalFlight) flightList[i];
 }
 }
 return list;
 }

  public Flight[] getAllAvailableFlights() {
     Flight[] list = new Flight[numFlights];
      int j = 0;
      for (int i = 0; i < numFlights; i++) {
         if (flightList[i].isAvailable(1)) {
             list[j++] = flightList[i];
          }
       }
       return list;
   }
}
```

```
package java2p;

import java.util.Scanner;

public class TestAirline {
   static Scanner input = new Scanner(System.in);
   static Airline airline = new Airline("Saudia", 1000);
```

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

  DomesticFlight DF1 = new DomesticFlight ("Riyadh" , 101 , 250 , 800);
  DomesticFlight DF2 = new DomesticFlight ("Jeddah" , 102 , 200 , 650);
  DomesticFlight DF3 = new DomesticFlight ("Dammam" , 103 , 300 , 600);

    InternationalFlight IF1 = new InternationalFlight("Rome" , true , 201 , 400 , 3400);
    InternationalFlight IF2 = new InternationalFlight("Tokyo" , true , 202 , 300 , 4000);
    InternationalFlight IF3 = new InternationalFlight("Tbilsi" , false , 203 , 400 , 2800);

    BusinessDomesticFlight BIF1 = new BusinessDomesticFlight("Abha", 104, 150, 1000, true,
true, true );
    BusinessDomesticFlight BIF2 = new BusinessDomesticFlight("AL Madinah", 105, 100, 800,
true, true, true );

    PremiumInternationalFlight PIF1 = new  PremiumInternationalFlight ("New York" , true ,
204 , 30 , 12000 , 8000);
    PremiumInternationalFlight PIF2 = new  PremiumInternationalFlight ("Paris" , true , 205 ,
20 , 16000 , 9000);

    airline.addFlight(DF1);
    airline.addFlight(DF2);
    airline.addFlight(DF3);

    airline.addFlight(IF1);
    airline.addFlight(IF2);
    airline.addFlight(IF3);

    airline.addFlight(BIF1);
    airline.addFlight(BIF2);

    airline.addFlight(PIF1);
    airline.addFlight(PIF2);

int choice;
      do {
         System.out.println("***********MENU***********");
         System.out.println("1- Book a flight ticket ");
         System.out.println("2- View flights");
         System.out.println("3- Cancel a booking");
         System.out.println("4- Search for a booking");
         System.out.println("5- Exit");
         choice = input.nextInt();
```

```
        switch (choice) {
            case 1:
                addNewReservation();
                break;
            case 2:
                viewFlights();
                break;
            case 3:
                cancelReservation();
                break;
            case 4:
                searchReservation();
                break;
            case 5:
                System.out.println("**** Goodbye! ****");
                break;
            default:
                System.out.println("Invalid input, try again.");
        }
    } while (choice != 5);
}

 public static void addNewReservation() {
   System.out.println("-------------------------");
     System.out.println("Enter your choice : ");
System.out.println("1- Add a reservation: ");
System.out.println("2- Return to main menu ");
int c = input.nextInt();
switch(c){

  case 1:
    System.out.println("Enter flight number: ");
    int flightNo = input.nextInt();
    Flight flightObj = airline.searchFlight(flightNo);

    int seat = 0;
    if (flightObj == null || !flightObj.isAvailable(seat)) {
       System.out.println("This flight is not available, try again.");
       return;
    }

    System.out.println("Enter your Full name : ");
    input.nextLine();
    String name = input.nextLine();
```

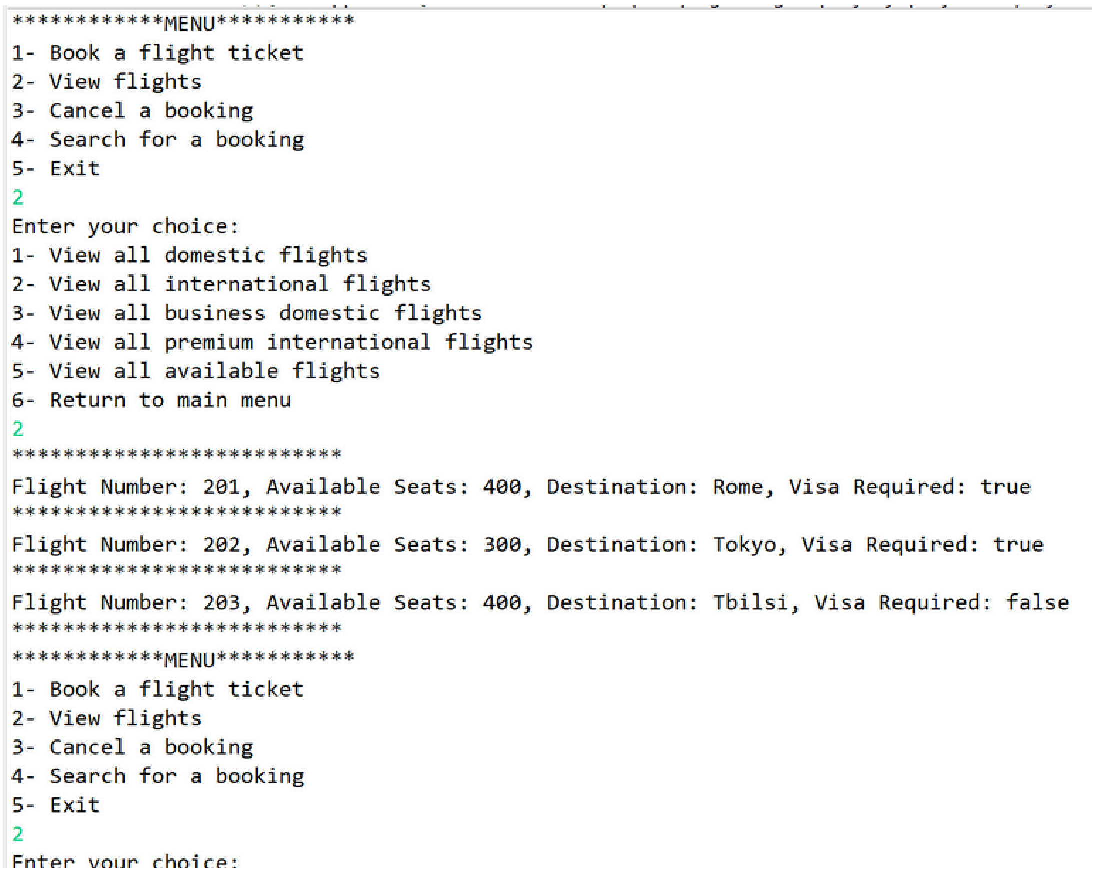
```
// Validate mobile number
        String mobile;
        boolean isValid;
        do {
            System.out.println("Enter your Mobile number (10 digit): ");
            mobile = input.next();
            isValid = true;

            // Check length
            if (mobile.length() != 10) {
                isValid = false;
                System.out.println("Invalid mobile number");
            } else {
                // Check if all characters are digits
                for (int i = 0; i < mobile.length(); i++) {
                    if (!Character.isDigit(mobile.charAt(i))) {
                        isValid = false;
                        System.out.println("Invalid mobile number");
                        break;
                    }
                }
            }
        } while (!isValid);

     // Validate ID
        String id;
        do {
            System.out.println("Enter customer ID (10 digit): ");
            id = input.next();
            isValid = true;

            // Check length
            if (id.length() != 10) {
                isValid = false;
                System.out.println("Invalid ID");
            } else {
                // Check if all characters are digits
                for (int i = 0; i < id.length(); i++) {
                    if (!Character.isDigit(id.charAt(i))) {
                        isValid = false;
                        System.out.println("Invalid ID");
                        break;
                    }
    }
    }
    } while (!isValid);
        System.out.println("Enter number of seats to book: ");
        int seats = input.nextInt();

        if (seats > flightObj.getAvailableSeats()) {
            System.out.println("Not enough seats available.");
            return;
        }

        Reservation res = new Reservation(name, mobile, id);
        res.checkIn(flightObj, seats);
        airline.addReservation(res);

        System.out.println("The seats have been successfully booked , We wish you a pleasent
trip!");
        System.out.println("*********************************");
        System.out.println(res.toString());
        System.out.println("*********************************");
        break ;
     case 2 :
      System.out.println("Returning to main menu...");
      break;
      default :
      System.out.println("invalid input , try again");
     }
      }

      public static void viewFlights() {
        System.out.println("Enter your choice: ");
        System.out.println("1- View all domestic flights");
        System.out.println("2- View all international flights");
        System.out.println("3- View all business domestic flights");
        System.out.println("4- View all premium international flights");
        System.out.println("5- View all available flights");
        System.out.println("6- Return to main menu");
        int choice = input.nextInt();

        switch (choice) {
        case 1:
            DomesticFlight[] domesticFlights = airline.getAllDomesticFlights();
            System.out.println("*************************");
```

```
for (int i = 0; i < domesticFlights.length; i++) {
 if (domesticFlights[i] != null && !(domesticFlights[i] instanceof BusinessDomesticFlight)) {
 System.out.println(domesticFlights[i].toString());
 System.out.println("*************************");
 }
 }
 break;

       case 2:
          InternationalFlight[] internationalFlights = airline.getAllInternationalFlights();
          System.out.println("*************************");
          for (int i = 0; i < internationalFlights.length; i++) {
             if (internationalFlights[i] != null && !(internationalFlights[i] instanceof
PremiumInternationalFlight)) {
                System.out.println(internationalFlights[i].toString());
                System.out.println("*************************");
             }
          }
          break;

       case 3:
          System.out.println("***** Business Domestic Flights *****");
          DomesticFlight[] allDomestic = airline.getAllDomesticFlights();
          for (int i = 0; i < allDomestic.length; i++) {
             if (allDomestic[i] instanceof BusinessDomesticFlight) {
                System.out.println(allDomestic[i].toString());
                System.out.println("*************************");
             }
          }
          break;

       case 4:
          System.out.println("***** Premium International Flights *****");
          InternationalFlight[] allInternational = airline.getAllInternationalFlights();
          for (int i = 0; i < allInternational.length; i++) {
             if (allInternational[i] instanceof PremiumInternationalFlight) {
                System.out.println(allInternational[i].toString());
                System.out.println("*************************");
             }
          }
          break;
       case 5:
         Flight[] availableFlights = airline.getAllAvailableFlights();
         System.out.println("*************************");
```

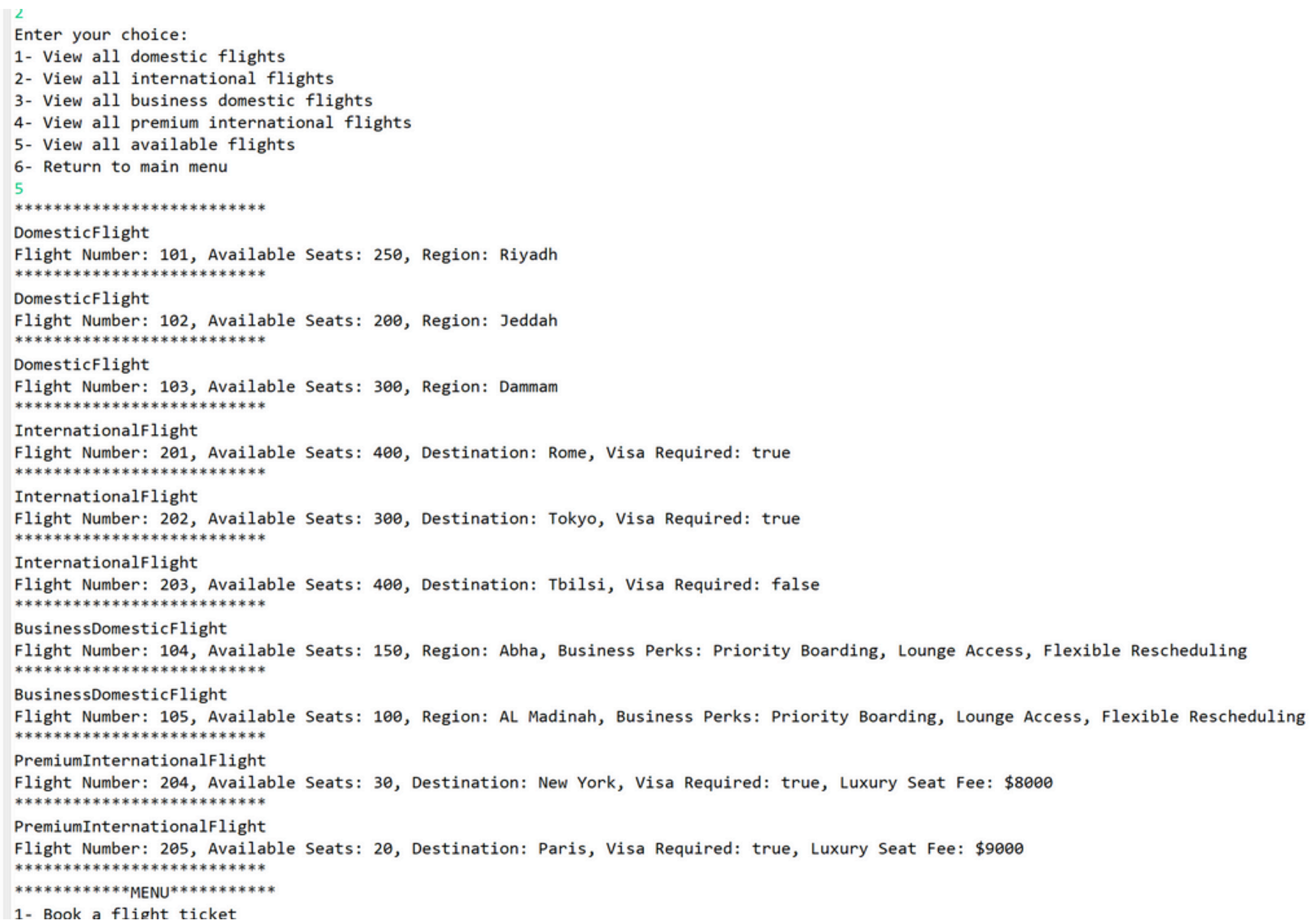
```
        System.out.println("*************************");
          for (int i = 0; i < availableFlights.length; i++) {
             if (availableFlights[i] != null) {
                System.out.println(availableFlights[i].getClass().getSimpleName());
                System.out.println(availableFlights[i].toString());
                System.out.println("*************************");
             }
          }
          break;

       case 6:
          System.out.println("Returning to main menu...");
          break;

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

    public static void cancelReservation() {
      System.out.println("-------------------------");
      System.out.println("Enter your choice : ");
   System.out.println("1- Cancel reservation : ");
   System.out.println("2- Return to main menu ");
   int ch2 = input.nextInt();
   switch(ch2){
   case 1 :
       System.out.println("Enter customer ID: ");
       String id = input.next();
       System.out.println("Enter flight number: ");
       int flightNo = input.nextInt();

       if (airline.cancelReservation(id, flightNo)) {
       } else
          System.out.println("Cancellation failed.... Reservation not found");
       break ;
   case 2 :
    System.out.println("Returning to main menu...");
    break;
   default :
   System.out.println("invalid input , try again");
   }
    }
```

```
public static void searchReservation() {
System.out.println("-------------------------");
System.out.println("Enter your choice : ");
System.out.println("1- Search reservation : ");
System.out.println("2- Return to main menu ");
int ch2 = input.nextInt();
switch(ch2){
case 1 :
System.out.println("Enter customer ID: ");
String id = input.next();
System.out.println("Enter flight number: ");
int flightNo = input.nextInt();

Reservation res = airline.searchReservation(id, flightNo);
if (res == null) {
System.out.println("Reservation not found.");
} else {
System.out.println(res.toString());
System.out.println("---------------------");
}
break ;
case 2 :
System.out.println("Returning to main menu...");
break;
default :
System.out.println("invalid input , try again");
}
}
}
```

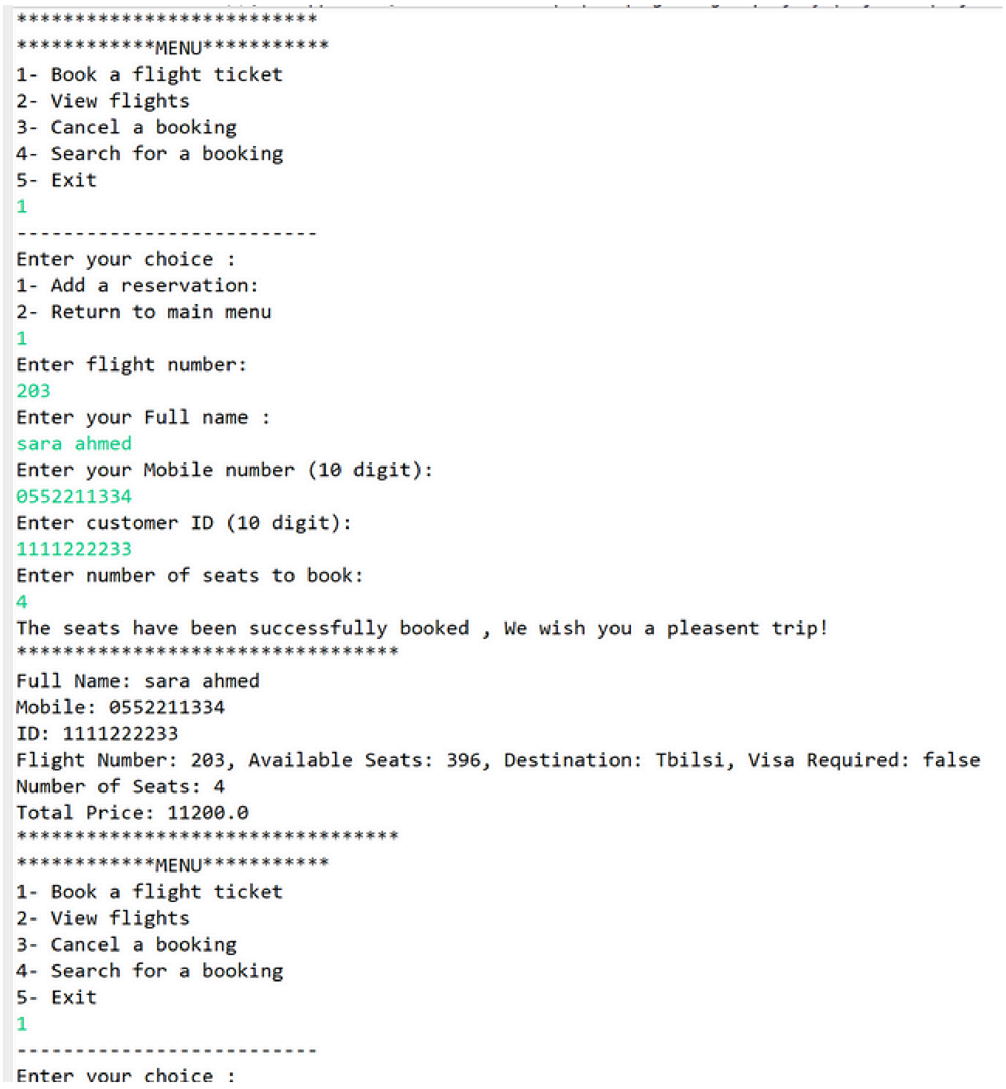
Screen shot of sample Run :



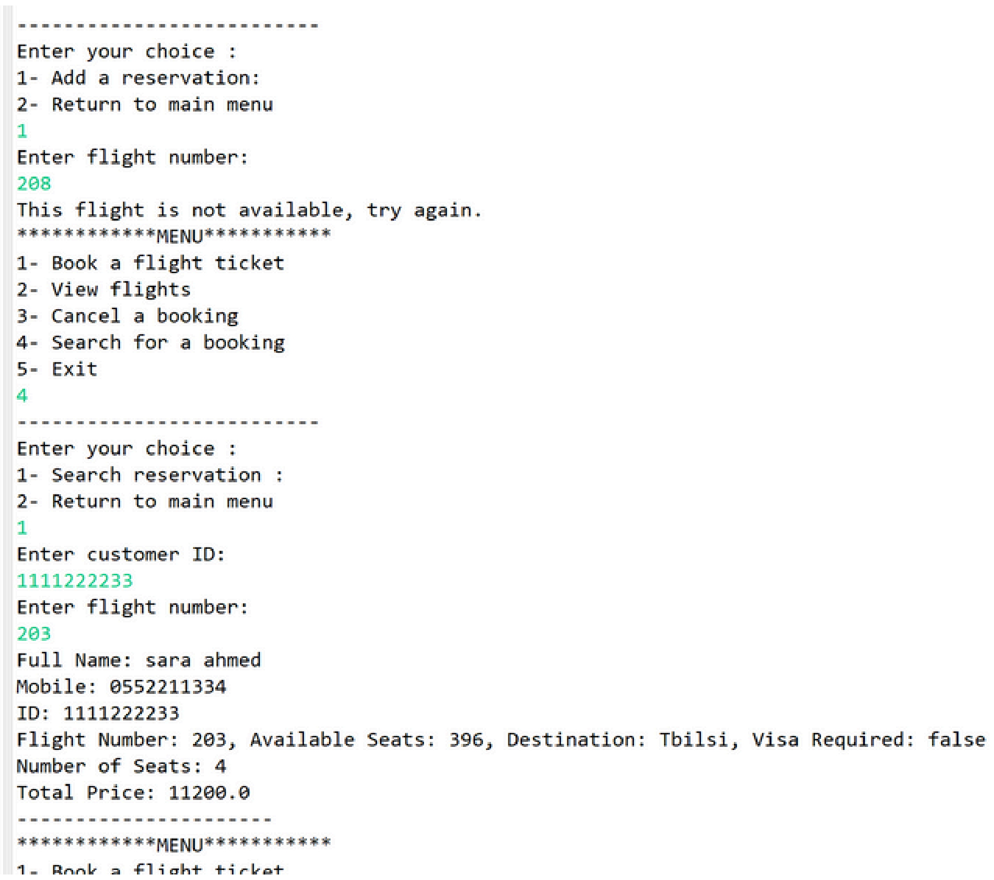
```
************MENU***********
1- Book a flight ticket
2- View flights
3- Cancel a booking
4- Search for a booking
5- Exit
2
Enter your choice:
1- View all domestic flights
2- View all international flights
3- View all business domestic flights
4- View all premium international flights
5- View all available flights
6- Return to main menu
2
**************************
Flight Number: 201, Available Seats: 400, Destination: Rome, Visa Required: true
**************************
Flight Number: 202, Available Seats: 300, Destination: Tokyo, Visa Required: true
**************************
Flight Number: 203, Available Seats: 400, Destination: Tbilsi, Visa Required: false
**************************
************MENU***********
1- Book a flight ticket
2- View flights
3- Cancel a booking
4- Search for a booking
5- Exit
2
Enter your choice:
```



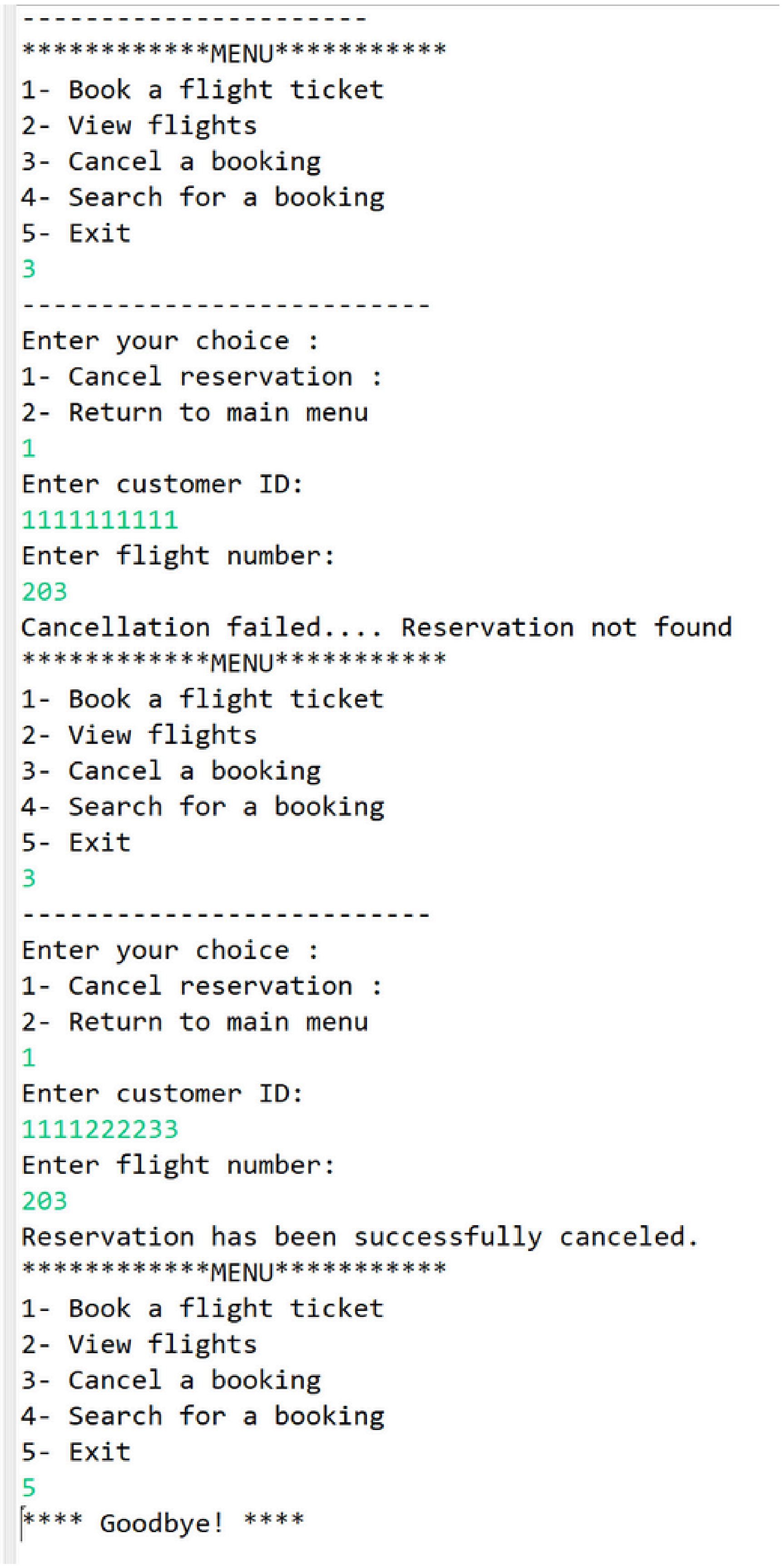
```
2
Enter your choice:
1- View all domestic flights
2- View all international flights
3- View all business domestic flights
4- View all premium international flights
5- View all available flights
6- Return to main menu
5
**************************
DomesticFlight
Flight Number: 101, Available Seats: 250, Region: Riyadh
**************************
DomesticFlight
Flight Number: 102, Available Seats: 200, Region: Jeddah
**************************
DomesticFlight
Flight Number: 103, Available Seats: 300, Region: Dammam
**************************
InternationalFlight
Flight Number: 201, Available Seats: 400, Destination: Rome, Visa Required: true
**************************
InternationalFlight
Flight Number: 202, Available Seats: 300, Destination: Tokyo, Visa Required: true
**************************
InternationalFlight
Flight Number: 203, Available Seats: 400, Destination: Tbilsi, Visa Required: false
**************************
BusinessDomesticFlight
Flight Number: 104, Available Seats: 150, Region: Abha, Business Perks: Priority Boarding, Lounge Access, Flexible Rescheduling
**************************
BusinessDomesticFlight
Flight Number: 105, Available Seats: 100, Region: AL Madinah, Business Perks: Priority Boarding, Lounge Access, Flexible Rescheduling
**************************
PremiumInternationalFlight
Flight Number: 204, Available Seats: 30, Destination: New York, Visa Required: true, Luxury Seat Fee: $8000
**************************
PremiumInternationalFlight
Flight Number: 205, Available Seats: 20, Destination: Paris, Visa Required: true, Luxury Seat Fee: $9000
**************************
************MENU***********
1- Book a flight ticket
```



```
**************************
************MENU***********
1- Book a flight ticket
2- View flights
3- Cancel a booking
4- Search for a booking
5- Exit
1
--------------------------
Enter your choice :
1- Add a reservation:
2- Return to main menu
1
Enter flight number:
203
Enter your Full name :
sara ahmed
Enter your Mobile number (10 digit):
0552211334
Enter customer ID (10 digit):
1111222233
Enter number of seats to book:
4
The seats have been successfully booked , We wish you a pleasent trip!
*********************************
Full Name: sara ahmed
Mobile: 0552211334
ID: 1111222233
Flight Number: 203, Available Seats: 396, Destination: Tbilsi, Visa Required: false
Number of Seats: 4
Total Price: 11200.0
*********************************
************MENU***********
1- Book a flight ticket
2- View flights
3- Cancel a booking
4- Search for a booking
5- Exit
1
--------------------------
Enter your choice :
```



```
--------------------------
Enter your choice :
1- Add a reservation:
2- Return to main menu
1
Enter flight number:
208
This flight is not available, try again.
************MENU***********
1- Book a flight ticket
2- View flights
3- Cancel a booking
4- Search for a booking
5- Exit
4
--------------------------
Enter your choice :
1- Search reservation :
2- Return to main menu
1
Enter customer ID:
1111222233
Enter flight number:
203
Full Name: sara ahmed
Mobile: 0552211334
ID: 1111222233
Flight Number: 203, Available Seats: 396, Destination: Tbilsi, Visa Required: false
Number of Seats: 4
Total Price: 11200.0
----------------------
************MENU***********
1- Book a flight ticket
```



```
----------------------
***********MENU***********
1- Book a flight ticket
2- View flights
3- Cancel a booking
4- Search for a booking
5- Exit
3
--------------------------
Enter your choice :
1- Cancel reservation :
2- Return to main menu
1
Enter customer ID:
1111111111
Enter flight number:
203
Cancellation failed.... Reservation not found
***********MENU***********
1- Book a flight ticket
2- View flights
3- Cancel a booking
4- Search for a booking
5- Exit
3
--------------------------
Enter your choice :
1- Cancel reservation :
2- Return to main menu
1
Enter customer ID:
1111222233
Enter flight number:
203
Reservation has been successfully canceled.
***********MENU***********
1- Book a flight ticket
2- View flights
3- Cancel a booking
4- Search for a booking
5- Exit
5
**** Goodbye! ****
```