MINI PROJECT REPORT

UCS2303 – Object OrientedProgramming

Submitted By

Preethi Prative -3122 22 5001 098
Ram Prasath P -3122 22 5001 103
Samyuktha D – 3122 22 5001 309

Department of Computer Science and Engineering

Sri Sivasubramaniya Nadar College of Engineering
(An Autonomous Institution, Affiliated to Anna University) Kalavakkam – 603110

PROJECT TITLE : AIRLINE RESERVATION

PROJECT SUMMARY:

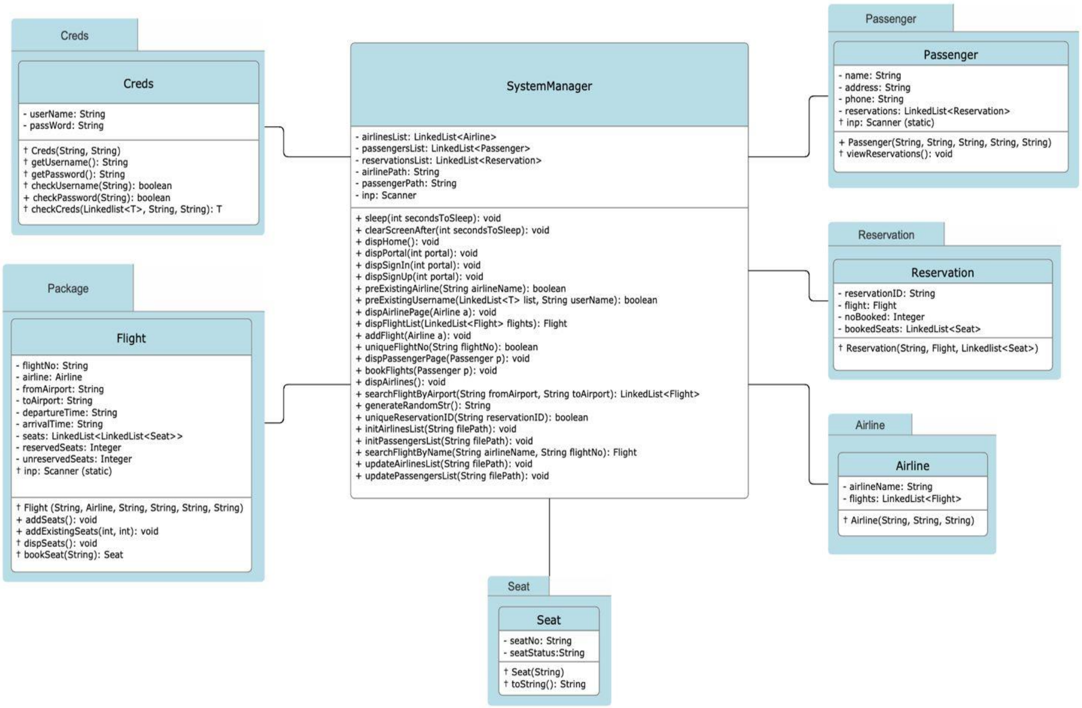
OBJECTIVES:

- User Authentication
- Passenger Booking
- Reservation Tracking:

DEFINITION OF THE PROBLEM:

The Airline Reservation System is a console-based Java application managing airline operations and passenger bookings. The system facilitates user authentication, enables airlines to add flights, allows passengers to book flights and view reservations, and ensures data persistence through file handling. It provides a simple and efficient interface for users to interact with the airline services. The project aims to streamline the process of flight booking and management for both airlines and passengers.

CLASS



Creds
Creds
- userName: String
- passWord: String
† Creds(String, String)
† getUsername(): String
† getPassword(): String
† checkUsername(String): boolean
+ checkPassword(String): boolean
† checkCreds(Linkedlist<T>, String, String): T
Package
Flight
- flightNo: String
- airline: Airline
- fromAirport: String
- toAirport: String
- departureTime: String
- arrivalTime: String
- seats: LinkedList<LinkedList<Seat>>
- reservedSeats: Integer
- unreservedSeats: Integer
† inp: Scanner (static)
† Flight (String, Airline, String, String, String, String)
+ addSeats(): void
+ addExistingSeats(int, int): void
† dispSeats(): void
† bookSeat(String): Seat
SystemManager
- airlinesList: LinkedList<Airline>
- passengersList: LinkedList<Passenger>
- reservationsList: LinkedList<Reservation>
- airlinePath: String
- passengerPath: String
- inp: Scanner
+ sleep(int secondsToSleep): void
+ clearScreenAfter(int secondsToSleep): void
+ dispHome(): void
+ dispPortal(int portal): void
+ dispSignIn(int portal): void
+ dispSignUp(int portal): void
+ preExistingAirline(String airlineName): boolean
+ preExistingUsername(LinkedList<T> list, String userName): boolean
+ dispAirlinePage(Airline a): void
+ dispFlightList(LinkedList<Flight> flights): Flight
+ addFlight(Airline a): void
+ uniqueFlightNo(String flightNo): boolean
+ dispPassengerPage(Passenger p): void
+ bookFlights(Passenger p): void
+ dispAirlines(): void
+ searchFlightByAirport(String fromAirport, String toAirport): LinkedList<Flight>
+ generateRandomStr(): String
+ uniqueReservationID(String reservationID): boolean
+ initAirlinesList(String filePath): void
+ initPassengersList(String filePath): void
+ searchFlightByName(String airlineName, String flightNo): Flight
+ updateAirlinesList(String filePath): void
+ updatePassengersList(String filePath): void
Passenger
Passenger
- name: String
- address: String
- phone: String
- reservations: LinkedList<Reservation>
† inp: Scanner (static)
+ Passenger(String, String, String, String, String)
† viewReservations(): void
Reservation
Reservation
- reservationID: String
- flight: Flight
- noBooked: Integer
- bookedSeats: LinkedList<Seat>
† Reservation(String, Flight, Linkedlist<Seat>)
Airline
Airline
- airlineName: String
- flights: LinkedList<Flight>
† Airline(String, String, String)
Seat
Seat
- seatNo: String
- seatStatus:String
† Seat(String)
† toString(): String

CLASS USED:

- Airline
- Creds
- Flight
- Passenger
- Reservation
- Seat
- SystemManager

MODULES USED:

1. SystemManager:
    - The central module managing the overall system.
    - Responsible for initializing, updating, and interacting with airlines, passengers, and reservations.
2. Creds:
    - Provides credential-checking functionality for both airlines and passengers during sign-in.
3. Airline:

- Represents an airline entity, including details such as name, username, password, and a list of associated flights.

4. Passenger:
   - Represents a passenger entity, storing information like name, address, phone, username, password, and a list of reservations.
5. Flight:
   - Represents a flight entity, containing details like flight number, airline, departure, and arrival information, as well as a list of seats.
6. Seat:
   - Represents a seat entity within a flight, including a seat number and reservation status.
7. Reservation:
   - Represents a reservation entity, linking a passenger to a specific flight and a set of booked seats.
8. File Handling:
   - Includes methods to initialize and update data by reading from and writing to external files.
9. User Interface:
   - Manages the console-based user interface, handling user input, displaying menus, and providing feedback.

CODE IMPLEMENTATION:

```
package AirlineReservationPackage;


import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintStream;

import java.util.Iterator;

import java.util.LinkedList;

import java.util.Scanner;

import java.util.concurrent.TimeUnit;

import org.apache.commons.lang3.RandomStringUtils;


public class SystemManager {

  LinkedList<Airline> airlinesList = new LinkedList();
```

```
LinkedList<Passenger> passengersList = new LinkedList();
LinkedList<Reservation> reservationsList = new LinkedList();
static String airlinePath;
static String passengerPath;
static Scanner inp;

static {
  inp = new Scanner(System.in);
}

public SystemManager(String airlinePath, String passengerPath) {
  SystemManager.airlinePath = airlinePath;
  SystemManager.passengerPath = passengerPath;
}

static void sleep(int secondsToSleep) {
  try {
    TimeUnit.SECONDS.sleep((long)secondsToSleep);
  } catch (InterruptedException var2) {
    Thread.currentThread().interrupt();
  }

}

static void clearScreenAfter(int secondsToSleep) {
  sleep(secondsToSleep);
  System.out.print("\u001b[H\u001b[2J");
  System.out.flush();
}

public void dispHome() {
```

```
clearScreenAfter(0);
System.out.println("\n\t\t\t\t\t\tAirline Reservation System\t\t\t\t\t\t\n");
System.out.println("\nPortals:\n\t\t1. Airline\n\t\t2. Passenger\n\t\t3. Exit\n");
System.out.print("\nEnter Portal: ");
String portal = inp.nextLine();

try {
  if (!portal.equals("1") && !portal.equalsIgnoreCase("airline")) {
    if (!portal.equals("2") && !portal.equalsIgnoreCase("passenger")) {
      if (!portal.equals("3") && !portal.equalsIgnoreCase("exit")) {
        throw new Exception("\nInvalid Portal Input\nEnter '1' or 'Airline' (case insensitive) for Airline Portal\nEnter '2' or 'Passenger' (case insensitive) for Passenger Portal\nEnter '3' or 'Exit' (case insensitive) for Exiting application\n");
      } else {
        this.updateAirlinesList(airlinePath);
        this.updatePassengersList(passengerPath);
        System.out.println("\nExiting Application\n");
        clearScreenAfter(2);
        System.exit(0);
      }
    } else {
      clearScreenAfter(0);
      this.dispPortal(2);
    }
  } else {
    clearScreenAfter(0);
    this.dispPortal(1);
  }
} catch (Exception var3) {
  System.out.println(var3.getMessage());
  clearScreenAfter(2);
```

```
      this.dispHome();
    }
  }

  void dispPortal(int portal) {
    String portalName = null;
    if (portal == 1) {
      portalName = "Airline";
    } else if (portal == 2) {
      portalName = "Passenger";
    }

    System.out.println("\n\t\t\t\t\t\t\t" + portalName + " Portal\t\t\t\t\t\t\t\n");
    System.out.println("\nOptions:\n\t\t1. Sign In\n\t\t2. Sign Up\n\t\t3. Back\n");
    System.out.print("\nEnter Option: ");
    String option = inp.nextLine();

    try {
      if (!option.equals("1") && !option.equalsIgnoreCase("sign in")) {
        if (!option.equals("2") && !option.equalsIgnoreCase("sign up")) {
          if (!option.equals("3") && !option.equalsIgnoreCase("back")) {
            throw new Exception("\nInvalid Option Input\nEnter '1' or 'Sign In' (case insensitive) for Sign In Page\nEnter '2' or 'Sign Up' (case insensitive) for Sign Up Page\nEnter '3' or 'Back' (case insensitive) for Back Page");
          } else {
            clearScreenAfter(0);
            this.dispHome();
          }
        } else {
          clearScreenAfter(0);
          this.dispSignUp(portal);
        }
      } else {
        clearScreenAfter(0);
        this.dispSignIn(portal);
      }
    } catch (Exception var5) {
      System.out.println(var5.getMessage());
      clearScreenAfter(2);
      this.dispPortal(portal);
    }
  }

  void dispSignIn(int portal) {
    String portalName = null;
    if (portal == 1) {
      portalName = "Airline";
    } else if (portal == 2) {
      portalName = "Passenger";
    }

    System.out.println("\n\t\t\t\t\t\t\t" + portalName + " Sign In\t\t\t\t\t\t\t\n");
    System.out.print("\nEnter Username: ");
    String userName = inp.nextLine();
    System.out.print("\nEnter Password: ");
    String passWord = inp.nextLine();

    try {
      Airline airlineTemp = null;
      Passenger passengerTemp = null;
      if (portal == 1) {
        airlineTemp = (Airline)Creds.checkCreds(this.airlinesList, userName, passWord);
      } else if (portal == 2) {
        passengerTemp = (Passenger)Creds.checkCreds(this.passengersList, userName, passWord);
      }

      if (portal == 1 && airlineTemp == null || portal == 2 && passengerTemp == null) {
        throw new Exception("\nIncorrect Username or Password\n");
      } else {
        System.out.println("\nSigned In Successfully\n");
        if (portal == 1) {
          clearScreenAfter(2);
          this.dispAirlinePage(airlineTemp);
        } else if (portal == 2) {
          clearScreenAfter(2);
          this.dispPassengerPage(passengerTemp);
        }
      }
    } catch (Exception var7) {
      System.out.println(var7.getMessage());
      clearScreenAfter(2);
      this.dispPortal(portal);
    }
  }

  void dispSignUp(int portal) {
    String portalName = null;
    if (portal == 1) {
      portalName = "Airline";
    } else if (portal == 2) {
      portalName = "Passenger";
    }
```

```
System.out.println("\n\t\t\t\t\t\t" + portalName + " Sign Up\t\t\t\t\t\t\n");
System.out.print("\nEnter " + portalName + " Name: ");
String name = inp.nextLine();
if (name.equalsIgnoreCase("exit")) {
  clearScreenAfter(0);
  this.dispPortal(portal);
} else {
  System.out.print("\nEnter Username: ");
  String userName = inp.nextLine();
  System.out.print("\nEnter Password: ");
  String passWord = inp.nextLine();

  try {
    if (portal == 1 && this.preExistingAirline(name)) {
      throw new Exception("\nAirline already exists\n");
    } else if (portal == 1 && this.preExistingUsername(this.airlinesList, userName)) {
      throw new Exception("\nUsername already exists\n");
    } else if (portal == 2 && this.preExistingUsername(this.passengersList, userName)) {
      throw new Exception("\nUsername already exists\n");
    } else if (portal == 1) {
      Airline airlineTemp = new Airline(name, userName, passWord);
      this.airlinesList.add(airlineTemp);
      System.out.println("\nAirline Created Successfully\n");
      clearScreenAfter(2);
      this.dispAirlinePage(airlineTemp);
    } else if (portal == 2) {
      System.out.print("\nEnter Address: ");
      String address = inp.nextLine();
      System.out.print("\nEnter Phone: ");
      String phone = inp.nextLine();
```

```
          Passenger passengerTemp = new Passenger(name, address, phone, userName,
passWord);
          this.passengersList.add(passengerTemp);
          System.out.println("\nPassenger created Successfully\n");
          clearScreenAfter(2);
          this.dispPassengerPage(passengerTemp);
        }
      } catch (Exception var9) {
        System.out.println(var9.getMessage());
        clearScreenAfter(2);
        this.dispPortal(portal);
      }
    }
  }

  boolean preExistingAirline(String airlineName) {
    Iterator var3 = this.airlinesList.iterator();

    while(var3.hasNext()) {
      Airline a = (Airline)var3.next();
      if (a.airlineName.equals(airlineName)) {
        return true;
      }
    }

    return false;
  }

  <T extends Creds> boolean preExistingUsername(LinkedList<T> list, String userName) {
    Iterator var4 = list.iterator();
    while(var4.hasNext()) {
      T t = (Creds)var4.next();
      if (t.checkUsername(userName)) {
        return true;
      }
    }

    return false;
  }

  void dispAirlinePage(Airline a) {
    System.out.println("\n\t\t\t\t\t\t\tAirline: " + a.airlineName + "\t\t\t\t\t\t\t\n");
    System.out.println("\nOptions:\n\t\t1. Display Flights\n\t\t2. Add Flights\n\t\t3. Log
Out\n");
    System.out.print("\nEnter Option: ");
    String option = inp.nextLine();

    try {
      if (!option.equals("1") && !option.equalsIgnoreCase("display flights")) {
        if (!option.equals("2") && !option.equalsIgnoreCase("add flights")) {
          if (!option.equals("3") && !option.equalsIgnoreCase("log out")) {
            throw new Exception("\nInvalid Option Input\nEnter '1' or 'Display Flights' (case
insensitive) for List of Flights\nEnter '2' or 'Add Flight' (case insensitive) for Adding a new
Flight\nEnter '3' or 'Log Out' (case insensitive) for Logging Out\n");
          } else {
            System.out.println("\nLogging Out\n");
            clearScreenAfter(2);
            this.dispPortal(1);
          }
        } else {
          clearScreenAfter(0);
          this.addFlight(a);
          clearScreenAfter(2);
          this.dispAirlinePage(a);
        }
      } else {
        clearScreenAfter(0);
        this.dispFlightList(a.flights);
        clearScreenAfter(0);
        this.dispAirlinePage(a);
      }
    } catch (Exception var4) {
      System.out.println(var4.getMessage());
      clearScreenAfter(2);
      this.dispAirlinePage(a);
    }
  }

  Flight dispFlightList(LinkedList<Flight> flights) {
    System.out.printf("------------------------------------------------------------------------------------
--------------------------------%n");
    System.out.printf("List of Flights of Available%n");
    System.out.printf("------------------------------------------------------------------------------------
--------------------------------%n");
    System.out.printf("| %-2s | %-13s | %-12s | %-10s | %-14s | %-12s | %-16s | %-14s |%n",
"No", "Flight Number", "From Airport", "To Airport", "Departure Time", "Arrival Time",
"Unreserved Seats", "Reserved Seats");
    System.out.printf("------------------------------------------------------------------------------------
--------------------------------%n");
    int i = 0;
    Iterator var4 = flights.iterator();

    while(var4.hasNext()) {
      Flight f = (Flight)var4.next();
```

```
          PrintStream var10000 = System.out;
          Object[] var10002 = new Object[8];
          ++i;
          var10002[0] = i;
          var10002[1] = f.flightNo;
          var10002[2] = f.fromAirport;
          var10002[3] = f.toAirport;
          var10002[4] = f.departureTime;
          var10002[5] = f.arrivalTime;
          var10002[6] = f.unreservedSeats;
          var10002[7] = f.reservedSeats;
          var10000.printf("| %02d | %-13s | %-12s | %-10s | %-14s | %-12s | %-16s | %-14s |%n",
var10002);
        }


        System.out.printf("-------------------------------------------------------------------------------------
---------------------------------%n");
        System.out.print("\nFor Seats, Enter Flight Serial Number: ");
        String flightSerial = inp.nextLine();
        if (flightSerial.equalsIgnoreCase("exit")) {
          return null;
        } else {
          try {
            Integer serialNo = Integer.parseInt(flightSerial) - 1;
            if (serialNo >= 0 && serialNo < flights.size()) {
              Flight flight = (Flight)flights.get(serialNo);
              if (flight.unreservedSeats == 0) {
                throw new Exception("\nChosen flight has no available seats\n");
              } else {
                flight.dispSeats();
                System.out.print("\nTo Continue, enter '1' or 'Continue': ");
                String option = inp.nextLine();
                if (!option.equals("1") && !option.equalsIgnoreCase("continue")) {
                  clearScreenAfter(0);
                  this.dispFlightList(flights);
                  return null;
                } else {
                  return flight;
                }
              }
            } else {
              throw new Exception("\nEnter a valid serial number\n");
            }
          } catch (Exception var7) {
            System.out.println(var7.getMessage());
            clearScreenAfter(2);
            this.dispFlightList(flights);
            return null;
          }
        }
      }

      void addFlight(Airline a) {
        System.out.println("\nEnter New Flight Details of Airline: " + a.airlineName);
        System.out.print("\nFlight Number: ");
        String flightNo = inp.nextLine();
        if (!this.uniqueFlightNo(flightNo)) {
          System.out.println("\nThe entered flight number already exists\n");
          clearScreenAfter(2);
          this.addFlight(a);
        } else if (flightNo.equalsIgnoreCase("exit")) {
          clearScreenAfter(0);
```

```java
    this.dispAirlinePage(a);
  } else {
    System.out.print("\nFrom Airport: ");
    String fromAirport = inp.nextLine();
    System.out.print("\nTo Airport: ");
    String toAirport = inp.nextLine();
    System.out.print("\nDeparture Time: ");
    String departureTime = inp.nextLine();
    System.out.print("\nArrival Time: ");
    String arrivalTime = inp.nextLine();
    Flight f = new Flight(flightNo, a, fromAirport, toAirport, departureTime, arrivalTime);
    f.addSeats();
    a.flights.add(f);
    System.out.println("\nFlight Successfully Added\n");
  }
}

boolean uniqueFlightNo(String flightNo) {
  Iterator var3 = this.airlinesList.iterator();

  while(var3.hasNext()) {
    Airline a = (Airline)var3.next();
    Iterator var5 = a.flights.iterator();

    while(var5.hasNext()) {
      Flight f = (Flight)var5.next();
      if (f.flightNo.equalsIgnoreCase(flightNo)) {
        return false;
      }
    }
  }
```

```
    return true;
  }

  void dispPassengerPage(Passenger p) {
    System.out.println("\n\t\t\t\t\t\t\tPassenger: " + p.name + "\t\t\t\t\t\t\n");
    System.out.println("\nAddress: " + p.address);
    System.out.println("\nPhone: " + p.phone);
    System.out.println("\nOptions:\n\t\t1. Book Flights\n\t\t2. Reservations\n\t\t3. Log Out\n");
    System.out.print("\nEnter Option: ");
    String option = inp.nextLine();

    try {
      if (!option.equals("1") && !option.equalsIgnoreCase("book flights")) {
        if (!option.equals("2") && !option.equalsIgnoreCase("reservations")) {
          if (!option.equals("3") && !option.equalsIgnoreCase("log out")) {
            throw new Exception("\nInvalid Option Input\nEnter '1' or 'Book Flights' (case insensitive) for Booking Flights\nEnter '2' or 'Reservations' (case insensitive) for viewing all Reservations\nEnter '3' or 'Log Out' (case insensitive) for Logging Out\n");
          } else {
            System.out.println("\nLogging Out\n");
            clearScreenAfter(2);
            this.dispPortal(2);
          }
        } else {
          clearScreenAfter(0);
          p.viewReservations();
          clearScreenAfter(0);
          this.dispPassengerPage(p);
        }
      } else {
        clearScreenAfter(0);
        this.bookFlights(p);
        clearScreenAfter(2);
        this.dispPassengerPage(p);
      }
    } catch (Exception var4) {
      System.out.println(var4.getMessage());
      clearScreenAfter(2);
      this.dispPassengerPage(p);
    }
  }

  void bookFlights(Passenger p) {
    System.out.println("\n\t\t\t\t\t\t\tBook Flights\t\t\t\t\t\t\n");
    this.dispAirlines();
    System.out.print("\nEnter From Airport: ");
    String fromAirport = inp.nextLine();
    if (fromAirport.equalsIgnoreCase("exit")) {
      clearScreenAfter(0);
      this.dispPassengerPage(p);
    } else {
      System.out.print("\nEnter To Airport: ");
      String toAirport = inp.nextLine();
      LinkedList<Flight> filteredFlights = this.searchFlightByAirport(fromAirport,
toAirport);
      Flight selectedFlight = this.dispFlightList(filteredFlights);
      if (selectedFlight == null) {
        System.out.println("\nNo flights between given Airports\n");
        clearScreenAfter(2);
        this.bookFlights(p);
      } else {
```

```java
System.out.print("\nEnter number of seats to book: ");
Integer seatsSize = Integer.parseInt(inp.nextLine());
if (seatsSize > 10) {
  System.out.println("\nMaximum 10 Seats per Passenger\n");
  clearScreenAfter(2);
  this.bookFlights(p);
} else if (seatsSize > selectedFlight.unreservedSeats) {
  System.out.println("\nEntered number of seats is greater than available seats\n");
  clearScreenAfter(2);
  this.bookFlights(p);
} else {
  int i = 0;
  LinkedList<Seat> bookedSeats = new LinkedList();

  String genID;
  while(i < seatsSize) {
    System.out.print("\nEnter Seat: ");
    genID = inp.nextLine();
    Seat seat = selectedFlight.bookSeat(genID);
    if (seat != null) {
      bookedSeats.add(seat);
      ++i;
    }
  }

  for(genID = this.generateRandomStr(); !this.uniqueReservationID(genID); genID = this.generateRandomStr()) {
  }

  Reservation reservation = new Reservation(this.generateRandomStr(), selectedFlight, bookedSeats);
```

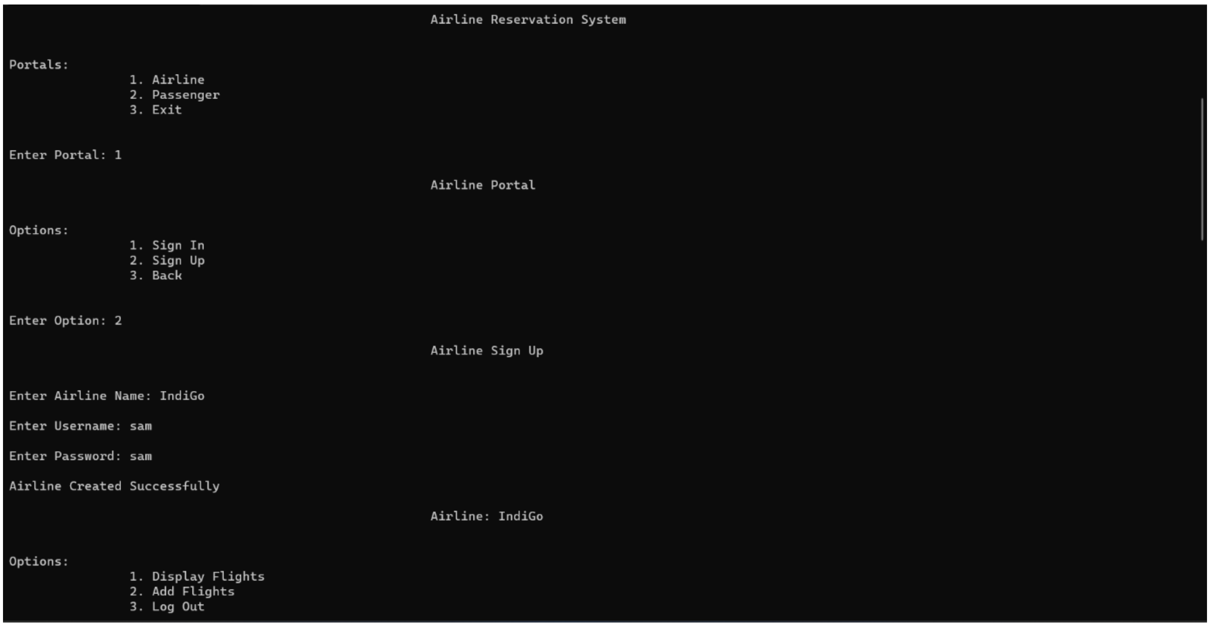
```
          p.reservations.add(reservation);
          System.out.println("\nSuccessfully Booked Seats\n");
        }
      }
    }
  }

  void dispAirlines() {
    Iterator var2 = this.airlinesList.iterator();

    while(var2.hasNext()) {
      Airline a = (Airline)var2.next();
      System.out.printf("------------------------------------------------------------------------------
---------------------------------%n");
      System.out.printf("List of Flights of Available in Airline: %s%n", a.airlineName);
      System.out.printf("------------------------------------------------------------------------------
---------------------------------%n");
      System.out.printf("| %-2s | %-13s | %-12s | %-10s | %-14s | %-12s | %-16s | %-14s |%n",
"No", "Flight Number", "From Airport", "To Airport", "Departure Time", "Arrival Time",
"Unreserved Seats", "Reserved Seats");
      System.out.printf("------------------------------------------------------------------------------
---------------------------------%n");
      int i = 0;
      Iterator var5 = a.flights.iterator();

      while(var5.hasNext()) {
        Flight f = (Flight)var5.next();
        PrintStream var10000 = System.out;
        Object[] var10002 = new Object[8];
        ++i;
        var10002[0] = i;
        var10002[1] = f.flightNo;
        var10002[2] = f.fromAirport;
        var10002[3] = f.toAirport;
        var10002[4] = f.departureTime;
        var10002[5] = f.arrivalTime;
        var10002[6] = f.unreservedSeats;
        var10002[7] = f.reservedSeats;
        var10000.printf("| %02d | %-13s | %-12s | %-10s | %-14s | %-12s | %-16s | %-14s |%n",
var10002);
      }

      System.out.printf("---------------------------------------------------------------------------------
----------------------------------%n");
    }

  }

  LinkedList<Flight> searchFlightByAirport(String fromAiport, String toAirport) {
    LinkedList<Flight> filteredFlights = new LinkedList();
    Iterator var5 = this.airlinesList.iterator();

    while(var5.hasNext()) {
      Airline a = (Airline)var5.next();
      Iterator var7 = a.flights.iterator();

      while(var7.hasNext()) {
        Flight f = (Flight)var7.next();
        if (f.fromAirport.equalsIgnoreCase(fromAiport) &&
f.toAirport.equalsIgnoreCase(toAirport)) {
          filteredFlights.add(f);
        }
      }
    }
    return filteredFlights;
  }

  String generateRandomStr() {
    String generatedString = RandomStringUtils.randomAlphanumeric(15);
    return generatedString;
  }

  boolean uniqueReservationID(String reservationID) {
    Iterator var3 = this.passengersList.iterator();

    while(var3.hasNext()) {
      Passenger p = (Passenger)var3.next();
      Iterator var5 = p.reservations.iterator();

      while(var5.hasNext()) {
        Reservation r = (Reservation)var5.next();
        if (r.reservationID.equalsIgnoreCase(reservationID)) {
          return false;
        }
      }
    }

    return true;
  }

  public void initAirlinesList(String filePath) {
    try {
      File Obj = new File(filePath);
      Scanner airlineReader = new Scanner(Obj);
```

```
      while(true) {
        String line;
        do {
          if (!airlineReader.hasNextLine()) {
            return;
          }

          line = airlineReader.nextLine();
        } while(line.trim().isEmpty());

        String[] airlineFields = line.split(",");
        Airline tempAirline = new Airline(airlineFields[0], airlineFields[1], airlineFields[2]);
        int n = airlineFields.length;

        for(int i = 3; i < n; i += 7) {
          Flight tempFlight = new Flight(airlineFields[i], tempAirline, airlineFields[i + 1], airlineFields[i + 2], airlineFields[i + 3], airlineFields[i + 4]);
          tempFlight.addExistingSeats(Integer.parseInt(airlineFields[i + 5]), Integer.parseInt(airlineFields[i + 6]));
          tempAirline.flights.add(tempFlight);
        }

        this.airlinesList.add(tempAirline);
      }
    } catch (FileNotFoundException var10) {
      var10.printStackTrace();
    }
  }

  public void initPassengesrList(String filePath) {
    try {
      File Obj = new File(filePath);
      Scanner passengerReader = new Scanner(Obj);

      while(true) {
        String line;
        do {
          if (!passengerReader.hasNextLine()) {
            return;
          }

          line = passengerReader.nextLine();
        } while(line.trim().isEmpty());

        String[] passengerFields = line.split(",");
        Passenger tempPassenger = new Passenger(passengerFields[0], passengerFields[1],
passengerFields[2], passengerFields[3], passengerFields[4]);

        for(int i = 5; i < passengerFields.length; i = i + Integer.parseInt(passengerFields[i + 3])
+ 4) {
          Flight tempFlight = this.searchFlightByName(passengerFields[i + 1],
passengerFields[i + 2]);
          LinkedList<Seat> bookedSeats = new LinkedList();

          for(int j = 0; j < Integer.parseInt(passengerFields[i + 3]); ++j) {
            bookedSeats.add(tempFlight.bookSeat(passengerFields[i + 4 + j]));
          }

          Reservation tempReservation = new Reservation(passengerFields[i], tempFlight,
bookedSeats);
          tempPassenger.reservations.add(tempReservation);
        }
        this.passengersList.add(tempPassenger);
      }
    } catch (FileNotFoundException var12) {
      var12.printStackTrace();
    }
  }

  Flight searchFlightByName(String airlineName, String flightNo) {
    Iterator var4 = this.airlinesList.iterator();

    while(true) {
      Airline a;
      do {
        if (!var4.hasNext()) {
          return null;
        }

        a = (Airline)var4.next();
      } while(!a.airlineName.equalsIgnoreCase(airlineName));

      Iterator var6 = a.flights.iterator();

      while(var6.hasNext()) {
        Flight f = (Flight)var6.next();
        if (f.flightNo.equalsIgnoreCase(flightNo)) {
          return f;
        }
      }
    }
  }
```

```
  public void updateAirlinesList(String filePath) {
    try {
      FileWriter Writer = new FileWriter(filePath);

      String airlineField;
      for(Iterator var4 = this.airlinesList.iterator(); var4.hasNext(); airlineField = "") {
        Airline a = (Airline)var4.next();
        airlineField = a.airlineName + "," + a.getUsername() + "," + a.getPassword() + ",";

        Flight f;
        for(Iterator var7 = a.flights.iterator(); var7.hasNext(); airlineField = airlineField + f.flightNo + "," + f.fromAirport + "," + f.toAirport + "," + f.departureTime + "," + f.arrivalTime + "," + f.seats.size() + "," + ((LinkedList)f.seats.getFirst()).size() + ",") {
          f = (Flight)var7.next();
        }

        airlineField = airlineField + "\n";
        Writer.write(airlineField);
      }

      Writer.close();
    } catch (IOException var8) {
      var8.printStackTrace();
    }

  }

  public void updatePassengersList(String filePath) {
    try {
      FileWriter Writer = new FileWriter(filePath);
      String passengerField;
      for(Iterator var4 = this.passengersList.iterator(); var4.hasNext(); passengerField = "") {
        Passenger p = (Passenger)var4.next();
        passengerField = p.name + "," + p.address + "," + p.phone + "," + p.getUsername() +
"," + p.getPassword() + ",";
        Iterator var7 = p.reservations.iterator();

        while(var7.hasNext()) {
          Reservation r = (Reservation)var7.next();
          passengerField = passengerField + r.reservationID + "," + r.flight.airline.airlineName
+ "," + r.flight.flightNo + "," + r.bookedSeats.size() + ",";

          Seat s;
          for(Iterator var9 = r.bookedSeats.iterator(); var9.hasNext(); passengerField =
passengerField + s.seatNo + ',') {
            s = (Seat)var9.next();
          }
        }

        passengerField = passengerField + "\n";
        Writer.write(passengerField);
      }

      Writer.close();
    } catch (IOException var10) {
      var10.printStackTrace();
    }

  }
}
```

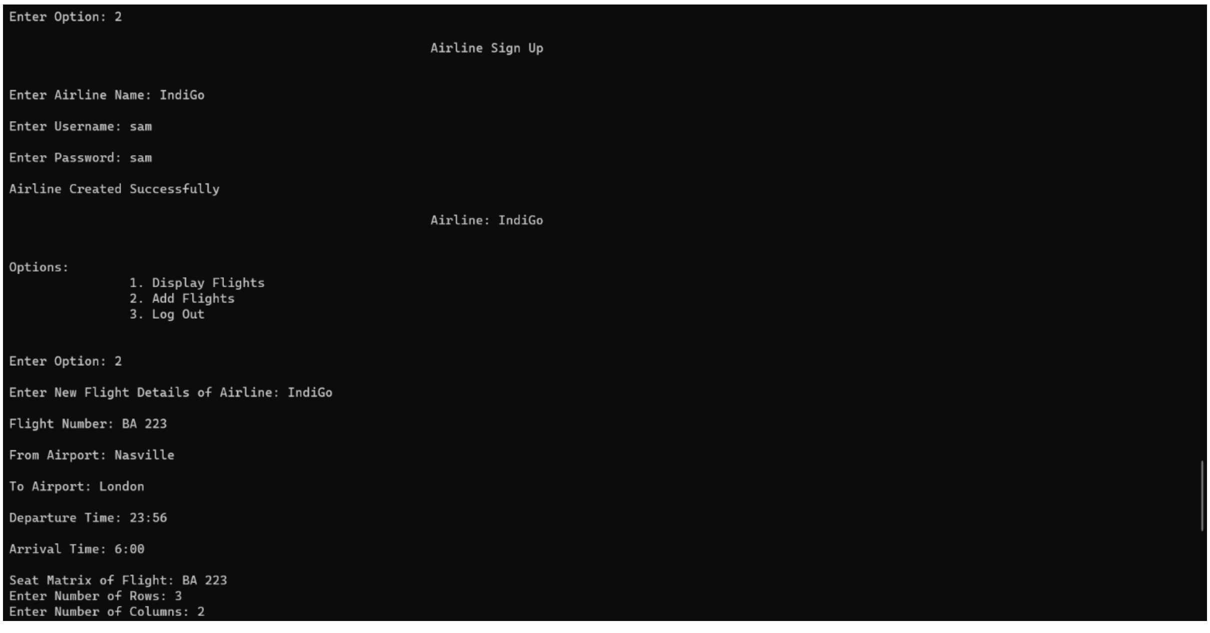
OUTPUT SCREENSHOTS:

SIGN UP

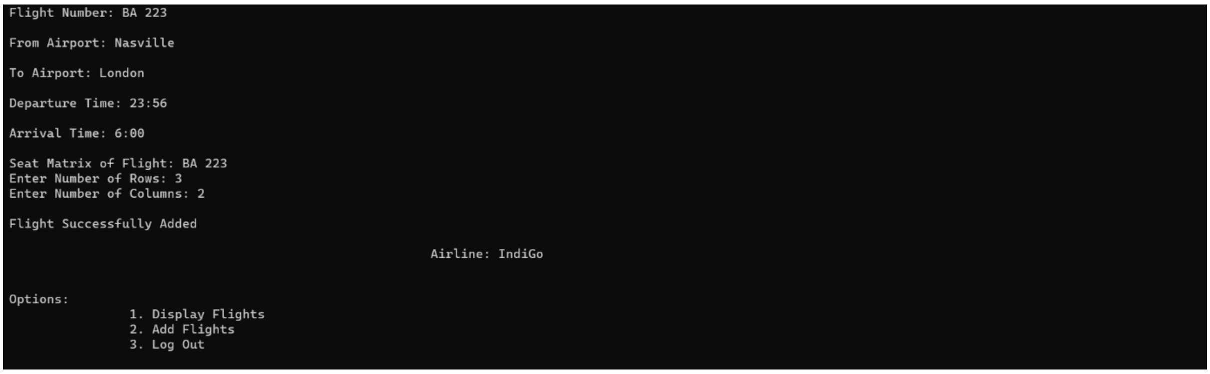


```
                                                        Airline Reservation System


Portals:
                1. Airline
                2. Passenger
                3. Exit


Enter Portal: 1
                                                        Airline Portal


Options:
                1. Sign In
                2. Sign Up
                3. Back


Enter Option: 2
                                                        Airline Sign Up


Enter Airline Name: IndiGo

Enter Username: sam

Enter Password: sam

Airline Created Successfully
                                                        Airline: IndiGo


Options:
                1. Display Flights
                2. Add Flights
                3. Log Out
```

ADD FLIGHT

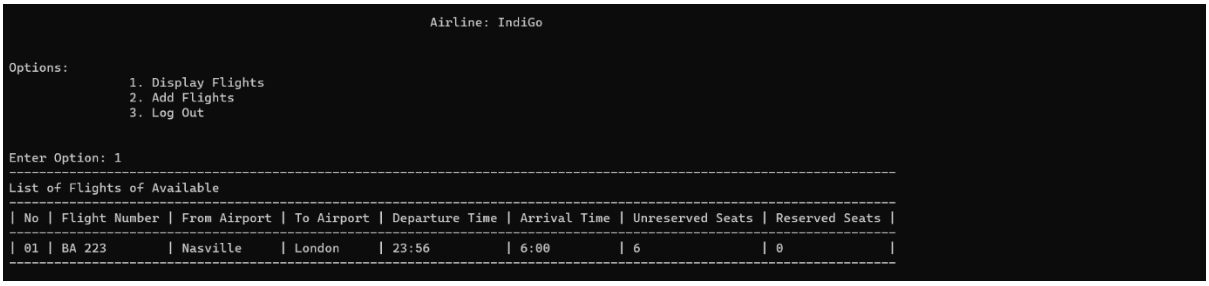


```
Enter Option: 2
                                                        Airline Sign Up


Enter Airline Name: IndiGo

Enter Username: sam

Enter Password: sam

Airline Created Successfully
                                                        Airline: IndiGo


Options:
                1. Display Flights
                2. Add Flights
                3. Log Out


Enter Option: 2

Enter New Flight Details of Airline: IndiGo

Flight Number: BA 223

From Airport: Nasville

To Airport: London

Departure Time: 23:56

Arrival Time: 6:00

Seat Matrix of Flight: BA 223
Enter Number of Rows: 3
Enter Number of Columns: 2
```



```
Flight Number: BA 223

From Airport: Nasville

To Airport: London

Departure Time: 23:56

Arrival Time: 6:00

Seat Matrix of Flight: BA 223
Enter Number of Rows: 3
Enter Number of Columns: 2

Flight Successfully Added

                                                        Airline: IndiGo


Options:
                1. Display Flights
                2. Add Flights
                3. Log Out
```

DISPLAY:



```
                                                        Airline: IndiGo


Options:
                1. Display Flights
                2. Add Flights
                3. Log Out


Enter Option: 1
----------------------------------------------------------------------------------------------------------------------
List of Flights of Available
----------------------------------------------------------------------------------------------------------------------
| No | Flight Number | From Airport | To Airport | Departure Time | Arrival Time | Unreserved Seats | Reserved Seats |
----------------------------------------------------------------------------------------------------------------------
| 01 | BA 223        | Nasville     | London     | 23:56          | 6:00         | 6                | 0              |
----------------------------------------------------------------------------------------------------------------------
```

OOPs CONCEPTS USED:

- Inheritance
- Data abstraction
- Encapsulation
- Dynamic binding
- Packages
- Exception Handling
- File Handling
- Collections

ELABORATION:

- Inheritance: Enables classes Passenger and Airline to inherit common functionalities from the Creds class.

- Data Abstraction: Classes abstract essential features of entities, like Airline, Passenger, and Flight, hiding unnecessary details.

- Encapsulation: Attributes (e.g., airlinesList) and methods are encapsulated within classes, ensuring controlled access.

- Dynamic Binding: Method overriding allows the selection of specific implementations at runtime, demonstrating dynamic binding.

Java Features:

- Packages: Though not explicit, organizing related classes into packages enhances code structure and modularity.

- Exception Handling: The code uses exception handling to gracefully manage runtime errors and prevent abrupt crashes.

- File Handling: File handling is employed for initializing and updating airline and passenger data, ensuring persistence.

- Collections: LinkedList is utilized for efficient storage of entities like airlines, passengers, reservations, and flights.

LIMITATIONS:

Limited Scalability:

The code may face challenges in handling a large-scale system with a substantial number of airlines, passengers, and reservations. Performance issues could arise as the data grows.

Limited Security Measures:

The code might lack robust security features, such as encryption for sensitive data (e.g., passwords). This could pose a security risk in a real-world application.

Synchronous Execution:

The code seems to follow a synchronous execution model, which might lead to delays in processing, especially when dealing with time-consuming operations or external dependencies.

Minimal Error Handling:

While the code includes basic exception handling, it may benefit from more comprehensive error handling mechanisms to handle a wider range of potential issues.

File Handling Assumptions:

The code assumes that file operations (reading/writing) will always succeed, which might not be the case in a real-world scenario. Robust error handling for file operations is essential.

Limited User Interface (UI):

The code lacks a graphical user interface (GUI), which may limit its user-friendliness. A graphical interface could enhance the user experience.

Single-Threaded Execution:

The code appears to operate in a single-threaded manner. In a more complex system, a multi-threaded approach might be necessary for better performance and responsiveness.

Limited Documentation:

The code may lack comprehensive documentation, making it challenging for developers to understand and maintain the system in the long run.

Assumption of Unique Flight Numbers:

The code assumes that flight numbers are unique across all airlines. In a real-world scenario, flight numbers might need to be unique within each airline.

Assumption of Valid User Input:

The code assumes that users will provide valid input. Adding more input validation and handling edge cases could enhance the system's robustness.

FUTURE EXTENSION:

1. Advanced Authentication: Implement multi-factor authentication for enhanced security.
2. Graphical User Interface (GUI): Develop a visually appealing and intuitive GUI for better user experience.
3. Dynamic Flight Schedule: Allow real-time management of flight schedules, including additions and modifications.
4. Payment Integration: Integrate secure payment gateways for online reservations.
5. Notification System: Implement alerts for reservation confirmations and flight updates.
6. Seat Selection and Customization: Enable passengers to choose specific seats and customize preferences.

7. Dynamic Pricing and Discounts: Introduce dynamic pricing based on demand and seat availability.
8. Mobile Application: Develop iOS and Android apps for on-the-go access to flight information.
9. Integration with External Systems: Connect with external services like weather updates and traffic information.
10. Analytics and Reporting: Implement analytics tools for user behavior insights and data-driven decisions.
11. Internationalization and Localization: Support multiple languages and currencies for a diverse user base.
12. Advanced Reservation Options: Allow round-trip, open-jaw, and multi-city bookings.

13. Feedback and Rating System: Incorporate a passenger feedback and rating system.
14. Integration with Loyalty Programs: Collaborate with airline loyalty programs for rewards and redemptions.
15. Blockchain Security: Explore blockchain technology for secure transactions and data storage.