Final Project

Object Oriented Programming

Project name: **E-PORT**

Name: **Sri Kalyan Rohan**

Student ID: **2440090266**

Class: **L2BC**

Project Specification

The purpose for this project is to simulate an airport from activities such as buying a ticket to the cancellation of tickets. The thought of this came from the fact that people had to wait in lines in order to check in or when buying a ticket. This can be a nuisance and considering that we are in a pandemic having a crowd or long lines is never a good idea. Creating an application that can handle all the activities that are being done in an airport would save people a lot of time as it would be more efficient. Users will have to sign up first before being greeted by the main menu which allows the user to navigate through the different services that the airport offers. For this project I am just going to take the most common activities that are being done in an airport which is the checking in, immigration, buying tickets and cancelling of tickets. Users can also deposit money into their account which will then be used for purchases. I will also be using CLI for a straighter forward demonstration and users will use numbers to navigate around the application.

Input:

Name

Age

Nationality

Gender

Integer (for choices and navigation)

Float (for deposit)

Flight Number when purchasing ticket

Output:

Ticket details

Balance

Tickets bought for other people

Flight details

Passenger details

Solution Design

MAIN MENU

The main menu is used to navigate through the different services that the application provides. It includes:

1. Book a flight
2. Check in/Immigration
3. Deposit
4. Account Information
5. Cancel Booking
6. Exit/Logout

BOOK A FLIGHT

When the user picks this option it will ask the user whether or not the user wants to buy a ticket for him/herself or for someone else. If the user picks to buy a ticket for someone else, then it will direct the user into a series of inputs of the person's details. The inputs will be the same with the ones that were used during the sign up. These details are then used to create another passenger object. Then the user will be brought into a sub menu consisting of (if users choose to buy tickets for themselves then they skip to this part immediately):

1. View Flights
2. Custom Search
3. Back

The view flights sub menu will show the users all the available flights and it will have the user pick whether or not they want to continue with their transaction. The users will then pick the flight that they want by entering the number corresponding to the airline shown. The transaction will be made and then the ticket details will be printed out along with the remaining balance of the account.

The custom search sub menu will require users to enter their own flight name, destination and maximum price they are willing to pay. This will filter out the airlines available according to the users input. The users will then pick the flight that they want by entering the number corresponding to the airline shown. The transaction will be made and then the ticket details will be printed out along with the remaining balance of the account.

The back sub menu will just bring the user back to the main menu.

CHECK IN/IMMIGRATION

This will first check whether the user has already bought a ticket or not. If they have already purchased a ticket, it will show the ticket details and ask the user to confirm. Once the user confirms, the checked in and immigration attribute of the passenger will be set to TRUE.

DEPOSIT

The user will enter the amount that they want to deposit and it will be added to their account.

ACCOUNT INFORMATION

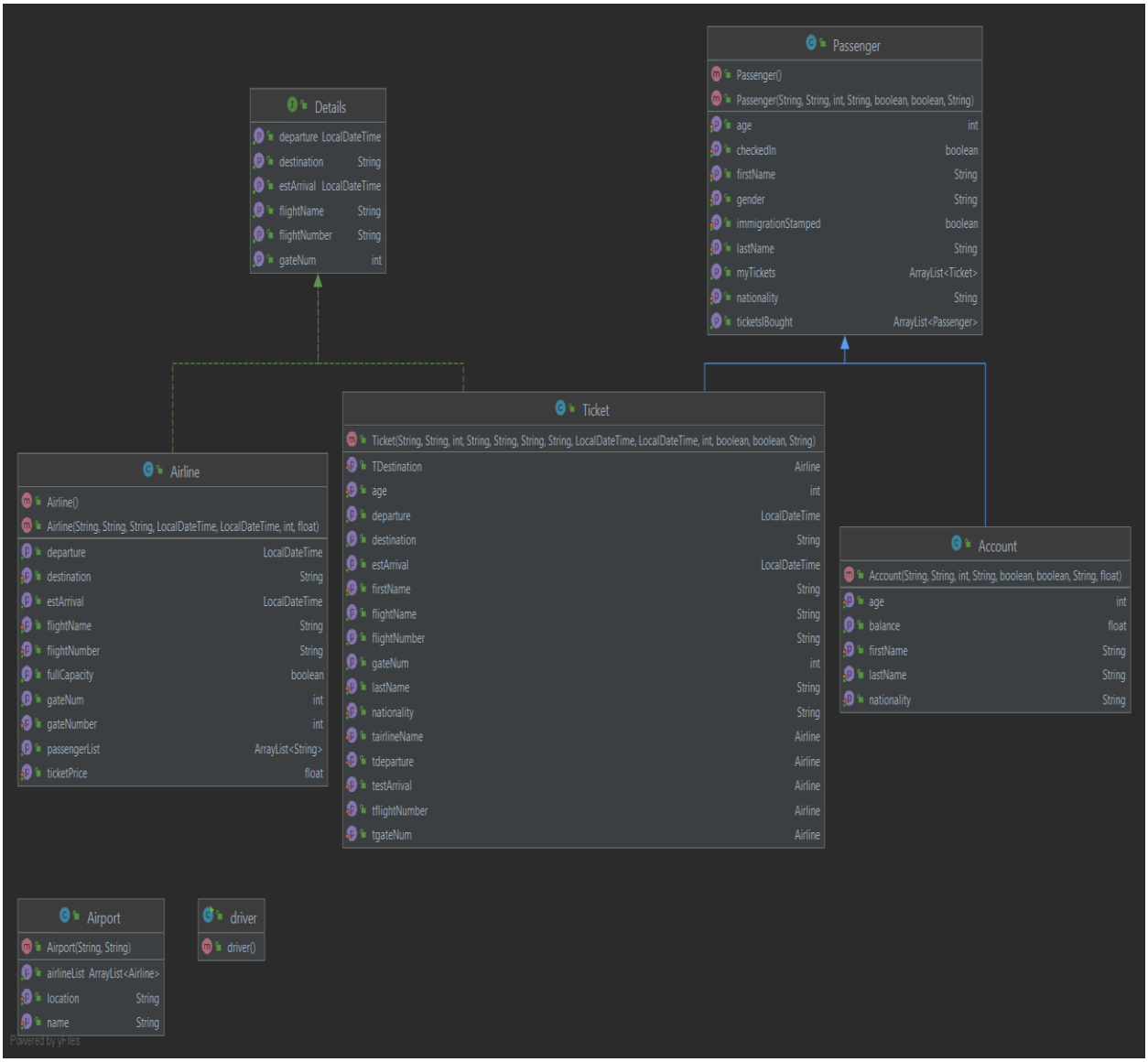
This will display the user's personal information along with the ticket details if the user has bought a ticket. The tickets that the user bought for other people will also be shown and the balance of the account as well.

CANCEL BOOKING

This will first check whether the user has bought any tickets. The user will then be shown a sub menu where they will have to choose whether they want to cancel their own ticket or someone else's. If they want to cancel their own ticket, it will check whether they have checked in or not. If they have checked in, then they cannot cancel booking. If they have not, then the user will have to enter the flight number of the ticket that they want to cancel. The program will then print out the new balance of the account indicating the success of the cancellation.

If they pick to cancel a ticket that they bought for someone else, then they will have to enter the first and last name of the person that they bought the ticket for and the flight number of the ticket. The program will then print out the new balance of the account indicating the success of the cancellation.

UML DIAGRAM


Passenger
Passenger()
Passenger(String, String, int, String, boolean, boolean, String)
age int
checkedIn boolean
firstName String
gender String
immigrationStamped boolean
lastName String
myTickets ArrayList<Ticket>
nationality String
ticketsIBought ArrayList<Passenger>
Details
departure LocalDateTime
destination String
estArrival LocalDateTime
flightName String
flightNumber String
gateNum int
Ticket
Ticket(String, String, int, String, String, String, String, LocalDateTime, LocalDateTime, int, boolean, boolean, String)
TDestination Airline
age int
departure LocalDateTime
destination String
estArrival LocalDateTime
firstName String
flightName String
flightNumber String
gateNum int
lastName String
nationality String
tairlineName Airline
tdeparture Airline
testArrival Airline
tflightNumber Airline
tgateNum Airline
Airline
Airline()
Airline(String, String, String, LocalDateTime, LocalDateTime, int, float)
departure LocalDateTime
destination String
estArrival LocalDateTime
flightName String
flightNumber String
fullCapacity boolean
gateNum int
gateNumber int
passengerList ArrayList<String>
ticketPrice float
Account
Account(String, String, int, String, boolean, boolean, String, float)
age int
balance float
firstName String
lastName String
nationality String
Airport
Airport(String, String)
airlineList ArrayList<Airline>
location String
name String
driver
driver()
Powered by yFiles

CODE EXPLANATION

For this program I used 6 classes and 1 interface and I will be explaining them one by one. The list of classes are as follows:

1. Passenger
2. Account
3. Ticket
4. Airline
5. Airport
6. Driver

Interface:

1. Details

PASSENGER CLASS

```
import java.util.ArrayList;

public class Passenger  {
    private String firstName;
    private String lastName;
    private String gender;
    private int age;
    private String nationality;
    private boolean checkedIn;
    private boolean immigrationStamped;
    //array list to store own tickets
    private ArrayList<Ticket>myTickets=new ArrayList<>();
    //array list to store tickets that are bought for someone else
    private ArrayList<Passenger>TicketsIBought=new ArrayList<>();

    //constructor
    public Passenger(String firstName,String lastName,int age, String nationality,boolean checkedIn,boolean immigrationStamped,String gender){
        this.firstName=firstName;
        this.lastName=lastName;
        this.gender=gender;
        this.age=age;
        this.nationality=nationality;
        this.checkedIn=checkedIn;
        this.immigrationStamped=immigrationStamped;
    }
    //default constructor
    public Passenger(){
    }
```

The passenger class is used to create passenger objects that holds the details of the users such as their first name, last name, age, gender and nationality. This class also indicates whether a passenger has checked in or not through a private Boolean variable. This goes the same for the immigration status of the passenger.

A passenger object will also have an array list that will store their own tickets and another array to store the passengers whose tickets were bought for. There are two constructors for the class, the main constructor and the default constructor. The reason for this is that later on in the driver class when we want to create a passenger object we will have to create it using a default constructor first and then call the setters to set the attributes of the new passenger.

```
//looping through the myTickets array and printing out every detail the ticket has
public void getTicketDetails(){
    for (Ticket myTicket : myTickets) {
        System.out.println("First Name: " + getFirstName());
        System.out.println("Last Name: " + getLastName());
        System.out.println("Age: " + getAge());
        System.out.println("Gender: " + getGender());
        System.out.println("Nationality: " + getNationality());
        System.out.println();
        System.out.println("Flight Name: " + myTicket.getFlightName());
        System.out.println("Flight Number: " + myTicket.getFlightNumber());
        System.out.println("Destination: " + myTicket.getDestination());
        System.out.println("Departure date and time: " + myTicket.getDeparture());
        System.out.println("Arrival date and time: " + myTicket.getEstArrival());
        System.out.println("Gate Number: " + myTicket.getGateNum());
        System.out.println();
    }
}

//get details of passenger only
public void getDetails(){
    System.out.println("First Name: "+getFirstName());
    System.out.println("Last Name: "+getLastName());
    System.out.println("Age: "+getAge());
    System.out.println("Gender: "+getGender());
    System.out.println("Nationality: "+getNationality());
    System.out.println();
}
```

The get ticket details function will loop throught the my ticket array list and get the ticket object. Using the ticket object, the getters will be called so that the information of the ticket will be printed out. The get details function will just call the getters of the passenger class to get the details of the passenger.

ACCOUNT CLASS

```java
public class Account extends Passenger{
    private float balance;

    //Class constructor
    public Account(String firstName, String lastName, int age, String nationality, boolean checkedIn, boolean immigrationStamped,
                   String gender,float balance ) {
        super(firstName, lastName, age, nationality, checkedIn, immigrationStamped, gender);
        this.balance=balance;
    }

    //get balance of account
    public float getBalance() { return balance; }

    //for putting money into the account
    public void deposit(float amt) {
        balance+=amt;
        System.out.println("Your current balance is: "+balance);
    }
    //withdraws if amount is equal or less than balance
    public void withDraw(float amt){
        if(amt<=balance){
            balance-=amt;

        }
    }
}
```

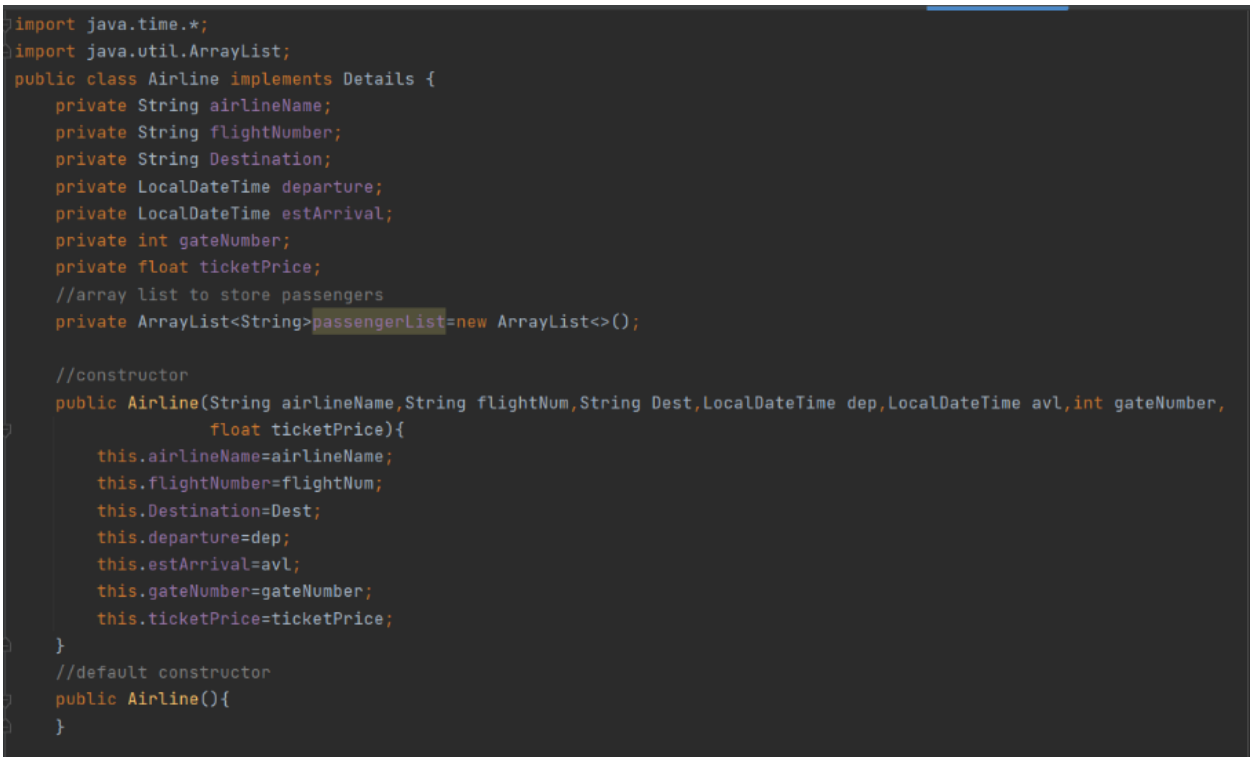
The account class extends the passenger class. It has the get balance function to return the balance of the account, deposit function to store money into the account by taking in a float parameter and a withdraw function which takes money out of the account if the float parameter is equal or lesser than the balance.

INTERFACE – DETAILS

```java
import java.time.LocalDateTime;

//interface for classes such as ticket and airline which inherits the same set of functions
public interface Details {
    String getFlightName();

    LocalDateTime getDeparture();

    String getDestination();

    String getFlightNumber();

    LocalDateTime getEstArrival();

    int getGateNum();


}
```

The interface contains getter functions of details that both the ticket and airline class shares. For example, an airline detail such as the destination of a flight will also be included in a ticket.

AIRLINE CLASS



```
import java.time.*;
import java.util.ArrayList;
public class Airline implements Details {
    private String airlineName;
    private String flightNumber;
    private String Destination;
    private LocalDateTime departure;
    private LocalDateTime estArrival;
    private int gateNumber;
    private float ticketPrice;
    //array list to store passengers
    private ArrayList<String>passengerList=new ArrayList<>();

    //constructor
    public Airline(String airlineName,String flightNum,String Dest,LocalDateTime dep,LocalDateTime avl,int gateNumber,
                   float ticketPrice){
        this.airlineName=airlineName;
        this.flightNumber=flightNum;
        this.Destination=Dest;
        this.departure=dep;
        this.estArrival=avl;
        this.gateNumber=gateNumber;
        this.ticketPrice=ticketPrice;
    }
    //default constructor
    public Airline(){
    }
```

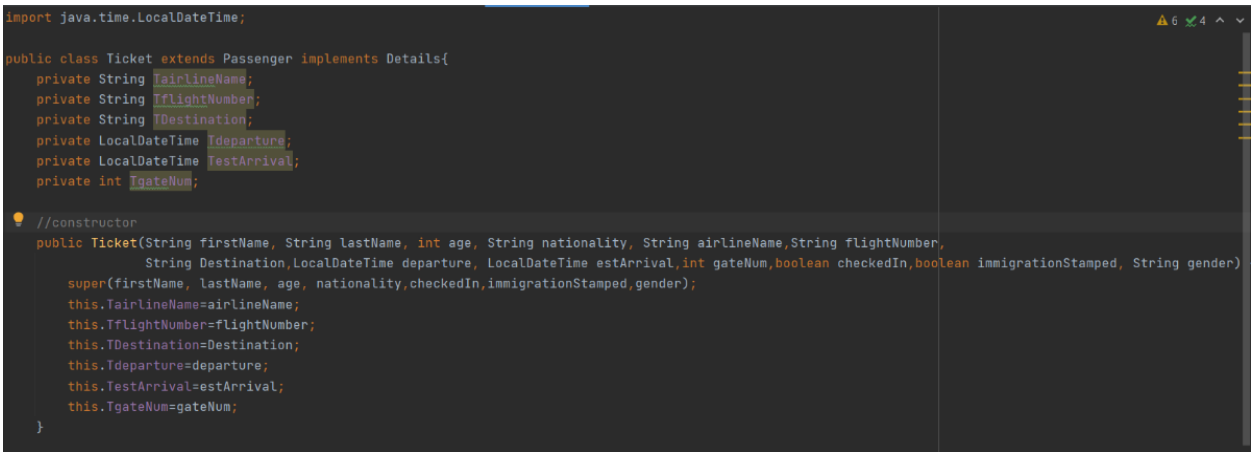
The airline class contains attributes that you would see in an airline schedule board in the airport such as flight name, flight number, destination, date and time and etc. The airline class implements the detail interface above as these details will also be in the ticket class. The airline also has an array list that will store the names of the passengers that have bought tickets of that airline. It also uses the local date time library from the time library of java to store the date and time of the arrival and the departure. Just like the passenger class it has a main constructor and a default constructor. Later on in the driver class, it will be explained as to why the default constructor exist.

```java
    //to check if airline already reach maximum capacity
    //if max capacity reached then return True
    public boolean isFullCapacity(){
        boolean fullCapacity = Boolean.FALSE;
        int maximumCapacity = 100;
        if(passengerList.size()>= maximumCapacity){
            fullCapacity =Boolean.TRUE;
        }
        return fullCapacity;
    }

    //to add passengers to the passenger array list
    //checks if airline have reached max capacity, if not then add passenger to array list
    public void addPassengers(Passenger passenger){
        if(isFullCapacity()==Boolean.FALSE){
            passengerList.add(passenger.getFirstName()+" "+passenger.getLastName());
        }
        else{
            System.out.println("Airline have reached maximum capacity. ");
        }
    }

}
```

The airline class has a function that checks whether the airline is already fully booked. The function works by having a Boolean value of false which indicates that it is not fully booked. The default maximum capacity that I set was 100 so it will check the passenger array list whether or not the size of the array is 100 or more. If it is, then the Boolean value will become true.

The add passenger function takes in a passenger type object as a parameter. It checks first whether the passenger array list has reached maximum capacity by using the is full capacity function. If the function returns false, then it will add the passenger's full name into the array list.

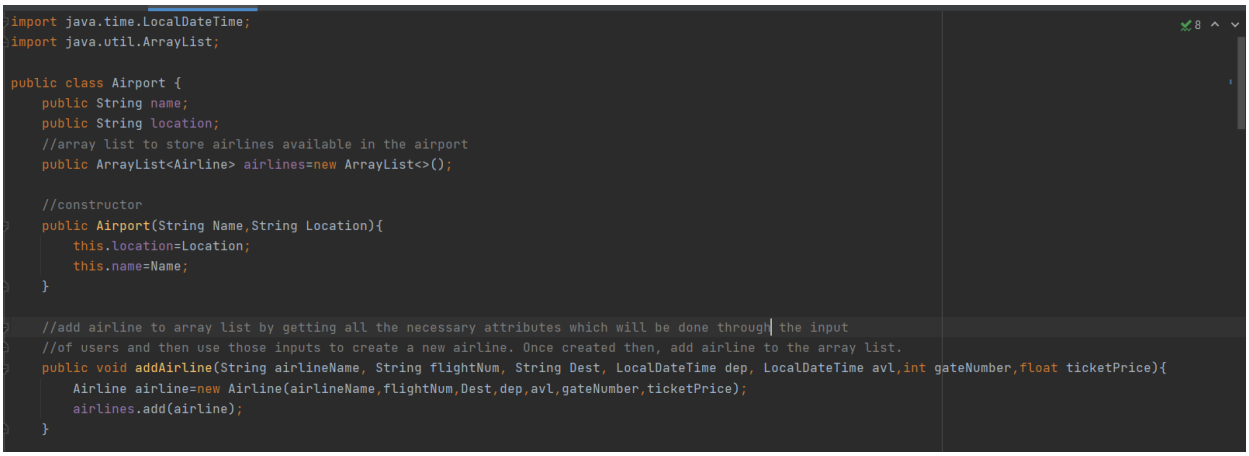
TICKET CLASS



```java
import java.time.LocalDateTime;

public class Ticket extends Passenger implements Details{
    private String TairlineName;
    private String TflightNumber;
    private String TDestination;
    private LocalDateTime Tdeparture;
    private LocalDateTime TestArrival;
    private int TgateNum;

    //constructor
    public Ticket(String firstName, String lastName, int age, String nationality, String airlineName,String flightNumber,
                  String Destination,LocalDateTime departure, LocalDateTime estArrival,int gateNum,boolean checkedIn,boolean immigrationStamped, String gender)
        super(firstName, lastName, age, nationality,checkedIn,immigrationStamped,gender);
        this.TairlineName=airlineName;
        this.TflightNumber=flightNumber;
        this.TDestination=Destination;
        this.Tdeparture=departure;
        this.TestArrival=estArrival;
        this.TgateNum=gateNum;
    }
```

The ticket class will have the same attributes as the airline class but it will be unique to the ticket class. This is why there is a T at the start of every attribute to indicate that it's from the ticket class. The ticket class also extends the passenger class as the passenger details will be needed for the ticket. It also implements the details interface.

AIRPORT CLASS



```java
import java.time.LocalDateTime;
import java.util.ArrayList;

public class Airport {
    public String name;
    public String location;
    //array list to store airlines available in the airport
    public ArrayList<Airline> airlines=new ArrayList<>();

    //constructor
    public Airport(String Name,String Location){
        this.location=Location;
        this.name=Name;
    }

    //add airline to array list by getting all the necessary attributes which will be done through the input
    //of users and then use those inputs to create a new airline. Once created then, add airline to the array list.
    public void addAirline(String airlineName, String flightNum, String Dest, LocalDateTime dep, LocalDateTime avl,int gateNumber,float ticketPrice){
        Airline airline=new Airline(airlineName,flightNum,Dest,dep,avl,gateNumber,ticketPrice);
        airlines.add(airline);
    }
```

The airport class will be used to create the airport object. It only has a name and location as it's attributes. It has an array list to store the airlines that are available in the airport. The airport class will contain most of the service functions that the user will use later on in the driver program.

The add airline function is used to set up the airport by creating airline objects and adding them to the airline array list that can be accessible later on.

```java
//iterate through the airlines array list and call the getters of the airline stored to get the details of each
// airline in the airport
public void displayAirlines() {
    int counter=1;
    for (Airline airline : airlines) {
        System.out.println(counter+". ");
        System.out.println("Flight Name: "+airline.getFlightName());
        System.out.println("Flight Number: "+airline.getFlightNumber());
        System.out.println("Destination: "+airline.getDestination());
        System.out.println("Departure date and time: "+airline.getDeparture());
        System.out.println("Arrival date and time: "+airline.getEstArrival());
        System.out.println("Gate Number: "+airline.getGateNum());
        System.out.println("Ticket Price: "+airline.getTicketPrice());
        System.out.println();
        counter++;
    }
}
```

The display airline function will loop through the airlines list and get each airline object. Then for each airline object I use the getters from the airline class to get the details of the flight and print it out.

```java
// An overlaoded method which takes in flightName, destination and max price
// This function will filter the airlines array list using the input given by the user and it will only display
// those airlines that matches the criteria given by the user.
public boolean displayAirlines(String flightName,String destination,float maxPrice) {
    int counter=0;
    //set boolean to TRUE
    boolean successful=Boolean.TRUE;
    for (Airline airline : airlines) {
        //if all three parameters are filled
        if(airline.getFlightName().equalsIgnoreCase(flightName) &&
                airline.getDestination().equalsIgnoreCase(destination) &&airline.getTicketPrice()<=maxPrice){
            counter++;
            System.out.println(counter+". ");
            System.out.println("Flight Name: "+airline.getFlightName());
            System.out.println("Flight Number: "+airline.getFlightNumber());
            System.out.println("Destination: "+airline.getDestination());
            System.out.println("Departure date and time: "+airline.getDeparture());
            System.out.println("Arrival date and time: "+airline.getEstArrival());
            System.out.println("Gate Number: "+airline.getGateNum());
            System.out.println("Ticket Price: "+airline.getTicketPrice());
            System.out.println();
```

This is an overloaded function of the display airlines above and it takes in flight name, destination and max price. A counter variable is also initialized along side with a boolean variable. This function does the same thing as the original display airlines function but it customizes which airlines to be shown by using the input of the user which will be passed in as parameters of the function. The function will check first if the parameters are present in the

airline and if it is present then it will print it out. Now there are a couple of variations as to how the user may input.

I want to give the user freedom so they can leave some criterias blank. For example if they are just looking flights to bali then they can just fill in the destination part. Using the 3 parameters I have made all the possible out comes:

```
if(airline.getFlightName().equalsIgnoreCase(flightName) &&
        airline.getDestination().equalsIgnoreCase(destination) &&airline.getTicketPrice()<=maxPrice)
```
If the user fills up all of the parameters

```
else if(flightName.equals("")&&airline.getDestination().equalsIgnoreCase(destination)
        &&airline.getTicketPrice()<=maxPrice)
```
if the user only skipped flight name

```
else
if(destination.equals("")&&airline.getFlightName().equalsIgnoreCase(flightName)&&airline.getTicketPrice(
)<=maxPrice)
```
if the user only skips the destination part

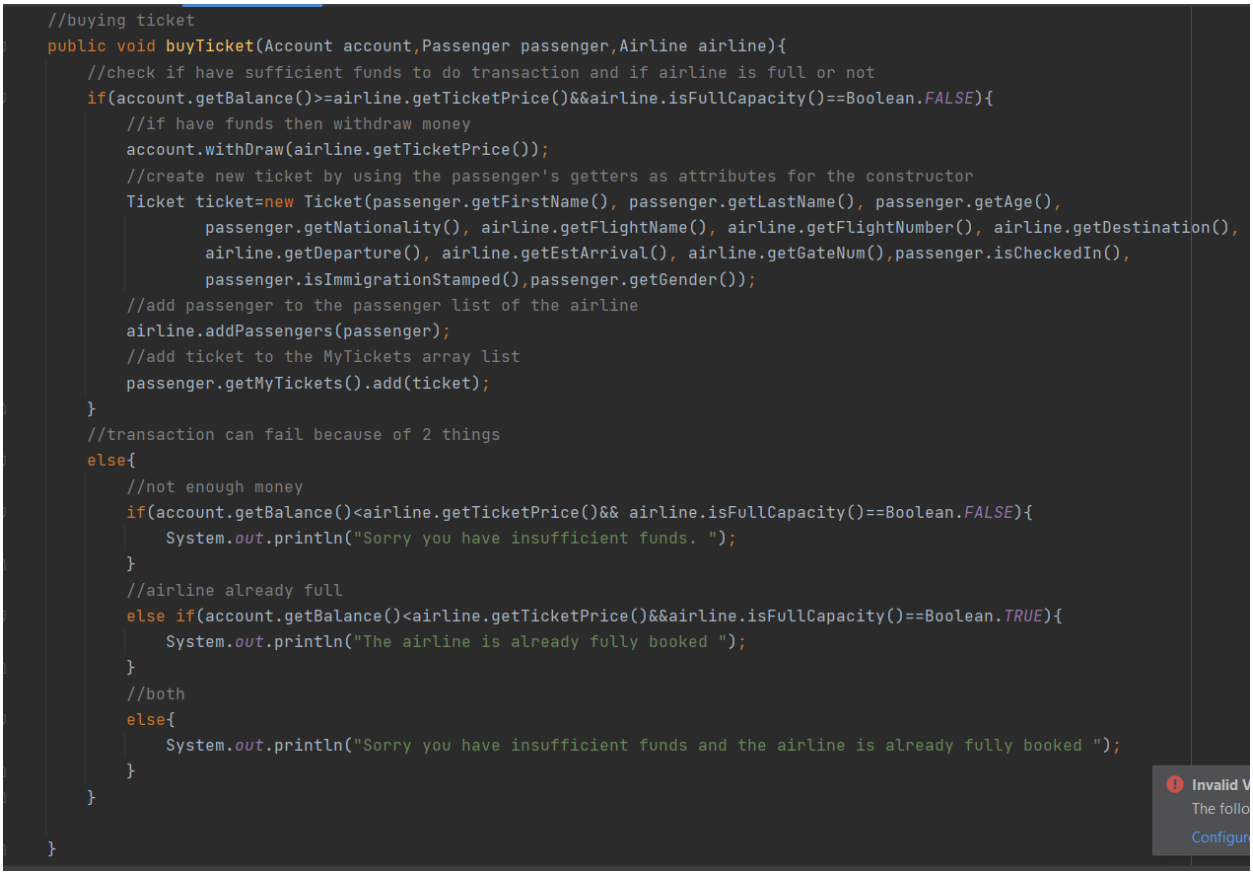
```
else
if(maxPrice==0&&airline.getFlightName().equalsIgnoreCase(flightName)&&airline.getDestination().equals
IgnoreCase(destination))
```
if the user only skips the max price

```
else if(flightName.equals("")&&destination.equals("")
        &&airline.getTicketPrice()<=maxPrice)
```
if the user only enters the max price

```
else if(flightName.equals("")&&airline.getDestination().equalsIgnoreCase(destination)
        &&maxPrice==0)
```
if the user enters only destination

```
else if(destination.equals("")&&airline.getFlightName().equalsIgnoreCase(flightName)
        &&maxPrice==0)
```
if the users only entered the flight name

The function will keep on looping through and for each iteration it will check with these conditions whether or not any of the conditions are met. Every time an airline is found the counter will be added by one. Once the looping is finished if the counter is still 0, this means that no airlines matches the criteria specified by the user and the function will return False. If there are airlines found then it will return true.



```java
//buying ticket
public void buyTicket(Account account,Passenger passenger,Airline airline){
    //check if have sufficient funds to do transaction and if airline is full or not
    if(account.getBalance()>=airline.getTicketPrice()&&airline.isFullCapacity()==Boolean.FALSE){
        //if have funds then withdraw money
        account.withDraw(airline.getTicketPrice());
        //create new ticket by using the passenger's getters as attributes for the constructor
        Ticket ticket=new Ticket(passenger.getFirstName(), passenger.getLastName(), passenger.getAge(),
                passenger.getNationality(), airline.getFlightName(), airline.getFlightNumber(), airline.getDestination(),
                airline.getDeparture(), airline.getEstArrival(), airline.getGateNum(),passenger.isCheckedIn(),
                passenger.isImmigrationStamped(),passenger.getGender());
        //add passenger to the passenger list of the airline
        airline.addPassengers(passenger);
        //add ticket to the MyTickets array list
        passenger.getMyTickets().add(ticket);
    }
    //transaction can fail because of 2 things
    else{
        //not enough money
        if(account.getBalance()<airline.getTicketPrice()&& airline.isFullCapacity()==Boolean.FALSE){
            System.out.println("Sorry you have insufficient funds. ");
        }
        //airline already full
        else if(account.getBalance()<airline.getTicketPrice()&&airline.isFullCapacity()==Boolean.TRUE){
            System.out.println("The airline is already fully booked ");
        }
        //both
        else{
            System.out.println("Sorry you have insufficient funds and the airline is already fully booked ");
        }
    }

}
```

This function is called whenever the user will buy a plane ticket. It takes in an account object, passenger object and an airline object. First it checks whether the balance in the account is smaller or equal to the ticket price of the airline object. It also checks whether the airline object is at full capacity or not. Then the withdraw function from the account will be called to withdraw the amount of the ticket price from the account. Then a new ticket object will be created using the passenger details by calling the get methods and set it as the attributes. Finally I add the passenger to the passenger list of the airline and the ticket to the my tickets array. The else statement will be called if either the airline is fully booked or the account has insufficient funds.

DRIVER CLASS

```
//initializing all required variables
int choice;
int subChoice;
int val;
int numPut = 0;
String firstname;
String lastname;
int age;
String nationality;
String gender;
//using scanner for user input
Scanner sc = new Scanner(System.in);
//creating airport object
Airport ePort=new Airport( Name: "E-PORT", Location: "JAKARTA");
//adding airlines to airline array list in airport object
ePort.addAirline( airlineName: "AIRMAX", flightNum: "AMX023", Dest: "Bali",
        LocalDateTime.of( year: 2021, month: 10, dayOfMonth: 15, hour: 12, minute: 15),
        LocalDateTime.of( year: 2021, month: 10, dayOfMonth: 15, hour: 15, minute: 15), gateNumber: 1, ticketPrice: 100);
ePort.addAirline( airlineName: "AIRMAX", flightNum: "AMX025", Dest: "Paris",
        LocalDateTime.of( year: 2021, month: 10, dayOfMonth: 15, hour: 12, minute: 15),
        LocalDateTime.of( year: 2021, month: 10, dayOfMonth: 15, hour: 23, minute: 15), gateNumber: 2, ticketPrice: 210);
ePort.addAirline( airlineName: "AIRJORDAN", flightNum: "AJD025", Dest: "Tokyo",
        LocalDateTime.of( year: 2021, month: 10, dayOfMonth: 15, hour: 12, minute: 15),
        LocalDateTime.of( year: 2021, month: 10, dayOfMonth: 15, hour: 19, minute: 15), gateNumber: 3, ticketPrice: 180);
ePort.addAirline( airlineName: "AIRJORDAN", flightNum: "AJD023", Dest: "Bali",
        LocalDateTime.of( year: 2021, month: 10, dayOfMonth: 15, hour: 15, minute: 15),
        LocalDateTime.of( year: 2021, month: 10, dayOfMonth: 15, hour: 18, minute: 15), gateNumber: 2, ticketPrice: 120);
ePort.addAirline( airlineName: "AIRFORCE", flightNum: "AFE237", Dest: "London",
        LocalDateTime.of( year: 2021, month: 10, dayOfMonth: 15, hour: 12, minute: 15),
        LocalDateTime.of( year: 2021, month: 10, dayOfMonth: 15, hour: 20, minute: 15), gateNumber: 4, ticketPrice: 180);
ePort.addAirline( airlineName: "AIRFORCE", flightNum: "AFE190", Dest: "New York",
        LocalDateTime.of( year: 2021, month: 10, dayOfMonth: 15, hour: 12, minute: 15),
        LocalDateTime.of( year: 2021, month: 10, dayOfMonth: 16, hour: 8, minute: 15), gateNumber: 5, ticketPrice: 250);
```

The driver class is where the code is all combined and run. First I created an airport object using the airport class and instantiated a couple of variables that are going to be used later on in the program. Then, I added airlines to the airport by calling the add ariline function which will allow me to create airline objects as well. Im also going to be using a scanner for input.

```
//using a do-while loop and try/catch to make sure users input an integer
do {
    sc.nextLine();
    try {
        System.out.println("Enter your age: ");
        numPut=sc.nextInt();
        val=0;
    //this will catch any non-integer input and make val==1 so that it keeps looping until correct input is made
    }catch (InputMismatchException e){
        val=1;
        System.out.println("Please enter an integer.");
    }
}while (val==1);
```

After seting up the airport, the user will be instructed to sign up by inputting a couple of details such as their first name, last name, age, gender and nationality. This code above shows the input for the age. Since the only sensible input for age is an integer, I used a try and catch block to catch any input error such as floats and strings. First a do while loop is created so that if the user enters a wrong input then they will try to input again. In the try block the user will enter their input and I set the value of val to 0. Then in the catch block is where I catch the exception when the user enters a wrong input and this will set the value of val equal to 1. The reason behind this is because the do while loop will keep running if val is equal to 1 which means that theres an error or a wrong input. If the user enters an input and the program doesn't catch anything then the input is correct and the value will still be 0 and it will break the loop. The exception that I used is the INPUTMISMATCHEXCEPTION which is an exception that catches any wrong-type input.

```
//using do-while loop to create menus
do {
    //MAIN MENU
    System.out.println("---E-PORT MENU---");
    System.out.println("1. BOOK A FLIGHT");
    System.out.println("2. CHECK IN/IMMIGRATION");
    System.out.println("3. DEPOSIT");
    System.out.println("4. ACCOUNT INFO");
    System.out.println("5. CANCEL BOOKING");
    System.out.println("6. EXIT/LOGOUT");
    System.out.println();
    //taking user input for navigating the menu
    choice=sc.nextInt();
```

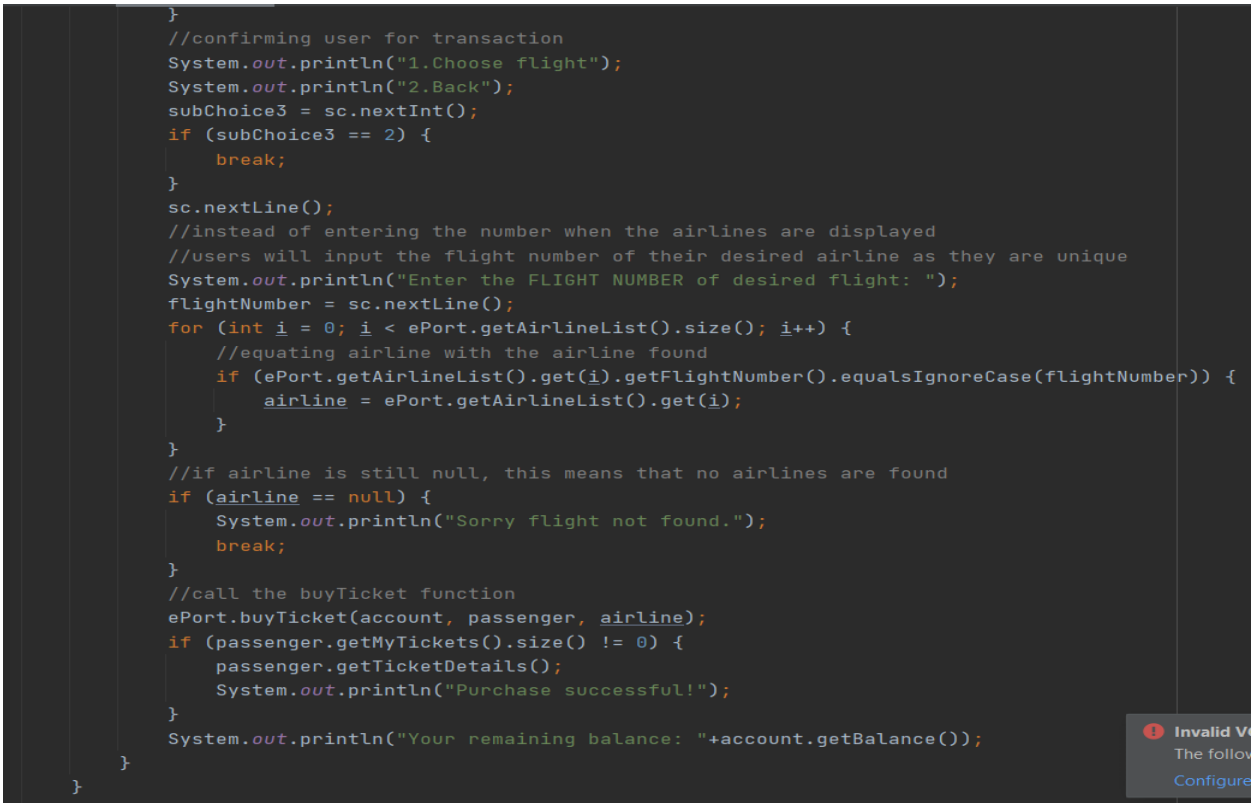
After the user sign up process, the user will be brought into the main menu where they can navigate through the program using numbers

```
do {
    System.out.println("BOOK A FLIGHT");
    System.out.println("1. VIEW FLIGHTS");
    System.out.println("2. CUSTOM SEARCH");
    System.out.println("3. BACK");
    subChoice = sc.nextInt();
    switch (subChoice) {
        //displaying all available flights
        case 1 -> {
            int chooseFlight;
            //calling the display function from the airport class
            ePort.displayAirlines();
            int subChoice3;
            //confirming with user whether they want to make a transaction
            System.out.println("1.Choose flight");
            System.out.println("2.Back");
            subChoice3 = sc.nextInt();
            if (subChoice3 == 2) {
                break;
            }
            //having users pick their flights
            System.out.println("Enter the number of desired flight: ");
            chooseFlight = sc.nextInt();
            //calling the buyTicket function from the airport class
            //since the flights are stored in an array, I decremented the chooseFlight variable
            //by 1 since indices starts at 0
            ePort.buyTicket(account, passenger, ePort.getAirlineList().get(chooseFlight - 1));
            if (passenger.getMyTickets().size() != 0) {
                passenger.getTicketDetails();
                System.out.println("Purchase successful!");
            }
            System.out.println("Your remaining balance: "+account.getBalance());
```

For the book a flight menu, this will be the code for when the user chooses to buy a ticket for themselves. The reason why buying tickets for someone else is separated from this is because buying tickets for oneself is easier since we do not need to create another passenger object. The user will then be brought to another sub menu where the program will provide a show all airlines option or show a customized search option. The code is for the show all option which is why the display airlines function will be called. Once the airlines are shown on the screen, the user can either choose to proceed with picking their flights or they can go back to the sub menu. If they want to proceed then they will choose the flight by inputting an integer which will be the order of airlines shown on the screen. Then the buy ticket function will be called using the account object that was created during the sign up, passenger object also during sign up and the airline object will be obtained from the access of the airline list by using the user input as the index because the order which it will be displayed will also be the same as the one stored in the array list. The user input will be decreased by 1 because indices start at 0 and there is no 0 order. In order to check whether the purchase is a success, the program will check whether the size of the my tickets array of the passenger object is 0 or not. If it is not 0, then the ticket details will be printed out using the get ticket details function from the passenger class. The balance will also be printed out.

```
//custom search
case 2 -> {
    //initialize local variables
    String flightName;
    String destination;
    float maxPrice;
    int subChoice3;
    //initializing new airline so that we can equalize it with the search result
    Airline airline = new Airline();
    //entering the necessary details for the custom search
    String flightNumber;
    System.out.println("Enter flight Name: ");
    sc.nextLine();
    flightName = sc.nextLine();
    System.out.println("Enter destination: ");
    destination = sc.nextLine();
    System.out.println("Enter max price (enter 0 to skip): ");
    maxPrice = sc.nextFloat();
    sc.nextLine();
    //calling the display function but this time overloading with the inputs
    //if the function return false it means that no airline is found, hence we break
    if (ePort.displayAirlines(flightName, destination, maxPrice) == Boolean.FALSE) {
        System.out.println("sorry no flights found.");
        break;
    }
```

The second option is to do a custom search. For this I have intialized local variables fro the flight name, destination and max price. An airline object using the default constructor is also instantiated here because it will be used for the buying of tickets later on. The default constructor is used because the user have not picked any flights. The user will be instructed to input the flight name, destination and max price but they can choose to skip some of the criterias if they wish to do so. This time instead of calling the normal display function, an overload of that function will be called using the local variables that were made. This will return false if no airlines matches any criteria of the airlines stored in the array and it will return true if an airline is found.



```
            }
            //confirming user for transaction
            System.out.println("1.Choose flight");
            System.out.println("2.Back");
            subChoice3 = sc.nextInt();
            if (subChoice3 == 2) {
                break;
            }
            sc.nextLine();
            //instead of entering the number when the airlines are displayed
            //users will input the flight number of their desired airline as they are unique
            System.out.println("Enter the FLIGHT NUMBER of desired flight: ");
            flightNumber = sc.nextLine();
            for (int i = 0; i < ePort.getAirlineList().size(); i++) {
                //equating airline with the airline found
                if (ePort.getAirlineList().get(i).getFlightNumber().equalsIgnoreCase(flightNumber)) {
                    airline = ePort.getAirlineList().get(i);
                }
            }
            //if airline is still null, this means that no airlines are found
            if (airline == null) {
                System.out.println("Sorry flight not found.");
                break;
            }
            //call the buyTicket function
            ePort.buyTicket(account, passenger, airline);
            if (passenger.getMyTickets().size() != 0) {
                passenger.getTicketDetails();
                System.out.println("Purchase successful!");
            }
            System.out.println("Your remaining balance: "+account.getBalance());
        }
    }
```

The same process as the display all flights option will be run but when choosing the airline, the user will have to enter the flight number instead of a number because the order will be different from the order of the airlines array. Then the for loop will look for a matching flight number in the airlines list and instantiate the airline object as the airline with the same flight number. Flight number was used because they are unique for every flight. If the airline is still a null object then no airlines matches the inputted flight number.

```java
//inputs details
// calling the setters to set the attributes of the new passenger object with the inputs
System.out.println("Please enter your first name: ");
sc.nextLine();
String firstname1 = sc.nextLine();
passenger1.setFirstName(firstname1);
System.out.println("Please enter your last name: ");
String lastname1 = sc.nextLine();
passenger1.setLastName(lastname1);
//using a do-while loop and try/catch to make sure users input an integer
do {
    sc.nextLine();
    try {
        System.out.println("Enter your age: ");
        numPut=sc.nextInt();
        val=0;
        //this will catch any non-integer input and make val==1 so that it keeps looping until correct input is made
    }catch (InputMismatchException e){
        val=1;
        System.out.println("Please enter an integer.");
    }
}while (val==1);
int age1=numPut;
sc.nextLine();
passenger1.setAge(age1);
System.out.println("Enter your nationality: ");
String nationality1 = sc.nextLine();
passenger1.setNationality(nationality1);
System.out.println("Enter your gender: ");
String gender1 = sc.nextLine();
passenger1.setGender(gender1);
passenger1.setCheckedIn(Boolean.FALSE);
passenger1.setImmigrationStamped(Boolean.FALSE);
```


If the user wants to buy a ticket for someone else then they will be brought to this menu. User will enter the details of the person that they want to buy the ticket for and then using the setters, the null passenger object will be set with those attributes.

```java
switch (choice) {
    //buying tickets
    case 1 -> {
        int yesOrNo;
        ///initializing new passenger object
        Passenger passenger1 = new Passenger();
        //asking user whether they want to buy a ticket for themselves or for someone else
        System.out.println("""
                Are you purchasing a personal ticket?(if no, you will be filling a form for the person you are buying the ticket for.)
                1. Yes, I am purchasing a ticket for myself
                2. No, I am purchasing a ticket for someone else.
                Please enter either the number 1 or 2""");
        yesOrNo = sc.nextInt();
```

Passenger object created with null constructor or default constructor.

```java
//displaying all the flights
case 1 -> {
    int chooseFlight;
    //calling display function
    ePort.displayAirlines();
    int subChoice3;
    //confirming with user
    System.out.println("1.Choose flight");
    System.out.println("2.Back");
    subChoice3 = sc.nextInt();
    if (subChoice3 == 2) {
        break;
    }
    System.out.println("Enter the number of desired flight: ");
    chooseFlight = sc.nextInt();
    //calling the buy function by using passenger1 object
    ePort.buyTicket(account, passenger1, ePort.getAirlineList().get(chooseFlight - 1));
    //if the length of the array is not 0, this means that a ticket is added
    //which means that the purchase was successful
    if (passenger1.getMyTickets().size() != 0) {
        passenger1.getTicketDetails();
        System.out.println("Purchase successful!");
        //add passenger1 to the array of the tickets that passenger bought
        for (int i = 0; i < passenger1.getMyTickets().size(); i++) {
            passenger.getTicketsIBought().add(passenger1);
        }
    }
    System.out.println("Your remaining balance: "+account.getBalance());
```

The rest of the program for this sub menu will be the same as when the user wants to buy a ticket for themselves. The only difference is that when buying a ticket, the new passenger object will be used and once the transaction is complete, the passenger will be added to the array list of “tickets that I bought” which will store passengers whoose tickets the user bought for.

```
//CHECK IN AND IMMIGRATION
case 2 -> {
    //check if passenger have tickets
    if (passenger.getMyTickets().size() == 0) {
        System.out.println("You have not purchased any tickets");
        break;
    }
    if (passenger.isCheckedIn()==Boolean.TRUE){
        System.out.println("You have already checked in");
        break;
    }
    //display ticket details and confirm check in
    passenger.getTicketDetails();
    System.out.println("Confirm check in?");
    System.out.println("1. Yes I want to confirm.");
    System.out.println("2. Back");
    int choice2;
    choice2 = sc.nextInt();
    //change the checkedIn and Immigration attribute to True
    if (choice2 == 1) {
        passenger.setCheckedIn(Boolean.TRUE);
        passenger.setImmigrationStamped(Boolean.TRUE);
    }
    System.out.println("Check in status: "+passenger.isCheckedIn());
}
```

This part of the program is responsible for the check in and immigration process. It is fairly simple as it sets the boolean attribute of the passenger which is the “is checked in” and “is immigration stamped” variables to true. However before that the program will check if the passenger has bought a ticket and if the boolean attribute is still false indicating that the passenger has not checked in. Once the passenger passes these conditions then the program will change the attribute to true.

```
}
//Deposit
case 3 -> {
    float amt;
    System.out.println("Enter deposit amount: ");
    amt = sc.nextFloat();
    //call the deposit function from the account class
    account.deposit(amt);
}
```

This part of the program will handle the deposit of any funds into the account of the user. It will ask an input from the user and call the deposit method from the account class using the user input as the parameter.

```java
//showing account info
case 4 -> {
    System.out.println("Here are your account details: ");
    //get details of passenger such as name, age and etc.
    passenger.getDetails();
    //for showing account balance
    System.out.println("Your balance in your account is: " + account.getBalance());
    System.out.println();
    System.out.println("My Ticket Details: ");
    //get details of users own ticket which includes flightName,flightNumber,Destination and etc.
    passenger.getTicketDetails();
    System.out.println("Other Tickets I bought: ");
    //get tickets that were bought for someone else and calling the get ticket details to get the details
    // of every ticket
    for (int i = 0; i < passenger.getTicketsIBought().size(); i++) {
        passenger.getTicketsIBought().get(i).getTicketDetails();
    }
}
```

This part of the program will show the account details and the details of tickets that were bought. First it will use the passenger object to call the get details method which will print out the passenger's or the user's personal details. It will then print out the balance of the account. Then, it will print out the ticket details by calling the get ticket details function. Lastly it will print out the ticket details of the people that the user bought the ticket for by looping the "tickets I bought" array and calling the get ticket details of each passenger.

```java
//cancelling tickets
case 5 -> {
    int subChoice1;
    //check if passenger has bought any tickets
    if (passenger.getMyTickets().size() == 0 && passenger.getTicketsIBought().size() == 0) {
        System.out.println("Sorry you have not purchased any tickets");
        break;
    }
    //ask whether they want to cancel their own tickets or the tickets that they bought for someone else
    System.out.println("Confirm cancellation of booking: ");
    System.out.println("1. I want to cancel my own ticket.");
    System.out.println("2. I want to cancel a ticket that I bought for someone else. ");
    System.out.println("3. Back");
    subChoice1 = sc.nextInt();
    sc.nextLine();
    //cancelling users own ticket
    if (subChoice1 == 1) {
        //check whether passenger has checked in, if yes then refund is not possible
        if (passenger.isCheckedIn() == Boolean.TRUE) {
            System.out.println("Refund is not available because you have checked in.");
            break;
        }
        //check if there are any tickets in the my tickets array
        if(passenger.getMyTickets().size()==0){
            System.out.println("Sorry you have not purchased any tickets");
            break;
        }
        //show ticket details
        passenger.getTicketDetails();
        //user enters the flight number stated on their tickets to confirm cancellation
        System.out.println("Enter flight number of the ticket to be cancelled: ");
        String flightNumber;
        flightNumber = sc.nextLine();
```

This part of the program will be about the cancellation of tickets. Here it will first check if there are any tickets stored in the “my tickets” array and in the “tickets I bought” array. If both of them are 0, this means that no tickets have been purchased. Then, the user will be brought to a sub menu where the program is asking which type of ticket are the users wanting to cancel. The first option is the user’s own ticket and the second option is the tickets the user bought for someone else. This part of the program shows the cancellation of the user’s own ticket. It will check whether the passenger has checked in or not and then it will check again the “my tickets” array list to check if it has any tickets. Once both conditions are met, the ticket details will be shown and then the user will have to enter the flight number of the airline that they want to cancel.

```
//loops through the array of myTickets to check whether any tickets match the inputted flight
//number. If no match then print out else statement. If found then remove the ticket from the
//array
for (int i = 0; i < passenger.getMyTickets().size(); i++) {
    if (passenger.getMyTickets().get(i).getFlightNumber().equalsIgnoreCase(flightNumber)) {
        passenger.getMyTickets().remove(passenger.getMyTickets().get(i));
    } else {
        System.out.println("Ticket not found.");
        break;
    }
}
//loop through the array that stores the airlines and check if the inputted flight number
// matches the flight number inputted by the user. When found,remove passenger from the
//passenger list array of the airline object and then refund money by using the deposit
//function to the account
for (int j = 0; j < ePort.getAirlineList().size(); j++) {
    if (flightNumber.equalsIgnoreCase(ePort.getAirlineList().get(j).getFlightNumber())) {
        ePort.getAirlineList().get(j).getPassengerList().remove( o: passenger.getFirstName() + " " + passenger.getLastName());
        account.deposit(ePort.getAirlineList().get(j).getTicketPrice());
    }

}
```

After entering the flight number, the for loop will loop through the my tickets array and look for the ticket that matches the flight number that was inputted by the user. Once it is found, it will remove it from the array. If it is not found then the else statement will print out.

The next for loop takes care of the airline data base. It loops through the list of airlines in the airport airlines list and checks if there are any airlines that matches the flight number inputted by the user. Once it is found, it will remove the passenger name from the passenger list of that airline object. Lastly the funds for that purchase will be refunded by calling the deposit function using the ticket price as the parameter.

```
} else if (subChoice1 == 2) {
    //show all the tickets bought for other people
    for (int i = 0; i < passenger.getTicketsIBought().size(); i++) {
        passenger.getTicketsIBought().get(i).getTicketDetails();
    }
    //get input
    String flightNumber;
    String firstname1;
    String lastname1;
    System.out.println("Enter flight number of the ticket to be cancelled: ");
    flightNumber = sc.nextLine();
    System.out.println("Enter first name of passenger: ");
    firstname1 = sc.nextLine();
    System.out.println("Enter last name of passenger: ");
    lastname1 = sc.nextLine();
    //looping through the array of tickets bought for other people
    for (int i = 0; i < passenger.getTicketsIBought().size(); i++) {
        //finding the ticket that matches the first and last name inputted and flight number
        //if found then remove from array
        if (passenger.getTicketsIBought().get(i).getMyTickets().get(0).getFlightNumber().equalsIgnoreCase(flightNumber)
                && firstname1.equalsIgnoreCase(passenger.getTicketsIBought().get(i).getFirstName())
                && lastname1.equalsIgnoreCase(passenger.getTicketsIBought().get(i).getLastName())) {
            passenger.getTicketsIBought().remove(passenger.getTicketsIBought().get(i));
            }
        }
    //loop through the array that stores the airlines and check if the inputted flight number
    // matches the flight number inputted by the user. When found,remove passenger from the
    //passenger list array of the airline object and then refund money by using the deposit
    //function to the account
    for (int j = 0; j < ePort.getAirlineList().size(); j++) {
        if (flightNumber.equalsIgnoreCase(ePort.getAirlineList().get(j).getFlightNumber())) {
            ePort.getAirlineList().get(j).getPassengerList().remove( o: firstname1+" "+lastname1);
            account.deposit(ePort.getAirlineList().get(j).getTicketPrice());
```

This part of the program shows the cancellation of the tickets that the user bought for someone else. It goes through the same processes as the first one. However the only difference is that it takes in 2 extra parameters which is the first and last name of the passenger that the user bought the ticket for. Instead of finding a ticket that matches the flight number, now the loop also looks for a ticket that matches the first and last name that was inputted by the user. The array used here is also the “tickets I bought” array which stores the passengers that the user has bought tickets for. The rest of the process is the same. Once it has found a passenger that holds ticket details that have the first name, last name and flight number, it will remove the passenger from the array list. Then, it will remove the passenger from the list of passengers of the airline object and refund the funds used to purchase the ticket back to the account.

REFLECTION

When doing this project, there were a lot of setbacks especially when it comes to the custom search part of the program. It was a huge learning process for me as I found myself stuck and I did not know what to do. However, I was glad that I did not give up and it took me a few days to figure out the setbacks and problems. What I learned is that, coding needs patience and some times you can’t force an idea out to solve the coding issues that you face. You just have to relax for a momment and keep a clear mind.