**Final Project**

**Object Oriented Programming**

Project name: **E-PORT**

Name: **Sri Kalyan Rohan**

Student ID: **2440090266**

Class:

**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**

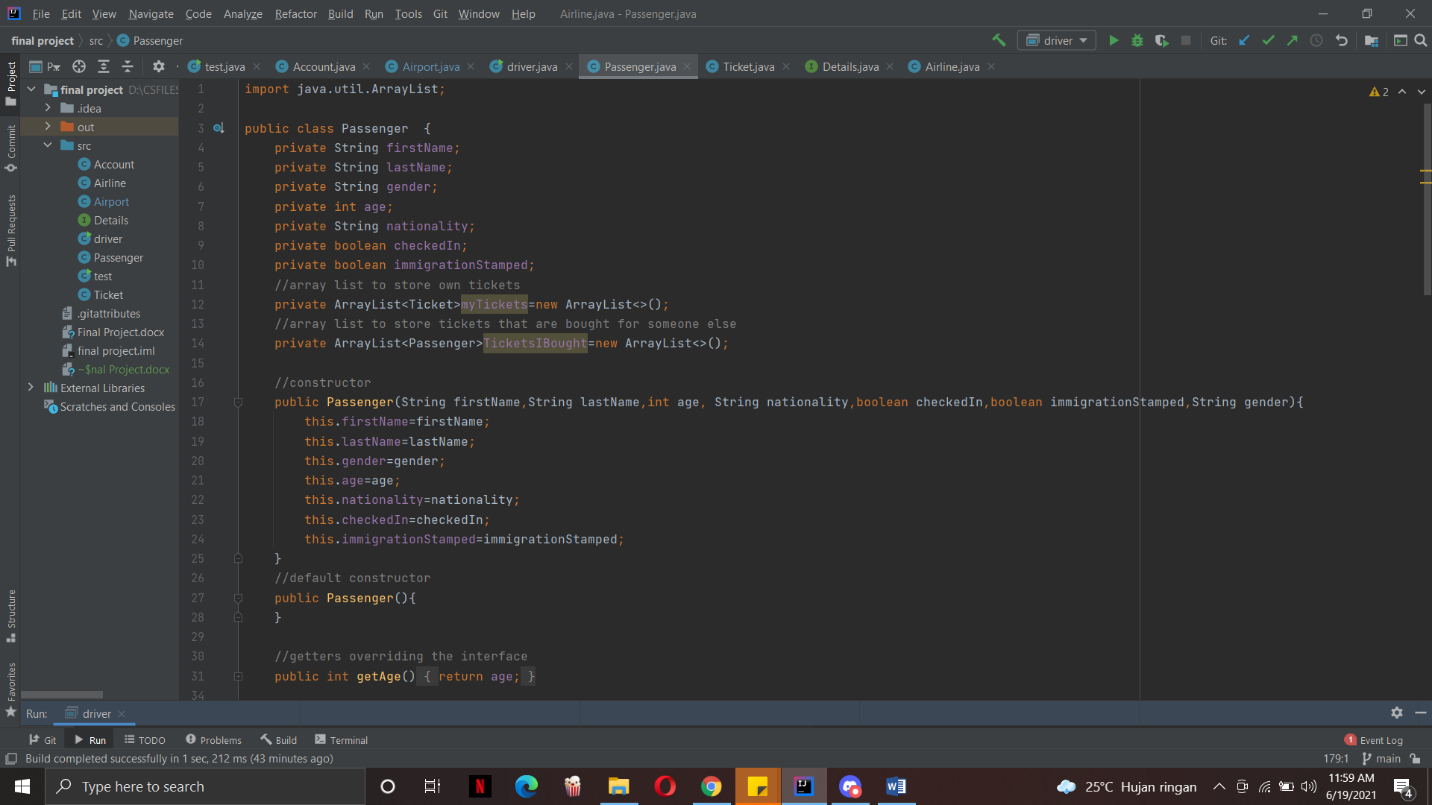
**CODE EXPLANATION**

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:

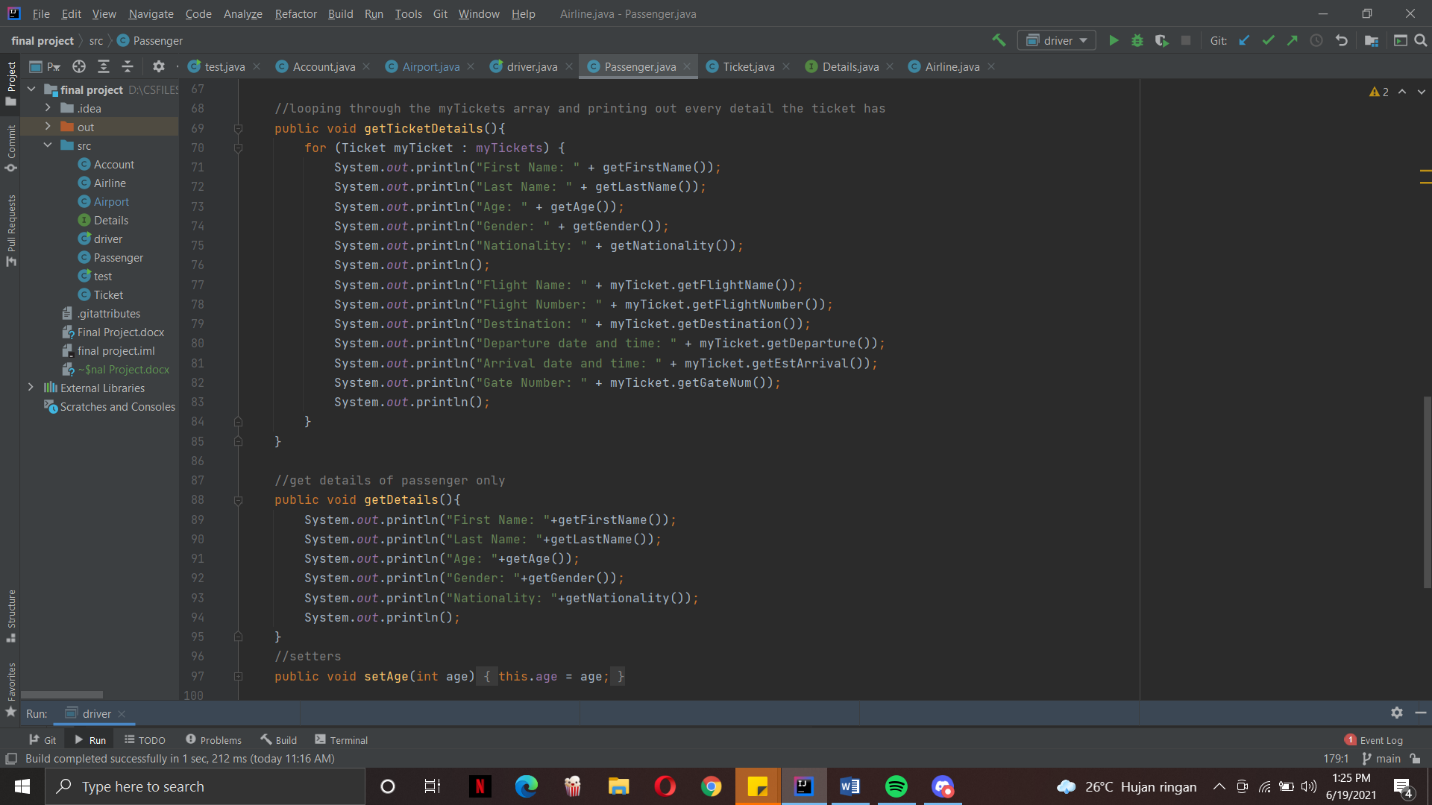
1. Passenger
2. Account
3. Ticket
4. Airline
5. Airport
6. Driver

Interface:

1. Details

**PASSENGER CLASS**

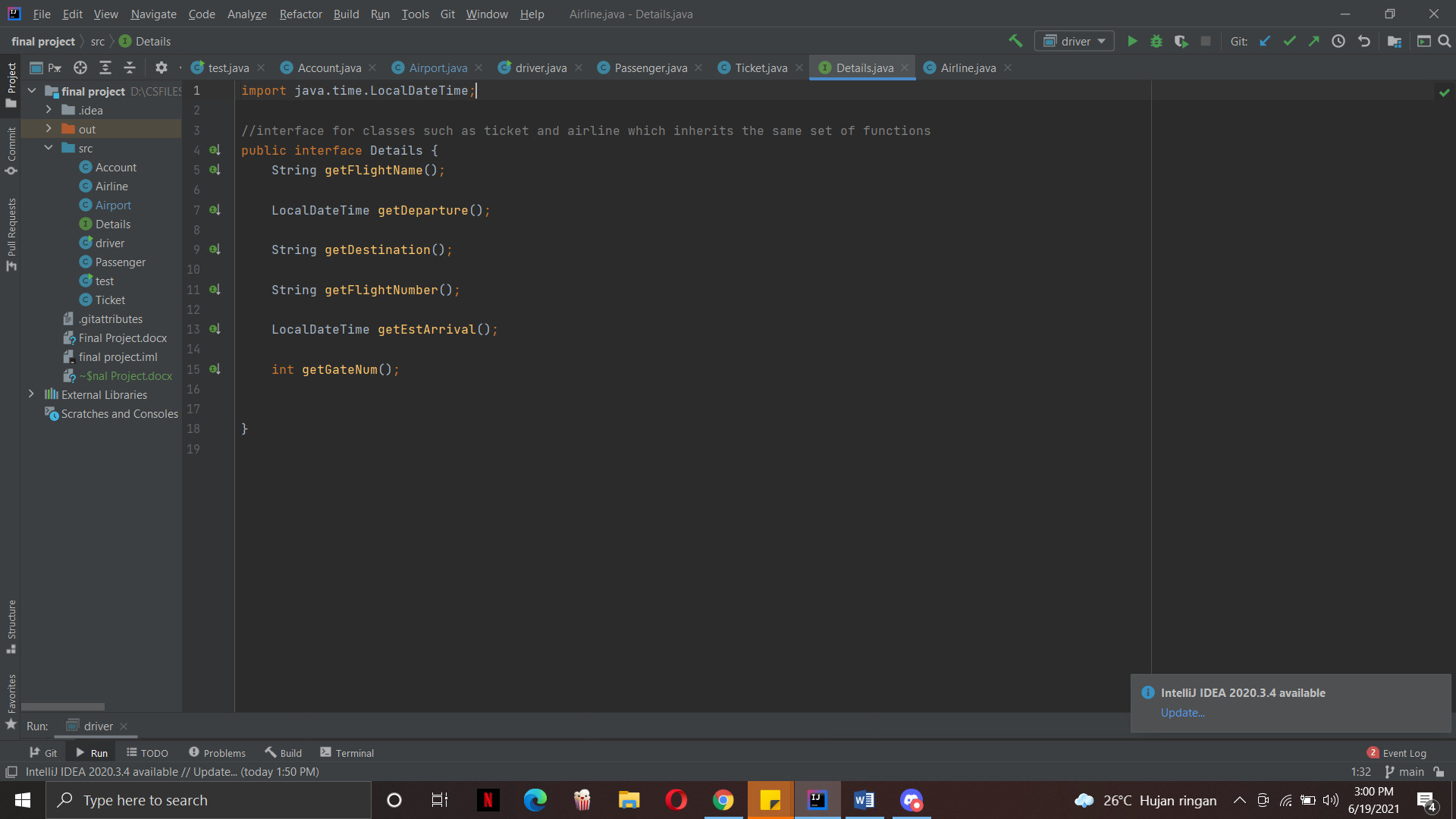
The passenger class is used to create passenger objects that holds the details of the users such as their first name, last name, age, gender and nationality. This class also indicates whether a passenger has checked in or not through a private Boolean variable. This goes the same for the immigration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

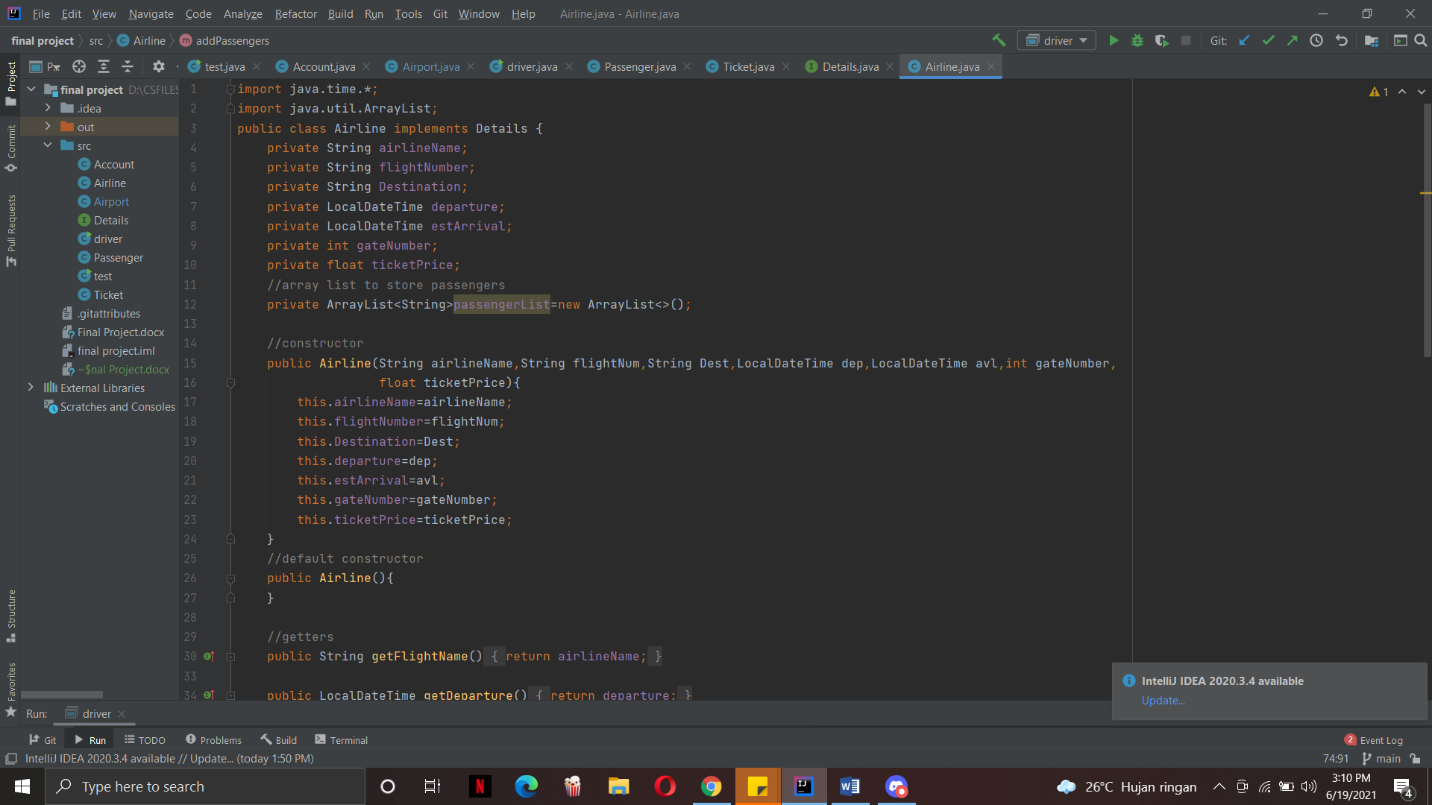
The get ticket details function will loop throught the my ticket array list and get the ticket object. Using the ticket object, the getters will be called so that the information of the ticket will be printed out. The get details function will just call the getters of the passenger class to get the details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

**ACCOUNT CLASS**

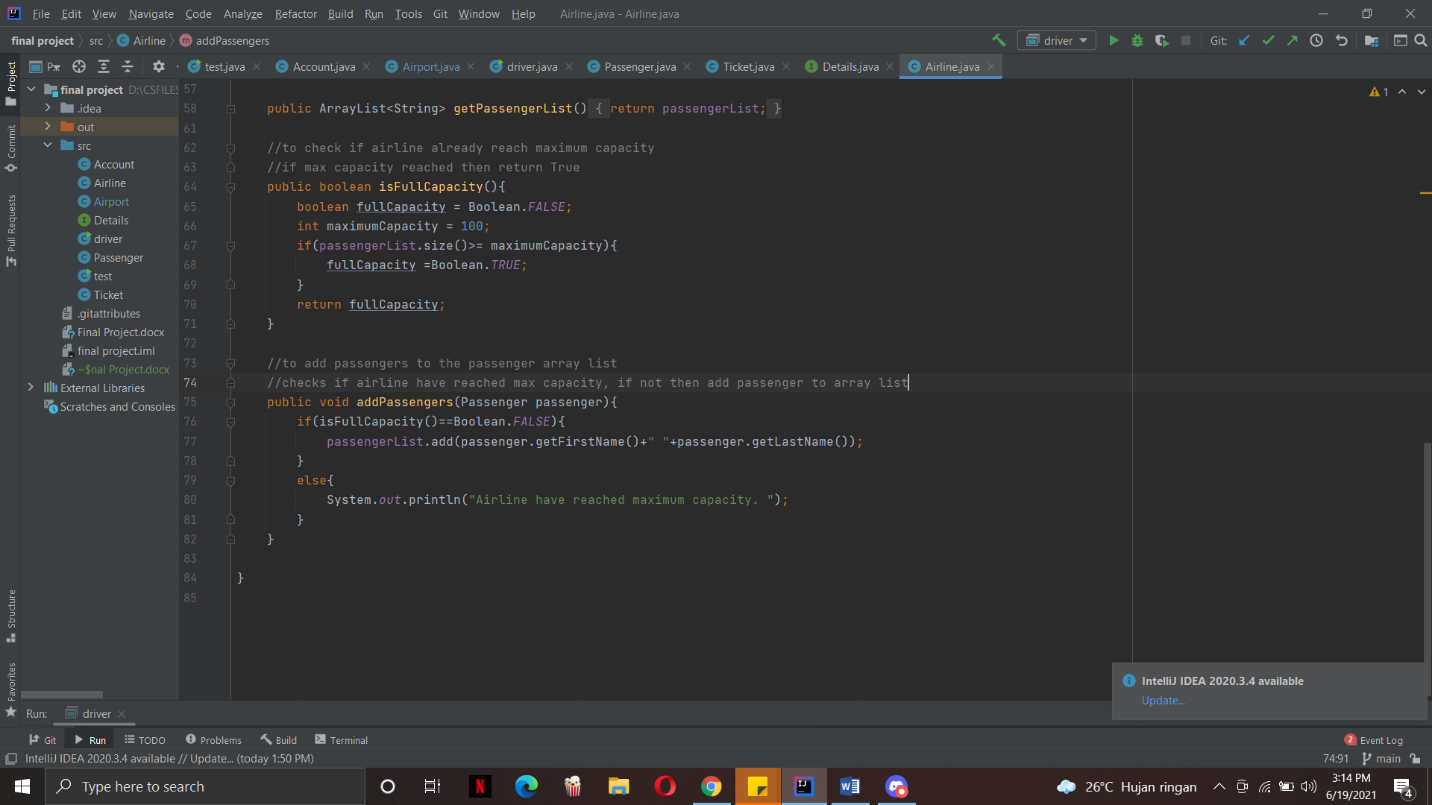
The account class extends the passenger class. It has the get balance function to return the balance of the account, deposit function to store money into the account by taking in a float parameter and a withdraw function which takes money out of the account if the float parameter is equal or lesser than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

**INTERFACE – DETAILS**

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.

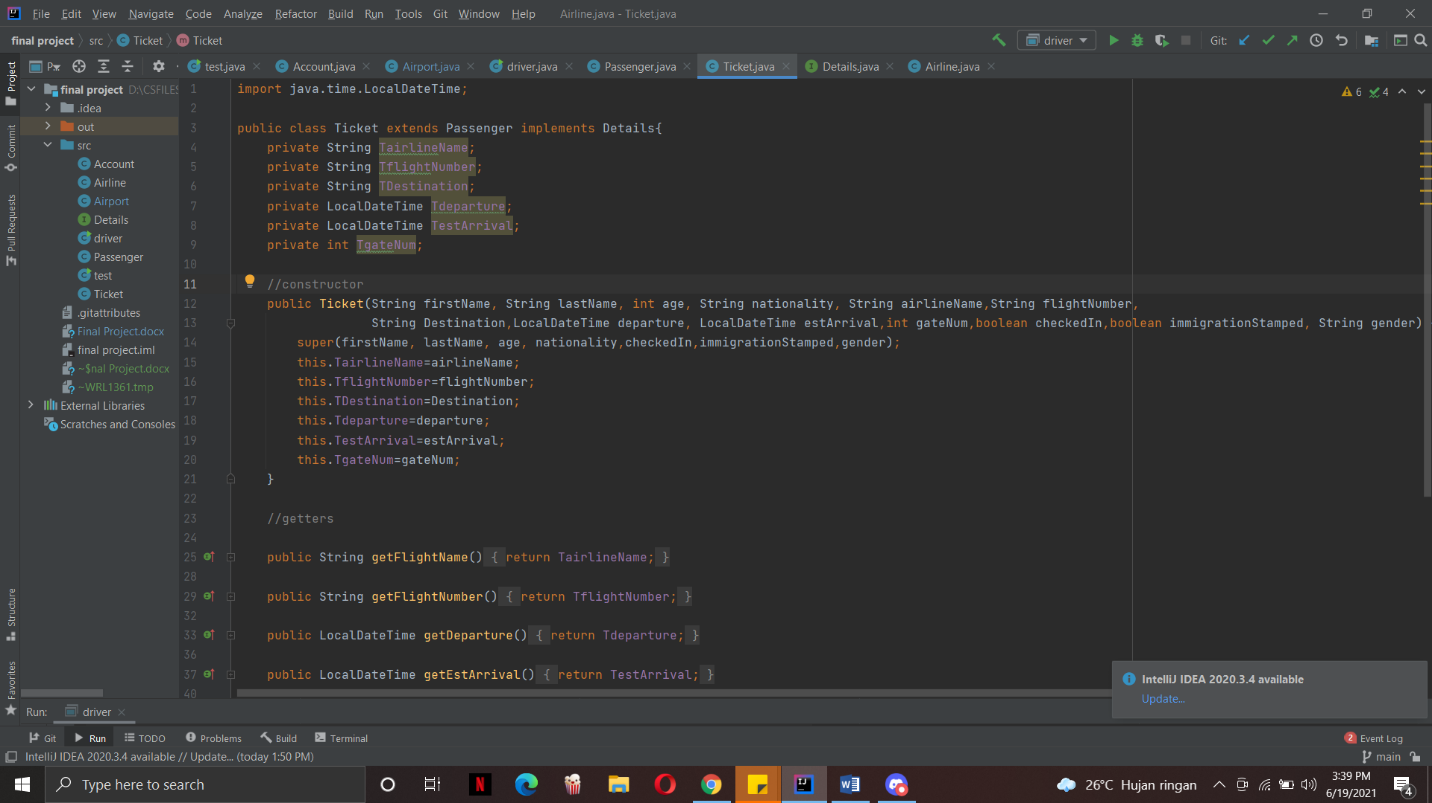
**AIRLINE CLASS**

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.

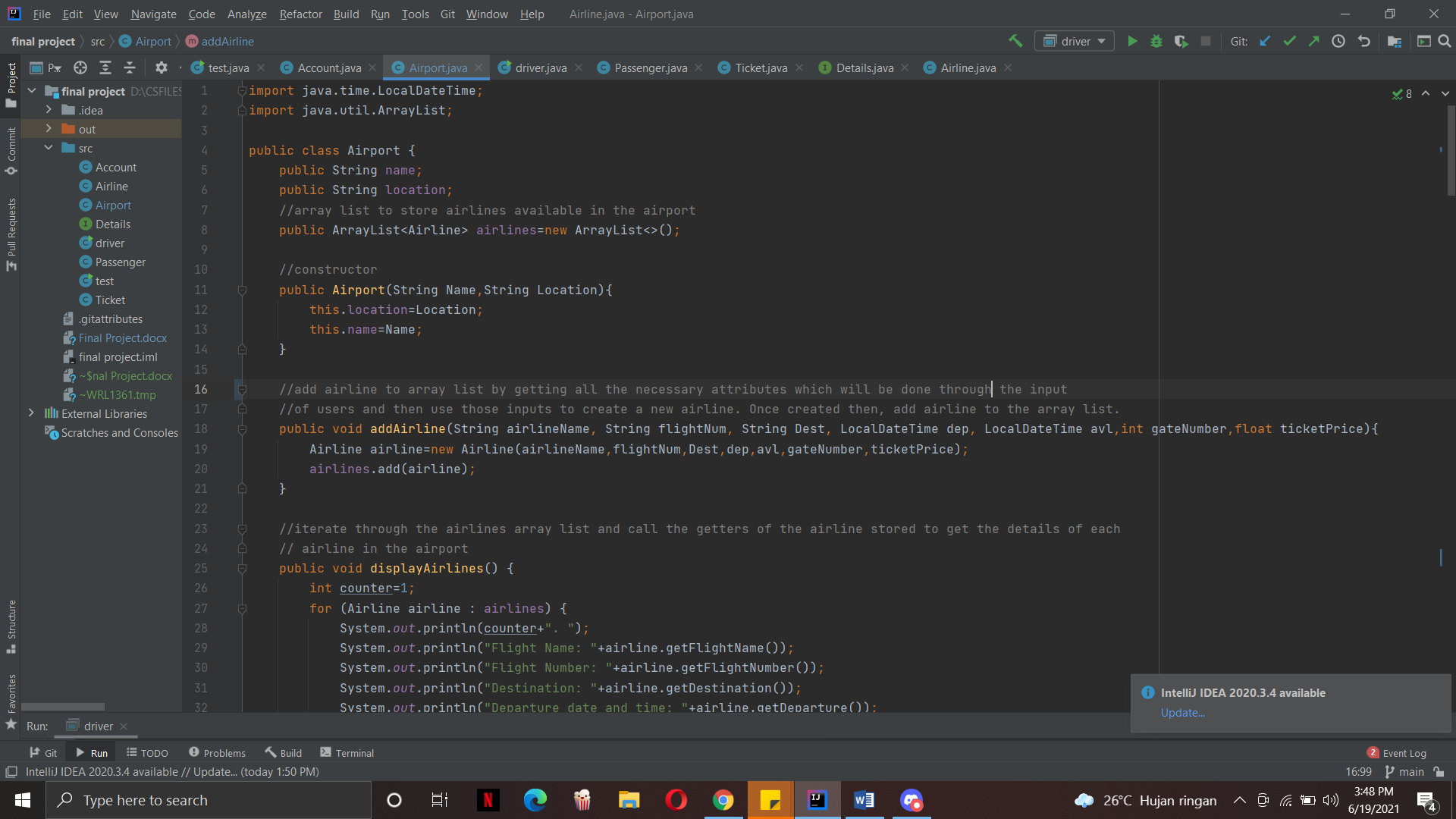


The airline class has a function that checks whether the airline is already fully booked. The function works by having a Boolean value of false which indicates that it is not fully booked. The default maximum capacity that I set was 100 so it will check the passenger array list whether or not the size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

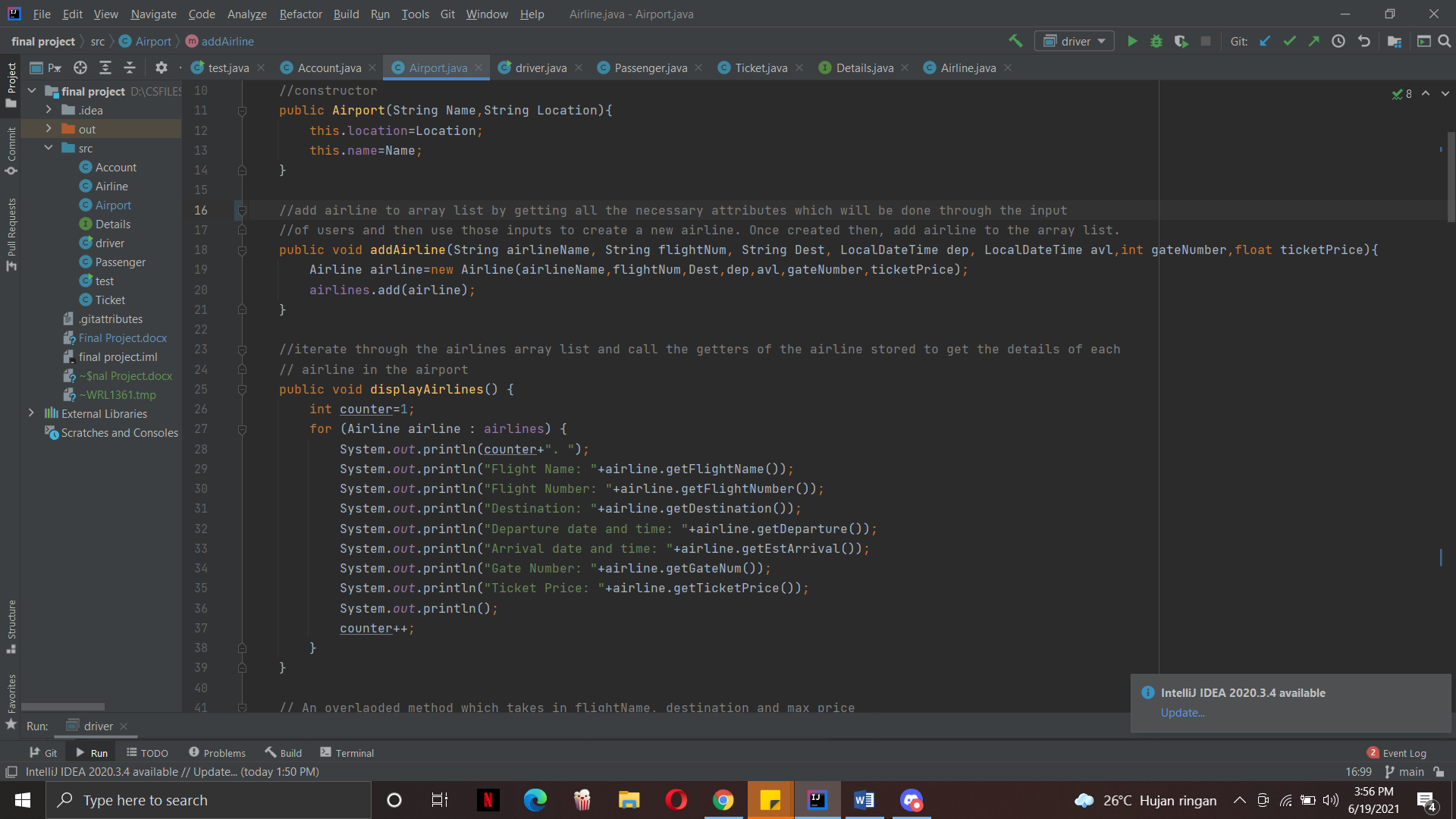
**TICKET CLASS**

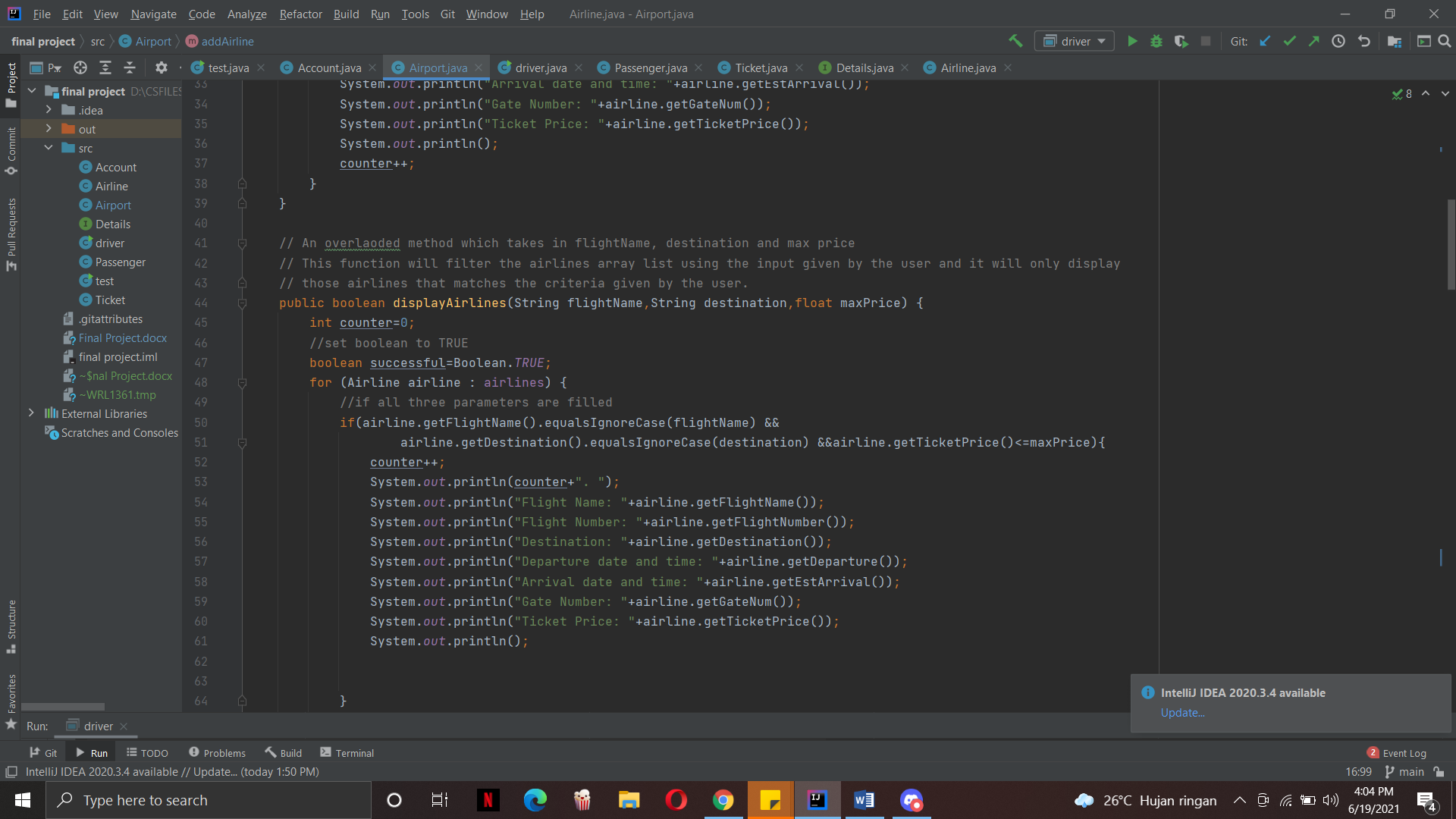
The ticket class will have the same attributes as the airline class but it will be unique to the ticket class. This is why there is a T at the start of every attribute to indicate that it’s from the ticket class. The ticket class also extends the passenger class as the passenger details will be needed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

**AIRPORT CLASS**

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.



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.

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().equalsIgnoreCase(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

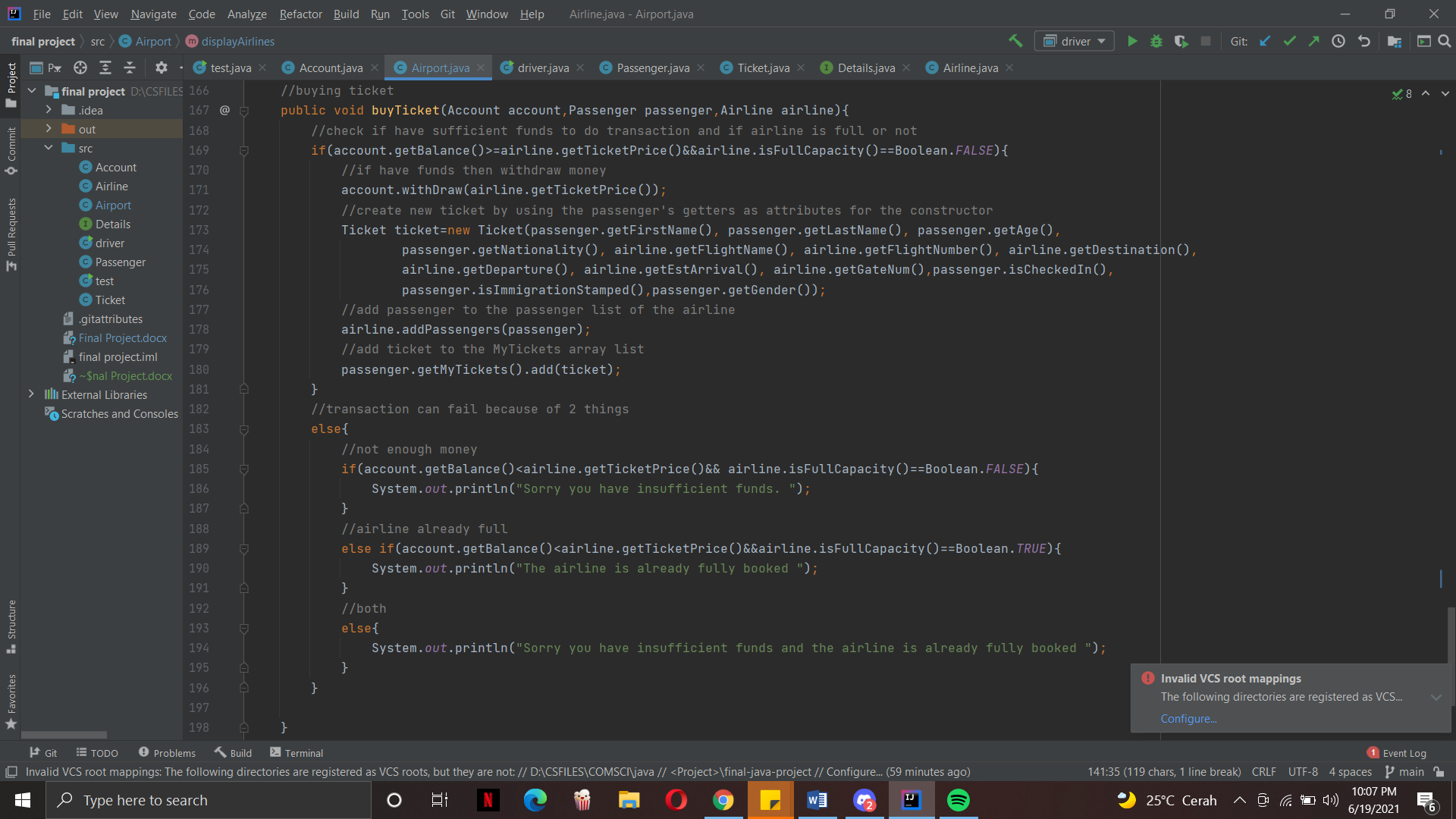
else if(flightName.equals("")&&airline.getDestination().equalsIgnoreCase(destination)  
 &&maxPrice==0)

if the user enters only destination

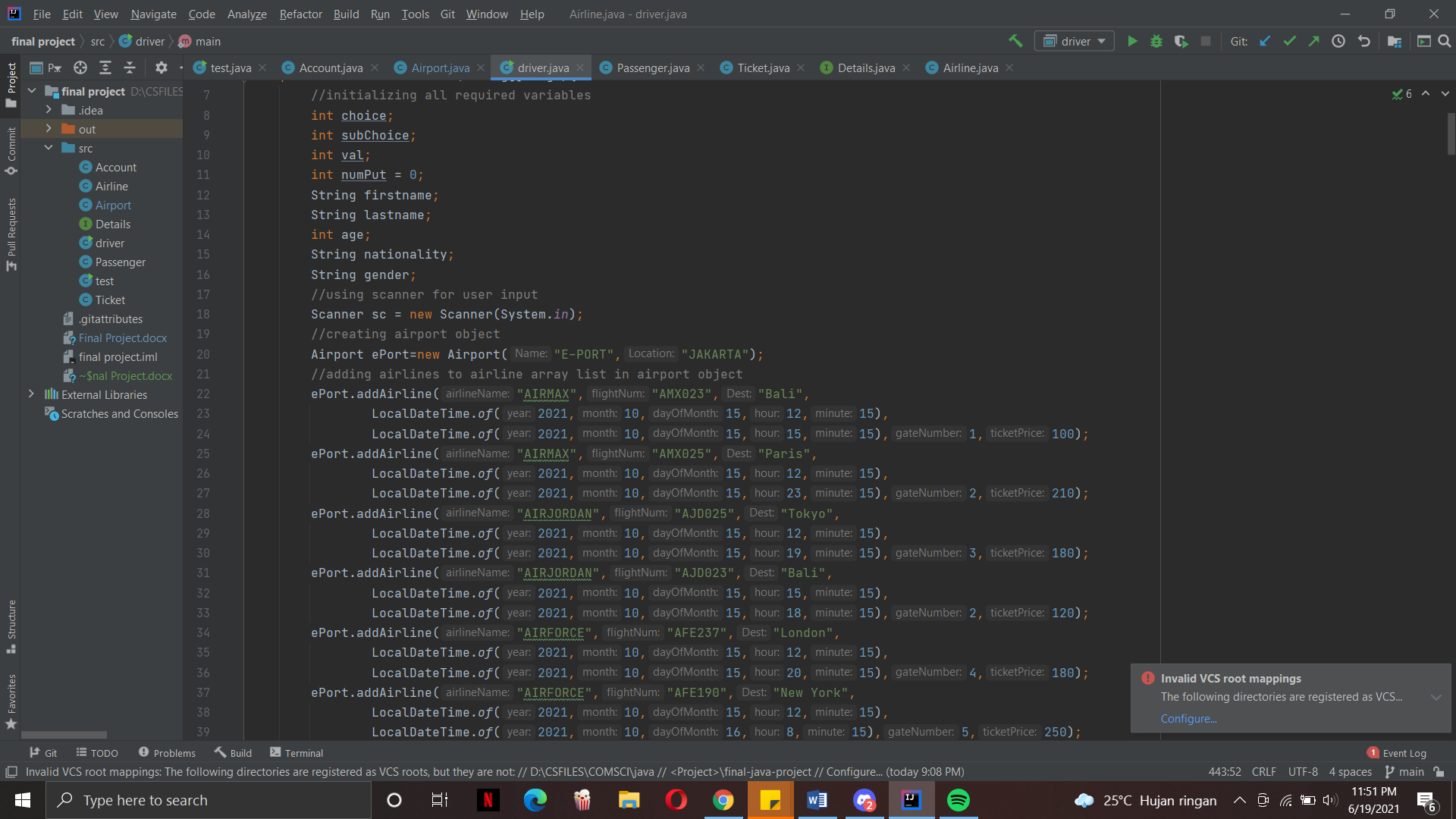
else if(destination.equals("")&&airline.getFlightName().equalsIgnoreCase(flightName)  
 &&maxPrice==0)

if the users only entered the flight name

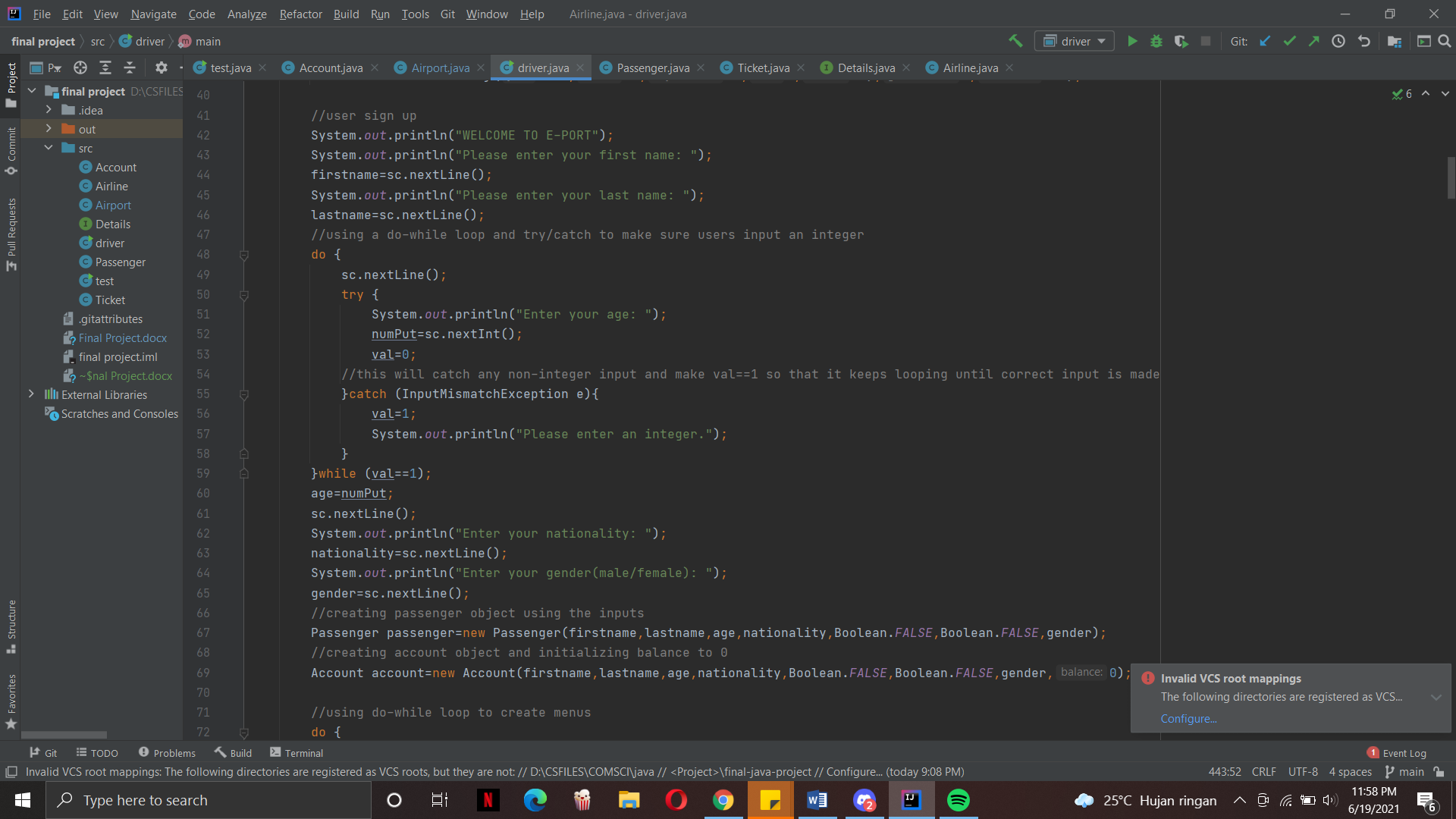
The function will keep on looping through and for each iteration it will check with these conditions whether or not any of the conditions are met. Every time an airline is found the counter will be added by one. Once the looping is finished if the counter is still 0, this means that no airlines matches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



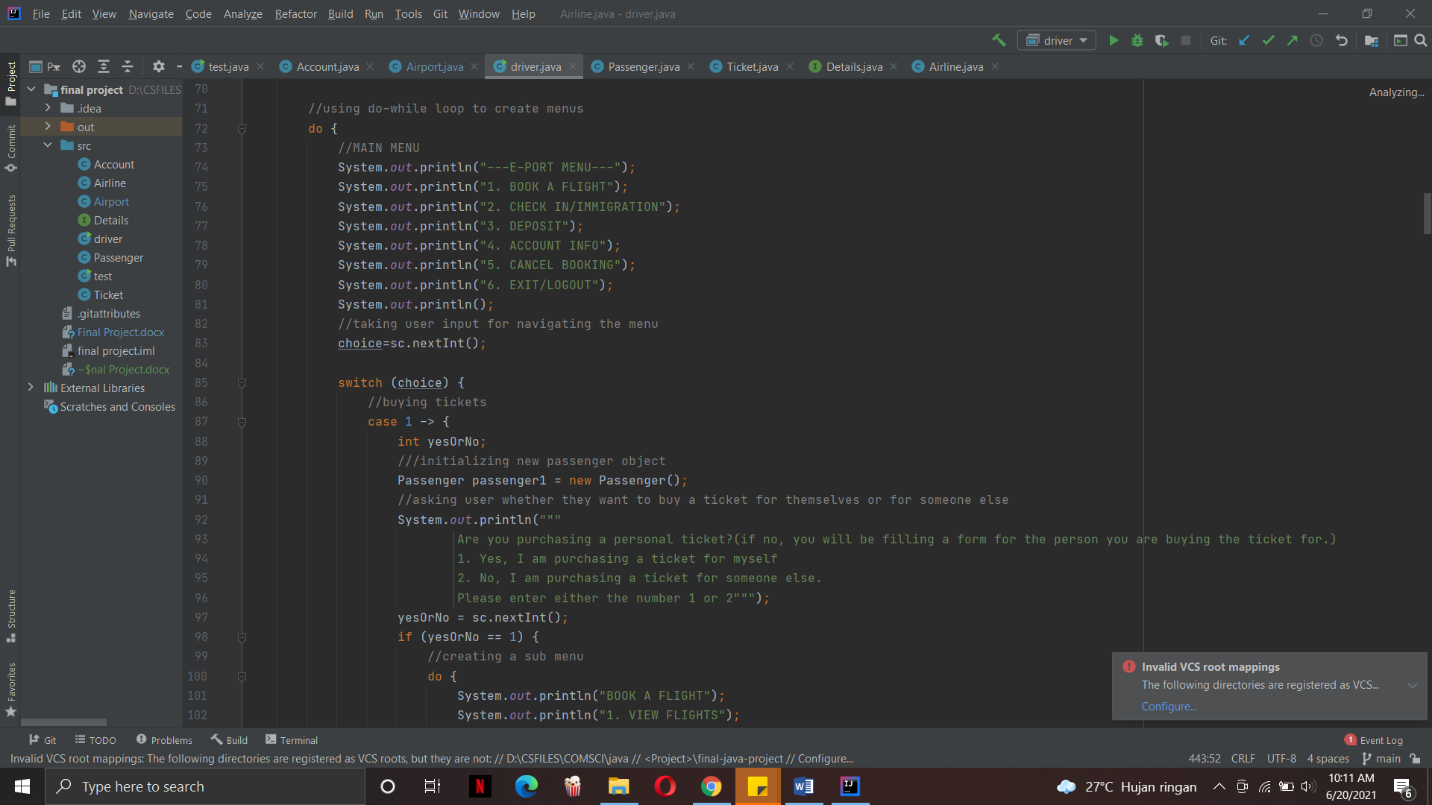
This function is called whenever the user will buy a plane ticket. It takes in an account object, passenger object and an airline object. First it checks whether the balance in the account is smaller or equal to the ticket price of the airline object. It also checks whether the airline object is at full 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**

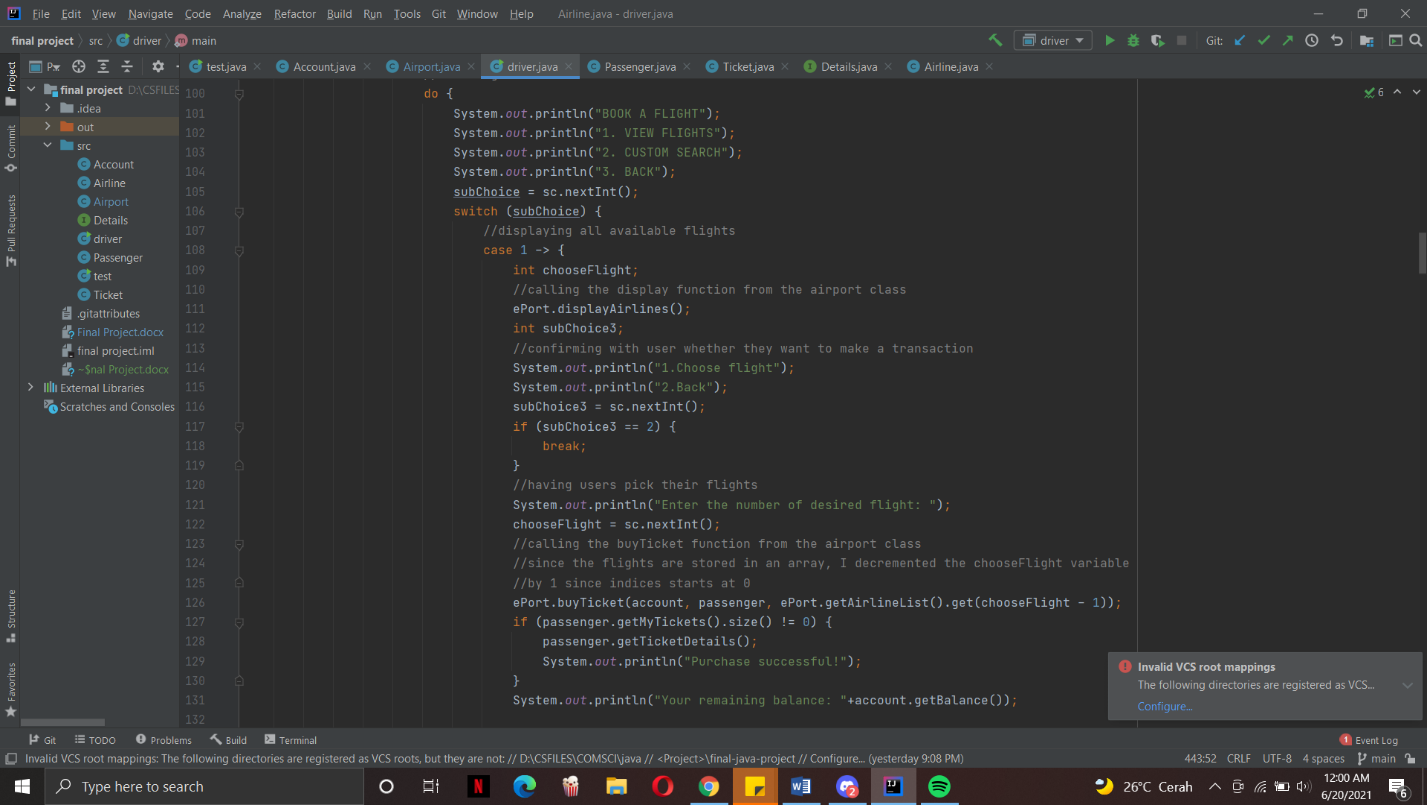
The driver class is where the code is all combined and run. First I created an airport object using the airport class and instantiated a couple of variables that are going to be used later on in the program. Then, I added airlines to the airport by calling the add ariline function which will allow me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

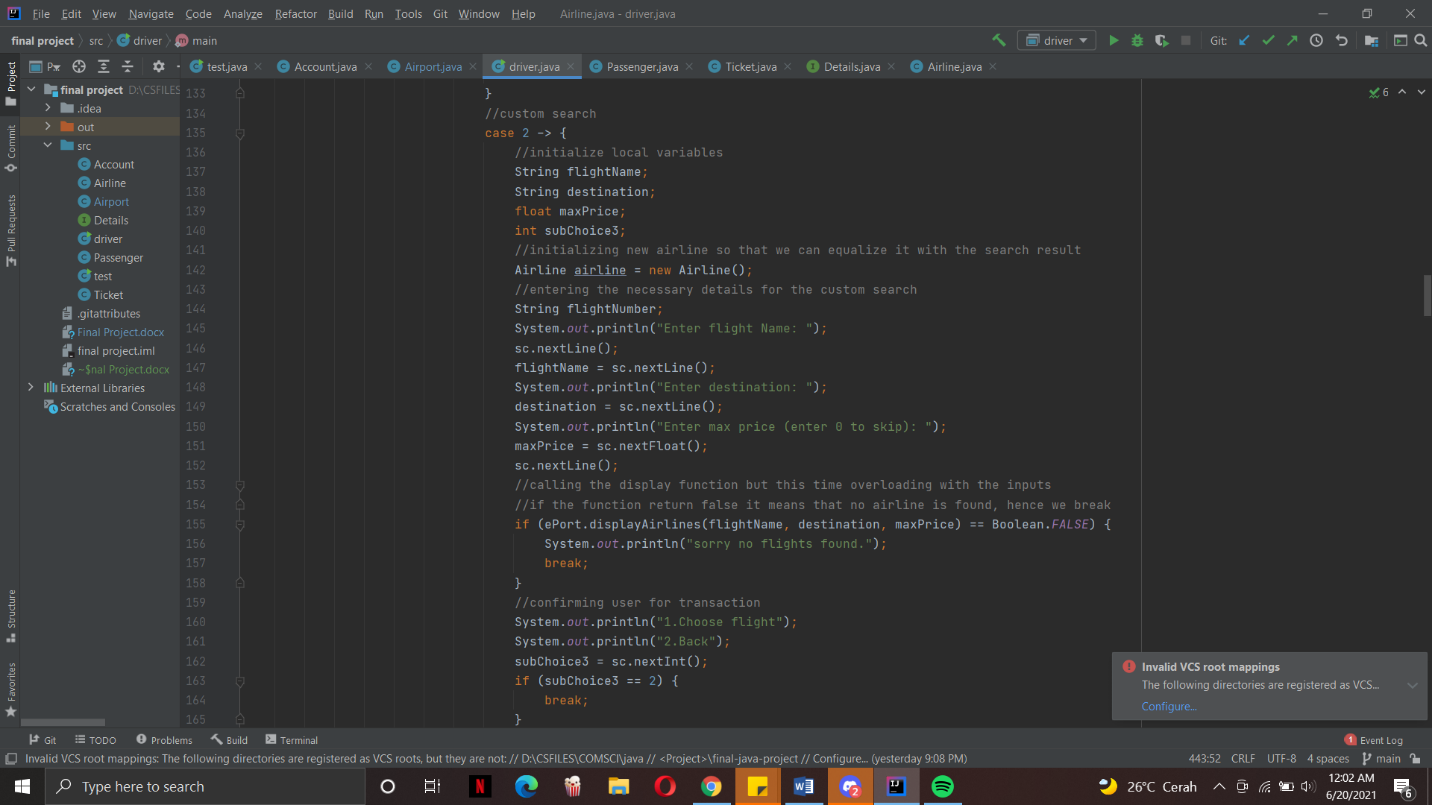


After seting up the airport, the user will be instructed to sign up by inputting a couple of details such as their first name, last name, age, gender and nationality. This code above shows the input for the age. Since the only sensible input for age is an integer, I used a try and catch block to catch 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.

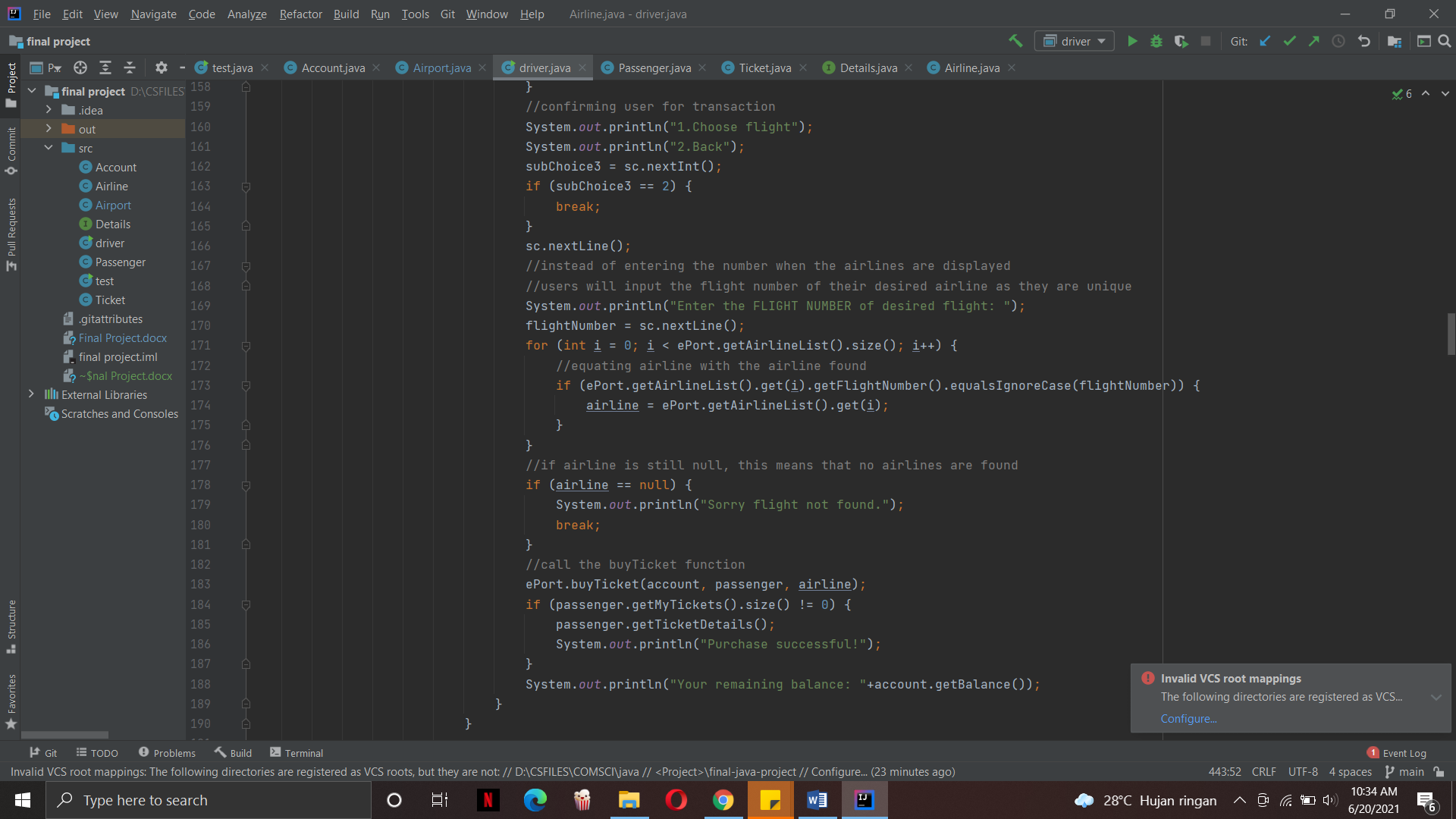


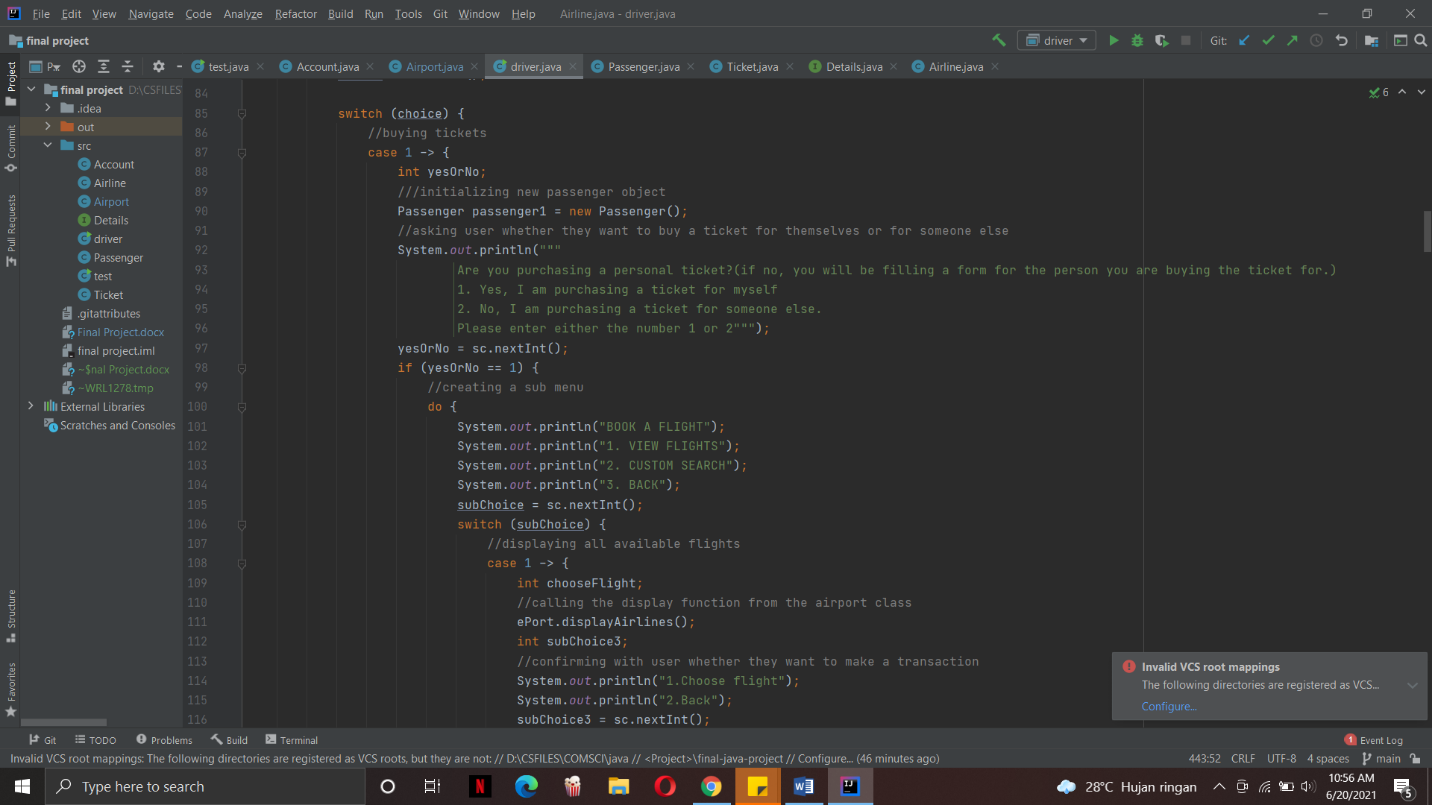
After the user sign up process, the user will be brought into the main menu where they can navigate through the program using numbers

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.

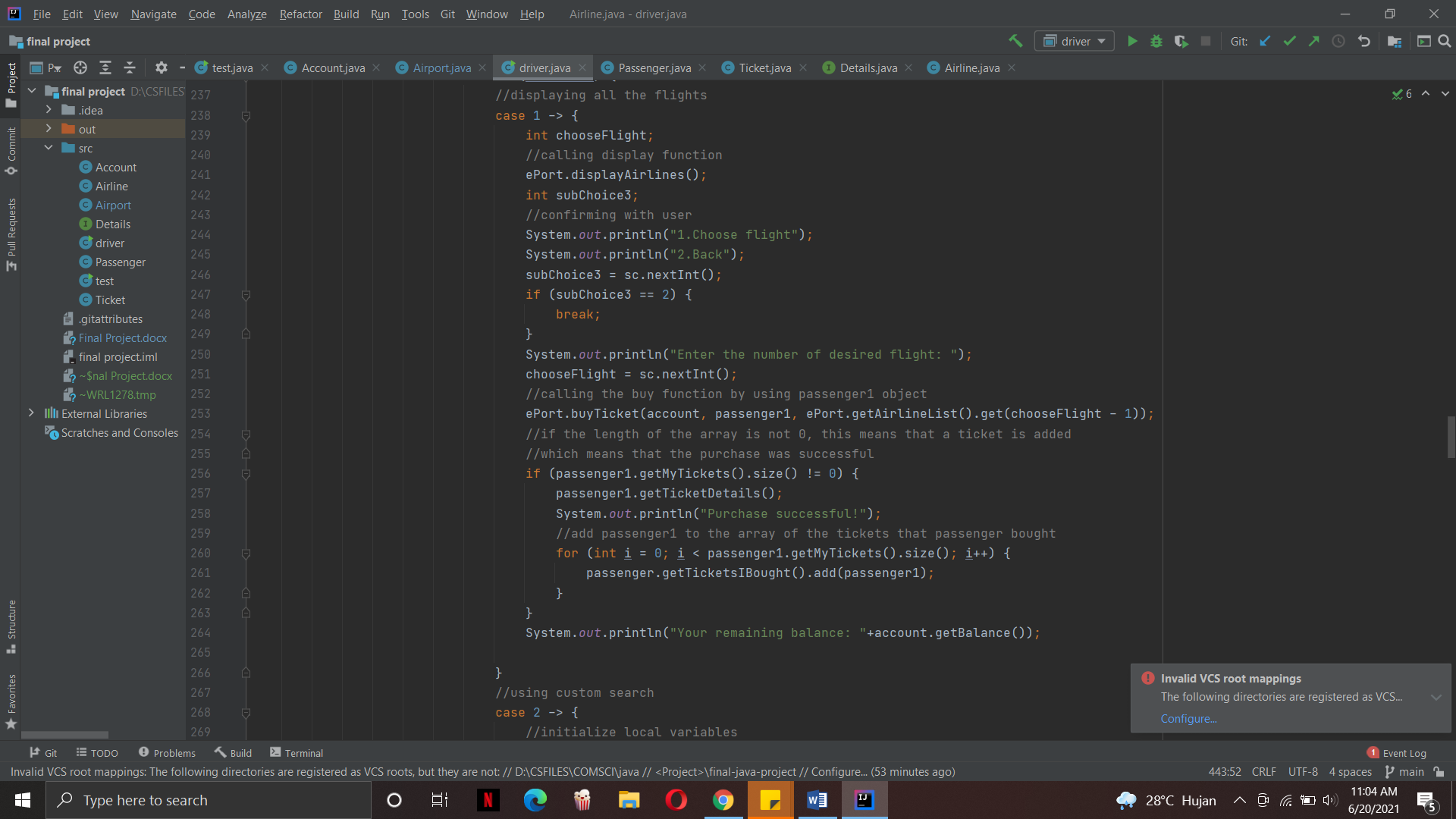


The second option is to do a custom search. For this I have intialized local variables fro the flight name, destination and max price. An airline object using the default constructor is also instantiated here because it will be used for the buying of tickets later on. The default constructor is used 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.

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.

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.

Passenger object created with null constructor or default constructor.

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.