***Theme:***

* Travel/Booking Application
  + You’re at the airport ordering your ticket or tickets. It will check the airplane status, and assign your ticket number

***Classes:***

* TicketBookingSystem (main)
* Plane
* SchedulePlanes
* Abstract Passenger
  + Adult extends Passenger
  + Senior extends Passenger (or Adult)
  + Child extends Passenger (or Adult)

**Class Building:**

The next few pages are explinations of each of the classes, one class per page to make them easier to see and build. They are written with the ‘what this does’ in mind. This is then followed by a ‘Things to Consider’ section which is just where some general notes on where parts of the project came from, for example: what a plane ticket receipt looks like.

**TicketBookingSystem: <Everyone>**

The main idea behind this class is to run the whole system. It will be responsible for reading in passenger information, booking them onto a flight, and printing out the passenger’s ticket. It will also let you know how much money you owe for your plane ticket or tickets (if you book for your whole family).

Additional options, for airport employees only, is to check the information about a flight if you give the day and month. It will also be responsible for calling the “update schedule” method to ensure that there are planes for both the current month and the following month ready to be booked.

* Variables: (All variables are private)
  + Final String FILEPATH\_SCHEDULE
  + Final String FILEPATH\_NUMBERS
  + Static Scanner in
* Methods: (All Methods are public and static unless otherwise specified)
  + Main
    - First it will run **StartUp()**.
    - It should print out a welcome message and give the options to either buy tickets, list flight details, or shut down the program. At the start of every loop it should call the **Schedule.updateSchedule** method
      * A switch statement can be used with **getInt** 0: Shut down, 1: Get Flight Details, or 2: Buy Tickets for example – if not one of these, just start the loop again.
    - It will then run ShutDown
  + Shutdown
    - This method saves the program state and shuts down.
      * Number of Adults
      * Number of Seniors
      * Number of Children
      * Number of Passengers
      * Plane Schedule
  + StartUp
    - This method attempts to get the previous state (from the Shutdown save) – if there are no previous saves, it will initialise everything to a fresh state.
  + viewPlaneInformation
    - use the getInt to get a month and day then lists total number of passengers and the breakdown of seniors, adults, and children for a particular **(uses the Schedule.getPlane and Plane.toString)**
  + getNextString
    - This will be used to get passenger details such as their first and last name
  + getNextInt
    - This will be used to get details such as departure day and month as well as get options such as “Type 0 if travelling alone”
  + buyTickets
    - it first asks the day and month for the ticket you want to buy. It then asks how many tickets there are to be bought (using getInt). It then checks there are enough free seats for the amount of tickets you wish to buy.
    - then for every ticket that needs to be bought you are prompted for what type of passenger (eg 1: Adult 2: Child 3: Senior) then you’re prompted for the given information using getPassengerInfo
    - Once all passengers are created it will then tell the user how much they owe as well as their tickets.
  + getPassengerInfo
    - This method is used to get the information about the passenger. It returns a String list in the order: First name, Last name, chosen seat, departure date
  + getTravelDate
    - This method gets the travel date from the passenger and returns the plane object associated with that travel day

**Plane: <Melissa>**

This monster class is to hold all the information about a particular flight. It includes a price breakdown for a given week in any month adjusted for the ‘mark-up’ depending on the day.

* Variables : (All variables are private)
  + Static char[] ROWS;
  + Static int[] SEATS\_IN\_ROW;
  + Static double[][][] PRICEs (Month, weekday, Adult / Child / SeniorPrice)
  + Static ArrayList<String> SEATS
  + Static totalSeats
  + Double[] prices
  + String departureDate
  + ArrayList<Adult> adults
  + AttayList<Children> children
  + ArrayList<Seniors> seniors
  + ArrayList<String> availableSeats
  + ArrayList<String> takenSeats
* Methods: (All Methods are public unless otherwise specified)
  + Constructor:
    - Default creates a plane object with a default price (no mark-up or discount)
    - Complex constructor takes in a month and a weekday and creates an empty array for adults, children, seniors and takenSeats. For availableSeats it copies the SEATS
  + ToString
    - Lists information about the plane
  + Getters and setters0.
    - GetChildPrice
      * Return the second price in GetPrice getPrice()[1]
    - GetAdultPrice
      * Return the first price in GetPrice getPrice()[0]
    - GetSeniorPrice
      * Return the third price in GetPrice getPrice()[2]
    - GetDepartureDate
      * Return the departure date in string format
  + AddChild, AddAdult, AddSenior:
    - Adds a child, adult, or senior to their respective lists.
  + ShowAvailableSeats
    - This method prints out a list of all the available seats
  + TakeSeat
    - This method takes a seat from availableSeats and puts it into takenSeats
  + createWeekdayVariations
    - This is a **static** method to create the price variations for the planes depending on their passenger age, weekday, and month
  + createSeatingChart
    - This is a **static** method that creates all the seats on the plane for SEATS

**Schedule: <Melissa>**

This class is to help take care of the scheduling of planes. It was originally part of the Plane class, however it was split up to make it easier to understand.

* Variables: (All variables are private)
  + Hashtable<String, ArrayList<Plane>> schedule
* Methods: (All Methods are public and static unless otherwise specified)
  + Private generateMonth
    - Generates a month’s worth of planes and puts them in order by date. Then it adds it to the hashtable under the provided key
  + updateSchedule
    - Updates a schedule by adding the current month if it isn’t there, add the next month if it isn’t there and removing the previous month (as you can’t board a plane in the past)
  + Private addMonthToSchedule
    - Adds a month to the schedule by calling generateMonth
  + Private removeMonthFromSchedule
    - Removes a month using the provided key
  + Getters and setters
    - getSchedule()
    - setSchedule()
    - getPlane(day, month)
      * Gets a plane for a given day and month choice
        + Day – any day within the month
        + Month – either this month (1) or next month (2)
    - getPlane(day, month, year)
      * Gets a plane for the given day month year

**Passenger: <Ciaran>**

Passenger is an **abstract class** holding information that all passengers need. This is primarily to show that we can use abstract classes, but allows us freedom to make changes depending on if someone is an adult, child, or senior.

* Variables: (All variables are private)
  + Static Int numberOfPassengers = 0
  + String firstName
  + String lastName
  + String seat
  + Int ticketNumber
  + String departureDate
* Methods: (All Methods are public unless otherwise specified)
  + Constructor – **do not create**
    - A constructor is not created for an abstract class.
  + Getters and Setters
    - getSeat, getFirstName, getLastName, getTicketNumber, getDepartureDate
      * Returns the information linked to the given variable
    - getNumberOfPassengers
      * Returns the number of passengers
      * This is a **static** method
    - setSeat, setFirstName, setLastName, setDepartureDate
      * sets the information linked to the given variable
    - setTicketNumber
      * sets the ticket number
      * ticket number should be hard to set, therefore this should be a **protected** method.
    - setNumberOfPassengers
      * sets the number of passengers
      * This is a **protected static** method.
    - addPassenger
      * this method literally is a void method and simply is numberOfPassengers++

**Adult: <Ciaran>**

The adult class is an extension of the Passenger class, it includes a count of all the adults that have been through the airport as well as an array list of Children that the adult is responsible for (in the even that they are not flying with children this will be an ArrayList of size zero). This class is to hold all information about an adult passenger.

* Variables: (All variables are private)
  + Static Int numberOfAdults = 0;
  + ArrayList<Child> accompanyingChildren
* Methods: (All Methods are public unless otherwise specified)
  + Constructor
    - Two constructors, one default, and one filling out most of the variables.
    - Example default constructor:
      * Adult(){  
         this(“Henry”, “Hoover”, “Baggage”, “None”);  
        }
    - Example complex constructor:
      * Adult(String first, String last, String chosenSeat, String departureDate){  
         Passenger.addPassenger();  
         super.setFirstName(first);  
         super.setLastName(last);  
         super.setSeat(chosenSeat);  
         super.setTicketNumber(Passenger.getNumberOfPassengers);  
         super.setDepartureDate(departureDate)  
         accompanyingChildren = new ArrayList<Child>();  
         numberOfAdults++;  
        }
  + Getters and setters:
    - getNumberOfAdults
      * This method gets the total number of adults
      * This is a **static** method
    - setNumberOfAdults:
      * This sets the total number of adults (used when loading the program)
      * This is a **static** method
    - getNumberOfAccompanyingChildren
      * This method uses accopanyingChildren.size() which returns an int value of the number of children the adult has with them
    - printAccompanyingChildren
      * This method uses a for loop to go through the ArrayList of children and uses the child.getFullName method. The children will be in a list, and can be separated using a “\n\t” to give them in a nice list form.
  + ToString
    - This to string method will be a print out of the information that is contained about the Adult in a nice ticket format. It should include their name, the date of their plane departure, ticket number etc.
    - Use getNumberOfAccompanyingChildren to display how many children the adult is traveling with. It will not include the actual names on the ticket. It can be added to the end such as (although this is just an example):
      * super.getFullName + “ is traveling with “ + this. getNumberOfAccompanyingChildren() + “children.”
  + AddChild
    - This method uses the .add method for the ArrayList accompanyingChildren and is used to add a child object to the adult.
  + RemoveChild
    - Literally just put //TODO: Future improvement. For now you’re stuck with all your kids!

**Child: <Michelle>**

This class is to hold information about a child that is flying. It also puts on their ticket if they’re flying alone or if they have a responsible adult. If there is a responsible adult, this responsible adult will be linked to them in case of a ‘missing child’ emergency.

* Variables: (All variables are private)
  + Static Int numberOfChildren
  + Boolean travelingAlone
  + String responsibleAdultName
* Methods: (All Methods are public unless otherwise specified)
  + Constructor
    - Three constructors, one default, one filling out most of the variables (but no adult name) and one filling out most of the variables with the adult.
    - Example default constructor:
      * Child(){  
         this(“Henry”, “Hoover”, “Baggage”, “None”, true, “None”);  
        }
    - Example complex constructor:
      * Child(String first, String last, String chosenSeat, String departureDate){  
         Passenger.addPassenger();  
         super.setFirstName(first);  
         super.setLastName(last);  
         super.setSeat(chosenSeat);  
         super.setTicketNumber(Passenger.getNumberOfPassengers);  
         super.setDepartureDate(departureDate)  
         travelingAlone = true;   
         numberOfChildren++;  
        }
  + Getters and setters:
    - getNumberOfChildren
      * This method gets the total number of children
      * This is a **static** method
    - setNumberOfChildren
      * This sets the total number of children (used when loading the program)
      * This is a **static** method
    - setResponsibleAdult
      * sets the responsible adult, and changes the traveling alone to false
  + ToString
    - This to string method will be a print out of the information that is contained about the Senior in a nice ticket format. It should include their name, the date of their plane departure, ticket number etc.

**Senior: <Michelle>**

To be completely honest there isn’t much difference between an adult and a senior, except for the fact that a senior gets a lower price. They still have the option of bringing children along on their flight (as some grandparents enjoy traveling with their grandchildren).

* Variables: (All variables are private)
  + Static Int numberOfSeniors = 0;
  + ArrayList<Child> accompanyingChildren
* Methods: (All Methods are public unless otherwise specified)
  + Constructor
    - Two constructors, one default, and one filling out most of the variables.
    - Example default constructor:
      * Senior(){  
         this(“Henry”, “Hoover”, “Baggage”, “None”);  
        }
    - Example complex constructor:
      * Senior(String first, String last, String chosenSeat, String departureDate){  
         Passenger.addPassenger();  
         super.setFirstName(first);  
         super.setLastName(last);  
         super.setSeat(chosenSeat);  
         super.setTicketNumber(super.getNumberOfPassengers);  
         super.setDepartureDate(departureDate)  
         accompanyingChildren = new ArrayList<Child>();  
         numberOfSeniors++;  
        }
  + Getters and setters:
    - getNumberOfSeniors
      * This method gets the total number of seniors
      * This is a **static** method
    - setNumberOfSeniors:
      * This sets the total number of seniors (used when loading the program)
      * This is a **static** method
    - getNumberOfAccompanyingChildren
      * This method uses accopanyingChildren.size() which returns an int value of the number of children the senior has with them
    - getNamesOfAccopanyingChildren
      * This method uses a for loop to go through the ArrayList of children and uses the child.getFullName method. The children will be in a list, and can be separated using a “\n\t” to give them in a nice list form.
  + ToString
    - This to string method will be a print out of the information that is contained about the Senior in a nice ticket format. It should include their name, the date of their plane departure, ticket number etc.
    - Use getNumberOfAccompanyingChildren to display how many children the senior is traveling with. It will not include the actual names on the ticket. It can be added to the end such as (although this is just an example):
      * super.getFullName + “ is traveling with “ + this. getNumberOfAccompanyingChildren() + “children.”
  + AddChild
    - This method uses the .add method for the ArrayList accompanyingChildren and is used to add a child object to the senior.
  + RemoveChild
    - Literally just put //TODO: Future improvement. For now you’re stuck with all your kids!

***Things to consider:***

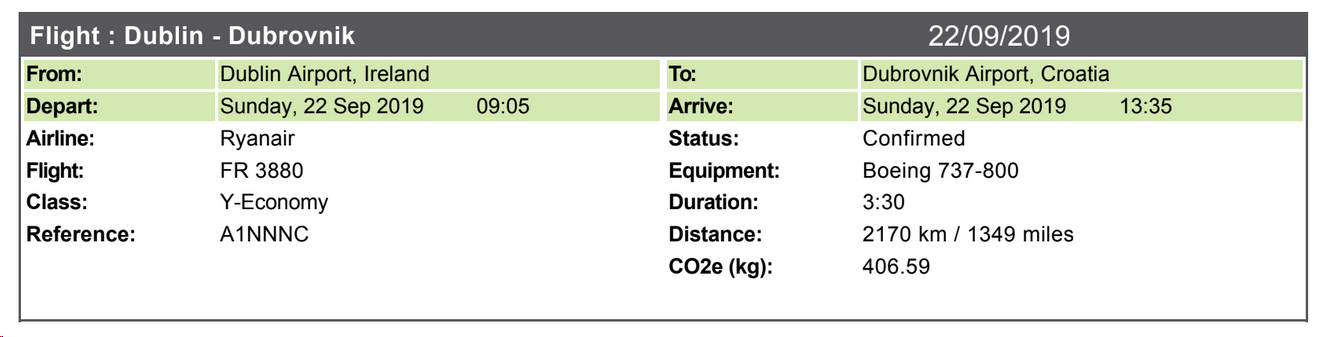


Figure 1: An example plane ticket confirmation for Melissa's trip to Dubrovnik back in 2019