F21AS

# Advanced Software Engineering

Coursework Stage 1

Airline Check-in System

by

| | |
|---|---|
| Maha Albalushi | H00245141 |
| Gita Permonaite | H00260666 |
| Nicholas Wiecek | H00156036 |
| Abdelraman Yaseen | H00281389 |

# CONTENTS

# INTRODUCTION

The purpose of this report is to develop and present simple check-in system at an airport. At this stage we only will develop the basic functionality of this system.

This report will go through development process of this program by presenting data structures, UML diagrams, decisions about program functionality and testing.

Please find the full code and copy of the project at the following link:

https://github.com/MildExile/F21AS_Gita_Abdelraman_Maha_Nicholas

## Group involvement:

Appendix A contains the development plan that was put together for the development of the system. Work was carried out according to that plan.

### Maha Albalushi – H00245141

Worked in Iteration 1 programming doing Flight class. Iteration 2 and 3 worked on the class diagram, requirements checklist and functional decisions.

### Gita Permonaite – H00260666

Worked in Iteration 1 programming doing PassengerFlightModel class and Iteration 2 and 3 the layout of the report as well as reanalysis of the code and redesign of the class diagram.

### Nicholas Wiecek – H00156036

Worked on Class diagram in Iteration 1 as well as refactoring Iteration 2 development into consistant form in line with planned work for Iteration. Changes to GUI, CheckInController and PassengerFlightModel were done as well as Junit tests for Passenger and Flight class.

### Abdelraman Yaseen – H00281389

Worked in Iteration 1 on Passenger class. In Iteration 2 and 3 worked on GUI class and CheckInController as well as the sequence diagram for the report.

All group members worked together to lay out the development plan as well as form the main class diagram that work was based off in Iteration 1. Decisions for functionality of the system were decided at the end of Iteration 1 through group effort.

# FUNCTIONAL REQUIREMENTS

The program meets all the specification requirements as listed below.

| Functional Requirements | Delivered: Yes/No/Partly |
|---|---|
| Text file with passenger details | Yes |
| Text file with flight details | Yes |
| Text files read at start of application | Yes |
| Display GUI for check-in | Yes |
| GUI captures passenger booking reference and last name | Yes |
| GUI captures passenger bag dimensions and weight for 1 luggage | Yes |
| GUI indicates excess baggage fee to be paid after calculation | Yes |
| Passenger booking reference and last name checked against flight booking list | Yes |
| GUI kiosk closes after all passengers checked in. | Partly |
| Report generated after GUI kiosk closes | Partly |
| Exceptions thrown if passenger was not found on the list | Partly |

## Initial Class Diagram

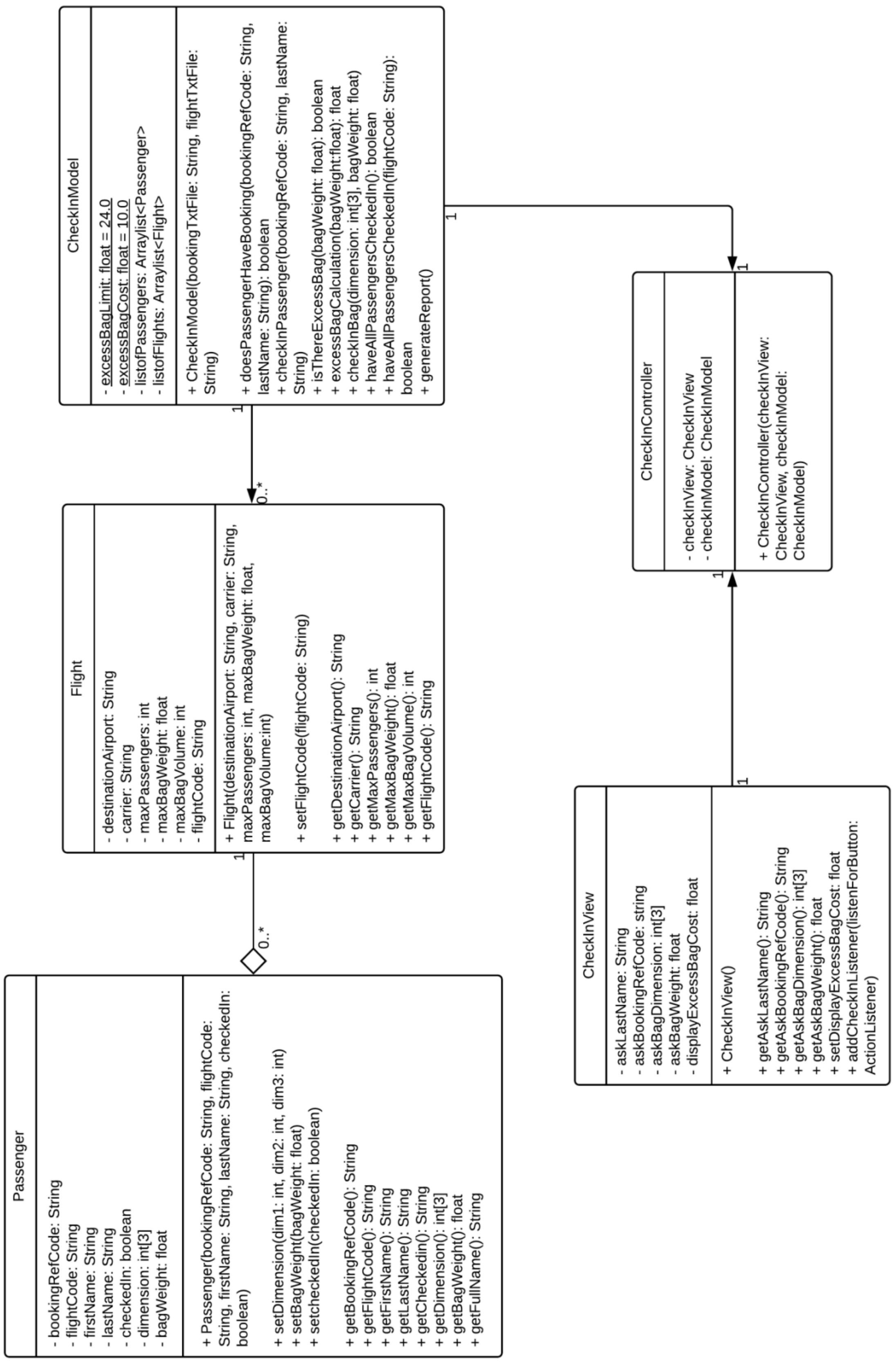

CheckInModel
- excessBagLimit: float = 24.0
- excessBagCost: float = 10.0
- listofPassengers: Arraylist<Passenger>
- listofFlights: Arraylist<Flight>
+ CheckInModel(bookingTxtFile: String, flightTxtFile: String)
+ doesPassengerHaveBooking(bookingRefCode: String, lastName: String): boolean
+ checkInPassenger(bookingRefCode: String, lastName: String)
+ isThereExcessBag(bagWeight: float): boolean
+ excessBagCalculation(bagWeight:float): float
+ checkInBag(dimension: int[3], bagWeight: float)
+ haveAllPassengersCheckedIn(): boolean
+ haveAllPassengersCheckedIn(flightCode: String): boolean
+ generateReport()
Flight
- destinationAirport: String
- carrier: String
- maxPassengers: int
- maxBagWeight: float
- maxBagVolume: int
- flightCode: String
+ Flight(destinationAirport: String, carrier: String, maxPassengers: int, maxBagWeight: float, maxBagVolume:int)
+ setFlightCode(flightCode: String)
+ getDestinationAirport(): String
+ getCarrier(): String
+ getMaxPassengers(): int
+ getMaxBagWeight(): float
+ getMaxBagVolume(): int
+ getFlightCode(): String
Passenger
- bookingRefCode: String
- flightCode: String
- firstName: String
- lastName: String
- checkedIn: boolean
- dimension: int[3]
- bagWeight: float
+ Passenger(bookingRefCode: String, flightCode: String, firstName: String, lastName: String, checkedIn: boolean)
+ setDimension(dim1: int, dim2: int, dim3: int)
+ setBagWeight(bagWeight: float)
+ setcheckedIn(checkedIn: boolean)
+ getBookingRefCode(): String
+ getFlightCode(): String
+ getFirstName(): String
+ getLastName(): String
+ getCheckedin(): String
+ getDimension(): int[3]
+ getBagWeight(): float
+ getFullName(): String
CheckInController
- checkInView: CheckInView
- checkInModel: CheckInModel
+ CheckInController(checkInView: CheckInView, checkInModel: CheckInModel)
CheckInView
- askLastName: String
- askBookingRefCode: string
- askBagDimension: int[3]
- askBagWeight: float
- displayExcessBagCost: float
+ CheckInView()
+ getAskLastName(): String
+ getAskBookingRefCode(): String
+ getAskBagDimension(): int[3]
+ getAskBagWeight(): float
+ setDisplayExcessBagCost: float
+ addCheckInListener(listenForButton: ActionListener)
1
0..*
1
0..*
1
1
1
1

FIGURE 1:INITIAL CLASS DIAGRAM OF THE AIRLINE CHECK IN SYSTEM

## Final Class Diagram

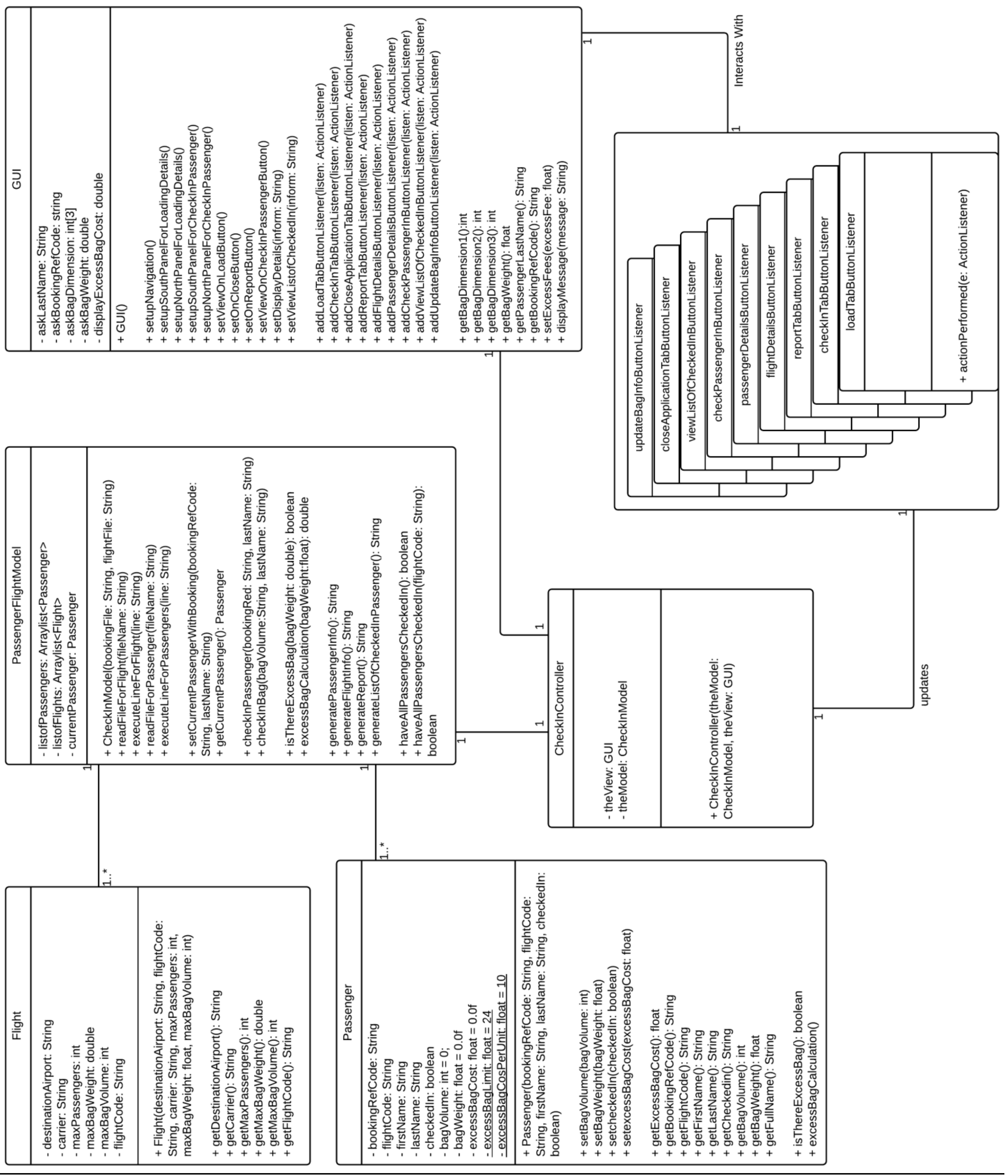

GUI
- askLastName: String
- askBookingRefCode: string
- askBagDimension: int[3]
- askBagWeight: double
- displayExcessBagCost: double
+ GUI()
+ setupNavigation()
+ setupSouthPanelForLoadingDetails()
+ setupNorthPanelForLoadingDetails()
+ setupSouthPanelForCheckInPassenger()
+ setupNorthPanelForCheckInPassenger()
+ setViewOnLoadButton()
+ setOnCloseButton()
+ setOnReportButton()
+ setViewOnCheckInPassengerButton()
+ setDisplayDetails(inform: String)
+ setViewListofCheckedIn(inform: String)
+ addLoadTabButtonListener(listen: ActionListener)
+ addCheckInTabButtonListener(listen: ActionListener)
+ addCloseApplicationTabButtonListener(listen: ActionListener)
+ addReportTabButtonListener(listen: ActionListener)
+ addFlightDetailsButtonListener(listen: ActionListener)
+ addPassengerDetailsButtonListener(listen: ActionListener)
+ addCheckPassengerInButtonListener(listen: ActionListener)
+ addViewListOfCheckedInButtonListener(listen: ActionListener)
+ addUpdateBagInfoButtonListener(listen: ActionListener)
+ getBagDimension1():int
+ getBagDimension2(): int
+ getBagDimension3(): int
+ getBagWeight(): float
+ getPassengerLastName(): String
+ getBookingRefCode(): String
+ setExcessFees(excessFee: float)
+ displayMessage(message: String)
1
Interacts With
1
updateBagInfoButtonListener
closeApplicationTabButtonListener
viewListOfCheckedInButtonListener
checkPassengerInButtonListener
passengerDetailsButtonListener
flightDetailsButtonListener
reportTabButtonListener
checkInTabButtonListener
loadTabButtonListener
+ actionPerformed(e: ActionListener)
1
updates
1
PassengerFlightModel
- listofPassengers: Arraylist<Passenger>
- listofFlights: Arraylist<Flight>
- currentPassenger: Passenger
+ CheckInModel(bookingFile: String, flightFile: String)
+ readFileForFlight(fileName: String)
+ executeLineForFlight(line: String)
+ readFileForPassenger(fileName: String)
+ executeLineForPassengers(line: String)
+ setCurrentPassengerWithBooking(bookingRefCode: String, lastName: String)
+ getCurrentPassenger(): Passenger
+ checkInPassenger(bookingRed: String, lastName: String)
+ checkInBag(bagVolume:String, lastName: String)
+ isThereExcessBag(bagWeight: double): boolean
+ excessBagCalculation(bagWeight:float): double
+ generatePassengerInfo(): String
+ generateFlightInfo(): String
+ generateReport(): String
+ generateListOfCheckedInPassenger(): String
+ haveAllPassengersCheckedIn(): boolean
+ haveAllPassengersCheckedIn(flightCode: String): boolean
1
1
1
1
1
CheckInController
- theView: GUI
- theModel: CheckInModel
+ CheckInController(theModel: CheckInModel, theView: GUI)
1
1..*
1..*
Flight
- destinationAirport: String
- carrier: String
- maxPassengers: int
- maxBagWeight: double
- maxBagVolume: int
- flightCode: String
+ Flight(destinationAirport: String, flightCode: String, carrier: String, maxPassengers: int, maxBagWeight: float, maxBagVolume: int)
+ getDestinationAirport(): String
+ getCarrier(): String
+ getMaxPassengers(): int
+ getMaxBagWeight(): double
+ getMaxBagVolume(): int
+ getFlightCode(): String
Passenger
- bookingRefCode: String
- flightCode: String
- firstName: String
- lastName: String
- checkedIn: boolean
- bagVolume: int = 0;
- bagWeight: float = 0.0f
- excessBagCost: float = 0.0f
- excessBagLimit: float = 24
- excessBagCosPerUnit: float = 10
+ Passenger(bookingRefCode: String, flightCode: String, firstName: String, lastName: String, checkedIn: boolean)
+ setBagVolume(bagVolume: int)
+ setBagWeight(bagWeight: float)
+ setcheckedIn(checkedIn: boolean)
+ setexcessBagCost(excessBagCost: float)
+ getExcessBagCost(): float
+ getBookingRefCode(): String
+ getFlightCode(): String
+ getFirstName(): String
+ getLastName(): String
+ getCheckedin(): String
+ getBagVolume(): int
+ getBagWeight(): float
+ getFullName(): String
+ isThereExcessBag(): boolean
+ excessBagCalculation()

FIGURE 2: FINAL CLASS DIAGRAM OF AIRLINE CHECK IN SYSTEM

Sequence Diagram

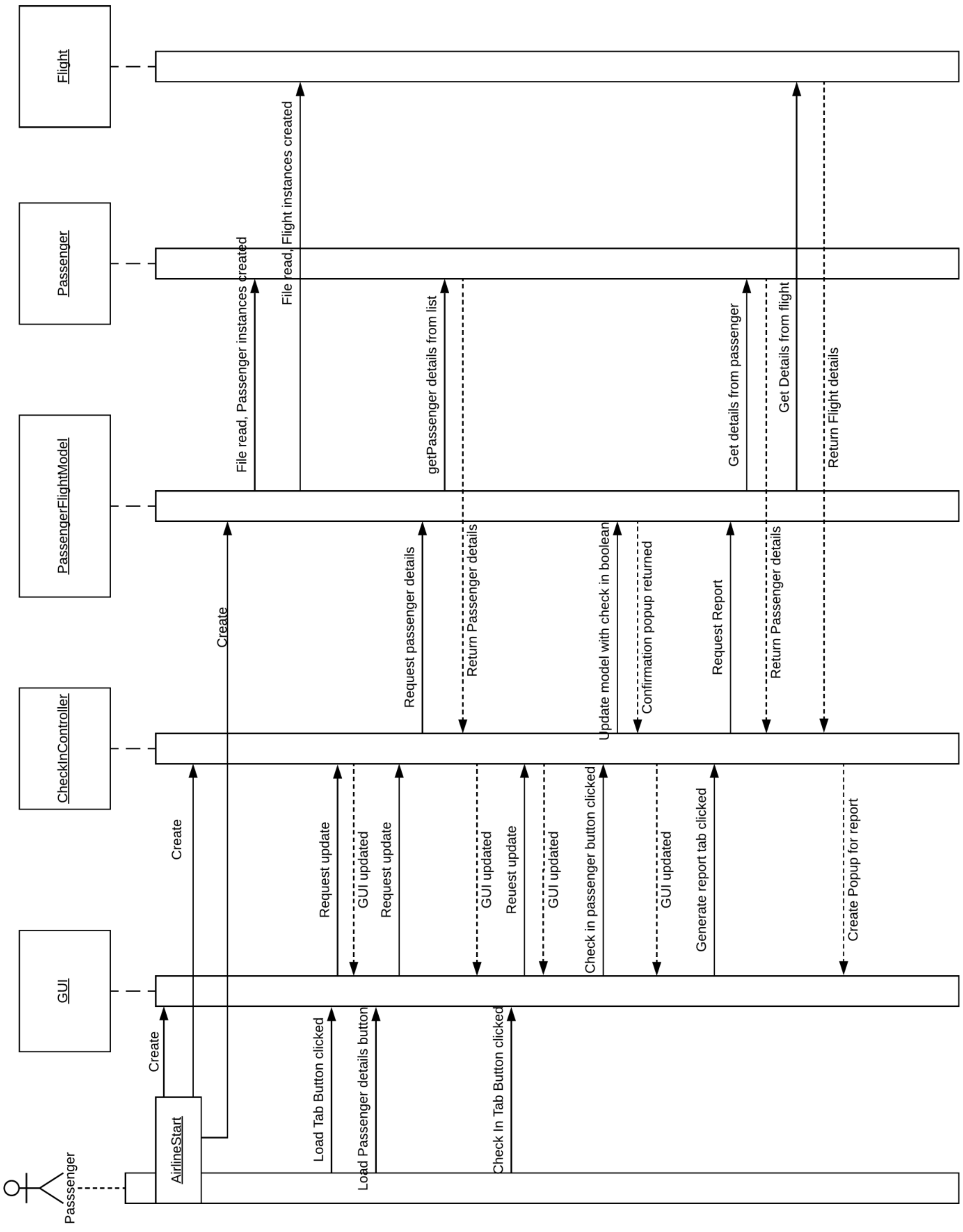


FIGURE 3: SEQUENCE DIAGAM SHOWING LIMITED INTERACTION OF PASSENGER WITH THE SYSTEM

## DESIGN CONSIDERATIONS & FUNCTIONALITY DECISIONS

For this application we decided to use these classes, Passenger, Flight, PassengerFlightModel, CheckInController, GUI as well as several Listener classes. The initial design of the system as can be seen in Figure 1 above had the relationship between the classes poorly thought out. This was later changed to associations between classes which made more sense. The goal was to remain within a Model View Controller (MVC) design pattern. The thought is that with the separation between the model and view, the system would be more robust requiring minimal work afterwards in order to make significant changes to the program.

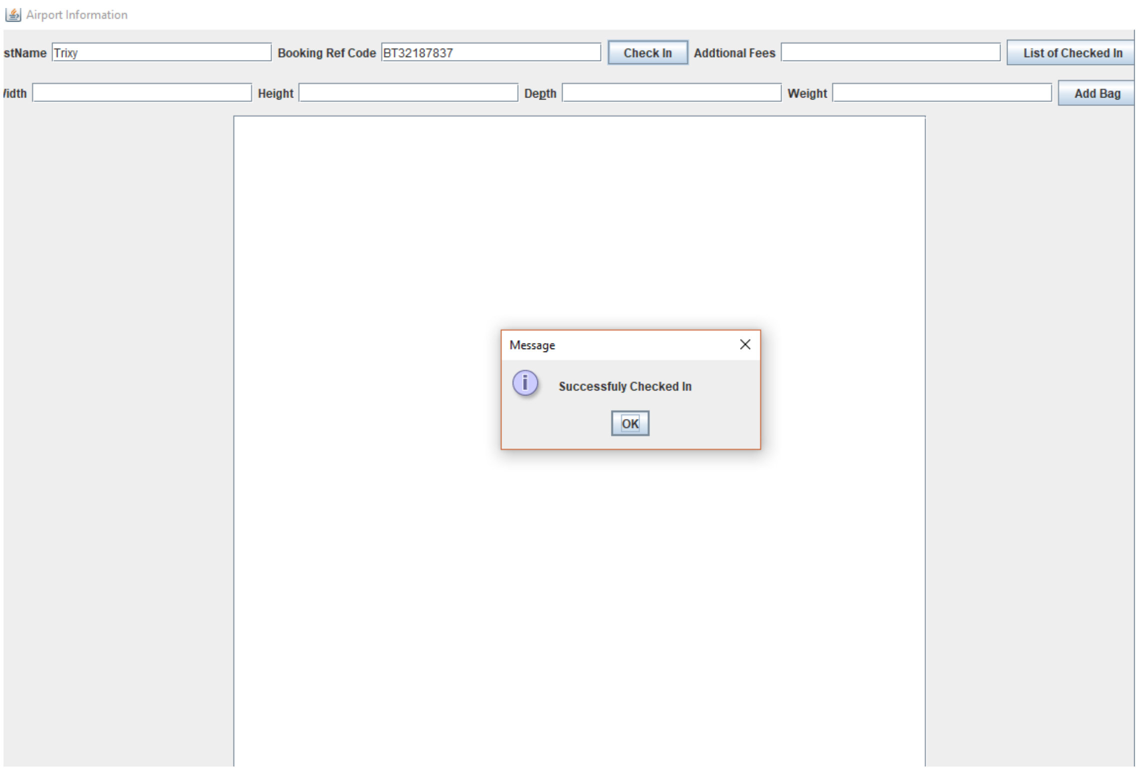
There could be some further separation of classes when looking at the class diagram specifically in the Passenger class which could have the baggage information moved to a separate class allowing a passenger to have many bags, we had agreed that the best way to design the system was to first let the system be as simple as possible since iterative development requires a great deal of time in the design of the plans prior to commencement of programming.

What could also be seen from the functional requirements is that the Flight class is not used greatly as all information that is needed to check in can be found in the booking list leaving the Flight class only truly useful in the generation of reports. The functional requirements and initial variables of the Flight class also mean that it would be impossible to ascertain exactly which flight a passenger as only the carrier and destination may be most helpful in creating a unique enough key to do searches by. For this reason it was decided that the flight code would also form part of the flight data file which would allow for flights to be much more uniquely matched to the passengers. As can be seen from Figure 2 the data structure used to keep the instances of the Passenger class is an arraylist, we had thought to change this to a hashmap as the only other suitable data structure for the given development project. Tree structures (treemap, treeset) were decided not to be adequate at all as they keep items in an order manner making retrieval much faster. With the functional requirement of a unique booking reference code for the passenger this would not make any sense as the booking reference code were likely not to correspond to any other information about the passenger allowing ordered structures like trees to be effective. Maps though allow unique key/value pairs which could work well in this project given the unordered nature of the information yet having a unique identifier for the passenger and creating a more unique identifier for the flight class by placing a flight code per flight. The arraylist used serves as a middle ground between the map and tree structures. Since the arraylist is traversed only once to get the current passenger and then all calculations are done using the current passengers values and the fact that the flight information does not need to be traversed during check in the arraylist data structure was used to store the list of passengers and flights.

The initial class diagram did not contain a variable for storing the overall excess baggage fees per flight nor the total volume and weight of bags, it was thought that as the lists would need to be traversed to get all passengers information the calculations could be performed at that time. With the changes in the final class diagram having a current passenger simulating one passenger being able to be checked in at any one time it made sense to store the bag and excess baggage costs separately. The calculations for excess baggage are done in the passenger class where all bag information is kept, a static variable is used – excessBagLimit is the variable which controls what the limitation of weight is for passengers, this was set at 24 to simulate a real world scenario and are stored as floats because baggage weight can severally affect a plane so precision is necessary. ExcessBagCostPerUnit is the variable used to measure the cost per kg of excess baggage, this was set at 10 to seem reasonable to still offer good testing parameters.

An example value was not provided for the variables to be used so we decided to use a 2 letter and 8 number combination (AA12345678) for the passenger's unique booking reference number, with such a large amount of numbers it is thought it would be unlikely for to individuals to encounter the same number. The decision for the flight code was to have it be 2 letters and 4 numbers (LX3214), there are always likely to be fewer flights than passengers so it is thought that such a combination would be sufficient for providing a unique enough number.

JUnit tests are fully done on the Passenger and Flight class's with exceptions being thrown at the read and write phase. JUnit testing on complex classes can be quite difficult and time consuming to create so time was instead taken to ensure that the overall testing of the system was done to ensure it did function as intended, primarily errors were found in the passenger and baggage information collection and while the program does not crash the handling of the responses could have been improved overall in the system. Appendix B contains the sample screenshots that were taken of the system while it was in use and testing. Detailed here are the screenshots of interest when testing.



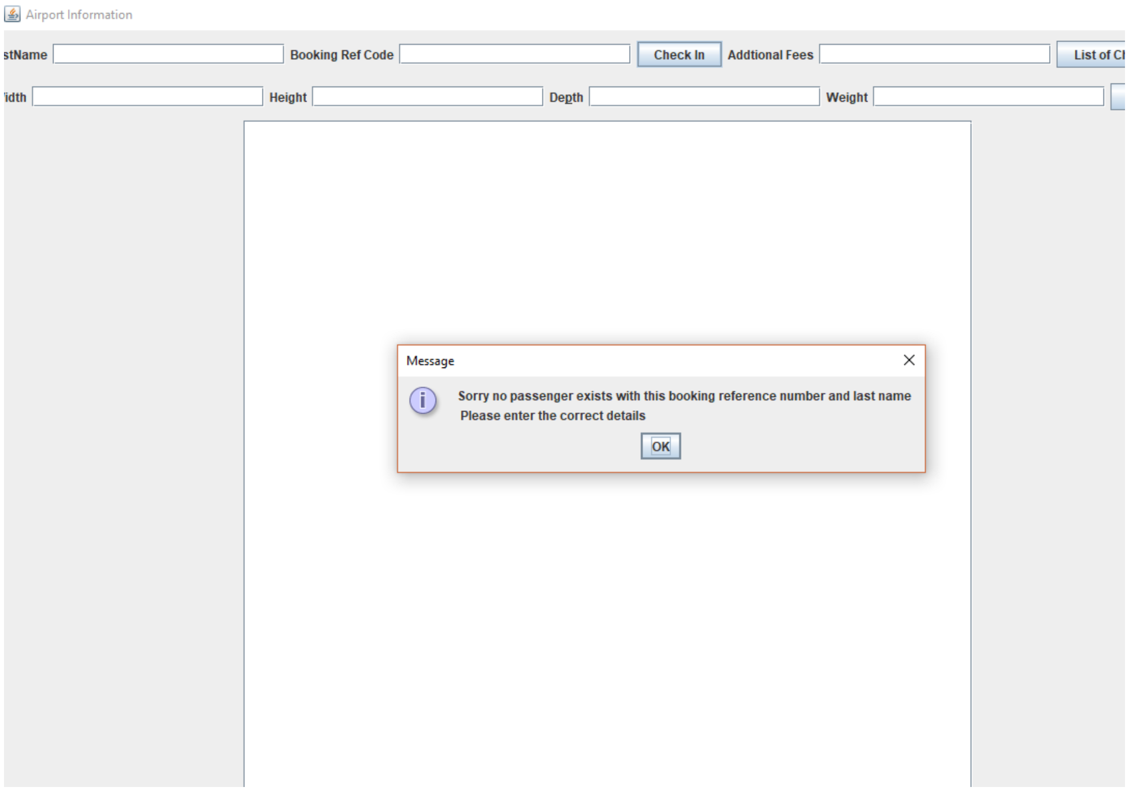
stName: Trixy

Booking Ref Code: BT32187837

Successfuly Checked In

FIGURE 4:BAG BEING SUCCESSFULLY CHECKED IN

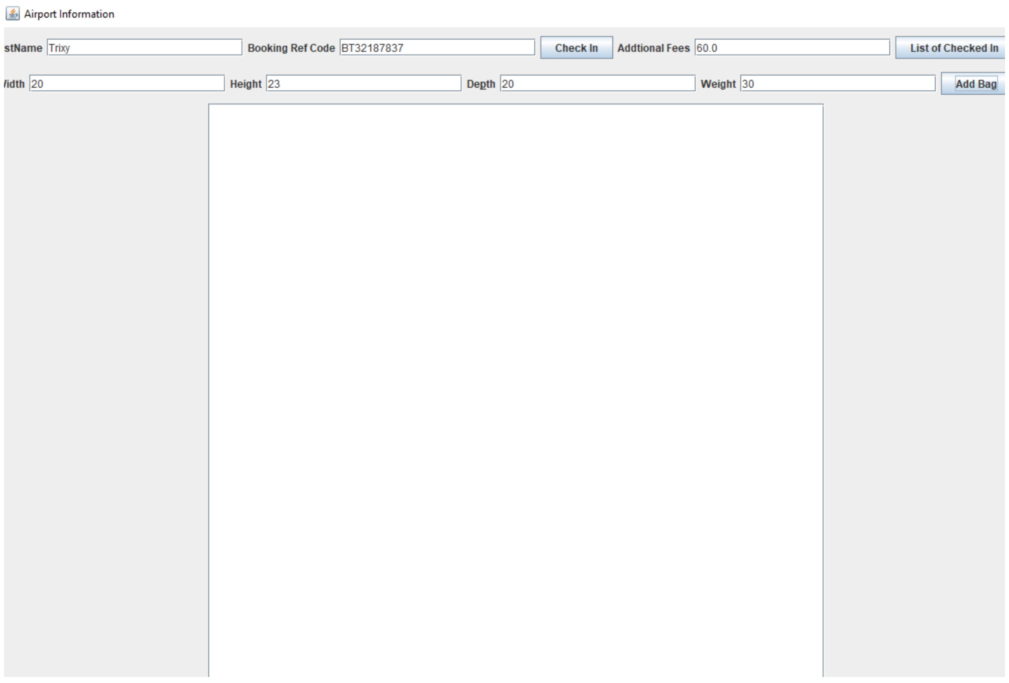
Figure 4 shows the confirmation message when a passenger successfully checks in while Figure 5 shows what the message response is when the passenger fields are empty or do not macth.



Sorry no passenger exists with this booking reference number and last name
Please enter the correct details

FIGURE 5:ERROR MESSAGE SHOWN WHEN NO PASSENGER INFORMATION IS GIVEN

Figure 6 below shows the excess baggage fee being shown after it is calculated. Figure 7 shows how the implementation of the error handling when no information is supplied for baggage still needs to be appropriately handled.



stName Trixy
Booking Ref Code BT32187837
Addtional Fees 60.0
idth 20
Height 23
Depth 20
Weight 30

FIGURE 6:EXCESS BAGGAGE ONCE CALCULATED

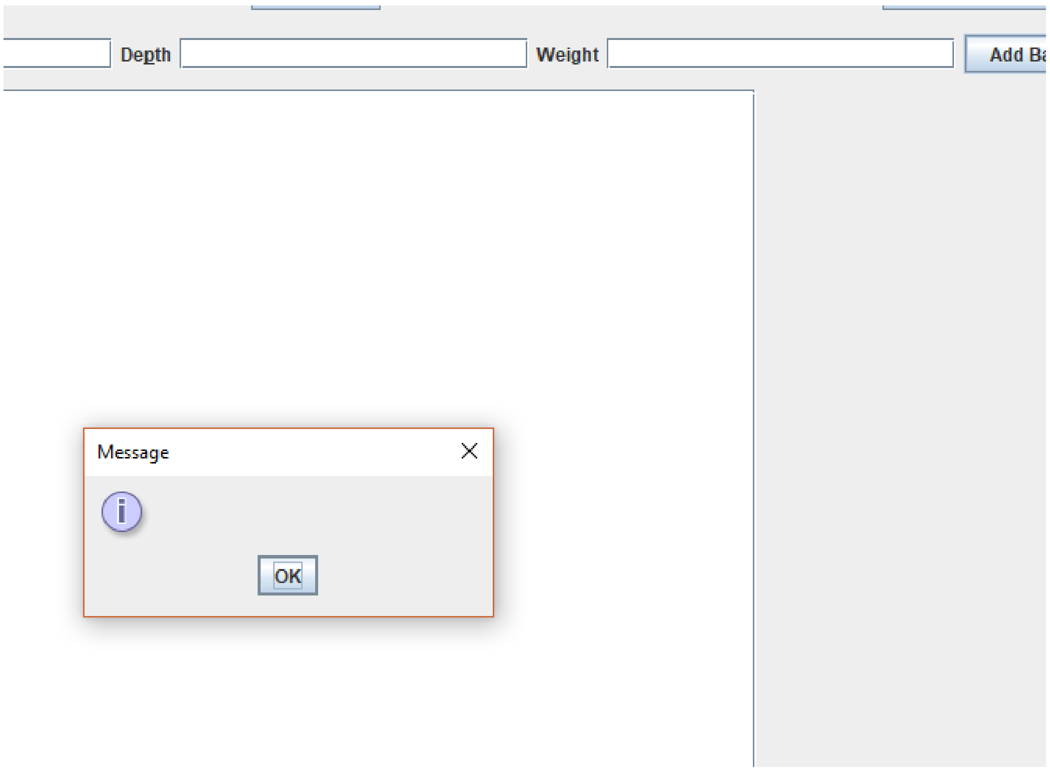


FIGURE 7:ERROR MESSAGE WHEN NO BAGGAGE INFORMATION IS GIVEN, STILL TO BE IMPLEMENTED

The testing that was done on the system was adequate to ensure that many of the functional requirements are met, the effective handling of all error messages and full JUnit testing still need to be fully implemented on the system.

## SUMMARY

JUnit testing and exception handling can be improved upon leading to and overall better performing system. The decisions made by the group lead to a system being developed that adequately handled the functional requirements and software engineering challenges of the project.

In iterative development where much time is spent designing systems a thorough unit testing can be seen to greatly improve better coding and greater consistency which in turn leads to faster development cycles with fewer errors. Redesign of the system was required at Iteration 2 and Iteration 3 of development showing how even careful planning is not fool proof. While Iteration 1 was done without much version control due to separation of the system from Iteration 2 it was necessary to have effective version control. We believe given extra time the project could be developed even further.

## Appendix A

<u>Development Plan</u>

| **Work Plan** | |
|---|---|
| Iteration 1 | 8 Feb |
| Analyse and/or redesign UML diagrams | |
| Group report write up – UML diagrams, data structures, specifications list, functional decisions | |
| Write JUnit Tests of classes – Passenger, Flight, CheckInModel | |
| Code classes – Passenger, Flight, CheckInModel | |
| | |
| Iteration 2 | 15 Feb |
| Analyse and/or redesign UML diagrams | |
| Group report write up – Update document with changes | |
| Write JUnit Tests of classes – GUI, CheckInController | |
| Code classes – GUI, CheckInController | |
| | |
| Iteration 3 | 22 Feb |
| Group report write up – testing, finalise | |
| Test full code | |
| Evaluation of program | |

## Appendix B

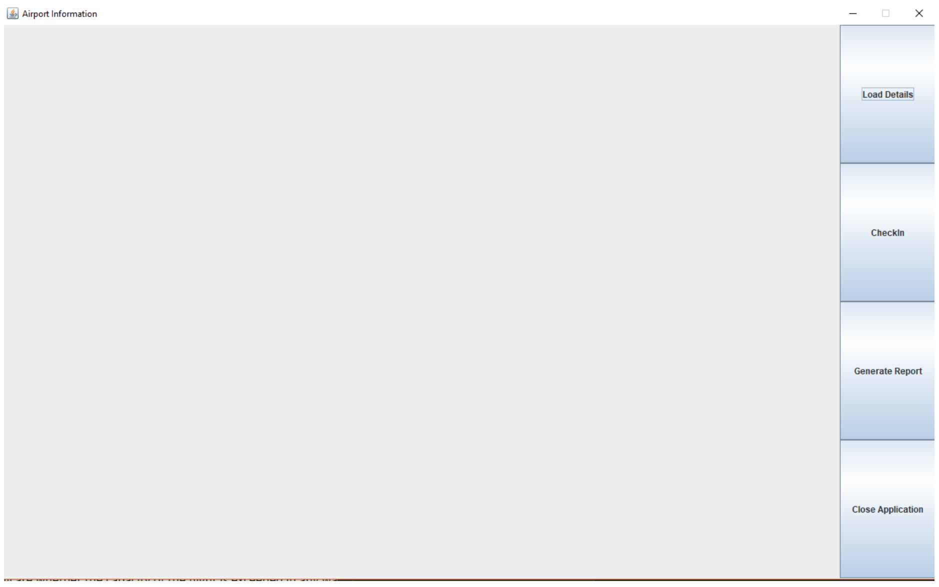
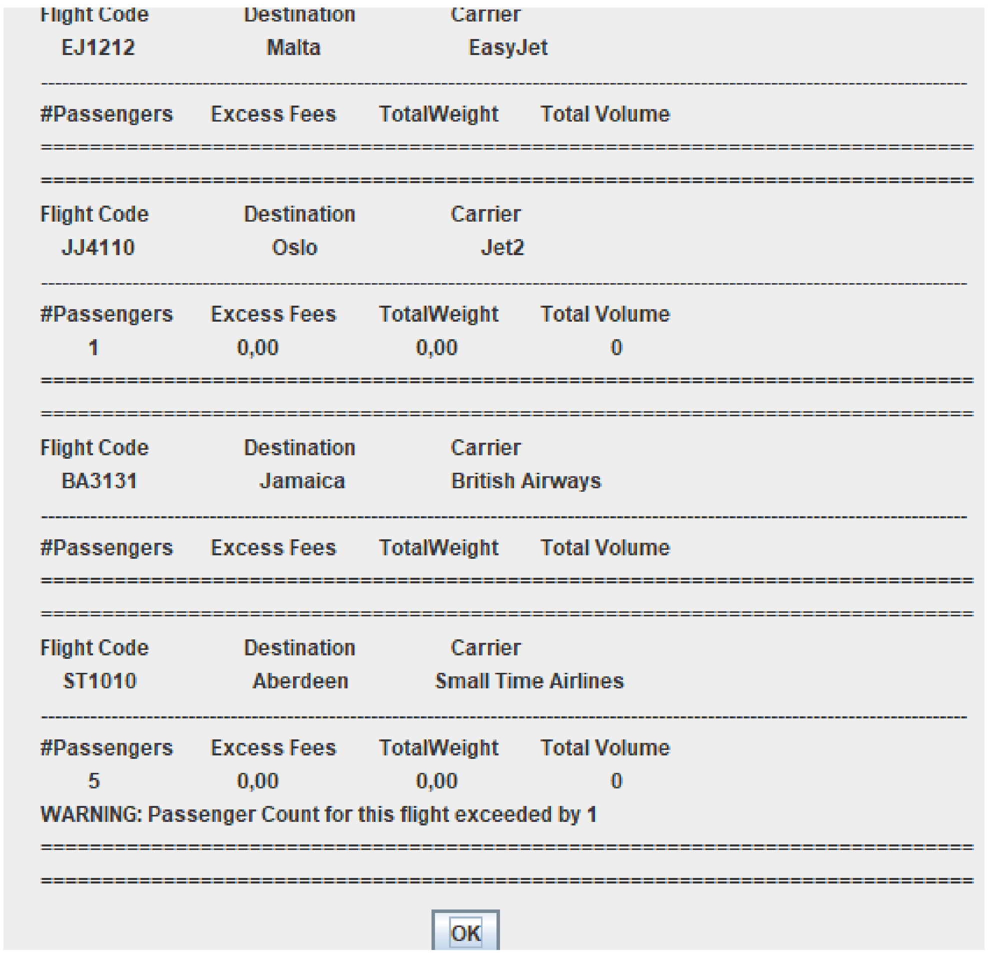
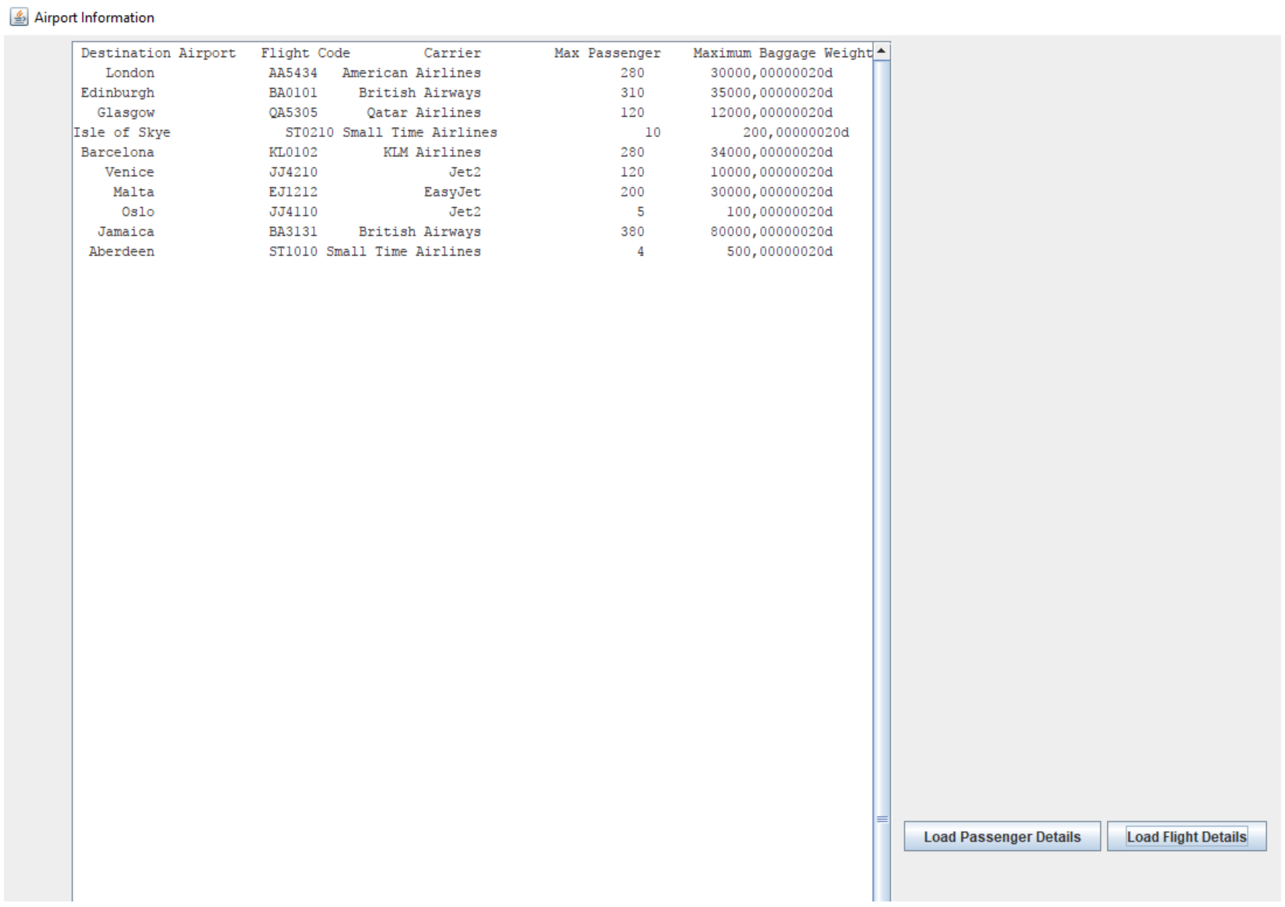


FIGURE 8:ENTRY POINT TO THE PROGRAM, TABS ON THE LEFT GUIDE THE USER



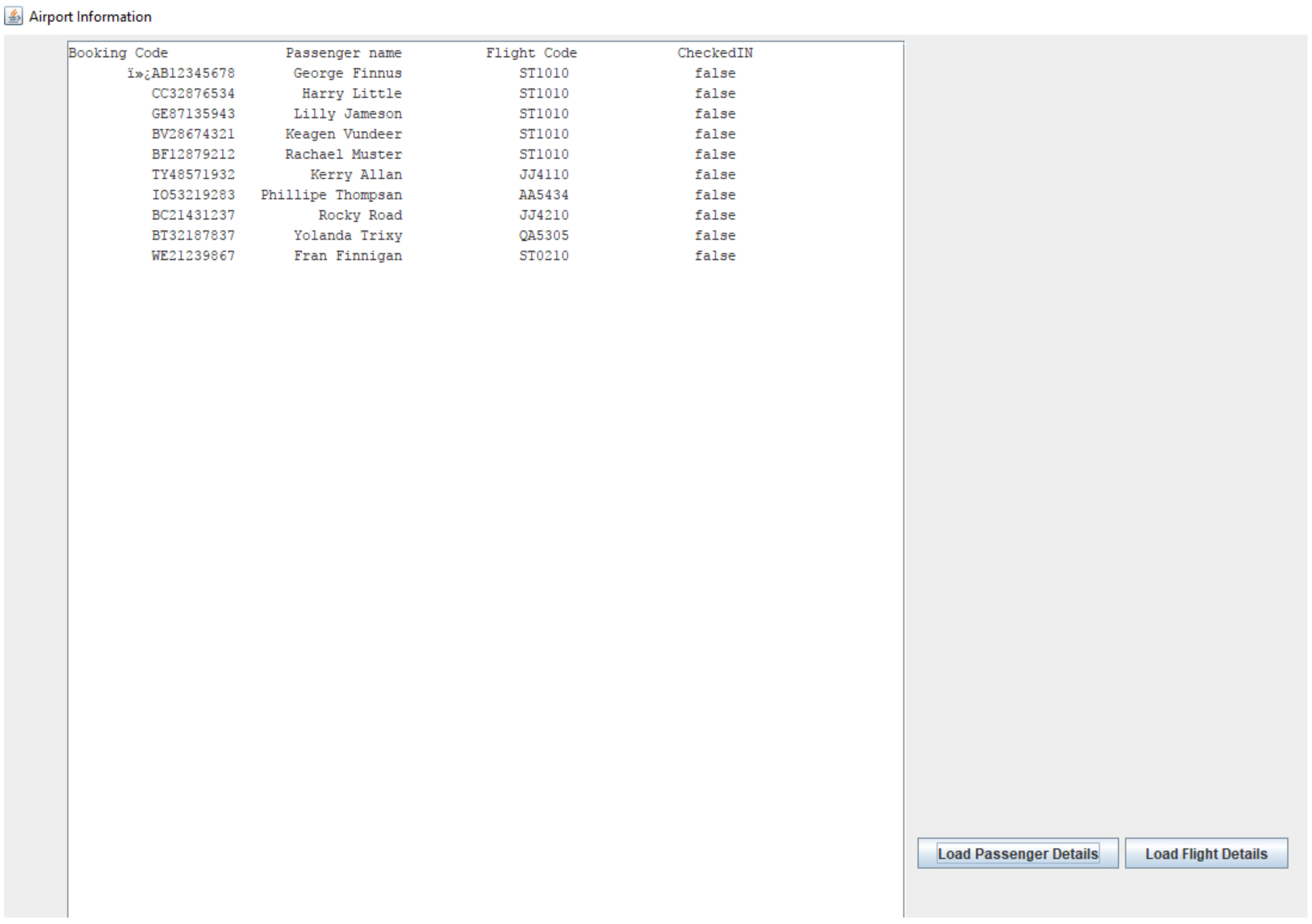
```
Flight Code        Destination        Carrier
  EJ1212             Malta              EasyJet
--------------------------------------------------------------------------------
#Passengers    Excess Fees    TotalWeight    Total Volume
================================================================================
================================================================================
Flight Code        Destination        Carrier
  JJ4110             Oslo                 Jet2
--------------------------------------------------------------------------------
#Passengers    Excess Fees    TotalWeight    Total Volume
     1             0,00           0,00             0
================================================================================
================================================================================
Flight Code        Destination        Carrier
  BA3131            Jamaica           British Airways
--------------------------------------------------------------------------------
#Passengers    Excess Fees    TotalWeight    Total Volume
================================================================================
================================================================================
Flight Code        Destination        Carrier
  ST1010            Aberdeen        Small Time Airlines
--------------------------------------------------------------------------------
#Passengers    Excess Fees    TotalWeight    Total Volume
     5             0,00           0,00             0
WARNING: Passenger Count for this flight exceeded by 1
================================================================================
================================================================================
```

FIGURE 9: THE REPORT GENERATED FOR FLIGHTS WITH A WARNING MESSAGE SHOWING PASSENGER COUNT BEING EXCEEDED



```
 Destination Airport   Flight Code         Carrier          Max Passenger    Maximum Baggage Weight
    London               AA5434   American Airlines                280        30000,00000020d
 Edinburgh               BA0101     British Airways                310        35000,00000020d
   Glasgow               QA5305      Qatar Airlines                120        12000,00000020d
Isle of Skye               ST0210 Small Time Airlines                 10          200,00000020d
 Barcelona               KL0102        KLM Airlines                280        34000,00000020d
    Venice               JJ4210                Jet2                120        10000,00000020d
     Malta               EJ1212             EasyJet                200        30000,00000020d
      Oslo               JJ4110                Jet2                  5          100,00000020d
   Jamaica               BA3131     British Airways                380        80000,00000020d
  Aberdeen               ST1010 Small Time Airlines                  4          500,00000020d
```

FIGURE 10:FLIGHT LIST SHOWING TRAVEL DESTINATION RECORDED IN A CSV FILE



```
Booking Code              Passenger name          Flight Code            CheckedIN
       ï»¿AB12345678         George Finnus             ST1010                false
          CC32876534          Harry Little             ST1010                false
          GE87135943         Lilly Jameson             ST1010                false
          BV28674321        Keagen Vundeer             ST1010                false
          BF12879212        Rachael Muster             ST1010                false
          TY48571932           Kerry Allan             JJ4110                false
          I053219283    Phillipe Thompsan             AA5434                false
          BC21431237            Rocky Road             JJ4210                false
          BT32187837         Yolanda Trixy             QA5305                false
          WE21239867        Fran Finnigan             ST0210                false
```

FIGURE 11: PASSENGER INFORMATION RECORDED IN A CSV FILE