

# Group Contribution

The members of the group consisted of Nadia Abulhawa, Haikah Ghoghari, Jamie Hill, Amy McFarland and Sam Muir.

In respect to the coding aspect of the group project, each member was assigned particular classes which was exhibited in the group planning schedule in the development report. Jamie was assigned the File IO and report generator classes, Nadia was assigned the flight classes, Haikah coded the passenger and baggage classes, Sam was assigned the Booking classes and Amy was designated the GUI classes. Throughout this part of the project, each group member assisted other members in their classes and testing which proved beneficial in completing the classes on schedule. Jamie offered assistance with the testing and flight classes and providing advice for other members classes. Nadia, Amy and Haikah worked together frequently helping each other complete their classes. Sam also provided assistance when it was required. Additionally, every group member read through each class that was committed to GitHub. This ensured that before each class was committed the group was happy that the classes included everything required. The integration of all the code was completed by \*\* which involved correcting any bugs that occurred in the program. After reading the report and ensuring the code worked effectively, the submission was done as a group to ensure everyone was satisfied with the work that had been accomplished.

As a group, we worked effectively together and often met to assist other members with their classes and discuss the objectives that had to be fulfilled for the next meeting. Each group member worked and contributed equally to the project.

In regard to the group report, each section was completed as a group with each member contributing equally. Amy wrote the summary and group contribution section of the report which was checked by the entire group. The class diagram restructuring was discussed as a group and individually completed by Nadia. The data structures section of the report was completed by Amy and checked by the group. The program functionality decisions were discussed a group during the coding part of the project and written by Amy. Finally, the testing section was completed as a group. Before submission, the report was read by each group member to ensure everyone was satisfied with the completed work.

# Repository Link

https://github.com/jamiefhill/hw-ase-stage-one-check-in-system

# Status Report

This program meets the specification fully.

# Class Diagram

Activity Diagram (for calculating the excess baggage fees)

# Data Structures

The data structures that were discussed in the development report have not changed and the decisions on which structures to use have remained the same.

The booking collection class used a HashMap structure, for the same reasons that were provided in the development report. For example, it would allow the finding of a booking quickly for the user when their booking code was entered into the GUI. This proved beneficial and effective as it did not disrupt the flow of the program for the user.

The flight collection class implemented a TreeSet structure as it provided the optimum functionality for the final output report as it iterated through the set of flights. Additionally, it offers a sorted list of flights for the report.

ArrayList for csv read

An Additional Structure we ended up using was a ArrayList of String Arrays we used for reading in csv files, we used an ArrayList as neither order nor accessing any particular element was important as the Structure was an inbetween structure for getting the data from the csv file input into the BookingCollection and FlightCollection.

Csv Structure

Our Bookings Csv file ended up being structured such that each Booking takes up a row with each element being separated such that the file could be read that the Strings retrieved from that row could be read as Flight Code , First name , Last name , Booking Code. The flight csv file was structured similarly with each flight taking up a row and read as Destination, Airline, Flightcode , MaximumPassengers , Max BaggageWeight , MaxBaggageVolume , Excesscharge

# Program Functionality

In regard to the functionality of the program, the booking reference codes followed a structure of two letters followed by three numbers with a – and then three numbers. For example, a booking coded used in the program was **BA123-121.** The excess baggage fees were calculated by firstly, subtracting the weight that was inputted by the user in the GUI from the allowed baggage weight per passenger. The allowed baggage weight per passenger value is included in the flight csv file and remains a constant number. This provided the excess baggage weight per passenger. The excess baggage weight was multiplied by the excess charge to calculate the excess fees. The excess charge value exists in in the flight csv file and fluctuates per flight.

# Testing

Junit testing

One our goals in our Junit testing was to have complete coverage of our code with our Junit tests. As a result our testing method included testing every method, every expected exception case and invalid inputs and boundary testing. For our booking this involved testing our capacity to Construct a booking object and then utilise all of the get methods involved in the class, additionally we expected it to throw our unique booking exception when it was constructed with an invalid booking code this exception is to be thrown in multiple cases involved in bookings including trying to retrieve a booking from booking collection with the wrong last name, trying to retrieve a booking with an invalid booking code and when trying to get a booking with an invalid flight code this was to allow the program to more accurately pinpoint the source of the error with a descriptive error message while allowing the GUI to handle these exceptions and not crash out when calling booking to distinguish from expected incorrect input errors and program errors that require a system shutdown.

The Booking collection class is capable of throwing more than just this error however and additionally throws the CheckInIOException because of utilising the csv processor, but this is checked in the csv processor test and as such is not analysed in the booking collection. The BookingCollectionTest is then left to test its initialisation, getting methods and that it correctly catches Booking exceptions.

Our Bag class was tested simply by justifying that a valid bag could be initialised and retrieve the information that it exceeds the capacity of the flight in volume or weight since it shouldn’t receive invalid inputs in the parts of the program where it is constructed. Our Passenger and Flight tests demonstrate the initialisation of a member of the class and the running of all their get methods while testing that the FlightException method is called appropriately in a similar manner to the BookingException throwing when a flight is not found so that it can be handled more elegantly .

Our ReportGeneratorTest Comprehensively runs all our collections initialising the Booking collection, the Flight Collection and loading our sample csv files then analysing all of the final results and comparing them to the expected result calculated independently from the program. In this manner the report generator test helps make sure that multiple collections integrate properly.