Design a console based software for an airline booking system

GHEORGHE MITREA, TOWER HAMLETS COLLEGE

Airline booking Application

Object Oriented Programming

***Table of Contents***

[1. Task 1 Key OOPS Concept 3](#_Toc462223329)

[2. Task 2 Design the code, plan the solution by producing a range of UML diagram and reflecting on my solution/ implemented algorithm 14](#_Toc462223330)

[2.1 UML Diagram 14](#_Toc462223331)

[2.2 Design the software code for an airline booking system 16](#_Toc462223332)

[***Customer class*** 16](#_Toc462223333)

[***Flight class*** 18](#_Toc462223334)

[2.3 Demonstrate how I overcame problems with the design critically reflecting on my solution / implemented algorithms and provide adequate reasons 24](#_Toc462223335)

[3. Task 3 Be able to implement object oriented programming solutions and Be able to test and document object oriented programming solutions 27](#_Toc462223336)

[3.1 Implementing my solution using industrial strength IDE and tools 27](#_Toc462223337)

[3.2 Provide a technical documentation for my software 29](#_Toc462223338)

[I will start the documentation by showing you how I coded the ***Customer class.*** 29](#_Toc462223339)

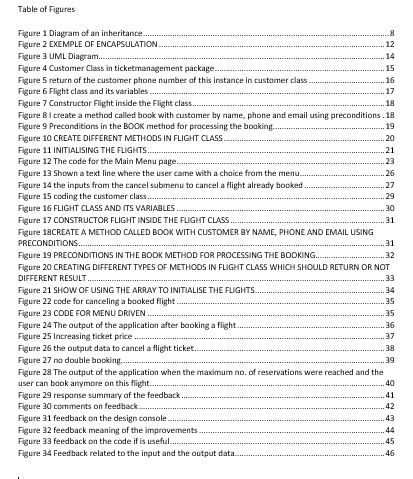
[***Flight class*** 30](#_Toc462223340)

[3.3 Testing phase of my application 36](#_Toc462223341)

[3.4 User feedback for the system and analyse and comment on the feedback 40](#_Toc462223342)

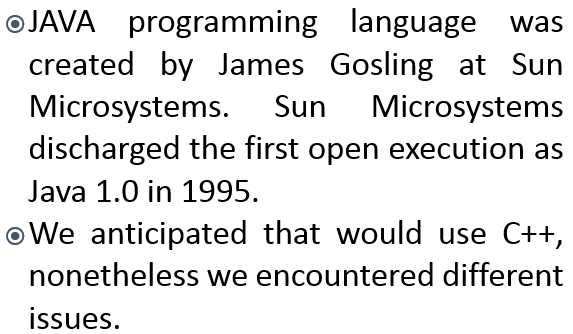
[3.5 Critical reflections on my system development process and lessons learnt 41](#_Toc462223343)

[1. References 42](#_Toc462223344)

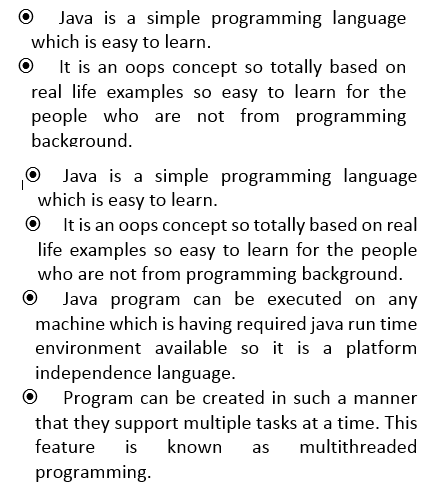


# Task 1 Key OOPS Concept

**Key oops concept**

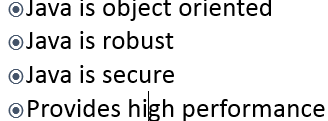




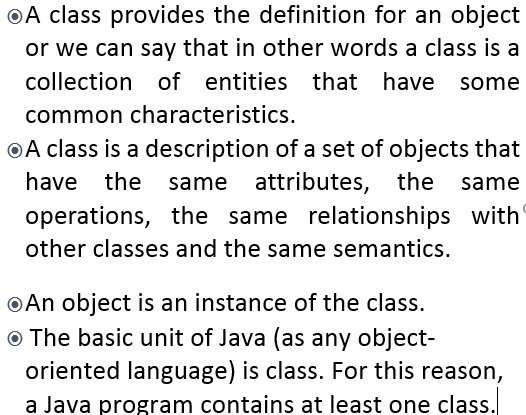


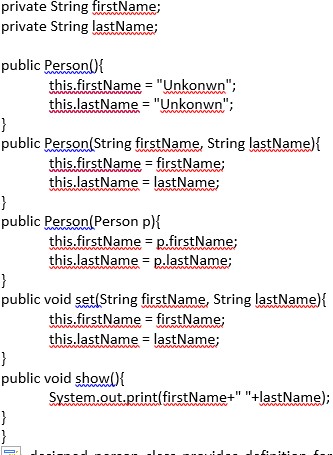
**Features of java programming language**

**Continue…**

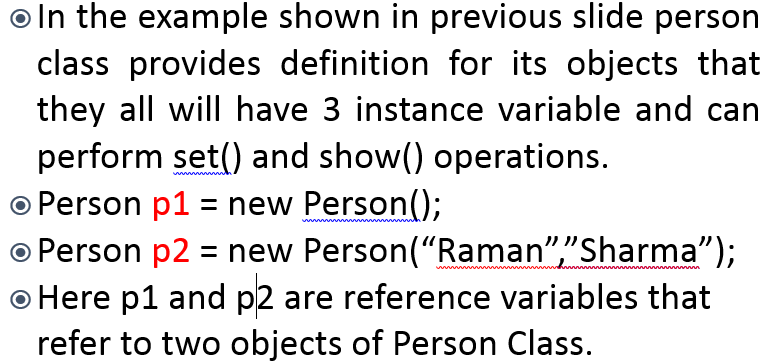


**Classes & Object**

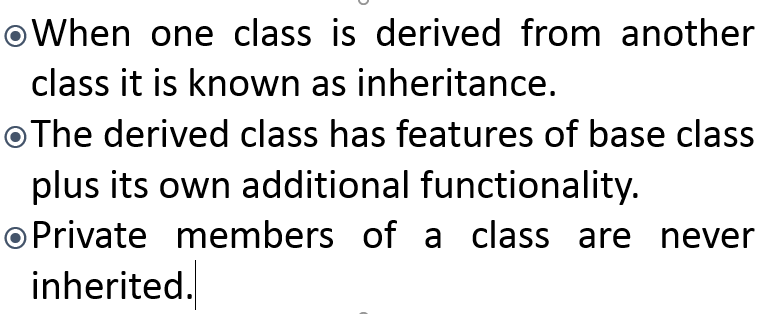




**Continue…**



**Inheritance…**



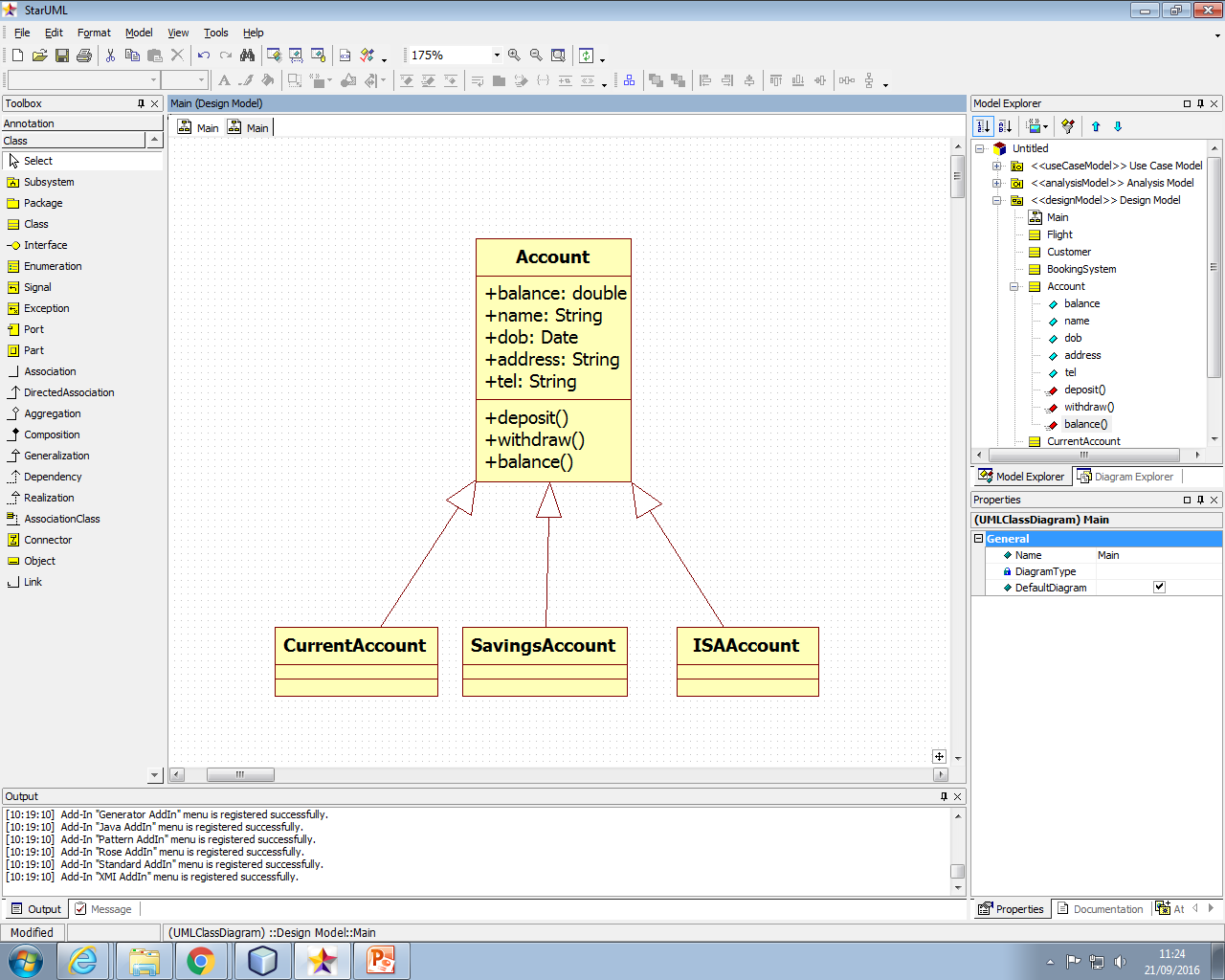
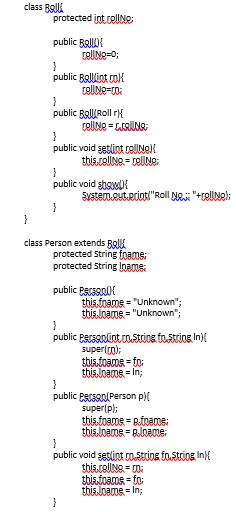
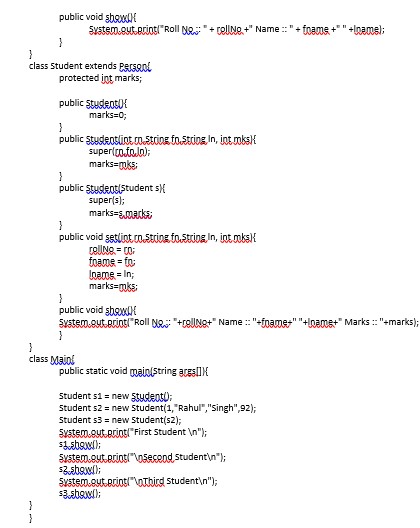
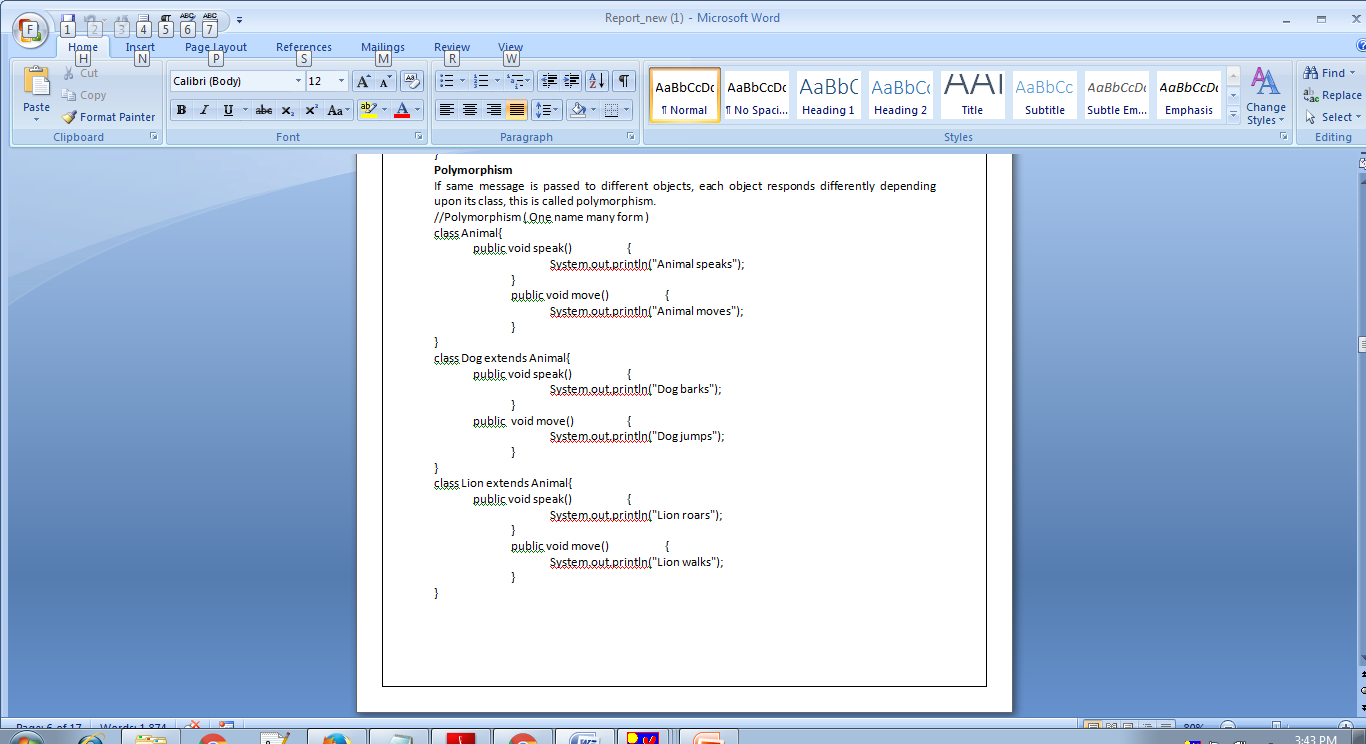


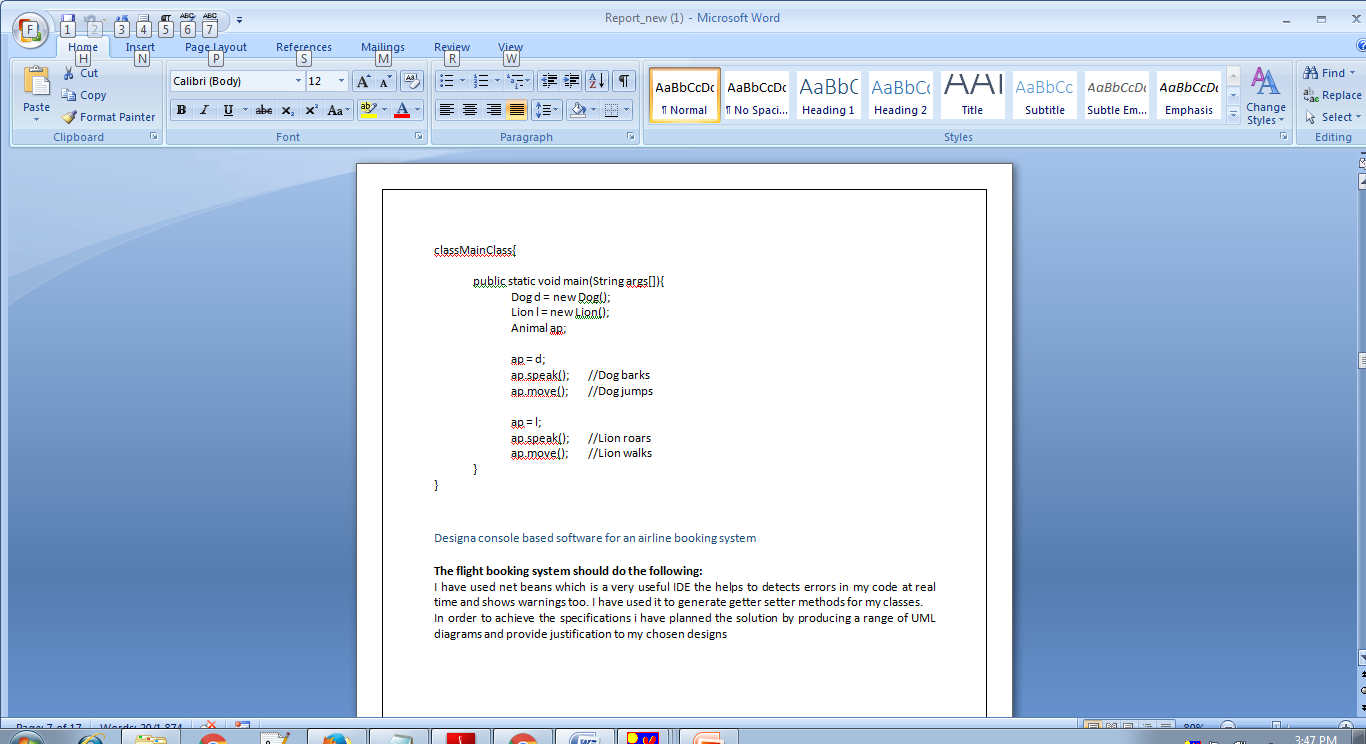
Figure 1 Diagram of an inheritance



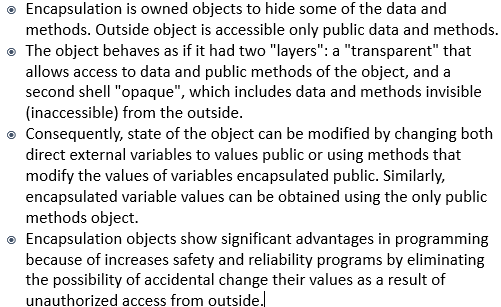
**Polymorphism…**

If same message is passed to different objects, each object responds differently depending upon its class, this is called polymorphism.





**Encapsulation**



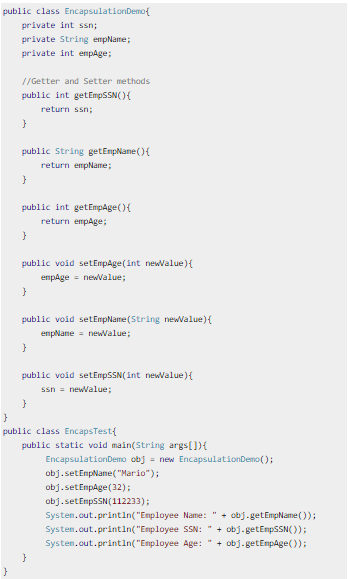


Figure EXEMPLE OF ENCAPSULATION

# Task 2 Design the code, plan the solution by producing a range of UML diagram and reflecting on my solution/ implemented algorithm

***Design a console based software for an airline booking system***

**The flight booking system should do the following:**

I have used net beans which is a very useful IDE the helps to detects errors in my code at real time and shows warnings too. I have used it to generate getter setter methods for my classes.

In order to achieve the specifications, I have planned the solution by producing a range of UML diagrams and provide justification to my chosen designs

## UML Diagram

To build the class diagram, we have to identify their relationships.

The main business objects identified in this application are ***Customer***, ***Flight*** and ***BookingManager***. From the diagram we can notice that one flight aggregates zero or more customers. For this reason, on a flight we can book zero or more customers.

A booking manager can register one or more flights.

All the customer details such as passport ID, name, phone email were encapsulated on ***Customer*** class. Future attributes of the customer will require to change only this class. The only attributes which can be change by an external caller (object) are only the phone and the email. The customer and the passport ID are not writable by the external callers, but only readable. I consider this approach given that our business objects are only persisted in memory at runtime, meaning that I want to preserve the consistency of the two fields passport ID and customer name for all the flights. In addition, I want to have a unique property in order to identify a customer, so that I can check if the same person is going to do a double booking.

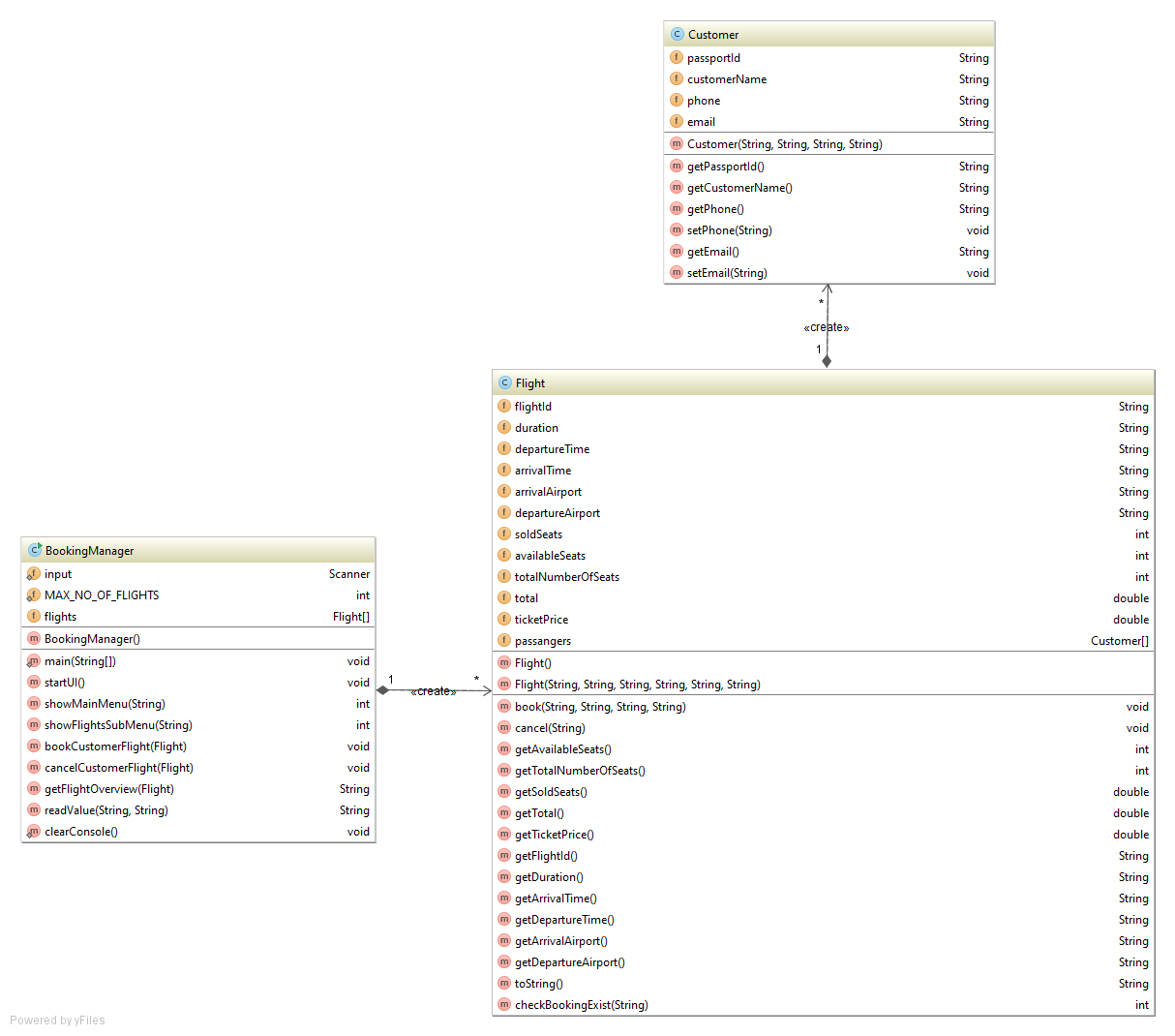
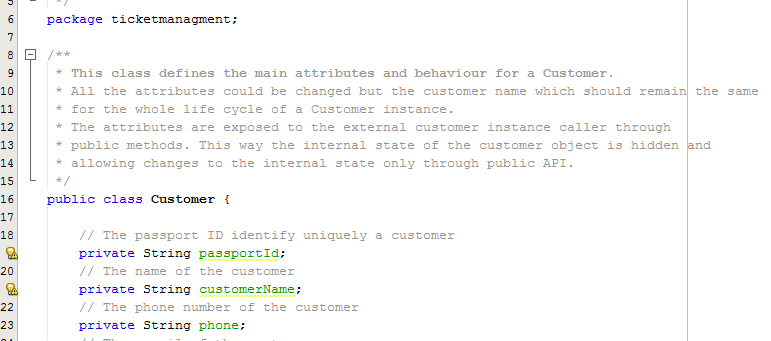


Figure UML Diagram

## Design the software code for an airline booking system

### ***Customer class***

Figure Customer Class in ticketmanagement package



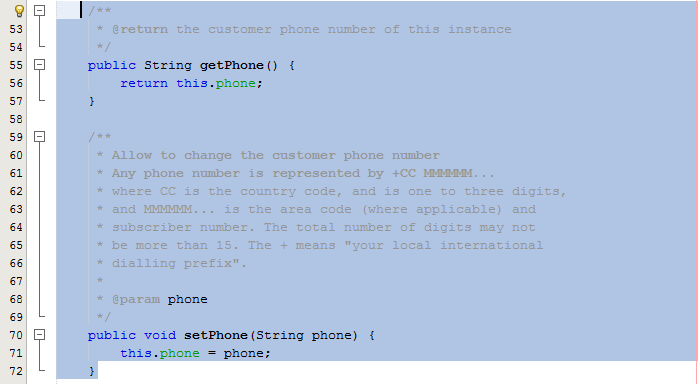


Figure return of the customer phone number of this instance in customer class

### ***Flight class***

Figure Flight class and its variables

***Constructor Flight*** that creates a flight with the mandatory elements (see below).

Flight must uniquely be identified and must contain details such as:

* + duration
  + departure time
  + arrival time
  + departure airport
  + arrival airport

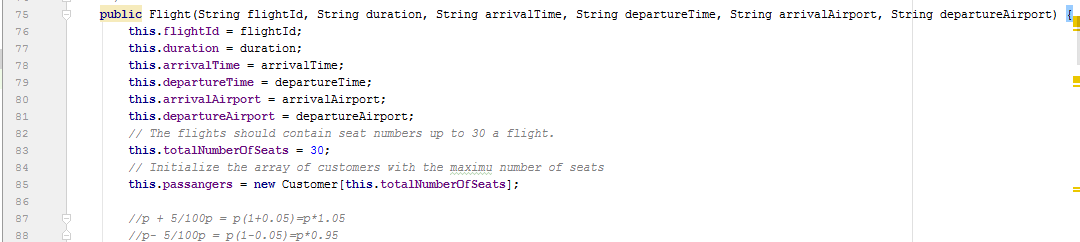


Figure Constructor Flight inside the Flight class

I will come with a ***method*** called ***Book*** customers on flights by customer name, customer phone and customer email.

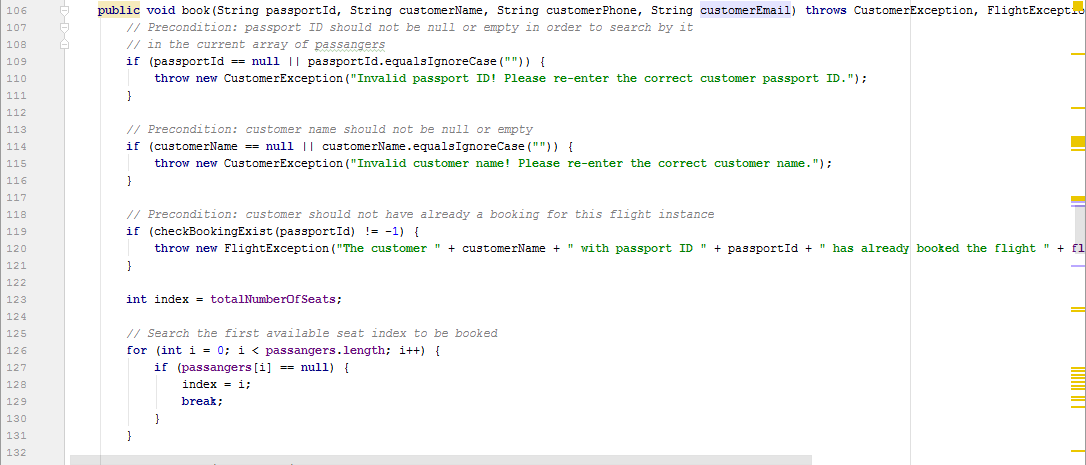


Figure I create a method called book with customer by name, phone and email using preconditions

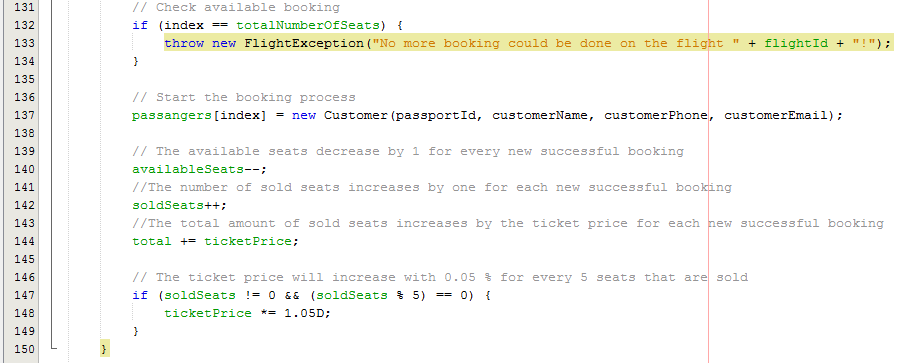


Figure Preconditions in the BOOK method for processing the booking

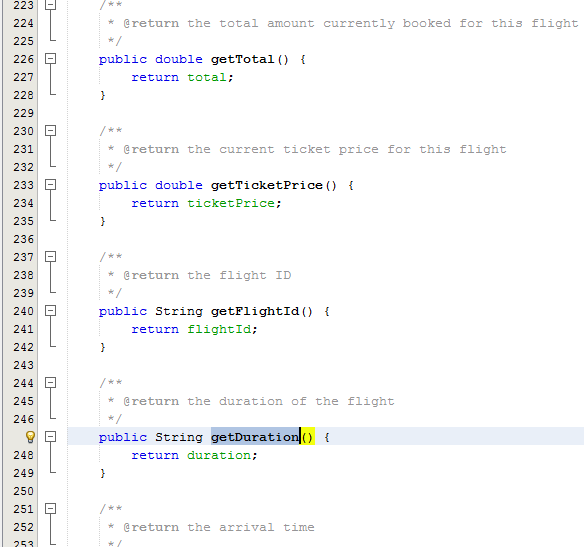


Figure CREATE DIFFERENT METHODS IN FLIGHT CLASS

***BookingManager Class***

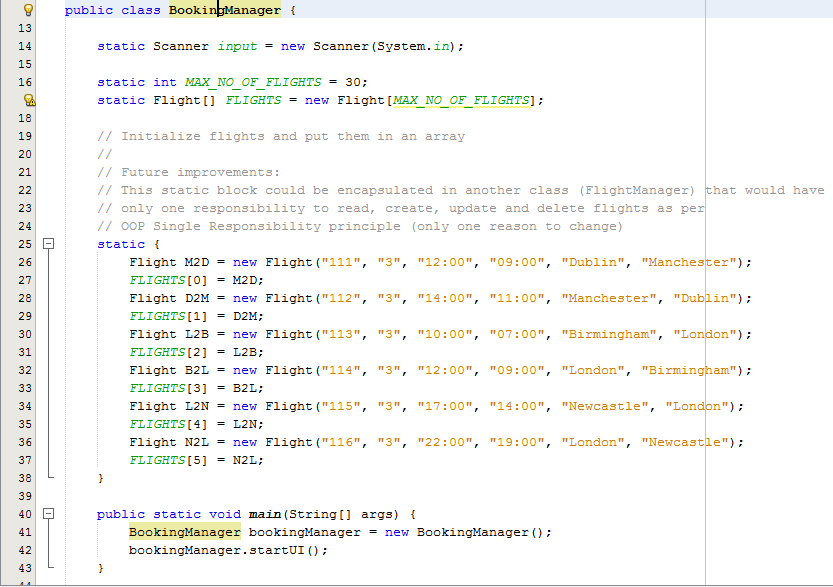


Figure INITIALISING THE FLIGHTS

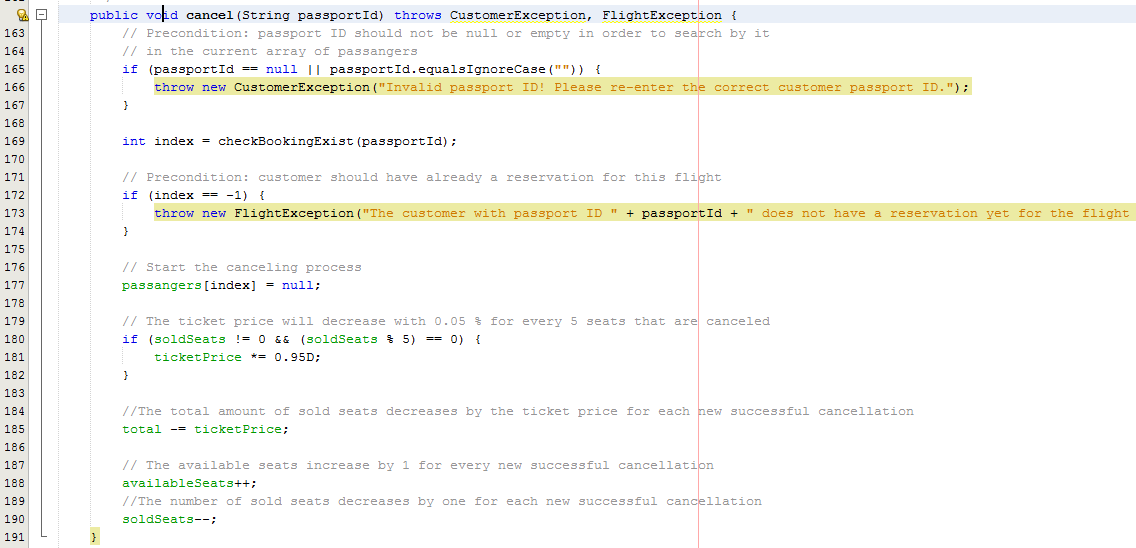


Figure 11 Canceling a booking flight for a customer

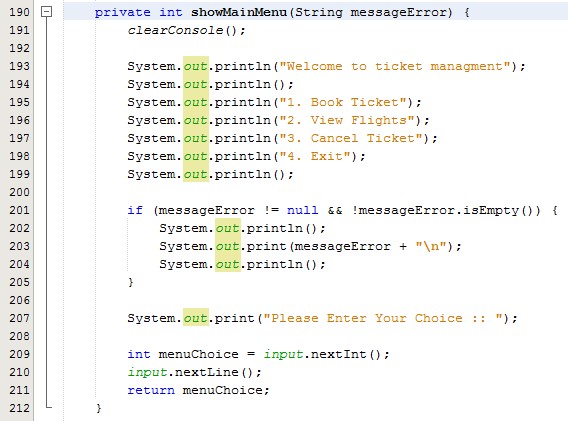


Figure The code for the Main Menu page

## Demonstrate how I overcame problems with the design critically reflecting on my solution / implemented algorithms and provide adequate reasons

|  |  |  |  |
| --- | --- | --- | --- |
| S. No. | Test case | Expected result | Actual Result |
| 1 | **Flight Creation** | The user must be able to create a new flight | Yes, user is able to create a flight |
| 2 | **Book a ticket** | The user must be able to book a ticket | Yes, the user is able to book a ticket. |
| 3 | **Cancel Ticket**. | The user must be able to cancel a ticket | Yes, the user is able to cancel a ticket. |
| 4 | **Menu driven** | The user must be able to view the Menu and the Submenus | Yes, the user is able to view the Menu and the Submenus |
| 5 | **Test for checking how much flights have made** | The user must be able to check how much flights have made | Yes, the user is able to check how much flights have made |
| 6 | **Flights contain details such as: departure, arrival time, duration, departure and arrival airport** | The user must be able to be able to view flight details | Yes, the user is able to see the flight details |
| 7 | **A flight should contain up to 30 seats** | The user must be able to book up to 30 seats for a flight | Yes, the user is able to book up to 30 seats for a flight |
| 8 | Price should increase with 0.05% after 5 seats are sold | The price should increase with 0.05% after 5 seats which are sold | Yes, the price is increase with 0,05 after 5 seats booked |
| 9 | No double booking is accepted | A customer should not be able to do a double reservation for the same flight | Yes, the customer can’t do a double reservation on the same flight |
| 10 | Prices starts from £150 | The price should start from £150 | Yes, the price is starting from £150 |
| 11 | Flights contains departure and destination time and location | The flight should contain the departure and destination time and location | Yes, the flight is showing the departure and destination time and location |

1. Task 3 Be able to implement object oriented programming solutions and Be able to test and document object oriented programming solutions

## Implementing my solution using industrial strength IDE and tools

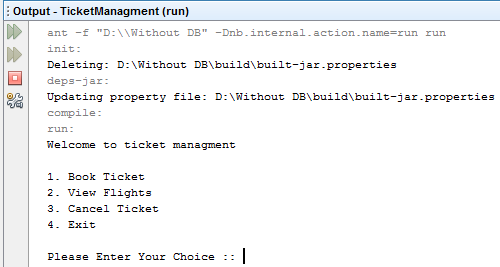
My system will have a help tool to assist users on screen. Screenshots of the latter to show evidence operation of the on screen help. When I run the application, the Menu will be shown with a content line to ask the user to choose a choice from the Menu.

Figure Shown a text line where the user came with a choice from the menu

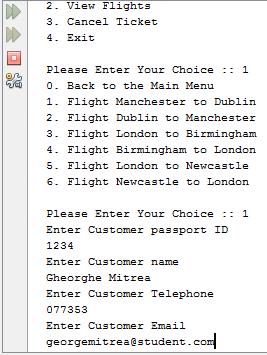
In case the user decides to make a flight reservation, a Submenu will appear and a page with a list of flights will be displayed. After this the user will select the flight for which he needs to add the details of the booking, like passport ID, name of the customer, phone number and email of the customer:

Figure 14 Input from the user when booking a flight

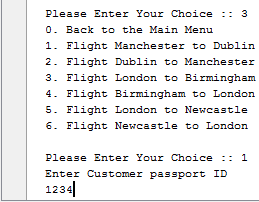
In case a customer already booked a flight and he wants to cancel it, the user must select the option for cancelling a book flight which is option 3 in my application. After selecting this option, the screen line is coming with a message where the user must to input the details of the customer and this action ask only the details of passport ID of the customer who wants to cancel a flight:

Figure the inputs from the cancel submenu to cancel a flight already booked

## Provide a technical documentation for my software

For my airline booking system application, it will be applied the fundamentals of object-oriented programming.

The reason why the programmers chose to create applications in object oriented language because it gives a proper solution to use the objects for an easier update, an easier way to build it and also a simpler way to maintain the application.

### I will start the documentation by showing you how I coded the ***Customer class.***

This class represents a customer or the person that books the ticket. It is much easy to handle a customer as an object rather then some individual class so I have implemented this class which contains instance variable that stores information about the customer.

* This class defines the main attributes and behavior for a Customer,
* All the attributes could be changed but the customer name which should remain the same, for the whole life cycle of a Customer instance,
* The attributes are exposed to the external customer instance caller through,

public methods. This way the internal state of the customer object is hidden and,

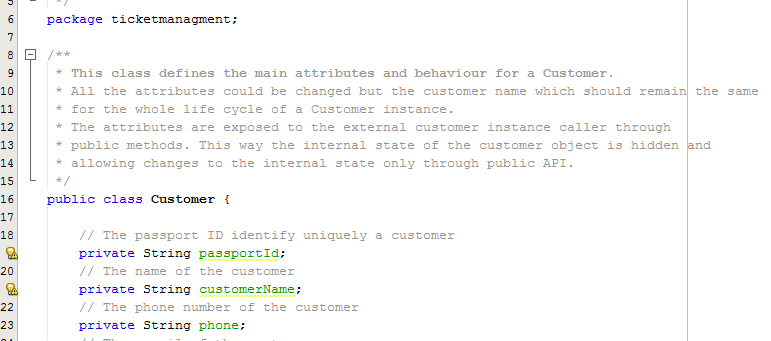
* Allow changes to the internal state only through public API.

Figure coding the customer class

### ***Flight class***

In ***Flight class*** I have encapsulated all the details related to a flight and also, the main logic for booking, cancelling and viewing a flight for a specific customer.

In Booking Manager class represents the interface to the flight operator user. It incorporates the logic for the showing the user interface to the user and to perform the main functionality support by the application such as cancel, book and show a flight overview. It delegates this main business functionality to the Flight class.

This class defines the main attributes and behavior for a Flight. The main responsibility of this class is to manage flights:

* cancel customer bookings on demand;
* flights should contain a departure location and destination location;
* price should be based on availability and the price should increase by 0.05%;

after every 5 seats are sold;

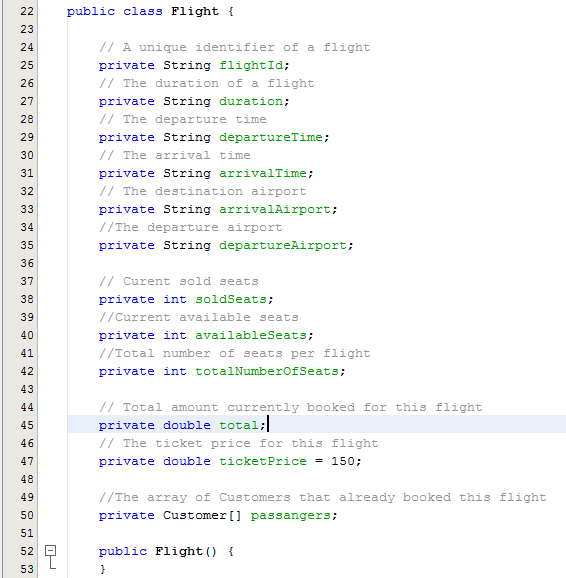
* there should be no double booking;
* a facility to show how much a flight has made should be available for management decision making;
* Prices to begin with, must be £150;

Figure FLIGHT CLASS AND ITS VARIABLES

***Constructor Flight*** that creates a flight with the mandatory elements (see below).

Flight must uniquely be identified and must contain details such as:

* + duration
  + departure time
  + arrival time
  + departure airport
  + arrival airport

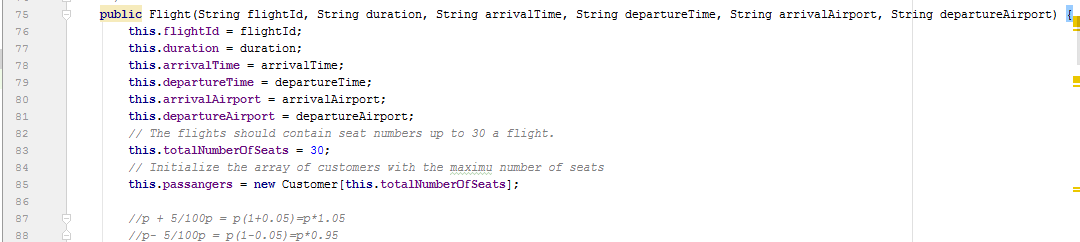


Figure CONSTRUCTOR FLIGHT INSIDE THE FLIGHT CLASS

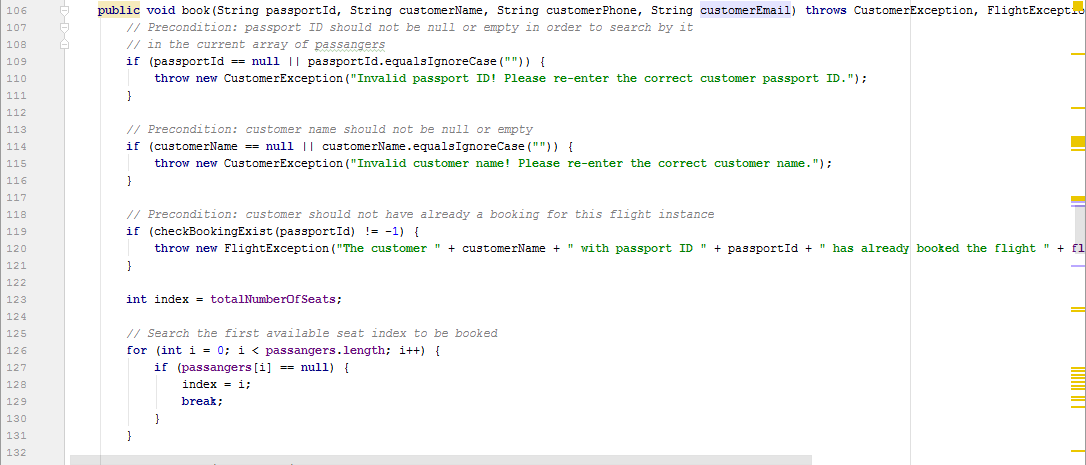
I will come with a ***method*** called ***Book*** customers on flights by customer name, customer phone and customer email.  
  
 As a precondition for a booking to be valid the same customer should not be  
already booked for the same flight. Another precondition is that Passport ID should not be null or empty in order to search by it in the current array of passengers. Then I come with a precondition where customer name should not be null or empty. The next precondition is that the customer should not have already a booking for this flight instance.

Figure 18CREATE A METHOD CALLED BOOK WITH CUSTOMER BY NAME, PHONE AND EMAIL USING PRECONDITIONS

Also, as a precondition I start to:

* check available booking,
* Configure the code for starting the booking process,
* I set the available seats to decrease by 1 for every new successful booking,
* I configure the number of sold seats increases by one for each new successful booking,
* I set up a code where the total amount of sold seats increases the ticket price for each new successful booking

These preconditions you can see them in the below picture:

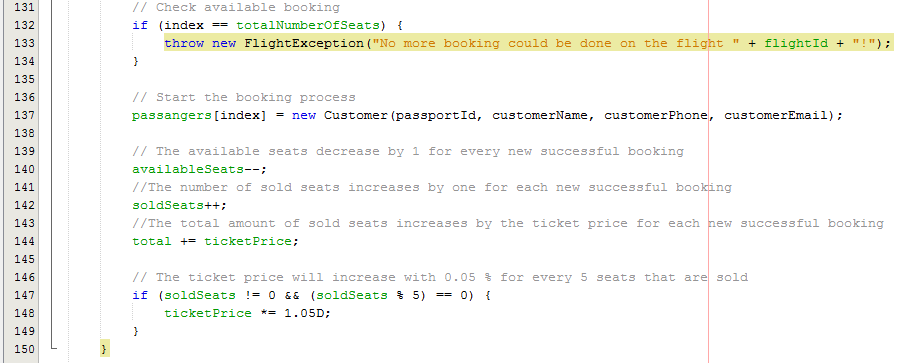


Figure PRECONDITIONS IN THE BOOK METHOD FOR PROCESSING THE BOOKING

Also, in the Flight Class, I have created different types of methods:

* Integer methods like ***getAvailableSeats*** which return the available seats and also ***getTotalNumberOfSeats*** which returns the total number of seats,
* Also, I used a Double method ***flightSales*** which return the total flight sales,
* After I come with a method called ***setTicketPrice*** with the price of the flight ticket,
* Then I have used some String methods (***getFlightId***, ***getArrivalTime***, ***getDepartureTime***, **getArrivalAirport**).

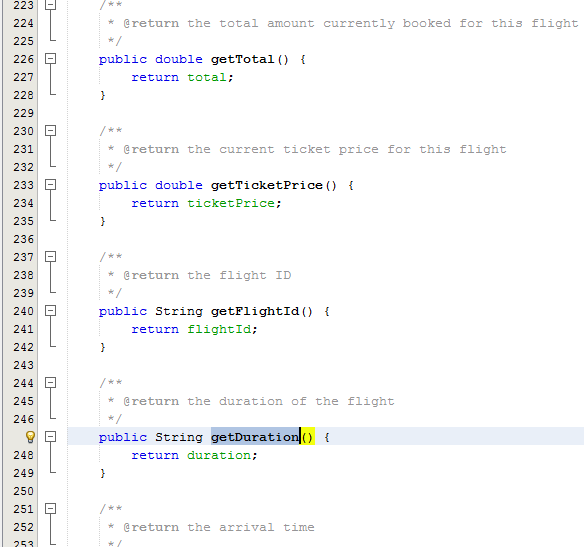


Figure CREATING DIFFERENT TYPES OF METHODS IN FLIGHT CLASS WHICH SHOULD RETURN OR NOT DIFFERENT RESULT

***BookingManager Class*** - The main responsibility of this class is to manage the UI with the

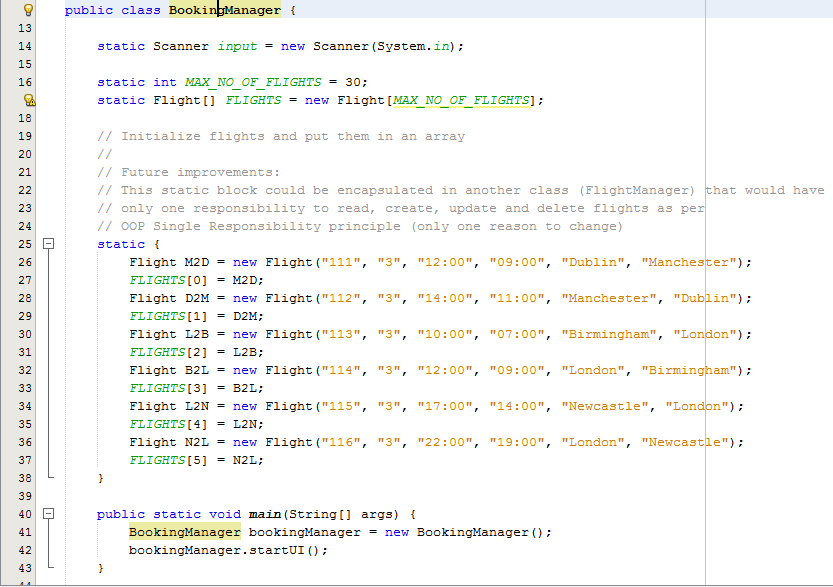
flight operator but also initializes and register a few default flights. For this reason, I used to initialise the flights by using an array

Figure SHOW OF USING THE ARRAY TO INITIALISE THE FLIGHTS

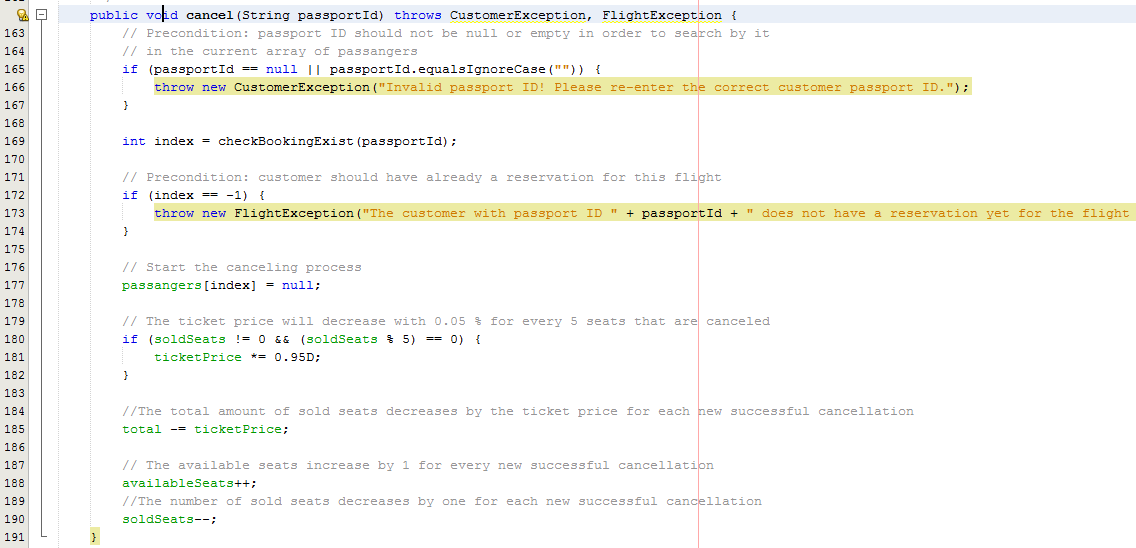
To cancel an existing booking flight for a customer based on his passport ID, I have used ***a String method called cancel:***

Figure code for canceling a booked flight

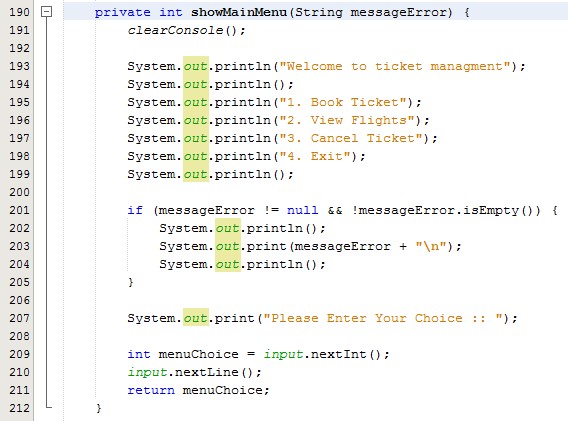
Also, in the ***BookingManager Class*** is shows the ***Menu Driven*** with ***the Main Menu page*** and ***the Submenus pages.*** 

Figure CODE FOR MENU DRIVEN

## Testing phase of my application

Testing stage is critical in the product advancement life cycle and it has the intend to enhance the quality and dependability and to check if the application has met the given prerequisites or the product is doing what it shouldn't do.

|  |  |
| --- | --- |
| Application Requirements | Achieved/not |
| Menu driven | ✓ |
| Book customers on flights | ✓ |
| No double booking is accepted | ✓ |
| Cancel customer’s bookings on demand | ✓ |
| Flights have departure and destination location | ✓ |
| Price should increase with 0.05% after 5 seats are sold | ✓ |
| A flight should contain up to 30 seats | ✓ |
| Prices starts from £150 | ✓ |
| A facility to show how much flight has made | ✓ |
| Flights contain details such as: departure, arrival time, duration, departure and arrival airport | ✓ |

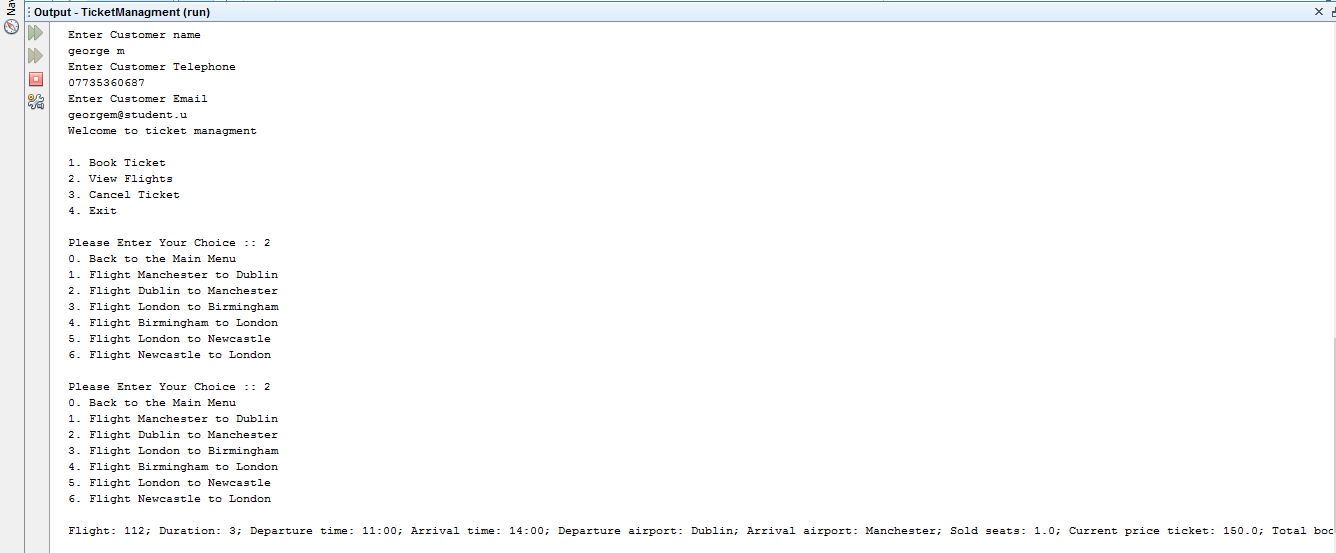
In figure 23 the application has a menu driven the value of a flight ticket begins from £150. On the off chance that the client chooses the book flight choice, the application will ask for passport ID, customer name, customer’s phone number and customer email details. The application will show the output data of the flight ID departure time arrival time, departure airport arrival airport, the number of sold seats for that flight the current price ticket and total amount for that flight:

Figure The output of the application after booking a flight

As you can see in the figure 24 the price is increasing with 0.05% after 5 bookings. The ticket price for the next 5 bookings will be £157.5:

Figure Increasing ticket price

In figure 25 the clients can cancel their booked flight. The seats are decremented by one and cost of the tickets diminished with 0.05% because there are just four flights booked:

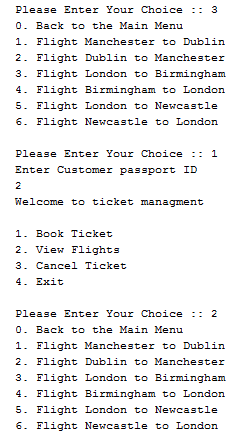
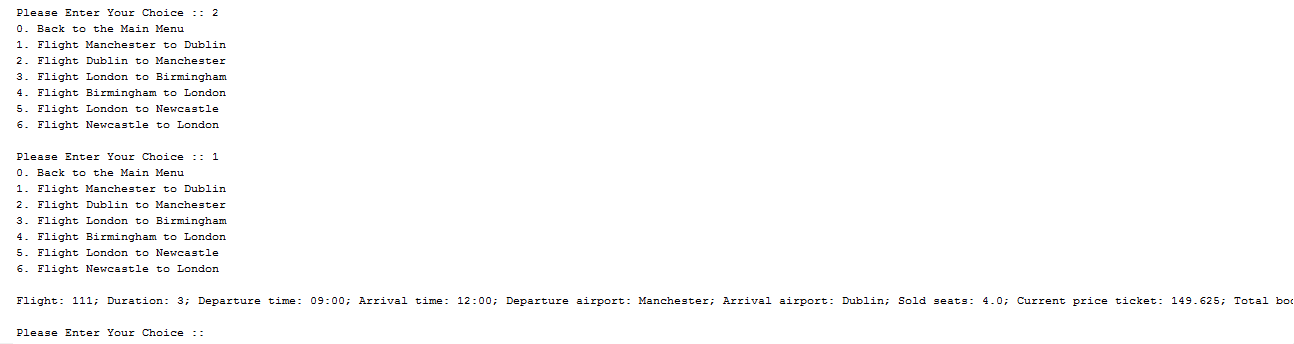


Figure 26 THE OUTPUT DATA TO CANCEL A FLIGHT TICKET

Figure the output data to cancel a flight ticket

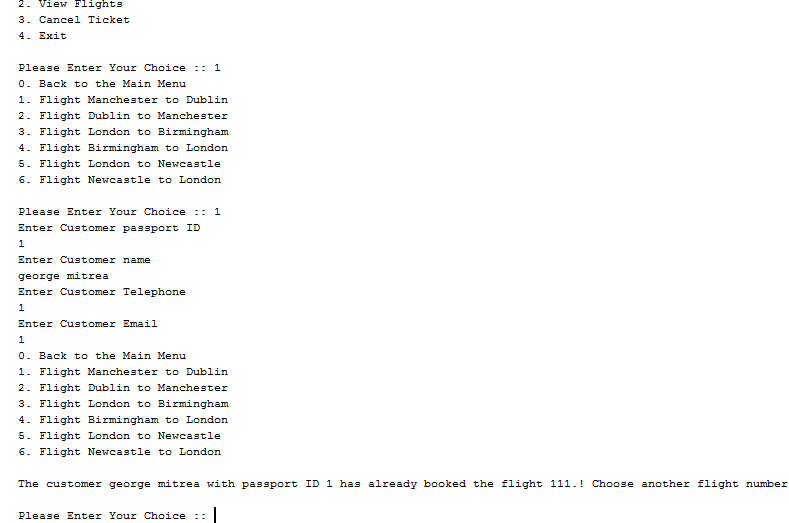
As you can see in figure27 the no-double booking requirement is achieved. This means that a customer cannot book a flight if he already reserved a seat in the same flight and a application will come with a message to create another flight number:

Figure no double booking

In figure 28 I will show you how the output of the application will display the message that all the available seats for that flight are reached and is not more possible to book another reservation on this flight:

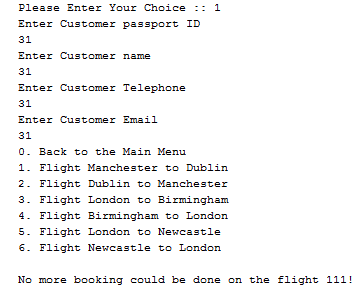


Figure The output of the application when the maximum no. of reservations were reached and the user can book anymore on this flight

## User feedback for the system and analyse and comment on the feedback

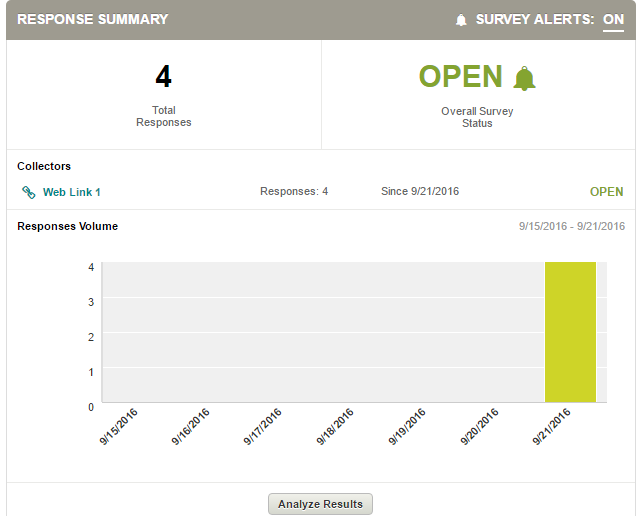


Figure response summary of the feedback

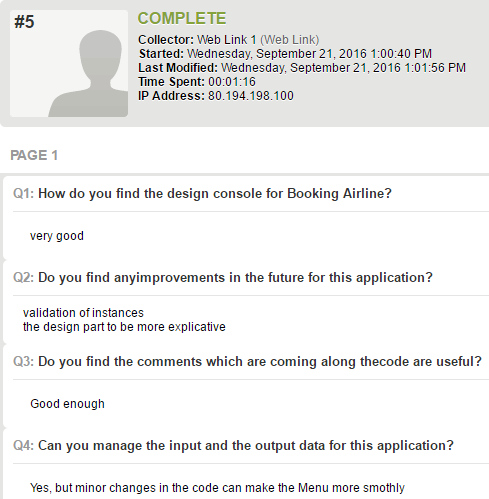


Figure comments on feedback

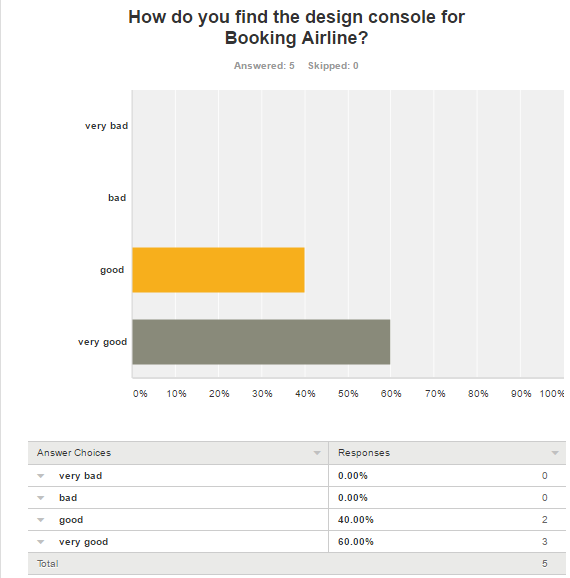


Figure feedback on the design console

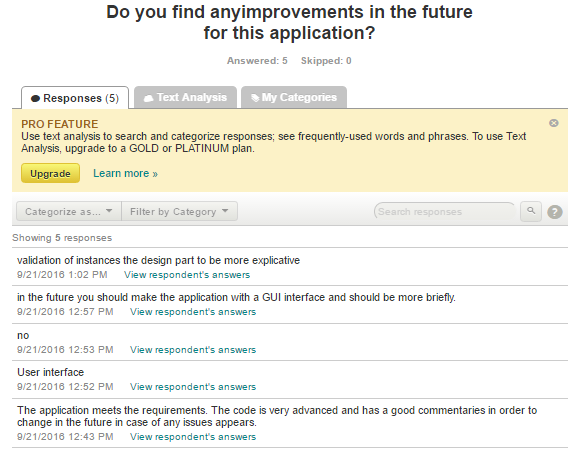


Figure feedback meaning of the improvements

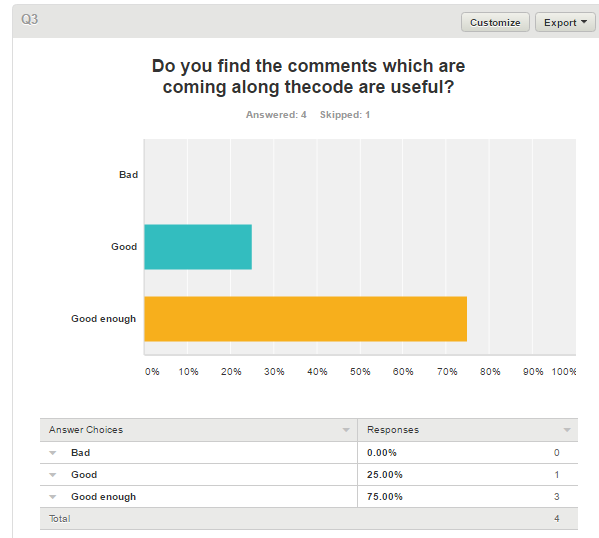


Figure feedback on the code if is useful

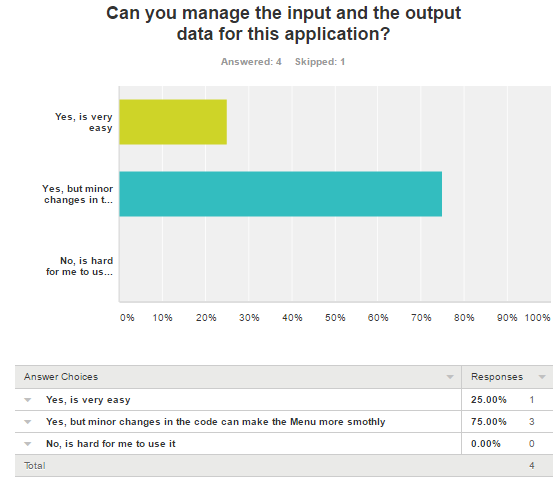


Figure Feedback related to the input and the output data

## Critical reflections on my system development process and lessons learnt

The report is generated in order to successfully implement the specified program and improve the learner’s skills. This assignment provides a good platform to understand the fundamental concepts of java and apply my knowledge and skills through implementation.

We must understand that by simply learning a language that supports object-oriented programming does not automatically learn to program correctly the model object! To learn this, I must understand and apply concepts and mechanisms underlying this model and I propose just that.

# References

JavaWorld.(2016). *Welcome to JavaWorld.com*. [online] Available at: http://www.javaworld.com/ [Accessed 4 May 2016].

Studytonight.com, (2016).Features of Java | Core Java Tutorial | Study tonight. [online] Available at: http://www.studytonight.com/java/features-of-java.php [Accessed 20 Jan. 2016].

www.javatpoint.com, (2015). JVM | Java Virtual Machine - Javatpoint. [online] Available at: http://www.javatpoint.com/internal-details-of-jvm [Accessed 19 Dec. 2015].

Javaperformancetuning.com, (2016).Java performance tuning tips. [online] Available at: http://www.javaperformancetuning.com/tips/rawtips.shtml [Accessed 20 Jan. 2016].

Docs.oracle.com. (2016).JavaServer Pages Technology - The Java EE 5 Tutorial. [online] Available at: http://docs.oracle.com/javaee/5/tutorial/doc/bnagx.html [Accessed 3 May 2016].