Object Oriented Programming

Philip Woulfe  | Dublin

Report, Design and Testing document

Programming Solutions Ltd.

2016

Contents

[Task One - Principles, Characteristics and Features of Object Oriented Programming 2](#_Toc446423219)

[Introduction 3](#_Toc446423220)

[Principles, Features & Characteristics 4](#_Toc446423221)

[Shared with Non Object Oriented Programming Languages 4](#_Toc446423222)

[Classes and Objects 4](#_Toc446423223)

[Inheritance 4](#_Toc446423224)

[Encapsulation 5](#_Toc446423225)

[Abstraction 5](#_Toc446423226)

[Polymorphism 5](#_Toc446423227)

[Conclusion 7](#_Toc446423228)

[Task Two - Airline Booking System Design Document 8](#_Toc446423229)

[User Stories 9](#_Toc446423230)

[0001 – Customer Password Validation (Sprint 2) 9](#_Toc446423231)

[0002 – Administrator Password Validation 9](#_Toc446423232)

[0003 – Booking flights 9](#_Toc446423233)

[0004 – Confirmation Screen 10](#_Toc446423234)

[0005 – Maintenance Suite 10](#_Toc446423235)

[GUI Layouts 11](#_Toc446423236)

[UML 17](#_Toc446423237)

[Database 18](#_Toc446423238)

[Task Three – Source Code 19](#_Toc446423239)

[Task Four 20](#_Toc446423240)

[Testing of the application 21](#_Toc446423241)

[Analysis of actual test results against expected results to identify discrepancies 0](#_Toc446423242)

[Customer Password Validation Test Cases (Sprint 2) 0](#_Toc446423243)

[Administrator Password Validation Test Cases (Sprint 2) 1](#_Toc446423244)

[Flight Booking Test Cases 2](#_Toc446423245)

[Flight Confirmation Test Cases 4](#_Toc446423246)

[Page may or may not have been intentionally left blank 6](#_Toc446423247)

[Recommendations for improvements to the program 0](#_Toc446423248)

[Documentation for use of the program by other programmers 1](#_Toc446423249)

[A program with onscreen help to guide users of your program 2](#_Toc446423250)

[Bibliography 3](#_Toc446423251)

# Task One - Principles, Characteristics and Features of Object Oriented Programming

## Introduction

Object-oriented programming (OOP) is a programming paradigm which is based on the concept of objects which contain attributes and methods. The attributes store data about the object and the methods are ways of manipulating that data. Many OOP languages use classes as the basis for the objects. Classes can be thought of as blueprints for the object that is created from them. OOP works by making these objects interact with each other.

One of the earliest languages to use the concept of objects was SIMULA 67 in 1961. (Holmevik, 1994) By the early 90s, OOP had become the dominant paradigm with languages like C++ and Visual FoxPro becoming available. Many non OOP languages like FORTRAN and COBOL had OOP features added to them. Two of the most important commercial OOP languages are Java and C#.

## Principles, Features & Characteristics

### Shared with Non Object Oriented Programming Languages

Some features of OOP that are shared with non OOP languages are variables and procedures.

#### Variables

Variables are used to store information. These can be primitive variables such as:

* natural numbers
* characters
* floating point numbers
* booleans

Or object variables such as:

* strings
* lists
* even object versions primitive variables

Primitive variable work by holding the information directly, whereas object variables work by holding a reference to the object location.

#### Procedures

Procedures are a way of manipulating data. This can be methods taking input and providing output or it can be structures like if statements or loops.

### Classes and Objects

#### Classes

Classes are the blueprint for an object, you may create a *Dog* class and give it a *numberOfLegs* variable and a *bark()* method. The *numberOfLegs* variable will hold a number denoting how many legs the dog has and whenever the *bark()* method is called, the dog will bark. However the dog cannot bark until it is created.

#### Objects

Objects are instances of a class. You may take the *Dog* class and instantiate it. In this Java example, the first *Dog* refers to the variable type, which is the object the variable is going to hold, *dougal* refers to the name of the variable (i.e. the dog’s name), new tells Java to create a new object and *Dog()* is the constructor method. This is located in the *Dog* class and tells Java how to create a *Dog* object.

**Dog** dougal = **new** Dog();

You can now check the number of legs the dog has by checking *dougal.numberOfLegs* and you can make him bark by calling *dougal.bark().*

### Inheritance

Inheritance is a way of reusing code in multiple classes. For example, the previously mentioned *Dog* class shares some attributes with a *Human* class. They both have legs, they both have eyes, and they both feed their young milk; although some of these values may be different. Instead of writing out the same code twice, we can write a superclass called *Mammal* and include all the common features of both in it. When the *Dog* object is created, *numberOfLegs* is set to *4* and when the human object is created *numberOfLegs* is set to *2*. The different methods like *bark()* and *speak()* can be written into the *Dog* and *Human* classes respectively, or we could have a *speak()* method in the *Mammal* class and implement it differently for *Dog* and *Human*.

### Encapsulation

Encapsulation is a way of hiding information to prevent it from being accessed outside a class. Java does this by using keywords such as *public*, *protected*, *private* and, although it is not a keyword, just an absence of one, *default*. *Public* information can be accessed from any object, *protected* can be accessed by the class, the package and subclasses, *default* can be accessed by the class and package and *private* can only be accessed by the class.

#### Accessor

Accessor methods can also be called getter methods. Accessor methods are used to allow information to be retrieved from variables you can’t access directly. This means you can read the contents of the variable, but not change it. To change the value, you must use a mutator method.

#### Mutator

Mutator methods allow private variables to be changed, but also allow validation of the data you are trying to store in the variable. Mutator methods can also be called setter methods.

In this Java example, this *setAge()* setter method won’t allow age to be set to a negative number from outside the class. Instead it prints an error message to the console. If the object age variable was public, it could be set to any number from negative 2 billion to positive 2 billion. Using setter methods like this prevents other programmers from implementing your classes in a way you didn’t intend.

**private** **int** age;

**public** **void** setAge(**int** age) {

**if** (age >= 0) {

**this**.age = age;

} **else** {

System.out.println("Age must be a positive integer");

}

}

### Abstraction

Abstract classes are classes that cannot be instantiated into objects. An example of this would be the Mammal class previously mentioned. There is no animal known only as a mammal, all mammals are another type of animal.

Abstract methods are methods which are defined by the input and output type but are not implemented.

The mammal class would have abstract methods such as *createYoung()*. The reason for this abstract method is the fact that all mammals produce offspring, however, not all mammals give birth to live young; monotremes such as the platypus and the echidna lay eggs instead; so while both create offspring, the logic for doing so would be different.

### Polymorphism

Polymorphism comes from the Greek, *polys* meaning “many” and *morphē* meaning “form”. Polymorphism is when calling code can accept a parent class or any of its sub classes.

In this Java example, *Platypus* implements *EggLayer* and extends *Mamma*l, therefore

* A Platypus IS-A EggLayer
* A Platypus IS-A Mammal
* A Platypus IS-A Object (All classes in Java extend object)

**public** **interface** EggLayer{}

**public** **abstract** **class** Mammal{}

**public** **class** Platypus **extends** Mammal **implements** EggLayer{}

This means these classes can be created and assigned as follows.

Platypus p = **new** Platypus();

Mammal m = p;

EggLayer e = p;

Object o = p;

All the reference variables p, m, e, o refer to the same *Platypus* object.

## Conclusion

In conclusion, OOP is a powerful programming paradigm which has many beneficial features and characteristics.

# Task Two - Airline Booking System Design Document

## User Stories

### 0001 – Customer Password Validation (Sprint 2)

As a customer, I would like to be able to login to the application so I can book a flight.

Customers should be provided with a username and password which allows them to login to the application. Entering an incorrect username or password should display an error message.

|  |  |
| --- | --- |
| Criteria 1 | Entering correct username and password allows access to the application |
| Criteria 2 | Cancel button exits application |

#### Happy Case:

#### Error Validation:

|  |  |
| --- | --- |
| Criteria 1 | If an incorrect username or password is entered, display message: “Username or password incorrect” |
| Criteria 2 | If an incorrect username or password is entered, the password field will clear on clicking the Ok button |
| Criteria 3 | If username field is left blank, display error “Username is a required field” |
| Criteria 4 | If password field is left blank, display error message “Password is a required field” |
| Criteria 5 | If the database cannot be reached, display error “Database cannot be reached” |

### 0002 – Administrator Password Validation

As an administrator, I would like to be able to login to the application so I can add or remove airports from the application and view details of booked flights.

Administrators should be provided with a username and password which allows them to login to the application. This should allow them access to the maintenance suite. Entering an incorrect username or password should display an error message.

#### Happy Case:

|  |  |
| --- | --- |
| Criteria 1 | Entering correct username and password allows access to the maintenance suite. |

#### Error Validation:

|  |  |
| --- | --- |
| Criteria 1 | If an incorrect username or password is entered, display message: “Username or password incorrect” |
| Criteria 2 | If an incorrect username or password is entered, the password field will clear on clicking the Ok button |
| Criteria 3 | If username field is left blank, display error “Username is a required field |
| Criteria 4 | If password field is left blank, display error message “Password is a required field” |
| Criteria 5 | If the database cannot be reached, display error “Database cannot be reached” |

### 0003 – Booking flights

As a user, I would like to be able to select criteria in the application so I can book flights.

Customers should be able to select three criteria, namely departure airport, destination airport and type of seat for each leg of the journey. The customer should be able to select a second leg, on from the first destination or back to the original airport. The passenger name should be recorded and a booking reference assigned. On completion, details of the flight(s) should be displayed on screen for the customer to confirm. On confirmation, details should be written to a file.

#### Happy Case:

|  |  |
| --- | --- |
| Criteria 1 | GUI should use combo boxes, radio buttons and at least one checkbox. |
| Criteria 2 | Application should have around eight airports. |
| Criteria 3 | Departure airport, destination airport and type of seat should be selectable for each leg of the journey. |
| Criteria 4 | Departure and destination airports should be displayed in alphabetical order. |
| Criteria 5 | It should be possible to book a second leg, onward from the first destination or back to the original departure airport. |
| Criteria 6 | There should be three choices of seat: Economy, Business & First Class |

#### Error Validation:

|  |  |
| --- | --- |
| Criteria 1 | Departure and destination airport cannot be the same. |
| Criteria 2 | If any combo box is blank, do not allow booking flights and display error message |

### 0004 – Confirmation Screen

As a user, I would like my booking details to be displayed on screen so I can review the details before I confirm them.

Users should have the booking details displayed on screen so they can ensure all the information is correct before confirming the flight.

#### Happy Case

|  |  |
| --- | --- |
| Criteria 1 | Passenger name should be recorded and a Booking Reference assigned. |
| Criteria 2 | When completed, details of the flight(s) booked should be displayed on the screen for the user to confirm. |
| Criteria 3 | On confirmation, details should be written to a file. |

#### Error Validation:

|  |  |
| --- | --- |
| Criteria 1 | If the information cannot be written to the database, display error message |

### 0005 – Maintenance Suite

As an administrator, I would like to be able to Access a maintenance suite so I can modify the application.

Administrators should be able to access the maintenance suite in order to add or remove airports to the application and also to view the booked flights. Each airport will have a unique three character code; this will prevent the same airport being added twice.

#### Happy Case:

|  |  |
| --- | --- |
| Criteria 1 | Administrators should be able to add or remove airports from the application. |
| Criteria 2 | When an airport is removed, it should no longer appear in the application. |
| Criteria 3 | When an airport is added, it should appear in the application. |
| Criteria 4 | Administrators should be able to view booked flights from the maintenance suite. |

#### Error Validation:

|  |  |
| --- | --- |
| Criteria 1 | Administrators should not be able to add the same airport twice. (Sprint 2) |

## GUI Layouts

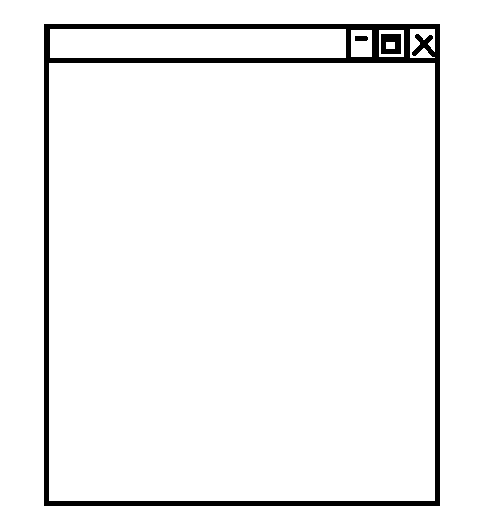


Figure 1 - Main GUI

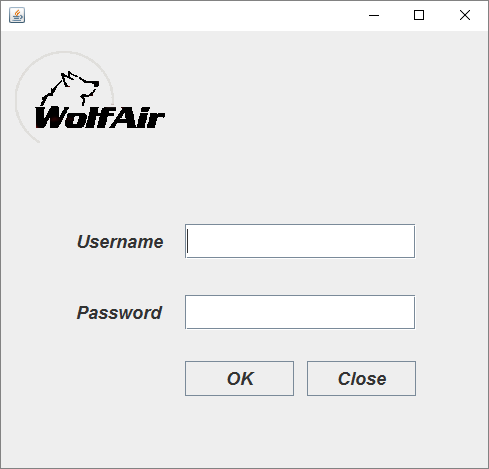


Figure 2 - Login GUI

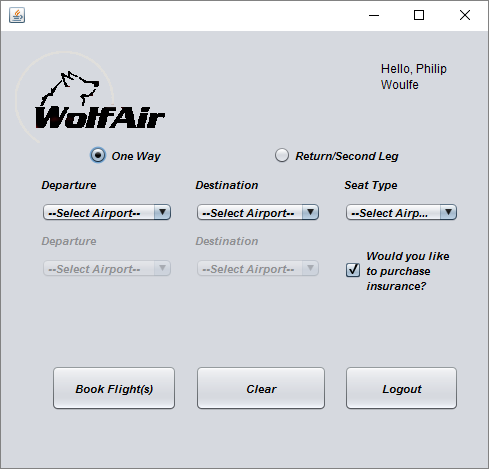


Figure 3 - Customer GUI

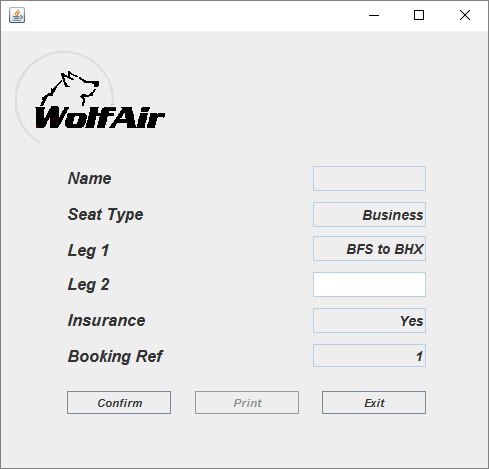


Figure 4 - Customer Confirmation Screen

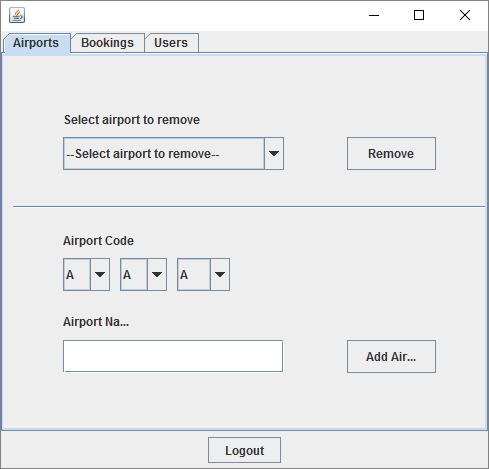


Figure 5 - Airport Maintenance Screen

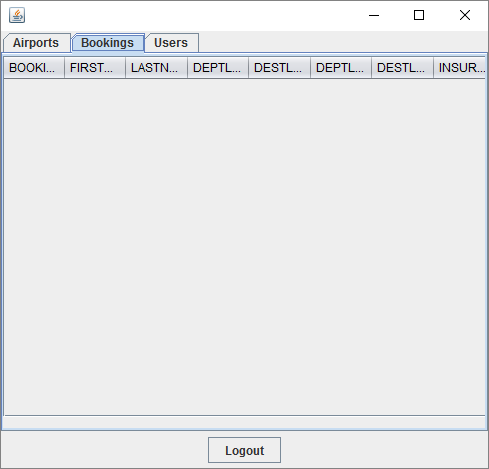
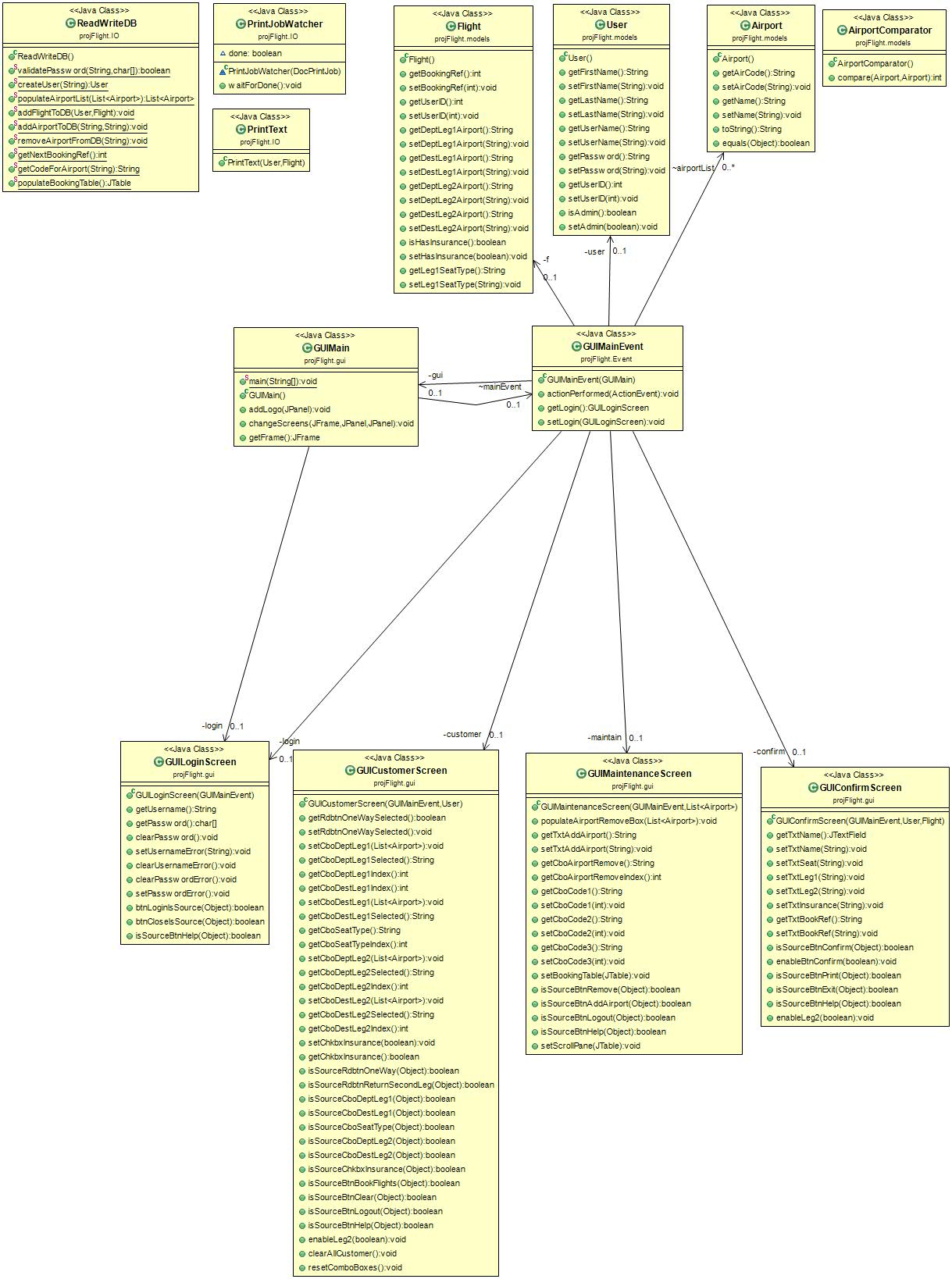


Figure 6 - Flight Maintenance Screen

## UML



## Database

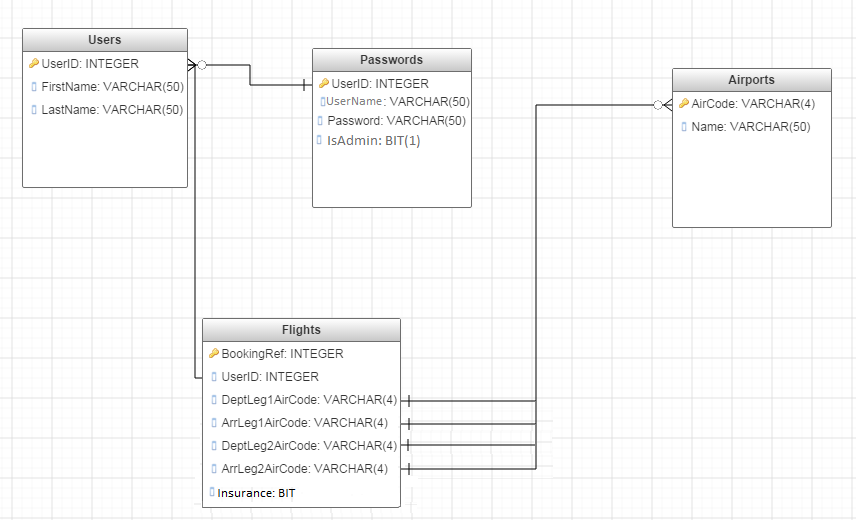


Figure 7- Database Diagram

# Task Three – Source Code

Source code can be found in the [..\src\projFlight](../src/projFlight) directory.

# Task Four

## Testing of the application

All features of the application will be tested manually. The test report can be found below.

## Analysis of actual test results against expected results to identify discrepancies

### Customer Password Validation Test Cases (Sprint 2)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Customer Password Validation (Sprint 2)** | | | | | |
| **Test Case ID** | **Condition** | **Expected Result** | **Actual Result** | **Pass/Fail** | **Comments** |
| 1.1 | Entering valid matching customer username and password from database and click Login | Allows access to the customer section of the application | As Expected | Pass |  |
| 1.2 | Clicking cancel | Exits Application | As Expected | Pass |  |
| 1.3 | Enter incorrect username or password and click Login | Display error message: “Invalid username or password” | As Expected | Pass |  |
| 1.4 | Leave username field blank and click Login | Display error message: “User name cannot be blank” | As Expected | Pass |  |
| 1.5 | Leave password field blank and click Login | Display error message: “Password cannot be blank” | As Expected | Pass |  |
| 1.6 | If the database cannot be reached | Display error message: "Unable to connect to Database. Please ensure database exists and there are no currently open connections" | As Expected | Pass |  |
| 1.7 | Help | Clicking help opens the instruction manual | As Expected | Pass |  |

### Administrator Password Validation Test Cases (Sprint 2)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Administrator Password Validation** | | | | | |
| **Test Case ID** | **Condition** | **Expected Result** | **Actual Result** | **Pass/Fail** | **Comments** |
| 2.1 | Entering valid matching administrator username and password from database and click Login | Allows access to the maintenance section of the application | As Expected | Pass |  |
|  |  |  |  |  |  |

### Flight Booking Test Cases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Booking Flights** | | | | | |
| **Test Case ID** | **Condition** | **Expected Result** | **Actual Result** | **Pass/Fail** | **Comments** |
| 3.1 | Navigate to customer screen | Departure airport, destination airport should be selectable for each leg of the journey | As Expected | Pass |  |
| 3.2 | Navigate to customer screen | Seat type should be selectable for the flights | As Expected | Pass |  |
| 3.3 | Navigate to customer screen | Departure and destination airports should be displayed in alphabetical order. | As Expected | Pass |  |
| 3.4 | Navigate to customer screen | It should be possible to book a second leg, onward from the first destination or back to the original departure airport. | As Expected | Pass |  |
| 3.5 | Navigate to customer screen | There should be three choices of seat: Economy, Business & First Class | As Expected | Pass |  |
| 3.6 | Navigate to customer screen | Departure and destination airport cannot be the same. | As Expected | Pass |  |
| 3.7 | Navigate to customer screen, leave an enabled combo blank and click Book Flights | Error Message: “Please select all airport and seat type” | As Expected | Pass |  |
| 3.8 | Navigate to the customer screen, click clear | All combo boxes, check boxes and radio buttons should return to their default state. | As Expected | Pass |  |
| 3.9.1 | Navigate to customer screen, click Logout. | User should be returned to the Login Screen | As Expected | Pass |  |
| 3.9.2 |  | Password Field should be blank after logout | As Expected | Pass |  |
| 3.10 | Help | Clicking help opens the instruction manual | As Expected | Pass |  |

### Flight Confirmation Test Cases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Confirming Flights** | | | | | |
| **Test Case ID** | **Condition** | **Expected Result** | **Actual Result** | **Pass/Fail** | **Comments** |
| 4.1 | Select flights and seats, click Book Flight | The passenger name and booking reference should be recorded on the Confirmation Screen | As Expected | Pass |  |
| 4.2 | Select flights and seats, click Book Flight | Details of flight should be displayed for the user to confirm | As Expected | Pass |  |
| 4.3 | Navigate to confirm screen, click confirm | Details of the flight should be written to the database | As Expected | Pass |  |
| 4.4 | Navigate to confirm screen, click confirm | Clicking confirm disables the confirm button and enables print button | As Expected | Pass |  |
| 4.5 | Navigate to confirm screen, click print | Booking details should be printed to the default printer if one exists | As Expected | Pass |  |
| 4.5 | If the information cannot be written to the database | Display error message | As Expected | Pass |  |
| 4.6 | Help | Clicking help opens the instruction manual | As Expected | Pass |  |
| 4.7 | Exit confirm screen and re-enter | Confirm enabled, print disabled | As Expected | Pass |  |
|  |  |  |  |  |  |

Maintenance Suite Test Cases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Maintenance Suite** | | | | | |
| **Test Case ID** | **Condition** | **Expected Result** | **Actual Result** | **Pass/Fail** | **Comments** |
| 5.1 | Login to application with admin password, select airport from dropdown list, click remove. | Airport should be removed from this combo box, the combo boxes in the customer screen and the database. | As Expected | Pass |  |
| 5.2.1 | Navigate to maintenance suite, select three letter code for new airport and fill name into text box | If the three letter code is unique, airport should appear in the application. | As Expected | Pass |  |
| 5.2.2 |  | If three letter code is not unique, display error message. | As Expected | Pass |  |
| 5.3 | Navigate to maintenance suite, flight tab | All booked flights should be viewable | As Expected | Pass |  |
| 5.4.1 | Clicking Logout | Return to Login Screen | As Expected | Pass |  |
| 5.4.2 | On return to Login Screen | Password should be blank | As Expected | Pass |  |
| 5.6 | Set three character airport code, try and create an airport with a name over 50 characters | Error Message | As Expected | Pass |  |
| 5.7 | Set three character airport code, try and create an airport without a name | Error Message | As Expected | Pass |  |
| 5.8 | Help | Clicking help opens the instruction manual | As Expected | Pass |  |

## Page may or may not have been intentionally left blank

## Recommendations for improvements to the program

Below are a list of recommendations for the program, some of these have already been implemented.

1. Create User, Flight and Airport objects to use with the program (Complete)
2. Create a database to store the date in (Complete)
3. Expand login to encompass user as well as admin (Complete)
4. Prevent Admin from adding an airport twice (Complete)
5. Allow user to print the flight from the confirm screen (Complete)
6. Allow users to be added and removed through the GUI (Out of scope for this sprint)
7. Include a date timer picker for selection of dates for the Flights (Out of scope for this sprint)

## Documentation for use of the program by other programmers

Please consult the JavaDoc located in the doc folder in the main OOPAssignmentJava-master folder. You can access it by following this link. [..\doc\index.html](../doc/index.html)

## A program with onscreen help to guide users of your program

Click help button on any GUI screen to view the instruction manual.

# Bibliography

Holmevik, J. R. (1994). *Compiling SIMULA: A Historical Study of Technological Genesis*. Retrieved 03 01, 2016, from www.ntnu.no: http://www.idi.ntnu.no/grupper/su/publ/simula/holmevik-simula-ieeeannals94.pdf

Lewallen, R. (2005, 07 19). *4 Major Principles of Object Oriented Programming*. Retrieved from codebetter.com: http://codebetter.com/raymondlewallen/2005/07/19/4-major-principles-of-object-oriented-programming/