Object Oriented Programming

Philip Woulfe | Dublin

Report, Design and Testing document

Woulfe, Philip

2016

Contents

[Task One - Principles, Characteristics and Features of Object Oriented Programming 2](#_Toc445640803)

[Introduction 3](#_Toc445640804)

[Principles, Features & Characteristics 4](#_Toc445640805)

[Shared with Non Object Oriented Programming Languages 4](#_Toc445640806)

[Objects and Classes 4](#_Toc445640807)

[Inheritance 4](#_Toc445640808)

[Encapsulation 4](#_Toc445640809)

[Abstraction 5](#_Toc445640810)

[Polymorphism 5](#_Toc445640811)

[Conclusion 6](#_Toc445640812)

[Task Two - Airline Booking System Design Document 7](#_Toc445640813)

[Specification 8](#_Toc445640814)

[User Stories 9](#_Toc445640815)

[0001 – Customer Password Validation 9](#_Toc445640816)

[0002 – Administrator Password Validation 9](#_Toc445640817)

[0003 – Booking flights 9](#_Toc445640818)

[Objects 10](#_Toc445640819)

[MainWindow 10](#_Toc445640820)

[PasswordFrame 11](#_Toc445640821)

[CustomerScreen 12](#_Toc445640822)

[ConfirmScreen 13](#_Toc445640823)

[MaintenanceScreen 14](#_Toc445640824)

[Methods 15](#_Toc445640825)

[Databases 15](#_Toc445640826)

[Task Three – Source Code 16](#_Toc445640827)

[Task Four – Testing Report 17](#_Toc445640828)

[Bibliography 18](#_Toc445640829)

# 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 methods.

#### Variables

Variables are used to store information. This can be primitive variables such as natural numbers, characters, floating point numbers, and booleans or object variables such as strings, lists and even object versions of integers and floating point number.

#### 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.

### Objects and 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 be set to 4 and whenever the *Bark()* method is called, the dog will bark. However the dog cannot bark until it is created.

Objects are instances of a class. You may take the *Dog* class and instantiate it. Now you may call the *Bark()* method.

### 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.

### 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. 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.

**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 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 most mammals give birth to live young, whereas monotremes such as the platypus and 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 example, there is an interface of type *EggLayer,* both *Platypus* and *Bird* implement *EggLayer* meaning*Bird* is an *EggLayer* and *Platypus* is an *EggLayer.* Similarly, *Human* and *Platypus* both extend *Mammal,* therefor *Human* is a *Mammal* and *Platypus* is a *Mammal.* Since *Bird* doesn’t extend *Mammal, Bird* is not a *Mammal.* Since *Human* doesn’t implement *EggLayer*, *Human* is not an *EggLayer.*

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

# Task Two - Airline Booking System Design Document

## User Stories

### 0001 – Customer Password Validation (Not part of customer specification)

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 (Optional) |
| Criteria 2 | Cancel button exits application (Optional) |

#### Happy Case:

#### Error Validation:

|  |  |
| --- | --- |
| Criteria 1 | If an incorrect username or password is entered, display message: “Username or password incorrect” (optional) |
| Criteria 2 | If username field is left blank, display error “Username is a required field (Optional) |
| Criteria 3 | If password field is left blank, display error message “Password is a required field” (Optional) |

### 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.

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

#### Happy Case:

#### Error Validation:

|  |  |
| --- | --- |
| Criteria 1 | If an incorrect username or password is entered, display message: “Username or password incorrect” |

### 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 |
| Criteria 7 | Passenger name should be recorded and a Booking Reference assigned. |
| Criteria 8 | When completed, details of the flight(s) booked should be displayed on the screen for the user to confirm. |
| Criteria 9 | On confirmation, details should be written to a file. |
| Criteria 10 | Clicking Logout button should return user to Login Screen (Optional) |
| Criteria 11 | Clicking in Clear all will reset all fields (Optional) |
| Criteria 12 | Clicking Book Flights should open flight details in a new screen for confirmation (Optional) |

#### 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 (Optional) |

### 0004 – 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:

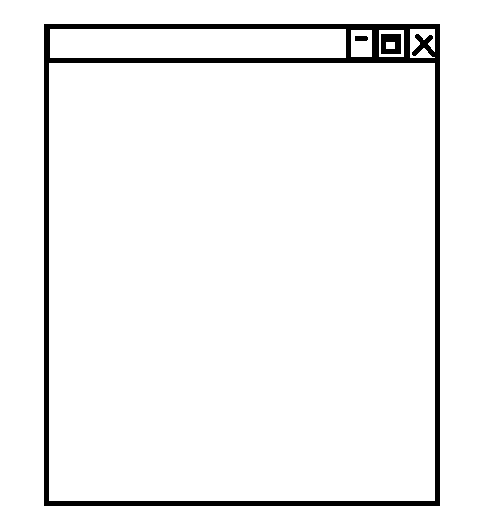
|  |  |
| --- | --- |
| Criteria 1 | Administrators should be able to add or remove airports from the application. |
| Criteria 2 | When an airport is removed, it should no longer appear in the application. |
| Criteria 3 | When an airport is added, it should appear in the application. |
| Criteria 4 | Administrators should be able to view booked flights from the maintenance suite. |
| Criteria 5 | Clicking logout button should return user to Login Screen |
| Criteria 6 | Administrator should be able to add new users (Optional) |

#### Error Validation:

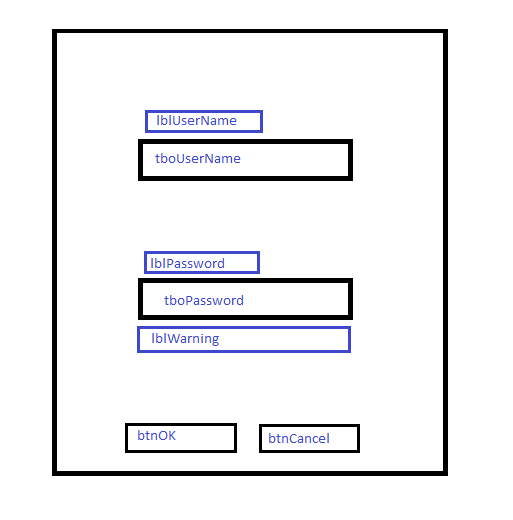
|  |  |
| --- | --- |
| Criteria 1 | Administrators should not be able to add the same airport twice. (Optional) |

## Objects

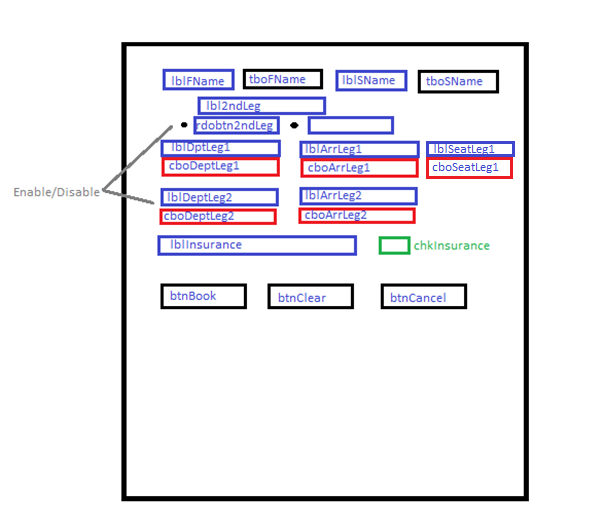
### MainWindow



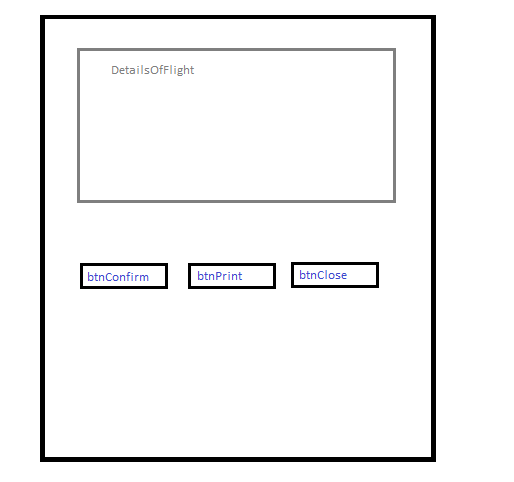
### PasswordFrame



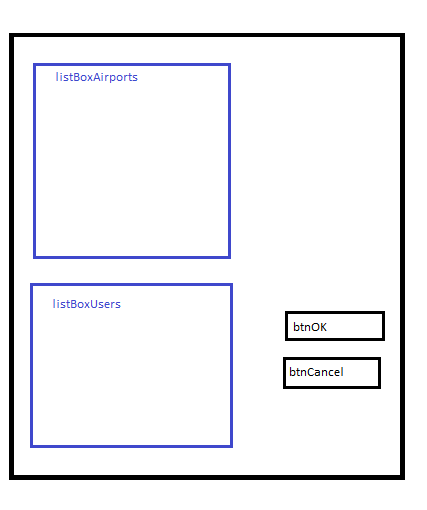
### CustomerScreen



### ConfirmScreen

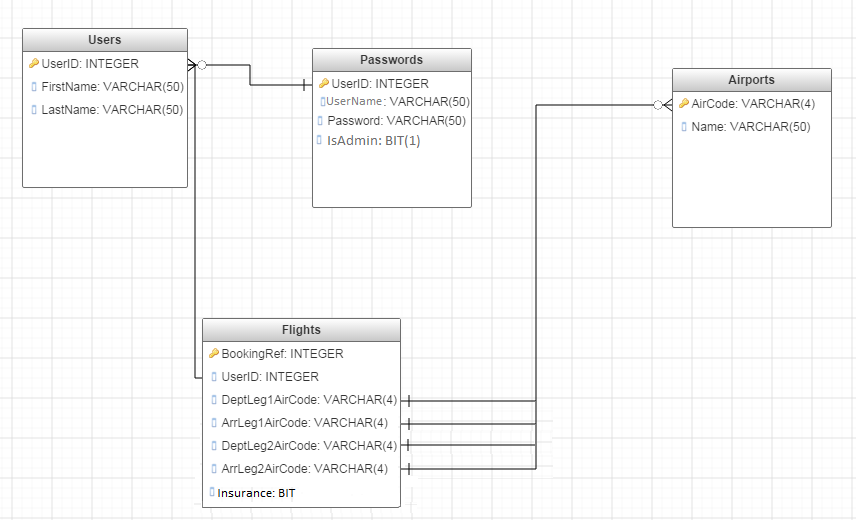


### MaintenanceScreen



## Methods

## Databases



# Task Three – Source Code

# Task Four – Testing Report

# 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/