**Assignment Report  
CZ20002 Object Oriented Design & Programming**

**Submitted by : Mavric Tan Soon Heng**

**Tan Junjie Alvin**

**Eugene Tan Ping Hong**

**Low Zi Qing**

**Lab Group : FSP4 Group 8**

**Date Submitted : : 15 April 2016  
Pages : 11 inclusive of Cover page and Annexes**

# Background

This report is submitted for CZ2002 Object-Oriented Design and Programming assignment.

This document is divided into the following sections. Section I will contain the class diagram of the entity classes and elaborate on the Object Oriented concepts as well as principle. Section II will contain the class diagram of the control classes and will discuss on the reasons for the implemented design. Section III will describes the class diagram of the boundary classes as well as the thoughts that had been placed into the design. Section IV will contain the class diagram of all classes (entity, control and boundary).

Next Section V will show the sequence diagram with notes. Lastly in Section VI, there will be a discussion on how further implementation can be added into the submitted assignment. The document will then be ended with a table of test cases that was written to test the application as well in the production of the video demonstration submitted together with this report.

The developed application is separated into 3 packages namely, *myEntities*, *myControl* and *myInterfaces*. The main class for the application is **MainView**.java, placed in *myInterfaces* package.

# Section I – Entity Classes

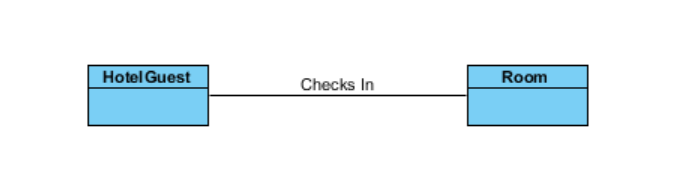
The application has 8 entity classes:

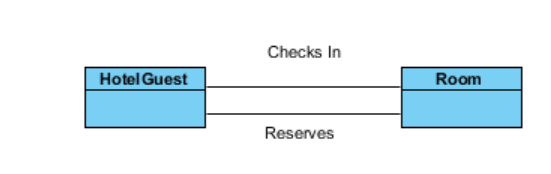
|  |  |
| --- | --- |
| * **Bill** | * **Reservation** |
| * **HotelGuest** | * **Room** |
| * **Guest** | * **RoomServiceItem** |
| * **CheckIn** | * **OrderLineItem** |

The relationship between all classes are shown in the class diagram attached on the next page. Some relationships are straightforward. For example, the relationship between **Bill** and **HotelGuest**, one bill is generated by each guest and each bill will cease to exist after the guest checkouts.

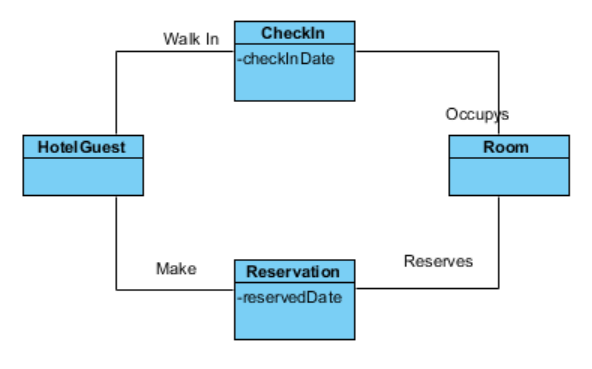
In the diagram, **HotelGuest** is implemented as a subclass of **Guest**. The consideration for this design is an application of the **Open-Close Principle** from **SOLID**. We closed the **Guest** class for modification but open up extension to the Guest class so that more codes can be added without major overhaul of the codes. This can serve as a base for other kind of guests that the hotel may serve in the future. An example will be *café guests* or *theme park guests* which have different requirements from hotel guests but are nonetheless possible guests that the hotel may serve. The above logic is also application for **OrderLineItem** and **RoomServiceItem**.

Next, the relationship between **HotelGuest** and **Room**. The initial idea for the relationship between the two classes is just a single association between them. A visual illustration is shown below



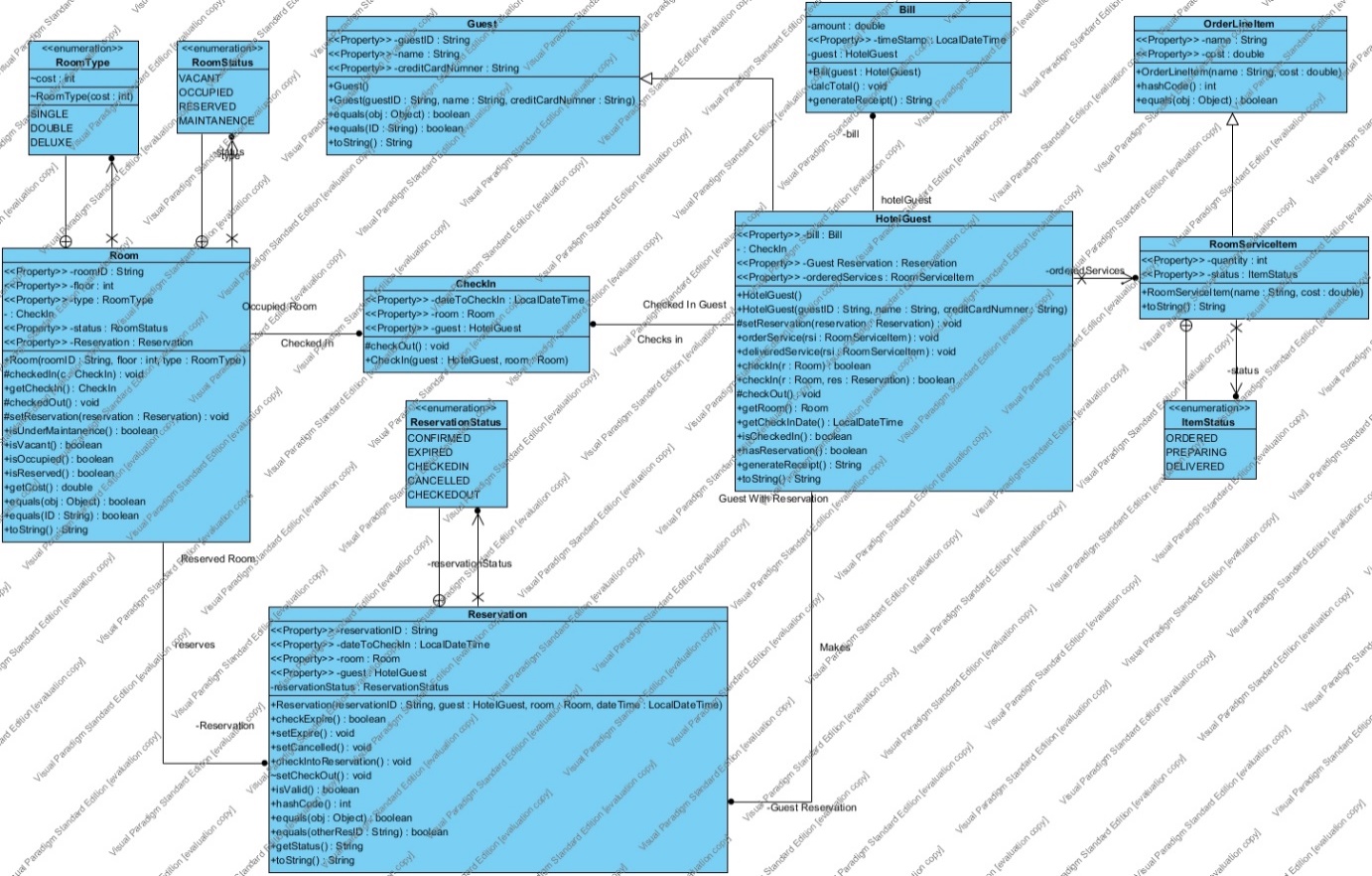
However, the above illustration is insufficient to meet the demands of the assignment. This is due to the requirement that a **HotelGuest** is allowed to walk in or reserves a **Room**. Hence, after revision, a new association is added between the two classes. 

Yet again, even with the above implementation, it is not enough to meet the requirement of the assignments as it is uncleared where the date to check in should be stored. Hence, after a third consideration, two associative classes are added to faciliate the relationship between room and HotelGuest.



With the above implementation, the date of check in or reservation can be stored, fulfilling the requirement for the assignment.

The final class diagram for the entity classes is shown below. A high resolution of this image (**entities.jpg**) is also submitted in the CD together with this report.



# Section II – Control Classes

This application has 4 control classes:

* **GuestManager**
* **RoomManager**
* **ReservationManager**
* **RoomServiceManager**

As the name of the control classes suggest, each manager manages the respective entity classes. The role of the manager classes are to ensure data integrity throughout the application in the form of validation as well as providing a linkage between the boundary classes and entities classes.

The control classes are implemented as static classes. The reason for that is because there will only be one instance for each manager and there will be no foreseeable circumstances in which the highly customised managers will be extended. Another reason for the static implementation is so that it will be easy to manage the references to each of the manager.

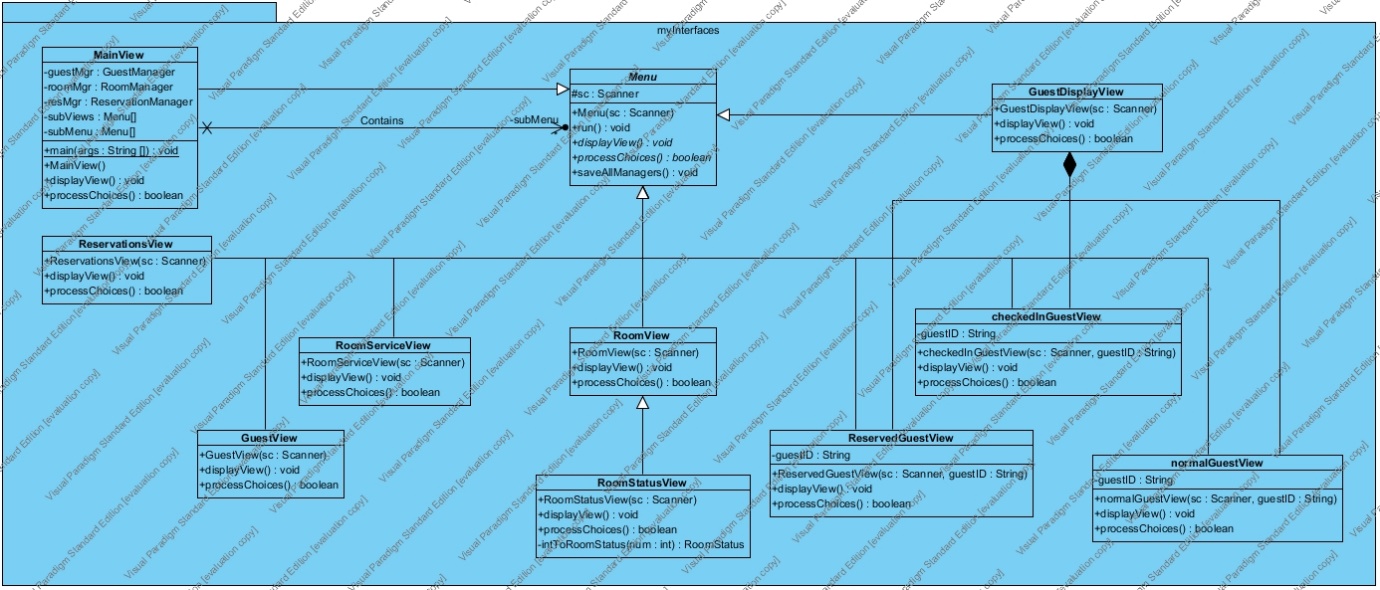
# Section III – Boundary Classes

The boundary classes of the applications are:

|  |  |
| --- | --- |
| * Menu {abstract} | * CheckInGuestView |
| * GuestView | * NormalGuestView |
| * RoomView | * ReservedGuestView |
| * RoomStatusView | * GuestDisplayView |
| * ReservationView | * MainView ( main class of the application ) |
| * RoomServiceView |

There are a total of 11 boundary classes. The reason for having many boundary classes is so that there will be a tidy way of managing the options presented to the user.

The diagram below shows the relationships between the 11 classes. (myBoundaries.jpg



All boundary classes has the abstract **Menu** class as the superclass. The **Menu** class has a method ***public final void run().*** This method enforces that all subclasses of **Menu** will have to display their options first and continue to process the input from the user until the users input “0” to explicitly escape from that menu.

This also method allows us to avoid the use of heavily nested cases statements in an overly complicated fashion inside a large java file.

The flow boundary classes can be represented with a tree structure.

**MainView**

* **GuestView**
* **RoomView**
  + **RoomStatusView**
* **ReservationView**
* **GuestDisplayView**

Some sort of access control is implemented by checking on the guest status.

This allows us to hide options that does not make sense for example, a guest who has yet to check in should not be able to see the checkout option.

* + [if guest is checked in]

**CheckinGuestView**

* + [if guest is not checked in but has reservation]

**ReservedGuestView**

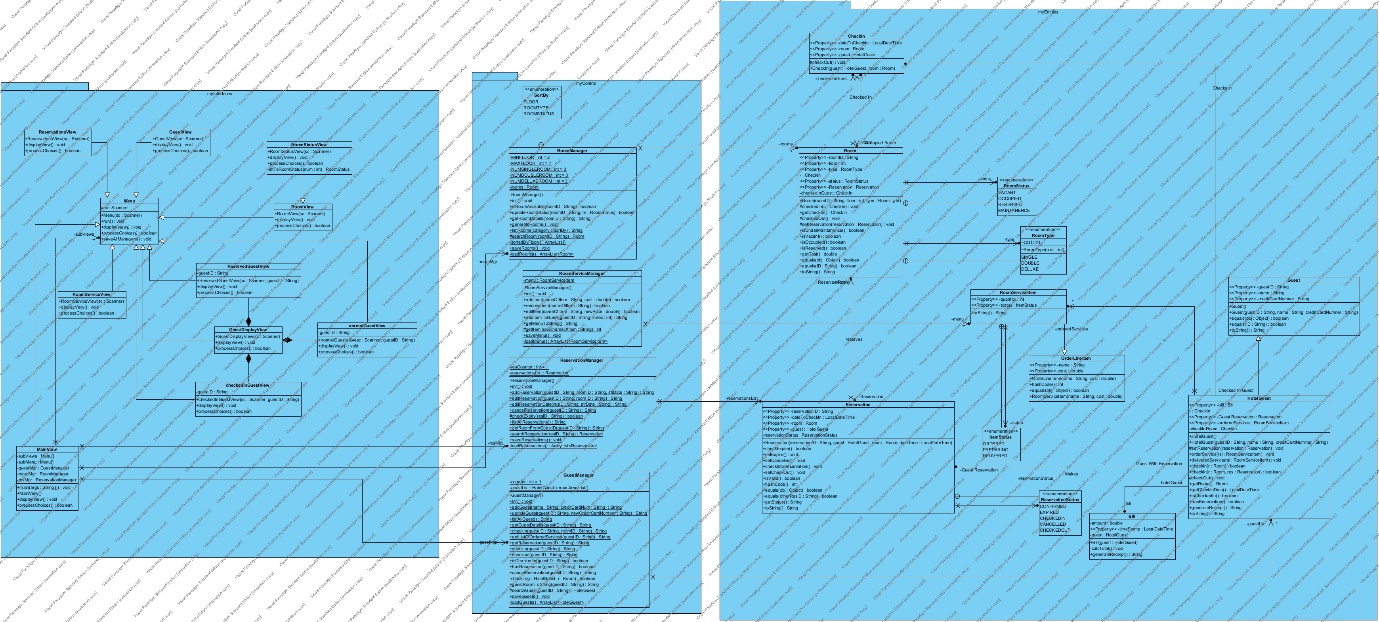
* + [if guest is neither check in and no reservation]

**NormalGuestView**

* **RoomServiceView**

# Section IV – Overview of all Classes

The following diagram below shows the relationship between all classes defined within the application. (HD version saved as ApplicationClassDiagram.jpg).



Left to right :

package of boundary classes, package of control classes, package of entity classes

# Section V – Sequence Diagram

# Section VI – Future implementations

The existing application allows for additional modification with minimal changes.

One of which will be the ability to add more kind of guests other than HotelGuest, guests like café guests and theme park guests can be added and extend the codes from Guests.

Another possible change will be to add even more submenus to the application. Because all submenus are derived class from the Menu class, it will not be difficult to add more submenus and expect the additional submenus to behave in the same manner as the exisiting submenu.