Department of Computer Engineering

Bilkent University

Object-Oriented Software Engineering

CS 319 Project: Hotel Master

Final Report

Fall 2015 / Group 10

Mesut Gürlek 21201157

Uğurcan Aytar 21200487

Deniz Salucu 21001433

Murat Can Akın 20601101

Table of Contents

Table of Figures

1. INTRODUCTION

In service sector, providing quality service to customers, implementing their requests efficiently, earning their satisfaction are most important responsibilities. Hotel management is one of the important service sector jobs. Hotel staffs play great role in providing qualified service to their customers. Staffs can implement their jobs effectively on the benefit of customers by using functional hotel management program.

The aim of this system is to contribute the hotel staffs to do their tasks effectively and by this way customers can get quick service. For example, when customers come to hotel, they do not waste time with filling the forms, they can easily do check-in and check-out process or they can do reservation by calling the hotel. Time is a valuable source in people’s daily life. Because people live their days intensively and so they want to use it efficiently and in this case, people especially who travelling and using hotels a lot will not spend so much time when they arrive or leave the hotel. This program will increase the productivity of the hotels. They will give qualified service to customers and then gain their satisfaction. The hotel can have the customer list and room list in the certain system and do specific exercises according to the requests easily.

Our report consists of two parts. In Requirement Analysis part, we will indicate functional and non-functional requirements and also constraints. Then we will define different scenarios concerning about the system to show its functionality and support them with use case diagrams which describing the behavior of the system from an actor’s point of view and we will indicate user interfaces. In analysis part, we will construct a class diagrams describing the structure of the system in terms of classes and objects. We will define state charts and also each scenario will be represented by sequence diagrams.

2. REQUIREMENT ANALYSIS

2.1. Overview

The purpose of “Hotel Master” program is to ease the hotel staffs’ works and present a good service to hotel customers. Hotel staff is the main actor of the system. In this program we mainly focus on admin side and try to create functional features to use the program effectively.

Firstly, to begin using the system, hotel staff enters name and password and then main menu screen is seen. Hotel staff select an action which he/she wants to do. The main functions are managing check-in, check-out and reservation process.

In check-in and reservation, hotel staff takes customer’s name, surname, phone number, arrival-departure time and calculates the total cost according to guest type and number, and room type. In reservation, hotel staff again takes same info and if customer is certain about staying at hotel, staff confirms the changes and this register will transfer to check-in section.

Hotel staff can see the all customers list and their specific information, also see the situation of rooms such as empty or reserved, its type, number of people staying according to the room number. Hotel staff can delete room from the room list and add a new room by entering its features such as room and floor number, its price, number of adult or child number it can take.

2.2 Functional Requirements

In this program, we mainly focus on staff’s role and he/she use different interfaces according to the task.

2.2.1. Functional Requirements for Hotel Staff

Start to Manage

- ➢ Hotel staff should be able to start managing hotel facilities.

Log In

- ➢ Hotel staff should be able to login to the system.
- ➢ Hotel staff must enter the username and password to login to the system.

Main Menu

- ➢ Hotel staff should be able to select an activity which he wants to do such as “Check-in”, “Customer List”, “Room List”, “Reservations”, “Room Editor”, “Logout!”.

Check-in

- ➢ Hotel staff should be able to enter customer’s name, surname and phone number.
- ➢ Hotel staff can select arrival time and departure time.
- ➢ Hotel staff should choose payment method which are credit card or cash.
- ➢ Hotel staff should be able to select guest type, room type and calculate total cost.
- ➢ Hotel staff can go back to the main menu or finish check-in process.

Customer List

- ➢ Hotel staff should be able to see the customer list and reach customers’ information such as name, phone number, purchase type, room number, arrival-departure time and total cost.
- ➢ Hotel staff can find customer by typing his/her name.
- ➢ Hotel staff should be able to check-out customer and confirm changes.
- ➢ Hotel staff can go back to main menu.

Reservation List

- Hotel staff should be able to do reservation for customers who are calling by telephone.
- Hotel staff should enter customer's name, surname and phone number and then confirm reservation.
- Hotel staff can go to the check-in section if customers confirm that they stay at hotel certainly.
- Hotel staff should be able to go back to main menu.

Room List

- Hotel staff should be able to see the specific info about the unique room such as room type, status, floor, number of adult and child and daily price.
- Hotel staff should delete rooms for under reconditioning.
- Hotel staff should be able to add a new room to the current list by entering its features.
- Hotel staff can go back to main menu.

2.3. Non-Functional Requirements

2.3.1. Performance

The system should work with high performance because the hotel staff should give effective and quick service to implement customers' tasks.

2.3.2. Supportability

The system should be maintainable easily to increase its functionalities over the time.

It can be adapted to any hotels with appropriate changes in the interface.

2.3.3. Usability

The program should have user friendly interface. Hotel staffs should be able to learn how to use it efficiently and then they should be comfortable while using the program.

2.4. Constraints

The implementation language will be Java.

Intellij IDEA will be used as development environment.

The system will be a desktop application.

2.5. Scenarios

Scenario Name: Create Room

Scenario: Ali is the one of the hotel staff. He wants to create new Room for the hotel. He passes to the login page. He enters username and the password to login the HotelMaster. After login, Ali sees the all menu options on the screen. He chooses the Room Editor item on the screen and opens the Create New Room panel. Ali selects room type, max adult and child guest number, daily price of the room and enters some extra information for the room. Then, he confirms the room information and creates a new room. Then he goes back to the main menu and log outs from the HotelMaster.

Scenario Name: Check in New Customer

Scenario: Ali is one of the hotel staff. He wants to check in new customer. He is already on the main menu. Ali sees the all menu options on the screen. He chooses the Check in from menu and opens new customer form. He enters the name and surname of the customer. Then, he chooses the check in and check out date from the DatePickers. After that, Ali selects an empty room which is suitable to customer's request from the interface. Then, He clicks the 'calculate' button and cost will be appear on the screen. Lastly, Ali selects the payment type of the customer and confirms the check in.

Scenario Name: Make a Reservation

Scenario: Ali is one of the hotel staff. He wants to make reservation through a customer call. He is already on the main menu of HotelMaster. Ali sees the all menu options on the screen. Firstly, he chooses the room list and selects an available room according to customer's criteria. Then he goes back to menu. After that, he chooses the Reservation option from menu and opens the New Reservation form. He enters the name and surname of the customer. Then, he enters the room number which he selected before in room list. After, he picks reservation date interval by using DatePicker. Then, Ali enters the customer phone number. Lastly, Ali confirms the reservation and reservation appears below list in reservation menu.

Scenario Name: View Customer and Room Lists

Scenario: Ali is one of the hotel staff. He wants to check current status of rooms and current customers. He is already on the main menu of HotelMaster. Ali sees the all menu options on the screen. He chooses the 'Room List' from menu and opens the room list screen. He observes the empty and reserved rooms in the list. Then he clicks to 'Back' button and goes main menu. After that, he chooses the 'Customer List' and opens customer list screen. He observes the current customers with their rooms and their enter-exit dates. Then he goes back to main menu.

Scenario Name: Delete Room

Scenario: Ali is one of the hotel staff. Employees are reconditioning one hotel room so Ali have to delete this room from room list for now. He is on the main menu of the HotelMaster. He chooses the 'Room List' from menu and opens the room list screen. He finds that specific room and mark its delete room checkbox. Then, he clicks the 'Confirm Changes' button and removes room from the list. Then he goes back to main menu.

Scenario Name: Check-Out Customer

Scenario: Ali is the one of the hotel staff. He is on the main menu of HotelMaster. He wants to check out for a customer. He chooses the 'Customer List' from main menu and opens the customer list screen. Then he marks the 'Check-out Customer' checkbox of customer from the list. Then, he clicks the 'Confirm Changes' button and removes the customer from list. After that he goes back to main menu.

2.6. Use Case Models

Use case modeling shows the connection between user and objects.

2.6.1. Use Case Diagram

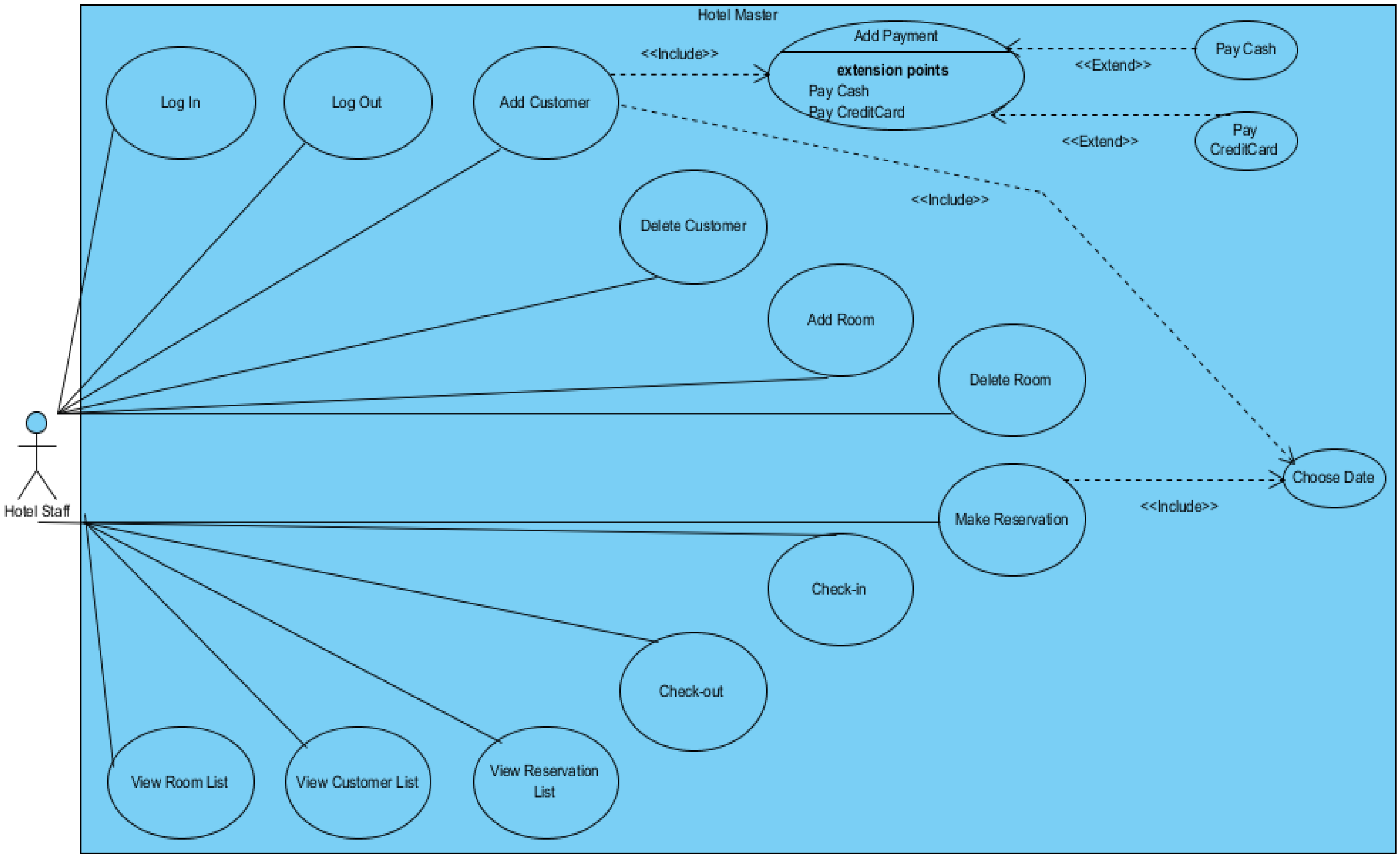


Figure 1: Use Case Diagram for Hotel Master

2.6.2 Textual Model

Use Case 1

Use Case Name: Login

Participating Actor: Hotel Staff

Entry Condition: Welcome page

Main Flow of Events:

- Hotel Staff starts the application and press the “Manage” button.
- Hotel Master opens the login page.
- Hotel management gives special username and password the hotel staffs.
- Staff writes his/her username.
- Staff writes his/her password.
- Staff selects the date.
- Staff pushes the “Log in!” button.
- Hotel Master directs the staff to “Main Menu” page.

Exit Condition: Main Menu page.

Use Case 2

Use Case Name: Logout

Participating Actor: Hotel Staff

Entry Condition: Main Menu page.

Main Flow of Events:

- Staff pushes the “Logout!” button.

Exit Condition: Login page.

Use Case 3

Use Case Name: Add Customer

Participating Actor: Hotel Staff

Entry Condition: Main Menu page

Main Flow of Events:

- Hotel staff presses the “Check-in” icon in “Main Menu” page.
- Hotel Master opens “Check-in” page.
- Hotel staff writes the customer’s name.
- Hotel staff writes the customer’s surname.
- Hotel staff writes the customer’s phone number.
- Hotel staff selects the customer’s arrival time from the calendar.
- Hotel staff selects the customer’s departure time from the calendar.
- Hotel staff selects the customer’s payment method.
- Hotel staff chooses the customer’s room type.
- Hotel staff pushes “Process Check-in” button.
- Hotel Master adds a new customer in the customer list.

Exit Condition: Main Menu page

Use Case 4

Use Case Name: Delete Customer

Participating Actor: Hotel Staff

Entry Condition: Main Menu page

Main Flow of Events:

- Hotel Staff pushes the “Customer List” icon in “Main Menu” page.
- Hotel Master opens the “Customer List” page.
- Staff uses search area to find the customer by writing customer’s name.
- Staff selects the check box the customer who wants the check-out from the hotel under the “Check-out Customer” column.
- Staff pushes the “Confirm Changes” button.

- Hotel Master deletes the customer from the system and customer list.

Exit Condition: Main Menu page

Use Case 5

Use Case Name: Add Room

Participating Actor: Hotel Staff

Entry Condition: Main Menu page

Main Flow of Events:

- Hotel Staff presses the “Room Editor” icon in “Main Menu” page.
- Hotel Master opens the “Room Editor” page.
- Staff selects the room type from select box.
- Staff enters room number.
- Staff enters floor number.
- Staff enters daily price of room in accordance with room type.
- Staff enters number of adult people staying at room.
- Staff enters number of children staying at room.
- Staff writes extra information if it needs.
- Staff pushes the “Confirm” button.
- Hotel Master adds a new room in the system and room list.

Exit Condition: Main Menu page

Use Case 6

Use Case Name: Delete Room

Participating Actor: Hotel Staff

Entry Condition: Main Menu page

Main Flow of Events:

- Hotel Staff presses the “Room List” icon in “Main Menu” page.
- Hotel Master opens the “Room List” page.
- Staff writes room number in the search area to find room easily.

- Staff selects the room under the "Delete Room" column.
- Staff pushes the "Confirm Changes" button.
- Hotel Master deletes the room from room list and the system.

Exit Condition: Main Menu page

Use Case 7

Use Case Name: View Room List

Participating Actor: Hotel Staff

Entry Condition: Main Menu page

Main Flow of Events:

- Hotel Staff presses the "Room List" icon in "Main Menu" page.
- Hotel Master opens the "Room List" page.
- Staff displays rooms' numbers, rooms' types, rooms' status, room's floor, rooms' capacities and rooms' prices.
- Staff presses the "Back" button.
- Hotel Master opens "Main Menu" page.

Exit Condition: Main Menu page.

Use Case 8

Use Case Name: View Customer List

Participating Actor: Hotel Staff

Entry Condition: Main Menu page

Main Flow of Events:

- Hotel Staff presses the "Customer List" icon in "Main Menu" page.
- Hotel Master opens the "Customer List" page.
- Staff uses search area to find the customer by writing customer's name.
- Staff displays customers' names, phone numbers, purchase types, room numbers, arrival-departure types and total costs.
- Staff presses the "Back" button.

- Hotel Master opens “Main Menu” page.

Exit Condition: Main Menu page.

Use Case 9

Use Case Name: View Reservation List

Participating Actor: Hotel Staff

Entry Condition: Main Menu page

Main Flow of Events:

- Hotel Staff presses the “Reservation” icon in “Main Menu” page.
- Hotel Master opens the “Reservations List” page.
- Staff displays customers’ names, phone numbers, room numbers, arrival-departure types and total costs.
- Staff presses the “Back” button.
- Hotel Master opens “Main Menu” page.

Exit Condition: Main Menu page.

Use Case 10

Use Case Name: Make Reservation

Participating Actor: Hotel Staff

Entry Condition: Main Menu page

Main Flow of Events:

- Hotel Staff presses the “Reservation” icon in “Main Menu” page.
- Hotel Master opens the “Reservations List” page.
- Staff enters customer’s name.
- Staff enters customer’s surname.
- Staff enters the customer’s phone number.
- Staff chooses the customer’s arrival time from the calendar.
- Staff chooses the customer’s departure time from the calendar.
- Staff enters customer’s room number.

- Staff presses the “Confirm Reservation” button.
- Hotel Master adds the reservation to reservation list above the screen.

Exit Condition: Main Menu page

Use Case 11

Use Case Name: Check-in

Participating Actor: Hotel Staff

Entry Condition: Main Menu page

Main Flow of Events:

- Hotel Staff presses the “Check-in” icon in “Main Menu” page.
- Hotel Master opens the “Check-in” page.
- Hotel staff writes the customer’s name.
- Hotel staff writes the customer’s surname.
- Hotel staff writes the customer’s phone number.
- Hotel staff chooses the customer’s arrival time from the calendar.
- Hotel staff chooses the customer’s departure time from the calendar.
- Hotel staff selects the customer’s payment method (cash or credit card).
- Hotel staff chooses the customer’s room type.
- Staff pushes “Calculate” button.
- Hotel Master calculates total cost.
- Hotel staff presses “Process Check-in” button.

Exit Condition: Main Menu page

If customer makes reservation before:

Entry Condition: Main Menu page

Alternative Flow of Events:

- Hotel Staff presses the “Reservation” icon in “Main Menu” page.
- Hotel Master opens “Reservation List” page.
- Staff writes customer’s name in search area to find the customer who makes reservation before.

- Staff pushes “Go” button under the “Check-in Reservation” column.

Exit Condition: Main Menu page

Use Case 12

Use Case Name: Check-out

Participating Actor: Hotel Staff

Entry Condition: Main Menu page

Main Flow of Events:

- Hotel Staff pushes the “Customer List” icon in “Main Menu” page.
- Staff uses search area to find the customer by writing customer’s name.
- Staff selects from the check box the customer who wants the check-out from the hotel under the “Check-out Customer” column.
- Staff pushes the “Confirm Changes” button.

Exit Condition: Main Menu page

2.7. User Interface

2.7.1. Welcome Page Screen

Welcome page is the system’s initial screen that when staff initializes the system, he/she will se this screen which is below. Welcome page includes the symbol of hotel and the name of the system. To start managing hotel facilities, there is a button below in the screen which is called “Manage!”.

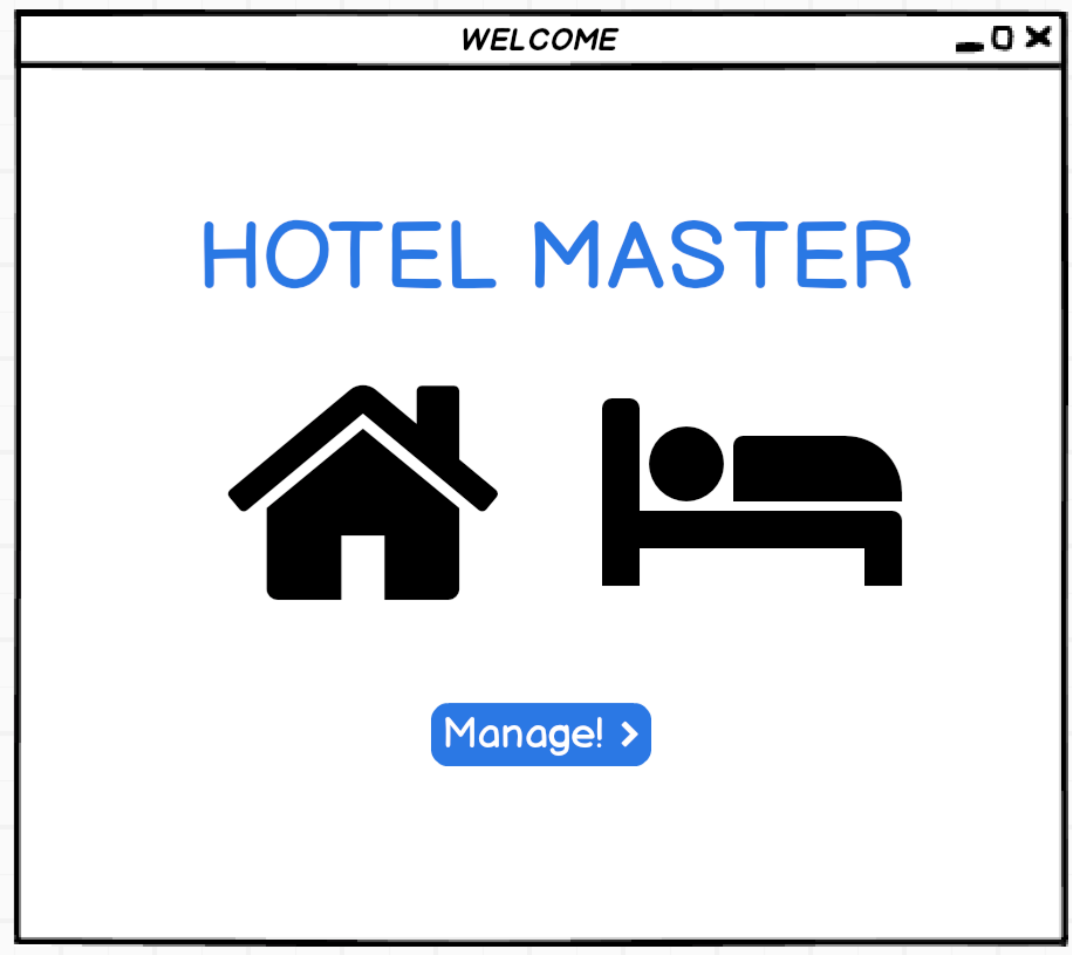


Figure 2: Welcome Page Mockup

2.7.2. Login Page Screen

Staff reaches and sees Login Page after tapping “Manage!” button from Welcome Page. Staff must enter “Staff Username” and his/her “Password” and enter “Log in!” button to login to the main system. Also, login page includes a calendar which staff can see the recent date. Login page screen is below.

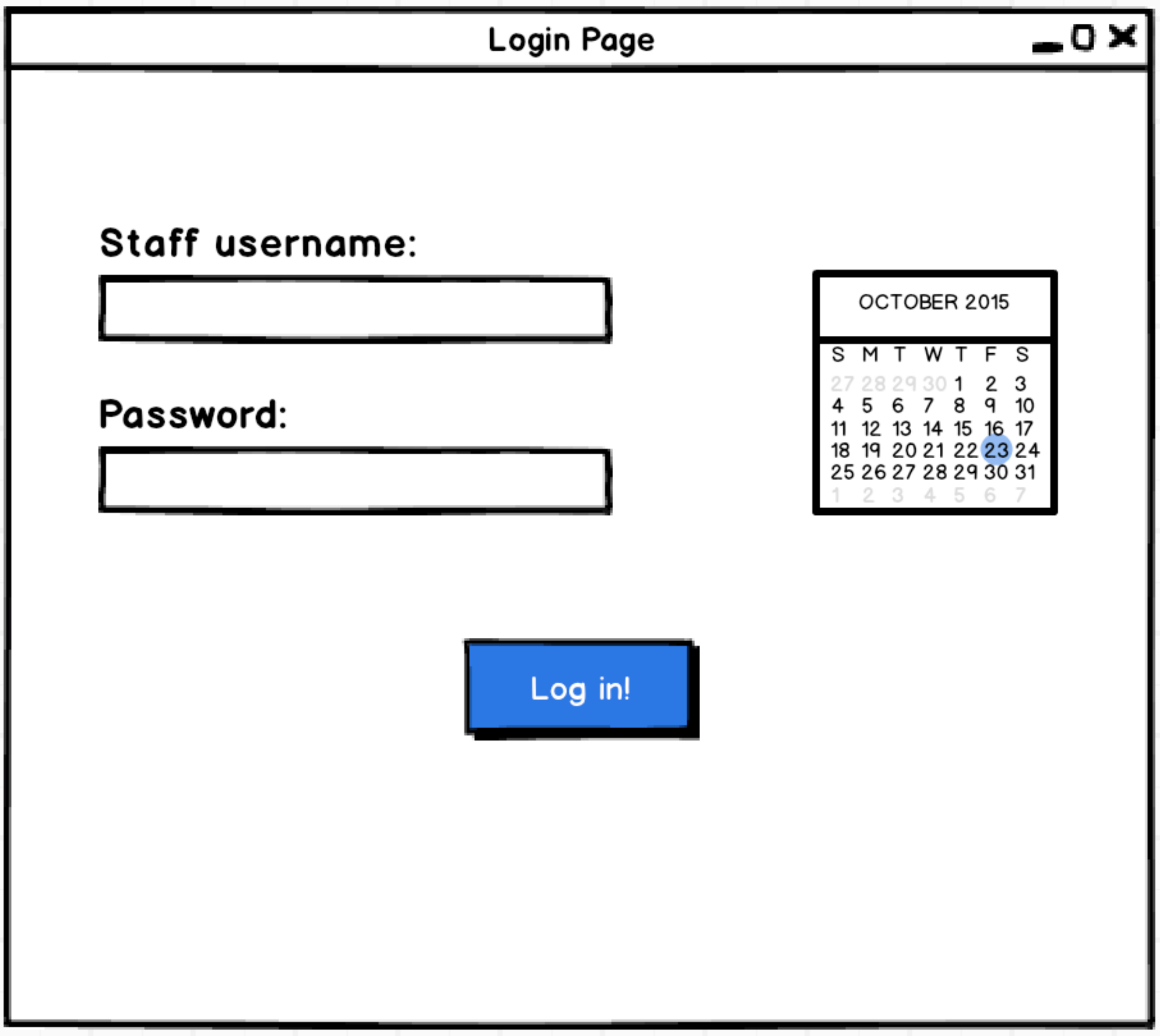


Figure 3: Login Page Mockup

2.7.3. Main Menu Page

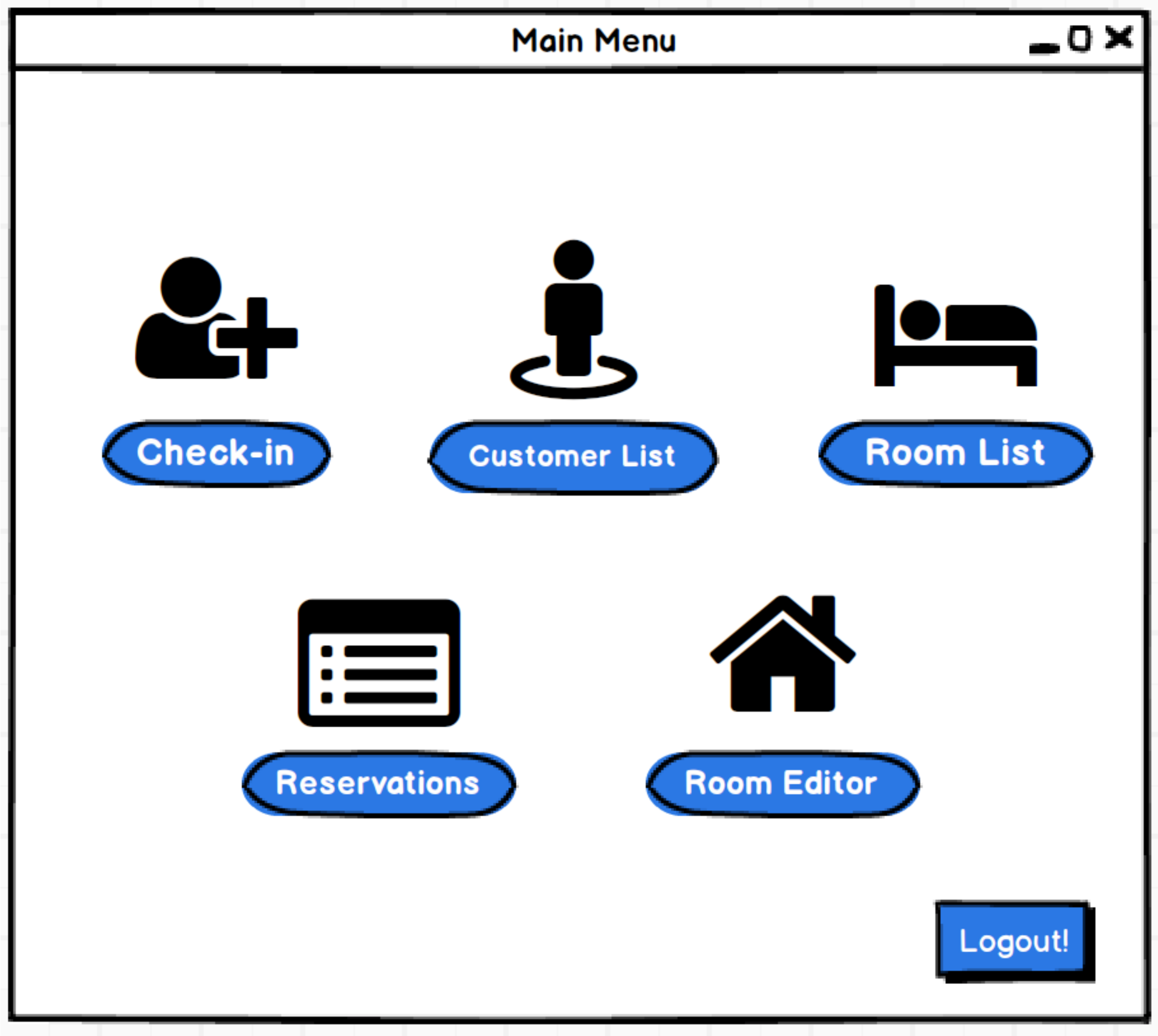
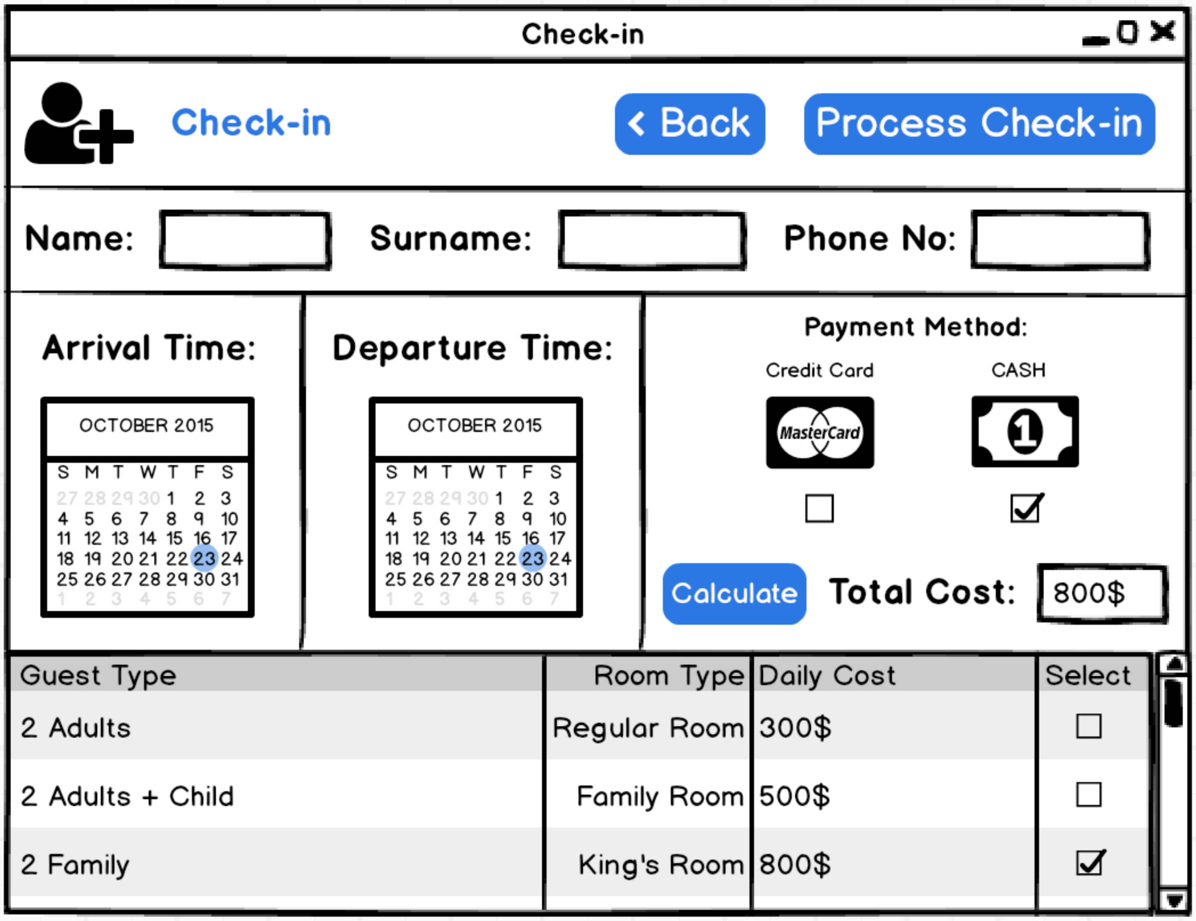


Figure 4: Main Menu Mockup

After staff logins the system, he/she will see the “Main Menu” which is main panel of the system. In this menu, staff can reach 5 different sub-systems like “Check-in”, “Customer List”, “Room List”, “Reservations”, “Room Editor”. Staff can reach these sections by pushing buttons below the image representations of each part of system. In addition to these, staff can use “Logout” button to logout from system.

2.7.4. Check-in Screen

Staff can enter Check-in section by pressing Check-in’s button. In Check-in sub-system, staff can do check-in processes of customers who will join his/her hotel definitely. In this section, staff must enter customer’s name, surname, and phone number by filling empty boxes. To continue with, by selecting dates from calendar, staff can choose arrival and departure time of customer, too. Also, from payment method, staff can select the intended payment method of his/her customer’s by clicking checkboxes which are below the representation of “Credit Card” and “Cash” types. Also, at the below of this screen, staff must select the type of the room that his/her customer wanted. In order to continue, by selecting arrival and departure time, type of the room and filling name, surname and phone number part, by “Calculate” button, staff can see “Total Cost” which will customer pay to stay at hotel. After staff finishes filling parts to register customer to hotel, by clicking “Process Check-in” button, he/she will finish the payment and registration to the hotel and customer will at “Customer List”. After pressing “Process Check-in” button, if staff finishes his/her job in Check-in part, staff can use “Back” button to turn back to main menu.



Name: Surname: Phone No:

Payment Method:

Credit Card □ CASH ☑

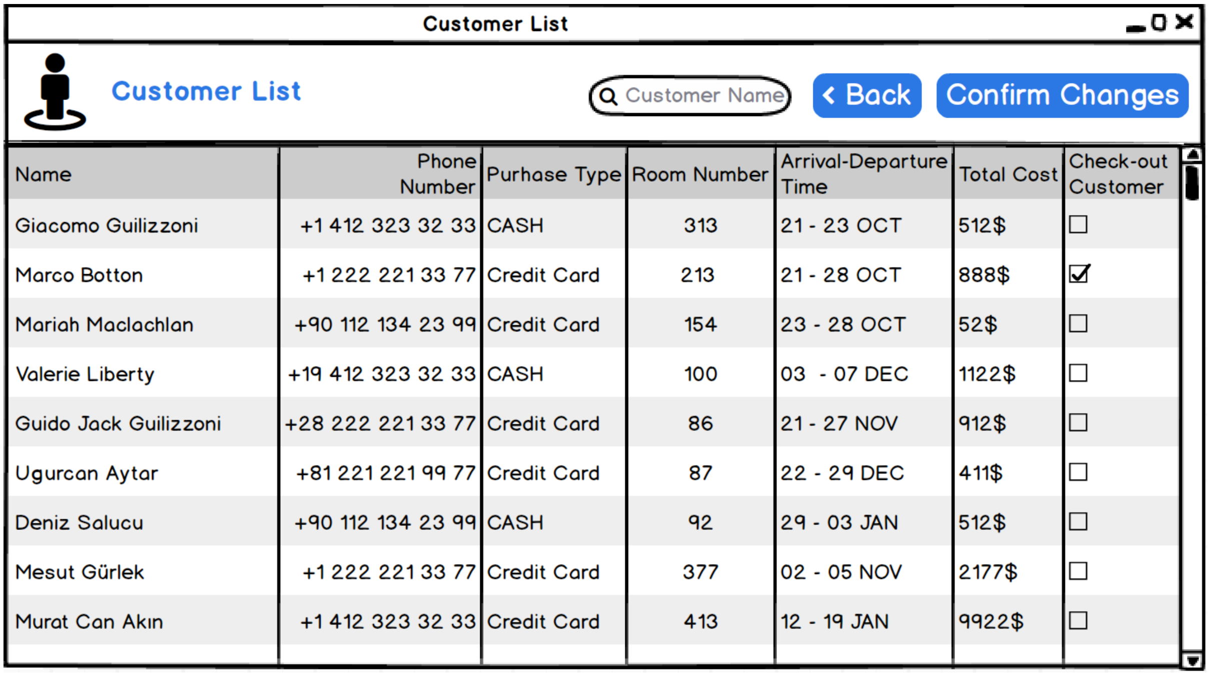
Total Cost: 800$

Guest Type	Room Type	Daily Cost	Select
2 Adults	Regular Room	300$	□
2 Adults + Child	Family Room	500$	□
2 Family	King's Room	800$	☑

Figure 5: Check-in Screen Mockup

2.7.5. Customer List Screen

From main menu, staff can enter “Customer List” sub-system from pressing button below the visual representation of Customer List section. In customer list section, staff can reach the customers that have been registered in the hotel. Staff can reach the information which are from check-in section like name, surname, phone number, purchase type, room number, arrival and departure time, total cost of living in a list. To continue with, there is a search bar at top of the section that staff can search customers from their names. In addition to these, there is a "Check-out Customer" section that staff can check-out the customers of his/hers. After clicking the check box of Check-out Customer and pressing "Confirm Changes" button, customers that their "Check-out Customer" checkboxes are pressed will be checked-out. At last, by "Back" button, staff can return to main menu page.

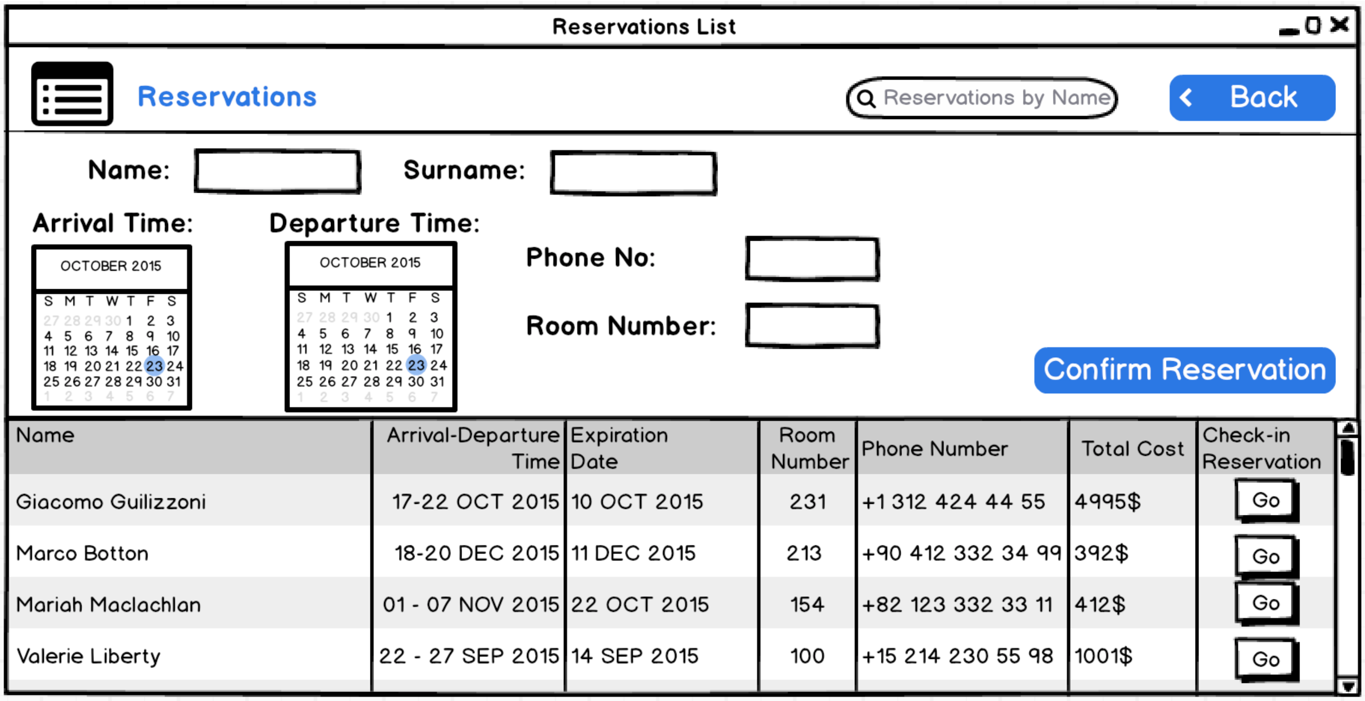


Name	Phone Number	Purhase Type	Room Number	Arrival-Departure Time	Total Cost	Check-out Customer
Giacomo Guilizzoni	+1 412 323 32 33	CASH	313	21 - 23 OCT	512$	□
Marco Botton	+1 222 221 33 77	Credit Card	213	21 - 28 OCT	888$	☑
Mariah Maclachlan	+90 112 134 23 99	Credit Card	154	23 - 28 OCT	52$	□
Valerie Liberty	+19 412 323 32 33	CASH	100	03 - 07 DEC	1122$	□
Guido Jack Guilizzoni	+28 222 221 33 77	Credit Card	86	21 - 27 NOV	912$	□
Ugurcan Aytar	+81 221 221 99 77	Credit Card	87	22 - 29 DEC	411$	□
Deniz Salucu	+90 112 134 23 99	CASH	92	29 - 03 JAN	512$	□
Mesut Gürlek	+1 222 221 33 77	Credit Card	377	02 - 05 NOV	2177$	□
Murat Can Akın	+1 412 323 32 33	Credit Card	413	12 - 19 JAN	9922$	□

Figure 6: Customer List Screen Mockup

2.7.6. Reservations Screen

Staff can select “Reservations” section by clicking button of its from main menu. In Reservations section, staff can add new reservation to the system. By filling name, surname, selecting arrival and departure time and also filling phone number and room number of customer from customer’s room wanting type. If staff press “Confirm Reservation” button, that reservation will be in queue at the below of the menu, which is reservation table. Also, at the top of the page, there is a search box that staff can search customers which have done reservations by name. In addition to these, if customers confirm that he/she will visit staff’s hotel precisely, staff can press “Go” button which is below of the section called “Check-in Reservation”. By clicking “Go” button, that customer will transfer to the “Check-in” section of the “Hotel Master” system.

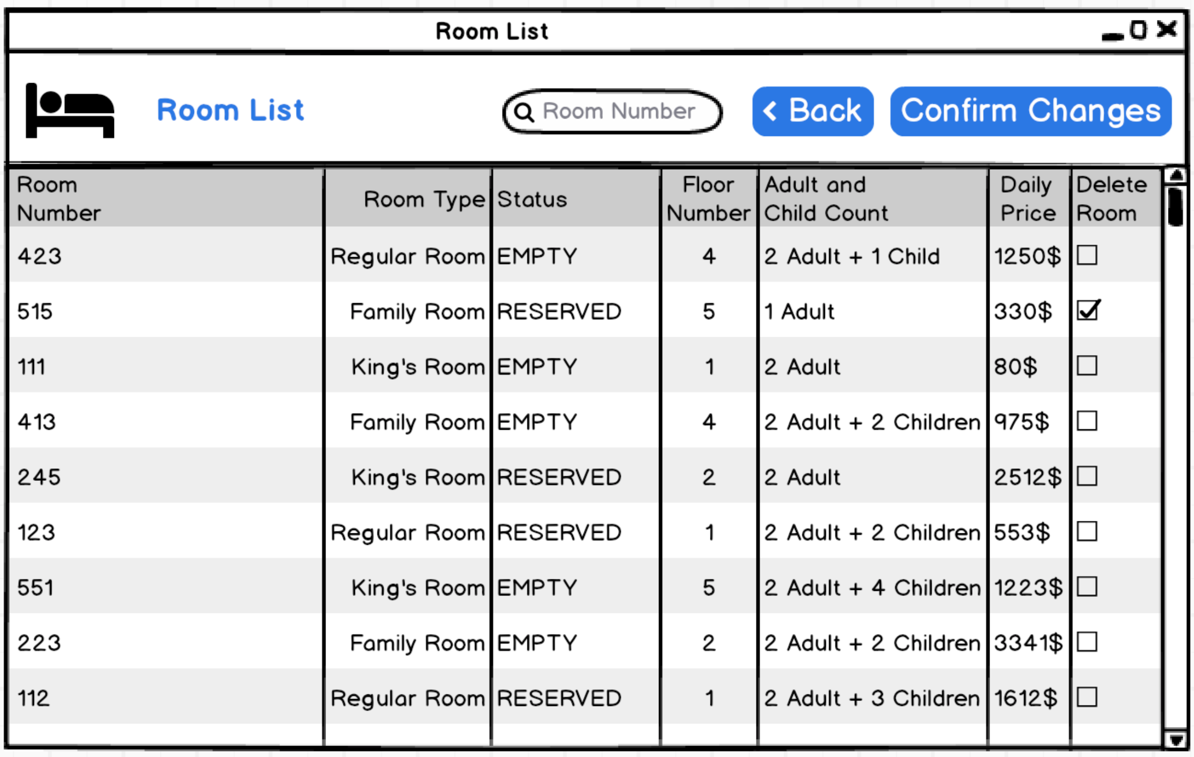


Name	Arrival-Departure Time	Expiration Date	Room Number	Phone Number	Total Cost	Check-in Reservation
Giacomo Guilizzoni	17-22 OCT 2015	10 OCT 2015	231	+1 312 424 44 55	4995$	Go
Marco Botton	18-20 DEC 2015	11 DEC 2015	213	+90 412 332 34 99	392$	Go
Mariah Maclachlan	01 - 07 NOV 2015	22 OCT 2015	154	+82 123 332 33 11	412$	Go
Valerie Liberty	22 - 27 SEP 2015	14 SEP 2015	100	+15 214 230 55 98	1001$	Go

Figure 7: Reservations Screen Mockup

2.7.7. Room List Screen

“Room List” section is another section of the system that staff can reach. From main menu, staff clicks Room List button and at room list part, staff can see the rooms which are empty. Also, staff can see reserved rooms of the hotel, too. In room list section, staff can reach rooms by entering numbers at search bar to see the status of that room. In Room List, staff can find specific information about certain room which are room type, status, floor number, adult and child count, daily price. To continue, staff can delete room by clicking checkboxes which are below “Delete Room” section. If staff select the check box and press “Confirm changes”, that specific room deletes.

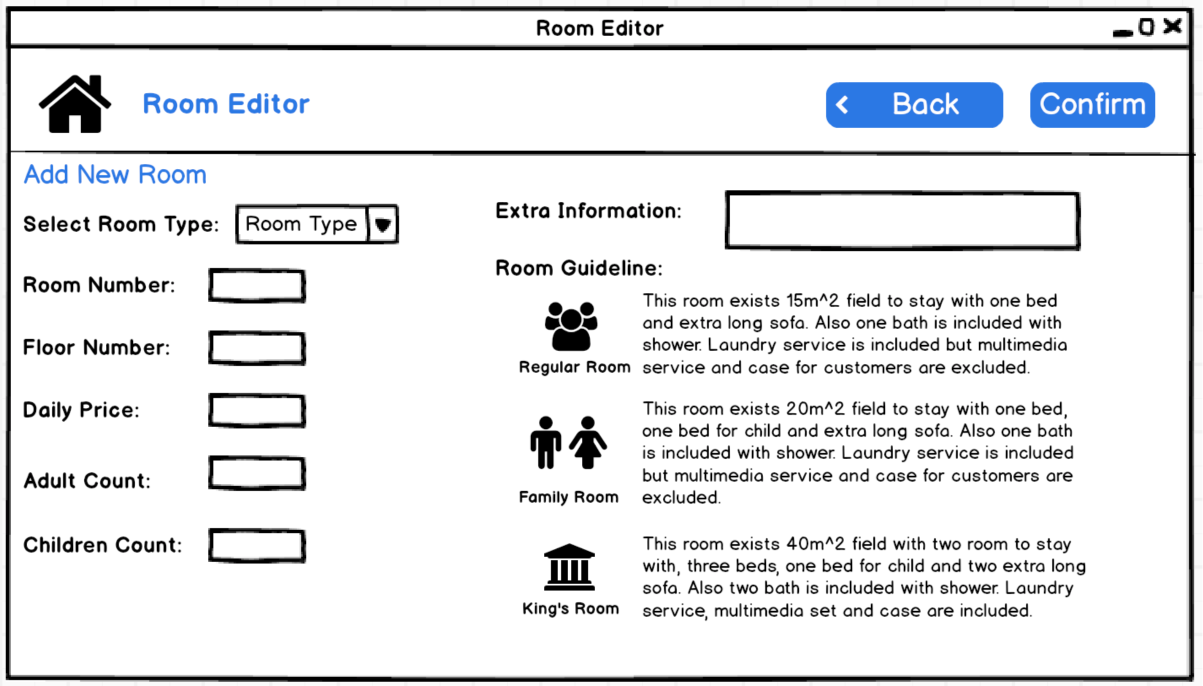


Room Number	Room Type	Status	Floor Number	Adult and Child Count	Daily Price	Delete Room
423	Regular Room	EMPTY	4	2 Adult + 1 Child	1250$	□
515	Family Room	RESERVED	5	1 Adult	330$	☑
111	King's Room	EMPTY	1	2 Adult	80$	□
413	Family Room	EMPTY	4	2 Adult + 2 Children	975$	□
245	King's Room	RESERVED	2	2 Adult	2512$	□
123	Regular Room	RESERVED	1	2 Adult + 2 Children	553$	□
551	King's Room	EMPTY	5	2 Adult + 4 Children	1223$	□
223	Family Room	EMPTY	2	2 Adult + 2 Children	3341$	□
112	Regular Room	RESERVED	1	2 Adult + 3 Children	1612$	□

Figure 8: Room List Screen Mockup

2.7.8 Room Editor Screen

Staff can reach Room Editor, if he/she clicks the “Room Editor” button from main menu. In room editor, staff can create a new room to hotel. From room editor section, staff can select room type from assigned categories. Staff must fill room number, floor number, daily price, adult count and children count at empty boxes. Staff can fill extra information empty boxes to add new information to room. Also in Room Editor Screen, there is a “Room Guideline” that staff can get information about some room types.



This room exists 15m^2 field to stay with one bed and extra long sofa. Also one bath is included with shower. Laundry service is included but multimedia service and case for customers are excluded.

This room exists 20m^2 field to stay with one bed, one bed for child and extra long sofa. Also one bath is included with shower. Laundry service is included but multimedia service and case for customers are excluded.

This room exists 40m^2 field with two room to stay with, three beds, one bed for child and two extra long sofa. Also two bath is included with shower. Laundry service, multimedia set and case are included.

Figure 9: Room Editor Screen Mockup

3. ANALYSIS

3.1. Object Model

3.1.1. Domain Lexicon

In this section we will provide some domain information and important terms to understand the concepts of the project.

Hotel: This class is an entity which reflects the real hotel in the system. It keeps the rooms, customers and reservations in the hotel. Hotel Staff who has authorization can manage the Hotel. New rooms can be added to Hotel and Staff can check in new Customers to Hotel. Therefore, this is an environment that contains other entities such as Room or Customer or Reservation.

Hotel Staff: This is an entity for real life Hotel Staff who manages the Hotel. Each staff should have username and password to use Hotel Management system. Therefore, Hotel Staff can add or delete new rooms, check-in or check-out customers and handle with room reservations. Staff also has access to all hotel information in the system.

Customer: This is an entity class for real Customer who stays in Hotel. They are represented in the system with Customer entity which has real life attributes such as name, surname, phone number etc. Each customer associated with one room in the Hotel during check in or reservation. Customers do not have authorization for the management system, they just stored in the system with their important information.

Reservation: This is an entity for real reservation for a room in the Hotel. Reservation includes the customer information and related room information with this customer. Hotel staff can add new reservations to the system. Each reservation has an expiration date which is assigned by the system. If Customer does not check in before the expiration date reservation is deleted by Reservation Controller.

Room: This class is an entity for a real Room in Hotel. Each Room has specific room number and room type. Also Room can keep a specific number of guests. Moreover, Room can has one of the 3 status which are full, empty or reserved at a specific time.

Date: This class is an entity for real Date. The Hotel Management System should keep the real life Dates of check-in, check-out and reservations. Day compose of 3 attributes which are day, month and year.

Payment: This is an abstract class which represents the real life payment methods.

Cash: This class is an entity class for real Cash. Customer can use to pay the cost of room with cash.

Credit Card: This class is an entity class for real Credit Card. Customer can use to pay the cost of room with credit card.

3.1.2. Class Diagram

Explanations of the classes are below and the class diagram is in the following page.

* Screen classes like RoomScreen and CustomerScreen are represent the user interface of the software. Therefore, they are boundary classes. User interaction is received by these classes.

* Entity classes like Hotel, Customer, HotelStaff, Reservation represents the objects in the real flow of the Hotel activities. In Domain Lexicon section we gave detailed information about these entity classes.

* Control classes like HotelController, RoomController and ReservationController are responsible of changes in the Hotel Management System. They manages data operation such as addition, deletion about the rooms, customers or reservations. Control classes provide the communication between Entity, Boundary and Storage classes.

* Storage classes like HotelStorage, CustomerStorage, RoomStorage are responsible with connection with the database. These classes can make changes in the database such as inserting or deleting data. Also, they gets the available data and transport to controller classes.

* We also added below class diagram which includes detailed class relations.

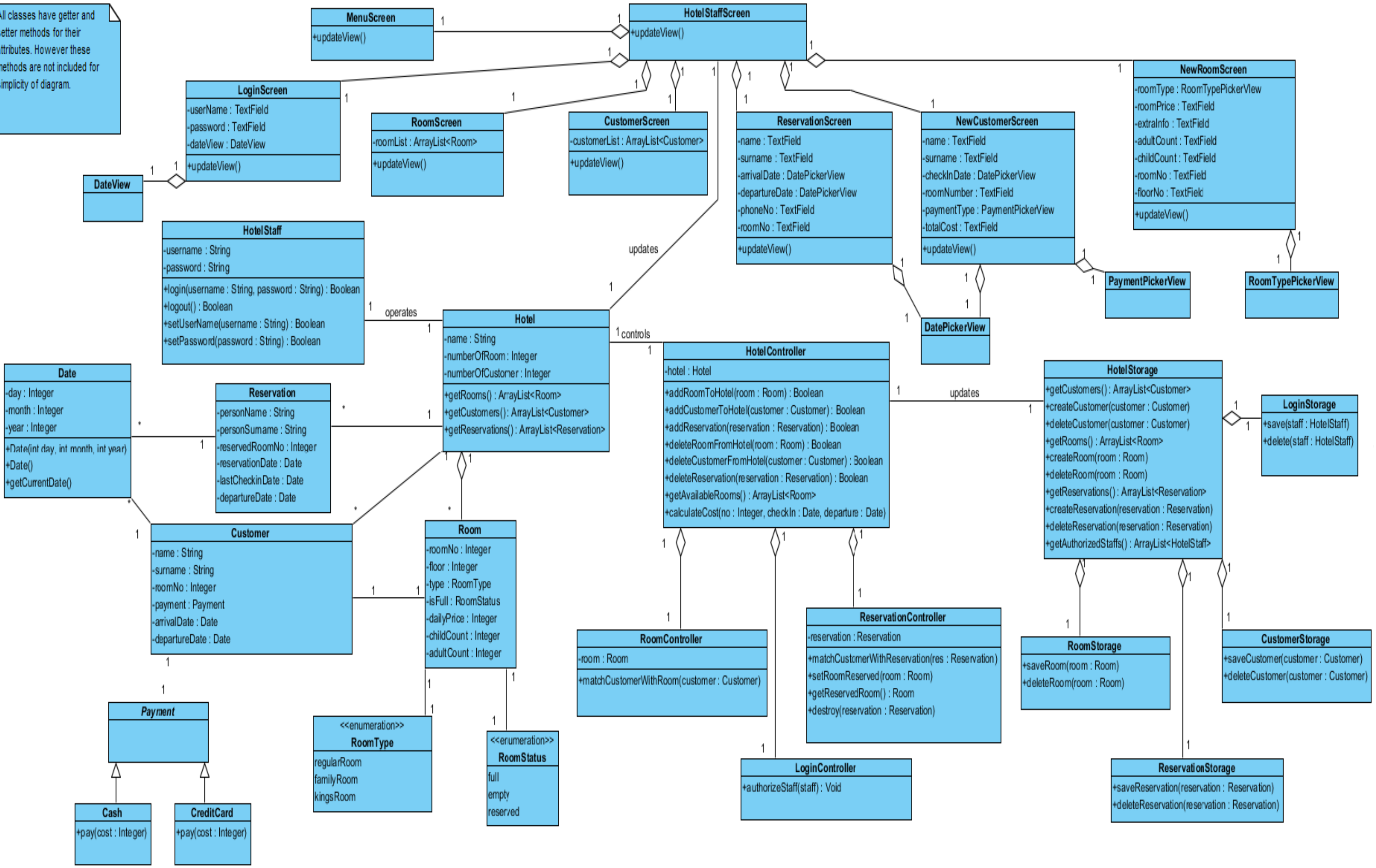

ll classes have getter and
etter methods for their
ttributes. However these
nethods are not included for
implicity of diagram.
MenuScreen
+updateView()
HotelStaffScreen
+updateView()
LoginScreen
-userName : TextField
-password : TextField
-dateView : DateView
+updateView()
DateView
RoomScreen
-roomList : ArrayList<Room>
+updateView()
CustomerScreen
-customerList : ArrayList<Customer>
+updateView()
ReservationScreen
-name : TextField
-surname : TextField
-arrivalDate : DatePickerView
-departureDate : DatePickerView
-phoneNo : TextField
-roomNo : TextField
+updateView()
NewCustomerScreen
-name : TextField
-surname : TextField
-checkInDate : DatePickerView
-roomNumber : TextField
-paymentType : PaymentPickerView
-totalCost : TextField
+updateView()
NewRoomScreen
-roomType : RoomTypePickerView
-roomPrice : TextField
-extraInfo : TextField
-adultCount : TextField
-childCount : TextField
-roomNo : TextField
-floorNo : TextField
+updateView()
PaymentPickerView
RoomTypePickerView
DatePickerView
updates
HotelStaff
-username : String
-password : String
+login(username : String, password : String) : Boolean
+logout() : Boolean
+setUserName(username : String) : Boolean
+setPassword(password : String) : Boolean
operates
Hotel
-name : String
-numberOfRoom : Integer
-numberOfCustomer : Integer
+getRooms() : ArrayList<Room>
+getCustomers(): ArrayList<Customer>
+getReservations(): ArrayList<Reservation>
controls
HotelController
-hotel : Hotel
+addRoomToHotel(room : Room) : Boolean
+addCustomerToHotel(customer : Customer) : Boolean
+addReservation(reservation : Reservation) : Boolean
+deleteRoomFromHotel(room : Room) : Boolean
+deleteCustomerFromHotel(customer : Customer) : Boolean
+deleteReservation(reservation : Reservation) : Boolean
+getAvailableRooms() : ArrayList<Room>
+calculateCost(no : Integer, checkIn : Date, departure : Date)
updates
HotelStorage
+getCustomers() : ArrayList<Customer>
+createCustomer(customer : Customer)
+deleteCustomer(customer : Customer)
+getRooms() : ArrayList<Room>
+createRoom(room : Room)
+deleteRoom(room : Room)
+getReservations() : ArrayList<Reservation>
+createReservation(reservation : Reservation)
+deleteReservation(reservation : Reservation)
+getAuthorizedStaffs() : ArrayList<HotelStaff>
LoginStorage
+save(staff : HotelStaff)
+delete(staff : HotelStaff)
Date
-day : Integer
-month : Integer
-year : Integer
+Date(int day, int month, int year)
+Date()
+getCurrentDate()
Reservation
-personName : String
-personSurname : String
-reservedRoomNo : Integer
-reservationDate : Date
-lastCheckinDate : Date
-departureDate : Date
Customer
-name : String
-surname : String
-roomNo : Integer
-payment : Payment
-arrivalDate : Date
-departureDate : Date
Room
-roomNo : Integer
-floor : Integer
-type : RoomType
-isFull : RoomStatus
-dailyPrice : Integer
-childCount : Integer
-adultCount : Integer
RoomController
-room : Room
+matchCustomerWithRoom(customer : Customer)
ReservationController
-reservation : Reservation
+matchCustomerWithReservation(res : Reservation)
+setRoomReserved(room : Room)
+getReservedRoom() : Room
+destroy(reservation : Reservation)
RoomStorage
+saveRoom(room : Room)
+deleteRoom(room : Room)
CustomerStorage
+saveCustomer(customer : Customer)
+deleteCustomer(customer : Customer)
Payment
Cash
+pay(cost : Integer)
CreditCard
+pay(cost : Integer)
<<enumeration>>
RoomType
regularRoom
familyRoom
kingsRoom
<<erumeration>>
RoomStatus
full
empty
reserved
LoginController
+authorizeStaff(staff) : Void
ReservationStorage
+saveReservation(reservation : Reservation)
+deleteReservation(reservation : Reservation)

Figure 10: Class Diagram

3.2. Dynamic Models

3.2.1. State Chart

Check-in State Chart Diagram

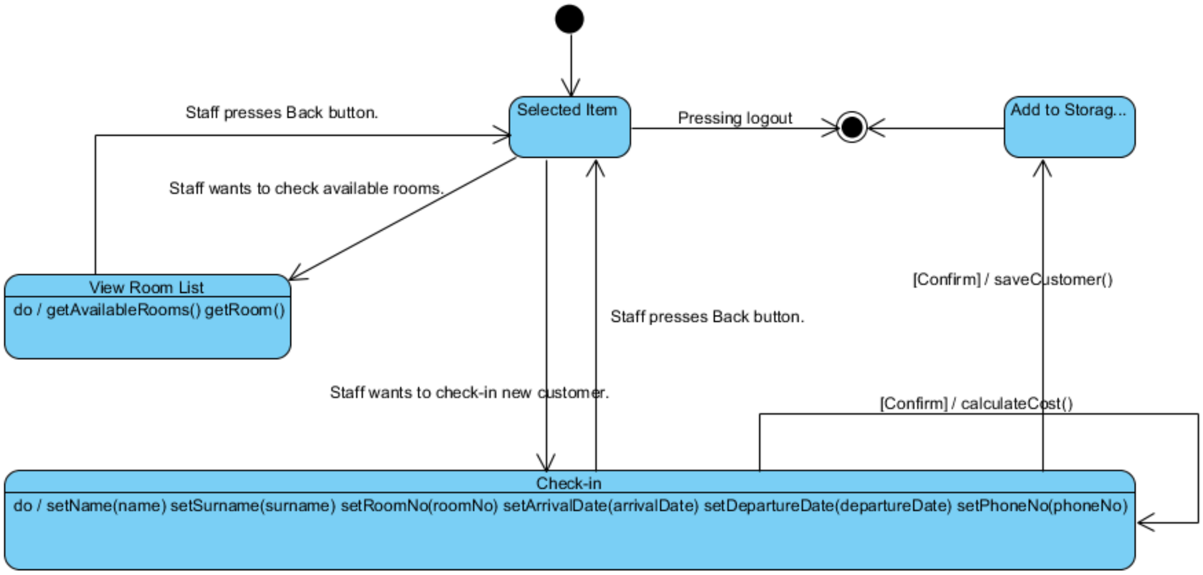

Selected Item
Pressing logout
Add to Storag...
Staff presses Back button.
Staff wants to check available rooms.
View Room List
do / getAvailableRooms() getRoom()
[Confirm] / saveCustomer()
Staff presses Back button.
Staff wants to check-in new customer.
[Confirm] / calculateCost()
Check-in
do / setName(name) setSurname(surname) setRoomNo(roomNo) setArrivalDate(arrivalDate) setDepartureDate(departureDate) setPhoneNo(phoneNo)

Figure 11: Check-in State Chart Diagram

If hotel staff wants to check-in, s/he should control available room list. After controlling, s/he enters check-in page and writes customer’s information. If s/he pushes “Confirm” button, Hotel Master saves customer’s information to storage and stays the check-in page. If s/he pushes “Back” button, s/he returns main menu.

Create Room State Chart Diagram

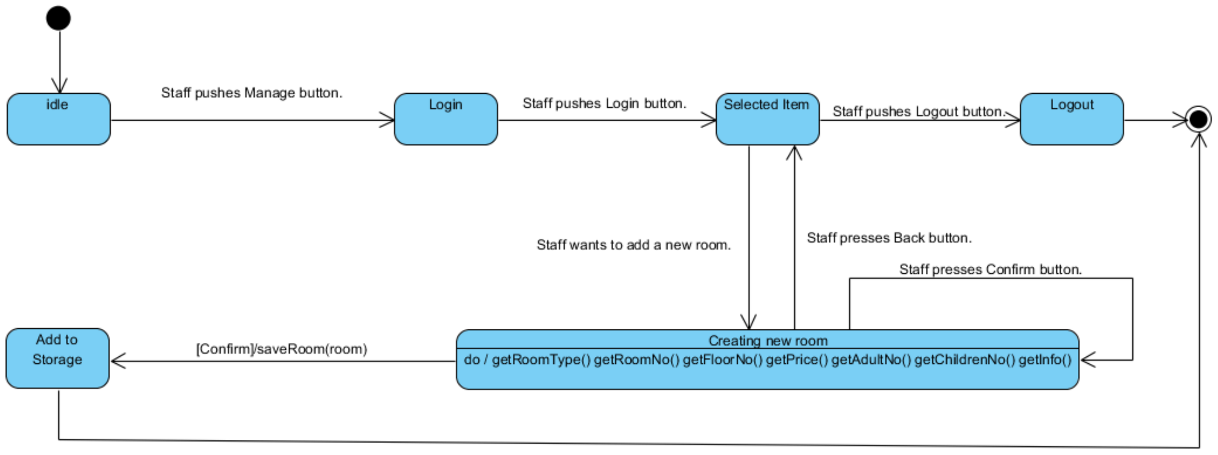

idle
Staff pushes Manage button.
Login
Staff pushes Login button.
Selected Item
Staff pushes Logout button.
Logout
Staff wants to add a new room.
Staff presses Back button.
Staff presses Confirm button.
Add to Storage
[Confirm]/saveRoom(room)
Creating new room
do / getRoomType() getRoomNo() getFloorNo() getPrice() getAdultNo() getChildrenNo() getInfo()

Figure 12: Create Room State Chart Diagram

If hotel staff pushes Manage button, s/he can enter the system. Also s/he wants to enter the system, s/he should his/her account information. After entering the system, if staff want to quit, his/her pushes “Logout” button. If staff wants to create a new room to their hotel, s/he should push “Room Editor” item and selected item directs person to creating room page. After entering room’s information to the system, staff pushes “Confirm” button, Hotel Master saves room’s information to storage and stays the creating room page. If s/he pushes “Back” button, s/he returns main menu.

Delete Customer State Chart Diagram

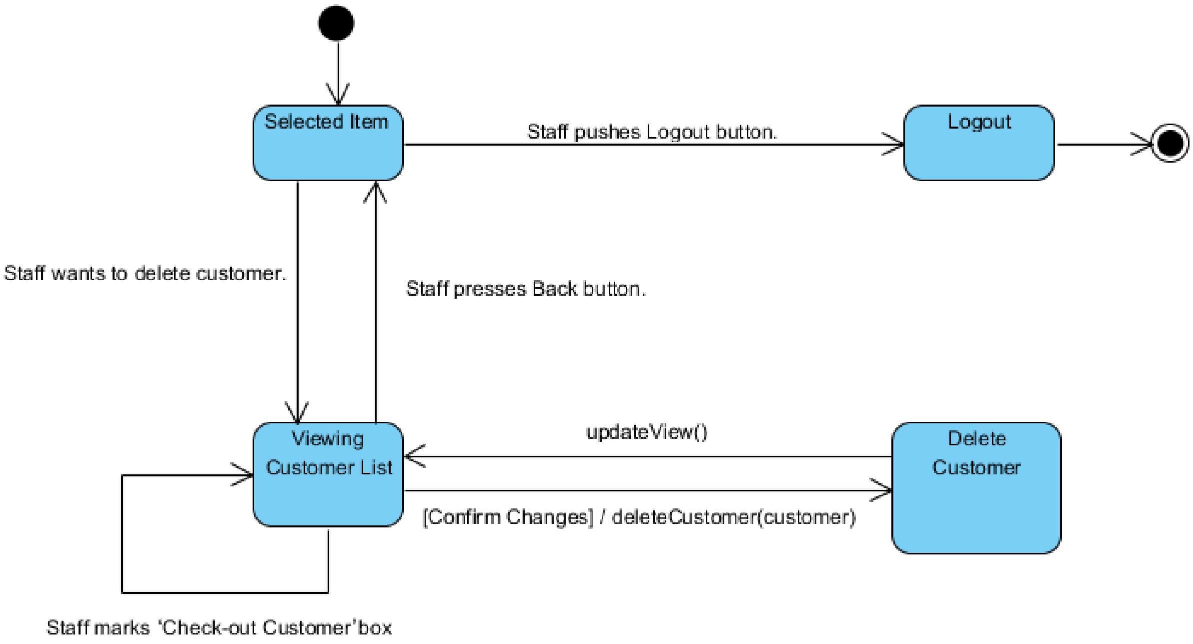

Selected Item
Staff pushes Logout button.
Logout
Staff wants to delete customer.
Staff presses Back button.
updateView()
Viewing Customer List
Delete Customer
[Confirm Changes] / deleteCustomer(customer)
Staff marks 'Check-out Customer' box

Figure 13: Delete Customer State Chart Diagram

If hotel staff wants to delete customer from the system, selected item directs person to viewing customer list page. If s/he marks the check box and pushes "Confirm" button, Hotel Master deletes customer's information from the list and the storage and stays the viewing customer list page. If s/he pushes "Back" button, s/he returns main menu. If staff want to quit, his/her pushes "Logout" button.

Delete Room State Chart Diagram

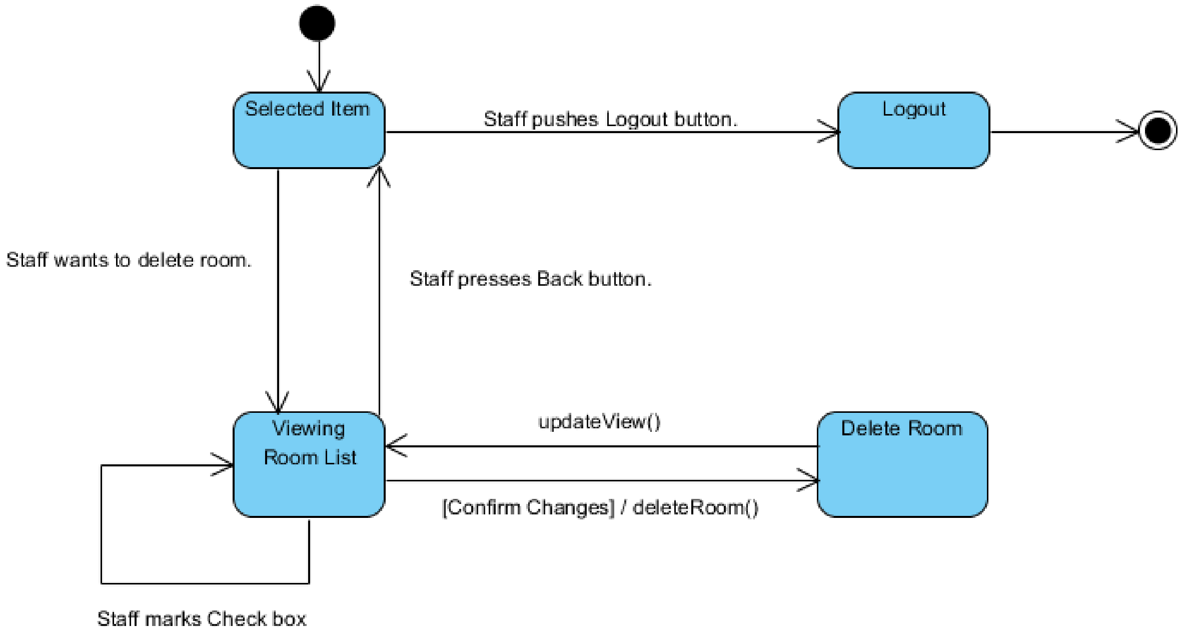

Selected Item
Staff pushes Logout button.
Logout
Staff wants to delete room.
Staff presses Back button.
updateView()
Viewing Room List
Delete Room
[Confirm Changes] / deleteRoom()
Staff marks Check box

Figure 14: Delete Room State Chart Diagram

If hotel staff wants to delete room from the system, selected item directs person to viewing room list page. If s/he marks the check box and pushes “Confirm” button, Hotel Master deletes room’s information from the list and the storage and stays the viewing room list page. If s/he pushes “Back” button, s/he returns main menu. If staff want to quit, his/her pushes “Logout” button.

Reservation State Chart Diagram

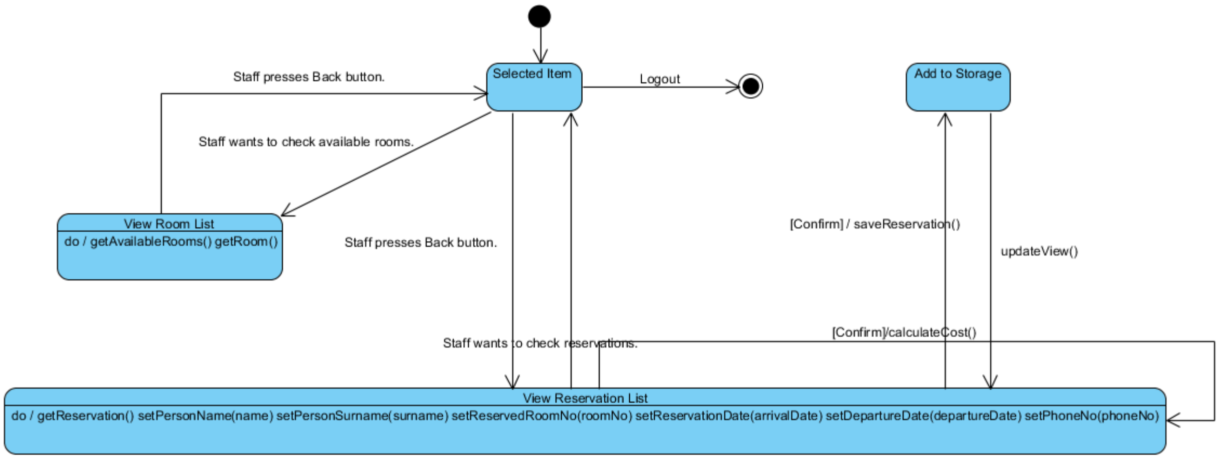

Selected Item
Logout
Add to Storage
Staff presses Back button.
Staff wants to check available rooms.
View Room List
do / getAvailableRooms() getRoom()
Staff presses Back button.
[Confirm] / saveReservation()
updateView()
Staff wants to check reservations.
[Confirm]/calculateCost()
View Reservation List
do / getReservation() setPersonName(name) setPersonSurname(surname) setReservedRoomNo(roomNo) setReservationDate(arrivalDate) setDepartureDate(departureDate) setPhoneNo(phoneNo)

Figure 15: Reservation State Chart Diagram

If hotel staff wants to reservation, s/he should control available room list. Staff returns the main menu page by using “Back” button. Then, selected item directs person to viewing reservation list page. After entering customer’s information to the system and selects room, staff pushes “Confirm” button, Hotel Master calculates total cost and saves reserved information to storage and stays the viewing reservation list page. If s/he pushes “Back” button, s/he returns main menu.

View List State Chart Diagram

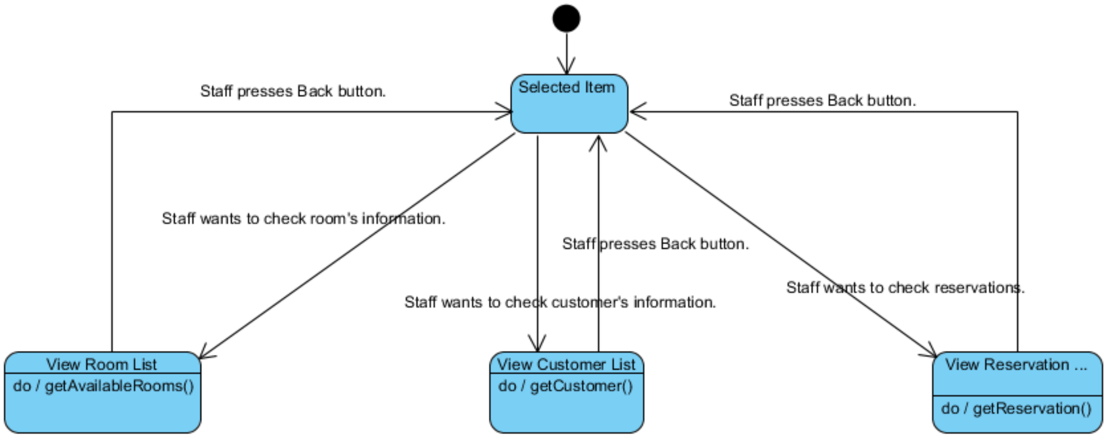

Selected Item
Staff presses Back button.
Staff presses Back button.
Staff wants to check room's information.
Staff presses Back button.
Staff wants to check customer's information.
Staff wants to check reservations.
View Room List
do / getAvailableRooms()
View Customer List
do / getCustomer()
View Reservation ...
do / getReservation()

Figure 16: View List State Chart Diagram

If hotel staff controls available rooms or room list, selected item directs person to viewing room list page. If hotel staff controls customer's information and their rooms, selected item directs person to viewing customer list page. If hotel staff controls reservations and make a reservation, selected item directs person to viewing reservation list page. If s/he pushes "Back" button, s/he returns main menu.

3.2.2. Sequence Diagram

Scenario Name: Create Room

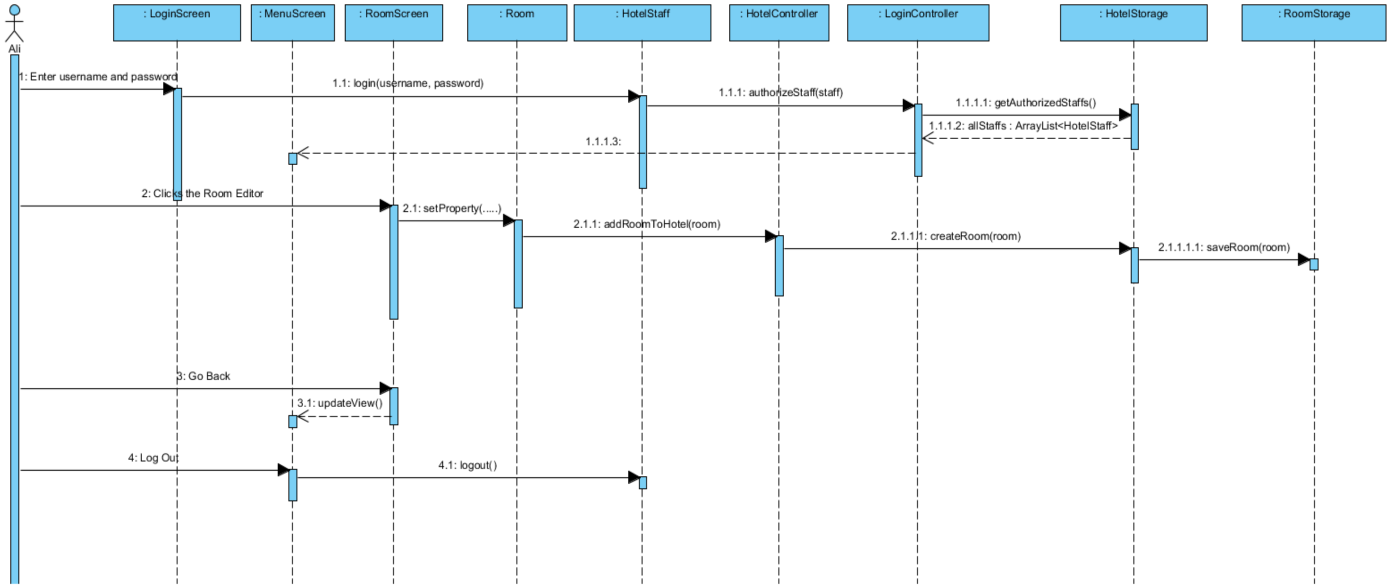

: LoginScreen
: MenuScreen
: RoomScreen
: Room
: HotelStaff
: HotelController
: LoginController
: HotelStorage
: RoomStorage
Ali
1: Enter username and password
1.1: login(username, password)
1.1.1: authorizeStaff(staff)
1.1.1.1: getAuthorizedStaffs()
1.1.1.2: allStaffs : ArrayList<HotelStaff>
1.1.1.3:
2: Clicks the Room Editor
2.1: setProperty(.....)
2.1.1: addRoomToHotel(room)
2.1.1.1: createRoom(room)
2.1.1.1.1: saveRoom(room)
3: Go Back
3.1: updateView()
4: Log Out
4.1: logout()

Figure 17: Scenario Create Room Sequence Diagram

Description: Ali enters his username and password in Login Screen. Then, HotelStaff calls its login method with Ali’s username and password. Then, LoginController checks for authorization of HotelStaff. HotelStorage gets the list of authorized staff and returns this list to LoginController. If HotelStaff is a valid staff then LoginContoller authorizes the staff and directs Ali to MenuScreen. Then, Ali clicks the Room Editor and opens RoomScreen. He enters the properties of the room. Then setter methods of the Room sets the values. After that, HotelController adds this values to hotel and HotelStorage creates a room for database then RoomStorage saves room to database.

Scenario Name: Check-in New Customer

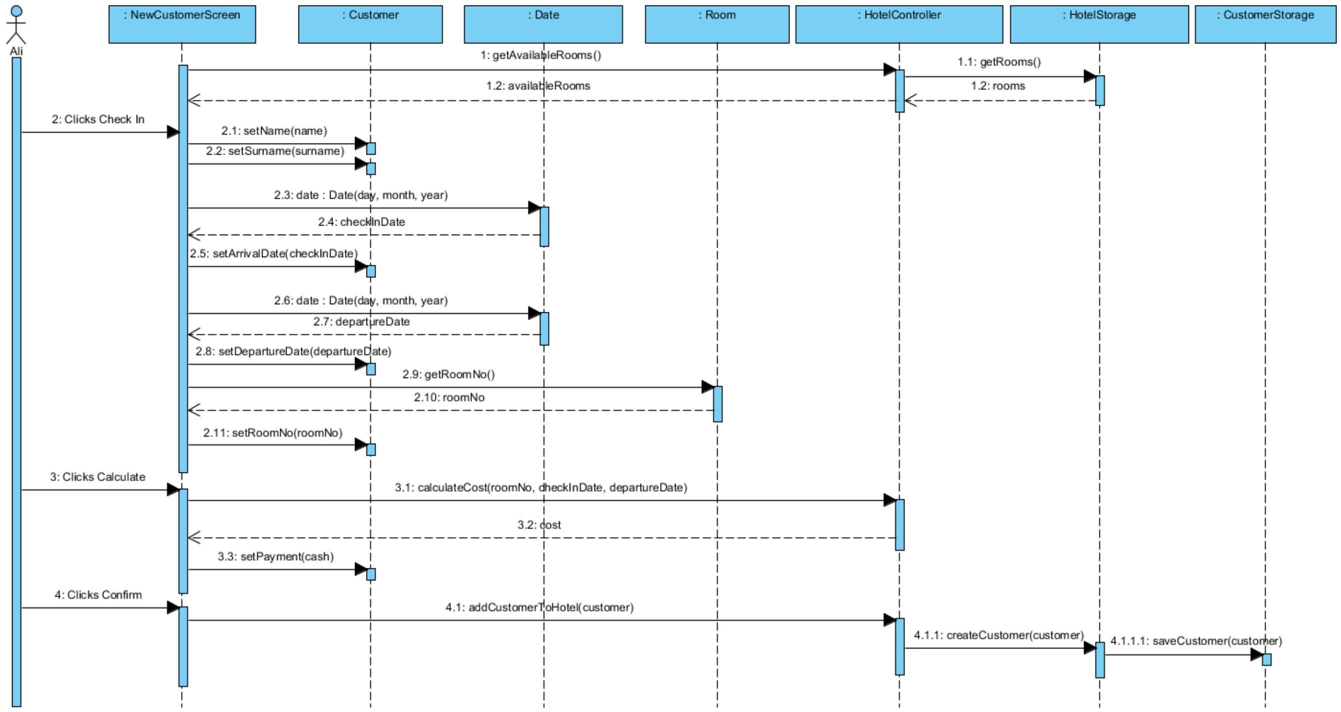
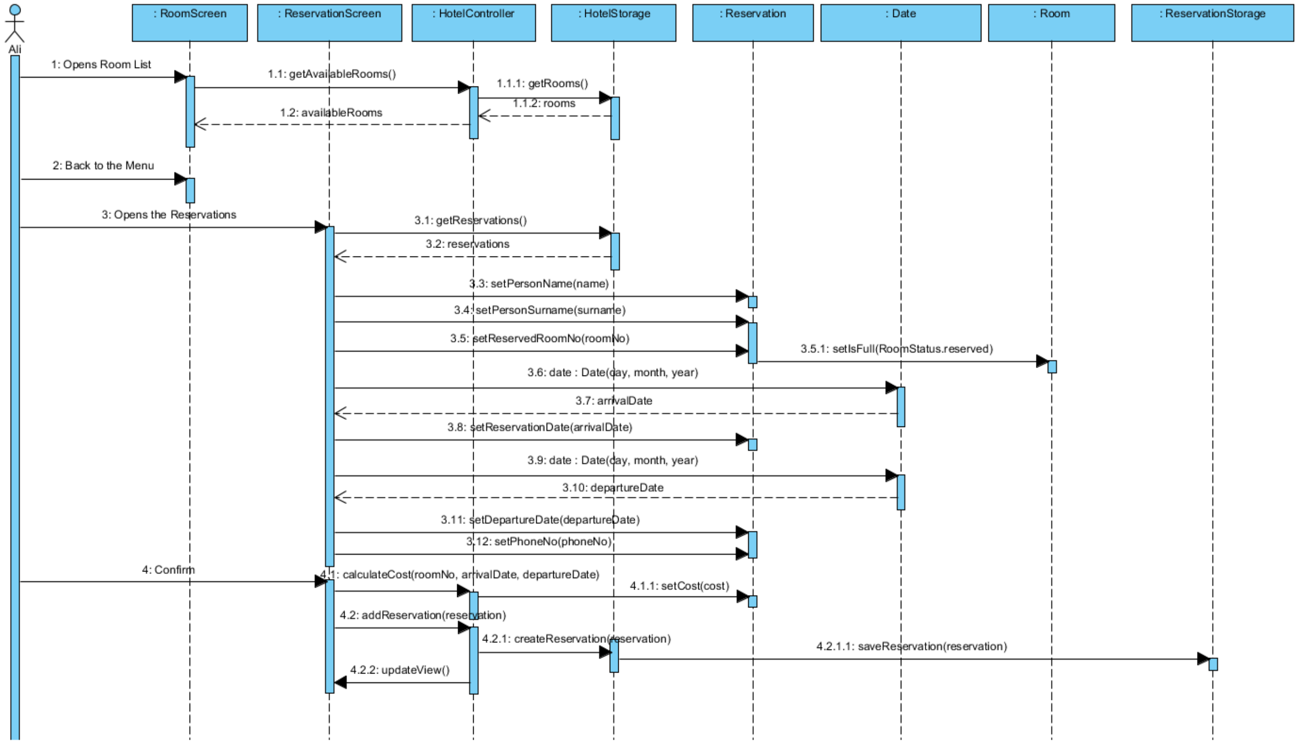


Figure 18: Scenario Check-in New Customer Sequence Diagram

Description: Ali clicks the Check in from main menu and opens the NewCustomerScreen. NewCustomer screen have to show available rooms. Therefore, HotelStorage returns rooms to HotelController then HotelController analyses rooms and return available rooms to screen. Ali enters the name and the surname of the customer so setter methods of the Customer set the values. Then, He chooses the arrival and departure dates from the screen. Later, he chooses an available room from screen. All these values are set by setter methods of Customer. After that, Ali clicks the calculate button and HotelController calculates the cost then returns the value. Lastly Ali selects payment type of customer and clicks confirms. HotelController directs this Customer to HotelStorage to add this new customer to Hotel database.

Scenario Name: Make a Reservation



: RoomScreen
: ReservationScreen
: HotelController
: HotelStorage
: Reservation
: Date
: Room
: ReservationStorage
Ali
1: Opens Room List
1.1: getAvailableRooms()
1.1.1: getRooms()
1.1.2: rooms
1.2: availableRooms
2: Back to the Menu
3: Opens the Reservations
3.1: getReservations()
3.2: reservations
3.3: setPersonName(name)
3.4: setPersonSurname(surname)
3.5: setReservedRoomNo(roomNo)
3.5.1: setIsFull(RoomStatus.reserved)
3.6: date : Date(day, month, year)
3.7: arrivalDate
3.8: setReservationDate(arrivalDate)
3.9: date : Date(day, month, year)
3.10: departureDate
3.11: setDepartureDate(departureDate)
3.12: setPhoneNo(phoneNo)
4: Confirm
4.1: calculateCost(roomNo, arrivalDate, departureDate)
4.1.1: setCost(cost)
4.2: addReservation(reservation)
4.2.1: createReservation(reservation)
4.2.1.1: saveReservation(reservation)
4.2.2: updateView()

Figure 19: Scenario Make a Reservation Sequence Diagram

Description: Ali opens the Room List to find an available room. After finding an available room, Ali goes back to main menu and opens the ReservationScreen. Then, He enters the name, surname, reserved room number. He picks the arrival date and departure date. Lastly, he enters the phone number of customer. All these attributes are set by setter methods of Reservation. Lastly, Ali clicks confirm button and HotelController adds Reservation to the storage and refreshes the ReservationScreen.

Scenario Name: View Customer and Room Lists

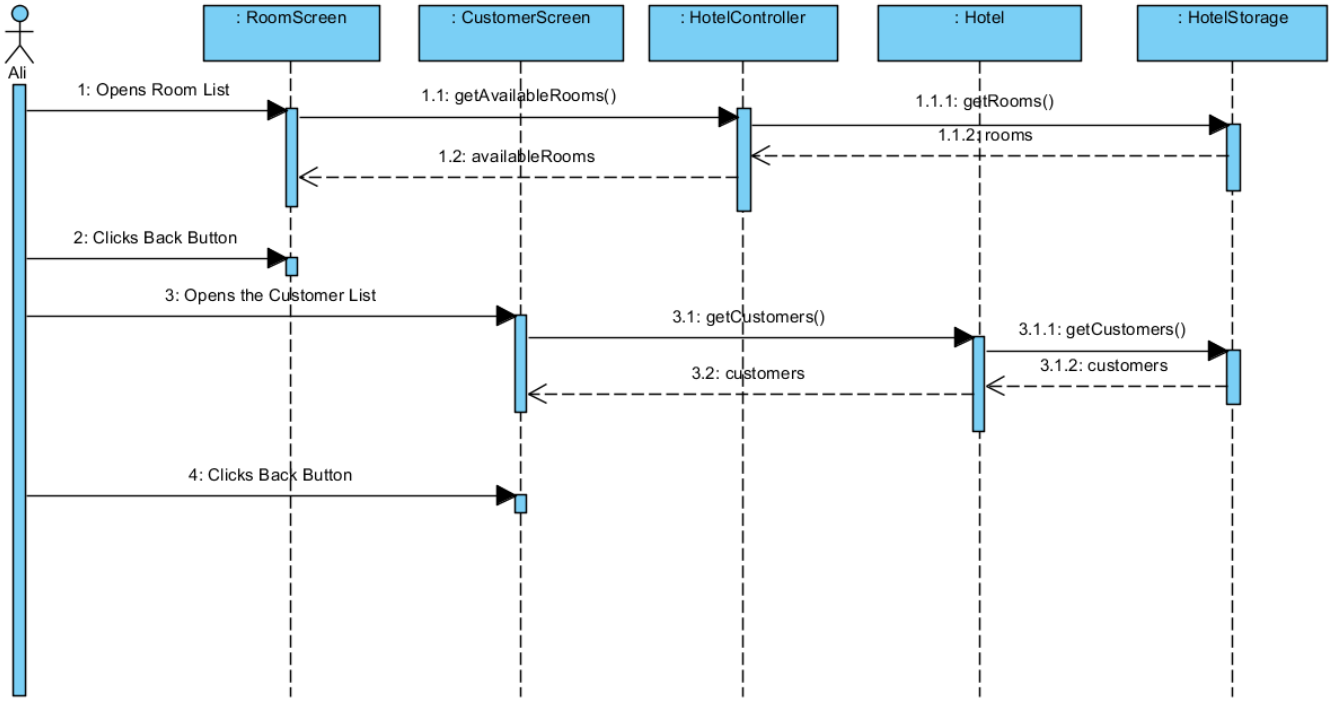


Figure 20: Scenario View Customer and Room Lists Sequence Diagram

Description: Ali first opens the RoomScreen to view available rooms (reserved rooms are included). HotelController get the available rooms from HotelStorage and updates RoomsScreen. Then, Ali goes back to the mane and opens the CustomerScreen. Hotel gets all customers from HotelStorage and updates the CustomerScreen. After checking customers, Ali goes back to the menu.

Scenario Name: Delete Room

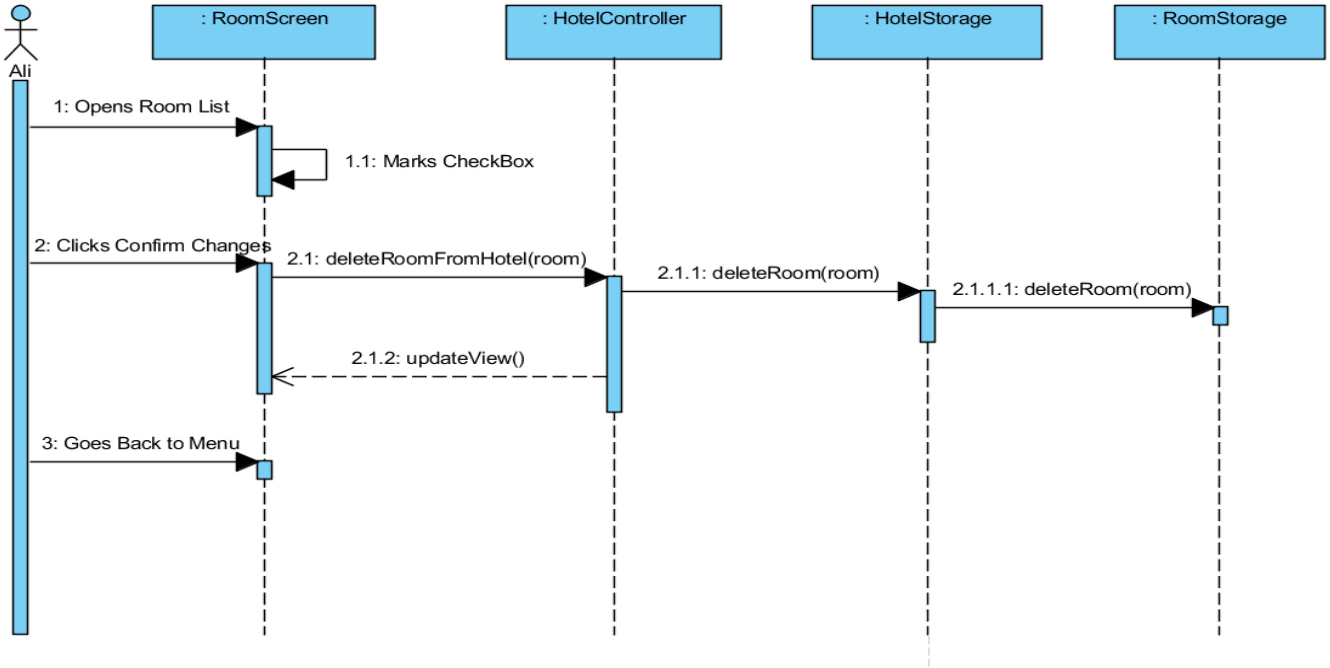


Figure 21: Scenario Delete Room Sequence Diagram

Description: Ali opens the RoomScreen. He finds the room which he wants to delete. He marks the delete checkbox of this room. Then, he clicks to confirm changes button. HotelContoller make request to delete this room from storage and RoomStorage deletes this room. Lastly, HotelController updates the Room List and Ali goes back to menu.

Scenario Name: Check-Out Customer

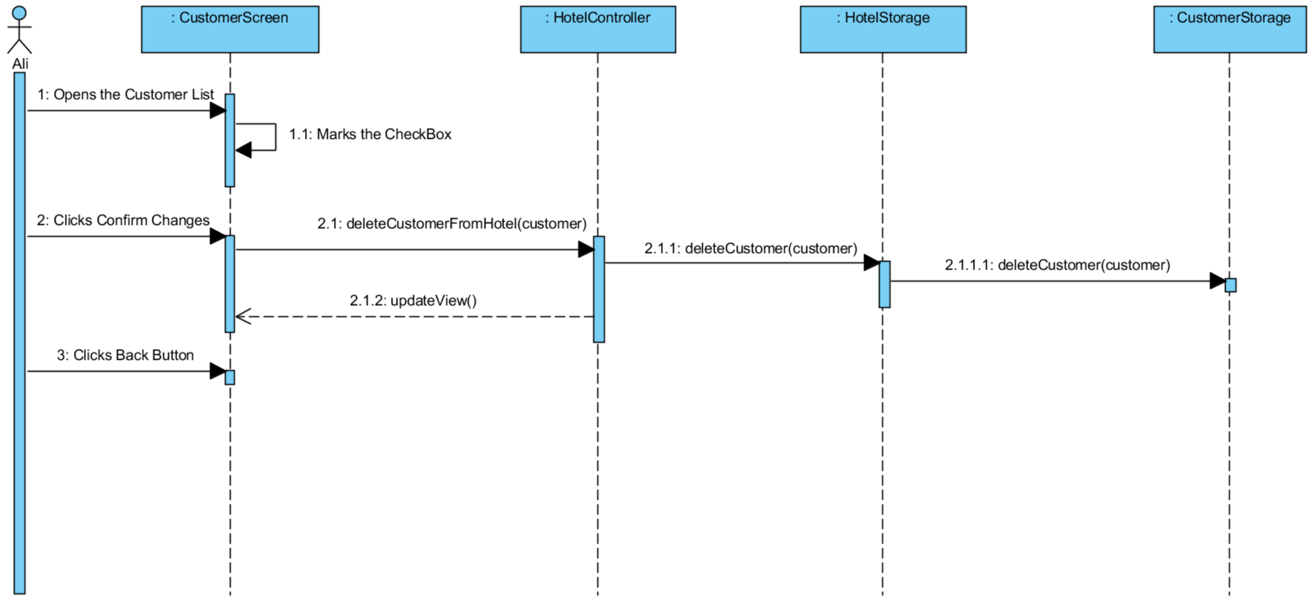

: CustomerScreen
: HotelController
: HotelStorage
: CustomerStorage
Ali
1: Opens the Customer List
1.1: Marks the CheckBox
2: Clicks Confirm Changes
2.1: deleteCustomerFromHotel(customer)
2.1.1: deleteCustomer(customer)
2.1.1.1: deleteCustomer(customer)
2.1.2: updateView()
3: Clicks Back Button

Figure 22: Scenario Check-out Customer Sequence Diagram

Description: Ali opens the Customer List to check out a customer. He finds the customer's column in the list and marks its checkbox. Then, he clicks the Confirm Changes Button. HotelController makes request to delete customer from storage and CustomerStorage deletes the customer from storage. After that, HotelController updates the CustomerScreen. Lastly, Ali goes back to menu.

4. Design

Design part includes design goals, subsystem decomposition, architectural patterns, hardware/software mapping and addressing key concerns.

4.1 Design Goals

There are many design goals for HotelMaster to create a solid and functional object-oriented system.

4.1.1 Maintainability and Reusability

Our program is permanently used by different hotel staffs. This means that different people use the same program and reach the same information about customers. Our program provides a staff to use and change another staff's entering if required. For example, a staff add new customer to the system or create a new room type for hotel. Another staff can change customer's information or delete a room which is created by a staff before. This interaction leads to maintainability system and eliminates unnecessarily usage. Also it saves information to database and another staff reach them easily. Deleting customer information or room does not affect the interface of the program so it makes the program reusable.

4.1.2 Low Coupling and High Coherence

In the design process low coupling and high coherence is important in HotelMaster. Low coupling reduces the complexity of the system and causes less dependency between classes to class. High coherence means the associations between subsystem having similar tasks. Low coupling and high allows the system maintainable, understandable, modifiable and having high performance. It provides our system to be good and well designed.

4.1.3 User Friendly

One of our purpose things is creating understandable and user-friendly program. User interface and functionality of our program are understandable for people who know beginner level of computer. If hotel staff logs on the passing the pages or enters customer's information wrongly, customer should not be waited by hotel staff and customer can troubled with waiting or getting wrong information. Also if a staff chooses wrong payment type because of the interface, the system calculates wrong bills. This kinds of mistakes may affect the hotel's prestige.

4.1.4 Ease of Use

People who use computer even beginner level can use our program easily. Hotel Master is supported by icons, buttons, text filed and selection items with easy language. Because of icons, it is easy to remember which event happened on where.

4.1.5 Reliability

Reliability is important to customer because the system include customer's name, surname, phone numbers and payment information and important to hotel staff for saving information to storage. Therefore, the program should not have any errors and exceptions. Program saves these information to database system and protect them. Any error or mistake based on program, causes information loss. Also, if the login page has an error, people who do not have a permission to enter the system cannot use any information about customers. Our purpose is to protect information and constructing reliability.

4.2 Sub-System Decomposition

Sub-System Decomposition is one of the crucial keys and design principles to obtain well prepared complex system. This principle divides a large system down into progressively smaller classes or objects that are responsible for some part of the problem domain. In order to continue, decomposition is a strategy to organize a program with some specific number of its parts and its aim to use is supplying maintainability and modularity to the program, in other words, system. To start with, we decomposed our program into four sub-systems. In order to give detail, these four sub-systems are called Hotel Management Sub-system, Hotel Entities Sub-system, Database Management Sub-system and Hotel Staff Screen Sub-system. High cohesion and low coupling are our aimed and key concern that we have given attention to them in order to supply well prepared program. Because of each sub-system handles only one concern, “high cohesion” is satisfied. In addition to these, to succeed “low coupling”, each sub-system has classes and these classes are not depended.

Hotel Staff Screen Subsystem:

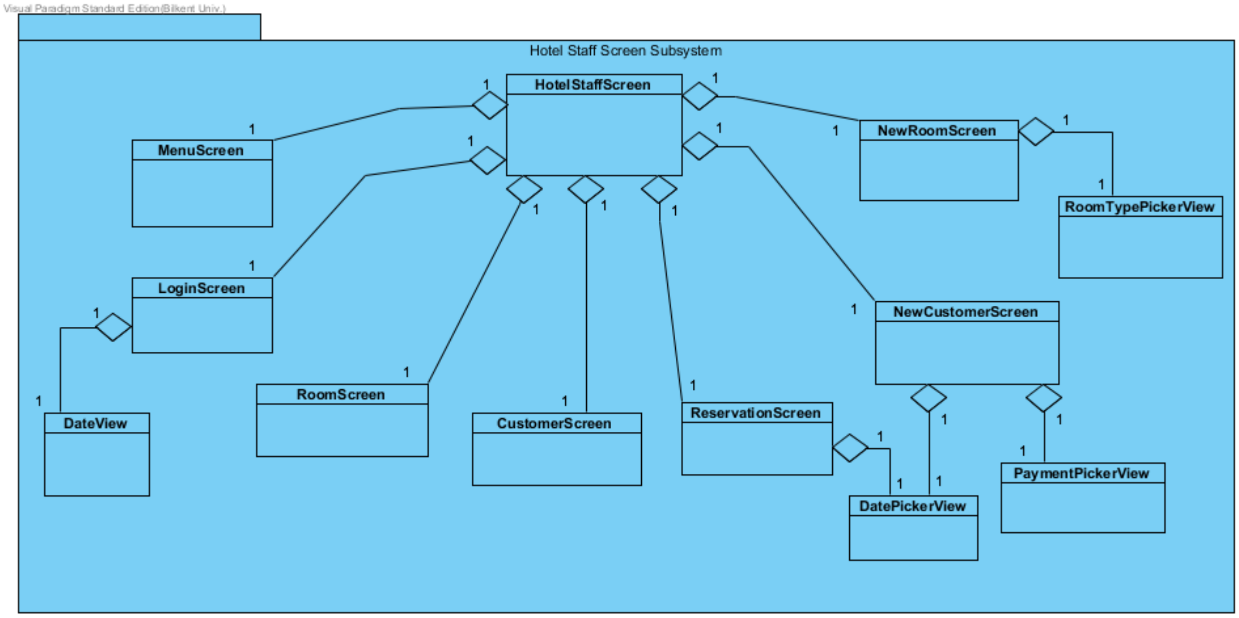


Figure 23: Hotel Staff Screen Subsystem

Hotel Staff Screen Subsystem consists of HotelStaffScreen, MenuScreen, LoginScreen, RoomScreen, CustomerScreen, ReservationScreen, NewCustomerScreen, NewRoomScreen, DateView, DatePickerView, PaymentPickerView, RoomTypePickerView classes. This subsystem updates the screen of staff's according to staff's choice and classes are called by the choices. These classes' tasks are followings:

HotelStaffScreen is responsible to manage all parts of hotel staff's screen in order to staff's action by using other classes.

MenuScreen is responsible from presentation of selected categories' menus.

LoginScreen is responsible from supplying staff to login the system by entering username and password.

RoomScreen is responsible from representing a screen and listing the room information with some customer information to the staff with searching bar and with some deletion from staff to implement room processes.

CustomerScreen is responsible from representing a screen and listing a customer information with some room information to the staff with searching bar and with some deletion from staff to implement customer processes.

ReservationScreen is responsible from representing and listing a screen to reserving information with some customer and room information to the staff with searching bar with some deletion from staff to cancel reservation or check-out reservations.

NewCustomerScreen is responsible from representing a screen to enter the new customer information with name, surname, phone number etc. in order to enroll new customer to the hotel by staff.

NewRoomScreen is responsible from representing a screen to add rooms to the hotel to staff with some room information and with some default room's guidelines.

DateView is responsible from representing a calendar at *LoginScreen* to the staff.

DatePickerView is responsible from representation a calendar to both *ReservationScreen* and *NewCustomerScreen* to select the date of customers come to visit their hotel.

PaymentPickerView is responsible from representing the payment types to the *NewCustomerScreen* in order to staff can select the payment type of specific customer.

Database Management Subsystem:

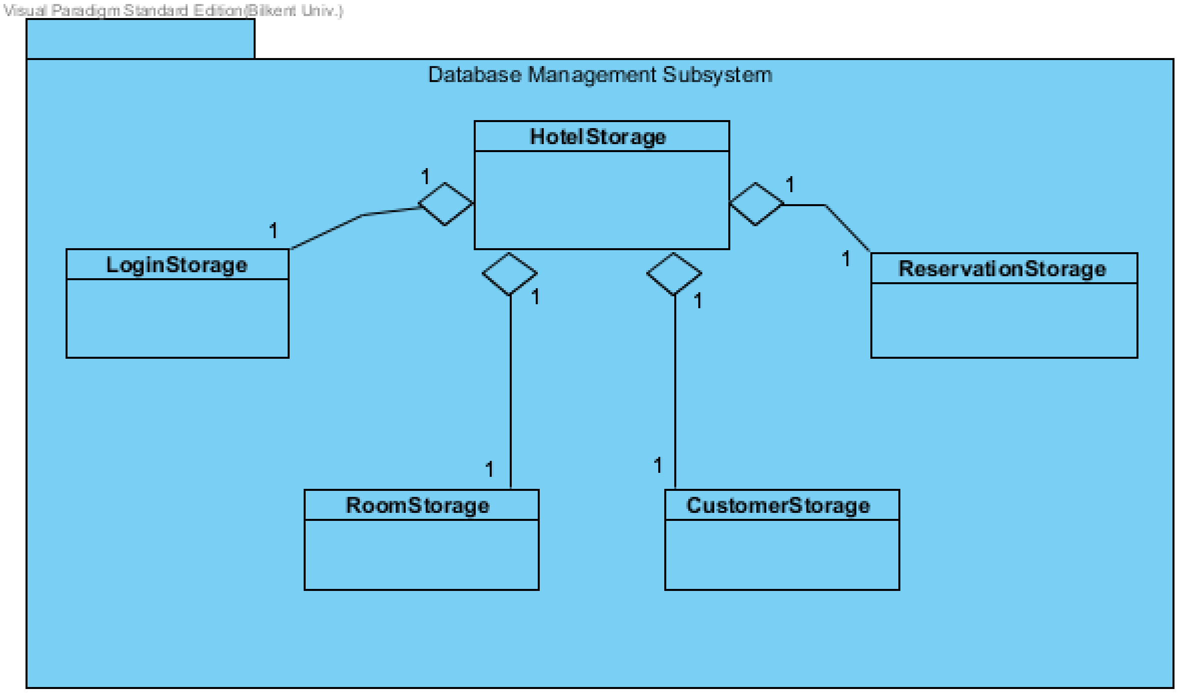


Figure 24: Database Management Subsystem

Database Management Subsystem consists of HotelStorage, LoginStorage, RoomStorage, CustomerStorage, ReservationStorage classes. This subsystem holds the information of both login, rooms, customers and reservations under the HotelStorage root class. This subsystem's main aim is creating required tables of rooms, customers, reservations to the specified screen classes for staff to see, investigate, manipulate and delete the processes. Hence, tables are interacting with the database when changes happen. To continue with, when a new customer is enrolled or new room is created or some new reservations are made, HotelStorage will update itself and change tuples of the database. To sum up, these classes will use JDBC library and change the state of the database according to change during interaction between system and hotel staff.

Hotel Management Subsystem:

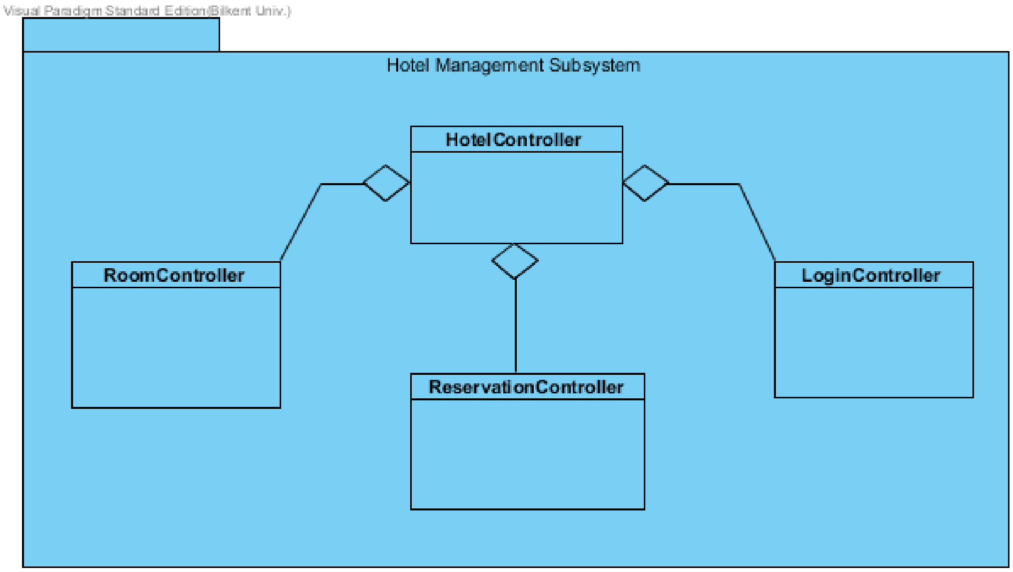

Visual Paradigm Standard Edition(Bilkent Univ.)
Hotel Management Subsystem
HotelController
RoomController
LoginController
ReservationController

Figure 25 : Hotel Management Subsystem

Hotel Management Subsystem consist of four classes. They are HotelController, RoomController, LoginController and ReservationController. This subsystem handles with changes of entities and database. These classes' tasks are like following:

HotelController is essentially responsible from managing data and entities according to the staff's interactions and actions at the system by using specified controllers. HotelController allows interactions between system and hotel staff and holds controlling of room, reservation and login events.

RoomController is essentially responsible from managing data and entities according to the staff's interactions and actions at the system by using specified controllers. To illustrate, if hotel staff creates a new room to add the hotel RoomController class creates new room under the circumstances of typed information of new room.

ReservationController is essentially responsible from managing data and entities according to the staff's interactions and actions at the system by using specified controllers. To illustrate, if hotel staff creates a new reservation, ReservationController class creates new reservation under the circumstances of typed information of reservation.

LoginController is essentially responsible from managing data and entities according to the staff's interactions and actions at the system by using specified controllers. LoginController allows hotel staff to enroll to the system to start processing about hotel by using related entities.

Hotel Entities Subsystem:

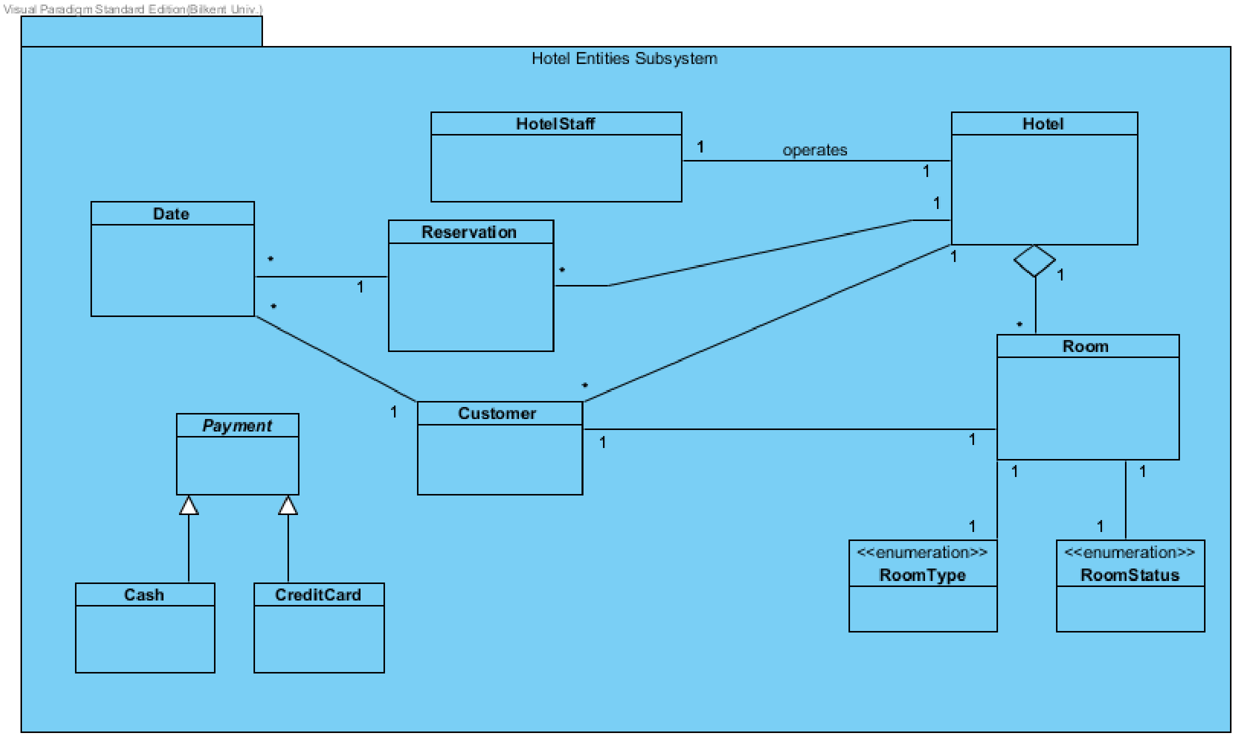


Figure 26 : Hotel Entities Subsystem

Hotel Entities Subsystem consists of eleven classes. They are HotelStaff, Hotel, Room, RoomType, RoomStatus, Date, Reservation, Customer, Payment, Cash and CreditCard. Entities of the system are included at this subsystem. In order to give detail, these are the representation of the objects of the software during the interaction between hotel staff and system. Also, these classes are responsible from interacting related representation classes when changes occur.

4.3 Architectural Patterns

According to HotelMaster subsystems we used three different architectural patterns which are Model-View-Controller Pattern, Client-Server Pattern and Layers Pattern.

4.3.1 (MVC) Model-View-Controller Pattern

In system design of HotelMaster, we use Model-View-Controller pattern to connect the subsystems to each other appropriately. When a Hotel staff uses the HotelMaster and makes some operations (such as login, add room, delete room etc.), controller part of the system which is “Hotel Management Subsystem” is updated. After that, controller updates the model parts of the system which are “Hotel Entities Subsystem” and “Database Management Subsystem”. Then, model notifies the view part of the system which is “Hotel Staff Screen Subsystem”. The view parts get data from the model parts and views are updated. The following diagram shows the relations between model-view-controller parts.

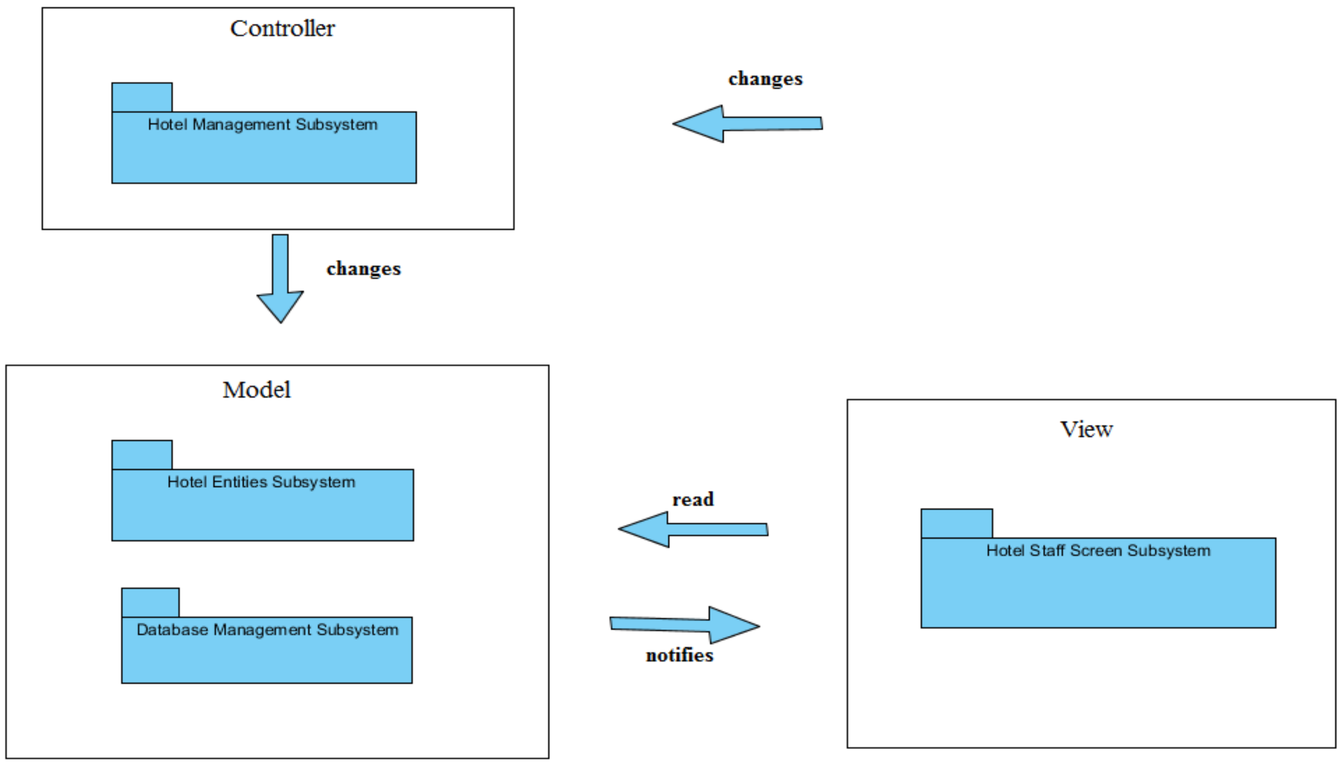

Controller
Hotel Management Subsystem
changes
changes
Model
Hotel Entities Subsystem
Database Management Subsystem
read
notifies
View
Hotel Staff Screen Subsystem

Figure 27 : Model-View-Controller

4.3.2 Client-Server Pattern

HotelMaster needs a database server to store room, customer and reservation information. We use MySQL database which runs on a server to store and provide information. Also, there is a Hotel Computer as a client. Therefore, we use Client-Server pattern to design our architecture.

Client requests some services and database server provides these services. Following diagram shows the relationship.

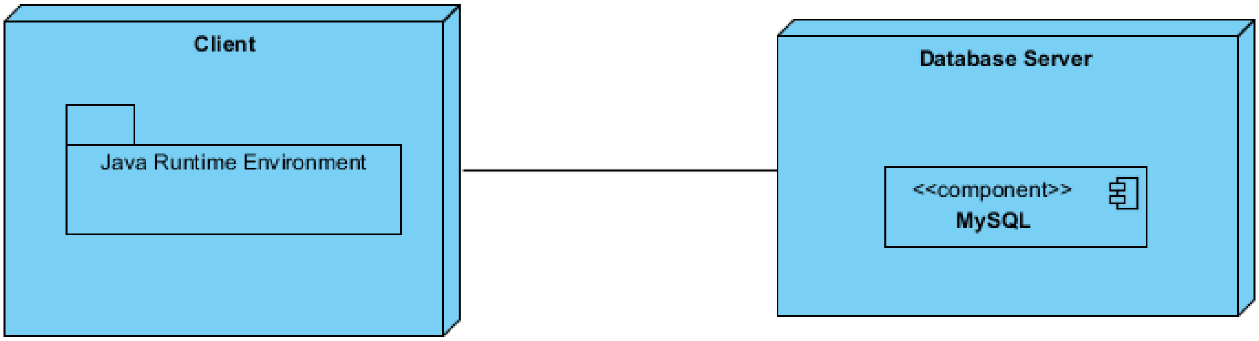


Figure 28 : Client-Server Pattern

4.3.3 Layer Pattern

Lastly, Layers Pattern is used in HotelMaster. This pattern shows the relations and data flows between packages in the system. The first layer is Presentation Layer. This layer consist of “Hotel Staff Screen Subsystem” and can access the services of “Hotel Management Subsystem”. Second layer is Application Logic Layer which includes “Hotel Management Subsystem”. In this layer, “Hotel Management Subsystem” is allowed to use “Database Management Subsystem” and “Hotel Entities Subsystem”. Last and the lowest layer is Data Layer which consist of “Database Management Subsystem” and “Hotel Entities Subsystem”. Each layer can be accessed from the layer above. Following diagram shows the relationship between layers.

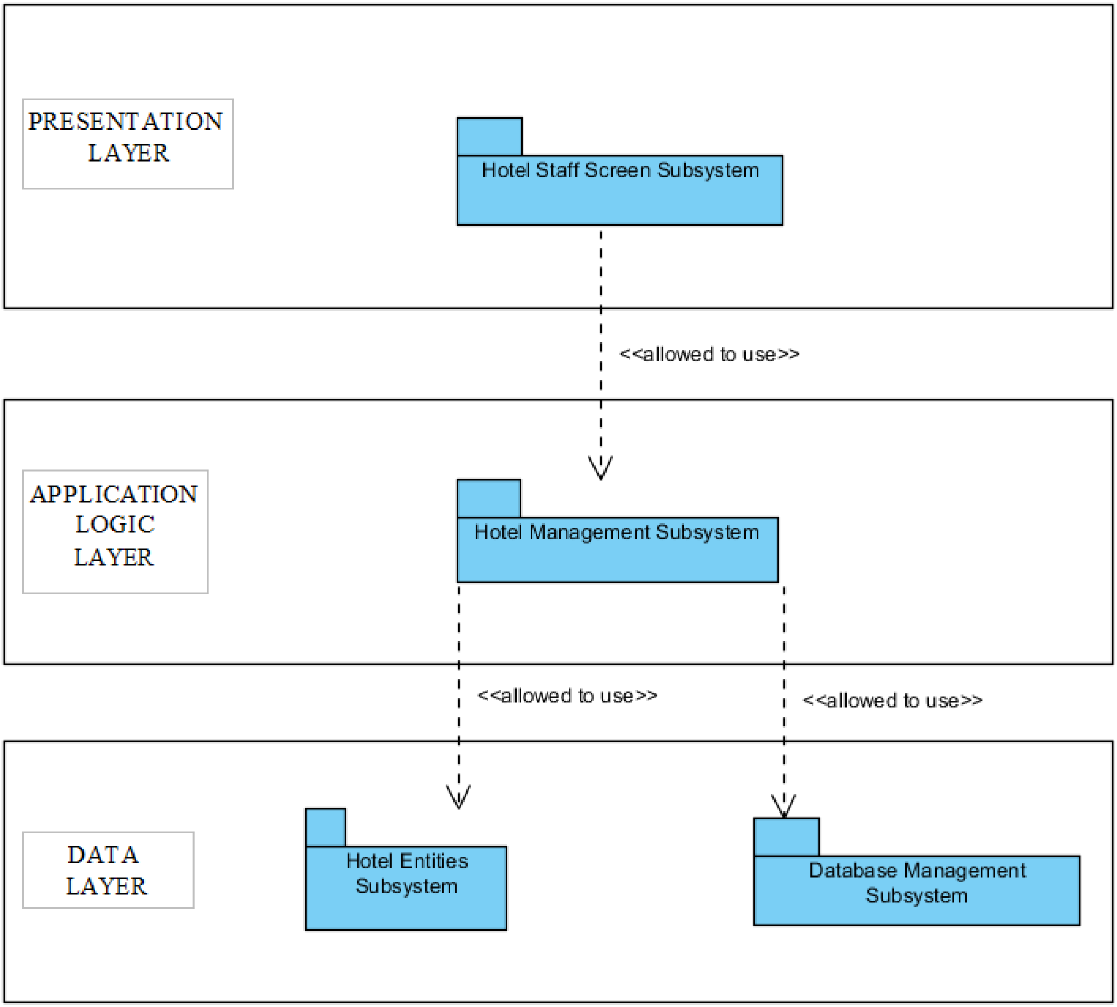


Figure 29 : Layer Pattern

4.4 Hardware/Software Mapping

HotelMaster is implemented in Java, therefore it is platform independent and can be run in any machine that has Java Runtime Environment. Java Runtime Environment can be downloaded from Oracle's website. After that, any Hotel Computer can setup and use HotelMaster. Also, for storage management HotelMaster has a MySQl Database. Database is on a server so there is no need a hardware. Connection between Hotel Computer and Database server established via JDBC (Java Database Connectivity) Driver.

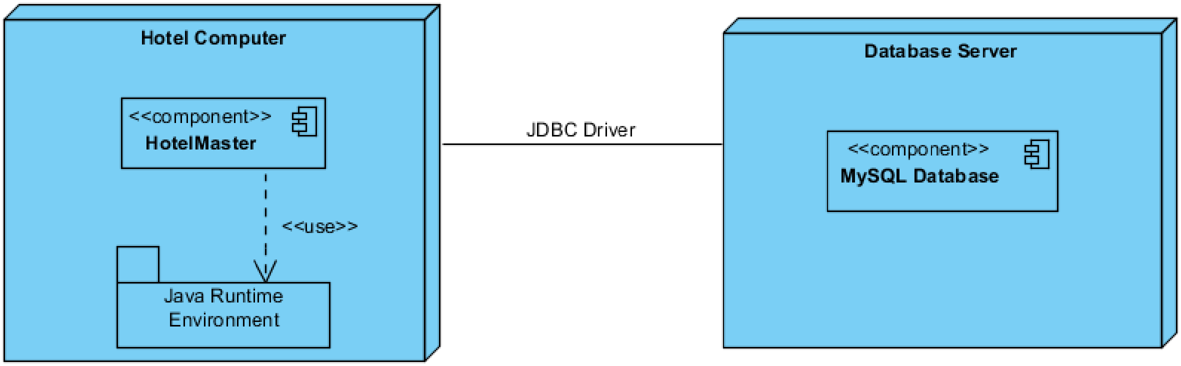


Figure 30 : Deployment Diagram

4.5 Addressing Key Concerns

4.5.1 Persistent Data Management

Data management is one of the significant part of the system. In HotelMaster, so many data are stored such as room, customer, reservation details or information. Persistency should be provided in these data management and we prefer database for storing data and it should be accessible by hotel staff. We use MySQL database system and persistent data storages are available such as login, rooms, customers and reservations storage. In order to begin using the system or login the system, hotel staff must enter his/her name and password so this information is stored in database. In room storage, there is an information table about the rooms such as their types, floor number, daily price, adult and child numbers. In reservation and customer storage, the information of customers' names, phone numbers, purchase type, room numbers, arrival-departure times, total costs are stored. Hotel staff can see these information and implement some tasks concerning about customer requests.

4.5.2 Access Control and Security

In HotelMaster, the only actor is hotel staff and firstly to use the system hotel staff logins. Then, he/she can access every functionality and data in the system. Therefore, hotel staff have the full control and access about the system.

Actors/Class	HotelController

Hotel Staff	addRoomToHotel()
	addCustomerToHotel()
	addReservation()
	deleteRoomFromHotel()
	deleteCustomerFromHotel()
	deleteReservation()

Figure 31 : Access Matrix

4.5.3 Global Software Control

Event driven control is appropriate software control for the HotelMaster. Event driven control flow of a program reacts to events that are external to the program itself. An event occurs and the system waits for it and then it will be dispatched to the proper object. In our system, Model-View-Controller is used as the architectural pattern. In MVC, an event is waited by the system and this event is tapping. When event happens through tapping, system updates and model notifies the view part of the MVC. In view part, there is a “Hotel Staff Screen Subsystem”. This subsystem reads data from the model parts and views are updated.

4.5.4 Boundary Conditions

Initialization: The system can be booted up in any operating system that being able to run Java. After booting up, firstly, the hotel staff must run the executable file of the program. After starting the system, every data concerning about the related user interface has to be accessed. Database access must be also provided because information about rooms, customers and reservations are stored in database.

Termination: There is no situation for the termination of single subsystems because of providing the system continuity. Therefore, rebooting of the whole system is implemented when one subsystem terminates. MySQL is used to provide the database management. The database is kept in the hotel staff’s computer. Appropriate changes are made through the interaction of the system.

Failure: The keeping of data is essential for customers, reservations and rooms. All of these information are available in the database and there will be no loss about these data. If there is a failure such as in communication link, the only thing occurred is like a system initialization.

5. Object Design

5.1 Pattern applications

5.1.1 Facade Pattern

Facade Pattern provides an interface for a subsystem and decrease the coupling between subsystems. In HotelMaster we used Facade Pattern and implemented unified interface classes to decrease coupling between subsystem. We implemented this pattern for two subsystems which are "DatabaseManagement Subsystem" and "HotelManagement Subsytem". "HotelControllerFacade" and "HotelStorageFacade" are the classes provide connection to these subsystems. Other subsystems and classes need to use only these facade classes to access related information.

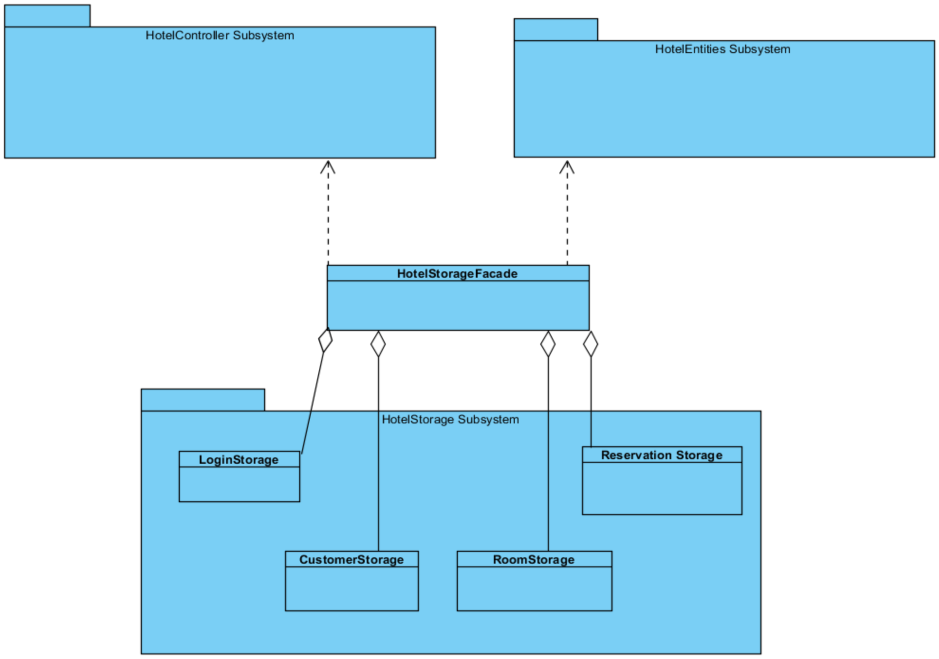


Figure 32: Façade Pattern

5.1.2 Observer Pattern

Main architectural pattern of HotelMaster is Model-View-Controller, therefore there are many model and view objects in the system that needs to be connected to each other. Also, when an model object changed, the related view objects needs to be updated. According to these concerns we applied Observer Pattern for HotelMaster. In HotelMaster, "Hotel" class is subject and "CustomerScreen", "ReservationScreen", RoomScreen classes are observers. Hence, when a change happens in Hotel class, we do not know how many objects will change. Subject class notifies the Observer class and observers are updated by Observer class.

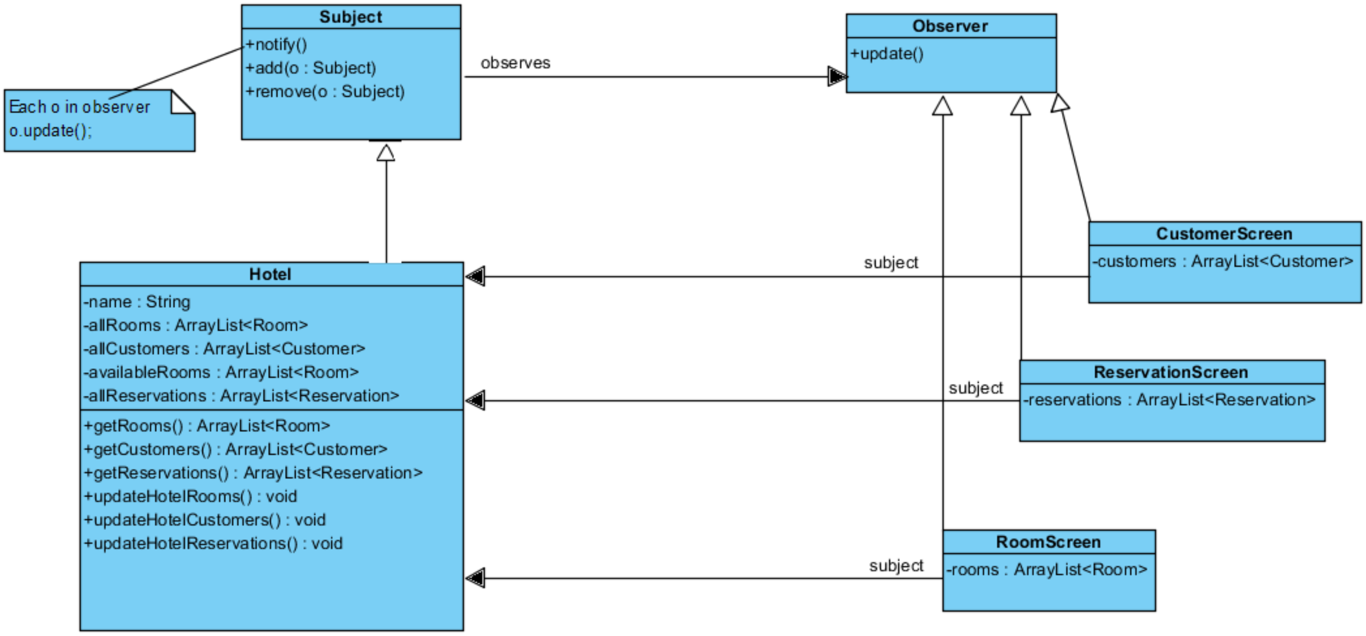

Subject
+notify()
+add(o : Subject)
+remove(o : Subject)
Each o in observer
o.update();
observes
Observer
+update()
CustomerScreen
-customers : ArrayList<Customer>
subject
Hotel
-name : String
-allRooms : ArrayList<Room>
-allCustomers : ArrayList<Customer>
-availableRooms : ArrayList<Room>
-allReservations : ArrayList<Reservation>
+getRooms() : ArrayList<Room>
+getCustomers() : ArrayList<Customer>
+getReservations() : ArrayList<Reservation>
+updateHotelRooms() : void
+updateHotelCustomers() : void
+updateHotelReservations() : void
ReservationScreen
-reservations : ArrayList<Reservation>
subject
RoomScreen
-rooms : ArrayList<Room>
subject

Figure 33: Observer Pattern

5.1.3 Adapter Pattern

In HotelMaster, we need DatePickers in several places in the system. User should be able to select the arrival and departure date of the customer. Also, implementing a DatePicker from the beginning is not an easy task. Additionally, the libraries that we found are not fully suitalbe for our system. Therefore, we used a Java DatePicker library which is a Legacy class and implemented our "HotelDatePicker" class which is Adapter class. This DatePicker class appears in the check-in screen and allows to choose arrival and departure dates.

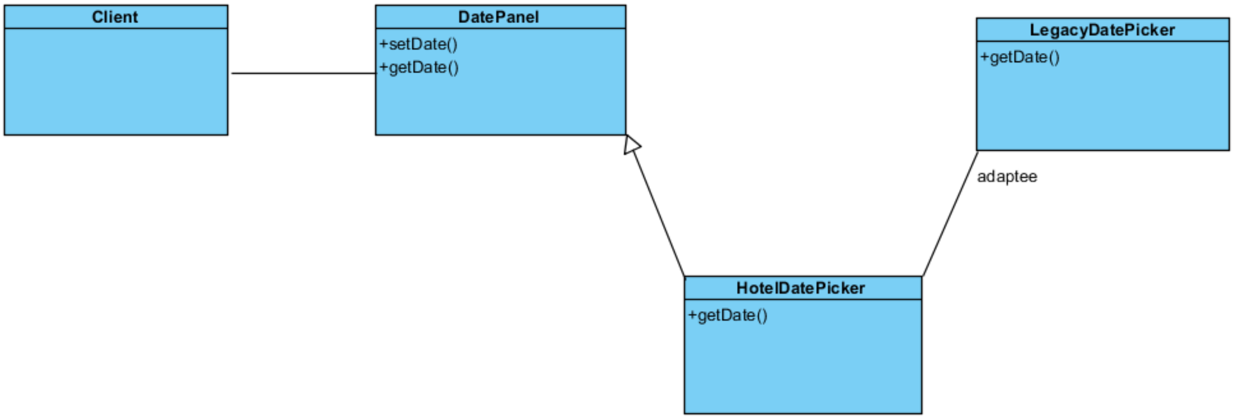


Figure 34: Adapter Pattern

5.1.4 Composite Pattern

In HotelMaster, the Views compose of many layers. Each Screen class has a background panel which is a Component and this panel includes many panels. Therefore, the Screen classes which are actually panels are inherently uses the composite pattern.

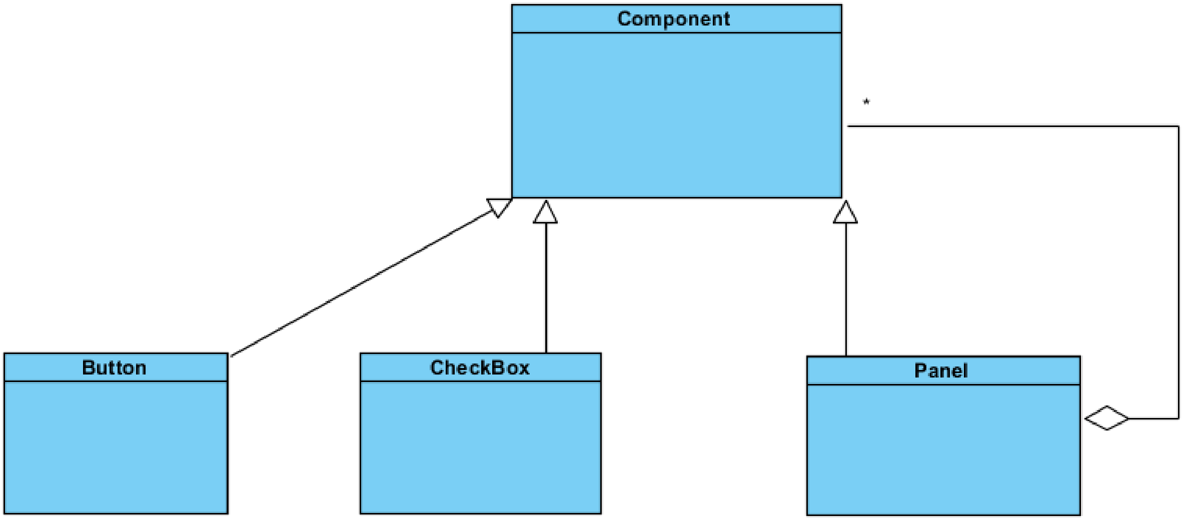


Figure 35: Composite Pattern

5.2 Class Interfaces

5.2.1 Management Package

CustomerController

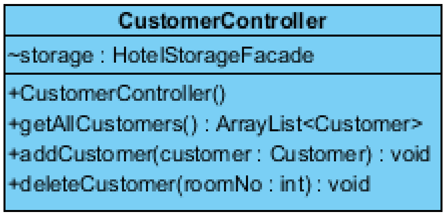

CustomerController
~storage : HotelStorageFacade
+CustomerController()
+getAllCustomers() : ArrayList<Customer>
+addCustomer(customer : Customer) : void
+deleteCustomer(roomNo : int) : void

Figure 36: CustomerController Class

In CustomerController Class, there are one private property, storage(HotelStorage). It has three public methods and a constructor : CustomerController(), getAllCustomers(), addCustomer(customer: Customer) and deleteCustomer(roomNo:int).

HotelControllerFacade

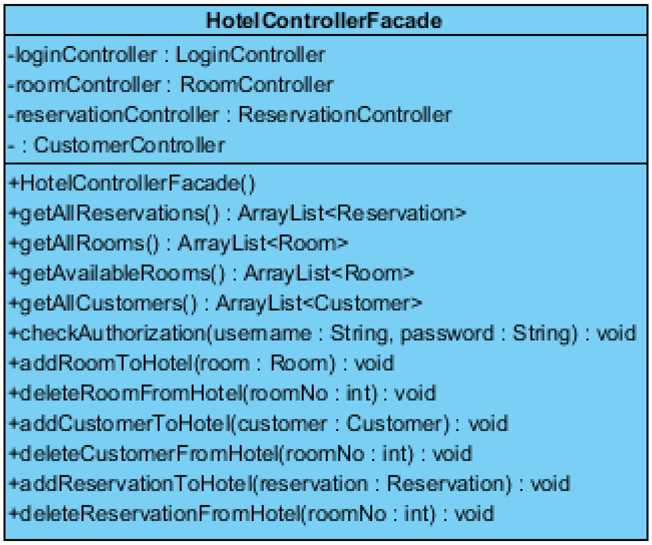

HotelControllerFacade
-loginController : LoginController
-roomController : RoomController
-reservationController : ReservationController
- : CustomerController
+HotelControllerFacade()
+getAllReservations() : ArrayList<Reservation>
+getAllRooms() : ArrayList<Room>
+getAvailableRooms() : ArrayList<Room>
+getAllCustomers() : ArrayList<Customer>
+checkAuthorization(username : String, password : String) : void
+addRoomToHotel(room : Room) : void
+deleteRoomFromHotel(roomNo : int) : void
+addCustomerToHotel(customer : Customer) : void
+deleteCustomerFromHotel(roomNo : int) : void
+addReservationToHotel(reservation : Reservation) : void
+deleteReservationFromHotel(roomNo : int) : void

Figure 37: HotelController Class

In HotelControllerFacade Class, it has four private properties such as loginController, roomController, customerController, reservationController properties. This class has one public constructor HotelController() and eleven public methods which are getAllReservations(), getAllRooms(), getAvailableRooms(), getAllCustomers(), checkAuthorization(username:

String, password : String), addRoomToHotel(room:Room), deleteRoomFromHotel(roomNo: int), addCustomerToHotel(customer:Customer), deleteCustomerFromHotel(roomNo: int), addReservationToHotel(reservation: Reservation), deleteReservationFromHotel(roomNo : int) methods.

LoginController

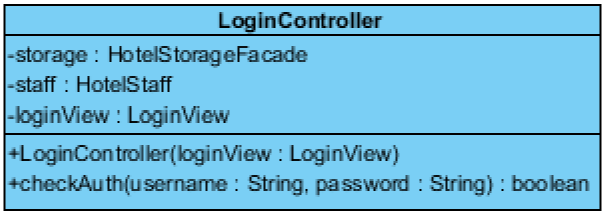

LoginController
-storage : HotelStorageFacade
-staff : HotelStaff
-loginView : LoginView
+LoginController(loginView : LoginView)
+checkAuth(username : String, password : String) : boolean

Figure 38 : LoginController Class

In LoginController Class, it has three private properties such as storage(HotelStorageFacade), staff(HotelStaff) and loginView(LoginView).This class has one public constructor LoginController and one public method which is checkAuth(username:String , password:String).

ReservationController

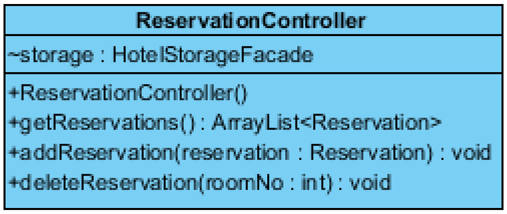

ReservationController
~storage : HotelStorageFacade
+ReservationController()
+getReservations() : ArrayList<Reservation>
+addReservation(reservation : Reservation) : void
+deleteReservation(roomNo : int) : void

Figure 39: ReservationController Class

In ReservationController Class, it has one private property such as storage(HotelStorageFacade). This class has one public constructor method ReservationController() and three public methods which are getReservation(), addReservation(reservation:Reservation), deleteReservation(roomNo:int).

RoomController

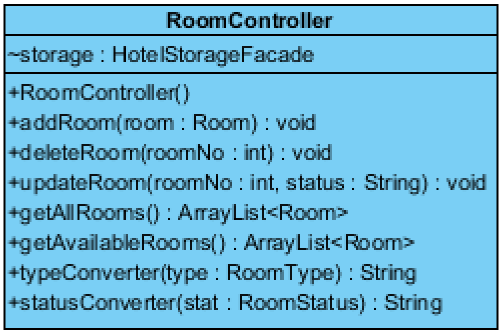

RoomController
~storage : HotelStorageFacade
+RoomController()
+addRoom(room : Room) : void
+deleteRoom(roomNo : int) : void
+updateRoom(roomNo : int, status : String) : void
+getAllRooms() : ArrayList<Room>
+getAvailableRooms() : ArrayList<Room>
+typeConverter(type : RoomType) : String
+statusConverter(stat : RoomStatus) : String

Figure 40: RoomController Class

In RoomController Class, it has one private property such as storage(HotelStorageFacade).This class has one public constructor RoomController() and seven public methods which are getAllRooms(), addRoom(room:Room), deleteRoom(roomNo:int), typeConverter(type: RoomType), statusConverter(stat:RoomStatus), updateRoom(roomNo:int, status:String), getAvailableRooms().

CustomerListController

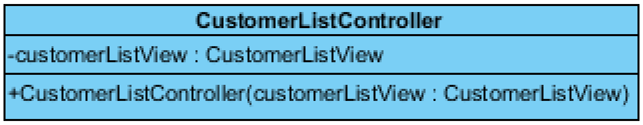

CustomerListController
-customerListView : CustomerListView
+CustomerListController(customerListView : CustomerListView)

Figure 41: CustomerListController Class

In CustomerListController Class there is one private property which is customerListView(CustomerListView) and one public constructor CustomerListController(CustomerListView customerListView).

MenuController

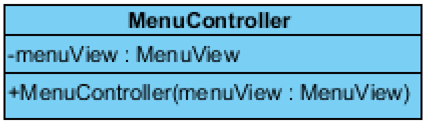


Figure 42: Menucontroller Class

In MenuController Class, there is one private property which is menuView(MenuVIew) and one public constructor which is MenuController(menuView:Menuview).

CheckInController

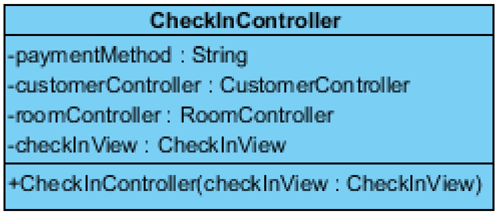

CheckInController
-paymentMethod : String
-customerController : CustomerController
-roomController : RoomController
-checkInView : CheckInView
+CheckInController(checkInView : CheckInView)

Figure 43: CheckInController Class

CheckInController Class has four private properties which are paymentMethod(String), customerController(CustomerController), roomController(RoomController), checkInView(CheckInview). Also it has one public constructor that is CheckInController(checkInView:CheckInView).

RoomEditorController

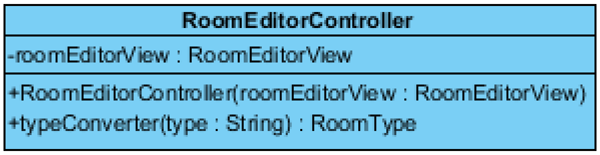

RoomEditorController
-roomEditorView : RoomEditorView
+RoomEditorController(roomEditorView : RoomEditorView)
+typeConverter(type : String) : RoomType

Figure 44: RoomEditorController

RoomEditorController class has one private property which is roomEditorView(RoomeditorView). Also, it has one public constructor and one method which are RoomEditorController(roomEditorView:RoomeditorView), typeConverter(type:String)

RoomListController

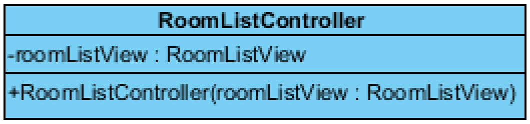

RoomListController
-roomListView : RoomListView
+RoomListController(roomListView : RoomListView)

Figure 45: RoomListController

RoomListController class has one private property which is roomListView(roomListView) and it has one public constructor RoomListController(roomListView:RoomListView)

5.2.2 Screens Package

CheckBoxTableCell

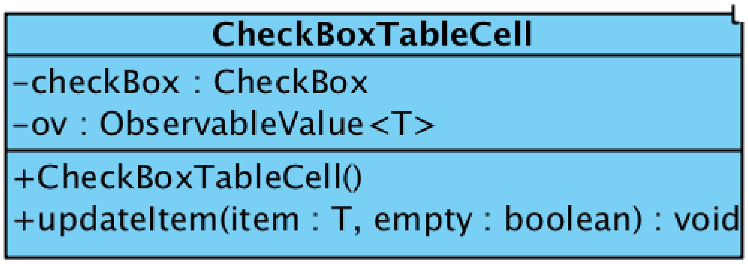
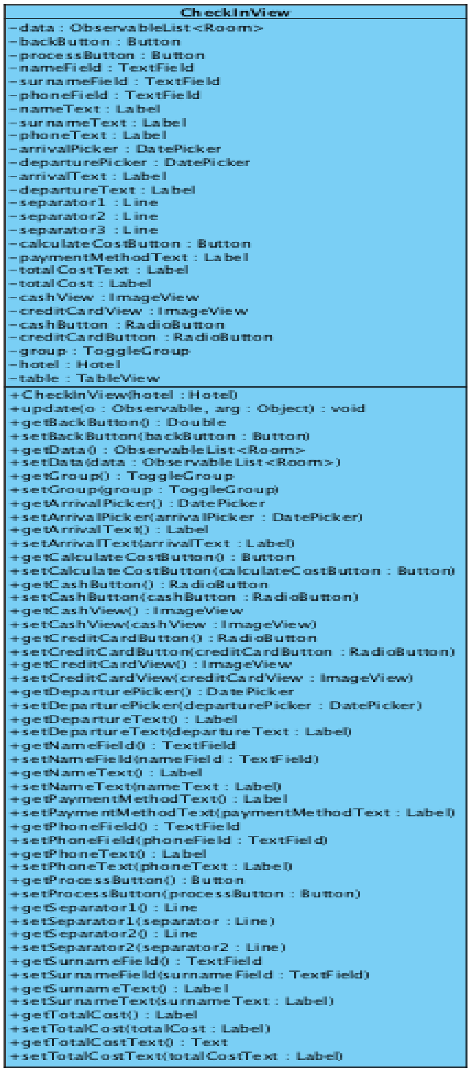


Figure 46: CheckBoxTableCell Class

CheckBoxTable class has two private variable which are checkBox : checkBox and ov : ObservableValue<T>. Also, this class includes two public methods which are CheckBoxTableCell() and updateItem(item : T, empty : boolean) : void.

CheckInView



CheckInView
-data : ObservableList<Room>
-backButton : Button
-processButton : Button
-nameField : TextField
-surnameField : TextField
-phoneField : TextField
-nameText : Label
-surnameText : Label
-phoneText : Label
-arrivalPicker : DatePicker
-departurePicker : DatePicker
-arrivalText : Label
-departureText : Label
-separator1 : Line
-separator2 : Line
-separator3 : Line
-calculateCostButton : Button
-paymentMethodText : Label
-totalCostText : Label
-totalCost : Label
-cashView : ImageView
-creditCardView : ImageView
-cashButton : RadioButton
-creditCardButton : RadioButton
-group : ToggleGroup
-hotel : Hotel
-table : TableView
+CheckInView(hotel : Hotel)
+update(o : Observable, arg : Object) : void
+getBackButton() : Double
+setBackButton(backButton : Button)
+getData() : ObservableList<Room>
+setData(data : ObservableList<Room>)
+getGroup() : ToggleGroup
+setGroup(group : ToggleGroup)
+getArrivalPicker() : DatePicker
+setArrivalPicker(arrivalPicker : DatePicker)
+getArrivalText() : Label
+setArrivalText(arrivalText : Label)
+getCalculateCostButton() : Button
+setCalculateCostButton(calculateCostButton : Button)
+getCashButton() : RadioButton
+setCashButton(cashButton : RadioButton)
+getCashView() : ImageView
+setCashView(cashView : ImageView)
+getCreditCardButton() : RadioButton
+setCreditCardButton(creditCardButton : RadioButton)
+getCreditCardView() : ImageView
+setCreditCardView(creditCardView : ImageView)
+getDeparturePicker() : DatePicker
+setDeparturePicker(departurePicker : DatePicker)
+getDepartureText() : Label
+setDepartureText(departureText : Label)
+getNameField() : TextField
+setNameField(nameField : TextField)
+getNameText() : Label
+setNameText(nameText : Label)
+getPaymentMethodText() : Label
+setPaymentMethodText(paymentMethodText : Label)
+getPhoneField() : TextField
+setPhoneField(phoneField : TextField)
+getPhoneText() : Label
+setPhoneText(phoneText : Label)
+getProcessButton() : Button
+setProcessButton(processButton : Button)
+getSeparator1() : Line
+setSeparator1(separator : Line)
+getSeparator2() : Line
+setSeparator2(separator2 : Line)
+getSurnameField() : TextField
+setSurnameField(surnameField : TextField)
+getSurnameText() : Label
+setSurnameText(surnameText : Label)
+getTotalCost() : Label
+setTotalCost(totalCost : Label)
+getTotalCostText() : Text
+setTotalCostText(totalCostText : Label)

Figure 47: CheckInView Class

CheckInView class lots of private properties and public methods. To start with, in private properties section, this class includes ObservableList<Room> data, backButton, processButton, nameField, surnameField, phoneField, nameText, surnameText, phoneText, arrivalPicker, departurePicker, arrivalText, departureText,separator1,separator2 and separator3. Also, this class includes calculateCostButton, paymentMethodText, totalCostText, totalCost, cashView, creditCardView, cashButton, creditCardButton, group, hotel, table. Finally, these private properties have their own getters and setters with public visibility. Also, this class includes one public constructor and update() method.

CustomerListView

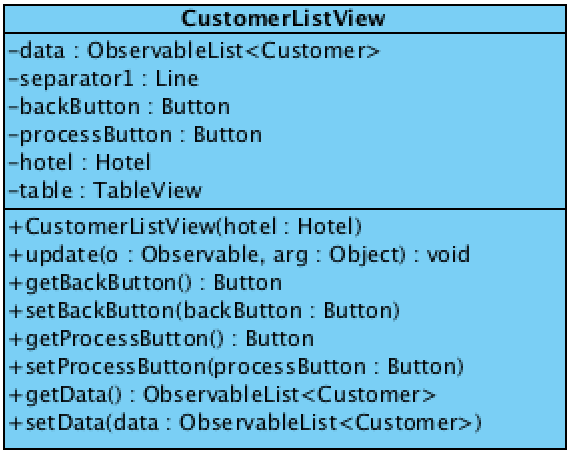

CustomerListView
-data : ObservableList<Customer>
-separator1 : Line
-backButton : Button
-processButton : Button
-hotel : Hotel
-table : TableView
+CustomerListView(hotel : Hotel)
+update(o : Observable, arg : Object) : void
+getBackButton() : Button
+setBackButton(backButton : Button)
+getProcessButton() : Button
+setProcessButton(processButton : Button)
+getData() : ObservableList<Customer>
+setData(data : ObservableList<Customer>)

Figure 48: CustomerListView Class

CustomerListView class have six private properties which are data, separator1, backButton, processButton, hotel and table. Also this class have several methods which are in public visibility. They are called update() : void, getBackButton() : Button, setBackButton(backButton : Button), getProcessButton(), setProcessButton(processButton : Button), getData(), setData() and this class include one public constructor CustomerListView(hotel : Hotel)

LoginView

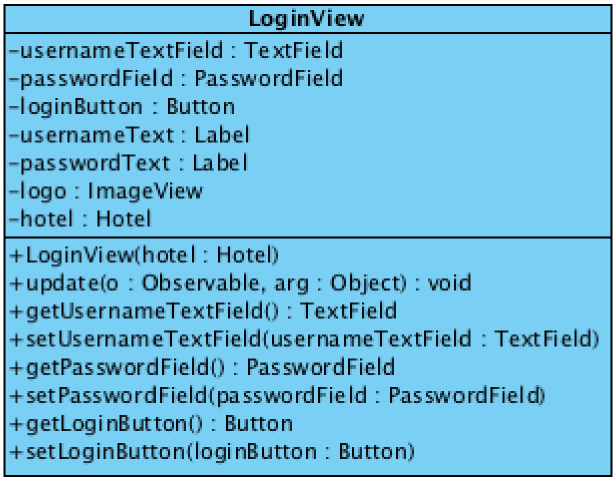

LoginView
-usernameTextField : TextField
-passwordField : PasswordField
-loginButton : Button
-usernameText : Label
-passwordText : Label
-logo : ImageView
-hotel : Hotel
+LoginView(hotel : Hotel)
+update(o : Observable, arg : Object) : void
+getUsernameTextField() : TextField
+setUsernameTextField(usernameTextField : TextField)
+getPasswordField() : PasswordField
+setPasswordField(passwordField : PasswordField)
+getLoginButton() : Button
+setLoginButton(loginButton : Button)

Figure 49: LoginView Class

LoginView class have several private variable which are usernameTextField, passwordField, loginButton, usernameText, passwordText, logo and hotel. Also this class includes several public methods. These are LoginView(hotel : Hotel), update(o : Observable, arg : Object), getUsernameTextField(), setUsernameTextField(), getPasswordField(), setPasswordField(), getLoginButton(), setLoginButton().

MenuView

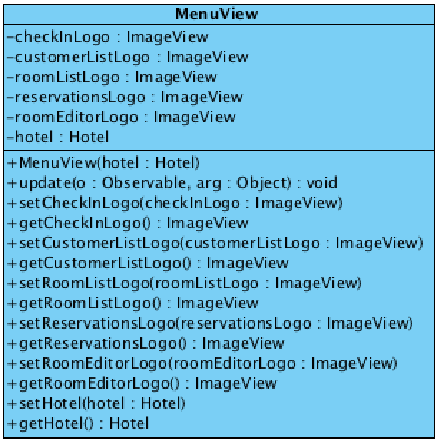
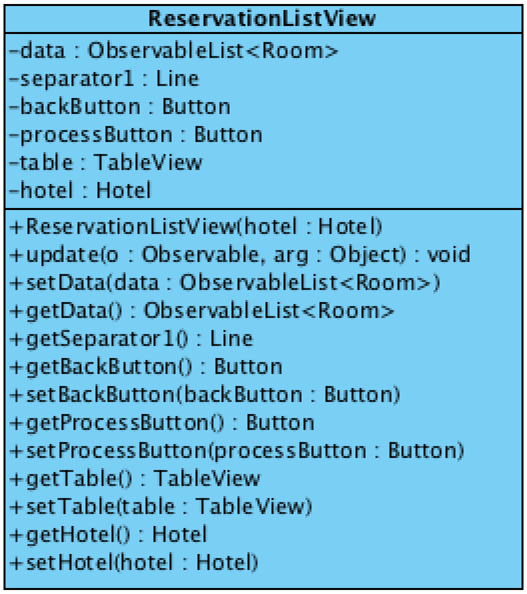

MenuView
-checkInLogo : ImageView
-customerListLogo : ImageView
-roomListLogo : ImageView
-reservationsLogo : ImageView
-roomEditorLogo : ImageView
-hotel : Hotel
+MenuView(hotel : Hotel)
+update(o : Observable, arg : Object) : void
+setCheckInLogo(checkInLogo : ImageView)
+getCheckInLogo() : ImageView
+setCustomerListLogo(customerListLogo : ImageView)
+getCustomerListLogo() : ImageView
+setRoomListLogo(roomListLogo : ImageView)
+getRoomListLogo() : ImageView
+setReservationsLogo(reservationsLogo : ImageView)
+getReservationsLogo() : ImageView
+setRoomEditorLogo(roomEditorLogo : ImageView)
+getRoomEditorLogo() : ImageView
+setHotel(hotel : Hotel)
+getHotel() : Hotel

Figure 50: MenuView Class

MenuView class has six private properties and several methods with public visibilities. To continue with, these six private properties are checkInLogo, customerListLogo, roomListLogo, reservationsLogo, roomEditorLogo and hotel. In order to continue with methods, this class include one public constructor MenuView() and update() methods. Also, this class have getters and setters of private properties. These are, setCheckInLogo(checkInLogo : ImageView), getCheckInLogo(), setCustomerListLogo(customerListLogo : ImageView), getCustomerListLogo(), setRoomListLogo(customerListLogo : ImageView), getRoomListLogo(), setReservationsLogo(), getReservationsLogo(), setRoomEditorLogo(roomEditorLogo : ImageView), getRoomEditorLogo(), setHotel(hotel : Hotel) and getHotel().

ReservationListView



```
ReservationListView
-data : ObservableList<Room>
-separator1 : Line
-backButton : Button
-processButton : Button
-table : TableView
-hotel : Hotel
+ReservationListView(hotel : Hotel)
+update(o : Observable, arg : Object) : void
+setData(data : ObservableList<Room>)
+getData() : ObservableList<Room>
+getSeparator1() : Line
+getBackButton() : Button
+setBackButton(backButton : Button)
+getProcessButton() : Button
+setProcessButton(processButton : Button)
+getTable() : TableView
+setTable(table : TableView)
+getHotel() : Hotel
+setHotel(hotel : Hotel)
```

Figure 51: ReservationListView

ReservationListView class has six private properties and several methods with public visibilities. To continue with, these six private properties are data, seperator1, backButton, processButton, table and hotel. In order to continue with, this class has one public constructor ReservationListView() and update() methods and some setters and getters. These are setData(data : ObservableList<Room>), getData(), getSeparator(), getBackButon(), setBackButton(backButton : Button), getProcessButton(), setProcessButton(processButton : Button), getTable(), setTable(table : TableView), getHotel() and setHotel(hotel : Hotel).

RoomEditorView

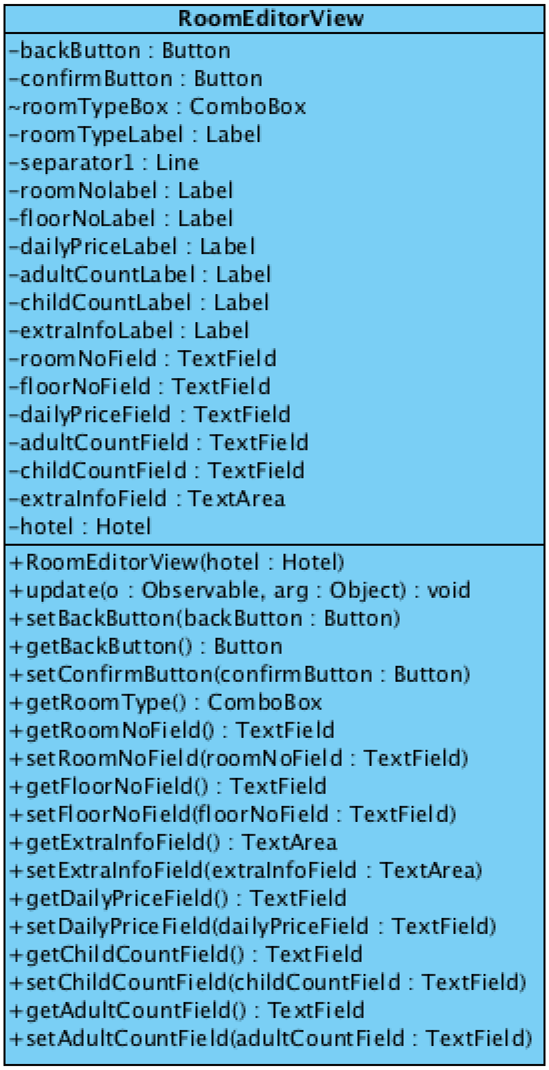

RoomEditorView
-backButton : Button
-confirmButton : Button
~roomTypeBox : ComboBox
-roomTypeLabel : Label
-separator1 : Line
-roomNolabel : Label
-floorNoLabel : Label
-dailyPriceLabel : Label
-adultCountLabel : Label
-childCountLabel : Label
-extraInfoLabel : Label
-roomNoField : TextField
-floorNoField : TextField
-dailyPriceField : TextField
-adultCountField : TextField
-childCountField : TextField
-extraInfoField : TextArea
-hotel : Hotel
+RoomEditorView(hotel : Hotel)
+update(o : Observable, arg : Object) : void
+setBackButton(backButton : Button)
+getBackButton() : Button
+setConfirmButton(confirmButton : Button)
+getRoomType() : ComboBox
+getRoomNoField() : TextField
+setRoomNoField(roomNoField : TextField)
+getFloorNoField() : TextField
+setFloorNoField(floorNoField : TextField)
+getExtraInfoField() : TextArea
+setExtraInfoField(extraInfoField : TextArea)
+getDailyPriceField() : TextField
+setDailyPriceField(dailyPriceField : TextField)
+getChildCountField() : TextField
+setChildCountField(childCountField : TextField)
+getAdultCountField() : TextField
+setAdultCountField(adultCountField : TextField)

Figure 52: RoomEditorView Class

RoomEditorView class includes lots of private variables and lots of methods which are in public visibilities. These are backButton, confirmButton, roomTypeBox, roomTypeLabel, separator1, roomNoLabel, floorNoLabel, dailyPriceLabel, adultCountLabel, childCountLabel, extraInfoLabel, roomNoField, floorNoField, dailyPriceField, adultCountField, childCountField, extraInfoField and hotel objects. Also this class include RoomEditorView(hotel : Hotel) as its public constructor. In order to continue, RoomEditorView class has update() method, too. Finally, RoomEditorView class include several setters and getters. These are setBackButton(backButton : Button), getBackButton(), setConfirmButton(confirmButton : Button), getRoomType(), getRoomNoField(), setRoomNoField(floorNoField : TextField), getFloorNoField(), setFloorNoField(floorNoField : TextField), setExtraInfoField(extraInfoField : TextArea), getDailyPriceField(), setChildCountField(childCountField : TextField), getAdultCountField(), setAdultCountField(adultCountField : TextField).

RoomListView

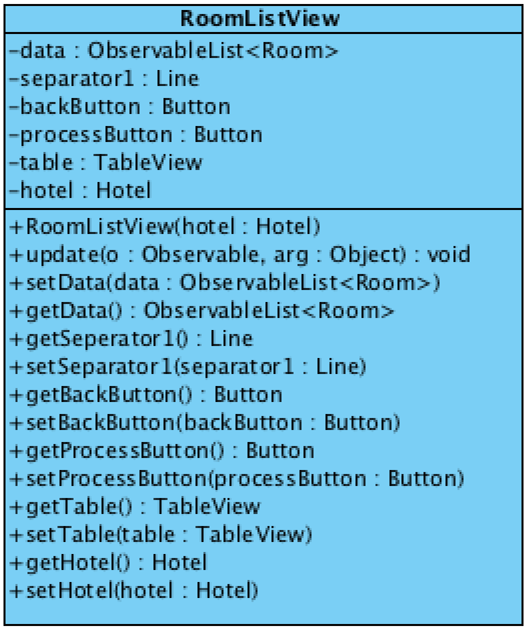

RoomListView
-data : ObservableList<Room>
-separator1 : Line
-backButton : Button
-processButton : Button
-table : TableView
-hotel : Hotel
+RoomListView(hotel : Hotel)
+update(o : Observable, arg : Object) : void
+setData(data : ObservableList<Room>)
+getData() : ObservableList<Room>
+getSeperator1() : Line
+setSeparator1(separator1 : Line)
+getBackButton() : Button
+setBackButton(backButton : Button)
+getProcessButton() : Button
+setProcessButton(processButton : Button)
+getTable() : TableView
+setTable(table : TableView)
+getHotel() : Hotel
+setHotel(hotel : Hotel)

Figure 53: RoomListView Class

RoomListView class include six private properties and several public methods. Data, separator1, backButton, processButton, table and hotel are private properties of RoomListView class. In order to continue, RoomListView class include its public constructor RoomListView() and update() methods. Finally, RoomListView class include several setters and getters. These are getData(), setData(data : ObservableList<Room>), getSeparator1(), getBackButton(), setBackButton(backButton : Button), getProcessButton(), setProcessButton(processButton : Button), getTable(), setTable(table : TableView), getHotel() and setHotel(hotel : Hotel).

Welcome Screen

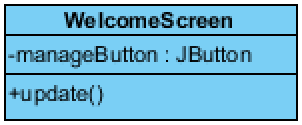
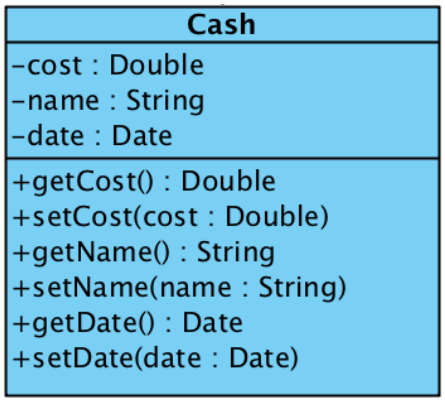


Figure 54: WelcomeScreen Class

In WelcomeScreen Class, it has one private property which is manageButton.This class has one public method which is update().

5.2.3 Entities Package

Cash



```
Cash
-cost : Double
-name : String
-date : Date
+getCost() : Double
+setCost(cost : Double)
+getName() : String
+setName(name : String)
+getDate() : Date
+setDate(date : Date)
```

Figure 55: Cash Class

In Cash class, there are three private property such as cost(Double), name(String), and date(Date). It has getCost(), setCost(cost: Double), getName(), setName(name : String), getDate(), setDate(date : Date) methods.

CreditCard

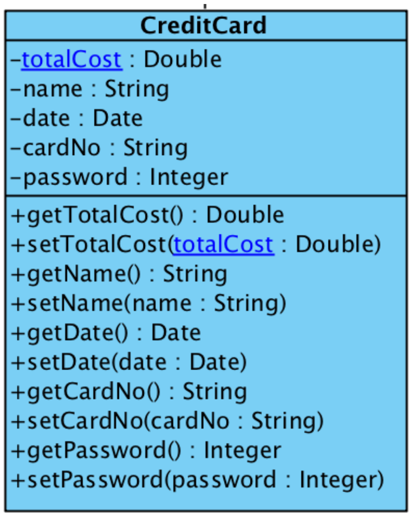


Figure 56: CreditCard Class

In CreditCard class, there are five private property such as totalCost(Double), name(String), date(Date), cardNo(String), password(Integer). It has several public methods which are getTotalCost(), setTotalCost(totalCost : Double), getName(), setName(name : String), getDate(), setDate(date : Date), getCardNo(), setCardNo(cardNo : String), getPassword(), setPassword(password : Integer).

Customer

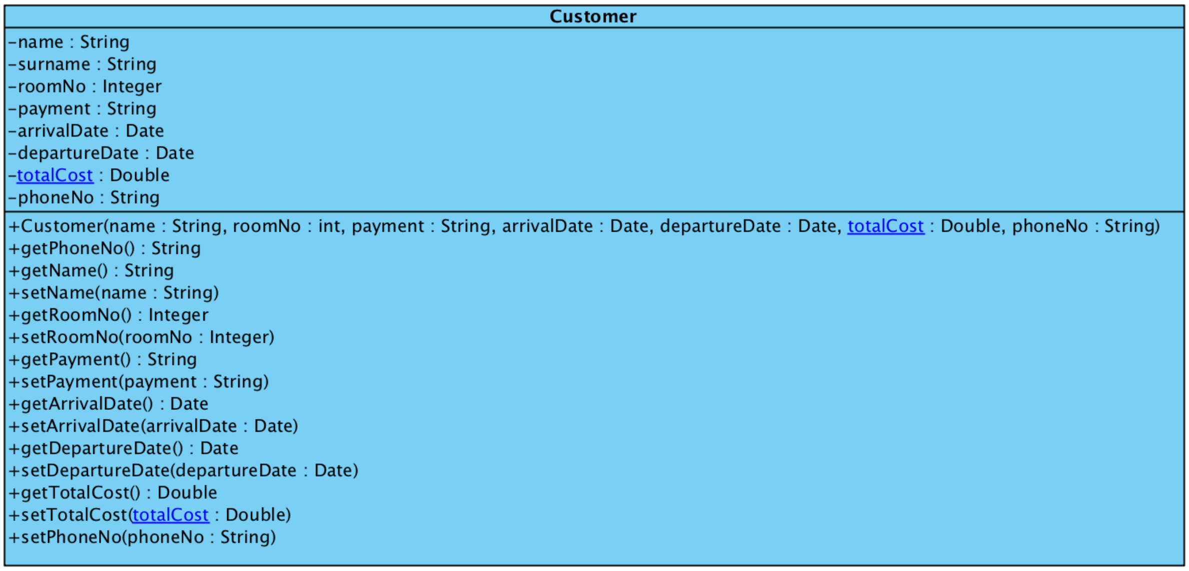

Customer
-name : String
-surname : String
-roomNo : Integer
-payment : String
-arrivalDate : Date
-departureDate : Date
-totalCost : Double
-phoneNo : String
+Customer(name : String, roomNo : int, payment : String, arrivalDate : Date, departureDate : Date, totalCost : Double, phoneNo : String)
+getPhoneNo() : String
+getName() : String
+setName(name : String)
+getRoomNo() : Integer
+setRoomNo(roomNo : Integer)
+getPayment() : String
+setPayment(payment : String)
+getArrivalDate() : Date
+setArrivalDate(arrivalDate : Date)
+getDepartureDate() : Date
+setDepartureDate(departureDate : Date)
+getTotalCost() : Double
+setTotalCost(totalCost : Double)
+setPhoneNo(phoneNo : String)

Figure 57: Customer Class

Customer class has seven private properties. These are name(String), surname(String), roomNo(Integer), payment(String), arrivalDate(Date), departureDate(Date), totalCost(Double), phoneNo(String). This class has also one public constructor and several public methods such as getPhoneNo(), getName(), setName(name : String), getRoomNo(), setRoomNo(roomNo : Integer), getPayment(), setPayment(payment : String), getArrivalDate(), setArrivalDate(arrivalDate : Date), getDepartureDate(), setDepartureDate(departureDate : Date), getTotalCost(), setTotalCost(totalCost : Double), setPhoneNo(phoneNo : String).

Hotel

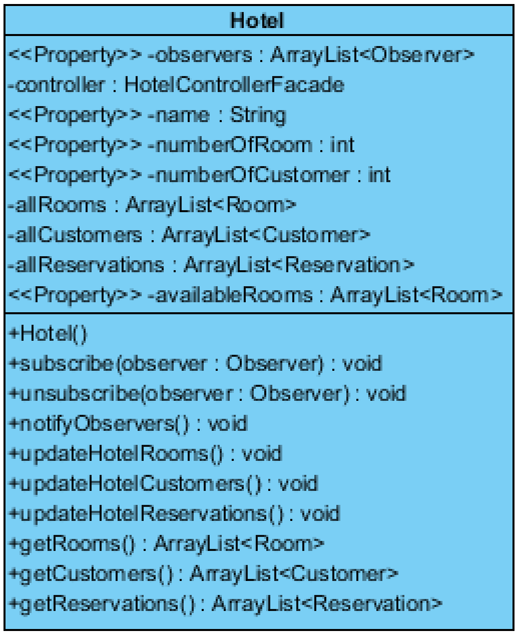

Hotel
<<Property>> -observers : ArrayList<Observer>
-controller : HotelControllerFacade
<<Property>> -name : String
<<Property>> -numberOfRoom : int
<<Property>> -numberOfCustomer : int
-allRooms : ArrayList<Room>
-allCustomers : ArrayList<Customer>
-allReservations : ArrayList<Reservation>
<<Property>> -availableRooms : ArrayList<Room>
+Hotel()
+subscribe(observer : Observer) : void
+unsubscribe(observer : Observer) : void
+notifyObservers() : void
+updateHotelRooms() : void
+updateHotelCustomers() : void
+updateHotelReservations() : void
+getRooms() : ArrayList<Room>
+getCustomers() : ArrayList<Customer>
+getReservations() : ArrayList<Reservation>

Figure 58: Hotel Class

Hotel class has nine private properties. These are controller(HotelController), name(String), numberOfRoom(Integer), numberOfCustomer(Integer), allRooms(ArrayList<Room>), allReservatinos(ArrayList<Reservation>), availableRooms(ArrayList<Room>), allCustomers(Customer), observers(ArrayList<Observer>). This class has also one public constructor and several public methods such as updateHotelRooms(), updateHotelCustomers(), updateHotelReservations(), getRooms(), getCustomers(), getReservations(), addAvailableRooms(), getName(), setName(name:String), getNumberOfRoom(), setNumberOfCustomer(numberOfCustomer : Integer), subscribe(observer : Observer), unsubscribe(observer : Observer), notifyObservers().

HotelStaff

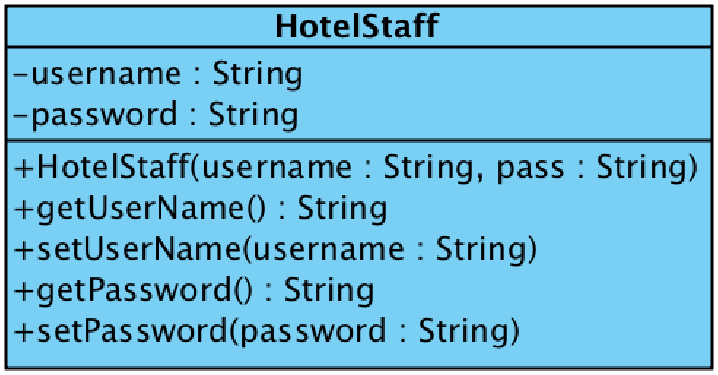

HotelStaff
-username : String
-password : String
+HotelStaff(username : String, pass : String)
+getUserName() : String
+setUserName(username : String)
+getPassword() : String
+setPassword(password : String)

Figure 59: HotelStaff Class

HotelStaff class has two private properties. These are username(String) and password(String). This class has also one public constructor and four public methods such as getUserName(), setUserName(userName : String), getPassword(), setPassword(password : String).

Payment

Figure 60: Payment Class

Payment class is an abstract class which CreditCard class and Cash class extends it.

Reservation

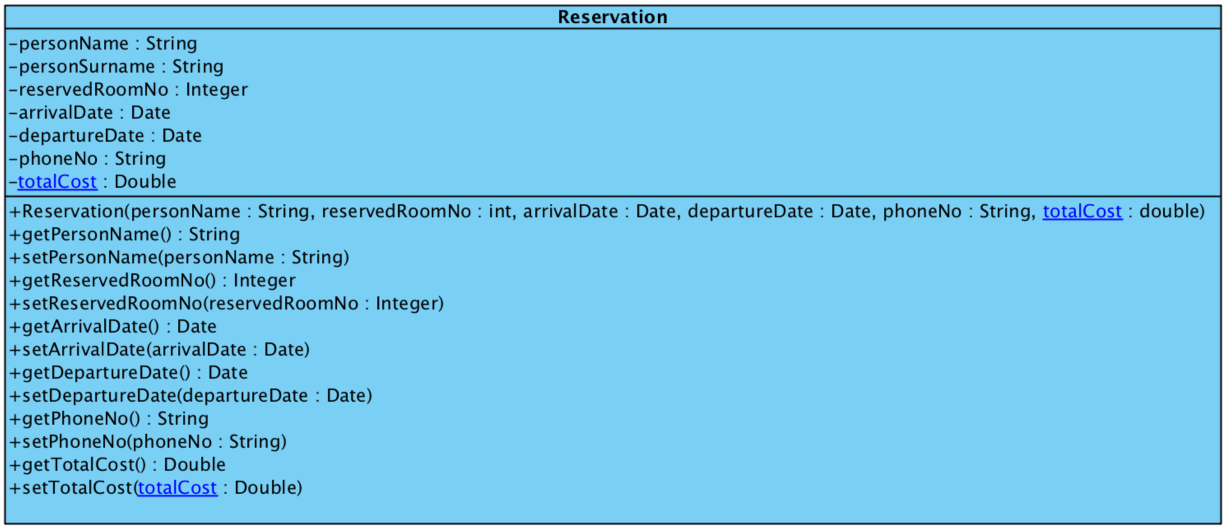
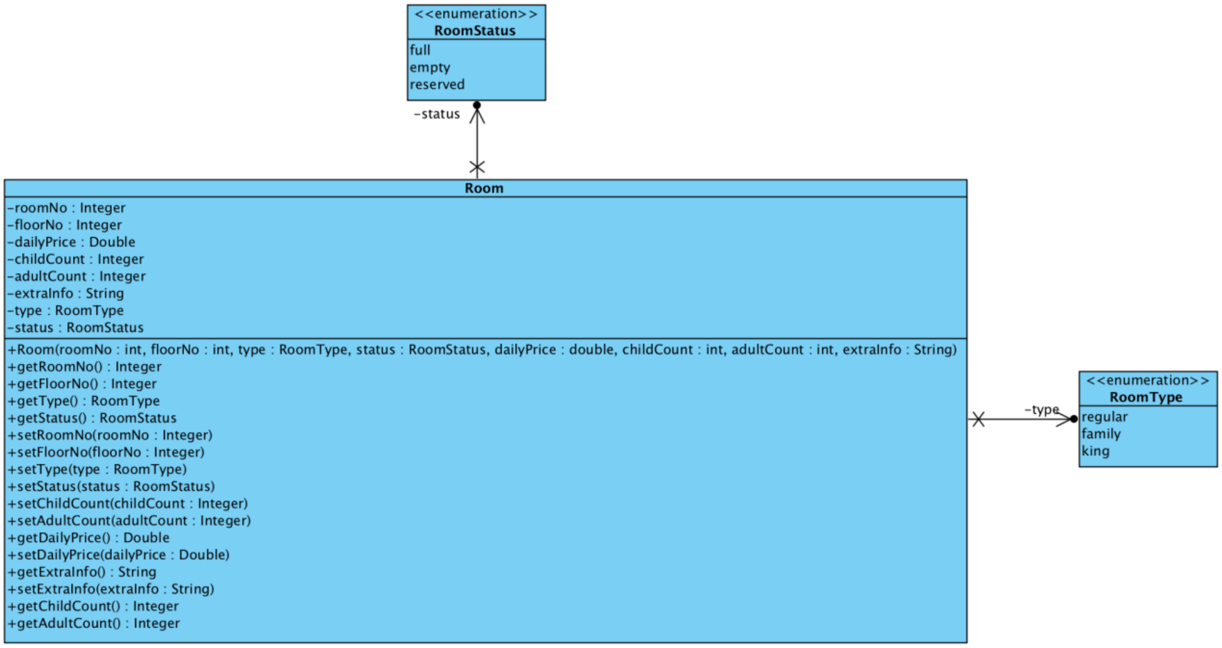

Reservation
-personName : String
-personSurname : String
-reservedRoomNo : Integer
-arrivalDate : Date
-departureDate : Date
-phoneNo : String
-totalCost : Double
+Reservation(personName : String, reservedRoomNo : int, arrivalDate : Date, departureDate : Date, phoneNo : String, totalCost : double)
+getPersonName() : String
+setPersonName(personName : String)
+getReservedRoomNo() : Integer
+setReservedRoomNo(reservedRoomNo : Integer)
+getArrivalDate() : Date
+setArrivalDate(arrivalDate : Date)
+getDepartureDate() : Date
+setDepartureDate(departureDate : Date)
+getPhoneNo() : String
+setPhoneNo(phoneNo : String)
+getTotalCost() : Double
+setTotalCost(totalCost : Double)

Figure 61: Reservation Class

Reservation class has seven private properties. These are personName(String), personSurname(String), reservedRoomNo(Integer), arrivalDate(Date), departureDate(Date), phoneNo(String), totalCost(Double). This class has also one public constructor and several public methods such as getPersonName(), setPersonName(personName : String), getReservedRoomNo(), setReservedRoomNo(reservedRoomNo : Integer), getArrivalDate(), setArrivalDate(arrivalDate : Date), getDepartureDate(), setDepartureDate(departureDate : Date), getPhoneNo(), setPhoneNo(phoneNo : String), getTotalCost(), setTotalCost(totalCost : Double).

Room



<<enumeration>>
RoomStatus
full
empty
reserved
-status
Room
-roomNo : Integer
-floorNo : Integer
-dailyPrice : Double
-childCount : Integer
-adultCount : Integer
-extraInfo : String
-type : RoomType
-status : RoomStatus
+Room(roomNo : int, floorNo : int, type : RoomType, status : RoomStatus, dailyPrice : double, childCount : int, adultCount : int, extraInfo : String)
+getRoomNo() : Integer
+getFloorNo() : Integer
+getType() : RoomType
+getStatus() : RoomStatus
+setRoomNo(roomNo : Integer)
+setFloorNo(floorNo : Integer)
+setType(type : RoomType)
+setStatus(status : RoomStatus)
+setChildCount(childCount : Integer)
+setAdultCount(adultCount : Integer)
+getDailyPrice() : Double
+setDailyPrice(dailyPrice : Double)
+getExtraInfo() : String
+setExtraInfo(extraInfo : String)
+getChildCount() : Integer
+getAdultCount() : Integer
-type
<<enumeration>>
RoomType
regular
family
king

Figure 62: Room Class

Room class has several private properties. These are roomNo(Integer), floorNo(Integer), dailyPrice(Double), childCount(Integer), adultCount(Integer), extraInfo(String), type(RoomType) and status(RoomStatus). This class has also one public constructor and several methods such as getRoomNo(), getFloorNo(), getType(), getStatus(), setRoomNo(roomNo : Integer), setFloorNo(RoomNo : Integer), setType(type : RoomType), setStatus(status : RoomStatus), setChildCount(childCount : Integer), setAdultCount(adultCount : Integer), getDailyPrice(), setDailyPrice(dailyPrice : Double), getExtraInfo(), setExtraInfo(extraInfo : String), getChildCount(), getAdultCount().

RoomStatus

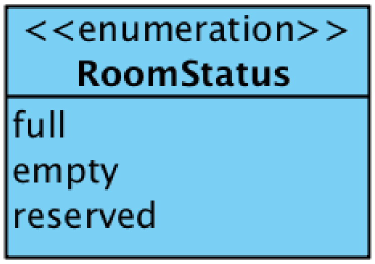


Figure 63: RoomStatus Class

RoomStatus is a enum class that provides status types to the Room class.

RoomType

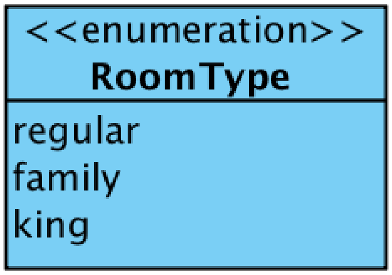
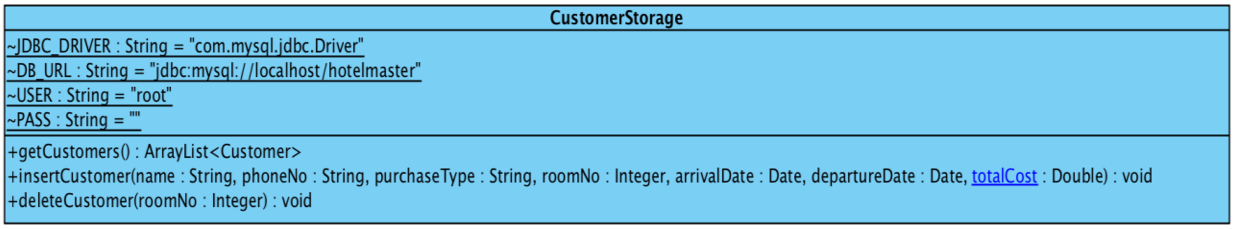


Figure 64: RoomType Class

RoomStatus is a enum class that provides status types to the Room class.

5.2.4 Storage Package

CustomerStorage

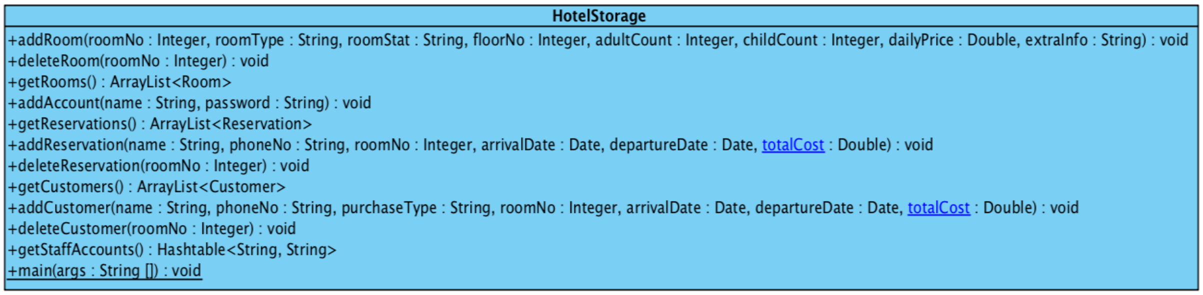


```
CustomerStorage
~JDBC_DRIVER : String = "com.mysql.jdbc.Driver"
~DB_URL : String = "jdbc:mysql://localhost/hotelmaster"
~USER : String = "root"
~PASS : String = ""
+getCustomers() : ArrayList<Customer>
+insertCustomer(name : String, phoneNo : String, purchaseType : String, roomNo : Integer, arrivalDate : Date, departureDate : Date, totalCost : Double) : void
+deleteCustomer(roomNo : Integer) : void
```

Figure 65: CustomerStorage Class

CustomerStorage class has four static constant variables which have package visibility. These are JDBC_DRIVER(“com.mysql.jdbc.Driver” : String), DB_URL(“jdbc:mysql://localhost/hotelmaster” : String), USER(“root” : String), PASS(“ “ : String). It has also three public methods which are getCustomers() and insertCustomerName(name : String, phoneNo : String, purchaseType : String, roomNo : Integer, arrivalDate : Date, departureDate : Date, totalCost : Double).

HotelStorage



```
HotelStorage
+addRoom(roomNo : Integer, roomType : String, roomStat : String, floorNo : Integer, adultCount : Integer, childCount : Integer, dailyPrice : Double, extraInfo : String) : void
+deleteRoom(roomNo : Integer) : void
+getRooms() : ArrayList<Room>
+addAccount(name : String, password : String) : void
+getReservations() : ArrayList<Reservation>
+addReservation(name : String, phoneNo : String, roomNo : Integer, arrivalDate : Date, departureDate : Date, totalCost : Double) : void
+deleteReservation(roomNo : Integer) : void
+getCustomers() : ArrayList<Customer>
+addCustomer(name : String, phoneNo : String, purchaseType : String, roomNo : Integer, arrivalDate : Date, departureDate : Date, totalCost : Double) : void
+deleteCustomer(roomNo : Integer) : void
+getStaffAccounts() : Hashtable<String, String>
+main(args : String []) : void
```

Figure 66: HotelStorage Class

HotelStorage class has several public methods. These are addRoom(roomNo : Integer, roomType : String, roomStat : String, floorNo : Integer, adultCount : Integer, childCount : Integer, dailyPrice : Double, extraInfo : String), deleteRoom(roomNo : Integer), getRoom(), addAccount(name : String, password : String), getReservations(), addReservation(name : String, phoneNo : String, roomNo : ınteger, arrivalDate : Date, departureDate : Date, totalCost : Double), deleteReservation(roomNo : Integer), getCustomers(), addCustomer(name : String, phoneNo : String, purchaseType : String, roomNo : Integer, arrivalDate : Date, departureDate : Date, totalCost : Double), deleteCustomer(roomNo : Integer), getStaffAccounts() and main() method.

LoginStorage

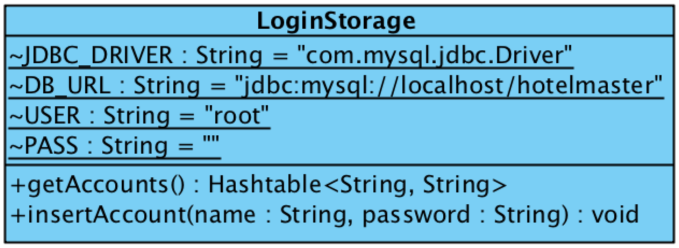

LoginStorage
~JDBC_DRIVER : String = "com.mysql.jdbc.Driver"
~DB_URL : String = "jdbc:mysql://localhost/hotelmaster"
~USER : String = "root"
~PASS : String = ""
+getAccounts() : Hashtable<String, String>
+insertAccount(name : String, password : String) : void

Figure 67: LoginStorage Class

LoginStorage class has four static constant variables which have package visibility. These are JDBC_DRIVER(“com.mysql.jdbc.Driver” : String), DB_URL(“jdbc:mysql://localhost/hotelmaster” : String), USER(“root” : String), PASS(“ “ : String). It has also two public methods which are getAccounts() and insertAccount(name: String, password : String).

ReservationStorage

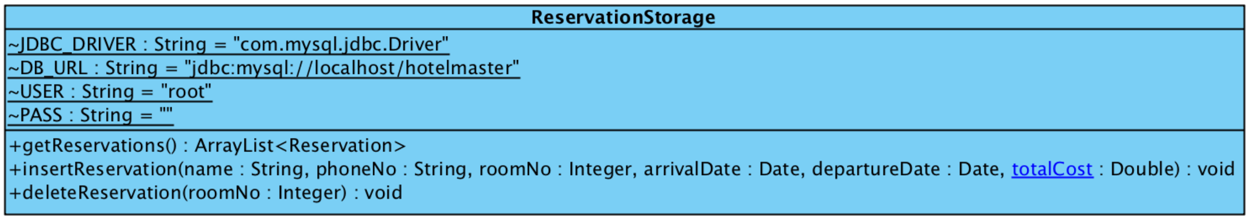

ReservationStorage
~JDBC_DRIVER : String = "com.mysql.jdbc.Driver"
~DB_URL : String = "jdbc:mysql://localhost/hotelmaster"
~USER : String = "root"
~PASS : String = ""
+getReservations() : ArrayList<Reservation>
+insertReservation(name : String, phoneNo : String, roomNo : Integer, arrivalDate : Date, departureDate : Date, totalCost : Double) : void
+deleteReservation(roomNo : Integer) : void

Figure 68: ReservationStorage Class

ReservationStorage class has four static constant variables which have package visibility. JDBC_DRIVER(“com.mysql.jdbc.Driver”), DB_URL(“jdbc:mysql://localhost/hotelmaster” : String), USER(“root” : String), PASS(“ “ : String). It has also three public methods which are getReservatios(), insertReservation(name : String, phoneNo : String, roomNo : int, arrivalDate : Date, departureDate : Date, totalCost : double) and deleteReservation(roomNo : Integer).

RoomStorage

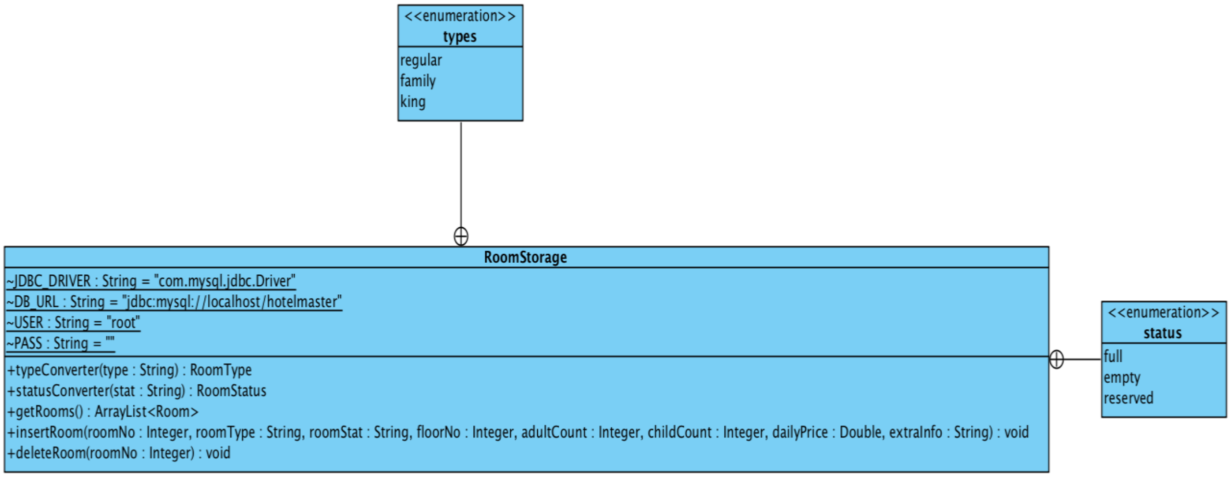

<<enumeration>>
types
regular
family
king
RoomStorage
~JDBC_DRIVER : String = "com.mysql.jdbc.Driver"
~DB_URL : String = "jdbc:mysql://localhost/hotelmaster"
~USER : String = "root"
~PASS : String = ""
+typeConverter(type : String) : RoomType
+statusConverter(stat : String) : RoomStatus
+getRooms() : ArrayList<Room>
+insertRoom(roomNo : Integer, roomType : String, roomStat : String, floorNo : Integer, adultCount : Integer, childCount : Integer, dailyPrice : Double, extraInfo : String) : void
+deleteRoom(roomNo : Integer) : void
<<enumeration>>
status
full
empty
reserved

Figure 69: RoomStorage Class

RoomStorage has imported two classes from HotelEntities, which are RoomStatus and RoomType classes. They are enum classes as above. RoomStorage class has four static constant variables which have package visibility. These are JDBC_DRIVER(“com.mysql.jdbc.Driver” : String), DB_URL(“jdbc:mysql://localhost/hotelmaster” : String), USER(“root” : String), PASS(“ “ : String). It has also five public methods which are typeConverter(type : String), statusConverter(stat : String), getRooms(), insertRoom(roomNo : Integer, roomType : String, floorNo : Integer, adultCount : Integer, childCount : Integer, dailyPrice : Double, extraInfo : String) and deleteRoom(roomNo : Integer) methods.

5.3 Specifying Contracts

Customer:

1. **Context** Customer:: getName(): String

   post: result = self.name

After the getName() method, name is returned.

2. **Context** Customer:: setName(name: String)

   post: self.name = name

The parameter must equal to name, after the setName(name: String) method.

3. **Context** Customer:: setTotalCost(totalCost: double)

   post: self.totalCost = totalCost

The parameter must equal to totalCost, after the setTotalCost(totalCost: double) method.

4. **Context** Customer:: getTotalCost(): double

   post: result = self.totalCost

After the getTotalCost(), totalCost is returned.

5. **Context** Customer:: setArrivalDate(arrivalDate: Date)

   post: self.arrivalDate = arrivalDate

The parameter must equal to arrivalDate, after the setArrivalDate(arrivalDate: Date) method

6. **Context** Customer:: getDepartureDate(): Date

   post: result = self.departureDate

After the getDepartureDate() method, departureDate is returned.

CreditCard:

7. **Context** CreditCard:: setCardNo(cardNo: String)

   post: self.cardNo = cardNo

After the setCardNo(cardNo: String) method, the parameter must equal to cardNo.

8. **Context** CreditCard:: getCardNo(): String

post: result = self.cardNo

After the getCardNo() method, cardNo is returned.

HotelController:

9. Context HotelController:: deleteRoomFromHotel(roomNo: int)

pre: roomStorage.getRooms()

post: roomStorage.deleteRoom(roomNo)

To be able to delete the room from hotel, firstly, rooms are taken from room storage and then if the room relating with room number is in the storage, it can be deleted.

10. Context HotelController:: addRoomToHotel(room: Room)

post: roomStorage.insertRoom(room)

After the addRoomToHotel(room), the room is added to storage with its relative information.

11. Context HotelController:: addCustomerToHotel(customer: Customer)

post: customerStorage.insertCustomer(customer)

After the addCustomertoHotel(customer), the customer is added to customerStorage with its relative information.

12. Context HotelController:: deleteCustomerFromHotel(roomNo : int)

pre: customerStorage.getCustomers()

post: customerStorage.deleteCustomer(roomNo)

To be able to delete customer from hotel, firstly, customer storage are taken and then customer can be deleted through room number.

HotelStorage:

13. Context HotelStorage:: deleteReservation (roomNo: int)

post: reservationStorage.deleteReservation (roomNo)

After the deleteReservation(roomNo:int) method, room is deleted from reservation storage.

14. Context HotelStorage:: getRooms(): ArrayList<Room>

post : result = roomStorage.getRooms()

Hotel must return rooms, after the getRooms() method.

15. Context HotelStorage:: getReservations(): ArrayList<Reservation>

post: result = reservationStorage.getReservations()

Hotel must return reservations, after the getReservations() method.

16. Context HotelStorage:: getCustomers(): ArrayList<Customer>

post: result = customerStorage.getCustomers()

Hotel must return customers, after the getCustomers() method.

Hotel:

17. Context Hotel:: setNumberOfRoom(numberOfRoom : int)

post: self.numberOfRoom = numberOfRoom

After the setNumberOfRoom (numberOfRoom:int) method, the parameter must equal to numberOfRoom.

18. Context Hotel:: getNumberOfRoom(): int

post: result = self.numberOfRoom

After the getNumberOfRoom() method, numberOfRoom is returned.

19. Context Hotel:: getAvailableRooms() : ArrayList<Room>

post: result = roomStorage.getAvailableRooms()

Hotel must return available rooms, after the getAvailableRooms() method.

20. Context Hotel:: updateHotelCustomers()

pre: controller.getAllCustomers()

To be able to update hotel customers, firstly all customers must be got.

21. Context Hotel:: updateHotelReservations()

pre: controller.getAllReservations()

All reservations must be taken to update hotel reservations.

LoginController:

22. Context LoginController:: checkAuth(username: String, password: String): boolean

pre: loginController.checkAuth(username, password)

To be able to login to the system, username and password should be checked.

RoomController:

23. Context RoomController:: typeConverter(type: RoomType): String

pre: getType()

To be able to convert the room type such as regular, family, king, it must get the type first.

24. Context RoomController:: statusConverter(type: StatusType): String

pre: getStatus()

Firstly get the status, to be able to indicate the room status such as empty, full or reserved.

25. Context RoomController:: addRoom(room: Room)

post: storage.addRoom(room)

After the addRoom(room: Room) method, room is added with its room no, type, status, floor no, adult and child count, daily price and extra info into room storage.

26. Context RoomController:: getAllRooms(): ArrayList<Room>

post: storage.getRooms()

After the getAllRooms() method, rooms are taken from storage.

Room:

27. Context Room:: getFloorNo(): int

post: result = self.floorNo

After the getFloorNo() method, floorNo is returned.

28. Context Room:: setFloorNo(floorNo: int)

post: self.floorNo = floorNo

After the setFloorNo(floorNo: int) method, the parameter must be equal to number of floor.

29. Context Room:: setDailyPrice(dailyPrice : double)

post: self.dailyPrice = dailyPrice

After the setDailyPrice(dailyPrice : double) method, the parameter must be equal to daily price.

30. Context Room:: getDailyPrice() : double

post: result = self.dailyPrice

After the getDailyPrice() method, dailyPrice is returned.

6. Conclusions and Lessons Learned

6.1 Conclusion

The main objective of this program is to give qualified service to customers such as their reservation, check-in, and check-out process. Hotel staff is main actor of this system. Hotel staff can do their jobs; implement customers' requests effectively by using this program. Customers can take quick service when they go to hotel. Hotel can store all customers or rooms information in the unique well-arranged system and so through this way their jobs will be eased.

The reports have mainly 2 parts which are Requirement Analysis and Analysis parts. In Requirement Analysis part, we gave an overview about the program and we explained functional and non-functional requirements and constraints. We defined different scenarios about the process of the system. We supported them with use case models that representing the functionality of the system from a user's point of view. We also provided user interfaces and mockups.

In Analysis part, we provide domain information with the important terms and constructed class diagrams, state charts and sequence diagrams. In class diagrams, we indicated classes and objects and relationships with each other in the structure of the system. We described the behavior of the system in state charts. In sequence diagrams, each scenario is represented corresponding diagram we showed the behavior of the scenarios among its participating objects.

In Design part, we decided our design goal before the implementation. We decomposed the system into subsystems. During decomposition low coupling and high cohesion are the main criteria. After decomposition we designed the Architectural Patterns and we decided the relationship between packages. We explained key concerns such as persistent data management, access control and security, global software control and boundary conditions. After Design Part, we explained our class interfaces and we specified the contacts using OCL.

6.1 Lessons Learned

In this project, we learned both theoretical information and practical coding skills. Mainly, this project provided us an insight of software development process and prepared us for business life. We realized that each Requirement Analysis, System Design and Object Design process has necessities that need to be done before development. Most importantly, it was a teamwork which needs interpersonal communication. We learned a lot of new experience from this project with teamwork. During project, Requirements Analysis part took the longest time but it was easy. System Design part was more difficult, because we had not implemented the project yet. The last part Object Design was also easy because we had code already. Hence, we thought a lot about the design and implementation of the project and tried to produce a user-friendly, well designed software.