Bilkent University
Department Of Computer Engineering

CS 319 Project :

Final Report

AutoHotel
Hotel Automation Software System

Group 3-7

İrem Hergüner	21201077
Halit Onur Karabaş	20901806
Çağlar Alim Ağlar	21101255

Can Alkan

CS319
Object Oriented Software Engineering

Contents

List Of Figures

Analysis and Design Report

AutoHotel : Hotel Automation Software System

1) Introduction

This project is an automation software system for the use of hotel employees. We are designing this system in order to be a desktop application on hotel's computers. With this project, each employee can see and set the information of rooms and customers according to their job description in order to maintain their works efficiently. So, the users of this software system is hotel employees, not customers. All adjustments are made by considering that every employee has access to a computer in order to enter informations according to their jobs into the system. With this system, hotels do not have any missing and incorrect document about operation of hotel in order to eliminate fraud. Besides this, AutoHotel can keep and store all information about hotel in one place.

In this project, we aimed to create an automation system that all workers can communicate easily and efficiently between themselves. For this reason, we will try to maintain a system which is able to use easily by all workers without any training. However, to avoid any possible problem about software system, there will be help page in order to guide employees about their abilities of access, usage manual of system and information about handling possible situations. For these reasons, our program has functions to provide every need of an hotel by reducing the complexity of the program and providing a user friendly environment.

2) Requirement Analysis

2.1 Overview

There are workers in hotels which are mainly receptionists, room keepers, managers, accountants and kitchen chef. Receptionist can make a reservation with the help of this software, managers can control operations, accountants can calculate economic issues, room keepers can see dirty and clean rooms and chef in kitchen can see order of customers that comes from room service. This system aims to ease the communication among hotel employees and improve efficiency of service.

2.2 Functional Requirements

- Users of this program are hotel employees, not customers.
- This program will be designed with considering the all employees have computer access to make changes on rooms and customers information.
- Each user types which are receptionist, manager, accountant, room keeper and kitchen chef have their own interface according to their worker ID and password on opening page.
- Receptionist will be able to access customer and room informations, make and cancel reservations depending on capacity of hotel.
- Manager will be able to access worker informations, set shifts, pay salaries and control the economic status.
- Accountant will be able to access calcualte workers salaries, pay hotel taxes, calculate economic status and report it to the manager and set currency.

- Room keeper will be able to access room informations that if they are empty, full, clean or dirty; get minibar status, laundry need and ordered room service.
- Kitchen chef will be able to access customers meal order detail to prepare them, update the their bill and manage the hotel meal plan.

2.3 Non-functional Requirements

- System’s user interface is designed according to employees types that means each employee can make changes only on their job related informations.
- System is user friendly that every employee -even not related with higher technology- can easily manage.
- We will use online databases to keep the information of customer, hotel, room and worker.
- System defines its technical details which are ease of use, ease of learning, extensibility, portability, modifiability and consistency in design goal part

2.3.1 User-friendly Interface

AutoHotel system is user friendly in many ways:

- The program is easy to install since it is desktop application and do not need any external database system. It can keep informations in text files.
- The system is intuitive that it can be work well with good GUI.
- Software is efficient in a way that users can complete their job, features are well located, not maste work time while manage the system.
- It is a menu-driven programs, employees are able to find their ways with toolbars unlikely than command-driven programs.

2.3.2 Extensibility

Extensibility means system ability about future growth according to requirements. Good extensibility feature of software system can be measured with effort while adding new functionalities to software.

Our project supports the white-box extensibility that modified with source code. It is flexible and less restrictive. This project is an object- oriented software that supports inheritance and dynamic binding. So with extensions, original system cannot be affected by changings.

As an example, during extensions developers can add new worker types without changing original source code because there are inheritance between worker class and worker types which are receptionist, manager, accountant, room keeper and kitchen chef. In order to add new worker type,the extension does not affect these classes under favour of object-oriented software structure and specifically glass-box extensibility.

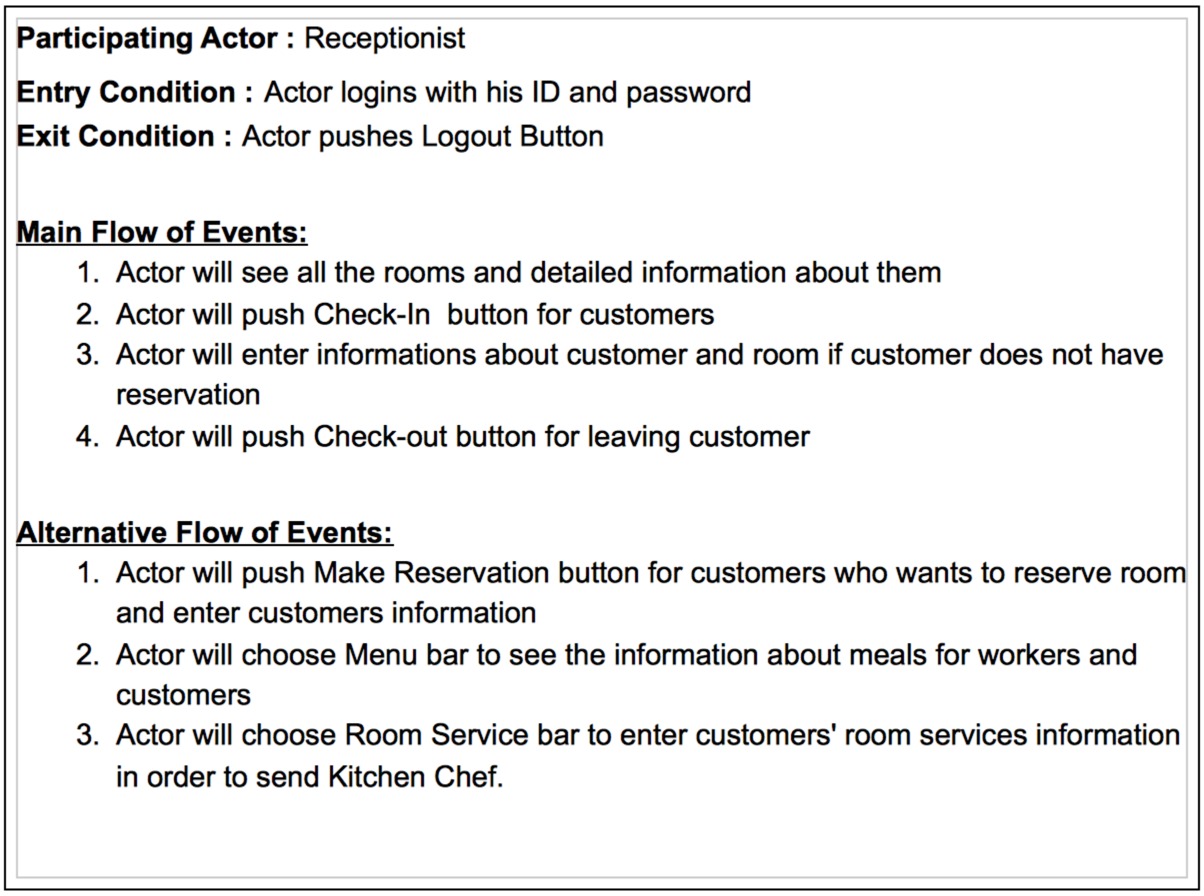
2.4 Pseudo Requirements

- The program will be implemented in Java.
- This program will implement for desktop computers.

2.5 Scenarios

There will be different scenarios for different users. There will be six different type of user which are receptionist, cleaner, manager, accountant and kitchen workers. Our program will detect users by their login information. These users will be have usernames and passwords, and these usernames will contain the information about their job description. Therefore, these scenarios show us how can these different users use this program depend on their job description.

2.5.1 Scenarios for Receptionists



Participating Actor : Receptionist

Entry Condition : Actor logins with his ID and password
Exit Condition : Actor pushes Logout Button

<u>Main Flow of Events:</u>

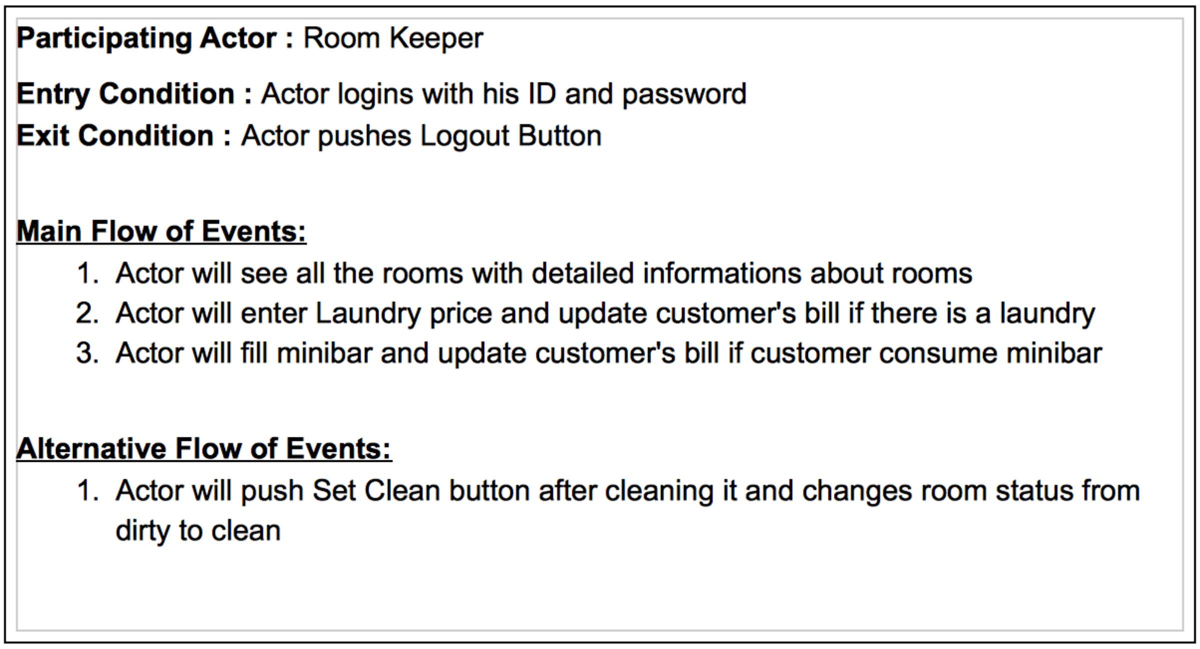
1. Actor will see all the rooms and detailed information about them
2. Actor will push Check-In button for customers
3. Actor will enter informations about customer and room if customer does not have reservation
4. Actor will push Check-out button for leaving customer

<u>Alternative Flow of Events:</u>

1. Actor will push Make Reservation button for customers who wants to reserve room and enter customers information
2. Actor will choose Menu bar to see the information about meals for workers and customers
3. Actor will choose Room Service bar to enter customers' room services information in order to send Kitchen Chef.

Table 1: Receptionist Scenario

2.5.2 Scenarios for Room Keeper



Participating Actor : Room Keeper

Entry Condition : Actor logins with his ID and password
Exit Condition : Actor pushes Logout Button

<u>Main Flow of Events:</u>

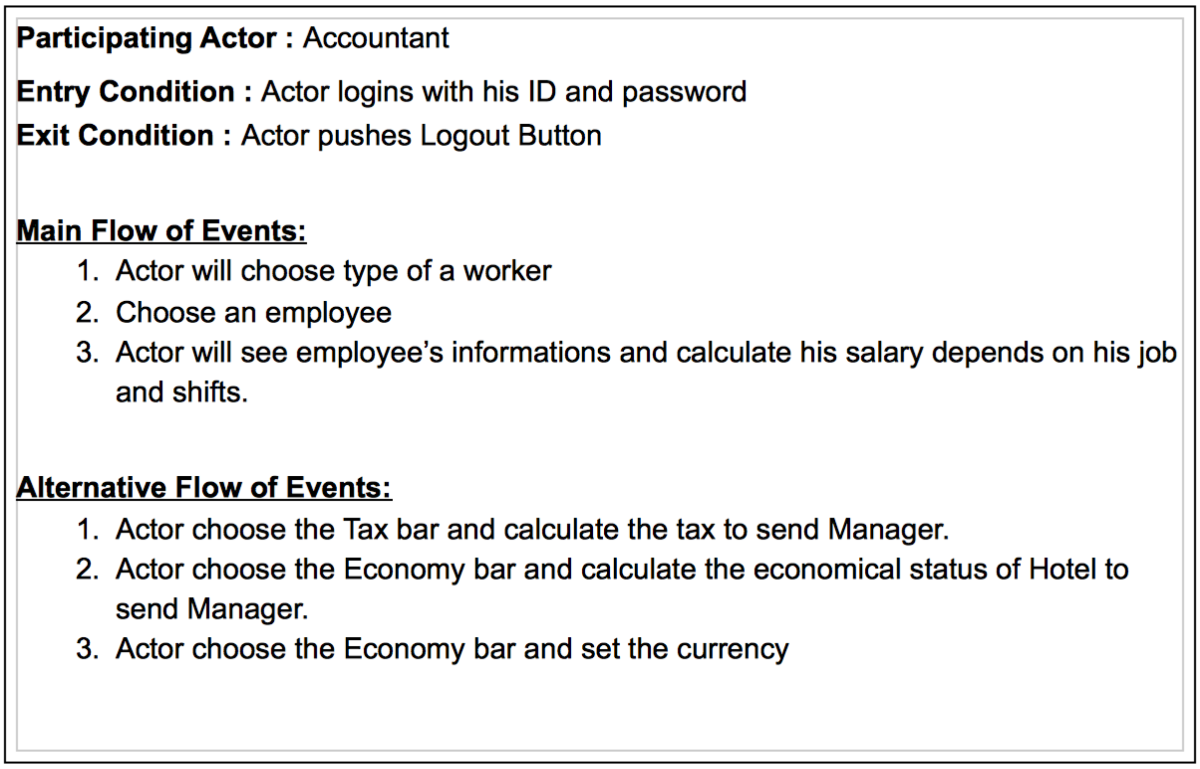
1. Actor will see all the rooms with detailed informations about rooms
2. Actor will enter Laundry price and update customer's bill if there is a laundry
3. Actor will fill minibar and update customer's bill if customer consume minibar

<u>Alternative Flow of Events:</u>

1. Actor will push Set Clean button after cleaning it and changes room status from dirty to clean

Table 2: Room Keeper Scenario

2.5.3 Scenarios for Accountant



Participating Actor : Accountant

Entry Condition : Actor logins with his ID and password
Exit Condition : Actor pushes Logout Button

Main Flow of Events:

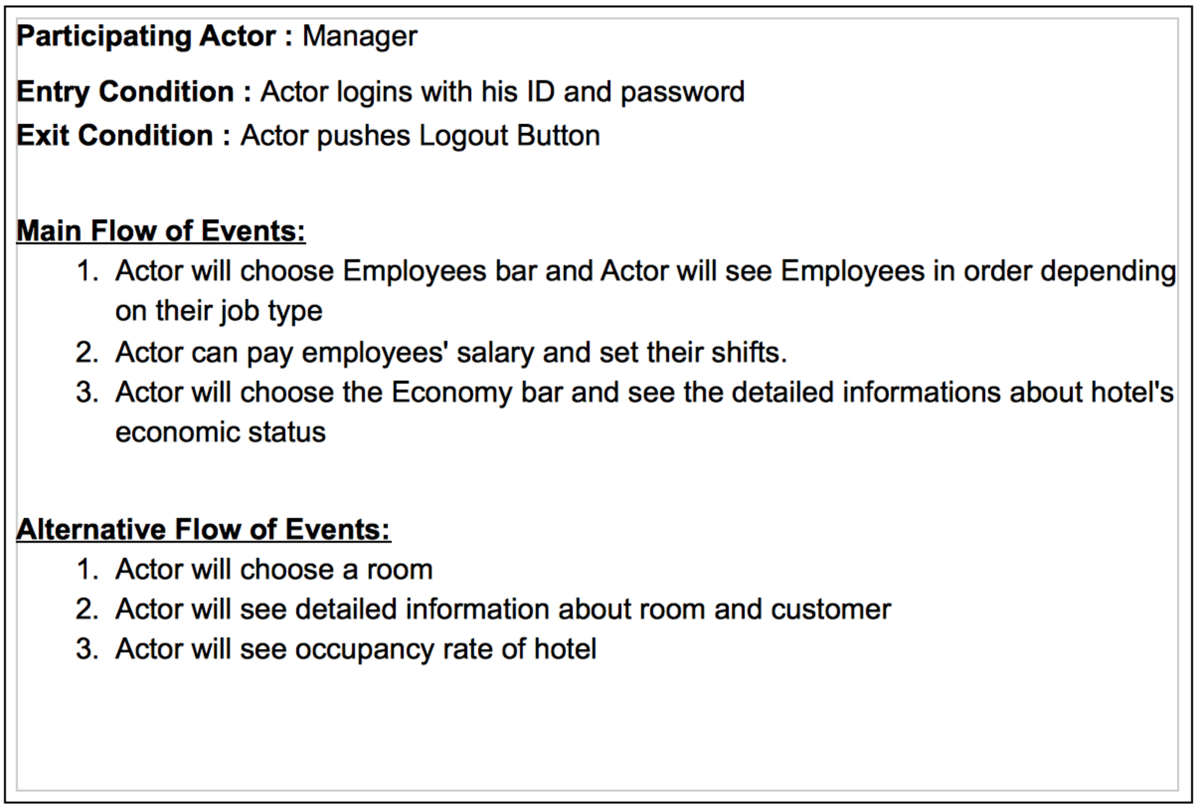
1. Actor will choose type of a worker
2. Choose an employee
3. Actor will see employee's informations and calculate his salary depends on his job and shifts.

Alternative Flow of Events:

1. Actor choose the Tax bar and calculate the tax to send Manager.
2. Actor choose the Economy bar and calculate the economical status of Hotel to send Manager.
3. Actor choose the Economy bar and set the currency

Table 3: Accountant Scenario

2.5.4 Scenarios for Manager



Participating Actor : Manager

Entry Condition : Actor logins with his ID and password
Exit Condition : Actor pushes Logout Button

Main Flow of Events:

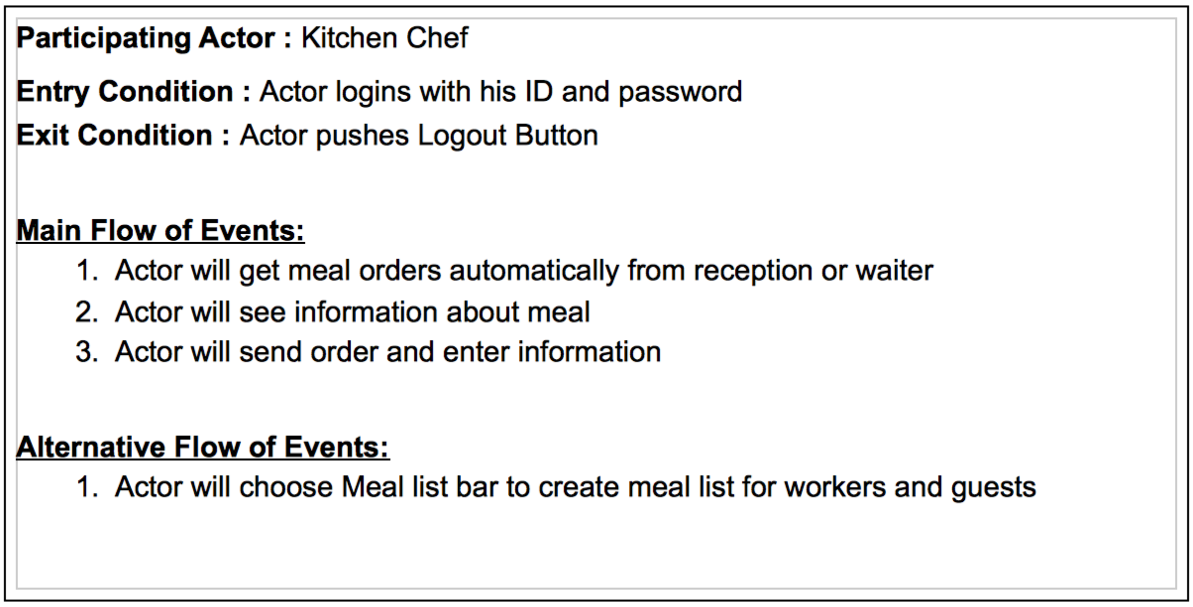
1. Actor will choose Employees bar and Actor will see Employees in order depending on their job type
2. Actor can pay employees' salary and set their shifts.
3. Actor will choose the Economy bar and see the detailed informations about hotel's economic status

Alternative Flow of Events:

1. Actor will choose a room
2. Actor will see detailed information about room and customer
3. Actor will see occupancy rate of hotel

Table 4: Manager Scenario

2.5.5 Scenarios for Kitchen Chef



Participating Actor : Kitchen Chef

Entry Condition : Actor logins with his ID and password
Exit Condition : Actor pushes Logout Button

Main Flow of Events:

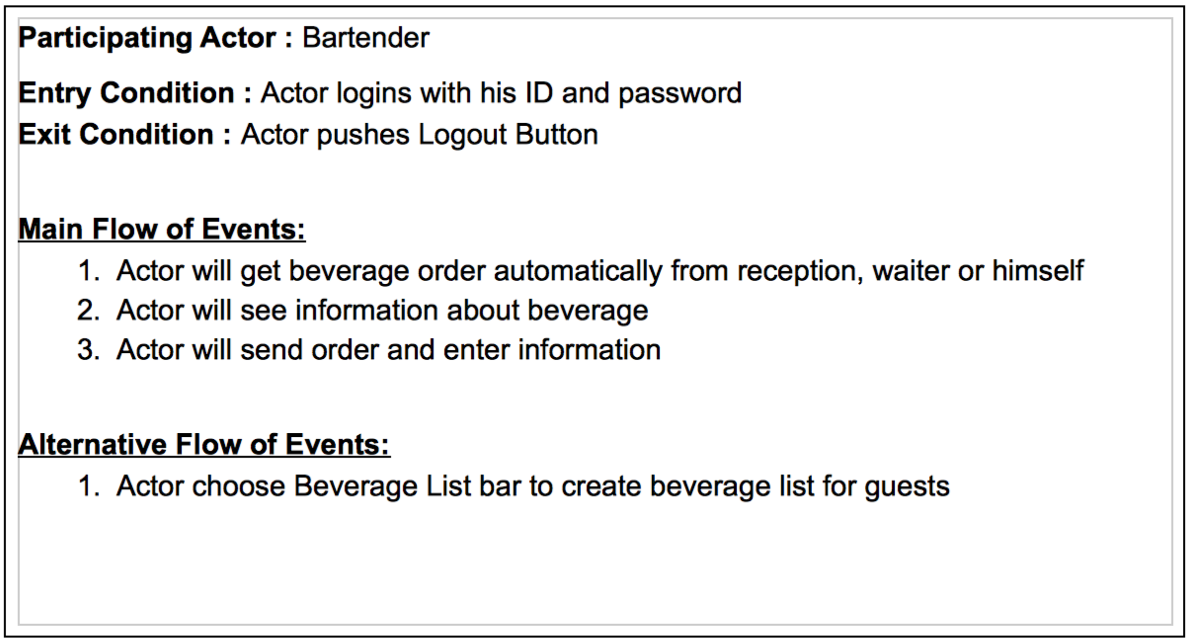
1. Actor will get meal orders automatically from reception or waiter
2. Actor will see information about meal
3. Actor will send order and enter information

Alternative Flow of Events:

1. Actor will choose Meal list bar to create meal list for workers and guests

Table 5: Kitchen Chef Scenario

2.5.6 Scenarios for Bartender



Participating Actor : Bartender

Entry Condition : Actor logins with his ID and password
Exit Condition : Actor pushes Logout Button

Main Flow of Events:

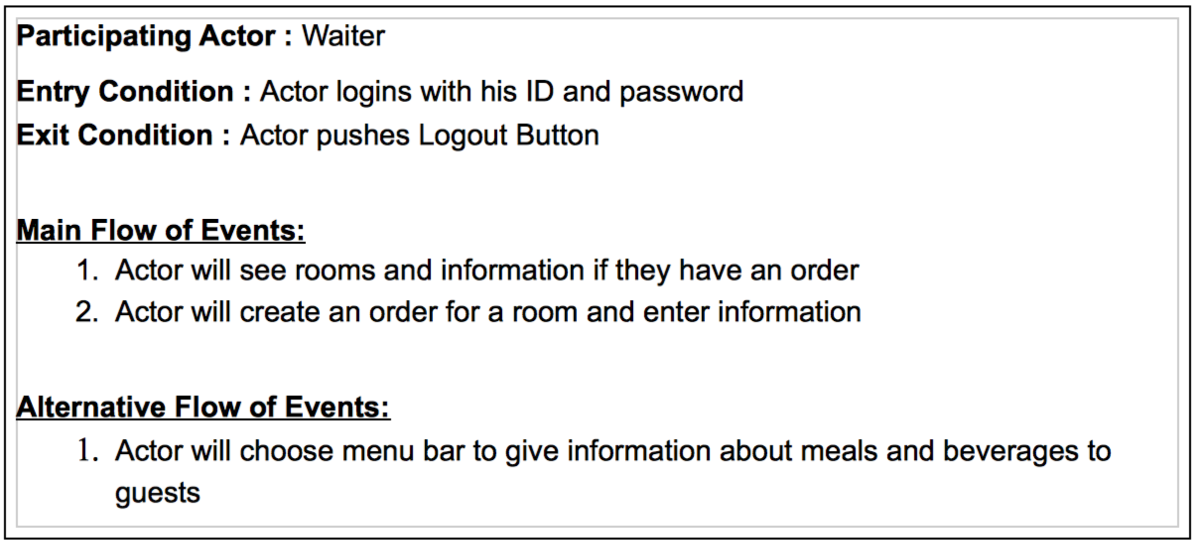
1. Actor will get beverage order automatically from reception, waiter or himself
2. Actor will see information about beverage
3. Actor will send order and enter information

Alternative Flow of Events:

1. Actor choose Beverage List bar to create beverage list for guests

Table 6: Bartender Scenario

2.5.7 Scenarios for Waiter



Participating Actor : Waiter

Entry Condition : Actor logins with his ID and password
Exit Condition : Actor pushes Logout Button

Main Flow of Events:

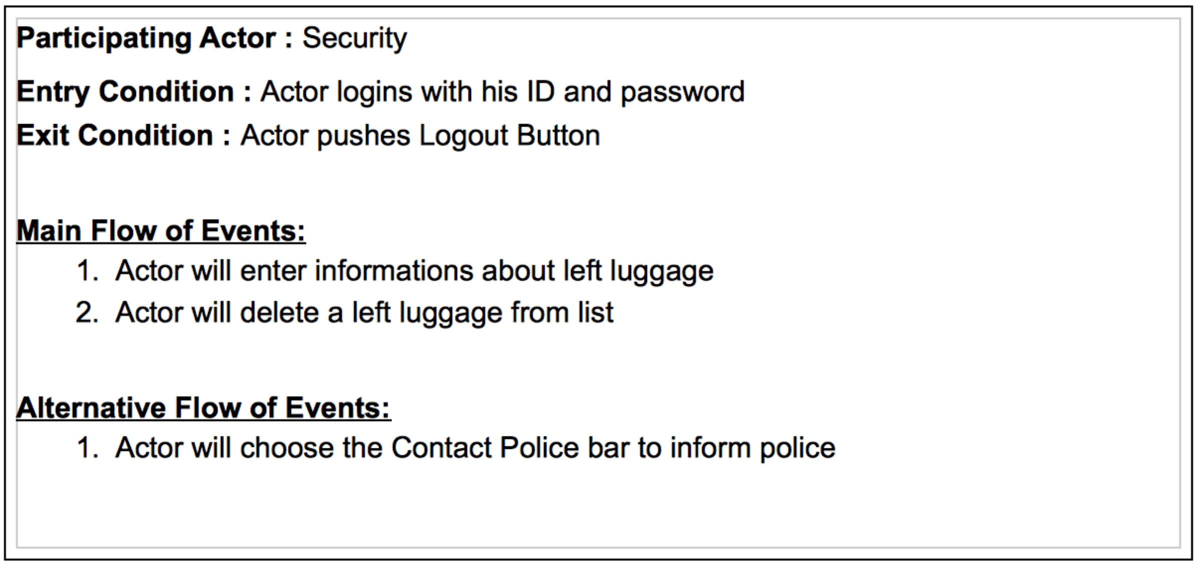
1. Actor will see rooms and information if they have an order
2. Actor will create an order for a room and enter information

Alternative Flow of Events:

1. Actor will choose menu bar to give information about meals and beverages to guests

Table 7: Room Keeper Scenario

2.5.8 Scenarios for Security



Participating Actor : Security

Entry Condition : Actor logins with his ID and password
Exit Condition : Actor pushes Logout Button

Main Flow of Events:

1. Actor will enter informations about left luggage
2. Actor will delete a left luggage from list

Alternative Flow of Events:

1. Actor will choose the Contact Police bar to inform police

Table 8: Security Scenario

2.6 Use-Case Model

In AutoHotel Project, there will be eight different users of the program and they have different screens according to their login informations and inputs about customers, hotel and employees. We use this use case model for requirements elicitation and analysis to represent the functionality of hotel automation system. The diagram represents the behaviour of the project from external view which is actor's view. This diagram also clarify boundary of the system. The actors which receptionist, security, bartender, waiter, accountant, manager, room keeper and kitchen chef are outside of the boundary in diagram. And use cases are inside the boundary. Each rectangle defines subsystems and dashed arrows defines the interactions between them.

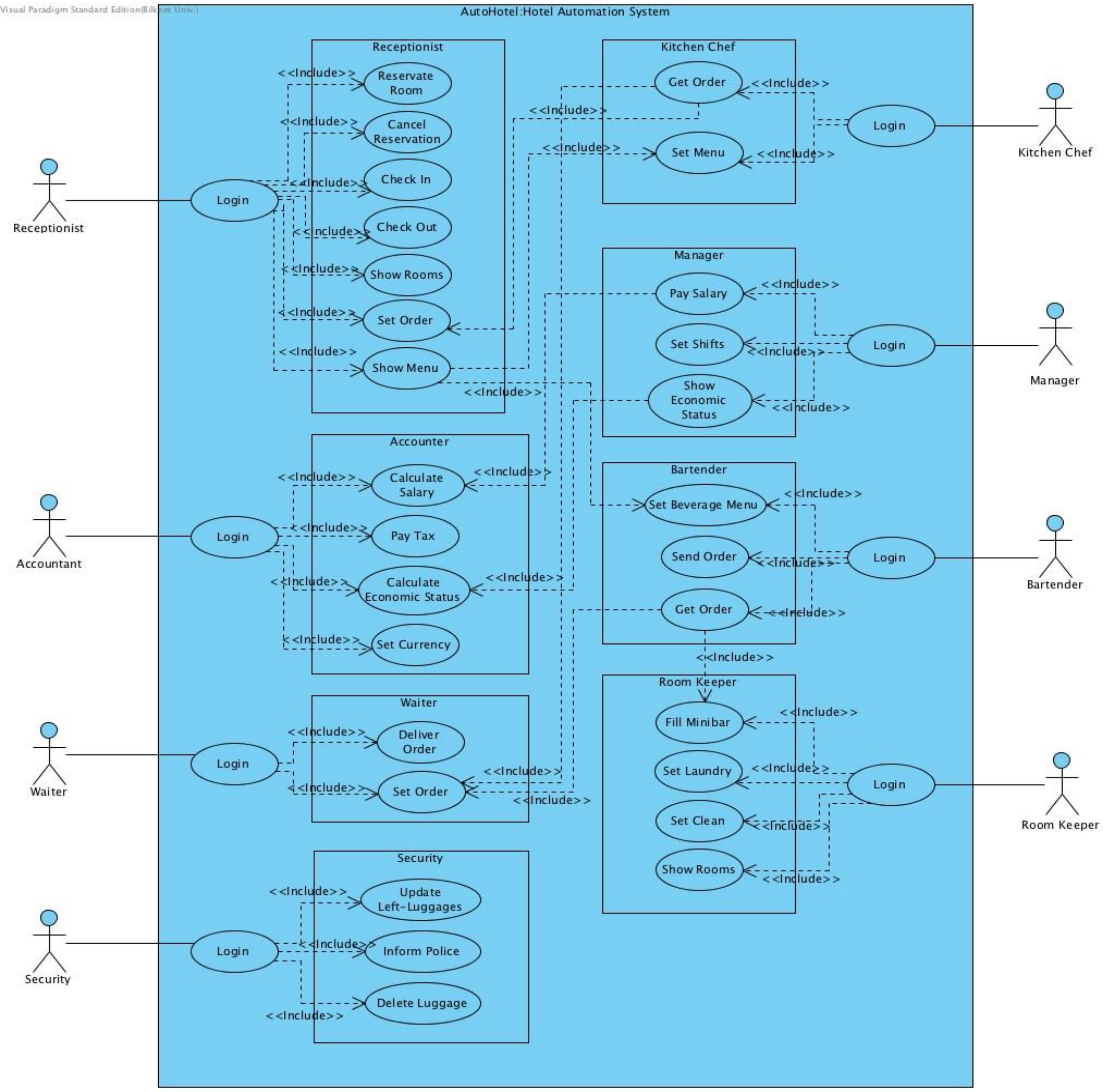
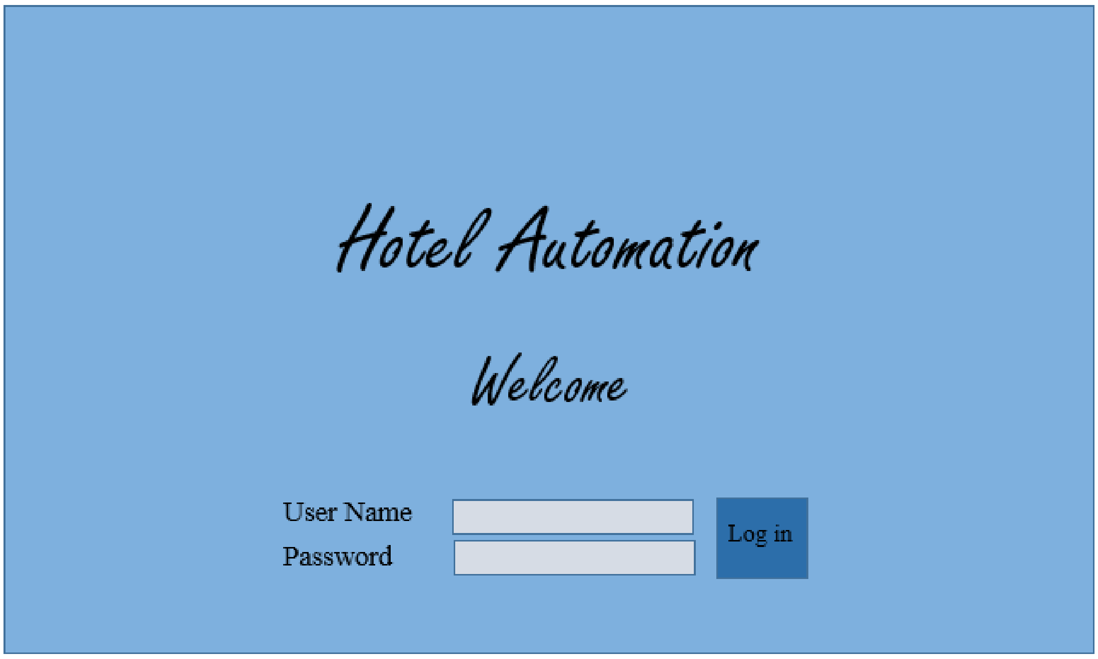


Figure 1: Use case model of whole system

2.7 User Interface

In this section, there are user interface prototypes. They do not provide much information about functionality and source codes but represent interfaces for most users. When the application runs, all users will see the welcoming screen with two text fields for username and password, and a login button. After login, the system

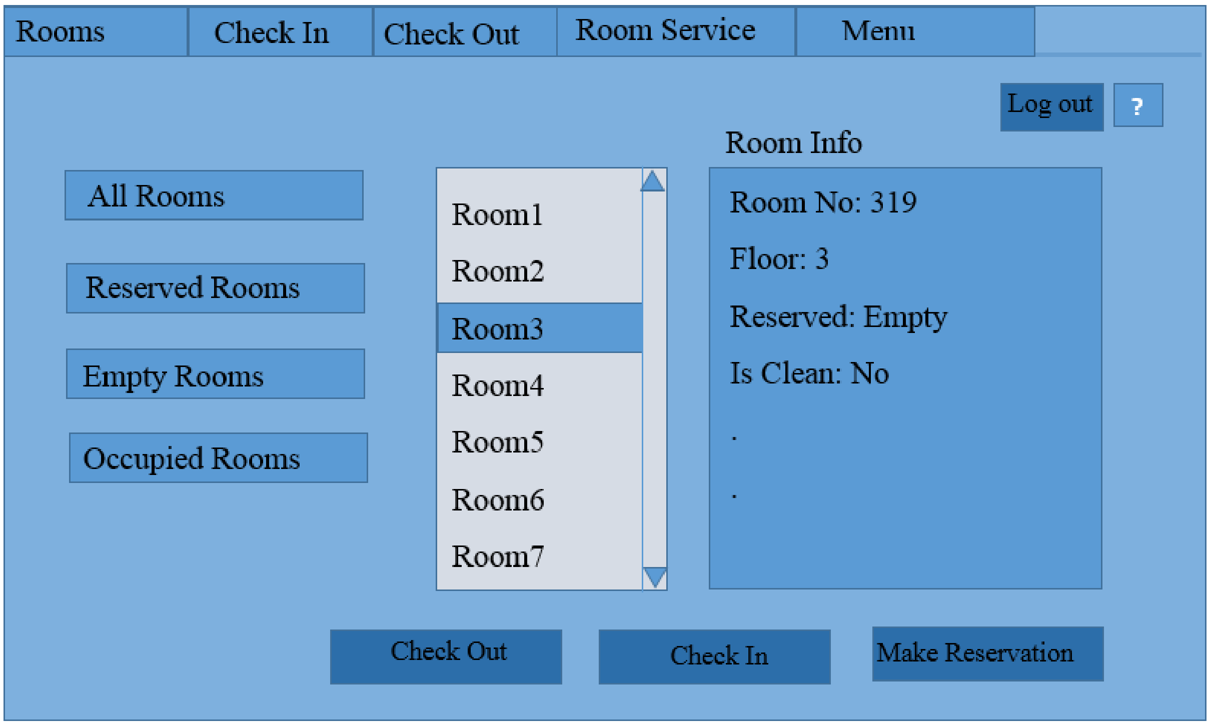


direct the users to different pages according to their authorisation.

Figure 2: Opening Page

2.7.1 Receptionist Interface

Receptionists will be directed to the screen below. This is main control screen of receptionist. In the Rooms tab, a receptionist can see informations about all the rooms. To make an easier access to different groups of rooms, there are four buttons as follows, all rooms, reserved rooms, occupied rooms and empty rooms. Receptionists can see informations about each room, and with 'check in', 'check out' and 'make reservation' buttons, they can change the status of the selected room according to customer request. Only the receptionist has the rights to change reservations.



Room Info

Room No: 319

Floor: 3

Reserved: Empty

Is Clean: No

.

.

Figure 3: Receptionist Interface

In 'Check in' tab, customer check in can be made by the receptionists. Receptionists will enter the customer info, and choose a room type depends on customer's request. According to chosen room type, list of available rooms will be displayed on the screen. Receptionist will be able to choose a room from the list, and by clicking the 'make reservation' button, receptionist can take the customer's information and store them into the system.

Here is a reservation page that customer's information is stored. Customer informations are needed to provide a better hotel service. Beside this, if customer reenter the system in further dates, system can remember customer information to ease reservation. In 'Check Out' tab, customer check out will be made by the receptionists.

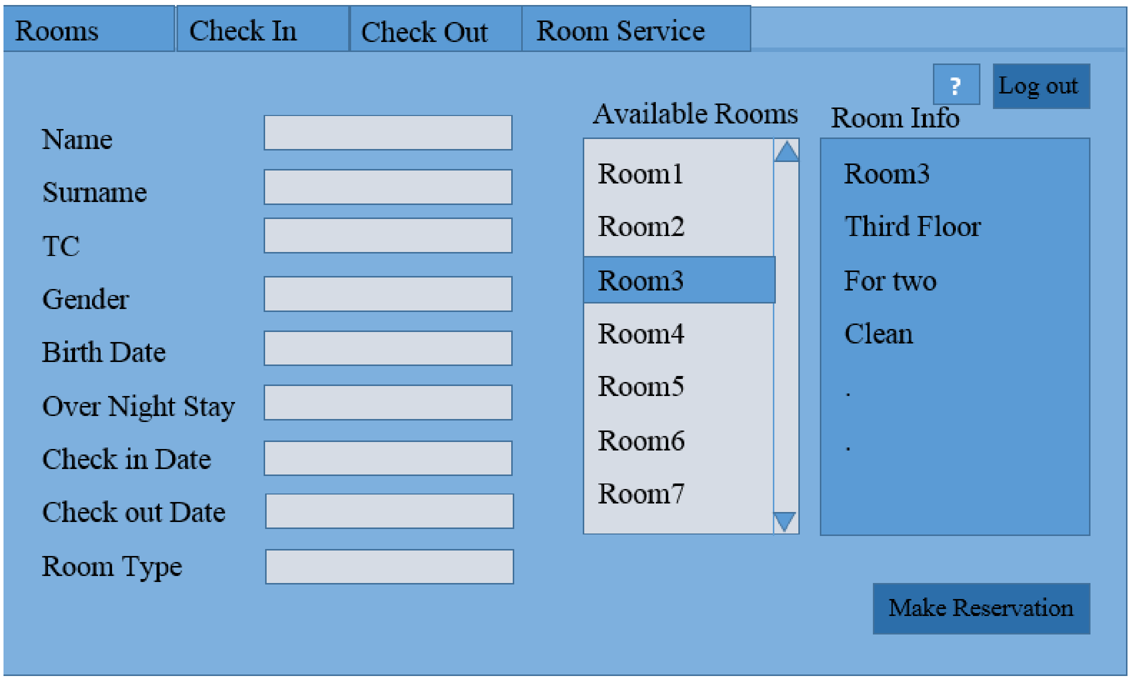


Figure 4: Room Reservation Interface

Receptionists will be able to search according to customer name or room number. After search, customer info and room info will be displayed on the screen. By clicking the ‘checkout’ button receptionist is able to end the checkout process.

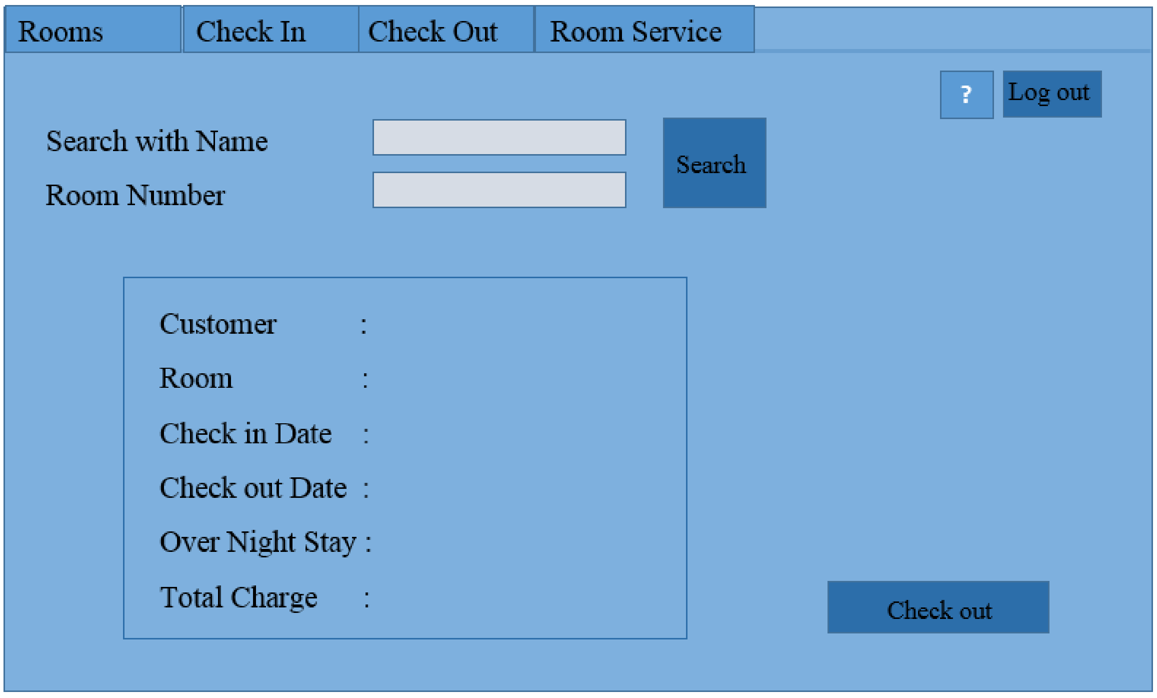


Figure 5: Room Information Page

In the menu tab, receptionists will only see the daily menu. Receptionist cannot change meal plan, it is kitchen chef's duty. They will just inform the customers about the menu if it's needed.

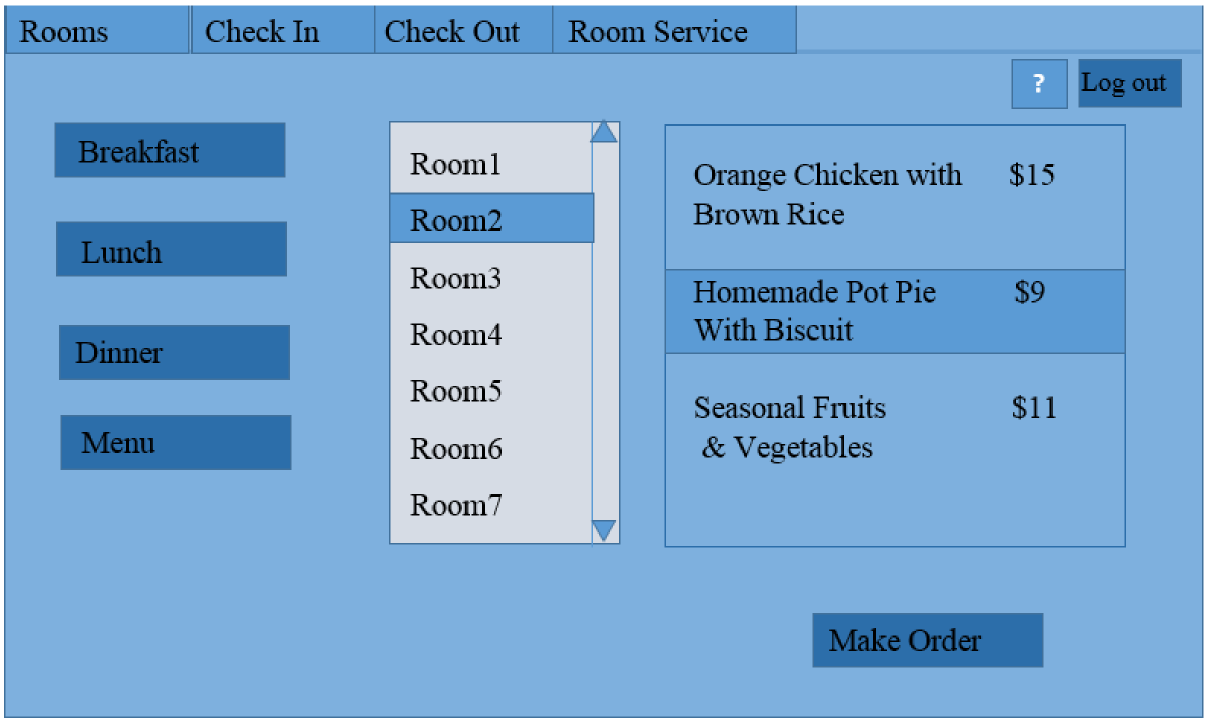


Figure 6: View of Meal Plan

2.7.2 Manager Interface

Manager will be directed to the screen below where he can see the general informations of the hotel and the rooms. The Hotel tab is similar with 'Rooms' tab where in the receptionist interface but the difference is that the manager can not change anything.

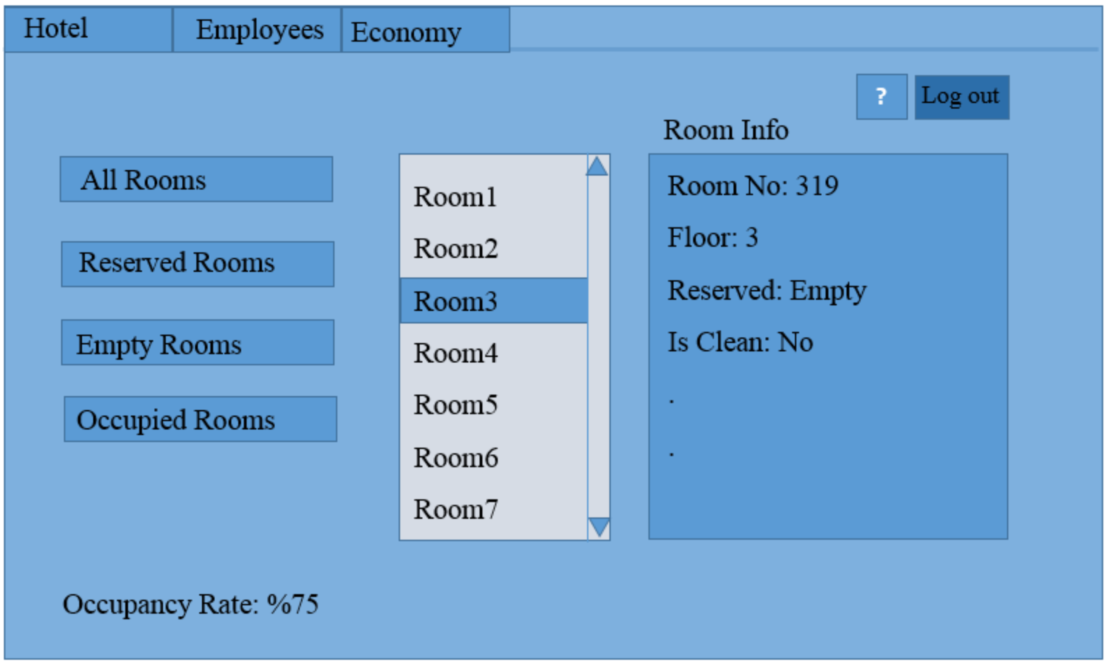


Figure 7: View of Room Status

In ‘Employees’ tab, manager is able to see employees from different departments. The manager can check the informations about the employees, set their shits, and pay their salaries.

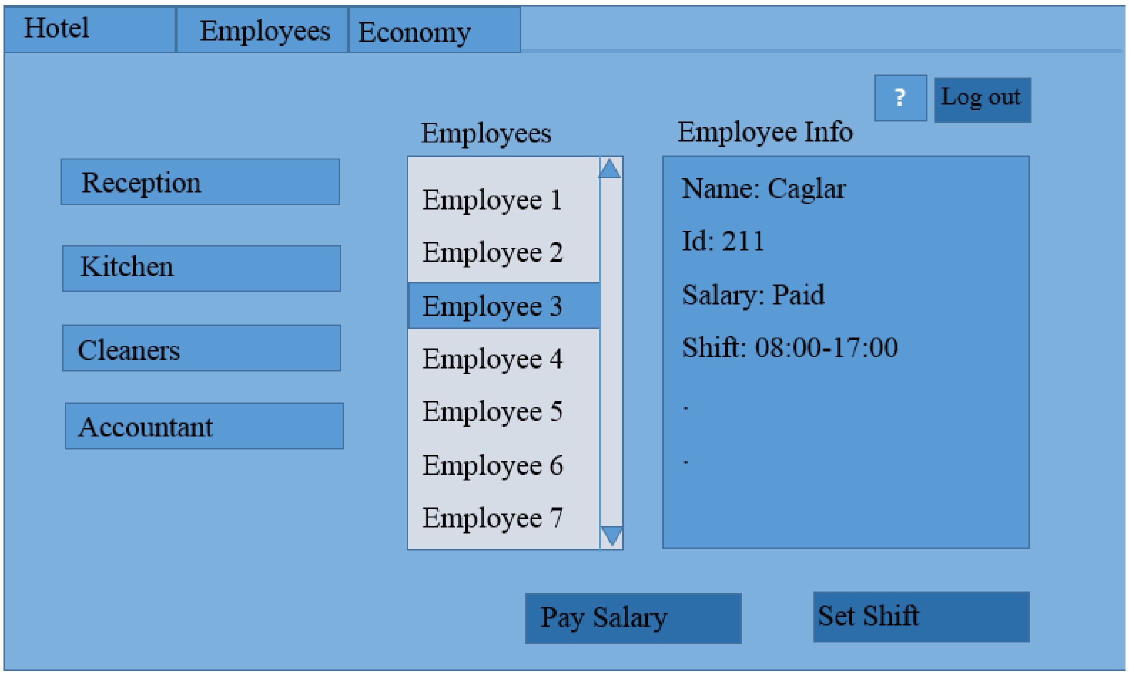
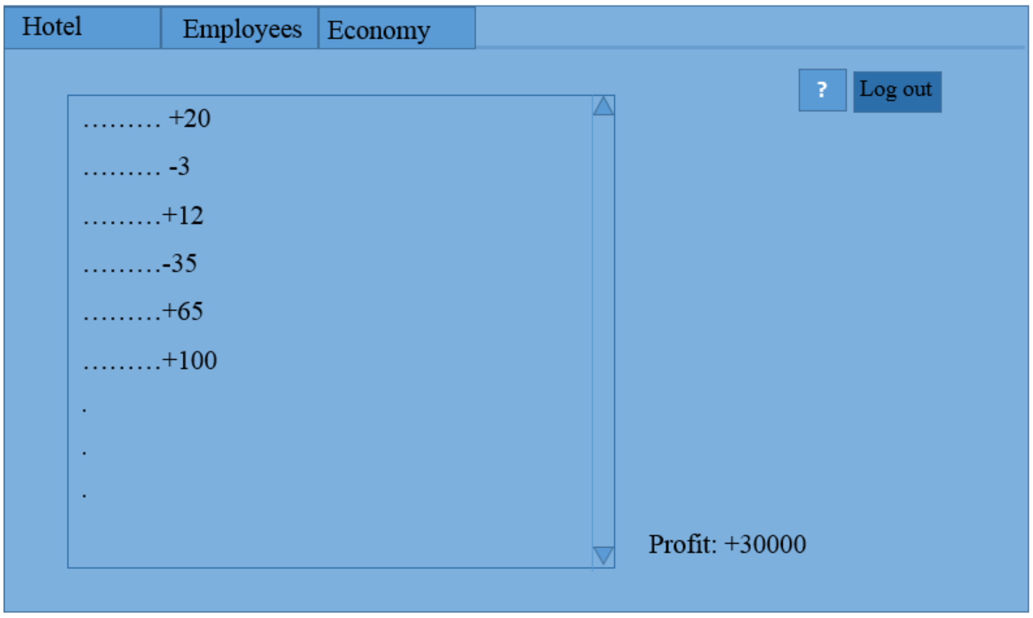


Figure 8: View of Employee Information

In the 'Economy' tab, the manager can see the economic status of the hotel, which is organized by the accountant.



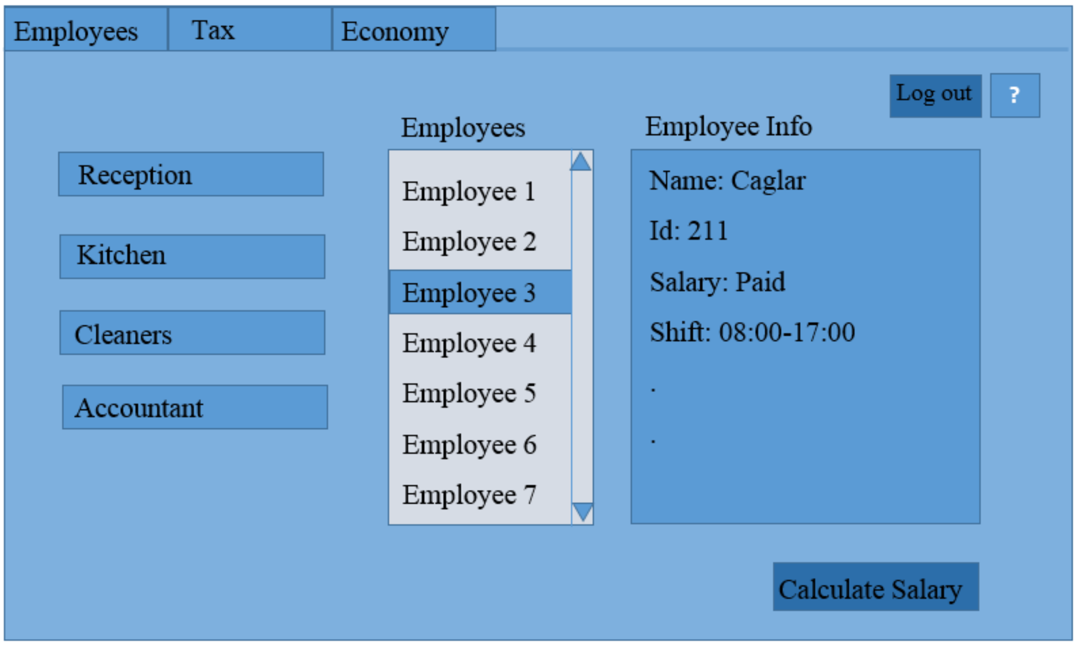
……… +20
……… -3
………+12
………-35
………+65
………+100
.
.
.

Profit: +30000

Figure 9: View of Economic Status

2.7.3 Accountant Interface

Accountant will see the screen below where he can see the employees from different departments and calculate their salaries according to their shifts.

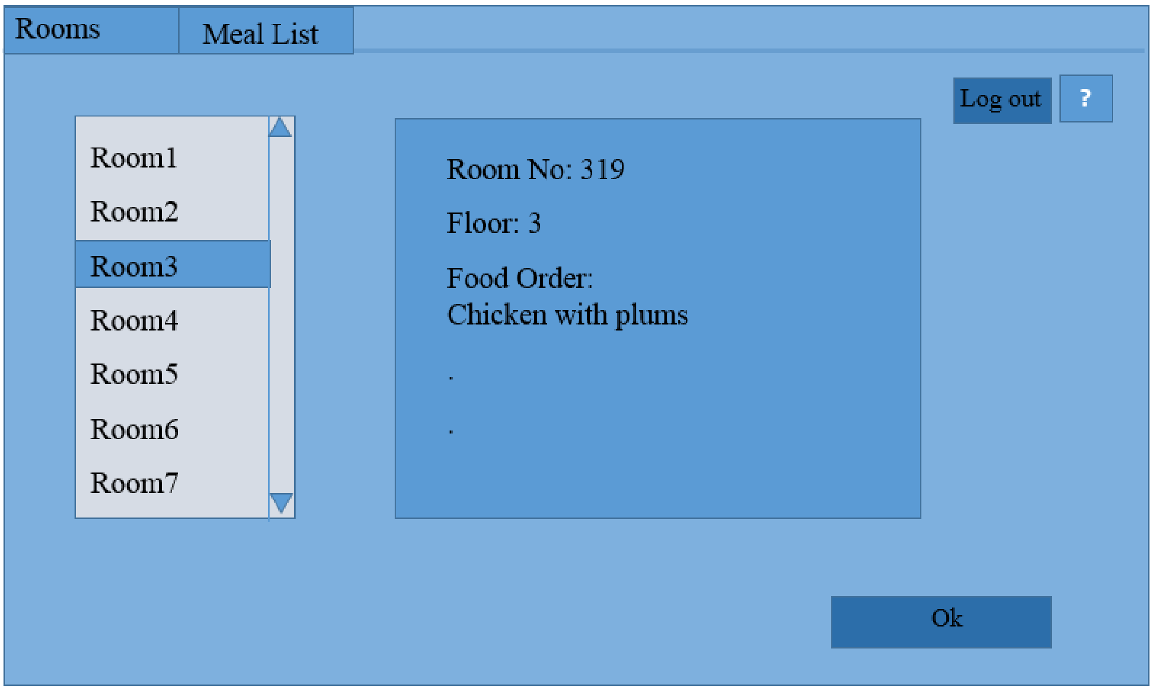


Name: Caglar
Id: 211
Salary: Paid
Shift: 08:00-17:00
.
.

Figure 10: Accountant Interface

2.7.4 Kitchen Interface

Employees from the kitchen department will be directed to the screen below where they can enter the price of ordered meals after they prepare them. The meals are ordered by the customers as a room service.



Room No: 319

Floor: 3

Food Order:
Chicken with plums

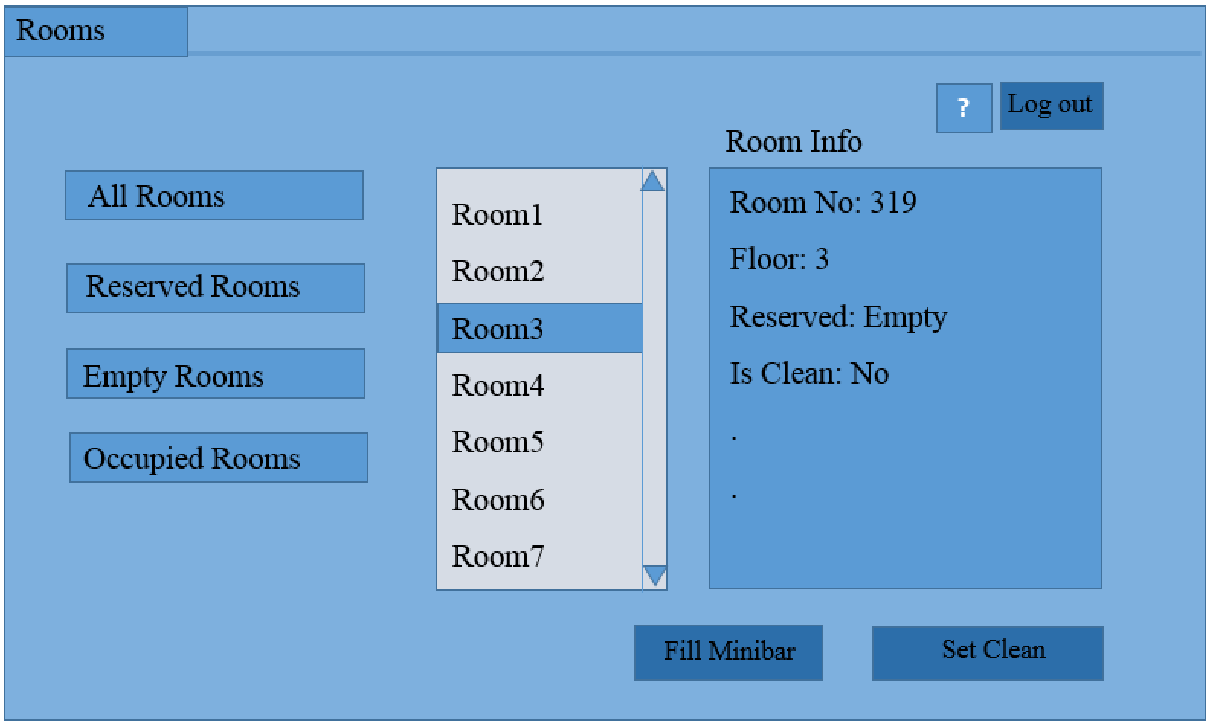
.

.

Figure11: Kitchen Chef Interface

2.7.5 Room Keeper Interface

Employees from the cleaning department which are room keepers will be directed to the screen below where they can enter customers' laundry price, change the cleaning status of the room and change the packness status of the minibar.

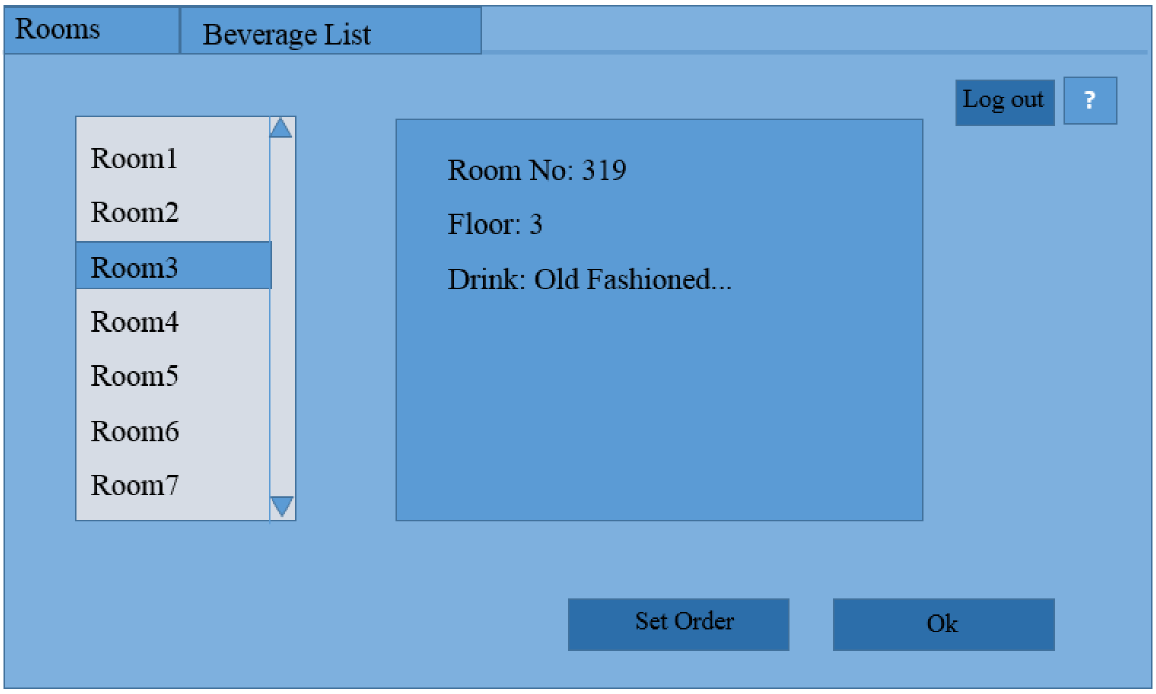


Room No: 319
Floor: 3
Reserved: Empty
Is Clean: No
.
.

Figure12: Room Keeper Interface

2.7.6 Bartender Interface

Bartenders will be directed to the screen below where they will be able to see the customers' order and change the beverage list.

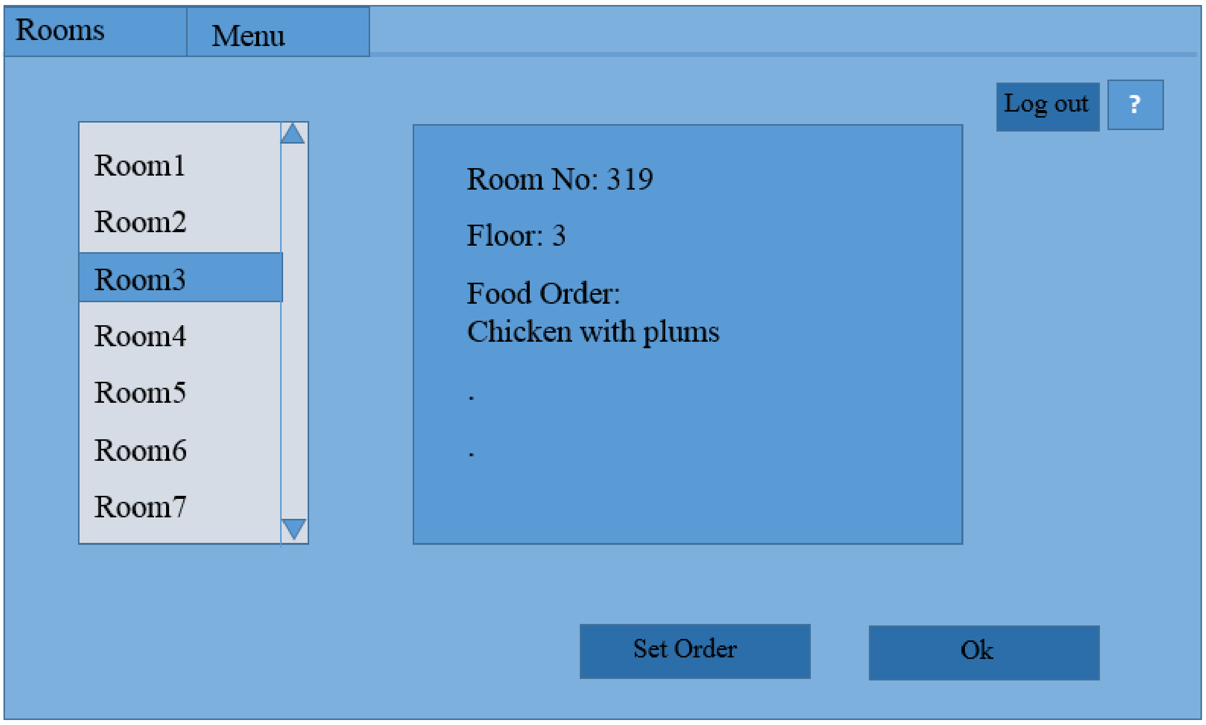


Room No: 319
Floor: 3
Drink: Old Fashioned...

Figure13: Bartender Interface

2.7.7 Waiter Interface

Waiters will be directed to the screen below where they will be able to see the customers' order and set orders of the customers. They also will be able to see the Menu but they won't be able to change it.



Room No: 319

Floor: 3

Food Order:
Chicken with plums

.

.

Figure 14: Waiter Interface

2.7.8 Security Interface

Workers from security department will be directed to the screen below where they will be able to add left luggage. They will be able to search and delete luggage.

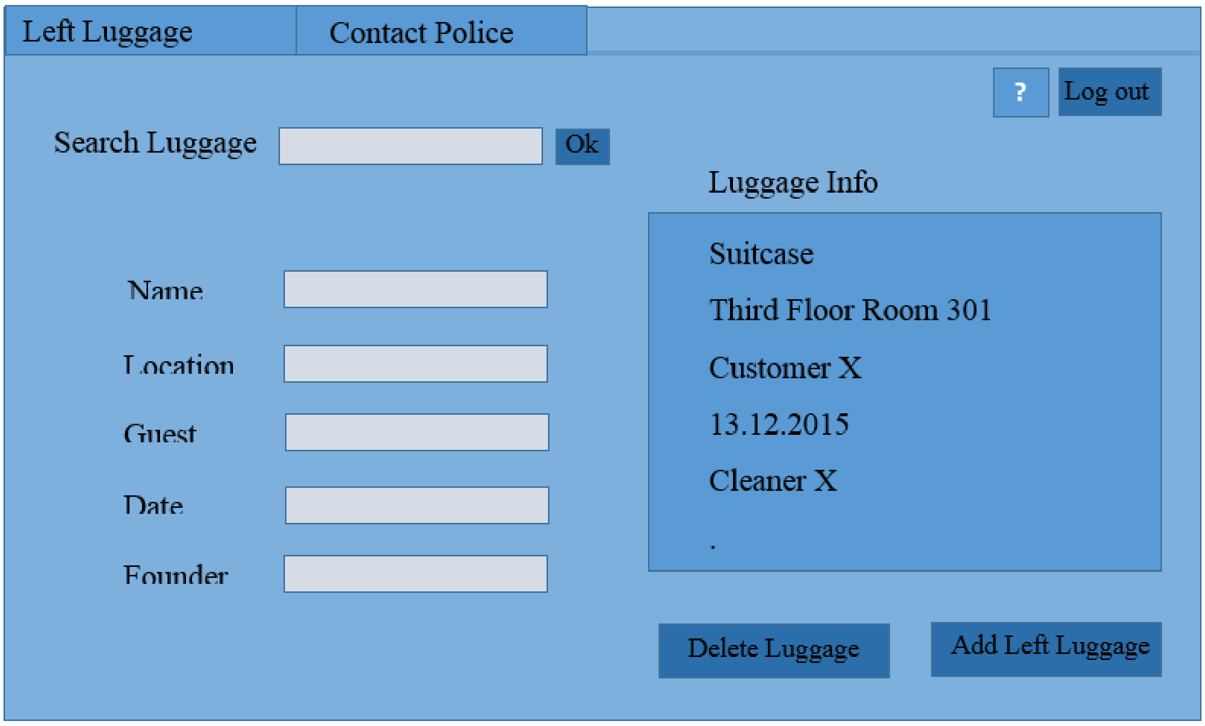


Figure 15: Security Interface

3) Analysis

3.1 Object Model

3.1.1 Domain Lexicon

In AutoHotel system, “Shift” class identifies employees working period. It is separated into two which are early riser and working in evening.

“BeverageList” class identifies drinks which hotel have in its stocks and also it is used to list in drink menu.

Hotel have laundry services for customers if they want to clean up their clothes.

Hotel have left luggage service which stores the information of left luggages with found date when customers forget them in hotel.

3.1.2 Class Diagram

AutoHotel software system have 22 classes, 5 of them are enumeration class which have fixed attributes. Hotel class is main class that can be considered as controller class. It has interaction with all classes and BeverageList, MealList, Worker, Room and LeftLuggage classes are part of Hotel. Main incidents actualise on Room and Worker classes since all employee types are kind of Worker. Customer is part of Room because in hotel, there could not be a customer without room and other employees should not learn details about customer. Thus customer class is embedded into room class and not to attend class interaction too much.

Waiter, KitchenChef and Bartender are kind of KitchenWorker, they do not have common operations but their id will start with same special number according to KithenWorkers to make operations more easily. We will give special starting numbers for each employee type in their id numbers to make search and retrieval operations easily.

In class diagram, nearly each class have set and get methods. Get method search and retrieval required information from database with its parameter. Set method make changes on databases with proper parameter.

HelpForEmployee class include texts for each employee that give information about details of their job, usage of system and what to do in the resulting possible errors.

The Room class do not have impact on worker class, it just stores customer and room information for hotel management.

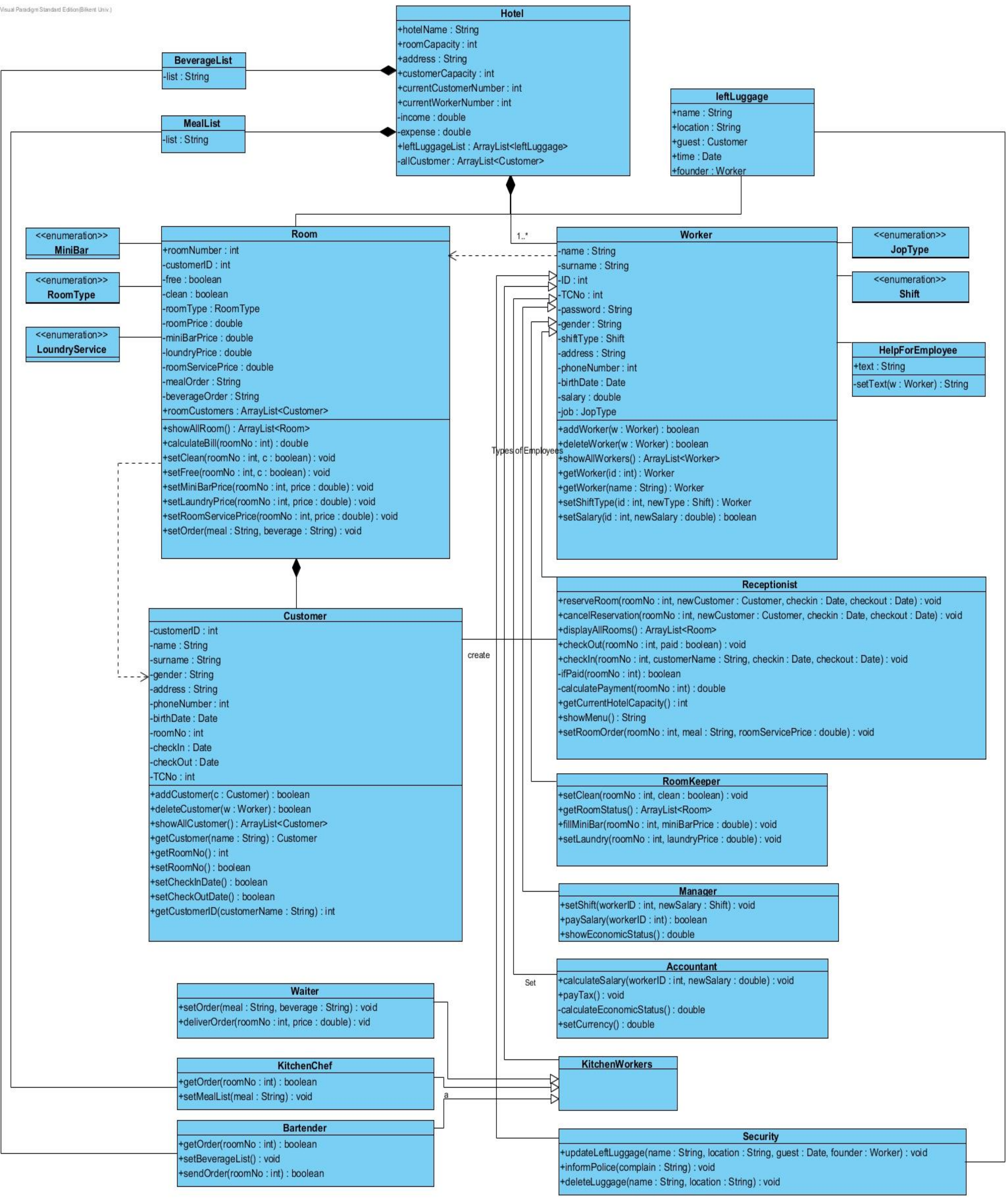

Visual Paradigm Standard Edition(Bilkent Univ.)
Hotel
+hotelName : String
+roomCapacity : int
+address : String
+customerCapacity : int
+currentCustomerNumber : int
+currentWorkerNumber : int
-income : double
-expense : double
+leftLuggageList : ArrayList<leftLuggage>
-allCustomer : ArrayList<Customer>
BeverageList
-list : String
MealList
-list : String
leftLuggage
+name : String
+location : String
+guest : Customer
+time : Date
+founder : Worker
<<enumeration>>
MiniBar
<<enumeration>>
RoomType
<<enumeration>>
LoundryService
Room
+roomNumber : int
-customerID : int
-free : boolean
-clean : boolean
-roomType : RoomType
-roomPrice : double
-miniBarPrice : double
-loundryPrice : double
-roomServicePrice : double
-mealOrder : String
-beverageOrder : String
+roomCustomers : ArrayList<Customer>
+showAllRoom() : ArrayList<Room>
+calculateBill(roomNo : int) : double
+setClean(roomNo : int, c : boolean) : void
+setFree(roomNo : int, c : boolean) : void
+setMiniBarPrice(roomNo : int, price : double) : void
+setLaundryPrice(roomNo : int, price : double) : void
+setRoomServicePrice(roomNo : int, price : double) : void
+setOrder(meal : String, beverage : String) : void
1..*
Worker
-name : String
-surname : String
-ID : int
-TCNo : int
-password : String
-gender : String
-shiftType : Shift
-address : String
-phoneNumber : int
-birthDate : Date
-salary : double
-job : JopType
+addWorker(w : Worker) : boolean
+deleteWorker(w : Worker) : boolean
+showAllWorkers() : ArrayList<Worker>
+getWorker(id : int) : Worker
+getWorker(name : String) : Worker
+setShiftType(id : int, newType : Shift) : Worker
+setSalary(id : int, newSalary : double) : boolean
<<enumeration>>
JopType
<<enumeration>>
Shift
HelpForEmployee
+text : String
-setText(w : Worker) : String
Types of Employees
Customer
-customerID : int
-name : String
-surname : String
-gender : String
-address : String
-phoneNumber : int
-birthDate : Date
-roomNo : int
-checkIn : Date
-checkOut : Date
-TCNo : int
+addCustomer(c : Customer) : boolean
+deleteCustomer(w : Worker) : boolean
+showAllCustomer() : ArrayList<Customer>
+getCustomer(name : String) : Customer
+getRoomNo() : int
+setRoomNo() : boolean
+setCheckInDate() : boolean
+setCheckOutDate() : boolean
+getCustomerID(customerName : String) : int
create
Receptionist
+reserveRoom(roomNo : int, newCustomer : Customer, checkin : Date, checkout : Date) : void
+cancelReservation(roomNo : int, newCustomer : Customer, checkin : Date, checkout : Date) : void
+displayAllRooms() : ArrayList<Room>
+checkOut(roomNo : int, paid : boolean) : void
+checkIn(roomNo : int, customerName : String, checkin : Date, checkout : Date) : void
-ifPaid(roomNo : int) : boolean
-calculatePayment(roomNo : int) : double
+getCurrentHotelCapacity() : int
+showMenu() : String
+setRoomOrder(roomNo : int, meal : String, roomServicePrice : double) : void
RoomKeeper
+setClean(roomNo : int, clean : boolean) : void
+getRoomStatus() : ArrayList<Room>
+fillMiniBar(roomNo : int, miniBarPrice : double) : void
+setLaundry(roomNo : int, laundryPrice : double) : void
Manager
+setShift(workerID : int, newSalary : Shift) : void
+paySalary(workerID : int) : boolean
+showEconomicStatus() : double
Accountant
+calculateSalary(workerID : int, newSalary : double) : void
+payTax() : void
-calculateEconomicStatus() : double
+setCurrency() : double
Set
Waiter
+setOrder(meal : String, beverage : String) : void
+deliverOrder(roomNo : int, price : double) : vid
KitchenChef
+getOrder(roomNo : int) : boolean
+setMealList(meal : String) : void
a
KitchenWorkers
Bartender
+getOrder(roomNo : int) : boolean
+setBeverageList() : void
+sendOrder(roomNo : int) : boolean
Security
+updateLeftLuggage(name : String, location : String, guest : Date, founder : Worker) : void
+informPolice(complain : String) : void
+deleteLuggage(name : String, location : String) : void

Figure 16: Class Diagram

3.2.2 State Chart Diagram

In this section, there are State Chart diagrams that are used to formalize the dynamic behaviour of individual objects it the system. They start with initial condition after login step and states of the objects are shown with the events affect them.

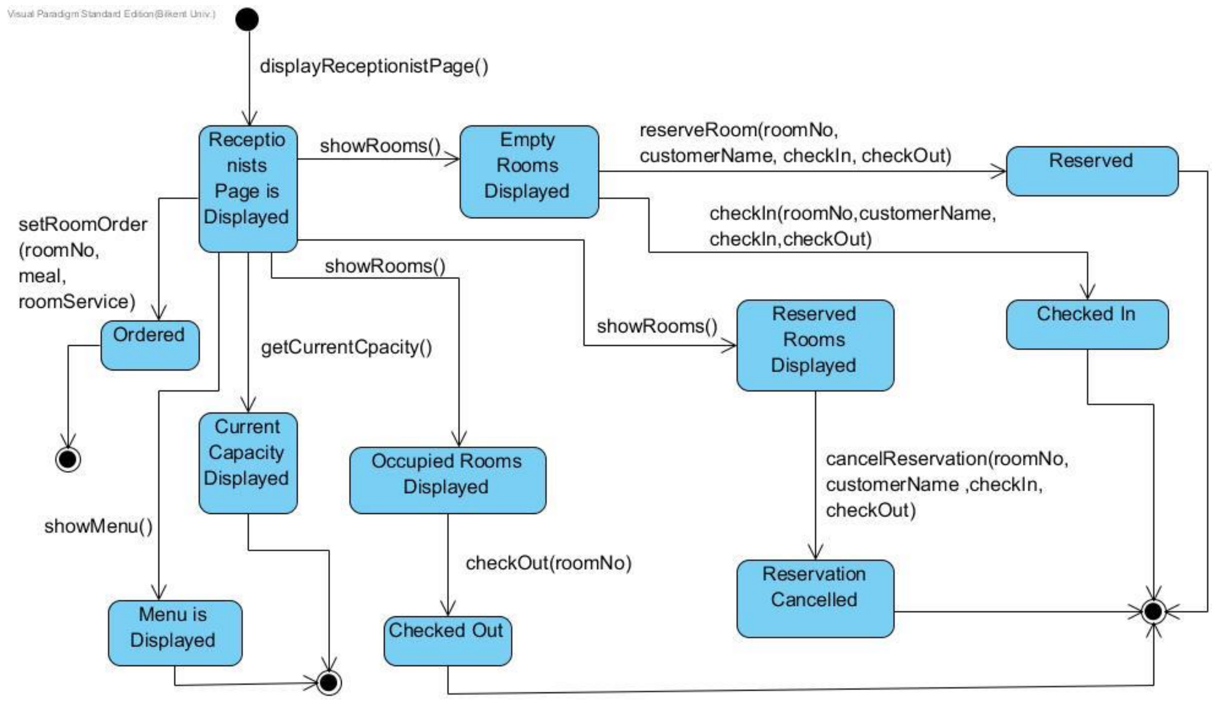

Visual Paradigm Standard Edition(Bilkent Univ.)
displayReceptionistPage()
Receptionists Page is Displayed
showRooms()
Empty Rooms Displayed
reserveRoom(roomNo, customerName, checkIn, checkOut)
Reserved
checkIn(roomNo,customerName, checkIn,checkOut)
Checked In
setRoomOrder (roomNo, meal, roomService)
Ordered
showRooms()
showRooms()
Reserved Rooms Displayed
getCurrentCpacity()
Current Capacity Displayed
Occupied Rooms Displayed
cancelReservation(roomNo, customerName ,checkIn, checkOut)
showMenu()
checkOut(roomNo)
Reservation Cancelled
Menu is Displayed
Checked Out

Figure 17: State chart diagram for Receptionist

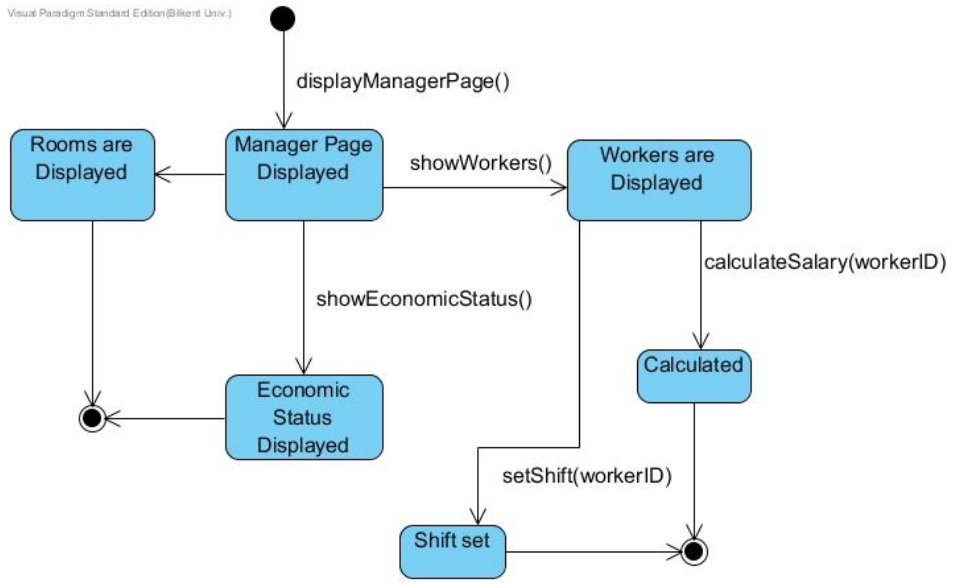


Figure 18: State chart diagram for Manager

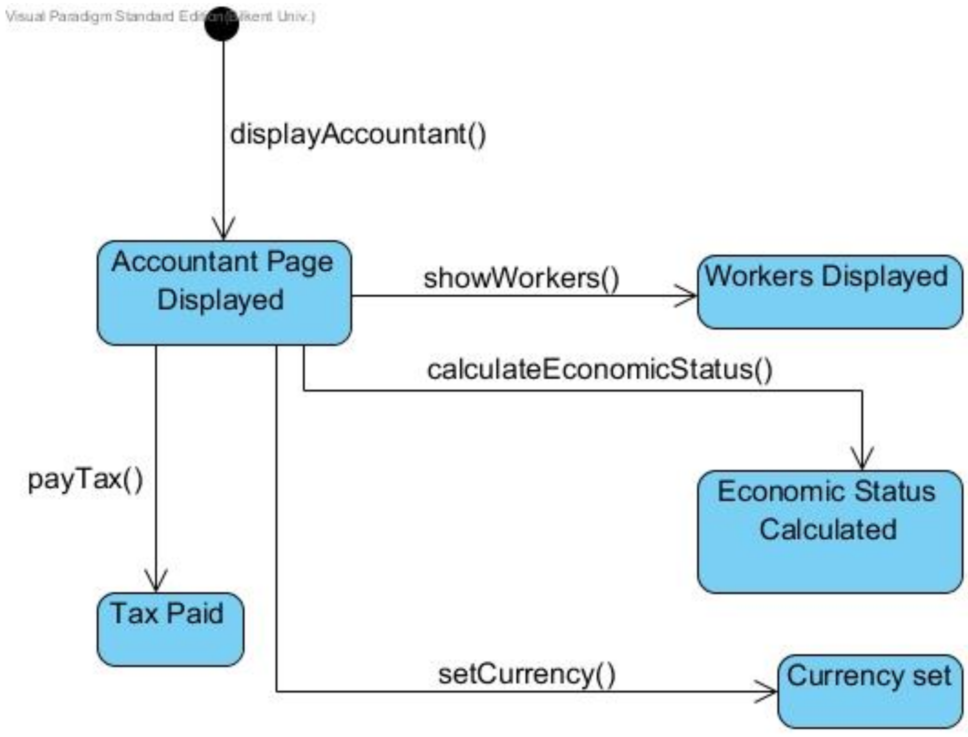

Visual Paradigm Standard Edition(Bilkent Univ.)
displayAccountant()
Accountant Page Displayed
showWorkers()
Workers Displayed
calculateEconomicStatus()
Economic Status Calculated
payTax()
Tax Paid
setCurrency()
Currency set

Figure 19: State chart diagram for Accountant

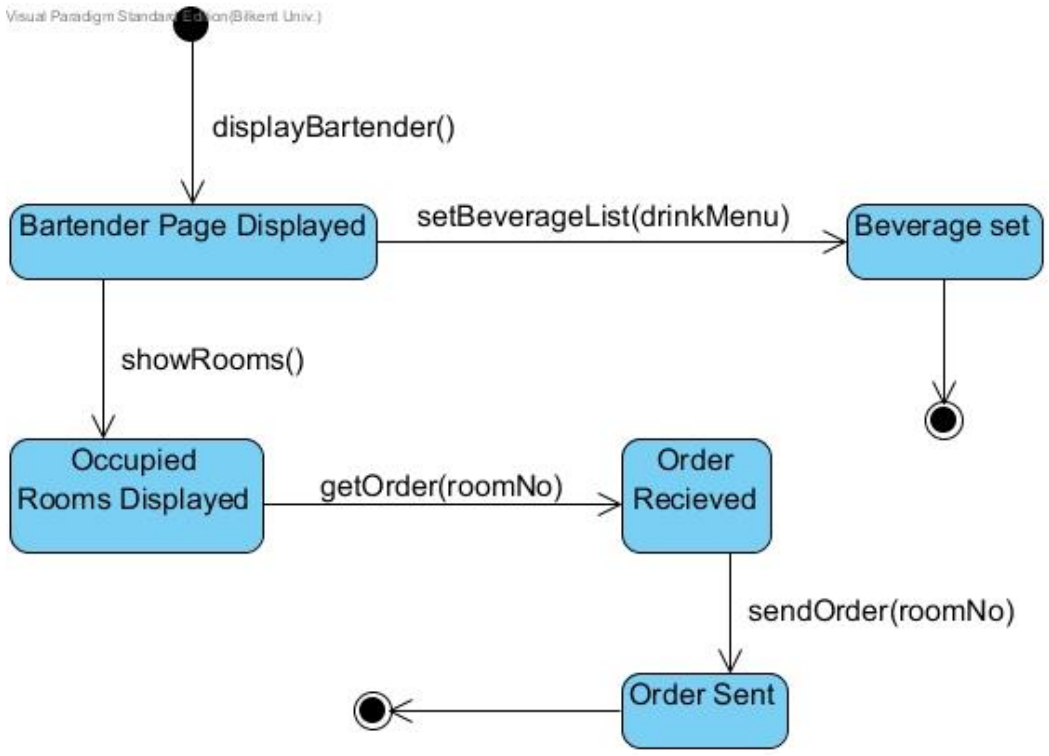


Figure 20: State chart diagram for Bartender

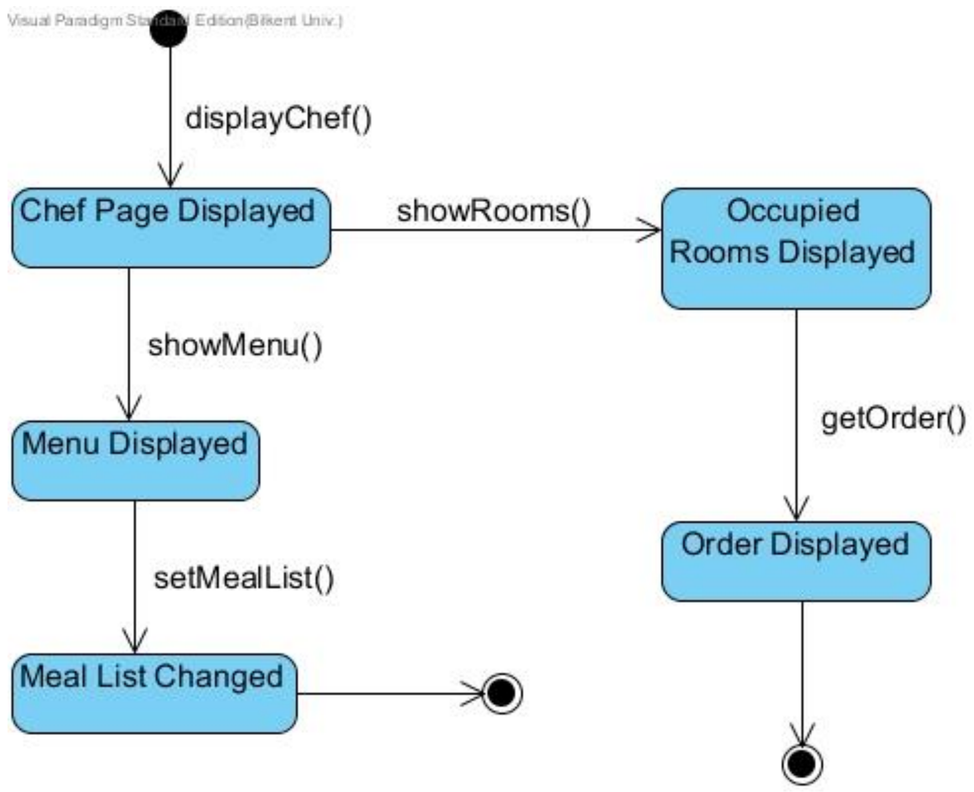


Figure 21: State chart diagram for Chef

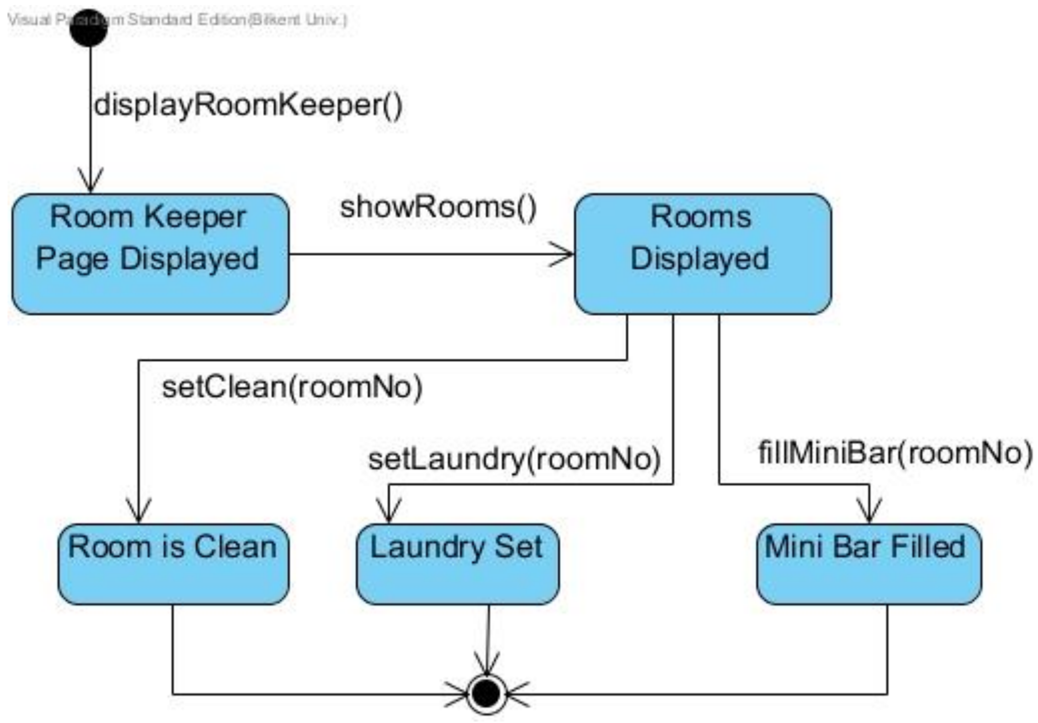


Figure 22: State chart diagram for Room Keeper

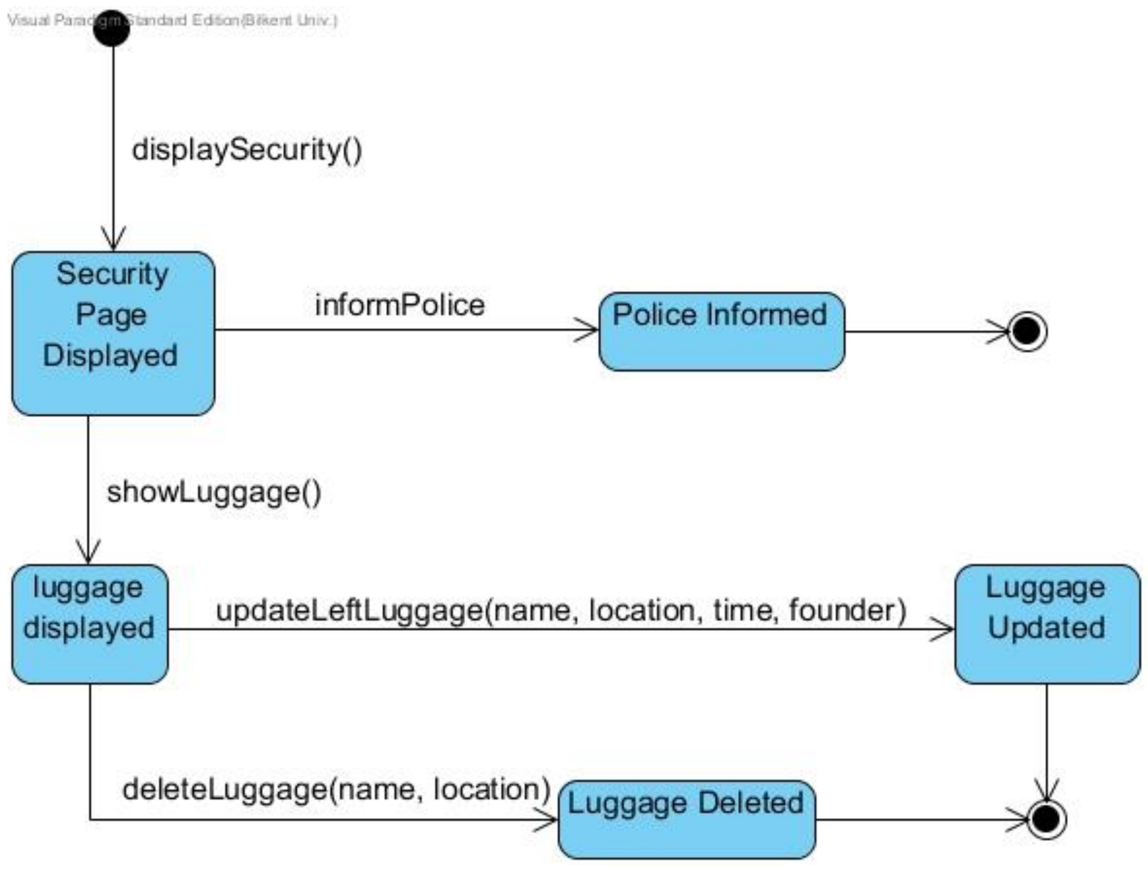


Figure 23: State chart diagram for Security

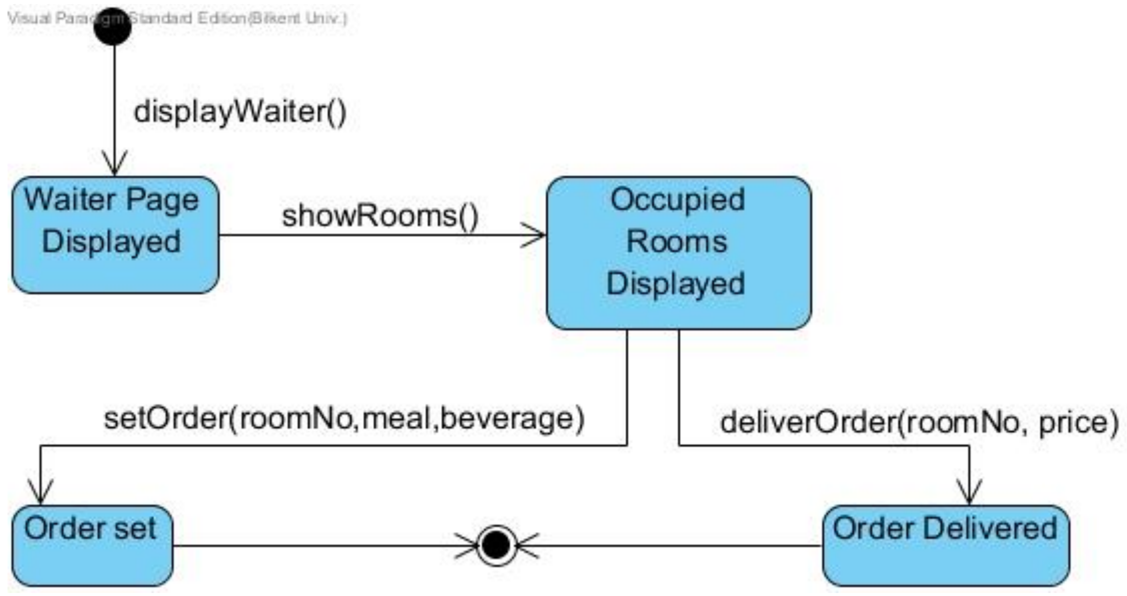

Visual Paradigm Standard Edition(Bilkent Univ.)
displayWaiter()
Waiter Page Displayed
showRooms()
Occupied Rooms Displayed
setOrder(roomNo,meal,beverage)
deliverOrder(roomNo, price)
Order set
Order Delivered

Figure 24: State chart diagram for Waiter

3.2.2 Sequence Diagram

In this section, there are sequence diagrams that are used to formalize the dynamic behaviour of the system and to visualize the communication among objects. In this diagrams, left most column represents the actor who initiates the use case. Labeled arrows represents the functions that actor or object send s to other object which functions are shown more clearly in class diagram.

In receptionist sequence diagram there are 2 combined fragments that are for if - else conditions that activity change among them.

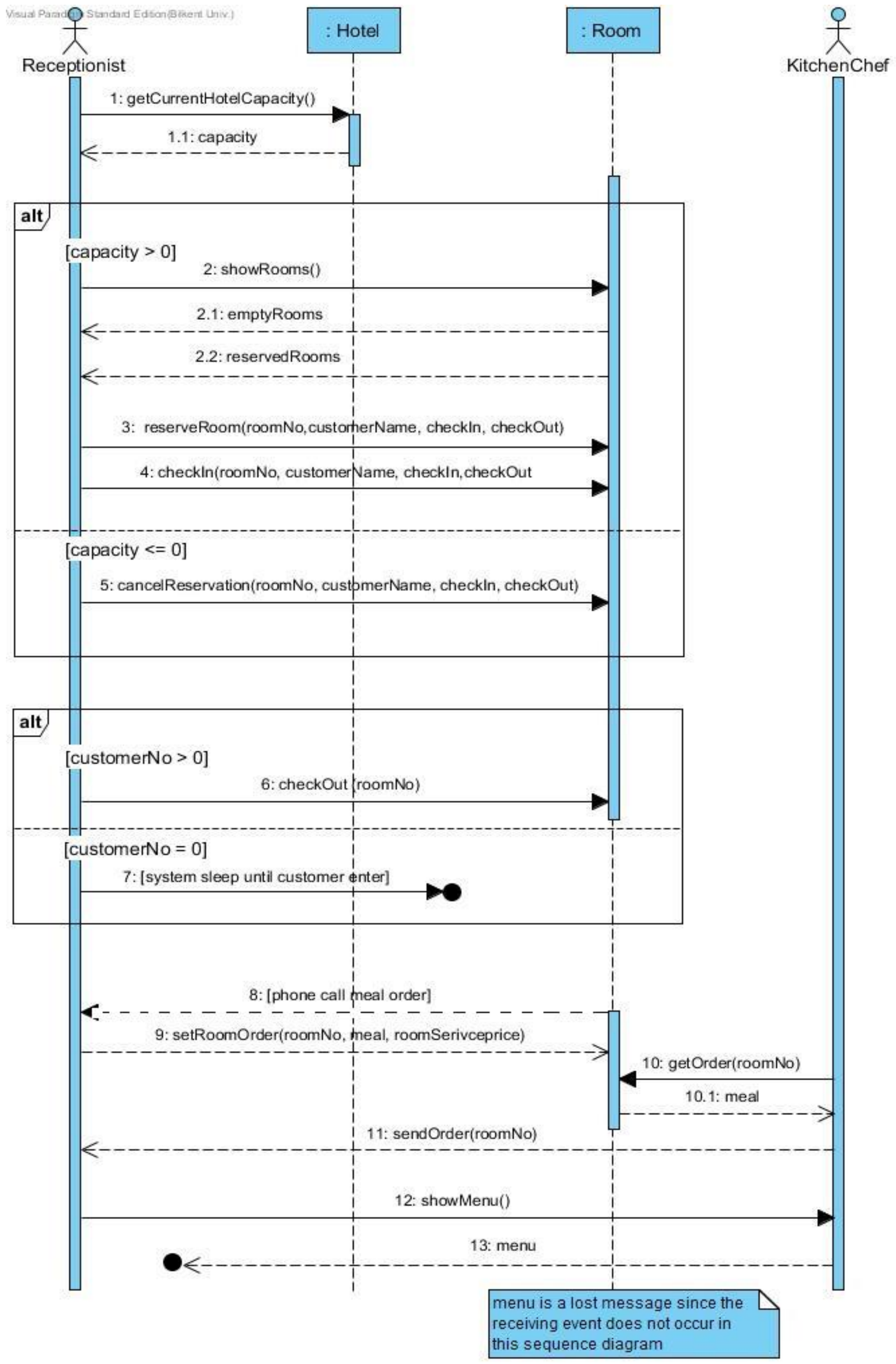
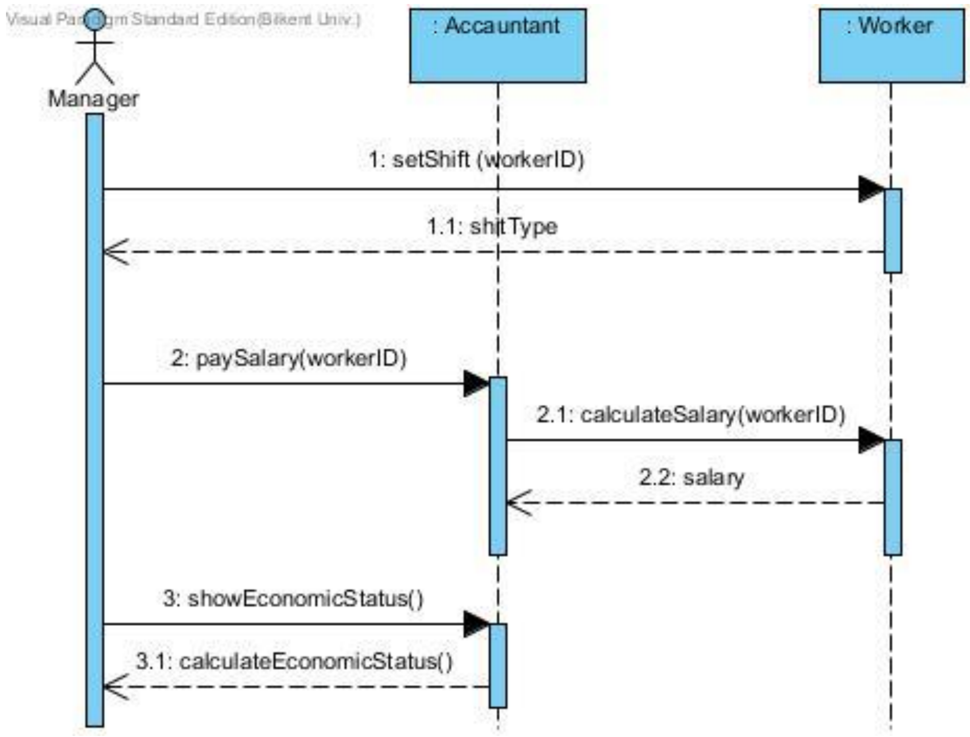
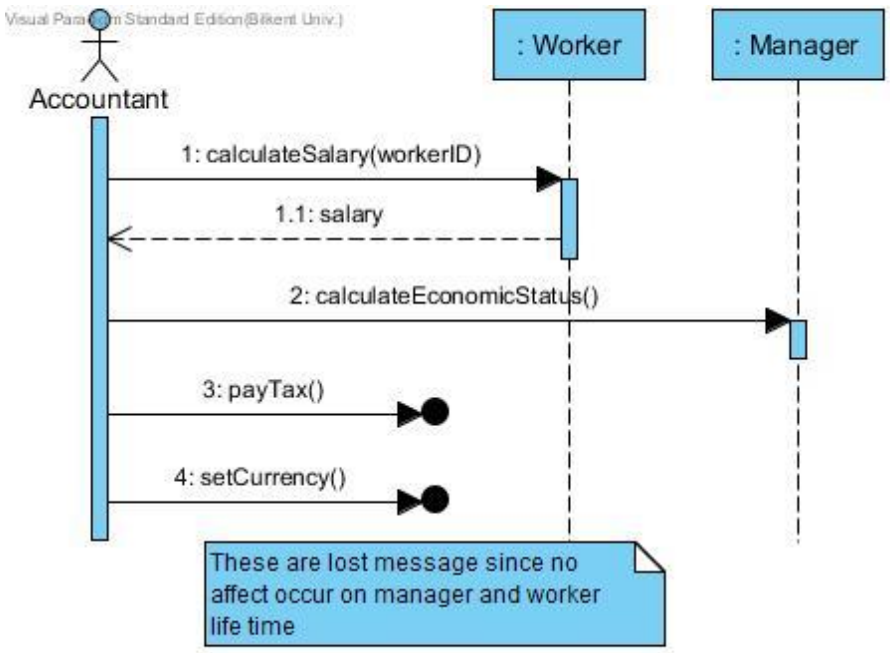

Visual Paradigm Standard Edition(Bilkent Univ.)
Receptionist
: Hotel
: Room
KitchenChef
1: getCurrentHotelCapacity()
1.1: capacity
alt
[capacity > 0]
2: showRooms()
2.1: emptyRooms
2.2: reservedRooms
3: reserveRoom(roomNo,customerName, checkIn, checkOut)
4: checkIn(roomNo, customerName, checkIn,checkOut
[capacity <= 0]
5: cancelReservation(roomNo, customerName, checkIn, checkOut)
alt
[customerNo > 0]
6: checkOut (roomNo)
[customerNo = 0]
7: [system sleep until customer enter]
8: [phone call meal order]
9: setRoomOrder(roomNo, meal, roomSerivceprice)
10: getOrder(roomNo)
10.1: meal
11: sendOrder(roomNo)
12: showMenu()
13: menu
menu is a lost message since the receiving event does not occur in this sequence diagram

Figure 25: Sequence diagram for Receptionist



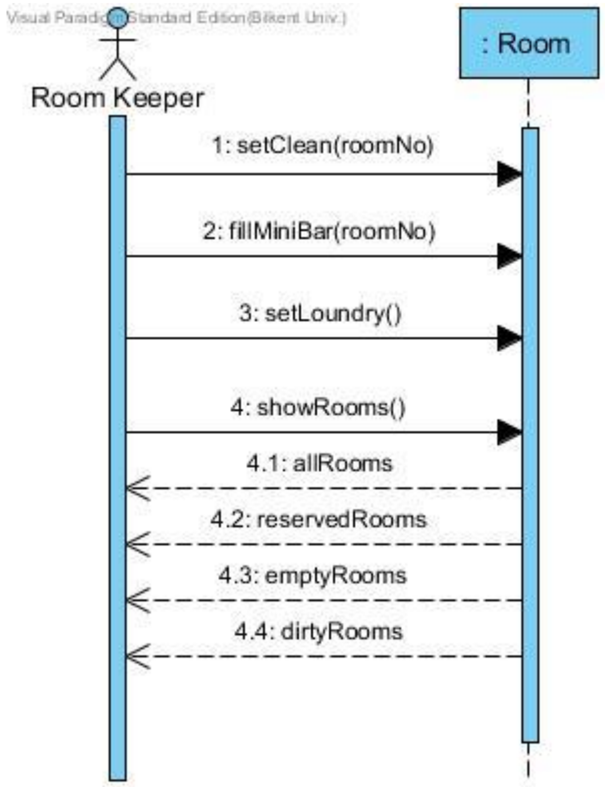
-

Figure 26: Sequence diagram for the Manager use case



Visual Paradigm Standard Edition(Bilkent Univ.)
Accountant
: Worker
: Manager
1: calculateSalary(workerID)
1.1: salary
2: calculateEconomicStatus()
3: payTax()
4: setCurrency()
These are lost message since no affect occur on manager and worker life time

Figure 27: Sequence diagram for the Accountant use case



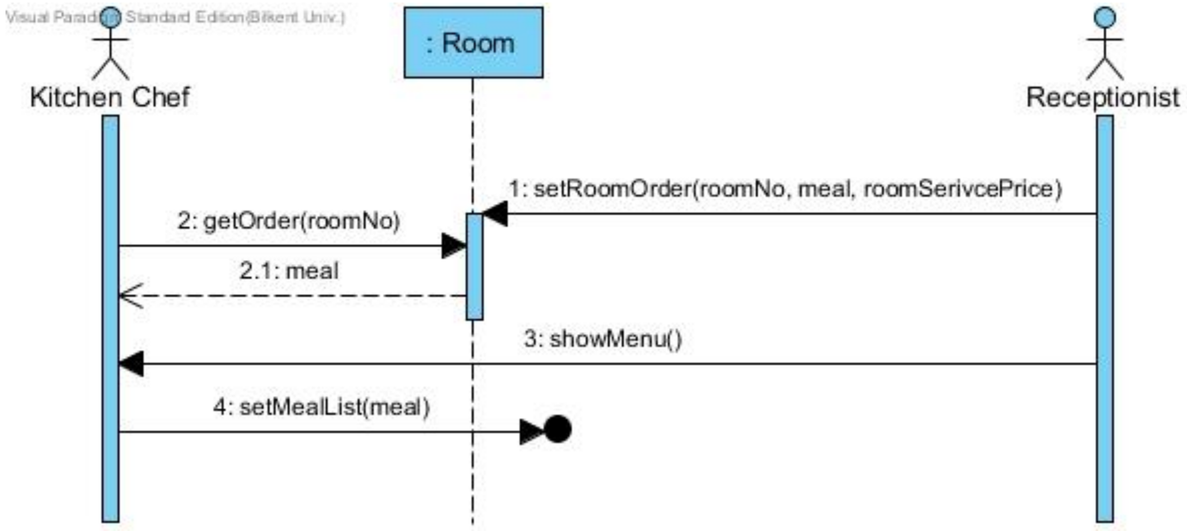

Visual Paradigm Standard Edition(Bilkent Univ.)
Kitchen Chef
: Room
Receptionist
1: setRoomOrder(roomNo, meal, roomSerivcePrice)
2: getOrder(roomNo)
2.1: meal
3: showMenu()
4: setMealList(meal)

Figure 28: Sequence diagram for the Room Keeper use case

Figure 29: Sequence diagram for the Kitchen Chef use case

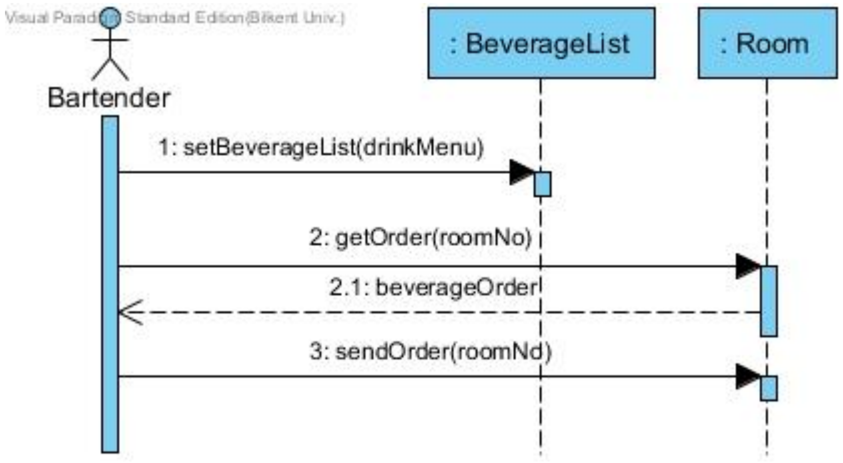


Figure 30: Sequence diagram for the Bartender use case

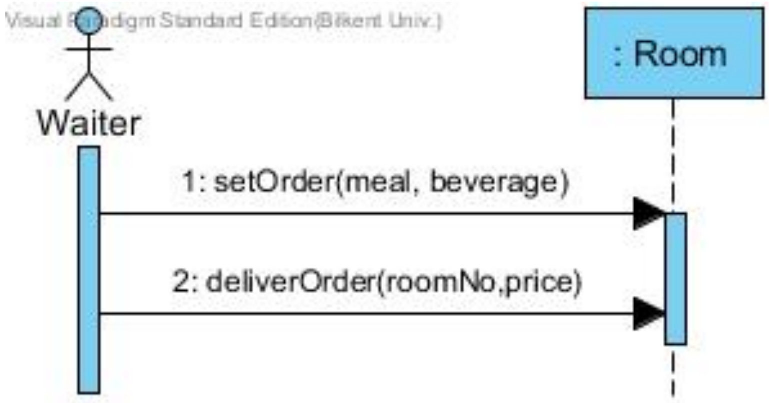

Visual Paradigm Standard Edition(Bilkent Univ.)
Waiter
: Room
1: setOrder(meal, beverage)
2: deliverOrder(roomNo,price)

Figure 31: Sequence diagram for the Waiter use case

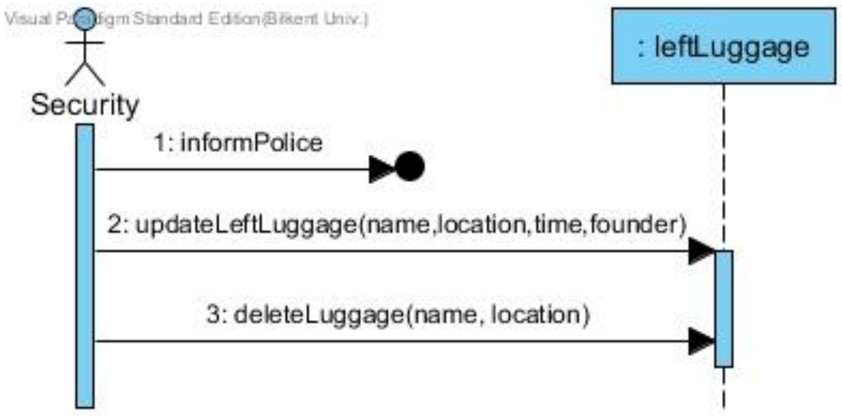

Visual Paradigm Standard Edition(Bilkent Univ.)
Security
: leftLuggage
1: informPolice
2: updateLeftLuggage(name,location,time,founder)
3: deleteLuggage(name, location)

Figure 32: Sequence diagram for the Security use case

4) Design

Auto Hotel is a hotel automation software system that keeps track of almost everything going on in the hotel. It provides several features for different employees, but in the system everything comes together to complete the all functionality of the hotel. With a simple and modest interface design, it provides an easy usage to the users. Auto Hotel aims to provide an easy and efficient way of controlling and keeping the track of all events happening in the hotel. Besides this, AutoHotel prevent any unregistered interaction about customers between employees. AutoHotel have 3-layer software architecture style that are presentation tier, business and data tier. Presentation tier is the user interface of employees see, business tier is control part of whole back-end system and data tier is online database that holds the hotel information in safety.

4.1 Design Goals

During this design phase, several elements will impact and shape the design of the automation system. Some of these elements are typically non-negotiable or finite resources, such as time, money, and manpower. Others, such as available technologies, knowledge, and skill that we learned in department, are dynamic and will vary throughout the development of system. So before programing the application, it is very important to determine design goals, which are going to shape the application. Therefore, we build our project on non-functional requirements of our system, which is explained in the analysis report.

4.1.1 End User Criteria:

Ease of Use: Auto Hotel is a hotel automation software system, and because of that, it is designed to be used by many different departments. Each department has different interfaces and different data accesses. That means there are many buttons, text fields, combo boxes and data. But this means difficulty for non-technical people such as hotel employees. Therefore, Auto Hotel is designed to extinguish all this complexity and offer a simple and efficient control of data. For all users, reachable options are grouped under the meaningful names that are represented with different tabs. Under each tab, choices are clearly stated. To increase the readability and to make it user friendly, interfaces designed in a way that they do not contain too many colors and symbols. Besides this, AutoHotel offers specific layouts according to job type of employees.

Ease of Learning: Since the user does not have any knowledge about the AutoHotel, it is essential to obtain information about the use of the application. To inform and help the user, there are help buttons that provides information about the system and guides the user. Any user can easily use the AutoHotel by following the instructions provided by the help button. The information in help buttons also varies according to job type of employees.

4.1.2 Maintenance Criteria:

Extensibility: Hotels are dynamic places that offer new options to the customers. In time they have future growth, newer, or sometimes they get smaller. Therefore, to support and easily settle those changes, Auto Hotel should support the extensibility. AutoHotel is designed to support adding functionalities and changes in hotel features like new rooms, new workers, new departments, etc. It is suitable to add new functionalities and modify existing structures because of its hierarchic structure. We try to aim minimize dependency between irrelevant classes and collect common things in interfaces and super classes.

Portability: AutoHotel is a desktop application that can be used in different environments. AutoHotel will be installed to the computers and the tablets in the hotel. It holds the information in online database so it does not require installation of database. For room keepers, it is required to use tablets because of transportation while visit the rooms. Therefore AutoHotel may be required to enhance.

Modifiability: It is important that future changes will be relatively easy to implement, since this decreases maintenance cost of software system. Corrections, improvements or adaptations of the AutoHotel to changes can be handled by 3-layer software architecture style and with extensibility feature.

Consistency: Consistency used for distributed system for shared data stores.The data consistency model specifies same data could be achieved and updated from different systems instantaneous. In our case, several employees take information from different computers at the same time about hotel. Therefore information should be achievable consistently. AutoHotel system eliminate this problem with using online database. When each user login the system, these datas will be loaded from database and whenever object is created, this information will be renew.

Concurrency: Many users can access data at the same time in AutoHotel software system and they can executable it simultaneously. Since our program is desktop application, each user can download and install this system into their computer to make changes simultaneously.

4.1.3 Performance Criteria:

Response Time: To provide a quick service to the customers of the hotel, Auto Hotel should respond the command very fast. Since different users access different data, and it is not a game that contains too much visual data to process, the application will respond to the user almost immediately.

4.1.4 Trade Offs:

Ease Of Use and Ease of Learning vs. Functionality: Since the hotels are crowded and there are too many things to be handled by different users, it is essential to make customers of the hotel happy. Therefore Auto Hotel is designed to be simple to avoid complex functionalities and increase the usability of the application, so that things should be done easily in a very short time of period for the customer satisfaction. In other words, priority of the usability is higher than the functionality.

Performance vs. Memory: In the hotels, there are many different departments and all these departments have too many data. As a matter of fact, hotel automation systems are based on data management. Auto Hotel, stores many important data such as customers' info, expenses, room info etc. Therefore the most important part of the system is keeping all the data safe. Auto Hotel saves big amount of data. Therefore it is expected to have a slightly slower application. In other words, memory is important than the high speed.

4.2 Subsystem Decomposition

Under this title, we will try to show the collection of classes, associations, operations, events and constraints which are related themselves from different layers. This decomposition is a type of 3-Tier Architecture and it consists of Interface, Application and Storage layers.

Actors will login and see the page according to their jobs as in shown Interface subsystem.Then, changes in user interfaces making by employees will manipulate databases through the objects on application subsystem. Therefore, we can realize that actors will interact with Interface layer, then the changes they made on this layer will make operations on object in the Application layer. Moreover, these operations will update Storage of the program and there will be 4 tables for storage datas which are customer, worker, room and hotel.

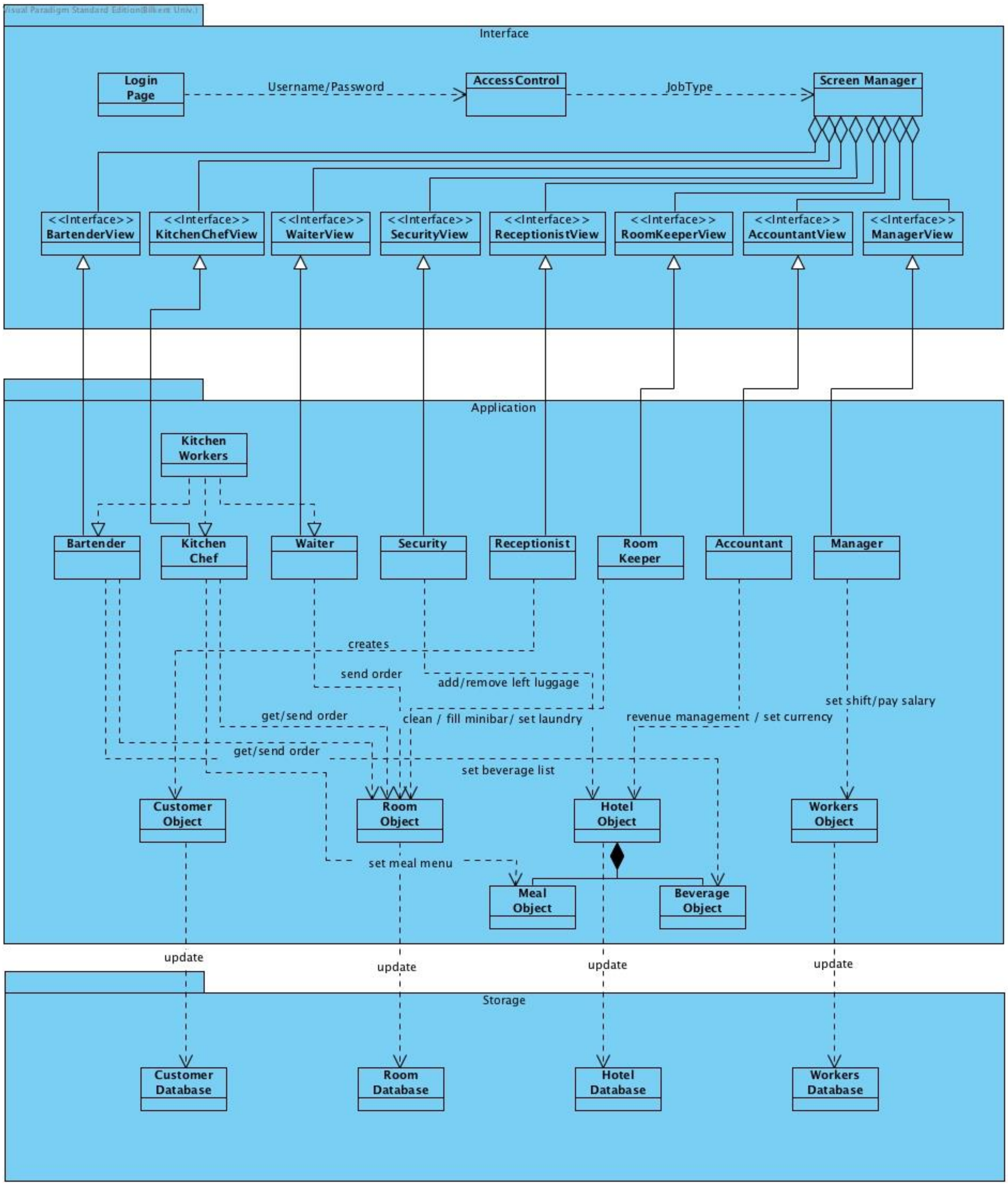
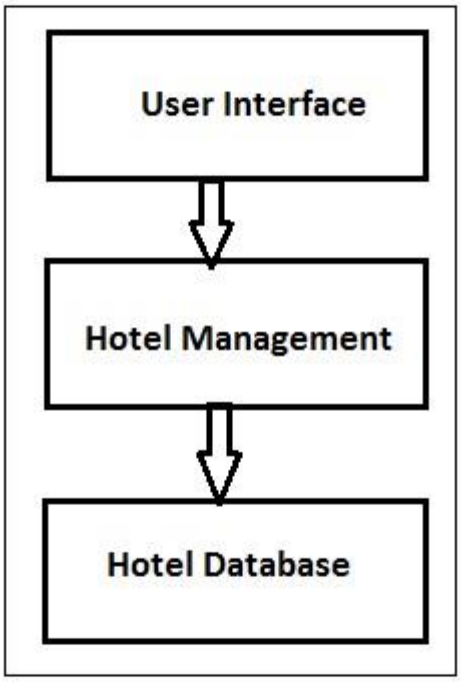


Figure 33: Diagram for the Subsystem Decomposition

4.3 Architectural Patterns

4.3.1 3-Layer Architecture Style

Our software system is 3-layer architecture.This system uses many layers for allocating the different responsibilities of a software subsystem and provides a model by which developers can create flexible and reusable applications. Application consists of 3 hierarchically ordered subsystems which are user interface, middleware and a database system. In interface subsystem, there are different pages according to users. In middleware subsystem, application, there are connections between user interface and database. This subsystem is required for updating databases and getting information from database. Our layer decomposition also proposes the closed architectural style, in which a layer can only access to the layer below it.



Therefore, 3-layer architecture is suitable for AutoHotel software system.

Figure 34: Layers Of System

4.3.2 Model-View- Controller

In this architectural style, the main approach is classifying the subsystems into three parts, called model, view and controller. The model, hotel database which is online database, stores data that is retrieved according to commands from the controller and displayed in the view. Hotel management constitute the controller in this design and can send commands to the model to update the hotel's states. It can also send commands to its associated view to change the information presented to employees from the database. User interface constitute the view part of MVC design, which shows data comes from controller to users which are employees in AutoHotel. For this reason, MVC system design is also suitable for our project.

4.4 Hardware/Software Mapping

One of the main design goals of the AutoHotel project is platform-independence which means implemented codes on developer's machine can be used on another machine without (or with minimal) changes. Thus, most subsystems have few specific hardware requirements. Exceptions to this are the database and some users access to system. The Database subsystem will be online therefore no specific hardware platform is required. Some employees' difference access to system is caused from their mobility such as room keeper and waiter.

The mobility problem can be solved by using tablets that are available with Android software. However in demo presentation of AutoHotel system, there will be no implementation for Android based AutoHotel system. Since our project requires the Java Runtime Environment because we implement this project by using Java. Also, our program requires a keyboard and a mouse for every actor in order to interact with the program.

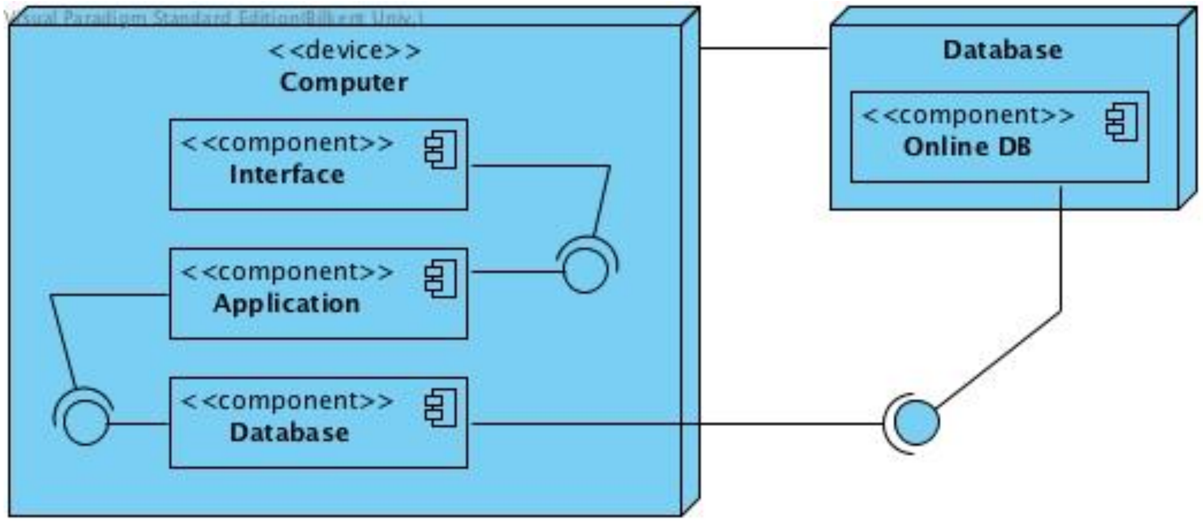


Figure 35: Deployment Diagram

4.5 Addressing Key Concerns

4.5.1 Persistent Data Management

AutoHotel Software will be stored data on online database because we need to access the proper real-time data in all computers in the hotel network as we stated in consistency part. When a worker makes an operation on objects, it will update the related database through objects, then this change will be shown in all computers. We will have four different type database table for four different objects which are customer, hotel, room and worker. These objects attributes will be column of database tables. When new object attributes is stored, database will be create new row to store them. The key point is object ids. Main search algorithms use this ids to find in database tables. There are also search algorithms that take name and room number as parameters in order to find specific element.

Our data will be downloaded from our online database when an actor logins the program. After that, if there is a change on database and new object is created, the software get these updates automatically from our online database, then send these updates to the screen of related actor.

4.5.2 Access Control and Security

In our automation program, we will implement an authentication system, we will use the worker database to keep information of workers' username and passwords. This database will be online and also contains all the information about workers. According to our design, workers will reach their related screens by entering their login information. Therefore, we will overcome the configuration of access control by using security system. An actor cannot change his username or password by himself, this change can be made by manager only, therefore, by this way, we expect to resist most of the security breaches.

4.5.3 Global Software Control

In AutoHotel software system, it is aimed establishing consistency among data from a source to a target and also it is aimed concurrency which means that many users can access data at the same time. Subsystems synchronize with using two-way file synchronization, updated files are copied in both directions. It is required for the purpose of keeping the two or more computers' datas identical to each other. We aimed the continuous harmonization of the data over time with updated data which is circulating among object periodically.

4.5.4 Boundary Conditions

The program will give an error if the data which is downloaded from online database is missing or corrupted. This situation can be happened when there is interruption in network connection.

This program is an automation software, therefore, actors will use the program all their shift long. When the shifts are end, actors will logout and close their screens. Then, login screen will be appeared for the new employee who will work at next shift.

The program give an error when manager could not pay salaries of employees with sending email to bank. This situation can also be happened when there is interruption in network connection.

Employees enter their passwords incorrectly only for 3 times in order to provide safety. After this situation, program will be locked and employees should be take new password from accountant.

5) Object Design

5.1 Pattern Applications

In AutoHotel system we use design pattern for general repeatable solution to a commonly occurring problems. We try to solve certain problems with software design techniques which are provided by design patterns. During this project, 3 developers work together in every step and try to be in harmony. Design patterns allow us to communicate using well-known, well understood names for software interactions.

There are structural and behavioral design patterns in AutoHotel system. For structural design patterns which is about class and object decomposition, we use Facade design pattern. For behavioral design patterns which is object communications, we use Strategy and Observer design pattern.

5.1.1 Facade Pattern

In AutoHotel Project, we used facade pattern as a structural pattern in order to hide system's complexity and provide an easier interface architecture. Firstly, facade pattern made our program; more readable, more flexible and easier to extend. By this way, client can easily see his requirements and developers can easily add new interfaces to the software. We will create a single simplified interface class for the subsystems, then encapsulate this subsystems with a wrapper class. Therefore, we decrease the complexity and all interface components will collaborate under the Facade.

Our interface diagram does not only show our interfaces but also it illustrates the facade pattern of AutoHotel project. Our interfaces are encapsulated and they are realized on "ScreenManager" class. By this way, it is easy to add new interfaces to the project through "ScrrenManager" class. Besides this, we separate interface components from actual classes in order to make easy to control and decode the exceptions. Interface classes and actual object classes can be evaluated separately. As we stated, we manage all interface classes from ScreenManager.

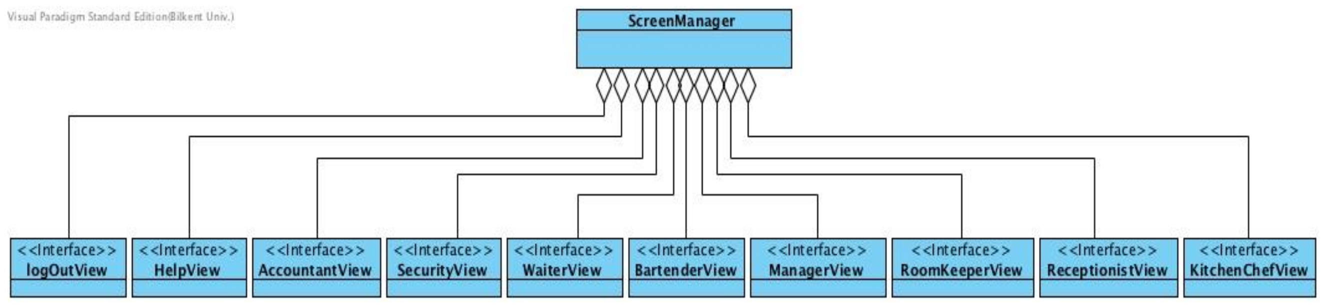

Visual Paradigm Standard Edition(Bilkent Univ.)
ScreenManager
<<Interface>> logOutView
<<Interface>> HelpView
<<Interface>> AccountantView
<<Interface>> SecurityView
<<Interface>> WaiterView
<<Interface>> BartenderView
<<Interface>> ManagerView
<<Interface>> RoomKeeperView
<<Interface>> ReceptionistView
<<Interface>> KitchenChefView

Figure 35: Facade Pattern Diagram

5.1.2 Observer Pattern

In Auto Hotel Observer pattern is used because there is one-to-many relationship between objects such as if one object is modified, its dependent objects have to be notified automatically. We have different users who can make changes on room objects and all these users should see the results, which are caused by actions of different users on the objects, immediately. Therefore, we used observer pattern. In this pattern, Room object is the subject and worker object is the observer. Any change in the room object will be seen by all kinds of worker objects such as Manager, Receptionist, Waiter to name but a few.

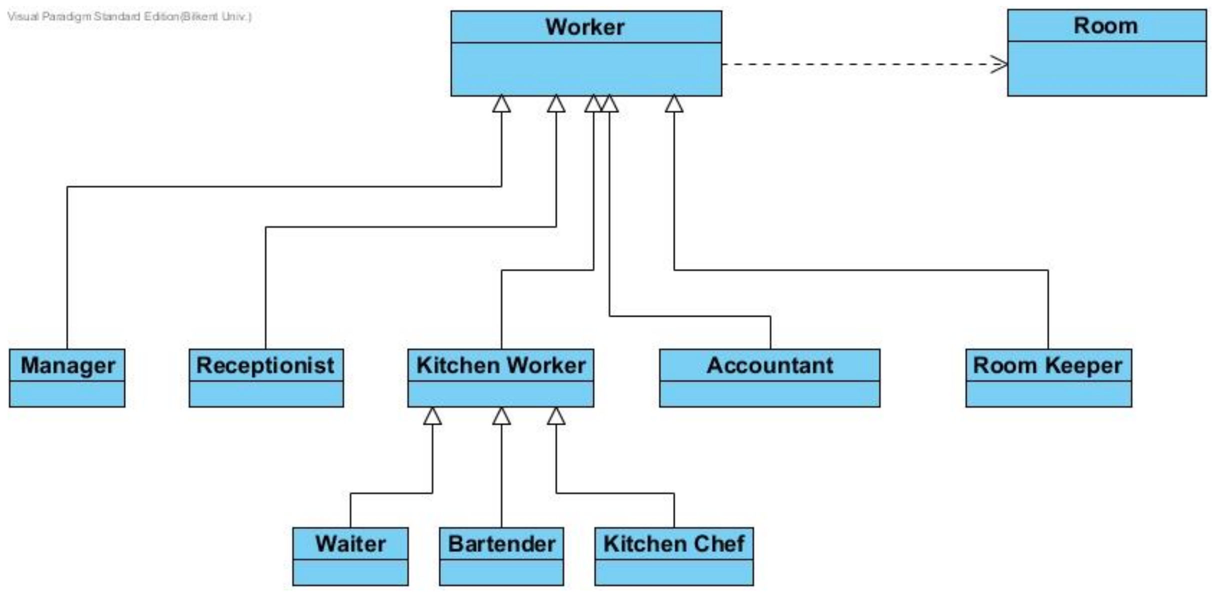


Figure 36: Observer Pattern Diagram

5.1.3 Strategy Pattern

Strategy pattern lets the algorithm vary independently from the clients which are different employees in AutoHotel case, that use it. In design patterns, ‘policy’ decide which algorithm will be use among different conditions. In AutoHotel case, interfaces and scope of applications differs according to job type. There are different algorithms for existing jobs. Therefore in AutoHotel case, policy will depend on employee ids which are indicate job types. When employees log in with their special ids, controller detect the job type and report it policy to perform appropriate algorithm with this signature. Policy has interchangeability within different algorithms for different employee type. Strategy pattern design enables us the algorithm's behavior to be selected at runtime.

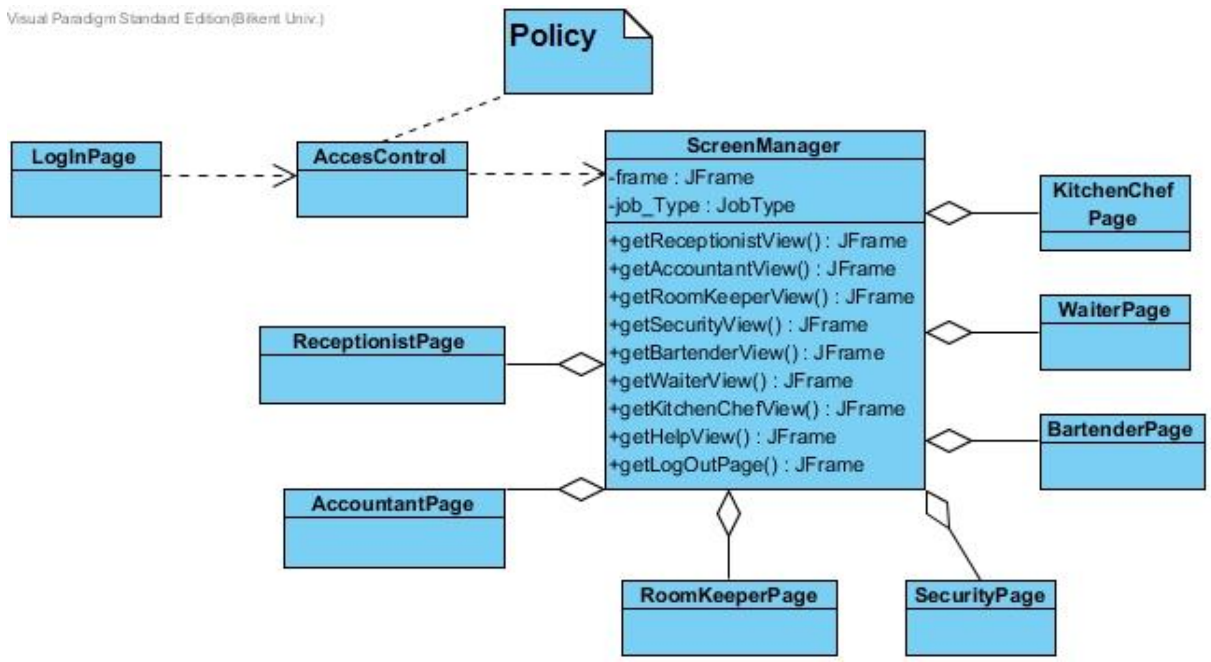
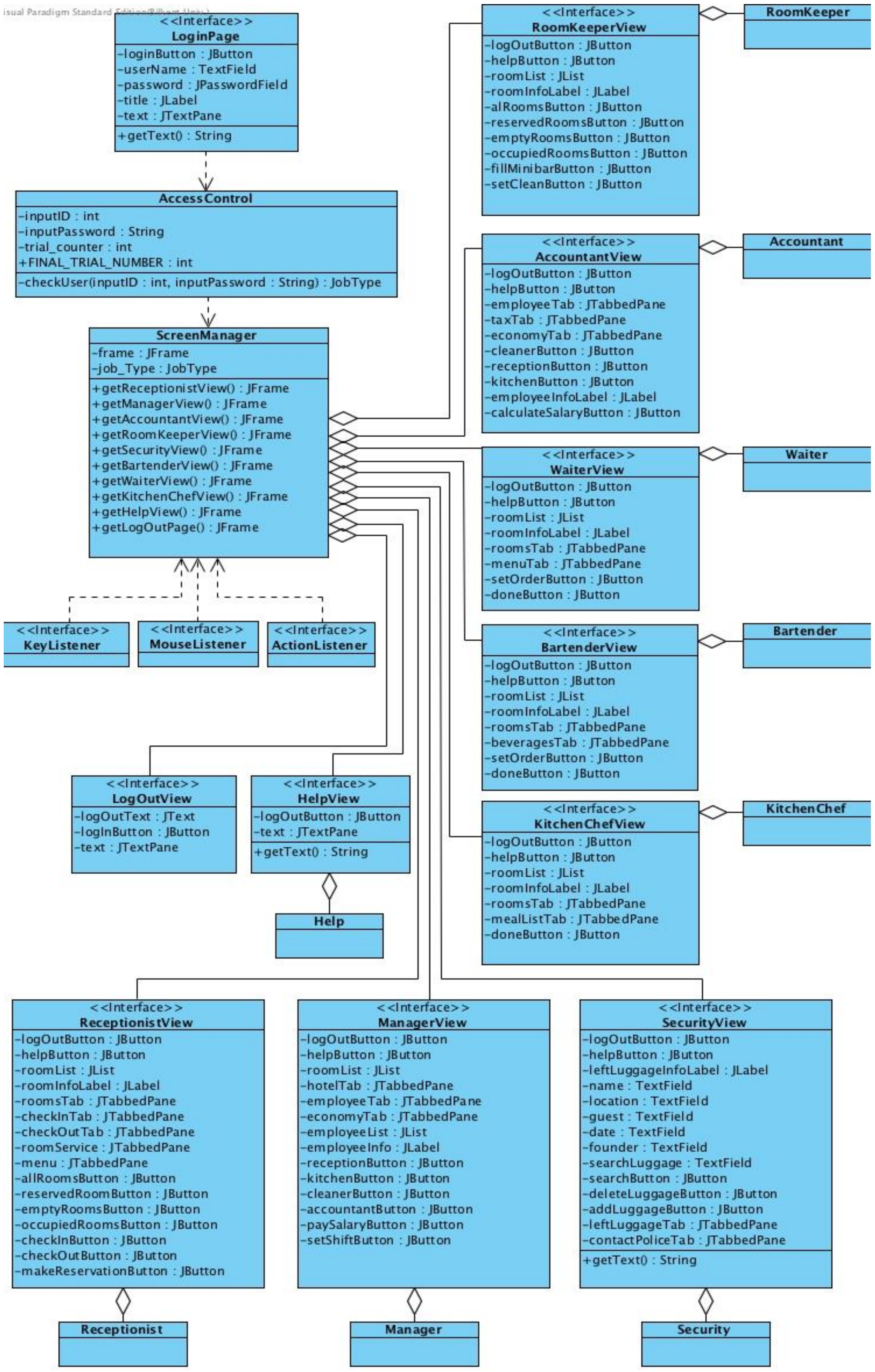

Visual Paradigm Standard Edition(Bilkent Univ.)
Policy
LogInPage
AccesControl
ScreenManager
-frame : JFrame
-job_Type : JobType
+getReceptionistView() : JFrame
+getAccountantView() : JFrame
+getRoomKeeperView() : JFrame
+getSecurityView() : JFrame
+getBartenderView() : JFrame
+getWaiterView() : JFrame
+getKitchenChefView() : JFrame
+getHelpView() : JFrame
+getLogOutPage() : JFrame
KitchenChef Page
WaiterPage
BartenderPage
ReceptionistPage
AccountantPage
RoomKeeperPage
SecurityPage

Figure 36: Strategy Pattern Diagram

5.2 Class Interfaces

We design our interface classes separate from our entity objects which are Receptionist, RoomKeeper, KitchenChef, Bartender, Waiter, Security, Accountant and Manager. Interface classes are boundary objects. We designed different pages and different classes for these views. When interface attributes changes, these changes affects entity objects. We designed in this way in order to make hierarchy, increase understandability and make changes more easily.


isual Paradigm Standard Edition(Bilkent Univ.)
<<Interface>> LoginPage
-loginButton : JButton
-userName : TextField
-password : JPasswordField
-title : JLabel
-text : JTextPane
+getText() : String
AccessControl
-inputID : int
-inputPassword : String
-trial_counter : int
+FINAL_TRIAL_NUMBER : int
-checkUser(inputID : int, inputPassword : String) : JobType
ScreenManager
-frame : JFrame
-job_Type : JobType
+getReceptionistView() : JFrame
+getManagerView() : JFrame
+getAccountantView() : JFrame
+getRoomKeeperView() : JFrame
+getSecurityView() : JFrame
+getBartenderView() : JFrame
+getWaiterView() : JFrame
+getKitchenChefView() : JFrame
+getHelpView() : JFrame
+getLogOutPage() : JFrame
<<Interface>> KeyListener
<<Interface>> MouseListener
<<Interface>> ActionListener
<<Interface>> LogOutView
-logOutText : JText
-logInButton : JButton
-text : JTextPane
<<Interface>> HelpView
-logOutButton : JButton
-text : JTextPane
+getText() : String
Help
<<Interface>> RoomKeeperView
-logOutButton : JButton
-helpButton : JButton
-roomList : JList
-roomInfoLabel : JLabel
-alRoomsButton : JButton
-reservedRoomsButton : JButton
-emptyRoomsButton : JButton
-occupiedRoomsButton : JButton
-fillMinibarButton : JButton
-setCleanButton : JButton
RoomKeeper
<<Interface>> AccountantView
-logOutButton : JButton
-helpButton : JButton
-employeeTab : JTabbedPane
-taxTab : JTabbedPane
-economyTab : JTabbedPane
-cleanerButton : JButton
-receptionButton : JButton
-kitchenButton : JButton
-employeeInfoLabel : JLabel
-calculateSalaryButton : JButton
Accountant
<<Interface>> WaiterView
-logOutButton : JButton
-helpButton : JButton
-roomList : JList
-roomInfoLabel : JLabel
-roomsTab : JTabbedPane
-menuTab : JTabbedPane
-setOrderButton : JButton
-doneButton : JButton
Waiter
<<Interface>> BartenderView
-logOutButton : JButton
-helpButton : JButton
-roomList : JList
-roomInfoLabel : JLabel
-roomsTab : JTabbedPane
-beveragesTab : JTabbedPane
-setOrderButton : JButton
-doneButton : JButton
Bartender
<<Interface>> KitchenChefView
-logOutButton : JButton
-helpButton : JButton
-roomList : JList
-roomInfoLabel : JLabel
-roomsTab : JTabbedPane
-mealListTab : JTabbedPane
-doneButton : JButton
KitchenChef
<<Interface>> ReceptionistView
-logOutButton : JButton
-helpButton : JButton
-roomList : JList
-roomInfoLabel : JLabel
-roomsTab : JTabbedPane
-checkInTab : JTabbedPane
-checkOutTab : JTabbedPane
-roomService : JTabbedPane
-menu : JTabbedPane
-allRoomsButton : JButton
-reservedRoomButton : JButton
-emptyRoomsButton : JButton
-occupiedRoomsButton : JButton
-checkInButton : JButton
-checkOutButton : JButton
-makeReservationButton : JButton
Receptionist
<<Interface>> ManagerView
-logOutButton : JButton
-helpButton : JButton
-roomList : JList
-hotelTab : JTabbedPane
-employeeTab : JTabbedPane
-economyTab : JTabbedPane
-employeeList : JList
-employeeInfo : JLabel
-receptionButton : JButton
-kitchenButton : JButton
-cleanerButton : JButton
-accountantButton : JButton
-paySalaryButton : JButton
-setShiftButton : JButton
Manager
<<Interface>> SecurityView
-logOutButton : JButton
-helpButton : JButton
-leftLuggageInfoLabel : JLabel
-name : TextField
-location : TextField
-guest : TextField
-date : TextField
-founder : TextField
-searchLuggage : TextField
-searchButton : JButton
-deleteLuggageButton : JButton
-addLuggageButton : JButton
-leftLuggageTab : JTabbedPane
-contactPoliceTab : JTabbedPane
+getText() : String
Security

Class Interfaces

ScreenManager

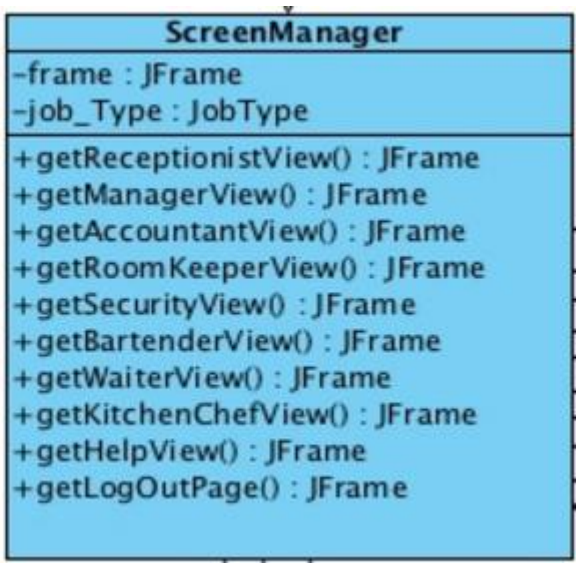

ScreenManager
-frame : JFrame
-job_Type : JobType
+getReceptionistView() : JFrame
+getManagerView() : JFrame
+getAccountantView() : JFrame
+getRoomKeeperView() : JFrame
+getSecurityView() : JFrame
+getBartenderView() : JFrame
+getWaiterView() : JFrame
+getKitchenChefView() : JFrame
+getHelpView() : JFrame
+getLogOutPage() : JFrame

Figure 39: ScreenManager Interface Class

Attributes:

private frame: JFrame: This attribute holds the frame object.

private job_Type: JobType: This attribute holds the job type of worker in order to open accurate screen for user.

Methods:

private getReceptionistView(): JFrame: This method open the receptionist's screen.

private getManagerView() :JFrame: This method open the manager's screen.

private getAccountantView() :JFrame: This method open the accountant's screen.

private getRoomKeeperrView() :JFrame: This method open the room keeper's screen.

private getSecurityView() :JFrame: This method open the security screen.

private getBartenderView() :JFrame: This method open the bartender's screen.

private getWaiterView() :JFrame: This method open the waiter's screen.

private getKitchenChefView() :JFrame: This method open the kitchen chef's screen.

private getHelpView() :JFrame: This method open the help screen.

private getLogOutView() :JFrame: This method open the logout screen.

BartenderView

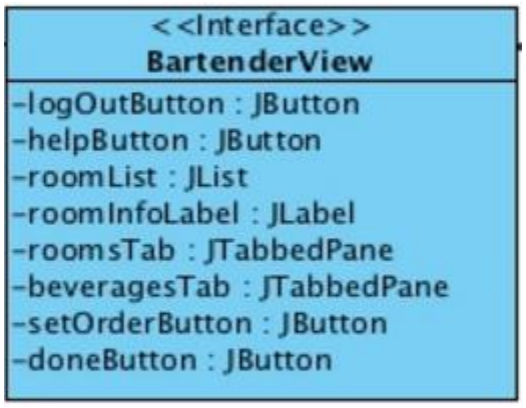


Figure 40: BartenderView Interface Class

Attributes:

private logOutButton: JButton: This button provides a secure exit to the user.

private helpButton: JButton: This button holds the information that helps and guides the user.

private roomList: JList: This attribute is used for to hold the list of rooms.

private roomInfoLabel: JLabel: This attribute is used for to provide a space to display room info.

private roomsTab: JTabbedPane: This attribute is used for hold the information of room's orders.

private beveragesTab: JTabbedPane: This attribute is used for hold the beverage menu.

private setOrderButton: JButton: This button provides to create a order for a room

private doneButton: JButton: This button provides to use information which is entered by waiter.

WaiterView

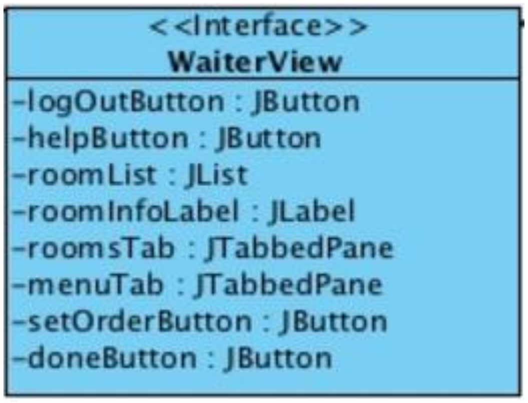


Figure 41:WaiterView Interface Class

Attributes:

private logOutButton: JButton: This button provides a secure exit to the user.

private helpButton: JButton: This button holds the information that helps and guides the user.

private roomList: JList: This attribute is used for holding the list of rooms.

private roomInfoLabel: JLabel: This attribute is used for providing a space to display room info.

private roomsTab: JTabbedPane: This attribute is used for hold the information of room's orders.

private menuTab: JTabbedPane: This attribute is used for hold the meal menu and beverage menu.

private setOrderButton: JButton: This button provides to create a order for a room

private doneButton: JButton: This button provides to use information which is entered by waiter.

KitchenChefView

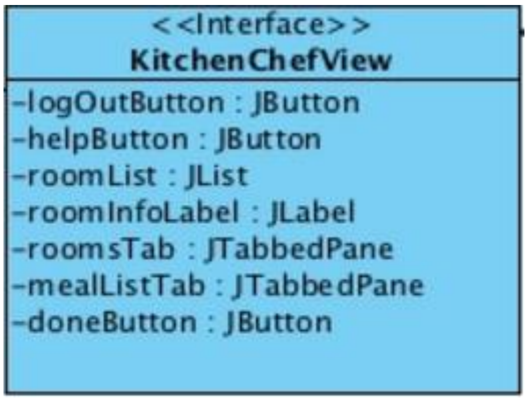


Figure 42: KitchenChefView Interface Class

Attributes:

private logOutButton: JButton: This button provides a secure exit to the user.

private helpButton: JButton: This button holds the information that helps and guides the user.

private roomList: JList: This attribute is used for to hold the list of rooms.

private roomInfoLabel: JLabel: This attribute is used for to provide a space to display room info.

private roomsTab: JTabbedPane: This attribute is used for hold the information of room’s orders.

private mealListTab: JTabbedPane: This attribute is used for hold the meal menu.

private doneButton: JButton: This button provides to use information which is entered by waiter.

HelpView

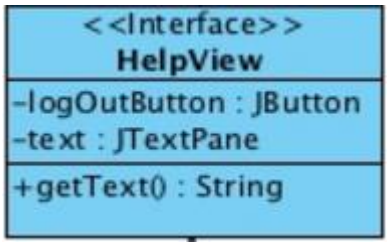


Figure 43: HelpView Interface Class

Attributes:

private logOutButton: JButton: This button provides a secure exit to the user.

private text: JTextPane: This attribute provides helper text for workers.

Methods:

getText() :String: This operation read text from a “.txt” file and write on the screen according to worker’s job type.

AccountantView

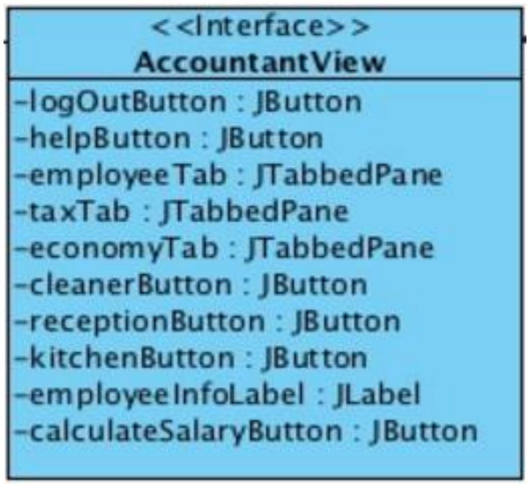


Figure 44: AccountantView Interface Class

Attributes:

private logOutButton: JButton: This button provides a secure exit to the user.

private helpButton: JButton: This button holds the information that helps and guides the user.

private employeeTab: JTabbedPane: This attribute is used for to reach the informations about the employees in a different tab.

private taxTab: JTabbedPane: This attribute is used for to reach the informations about the taxes in a different tab.

private ecconomyTab: JTabbedPane: This attribute is used for to reach the informations about the economic status of the hotel in a different tab.

private cleanerButton: JButton: This button is used for to display cleaners.

private receptionButton: JButton: This button is used for to display the workers from the reception department.

private kitchenButton: JButton: This button is used for to display the workers from the kitchen department.

private employeeInfoLabel: JLabel: This attribute is used for to provide a space to display worker info.

private calculateSalaryButton: JButton: This button is used for to calculate the salary of workers

RoomKeeperView

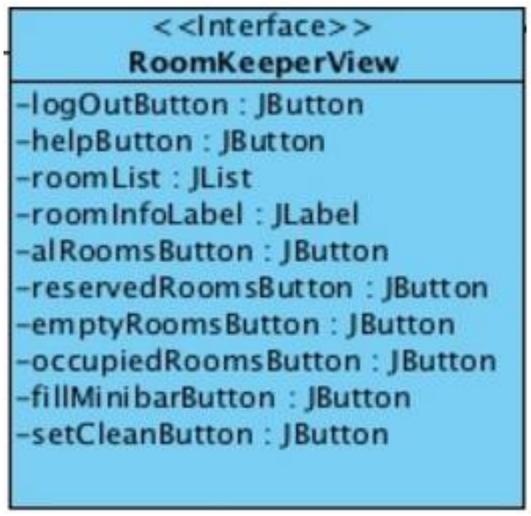

<<Interface>>
RoomKeeperView
-logOutButton : JButton
-helpButton : JButton
-roomList : JList
-roomInfoLabel : JLabel
-alRoomsButton : JButton
-reservedRoomsButton : JButton
-emptyRoomsButton : JButton
-occupiedRoomsButton : JButton
-fillMinibarButton : JButton
-setCleanButton : JButton

Figure 45: RoomKeeperView Interface Class

Attributes:

private logOutButton: JButton: This button provides a secure exit to the user.

private helpButton: JButton: This button holds the information that helps and guides the user.

private roomList: JList: This attribute is used for to hold the list of rooms.

private roomInfoLabel: JLabel: This attribute is used for to provide a space to display room info.

private allRoomsButton: JButton: This button is used for to display the information about all the rooms in the hotel.

private reservedRoomsButton: JButton: This button is used for to display the information about the reserved rooms in the hotel

private emptyRoomsButton: JButton: This button is used for to display the information about the empty rooms in the hotel

private occupiedRoomButton: JButton: This button is used for to display the information about the occupied rooms in the hotel

private fillMiniBarButton: JButton: This button is used for to change the status of the mini bar.

private setCleanButton: JButton: This button is used for to change the cleanliness of the room.

ManagerView

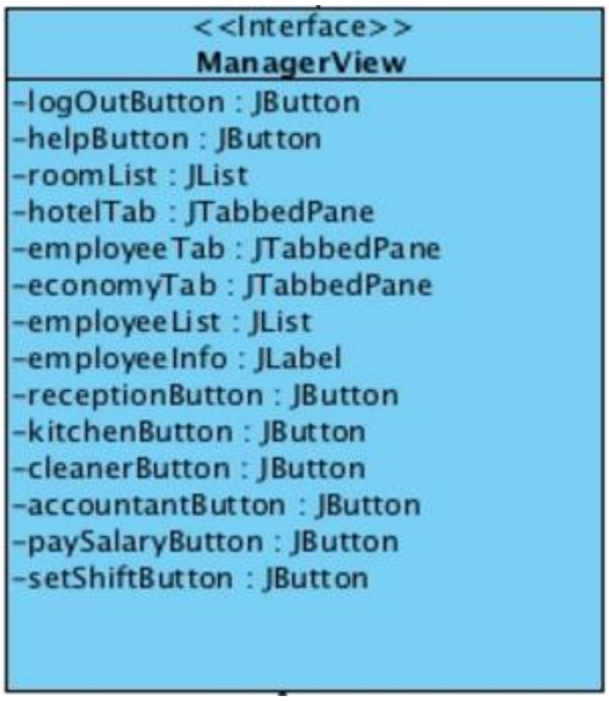


Figure 46: ManagerView Interface Class

Attributes:

private logOutButton: JButton: This button provides a secure exit to the user.

private helpButton: JButton: This button holds the information that helps and guides the user.

private roomList: JList: This attribute is used for to hold the list of rooms.

private hotelTab: JTabbedPane: This attribute is used for to reach the informations about the hotel in a different tab.

private employeeTab: JTabbedPane: This attribute is used for to reach the informations about the employees in a different tab.

private ecconomyTab: JTabbedPane: This attribute is used for to reach the informations about the economic status of the hotel in a different tab.

private employeeList: JList: This attribute is used for to hold the list of all the employees that are working in the different departments of the hotel.

private employeeInfoLabel: JLabel: This attribute is used for to provide a space to display worker info.

private receptionButton: JButton: This button is used for to display the workers from the reception department.

private kitchenButton: JButton: This button is used for to display the workers from the kitchen department.

private cleanerButton: JButton: This button is used for to display cleaners.

private accountantButton: JButton: This button is used for to display information about accountant.

private paySalaryButton: JButton: This button is used for to approve payment of the salaries of workers calculated by the accountant.

private setShiftButton: JButton: This button is used for to change the working hours of the employees.

AccessControl

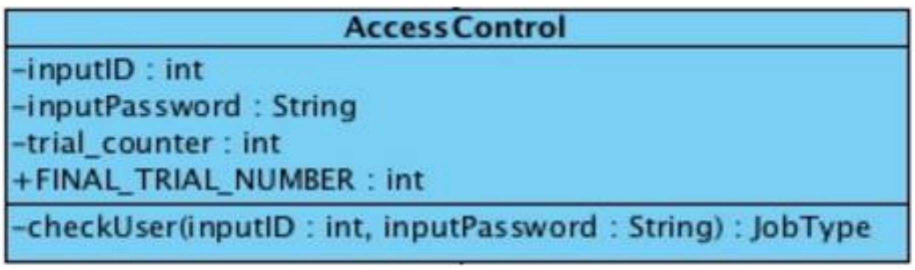

AccessControl
-inputID : int
-inputPassword : String
-trial_counter : int
+FINAL_TRIAL_NUMBER : int
-checkUser(inputID : int, inputPassword : String) : JobType

Figure 47: AccessControl Class

Attributes:

private inputID: int: This attribute is used for to hold and compare the id of the user who is trying to login the system.

private inputPasword: String: This attribute is used for to hold and compare the password of the user who is trying to login the system.

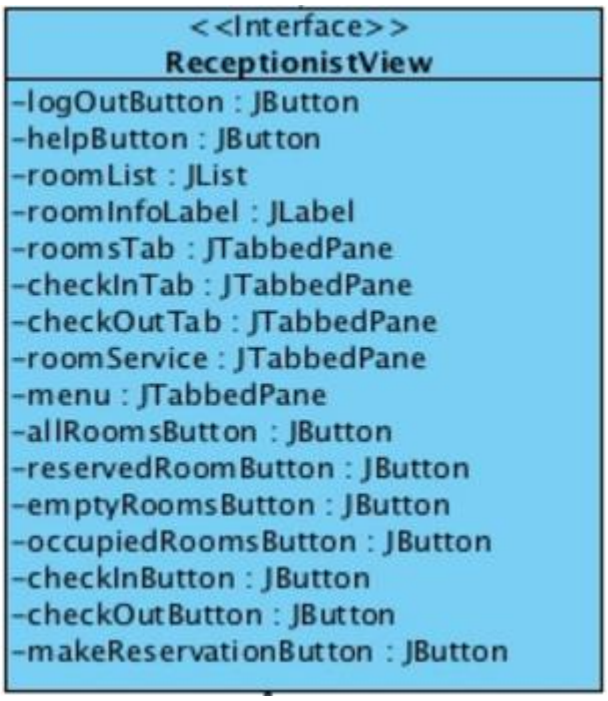
private trial_counter: int: This attribute is used for to count the number of login trials made by the user.

public FINAL_TRIAL_NUMBER: int: This attribute is used for to hold the maximum number of trials that can be made by a user.

Methods:

private checkUser(int inputID, String inputPassword): JobType: This method takes the inputID and inputPassword entered by the user, and compares it with the user ids and passwords that are stored in the database. If the entered id and password are valid, according to id, it determines the department of the user and directs the user proper interface.

ReceptionistView



<<Interface>>
ReceptionistView
-logOutButton : JButton
-helpButton : JButton
-roomList : JList
-roomInfoLabel : JLabel
-roomsTab : JTabbedPane
-checkInTab : JTabbedPane
-checkOutTab : JTabbedPane
-roomService : JTabbedPane
-menu : JTabbedPane
-allRoomsButton : JButton
-reservedRoomButton : JButton
-emptyRoomsButton : JButton
-occupiedRoomsButton : JButton
-checkInButton : JButton
-checkOutButton : JButton
-makeReservationButton : JButton

Figure 48: ReceptionistView Interface Class

Attributes:

private logOutButton JButton: This button provides a secure exit to the user.

private helpButton JButton: This button holds the information that helps and guides the user.

private roomList JList: Receptionist can see list of all rooms with JList component.

private roomInfoLabel JLabel: When receptionist select the room from roomList, this label shows the room information.

private rooomsTab JTabbedPane: Receptionist have different authority one of this is display all rooms information and we classify these authorities in tabs.

private checkInTab JTabbedPane: The other tab is checkIn tab which groups information fields in its tab.

private checkOutTab JTabbedPane: This attribute is for reaching checkOut components in different tab.

private roomService JTabbedPane: This attribute is for reaching room service information that changes from kitchen chef, bartender and waiter.

private menu JTabbedPane: This attribute is for displaying hotel meal menu day by day from receptionist view. It is needed when customers want information.

private allRoomsButton JButton: Receptionist have authority to display all kind of rooms.

private reservedRoomButton JButton: Receptionist can display reserved rooms in order to chenckin or cancel the reservation.

private emptyRoomsButton JButton: Receptionist can display empty rooms in order to occupied them according to customer’s request.

private occupiedRoomsButton JButton: Occupied rooms have guest at that moment which is different from reservation room. Receptionist can display occupied rooms and customer information.

private checkInButton JButton: Receptionist can select the room from list and make check in with customer info.

private checkOutButton JButton: Receptionist can select the room from list and make the room free with this button after room bill paid.

private makeReservationButton JButton: Receptionist can select the room from list and make reservation on it after pressing the button.

SecurityView

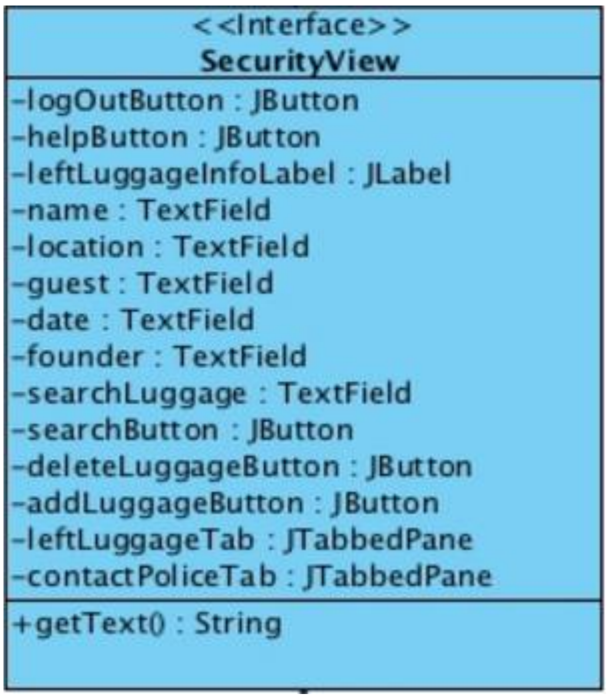


Figure 49: SecurityView Interface Class

Attributes:

private logOutButton JButton: This button provides a secure exit to the user.

private helpButton JButton: This button holds the information that helps and guides the user.

private leftLuggageInfoLabel JLabel: This label provides a space to display left luggage information.

private name TextField: This attribute is used for to hold the name of the left luggage.

private location TextField: This attribute is used for to hold the information about the location where the luggage is left.

private guest TextField: This attribute is used for to hold the name of the customer that lost his luggage.

private date TextField: This attribute is used for to hold the date that left luggage has been found.

private founder TextField: This attribute is used for to hold the name of the person who found the luggage.

private searchLuggage TextField: This attribute is used for to search a left luggage.

private searchButton JButton: This button is used for to search a luggage with given information.

private deleteLuggageButton JButton: This button is used for to delete a luggage that has been taken by its owner.

private addLuggageButton JButton: This button is used for to add a new left luggage item into the left luggage list.

private leftLuggageTab JTabbedPane: This pane is used for to display lost luggage information and operations in a different tab.

private contactPoliceTab JTabbedPane: This pane is used for to display information and operations to contact and inform the police, in a different tab.

Methods:

public getText() String: This method takes left luggage information from text fields to transfer it leftLuggage class

LoginPage

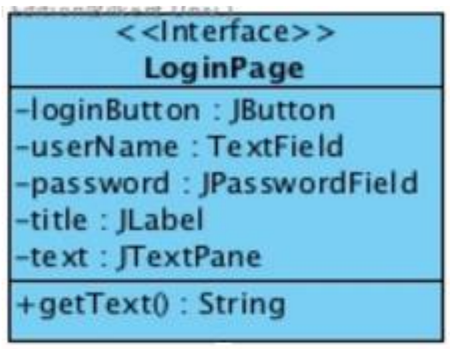


Figure 50: LoginPage Interface Class

Attributes:

private loginButton JButton: At the login page, after entering the username and password, user can enter the system with pressing the loginButton.

private userName TextField: In order to enter system, user need to type his userName which is given when hiring**.**

private password JPasswordField: In order to enter system, user need to type his accurate password. This password is entered wrongly only for 3 times. After 3 wrong trial, system is locked and user have to take his new password from accountant.

private title JLabel: This label includes information about login page.

private text JTextPane: This attribute holds only “user name” and “password” text to inform text field.

Methods:

public getText() String: This method takes userName and password information from text fields to transfer it AccesControl class to check from database.

LogOutView

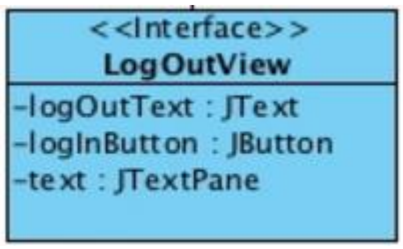


Figure 51: LogOutView Interface Class

Attributes:

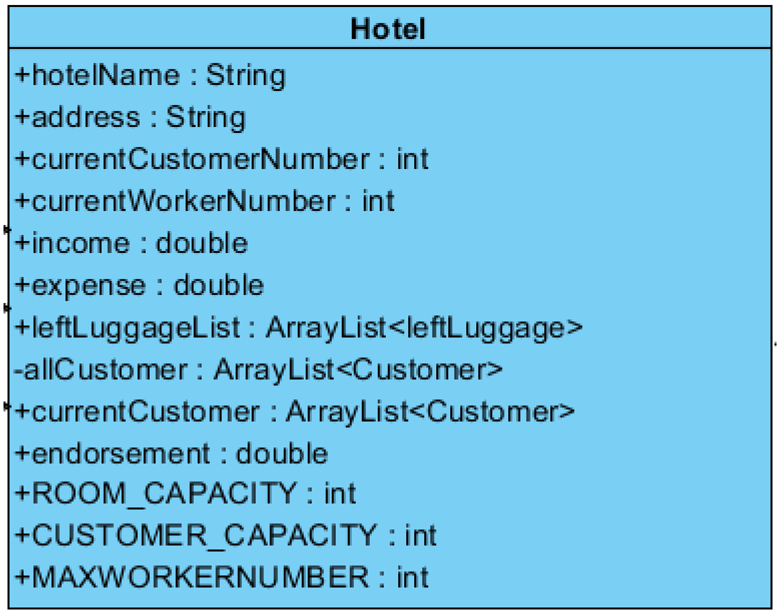
private logOutText JText: This attribute holds the information about logout page.

private logInButton JButton: This button is needed if user want to login to system again.

private text JTextPane: This attribute holds information which are login logout times and user id about log outed user.

Classes

Hotel Class



Hotel
+hotelName : String
+address : String
+currentCustomerNumber : int
+currentWorkerNumber : int
+income : double
+expense : double
+leftLuggageList : ArrayList<leftLuggage>
-allCustomer : ArrayList<Customer>
+currentCustomer : ArrayList<Customer>
+endorsement : double
+ROOM_CAPACITY : int
+CUSTOMER_CAPACITY : int
+MAXWORKERNUMBER : int

Figure 52: Hotel Class

Attributes:

public hotelName String: This attribute is used for hold the name of the hotel

public roomCapacity int : This attribute is used for hold the number of rooms

public adress String: This attribute is used for hold the address of hotel

public customerCapacity int: This attribute is used for hold the maximum number of customer

public currentCustomerNumber int: This attribute is used for hold the current number of customers

public currentWorkerNumber int: This attribute is used for hold the current number of worker

private income double: This attribute is used for hold the income of the hotel.

private expense double: This attribute is used for hold the expense of the hotel.

public leftLuggageList ArrayList<leftLuggage> : This attribute is used for hold the list of the left luggages at the hotel.

private allCustomer ArrayList<Customer>: This attribute is used for hold the information of customers.

private currentCustomer ArrayList<Customer>: This attribute is used for hold the information of current customers

private endorsement double: This attribute is used for hold the economic status of hotel

public MAXWORKERNUMBER final int: This attribute is used for hold the maximum number of worker

public CUSTOMER_CAPACITY final int: This attribute is used for hold the maximum number of customer.

public ROOM_CAPACITY final int: This attribute is used for hold the maximum number of rooms.

Room Class

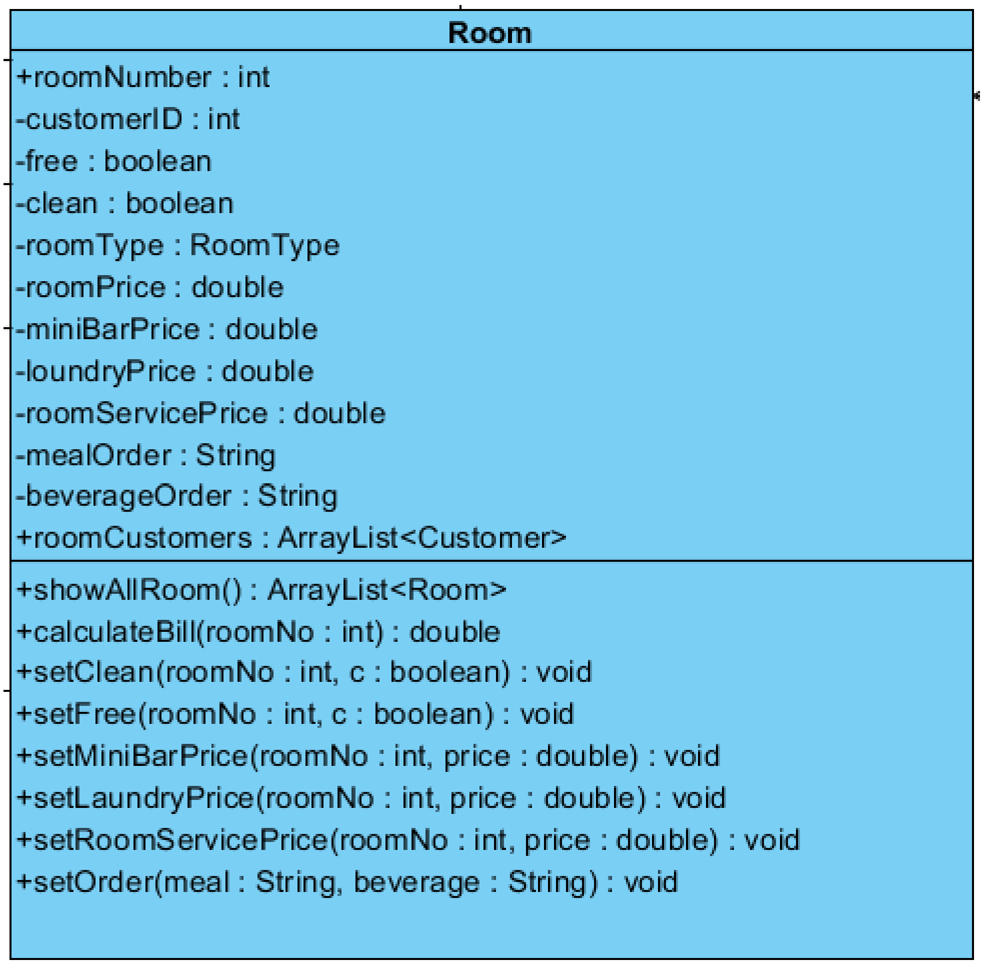

Room
+roomNumber : int
-customerID : int
-free : boolean
-clean : boolean
-roomType : RoomType
-roomPrice : double
-miniBarPrice : double
-loundryPrice : double
-roomServicePrice : double
-mealOrder : String
-beverageOrder : String
+roomCustomers : ArrayList<Customer>
+showAllRoom() : ArrayList<Room>
+calculateBill(roomNo : int) : double
+setClean(roomNo : int, c : boolean) : void
+setFree(roomNo : int, c : boolean) : void
+setMiniBarPrice(roomNo : int, price : double) : void
+setLaundryPrice(roomNo : int, price : double) : void
+setRoomServicePrice(roomNo : int, price : double) : void
+setOrder(meal : String, beverage : String) : void

Figure 53: Room Class

Attributes:

public roomNumber int: This attribute is used for to hold the number of the room

private customerID int: This attribute is used for to hold the customer id

private free boolean: This attribute is used for to see whether the room is empty or not.

private clean boolean: This attribute is used for to see whether the room is clean or not.

private roomType RoomType: This attribute is used for to hold the type of the room.

private roomPrice double: This attribute is used for to hold the price of the room according to its type.

private miniBarPrice double: This attribute is used for to hold the total price of the items used in the mini bar.

private loundryPrice double: This attribute is used for to hold the total price customers loundry if there is any.

private roomServicePrice double: This attribute is used for to hold the total price of the meal and beverage order of the room.

private mealOrder string: This attribute is used for to hold the meal ordered by the customer stays in the room.

private beverageOrder string: This attribute is used for to hold the beverage ordered by the customer in the room.

public roomCustomers ArrayList<Customer>: This attribute is used for to hold the list of customers who stays in the room.

Constructor:

public Room(): It initialize the room attributes with informations from database.

Methods:

public showAllRoom() : ArrayList<Room> : This method displays all rooms in the hotel for special types of workers to see general situations of the rooms.

public calculateBill(int roomNo) : void: This method calculates the bill price by adding all expenses together.

public setClean(int roomNo, boolean c): void: This method states whether the room is clean or not

public setFree(int roomNo, boolean f): void: This method sets the room free by changes its status from occupied to empty room.

public setMiniBarPrice(int roomNo, double price): void: This method sets the total price of the items in the mini bar.

public setLoundryPrice(int roomNo, double price): void: This method sets the total laundry of the room.

public setRoomService(int roomNo, double price): void: This method calculates the price of the room service ordered by the room.

public setOrder(String meal, String beverage): void: This method sets an order taken from a room for kitchen chef or bartender to prepare it.

Customer Class

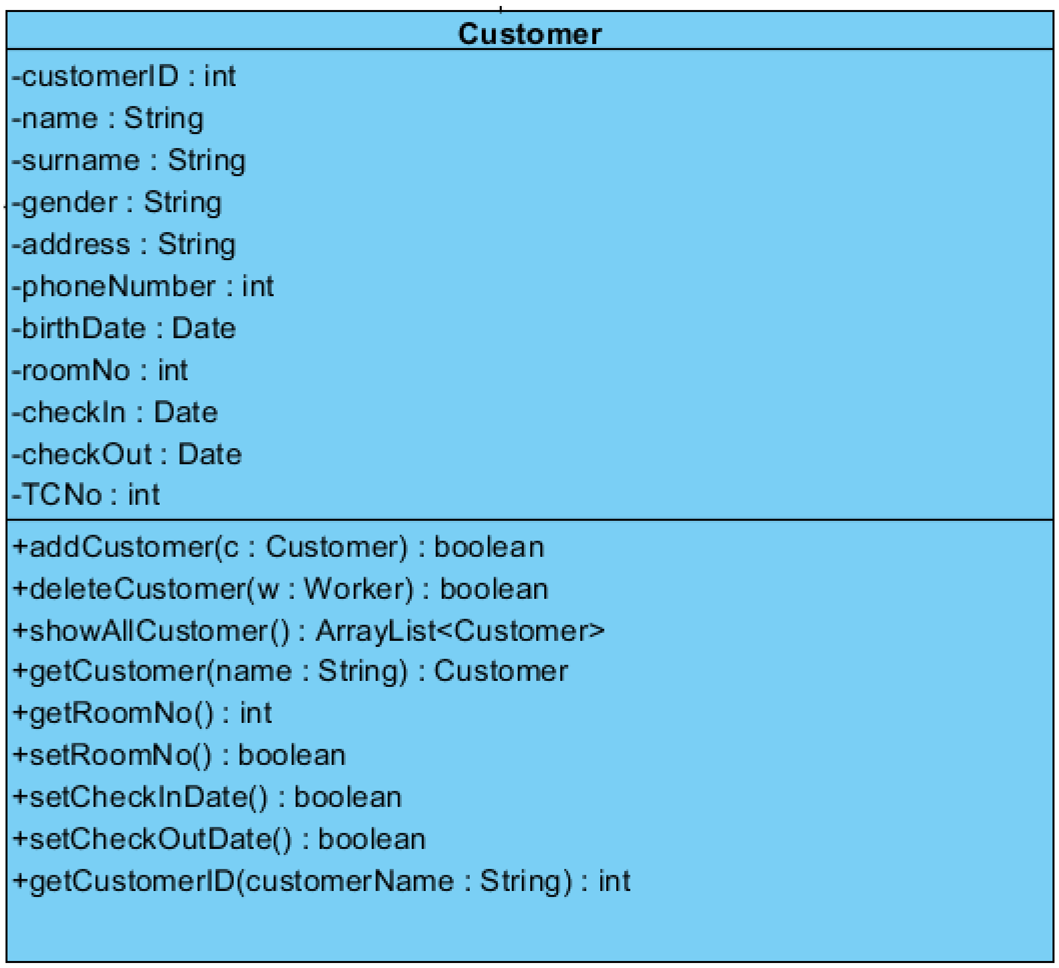

Customer
-customerID : int
-name : String
-surname : String
-gender : String
-address : String
-phoneNumber : int
-birthDate : Date
-roomNo : int
-checkIn : Date
-checkOut : Date
-TCNo : int
+addCustomer(c : Customer) : boolean
+deleteCustomer(w : Worker) : boolean
+showAllCustomer() : ArrayList<Customer>
+getCustomer(name : String) : Customer
+getRoomNo() : int
+setRoomNo() : boolean
+setCheckInDate() : boolean
+setCheckOutDate() : boolean
+getCustomerID(customerName : String) : int

Figure 54: Customer Class

Attributes:

private customerID int: This attribute is used to search customers quickly.

private name String: This attribute is used for to hold the name of the customer.

private surname String: This attribute is used for to hold the surname of the customer.

private gender String: This attribute is used for to hold the gender of the customer.

private address String: This attribute is used for to hold the address of the customer.

private phoneNumber int: This attribute is used for to hold the phone number of the customer.

private birthDate Date: This attribute is used for to hold the birth date of the customer.

private roomNo int: This attribute is used for to hold which room the customer stays in.

private checkIn Date: This attribute is used for to hold the date which the customer checked in.

private checkOut Date: This attribute is used for to hold the date which the customer checked out.

private TCNo int: This attribute is used for to hold TC id number of the customer.

Constructor:

public Customer(): It initialize the room attributes with informations from database.

Methods:

public addCustomer(Customer c): boolean: This method adds a new customer to the customer list that hold the list of the customers who stay in the hotel.

public deleteCustomer(Customer c): boolean: This method deletes the chosen customer from the customer list that hold the list of the customers who stay in the hotel.

public showAllCustomer(): void: This method displays all the customers in the hotel.

public getCustomer(String name): Customer: This method gets the customer according to its name.

public getRoomNo(): int: This method returns the customer's room number.

public setRoomNo(): boolean: This method sets the room information of the customer.

public setCheckInDate(): boolean: This method sets the date which customer checked in.

public setCheckOutDate(): boolean: This method sets the date which customer checked out.

Worker Class

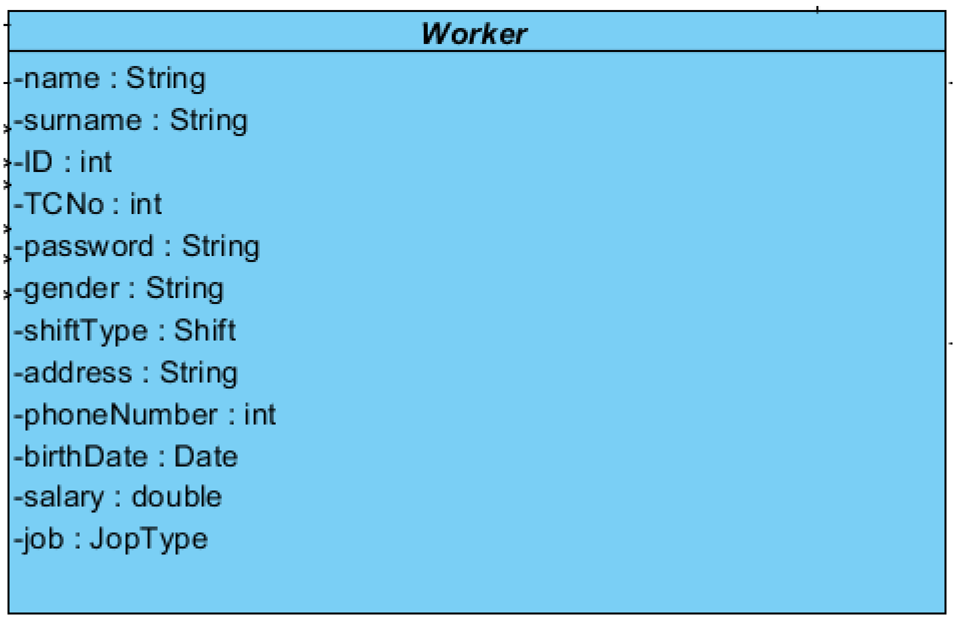


Figure 55: Worker Class

Attributes:

private name String: This attribute is used for hold the name of worker

private surname String: This attribute is used for hold the surname of worker

private ID int: This attribute is unique id number for employee which is given according to job type.

private TCNo int: This attribute is used while storing the workers information.

private password String: This attribute is private password of employees which is used while login to system. The password is given by accountant when employment.

private gender String: This attribute specify the sex of the worker.

private shiftType Shift: This attribute determine the shift type of worker which takes its value from Shift enum class.

private address String: This attribute hold the house address of worker in order to store in database for worker information.

private phoneNumber int: This attribute hold the phone number of worker in order to reach when emergency situation happens.

private birthDate Date: This attribute hold the birthdate of worker in order to calculate her/his age to check bigger than 18 years old.

private salary double: This attribute holds the monthly salary amount that differentiate according to job type and shift type.

private job JobType: This attribute specify the job type of worker that takes kind from JobType enum class.

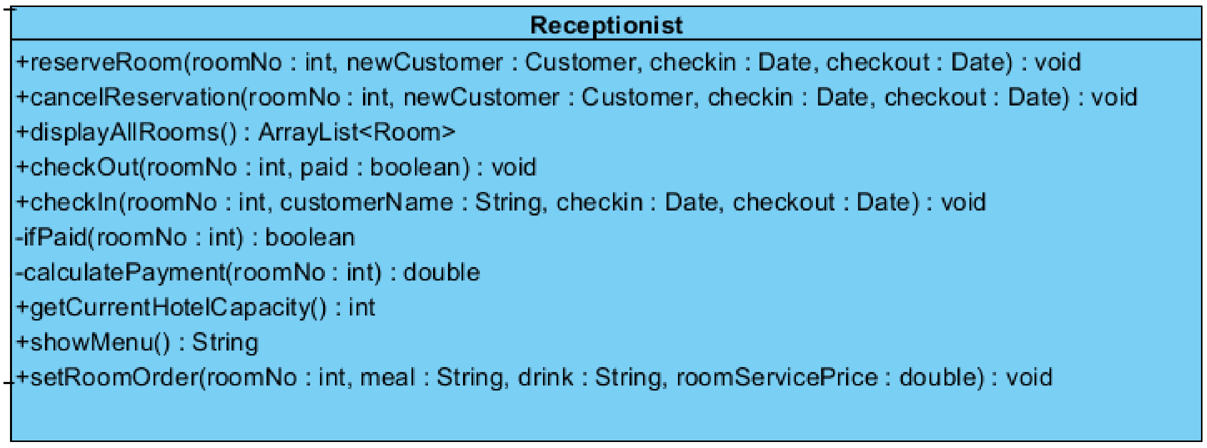
Constructor:

public Worker(): It initialize the worker attributes with informations from database.

Methods:

This class is abstract class which have no common methods between workers but they have same attributes.

Receptionist Class



Receptionist

+reserveRoom(roomNo : int, newCustomer : Customer, checkin : Date, checkout : Date) : void
+cancelReservation(roomNo : int, newCustomer : Customer, checkin : Date, checkout : Date) : void
+displayAllRooms() : ArrayList<Room>
+checkOut(roomNo : int, paid : boolean) : void
+checkIn(roomNo : int, customerName : String, checkin : Date, checkout : Date) : void
-ifPaid(roomNo : int) : boolean
-calculatePayment(roomNo : int) : double
+getCurrentHotelCapacity() : int
+showMenu() : String
+setRoomOrder(roomNo : int, meal : String, drink : String, roomServicePrice : double) : void

Figure 56: Receptionist Class

Attributes:

Receptionist class is type of Worker class therefore it takes worker attributes. It does not have special attributes.

Constructor:

public Receptionist(): It creates employee as Receptionist with attributes from database.

Methods:

public reserveRoom(roomNo: int, newCustomer: Customer, checkIn: Date, checkOut: Date) : void : This function reserve a room for a customer between a specific time interval

public cancelReservation(roomNo: int, newCustomer: Customer, checkIn: Date, checkOut: Date) : void : This function cancel a reservation for a customer’s in a specific time interval

public displayAllRooms() ArrayList<Room> : This function display all rooms with their features such as cleaning condition, occupation of rooms and minibar condition.

public checkOut(roomNo: int, paid: boolean) void : This function controls the user’s payment and if it is true, make the check-out for customers in a spesific room.

public checkIn(roomNo: int, newCustomer: Customer ,checkOut: Date) void : This function make check-in for customers for a specific room and date.

private ifPaid(roomNo: int) boolean : This function checks the payment for a room.

private calculatePayment(roomNo: int) double : This function calculate the payment of customers according to their rooms and return it to receptionist.

public getCurrentHotelCapacity() int : This function returns the number of empty rooms to the receptionist.

public showMenu() String : This function shows the meal and beverage menu to the screen of receptionist.

public setRoomOrder(roomNo: int, meal: String, drink: String, roomServicePrice: double) void : This function creates order for room service.

RoomKeeper Class

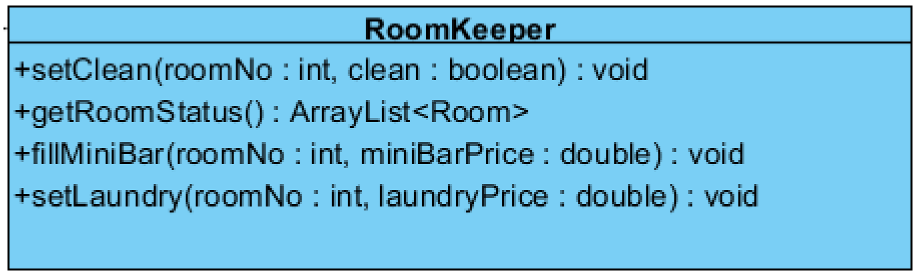

RoomKeeper
+setClean(roomNo : int, clean : boolean) : void
+getRoomStatus() : ArrayList<Room>
+fillMiniBar(roomNo : int, miniBarPrice : double) : void
+setLaundry(roomNo : int, laundryPrice : double) : void

Figure 57: RoomKeeper Class

Attributes:

RoomKeeper class is type of Worker class therefore it takes worker attributes. It does not have special attributes.

Constructor:

public RoomKeeper(): It creates employee as Room Keeper with attributes from database.

Methods:

public setClean(roomNo: int, clean: boolean) void : This function change cleaning situation of a room.

public getRoomStatus() ArrayList<Room> : This function shows the rooms' cleaning, laundry and minibar status.

public fillMiniBar(roomNo: int, miniBarPrice: double) void : This function is called after fill the minibar in order to calculate expense of minibar, and update the customer's' bill.

public setLaundry(roomNo: int, laundryPrice: double) void : This function is called after take laundry from room in order to calculate expense of laundry, and update the customer's' bill.

Manager Class

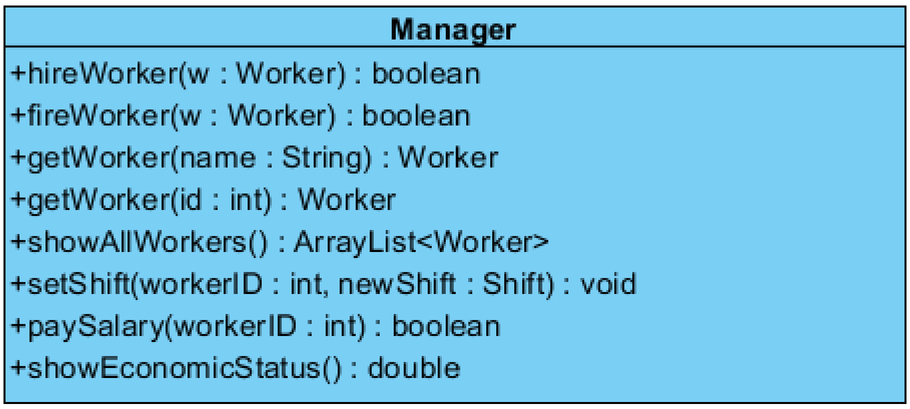

Manager
+hireWorker(w : Worker) : boolean
+fireWorker(w : Worker) : boolean
+getWorker(name : String) : Worker
+getWorker(id : int) : Worker
+showAllWorkers() : ArrayList<Worker>
+setShift(workerID : int, newShift : Shift) : void
+paySalary(workerID : int) : boolean
+showEconomicStatus() : double

Figure 58: Manager Class

Attributes:

Manager class is type of Worker class therefore it takes worker attributes. It does not have special attributes.

Constructor:

public Manager(): It creates employee as Manager with attributes from database.

Methods:

public hireWorker(w :Worker) boolean: This method is used when new worker hired. It stores new worker information into database.

public fireWorker(w :Worker) boolean: This method is used when worker lose his job in hotel. It deletes worker information from database.

public showAllWorkers() ArrayList<Worker> : This method takes all workers with their information from database for internal use.

public getWorker(id : int) Worker : This method is used in order to reach one specific worker object and its information with ‘id’ parameter.

public getWorker(name: String) Worker : This method is used in order to reach one specific worker object and its information with ‘name’ parameter.

public setShift(workerID : int, newShift :Shift) void: Manager have authority to change employees shift type with their id number.

public paySalary(workerID : int) boolean: Manager have authority to give order to the bank via email to pay employee’s salary. This method sends email to the bank with payment information.

public showEconomicStatus() double: Manager have control to on the overall hotel economic status which accountant organize and calculate.

Accountant Class

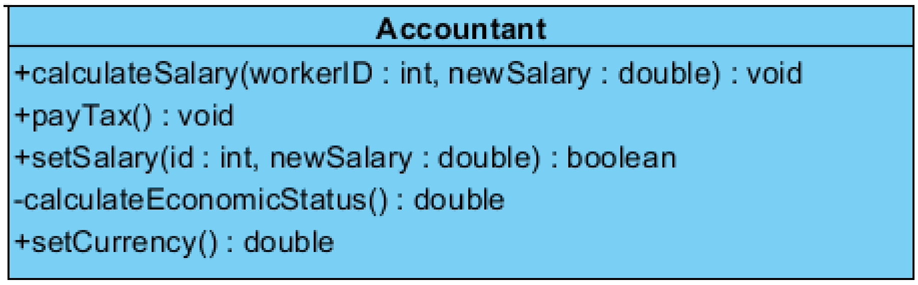

Accountant
+calculateSalary(workerID : int, newSalary : double) : void
+payTax() : void
+setSalary(id : int, newSalary : double) : boolean
-calculateEconomicStatus() : double
+setCurrency() : double

Figure 59: Accountant Class

Attributes:

Accountant class is type of Worker class therefore it takes worker attributes. It does not have special attributes.

Constructor:

public Accountant(): It creates employee as Accountant with attributes from database.

Methods:

public calculateSalary(workerID : int, newSalary : double) void: Accountant have task to calculate and change worker's salary according to their shift and job type.

public payTax() void: Accountant have task which calculate and pay tax with ordering to the bank via email. This method sends email to the bank with payment information.

public setSalary(id : int, newSalary : double) boolean: This method calculate and changes the salary attribute according to worker's job and shift type.

private calculateEconomicStatus() double: This method calls the hotel class's income and expense attributes to calculate the economic status and change hotel's endorsement attribute.

public setCurrency() : double: This method organize currency amount in daily period.

Security Class

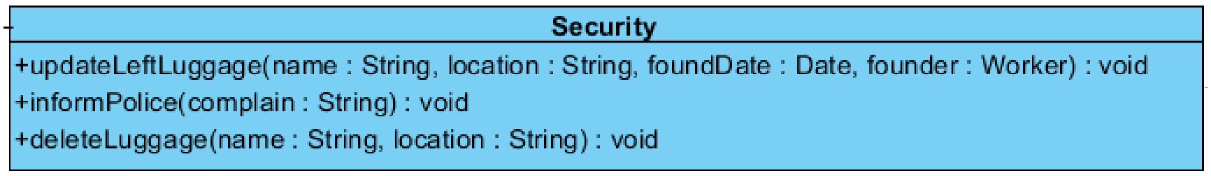

Security
+updateLeftLuggage(name : String, location : String, foundDate : Date, founder : Worker) : void
+informPolice(complain : String) : void
+deleteLuggage(name : String, location : String) : void

Figure 60: Security Class

Attributes:

Security class is type of Worker class therefore it takes worker attributes. It does not have special attributes.

Constructor:

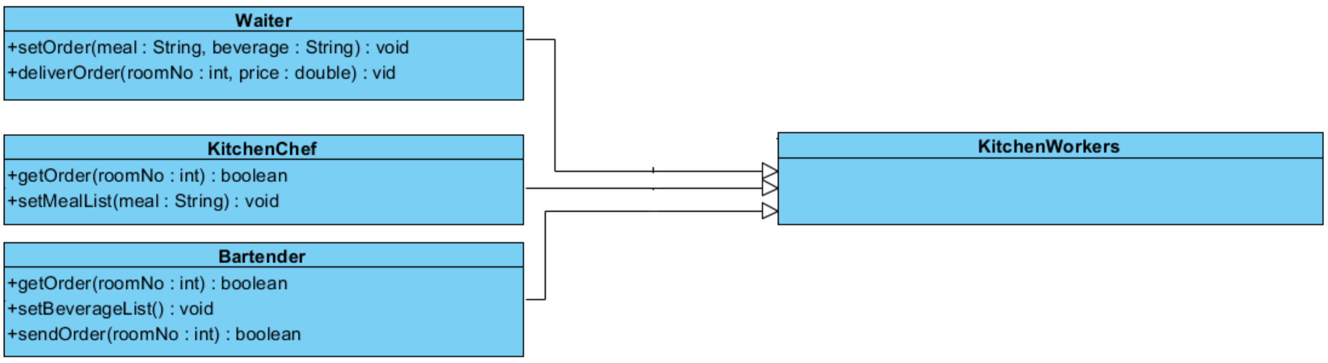
public Security(): It creates employee as Security with attributes from database.

Methods:

public updateLeftLuggage(name : String, location: String, foundDate: Date, founder: Worker) void : This function creates a new left luggage with its attributes and add it to the list of left luggages.

public informPolice(complaint: String) void: This function creates a report of complaint in order to inform local authorities.

public deleteLuggage(name: String, location: String) void : This function deletes a left luggage from the list if it is took by owner guest.


Waiter
+setOrder(meal : String, beverage : String) : void
+deliverOrder(roomNo : int, price : double) : vid
KitchenChef
+getOrder(roomNo : int) : boolean
+setMealList(meal : String) : void
Bartender
+getOrder(roomNo : int) : boolean
+setBeverageList() : void
+sendOrder(roomNo : int) : boolean
KitchenWorkers

KitchenWorkers Class

Figure 61: Worker Class

<u>KitchenChef</u>

Attributes:

KitchenChef class is type of Worker class therefore it takes worker attributes. It does not have special attributes.

Constructor:

public KitchenChef(): It creates employee as KitchenChef with attributes from database.

Methods:

public getOrder(roomNo:int) boolean : This function gets order from guests with the room number of customer and the order.

public setMealList() void : This function sets Meal List according to daily cooked meals.

public sendOrder(roomNo:int) void: This function approve the order has been delivered and it updates the customer's bill.

Waiter

Attributes:

Waiter class is type of Worker class. Therefore it takes worker attributes. It does not have special attributes.

Constructor:

public Waiter(): It creates employee as KitchenChef with attributes from database.

Methods:

public setOrder(String meal, String beverage): void: This method sets order from guests for Kitchen Chef and bartender to prepare it.

public deliverOrder(int roomNo, double price) boolean : This function approve the order's sent and update the customer's bill.

Bartender

Attributes:

Bartender class is type of Worker class therefore it takes worker attributes. It does not have special attributes.

Constructor:

public Bartender(): It creates employee as Bartender with attributes from database.

Methods:

public getOrder(roomNo:int) boolean : This function gets order from guests with the room number of customer and the order.

public setBeverageList() void : This function set Beverage List according to products in the warehouse.

public sendOrder(roomNo:int) boolean : This function approve the order's sent and update the customer's bill.

LeftLuggage Class

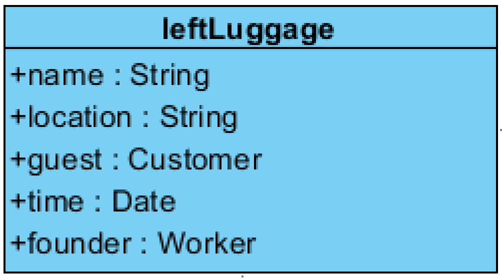


Figure 62: LeftLuggage Class

Attributes:

public String name: This attribute is used for hold the description of item

public String location: This attribute is used for hold the where the item found

public Customer guest: This attribute is used for hold the name of guest who own the item if its known

public Date time: This attribute is used for hold the date of item found

public Worker founder: This attribute is used for hold the name of worker who found the item

Constructor:

public leftLuggage(): It creates left luggage with attributes from input of security worker.

HelpForEmployee Class

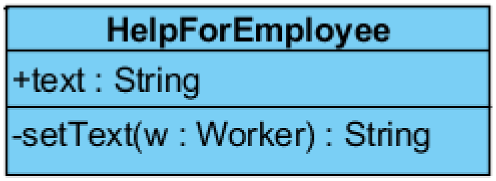


Figure 63: HelpForEmployee Class

BeverageList Class

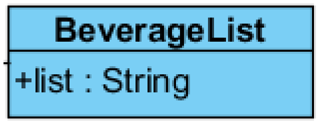


Figure 64: BeverageLiset Class

Attributes:

public list String: This attribute specify the general beverage list of hotel which kitchen workers make changes on it.

MealList Class

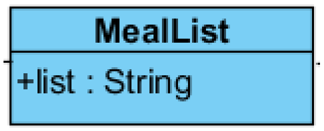


Figure 65: MealList Class

Attributes:

public list String: This attribute specify the general meal list of hotel which kitchen workers make changes on it.

Enum Classes

MiniBar

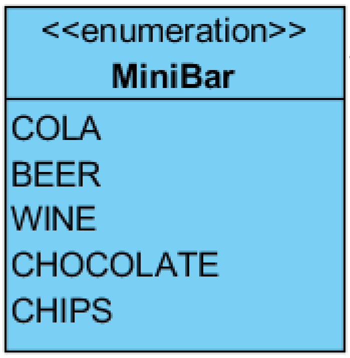


Figure 66: MiniBar Class

- This class is designed to determine miniBar stocks in rooms and prices. During the checkout, when customer report usage of miniBar, receptionist add their price into cost of room. Besides Room Keeper can also add their usage price into system if she/he notice while cleaning.

RoomType

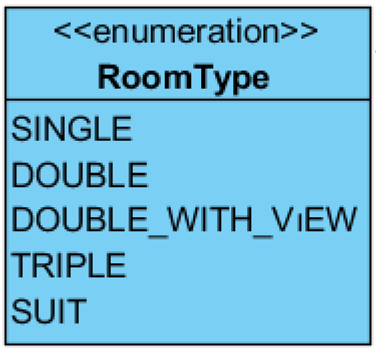


Figure 67: RoomType Class

- This class is designed for choose room type according to customer's preferences while reservation of room. The price of rooms vary depend on room types.

LaundryService

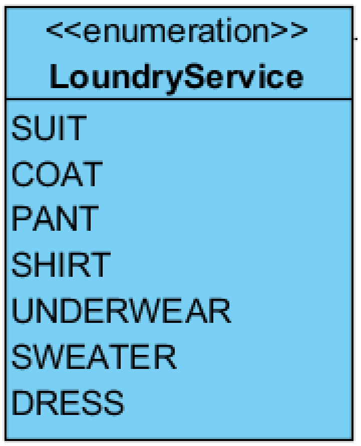


Figure 68: LoundryService Class

- This class is designed for laundry service and fix their price. When customer give his/her clothes to laundry via room keeper, room keeper can add their prices into room's bill with his/her system.

JobType

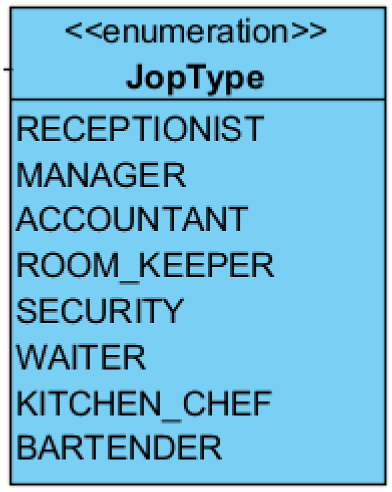


Figure 69: JobType Class

- This class is designed for specifying worker's job type. When customer is created, job type should be matched with this constant values.

Shift

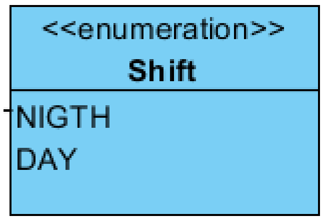


Figure 70: Shift Class

- This class is designed for specifying worker’s shift type. There are only 2 types with 12 hours period.

5.3 Specifying Contracts

Room Class Contracts

In order to reserve a room, the room must be empty.

To make a reservation, the room must be clean.

For the check out and calculate the bill, the room should be occupied.

In order to reserve a room, customer TCNo is mandatory. For example without customer address reservation can be completed but without TCNo it is impossible to reserve a room.

Laundry price, order price, beverage price should be calculated first in order to calculate the bill.

KitchenChef Class Contracts

To prepare the order, KitchenChef have to get order, which is set by receptionist or a waiter, by using getOrder() method.

Manager Class Contracts

In order to hire a new worker which can be done by hireWorker() method, worker's name, surname, TCNo, job type and age have to be specified. Worker's age have to be of legal age. JobType of worker have to match with JobType enum class values.

To hire new worker into hotel with hireWorker() method, Hotel class attribute currentWorkerNumber have to be lower than maximum worker number.

In order to pay a worker's salary, the worker has to be working at least for a month.

The economic status has to be calculated by the accountant first, so that the manager is able to see the economic status of the hotel.

To fire worker, fireWorker() method have pre-condition which is controlling the existence of worker in database

AccesControl Class Contracts

To enter the hotel system, worker can enter his/her password wrong only for 3 times which is FINAL_TRIAL_NUMBER. After 3 wrong password entering, the system lock itself and worker have to be take his/her new password from accountant by manually.

Hotel Class Contracts

Hotel cannot have 0 rooms and it cannot have infinitive number of rooms. It should have exact number of rooms.

In order to create a hotel object, room objects has to be created.

currentCustomerNumber has to be lower number than customerCapacity.

currentWorkerNumber has to be lower number than MAXWORKERNUMBER.

Receptionist Contracts

In order to use the method of reserveRoom() for a customer, Receptionist has to create a customer first because without a customer, the room cannot be reserved.

In order to use the method of cancelReservation() for a customer, there has to be a reservation of this customer.

In order to check-in a customer without parents, the customer has to be older than 18.

In order to check-out a customer, the customer has to be stayed in the room and it has to make his payment before check-out.

In order to check-in, the number of guests should be smaller than the room capacity. Otherwise the guest has to be distributed in separate rooms.

In order to use setRoomOrder() method, the room has to be occupied by a customer.

Meal Menu has to be created by the kitchen chef so that it can bee seen by receptionists.

Room Keeper Class Contracts

Room keeper cannot change the situation of a room from clean to dirty. This function works only one way from dirty to clean.

If minibar is full or there is no laundry in the room, room keeper cannot update the customer’s bill.

For the laundry, the room should be occupied in order to enter the laundry price.

Security Class Contracts

To delete a left luggage from the list, it has to be taken by its owner.

To add a left luggage, finder, location and the date which luggage has been found has to be recorded.

Waiter Class Contracts

In order to update the room service price, order should be delivered by Waiter and it should be entered the system by deliverOrder() method.

Worker Class Contracts

To be able to login the system, each worker should be assigned to a user name and a password given by the manager.

6) Conclusion and Lessons Learned

When we were making our analysis and design report, we used Visual Paradigm to prepare professional models and it was very useful tool to create our diagrams.

During analysis process, we learned how to start to create this project efficiently instead of jumping directly to writing code. We deeply discuss the structure of project, user communities and requirements of them, class diagrams, which storage technique will be useful and how user interface will be efficient. Also we improved our skills while decide the structure of classes and objects, visual representation of them and handle possible logic mistakes without any implementation. As we were creating this report, we focused on requirements of our users and tried to create a program which realistic, user-friendly and implementable. When we create our diagrams, we always care about implementation in order to create a realistic and meaningful analysis report.

After our analysis report, we focused on our design and during this process, we determined our functional requirements and dependencies of our classes. Thanks to this process, we learned different architectural design models and understand how software layers separate between them. We learned how mapping our software into different hardware devices.

After our design report, we focused on our interface classes, detailed explanation of attributes and methods, patterns, and contracts. Throughout this process, we learned many things about different pattern applications, and how classes, attributes interact with each other with methods.

Throughout this project, we learned how to be a team and work together to create something. To create this project, we tried to allocate time and come together as much as possible. We used different platforms like, social media, whatsapp, sms, phone to stay in touch with each other. We saw that the relationship between the members of the group is one of the most important key points that leads a project to success.

With this project and lecture, we mostly learned working with groups in harmony. The reports which increased progressively teached us how to enhance our project and design with taking feedbacks. We started from general project requirements and attributes; and after that detailed projects and diagrams deeply. Most importantly, we learned writing code should start after each requirements and class diagrams implemented deeply in order to decrease mistakes, increase time saving, ensure consistency and professionality.