ICT Engineering

Group 11

Shishir Sharma, 260073

Nikola Petkov Vasilev, 260099

Jeppe Graasbøll Jensen, 261566

Hristo Rumenov Getov, 260064

Course: SEP1

Supervisors:

Mona Wendel Andersen

Allan Henriksen

Date: **23 August 2017**

Abstract

The purpose of our project is to create a hotel booking system for "Overlook Hotel" in three-week project period. We had a detailed interview with the manager and he pointed us the problems faced during day to day activity. After reviewing the interview number of times, we created number of requirements and after prioritizing some of the requirement. We created six use cases. We divided various part of use cases among ourselves and created description of use cases, activity diagram and class diagram. After the class diagram was created we worked on the code and testing of code. We started making the code from small class diagram and the code was also divided among the group members. In our meeting, we discussed about the problems faced during making of the code and, we added some extra functionality in the code if required. When code was ready we made sequential diagram and their link between the diagram. Along with the hotel booking system we also created a user friendly graphical user interface for easy use of the system. All the method of the system was tested and checked by everyone before final additions and documentation.

Contents

Introduction

Stuart Ullman the owner of “Overlook Hotel” was worried about their hotel booking system due to doubling of booking among many other problems. They have been making the booking in a register from the past and due to various problems, he wanted us to make a simple single computer program booking system. He wanted to make us a simple booking system that could make booking, change booking, delete booking, add rooms and many more things. He wanted us to make the program consisting all the requirement in Java but he didn’t want an online booking system. We also have made a simple graphical user interference for easy excess of the program. The following pages contain the detail overview of the project with all the required diagram for proper understanding.

Project analysis

2.1 Functional system requirements

1. Save booking which includes: name of guest, arrival and departure date, room number, extra bed if needed.
2. Store information for check-in: name, home address, phone, date of birth, nationality, room number, arrival date, expected departure for every guest.
3. Register check-out: register the departure date, calculate the total price and give discount if required.
4. Store information about the rooms: type, smoking rules, standard price.
5. See the daily ledger: expected check-in and check-out.
6. See the daily ledger: currently checked-in and checked-out.
7. System must prevent double booking.
8. Check availability of the rooms based on their type, smoking allowed, price and dates for certain period.
9. The system should be able to search for a booking.
10. The system should prevent loss of entered information caused by accident.
11. The system should be able to change the booking: name of guest, arrival and departure date, room number (rooms numbers), extra bed, change price.
12. The system should be able to delete a booking (user should delete the booking if the guest doesn’t show up until 18:00, unless they have informed in advance).
13. The system should have simple user interface which could be used by mouse and keyboard.
14. The system should be operated by the receptionist from the reception desk.
15. One guest can book more than one room.
16. Change information about the rooms: type, smoking rules, standard price.
17. The system should be done in Java.

Upon conducting a careful analysis of the requirements and taking into account the user’s perspective of the program, the following set of use case were accomplished.

2.2 Use Case Modeling

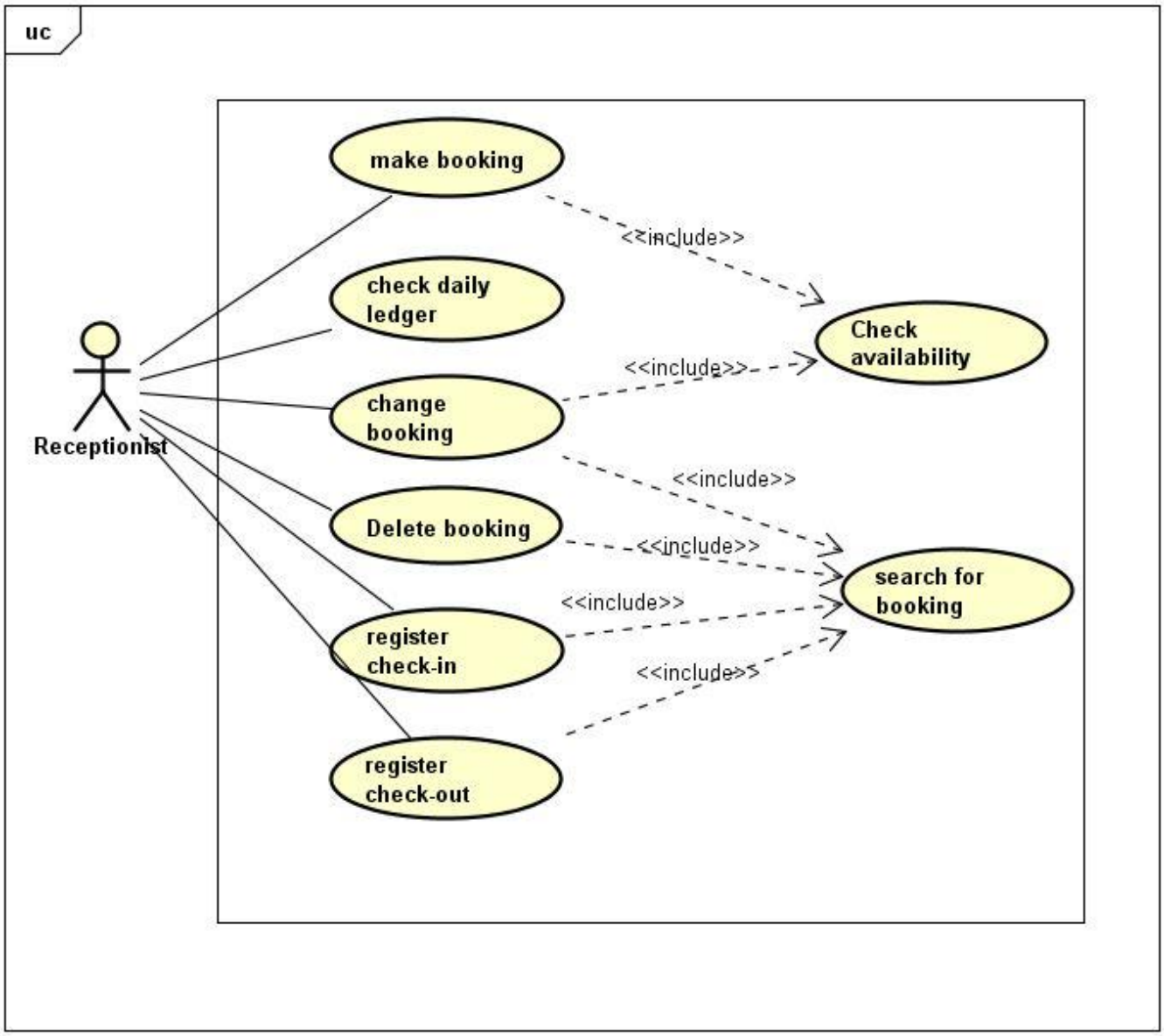


Figure 1: Use case diagram

Figure 1 Illustrates various scenarios the user might come across during the typical workday. The use case diagram demonstrates what the program should be able to do.

2.3 Use Case Description

Use Case	Make Booking
Summary	Staff should be able to make a booking.
Actor	User
Precondition	User wants to make a booking for a room at the hotel
Postcondition	A booking was made. Name of guest, arrival and departure date, room and extra bed is stored inside the booking.
Base Sequence	1. User asks for name of guest and enters the received information in the program. 2. User asks for date of arrival and expected departure date of the guest and enters the received information in the program. 3. System Checks availability (Sub use case). 4. System displays available rooms. 5. User asks which room type guest wants and choose room type from list. 6. System stores information.
Exception Sequence	Cancel: 1.-6. User selects cancel Data not valid: 1-6 System ask user to re-enter valid information.
Sub UseCase	Check availability
Note	

Table 1: Use case description of "Make a booking" use case

The following page contains *Table 1* with description of “Make a booking” use case. It is one of the most common situations user will come across during the working day and one of the most complicated.

2.4 Activity Diagram

The figure on this page illustrates activity diagram for “Make booking” use case.

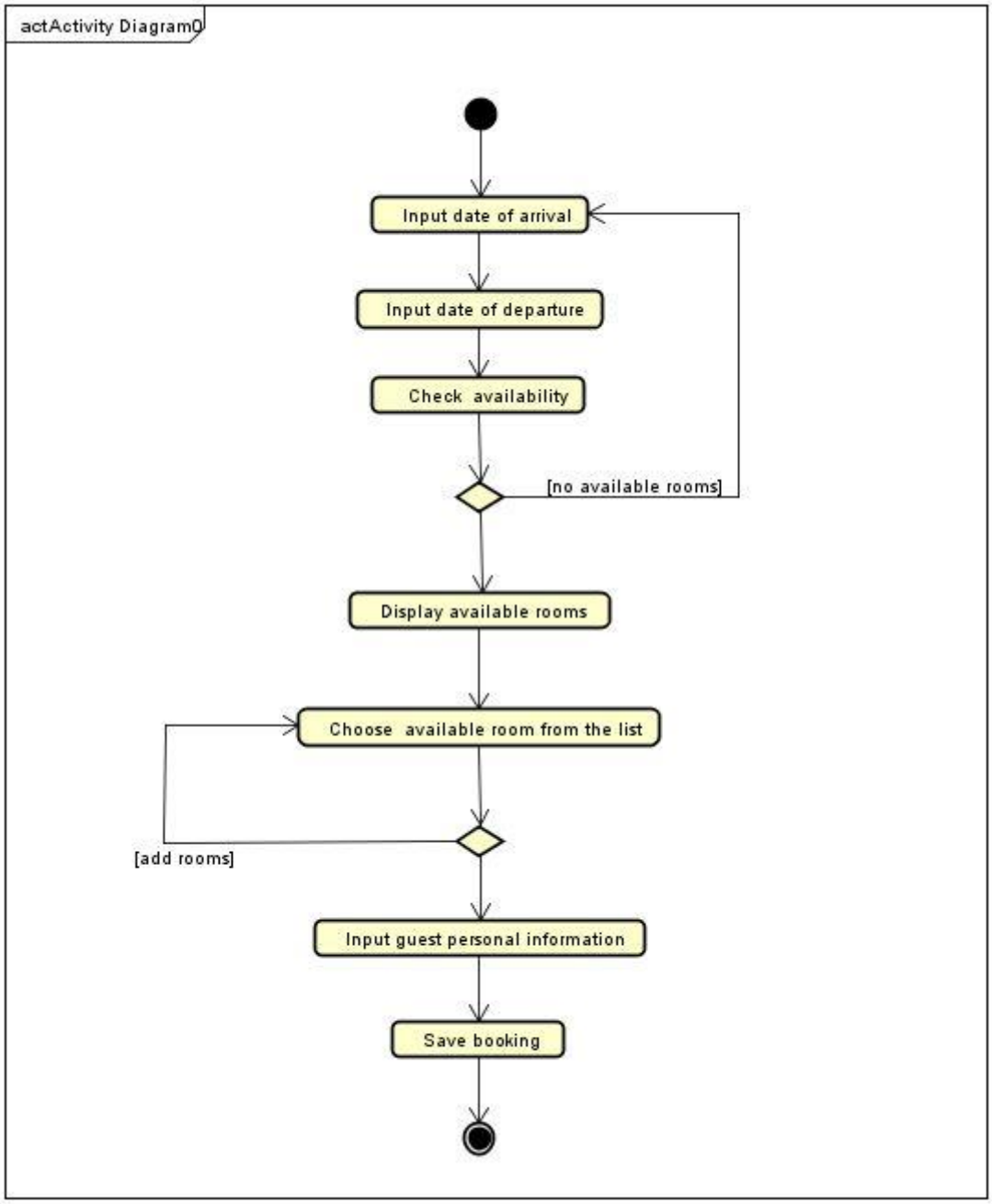


Figure 2: "Make a booking" activity diagram

Figure 2 represents “Search for booking” process. Diagram shows the necessary steps user must follow to create new booking. If hotel doesn’t have available rooms of desired type or number for the given time period, user is required to choose another arrival date. In the end the program saves the booking with all information about it.

For additional diagrams and use case description, address the appendix.

The illustration below depicts classes from the problem domain and few important attributes essential methods.

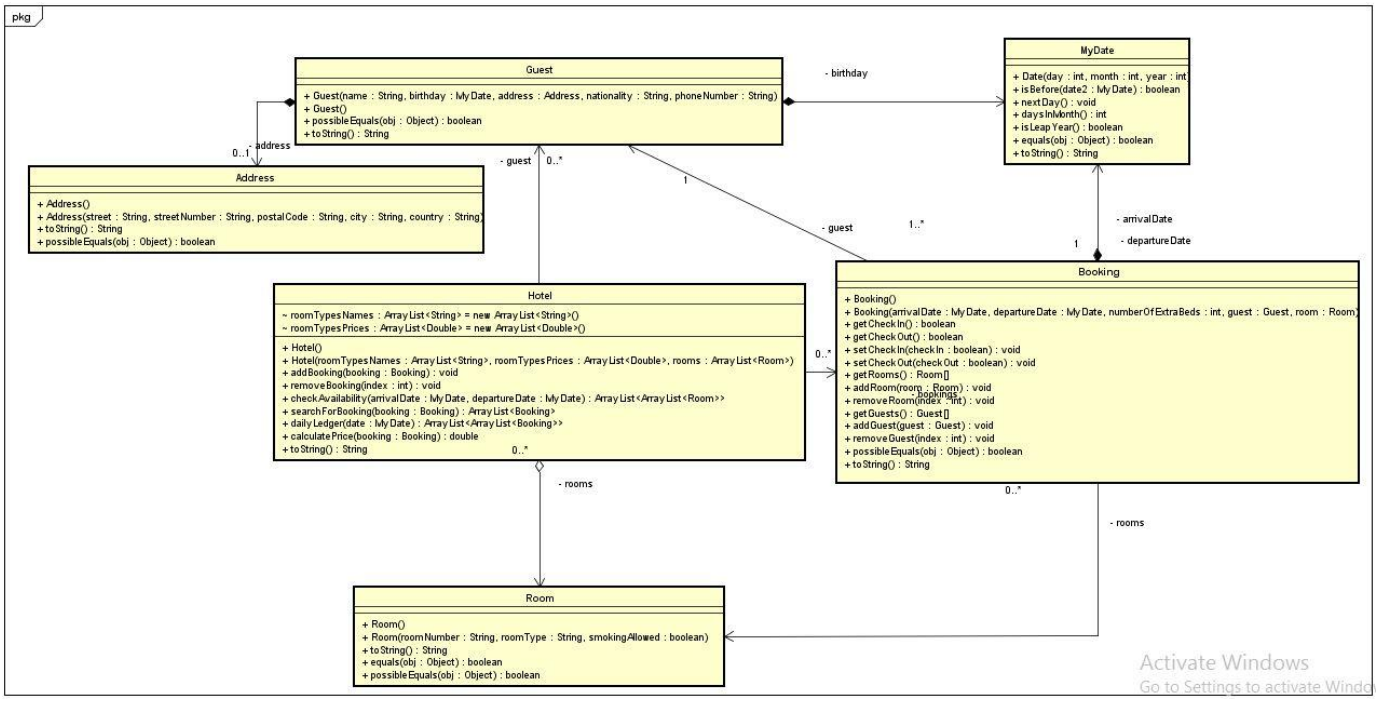


Figure 3: Class diagram

Figure 3 illustrates one of the most important classes and methods used in the system. “Booking” class creates an object of type guest which contain all the information about the guest. “Room” class holds information about the room, such as room number and room type. “Hotel” class holds information about all the rooms, guests and bookings in the hotel. “Hotel” class also checks availability of room. “MyDate” class is in composition both with the “Booking” class and “Guest” class. The “Address” class is in composition with “Guest” class.

Design

3.1 GUI Design

This section of the report portrays a graphical user interface through “Search for Booking”.

Upon starting the program, “Search for Booking” tab of the GUI is displayed as on the figure below:

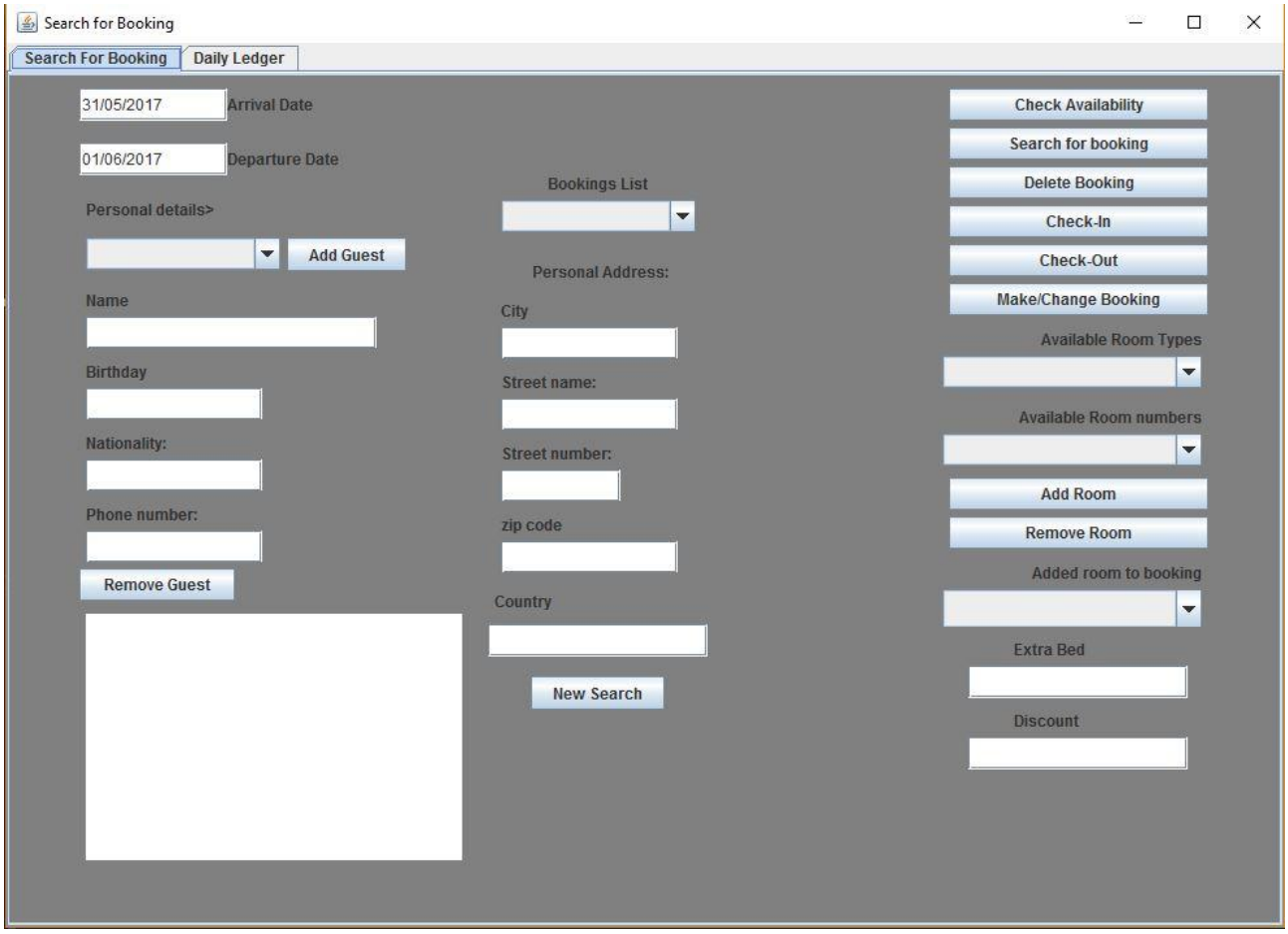


Figure 4: GUI “Search for Booking” tab

User is given an option to choose between two tabs situated on the top of interface. *“Search for Booking”* tab provides the user with a choice to search for booking. User types in the desired information in the corresponding text areas and selects button called “Search for booking”. Drop-down box labeled “Bookings List” provides a list of bookings corresponding to the given information.

If guest has booking, then user can select the booking in the drop-down box and if the guest wants to add guest then the user can click on “Add Guest” button. If the guest wants to delete booking, then the user can click on the “Delete Booking” button. If the user has arrived and is ready for check-in, then user can select on “Check-In” button. And if guest want to check-out then the user can select the “Check-Out” button.

If guest wants to add extra bed and if the user wants to give discount to the guest, then user can discount percentage in text field and “Discount”.

If the guest doesn't have a booking, then the user should click on the "Make/Change Booking" and enter requested detail about the guest. User can input arrival date and check whether there is room available or not by clicking on "Check Availability" button. Panel displays the confirmation message.

User can check bookings, expected check-ins and check-outs for the current day or any day in the "Daily Ledger" menu.

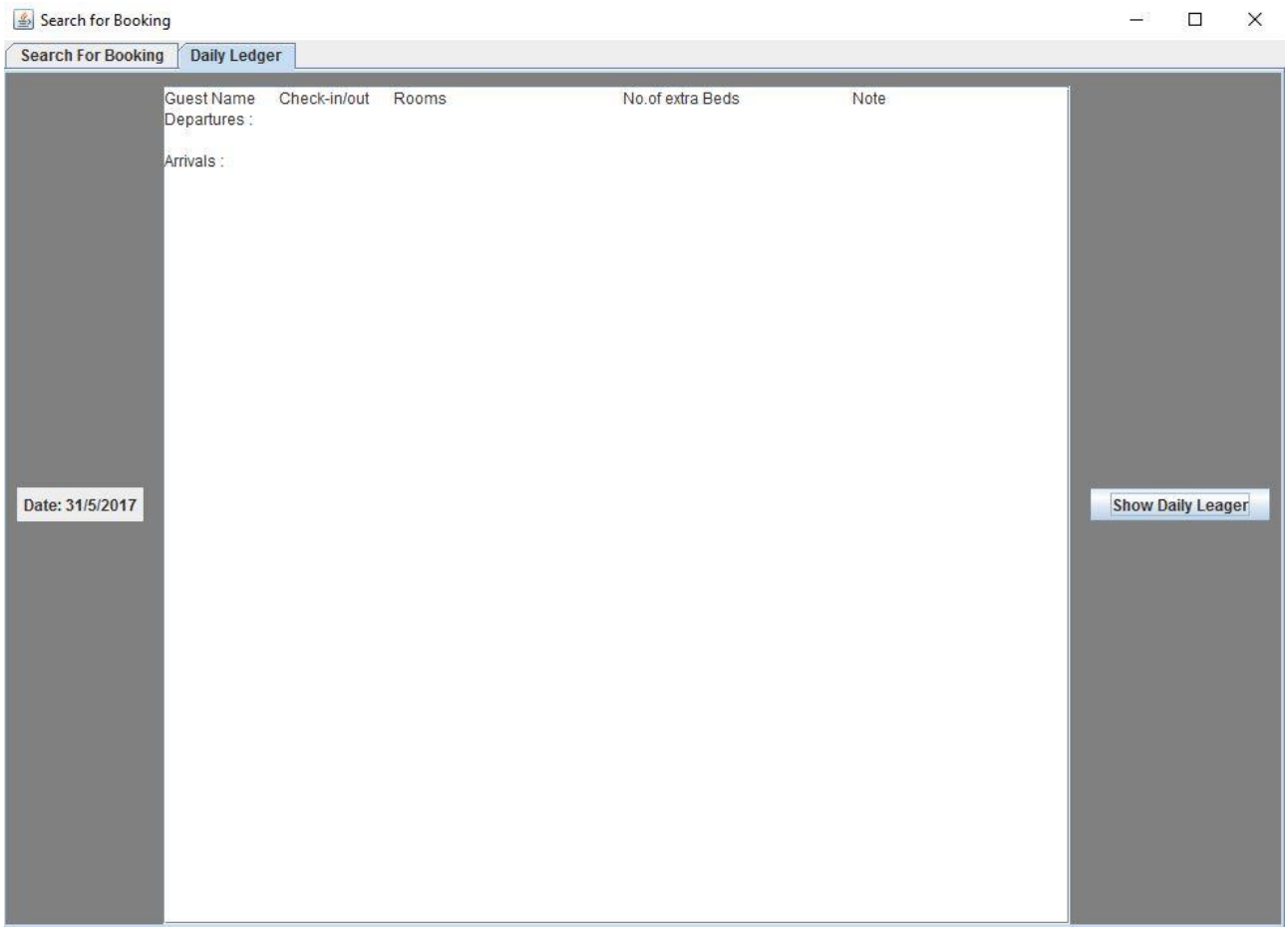


Figure 5: GUI "Daily Ledger" tab

Figure 4 shows the daily Ledger. By clicking on "Show daily Ledger" button panel shows Guest name, whether the guest is checked-in or checked-out, room number, number of extra bed needed for guest and note. It has three sections for departures, arrivals and another for guests staying in the hotel (bookings with specified date in between their arrival and departure date).

3.2 Design Class Diagram

Figure 6 depicts analysis class diagram as basis with some important attributes so as to understand the design of the program.

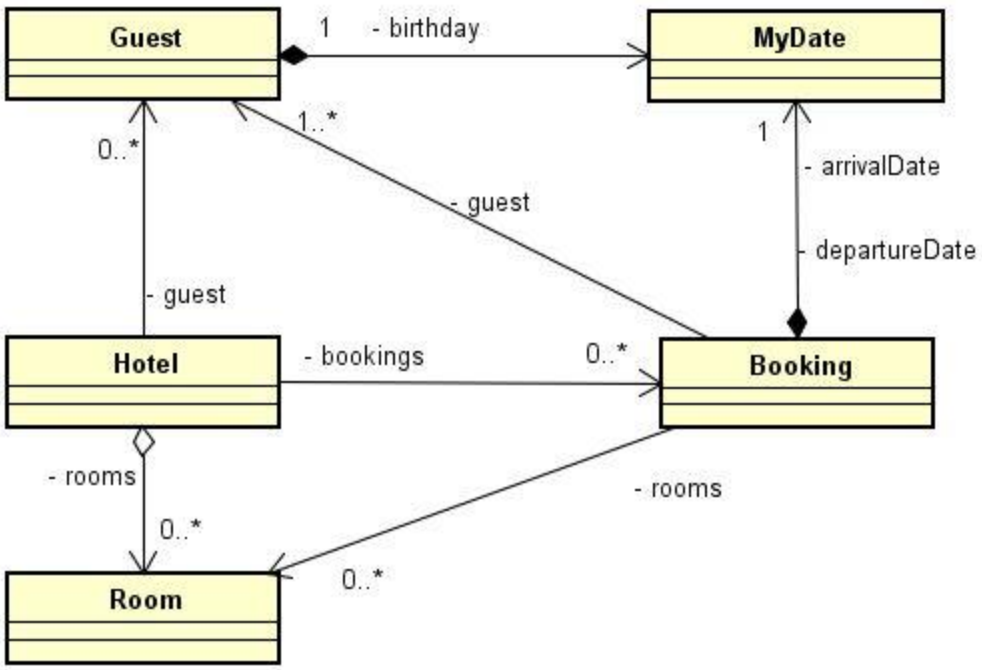


Figure 6: Minimized version of the Class diagram

Diagram shown in the *Figure 6* illustrates the object – oriented approach used in the buildup of the program. Relations between classes are also depicted here.

"Hotel" class contains objects and arrays from "Booking"," Room" and "Guest" classes. "Guest" and "Room" class are in association with "Booking" class. "MyDate" class is in composition with "Guest" and "Booking" class.

The Class Diagram figure on the Appendix illustrate all the classes along with the GUI, it contains all the important attributes and methods for the system classes.

3.3 Sequence Diagram

The figure below shows sequence diagram for “Make Booking” case.

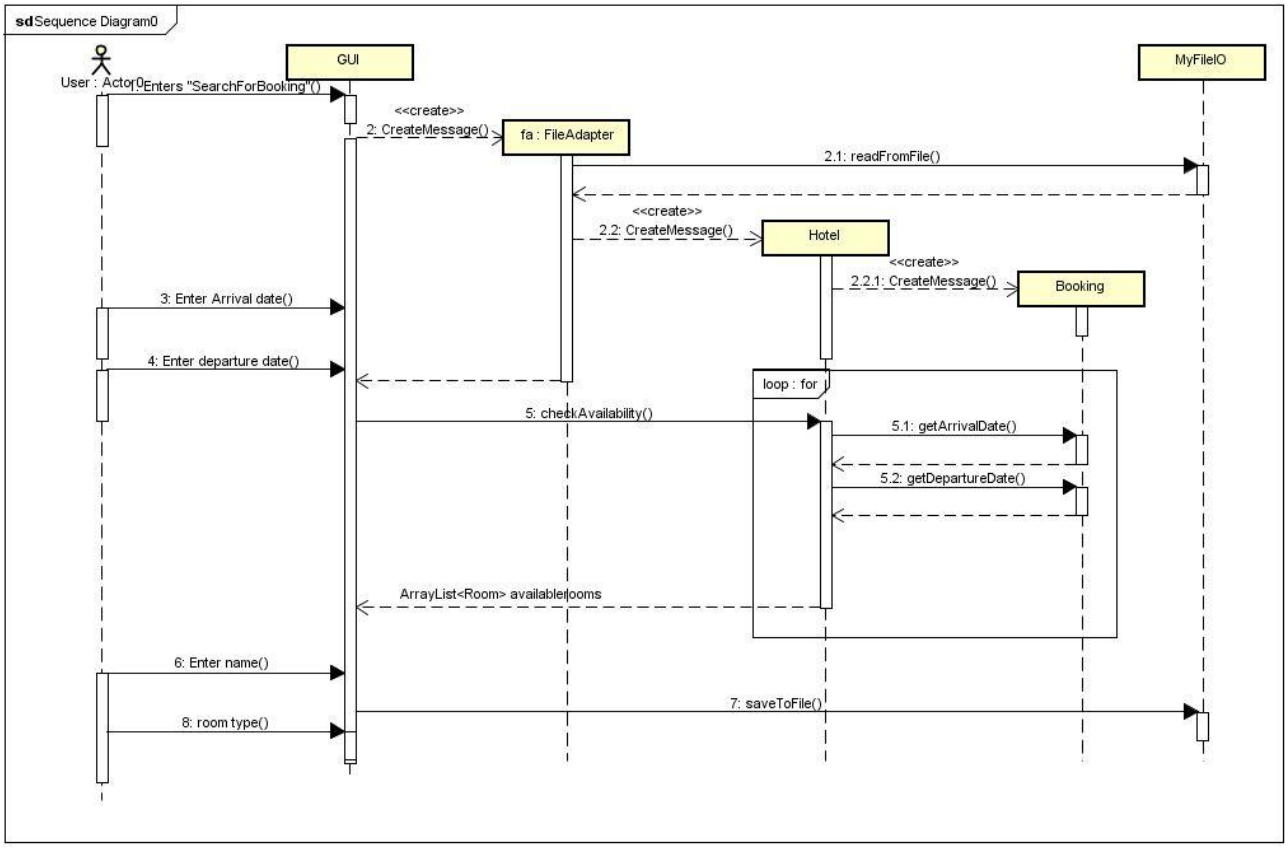


Figure 7: "Make booking" Sequence diagram

User enters “Search For Booking” menu, GUI creates “fa.FileAdapter”, which reads the file from “MyFileIO”. “MyFileIO” returns file to the “FileAdapter”. Using information from the file, “FileAdapter” creates new object of Hotel class. The Hotel object contains an array list with all bookings with name bookings. When user enters arrival and departure date and press button “Check Availability”, GUI uses the method “checkAvailability” to get a list with all available rooms. Then user enters the name of the guest and the room (type and room number) and save it into the file, by pressing “Make/Change Booking”.

Implementation

Hotel class

It is important to import `java.util.ArrayList` and `java.io.Serializable` in order to use ArrayList and be able to store and use data. Hotel class includes empty and full constructor. The full constructor sets the room types, prices, and rooms in the hotel. An empty hotel is generated and saved to file "hotelfile.bin" when running the Test class.

```
public class Hotel implements Serializable
{
   ArrayList<String> roomTypesNames = new ArrayList<String>();
   ArrayList<Double> roomTypesPrices = new ArrayList<Double>();
   ArrayList<Room> rooms = new ArrayList<Room>();
   ArrayList<Booking> bookings = new ArrayList<Booking>();
   ArrayList<Guest> guests = new ArrayList<Guest>();
   public Hotel()
   {
}
   public Hotel(ArrayList<String> roomTypesNames,
ArrayList<Double> roomTypesPrices, ArrayList<Room> rooms)
   {
      this.roomTypesNames=roomTypesNames;
      this.roomTypesPrices=roomTypesPrices;
      this.rooms=rooms;
   }
```

`getBookings()` method in `class Hotel` returns all current bookings in the hotel. It creates a temporary array called `temp` with same size as *`bookings` ArrayList.*

```
public Booking[] getBookings()
   {
      Booking[] temp = new Booking[bookings.size()];
```

```
        bookings.toArray(temp);
        return temp;
    }
```

checkAvailability()

Vital for checking availability of rooms in the hotel is the *boolean* method *isBefore()* situated in the `MyDate` *class.* This method returns true if a date is before another date.

```
public boolean isBefore(MyDate date2)
    {
        if(year<date2.getYear() || year==date2.getYear() &&
month<date2.getMonth() || year==date2.getYear() &&
month==date2.getMonth() && day<date2.getDay())
            return true;
        else
            return false;
    }
```

One of the most important methods for the correct functionality of the whole program is the `checkAvailability()`. This method returns all free rooms for the given time period. The method is defined as `ArrayList<ArrayList<Room>>` because it will return an *ArrayList* of *ArrayLists* of type *Room*.

```
public ArrayList<ArrayList<Room>> checkAvailability(MyDate
arrivalDate, MyDate departureDate)
{}
```

In this method we create a temporary *ArrayList* called `availableRooms` with the same size as the `ArrayList<Room> rooms`, which contain all the rooms in the hotel. *For loop* is used to go through all the bookings. Bookings inside the specified time period are removed from `availableRooms` ArrayList. Part of the code, which is shown below displays the first for loop with nested for loops and if statements inside.

This for loop will go through all booking.

```
        for(int i = 0; i<bookings.size(); i++)
        {
            MyDate bookingArrivalDate =
bookings.get(i).getArrivalDate();
```

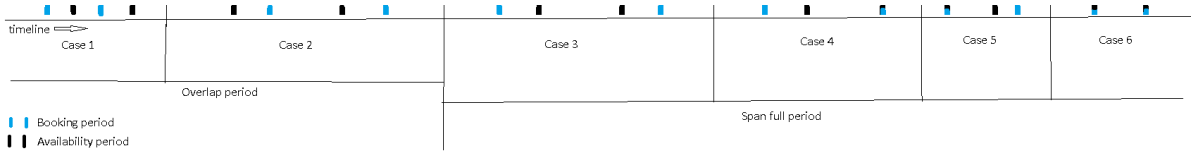
```
            MyDate bookingDepartureDate=
bookings.get(i).getDepartureDate();
```

If the arrival date or the departure date of the booking is inside the availability period (period overlap) the method will remove rooms of the booking from availability list.

```
    if(bookingArrivalDate.isBefore(departureDate) &&
arrivalDate.isBefore(bookingArrivalDate) ||
bookingDepartureDate.isBefore(departureDate) &&
arrivalDate.isBefore(bookingDepartureDate) || (
bookingArrivalDate.isBefore(arrivalDate) ||
bookingArrivalDate.equals(arrivalDate) ) && (
departureDate.isBefore(bookingDepartureDate) ||
departureDate.equals(bookingDepartureDate) ) )

            {
```

Below is shown a graphical representation of the six conditions under which a booking is overlapping the availability period.



If the date period overlaps, rooms will be removed from availability list.

```
            Room[] remove = bookings.get(i).getRooms();
```

This loop cycles through all the rooms in the booking.

```
            for(int k = 0;k<remove.length;k++)
            {
```

This loop compares available rooms and if rooms are equal, it removes them from the availability list.

```
                for(int j = 0; j < availableRooms.size(); j++)
                {.
                   if( remove[k].equals(availableRooms.get(j)) )
                   {
                      availableRooms.remove(j);
                      break;
```

```
} } } } }
```

A temporary two dimensional *ArrayList* called *temp* is used where all available rooms are separated by room type and stored. Inside is an ArrayList for every room type.

```
ArrayList<ArrayList<Room>> temp = new
ArrayList<ArrayList<Room>>();
ArrayList<ArrayList<Room>> temp = new
ArrayList<ArrayList<Room>>();
for(int i = 0; i<roomTypesNames.size(); i++)
{temp.add(new ArrayList<Room>());}
```

If the room type is same as an index in *roomTypesNames*, room is added to same index in *temp*.

```
for(int i = 0; i<availableRooms.size(); i++)
{for(int k = 0; k<roomTypesNames.size(); k++){.
if(availableRooms.get(i).getRoomType().equals(roomTypesNames.get(k
)))
temp.get(k).add(availableRooms.get(i)); }}
```

The method returns two dimensional ArrayLists with rooms divided by room type. Index of this ArrayList equals the index of the room type in *roomTypesNames*.

```
return temp; }
```

possibleEquals()

Finding specific booking is crucial part of our program. To implement this functionality method called `possibleEquals()` of type Boolean is used. This method works somewhat similar to `equals()` method. But it skips comparison of fields not specified in the argument object. Argument object will be created from search fields. Search fields left empty will not limit search results. The idea is that user should be able to search for an object, by specifying any combination of object fields.

Example:

User specifies only birthdate in the search fields. The method will return true for bookings containing a guest with that birthdate. More specified field will limit search further. Example: Both nationality and birthdate are specified. The method will only return true for bookings with a guest meeting both criteria.

```
if(!(birthday==null))
   if(!birthday.equals(temp.getBirthday()))
```

```
            return false;
      if(!(nationality==null || nationality.equals("")))
         if(!nationality.equals(temp.getNationality()))
            return false;
return true;
```

Loop can be used to make a list of objects meeting the search criteria.

possibleEquals can use any value to identify if a field of the argument object is not specified. If that same value is specified in the search field, it will be left out when limiting search results. We therefore use null or “” to specify that the search field is empty.

Primitive variable types cannot be set to null or “”. We made the decision to leave them out from comparison. Any value can be used to denote an empty field. This value must be specified in possibleEquals method. Empty integers search fields “” could be set to -1, or -200150. Some values that will never be specified in a search. Alternatively, an ArrayList of booleans inside all classes could specify if search fields were empty.

Search can be specified by a date. But not a partially defined date.

searchForBooking

Temporary ArrayList of type `Booking` inside the method is created. For loop with nested if statement is used, which will go through all the bookings and return the bookings matching the search criteria. If some search fields are left empty, bookings matching all other field are returned.

```
public ArrayList<Booking> searchForBooking(Booking booking) {
   ArrayList<Booking> temp = new ArrayList<Booking>();
      for(int i = bookings.size()-1; i>=0; i--) {
         if(booking.possibleEquals(bookings.get(i)))
            temp.add(bookings.get(i));
      }
return temp;}
```

Test

Each of the use cases were thoroughly tested by creating test class and testing each method. As seen in the table all the use case scenarios were successfully tested.

Use case	Test
Make Booking	Pass
Change a Booking	Pass
Delete a Booking	Pass
Check Daily Ledger	Pass
Check in a guest	Pass
Check out a guest	Pass

Result

In the test classes for the use case scenarios all the methods were successfully tested with all attributes.

Requirement	Implementation	Test
Make a new booking	Implemented	Works
Save Booking	Implemented	Works
Change a booking	Implemented	Works
Check Daily Ledger	Implemented	Works
Delete a Booking	Implemented	Works
Register check-in	Implemented	Works
Register check-out	Implemented	Works
Choose Room Type	Implemented	Works
Add or delete room	Implemented	Works
Show available rooms	Implemented	Works
Add extras and discount	Implemented	Works
Rent smoking room	Not Implemented	-
Add other guests	Implemented	Works

Conclusion

The program that we developed meets the important requirements. Firstly, we introduced all the challenges we were facing. We sorted them by most importance to least importance ones. We have made all necessary diagrams along with their descriptions and we started to implement the most important challenges. After we complete all main functionalities we started making the graphical user interface (GUI) that helps user to work with system easily.

All diagrams, descriptions, code and GUI were tested and checked by all the group members. This new program helps hotel staff to make bookings easily and prevents double booking.

References

Books

- Tony Gaddis *Starting out with Java: Early Objects.* Addison – Wesley, fifth edition.

Websites

- Allan Henriksen's website: http://www.allanhenriksen.dk/via/
- Stack Overflow topics on Java:
    a. https://stackoverflow.com/questions/5071040/java-convert-integer-to-string
    b. https://stackoverflow.com/questions/3187448/java-substring
    c. https://stackoverflow.com/questions/6431933/how-to-format-strings-in-java
- Studienet: https://studienet.via.dk/Class/IT-SEP1-S17/Session%20Material/Forms/Default.aspx

Supplementing material

Description of the materials supporting our solution are attached in appendices, as following:

Appendix 1: **Codes.**

Appendix 2: **Class Diagrams.**

Appendix 3: **Javadoc.**

Appendix 4: **Sequence Diagrams.**

Appendix 5: **Use Case.**

Appendix 6: **User Guide.**