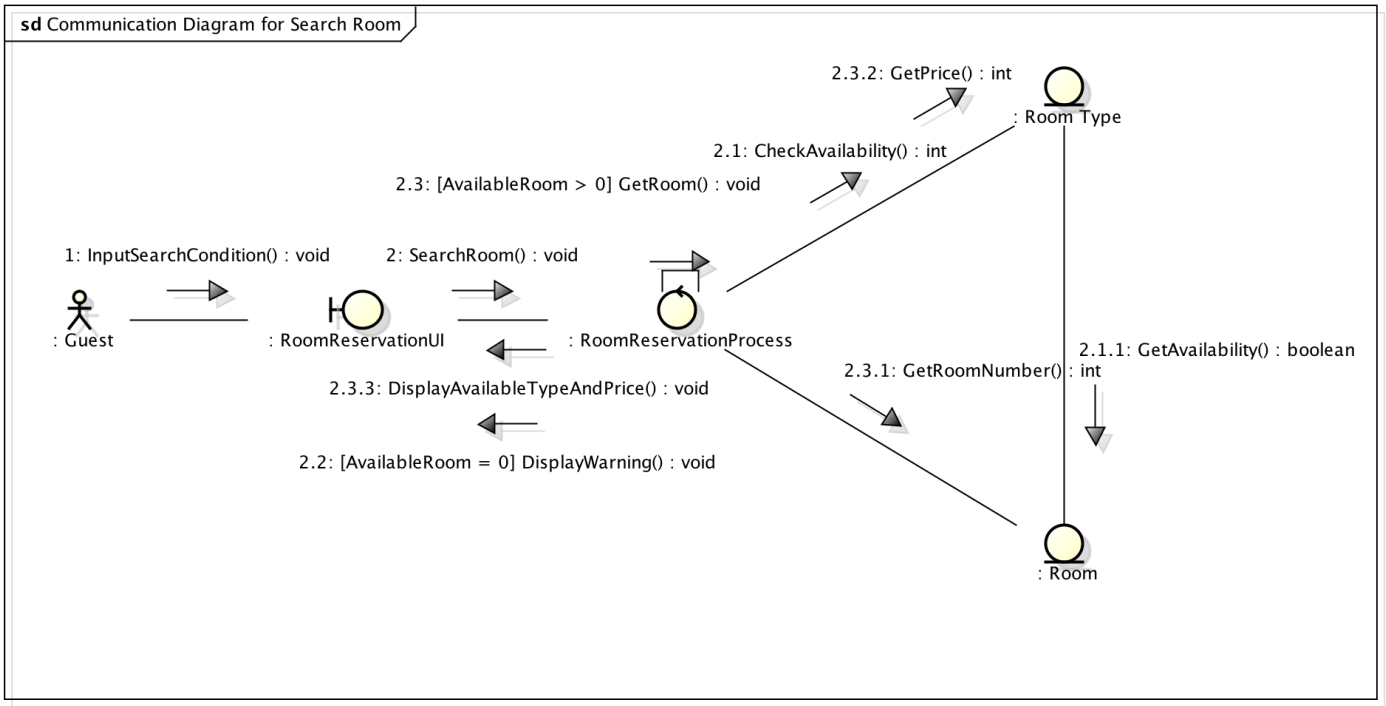
Results of Exercise 3

In exercise 3, System Analysis, 6 diagrams was made from the use case description in exercise 2. We made 1 Class diagram on a system level, and 5 communication diagrams. This is because we have split the process of checking in and out into 3 use cases, Check Reservation, Check In, and Check Out. This resulted in a very simplified diagram for Check In and Check Out, because most of the branching operations were handled by the Check Reservation process.

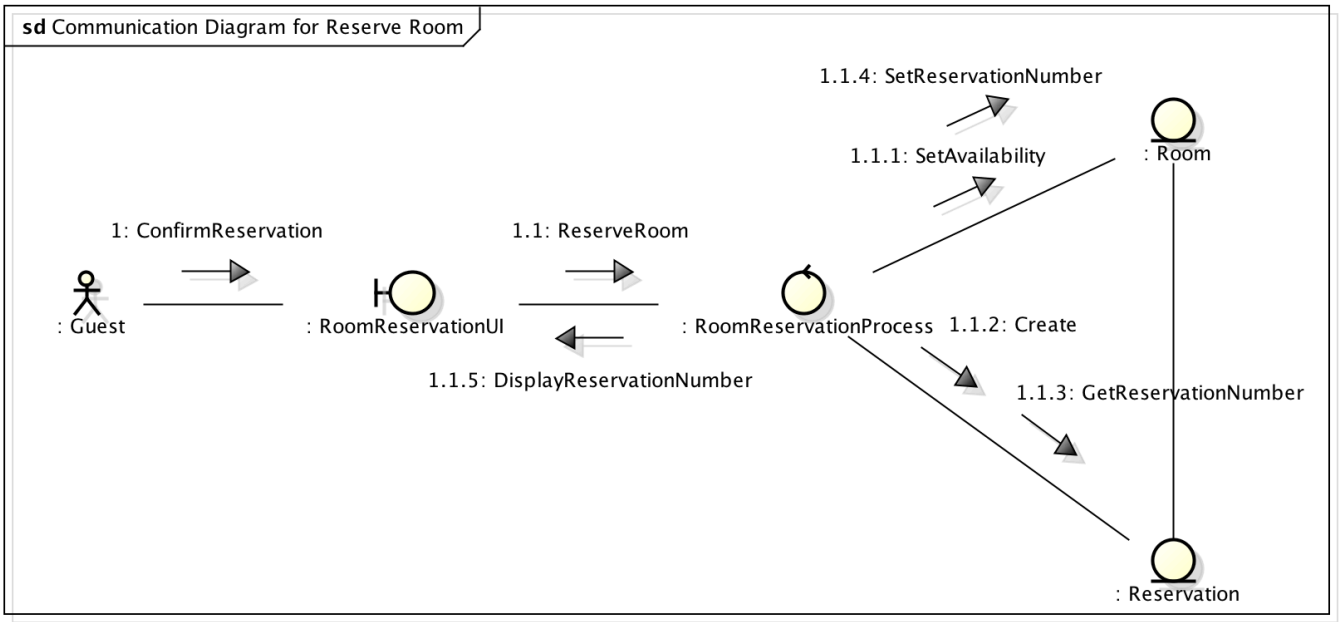


1 Communication Diagram for Search Room

In this process, the system would search for available room in the given time period. The user can also specify the room type they wanted. This process will either results in the system warning that there are no available rooms, or it displays the price per night of the room. In the latter case, the system will also remember which room will be used for reservation, if the Actor follows through with the next process.

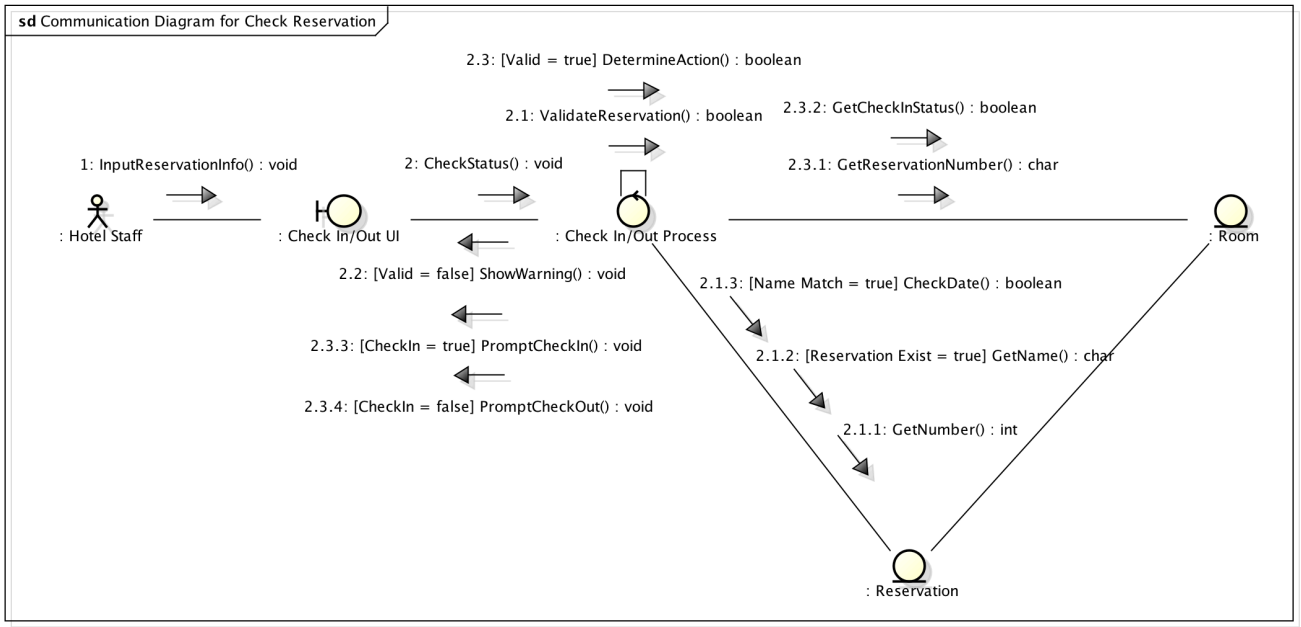
It should be noted that Room type calculate it's Availability by looking through the available period of each room. Thus, step 2.1.1, GetAvailability(), were outlined.

It can also be seen that after the Actor had input the search condition in step 1, there can be two outcomes.



2 Communication Diagram for Reserve Room

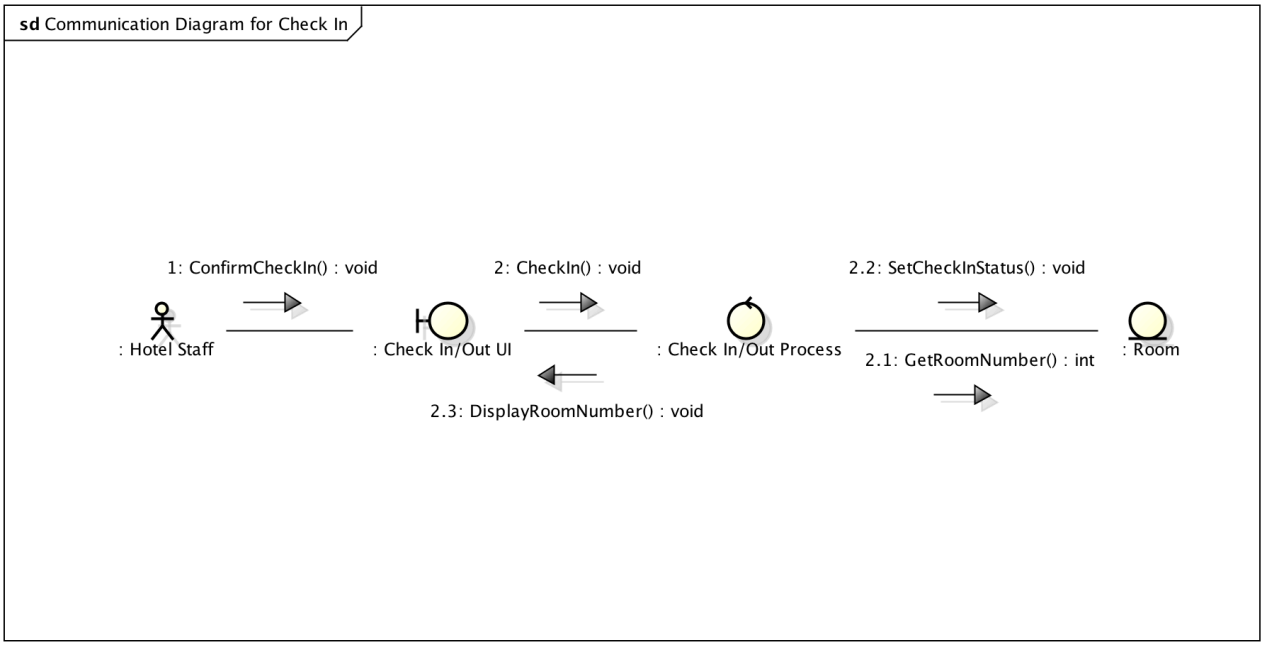
This process continues from the previous process, Search Room, and can only be used when a room has been successfully searched, and the Actor confirm the intention to reserve the room after reviewing the price. This process notify the system to reserve a room for the Actor with a specific reservation number, so no other Actor can reserve the same room at the same time. It also creates “Reservation” object, which will allow the hotel staff to identify the guests and their reserved room. It also provides the Actor with Reservation Number, which is used as a unique identifier (UID) for the reservation created by this process.



3 Communication Diagram for Check Reservation

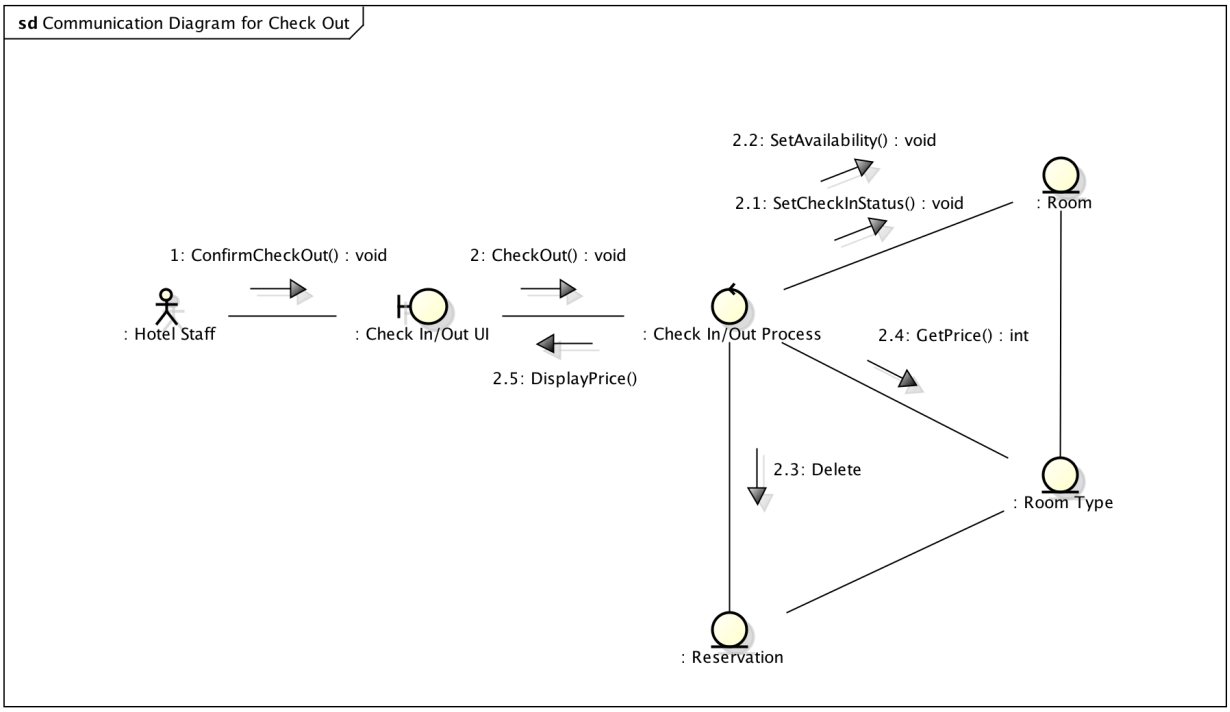
Check Reservation is the most complex process in this project, mainly due to the fact that a reservation must be thoroughly checked if it exists, matches the guest's identity, and is valid for the time period. It also results in another branch condition to determine whether the guest wants to check in or check out by checking the status of the room.

This time around, the Actor is the Hotel Staff, although it is possible to modify this process for the Guest to be the actor instead; incase we need to make a self-service system. The Actor simply has to input the Reservation Number and the Guest's name into the system, and the system will either warns that it is an invalid reservation number (the system will not indicates whether the name doesn't match, the reservation does not exist, nor if the reservation is not valid on that day for security reasons), but if it is valid, the system will search for room that has the same reservation number tag, and get the room's check in status. The system can then determine whether the next process will be check in, or check out.



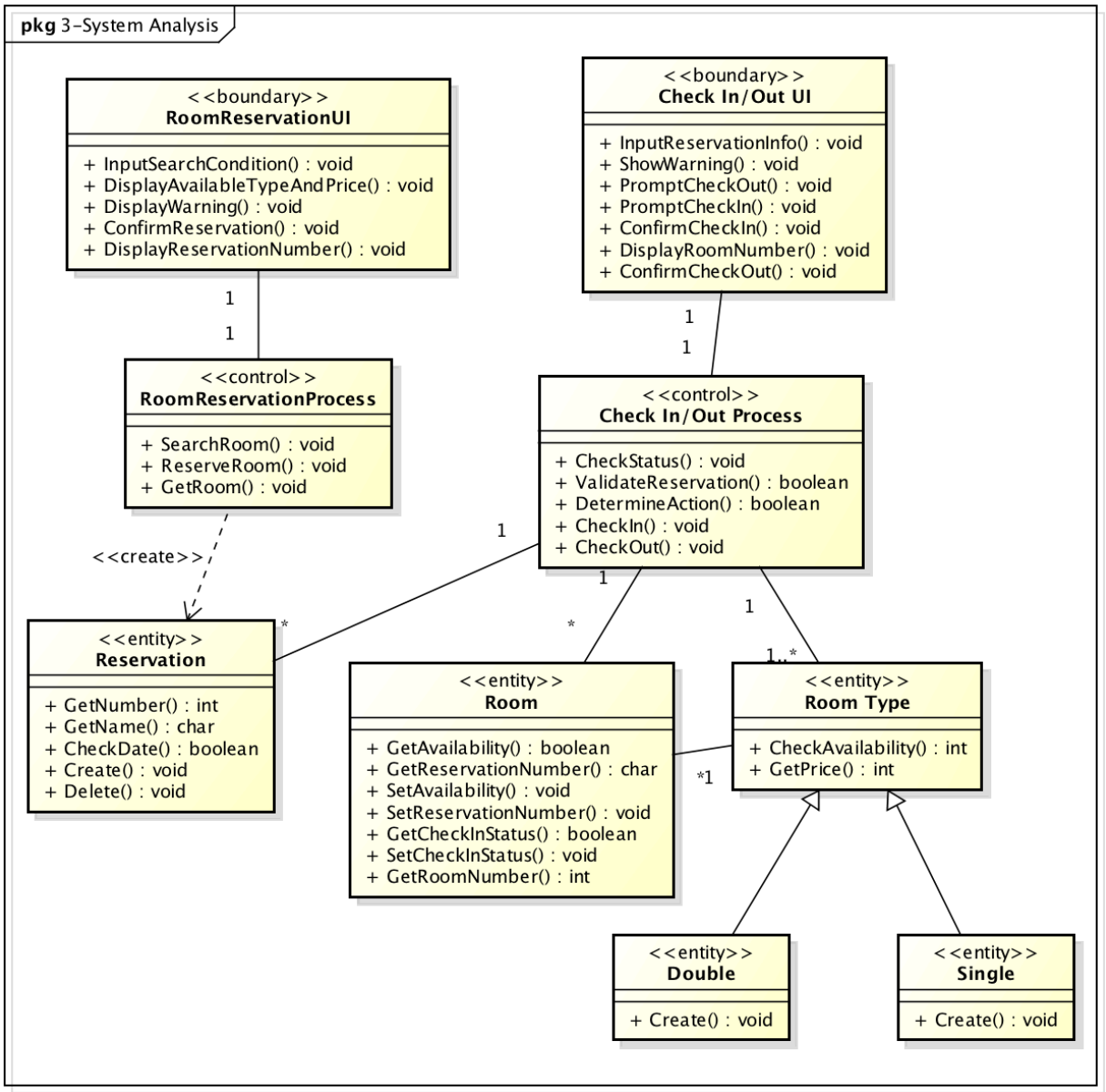
4 Communication Diagram for Check In

If the previous process was successful in identifying the Reservation Number, and determined that the Guest had not checked in yet, this process will start. The Actor will simply have to confirm the check in process, and the system will get the room number for the Actor. The status of the room will also be updated so that the next time the same reservation number is used, the system will be able to determine that the Guest with that reservation number needs to check out.



5 Communication Diagram for Check Out

However, if the Check Reservation process had identified that the reservation number had already been used, the system will determine that the Guest needs to check out. The actor only have to confirm checking out, and the system will delete the Reservation object, update the status of the room to be available (guest checking out early will make the room available in the next day), and display the price to be paid for the guest.

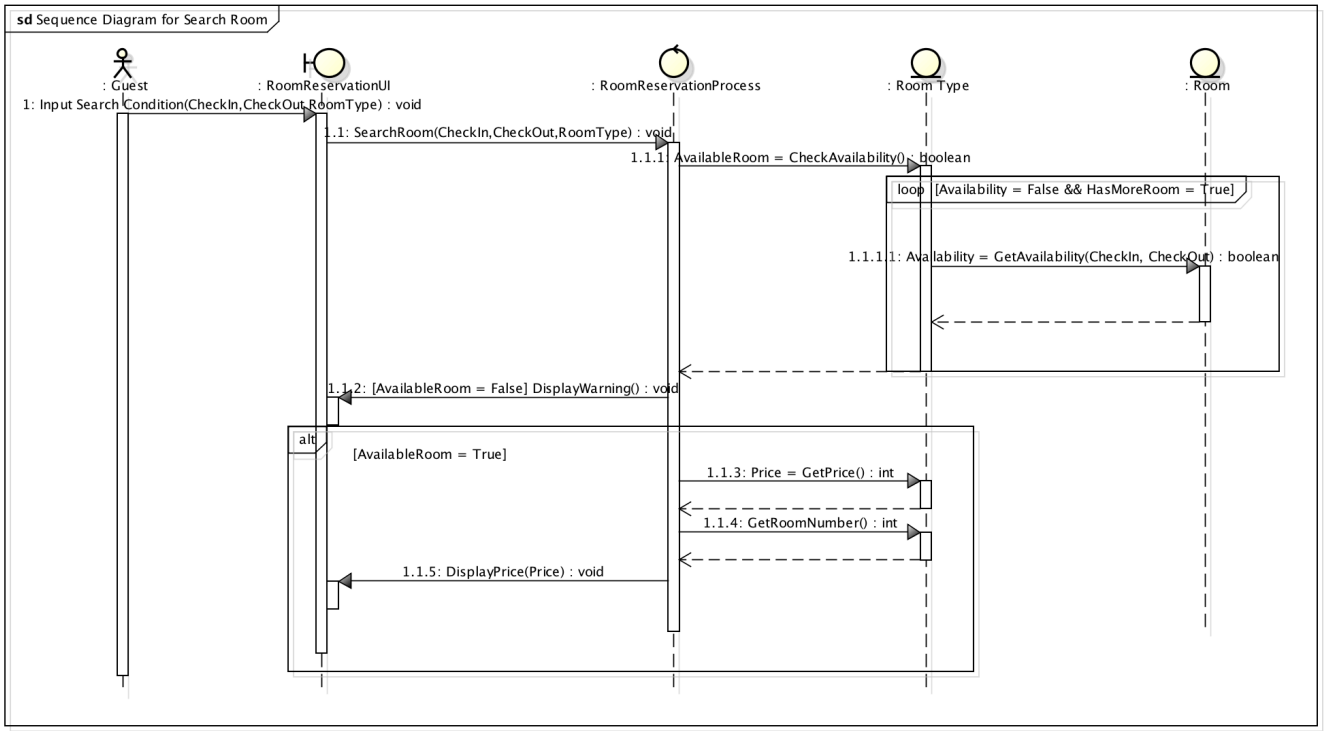


6 Updated Class Diagram (System Level)

After the communication diagram is made, we can conclude that these functions are needed within each class. Although we have added a “Create” function for Double and Single, to indicate that those two classes will require a constructor so that it can be initialized.

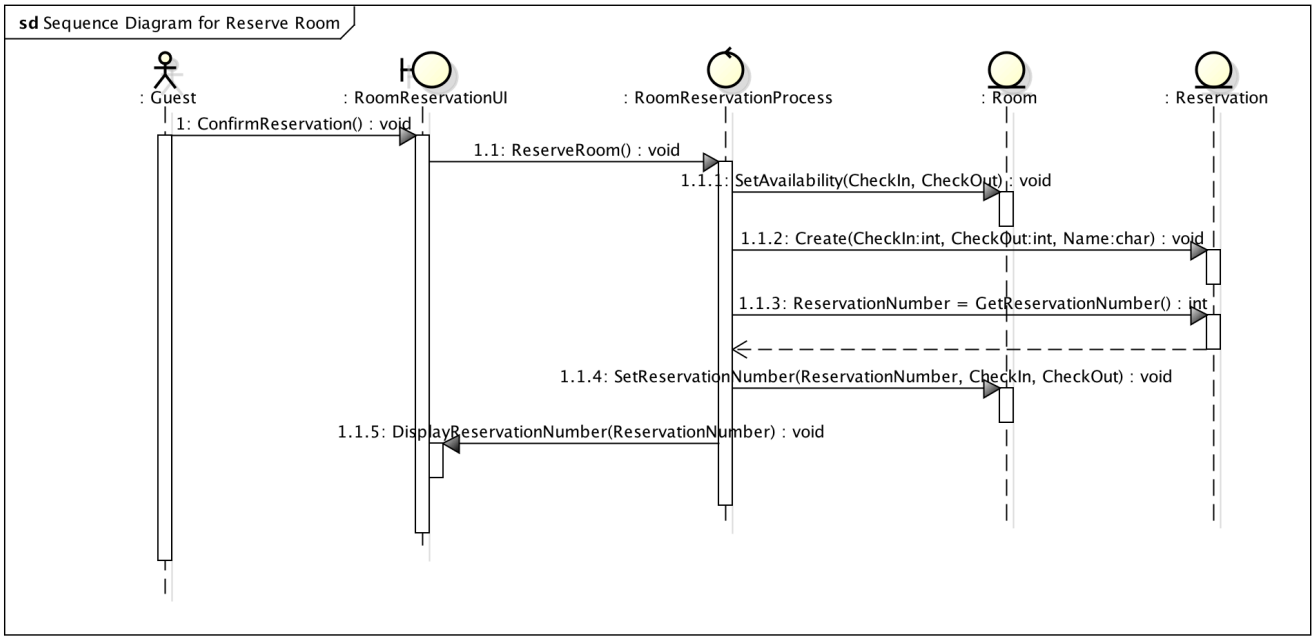
Results of Exercise 4

Based on the Communication diagram made from the previous exercise, we can develop a sequence diagram. In total, 6 diagrams were made; 1 Class diagram on design level, and 5 sequence diagrams. Since these sequence diagram will describe the same process as in exercise 3, only the important feature will be discussed.



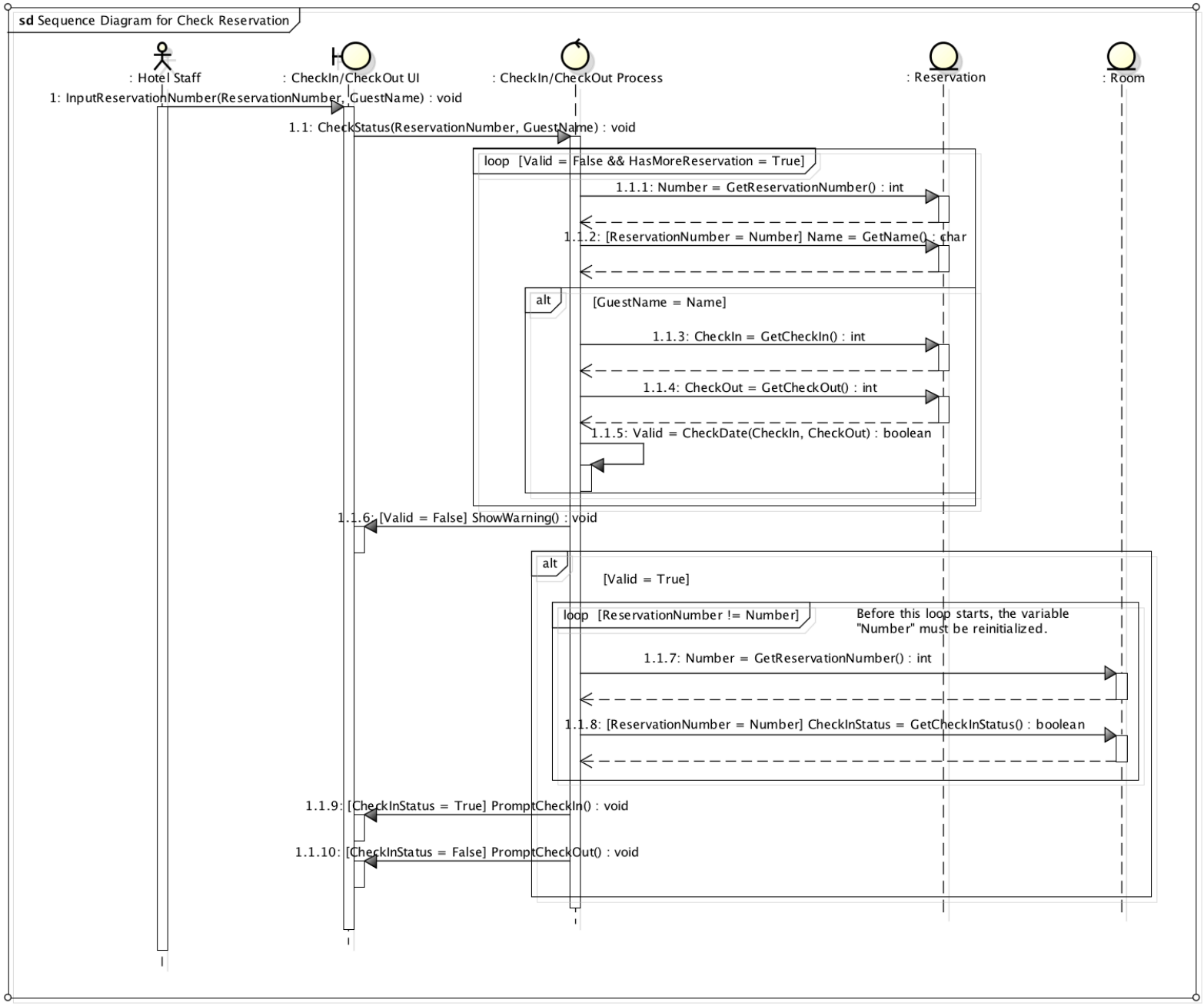
1 Sequence Diagram for Search Room

It can be seen that this process will require a loop in order to search for available room in a given time period. If the loop breaks because there is no more room, the system will warn that there is no available room.



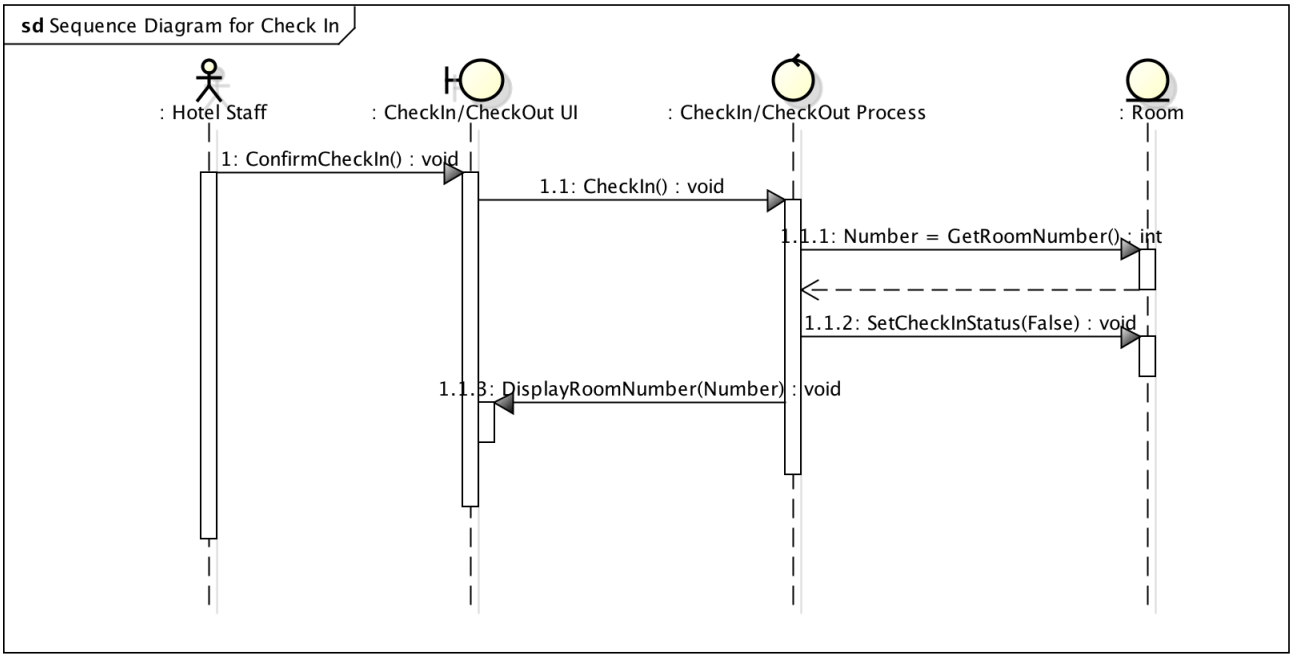
2 Sequence Diagram for Reserve Room.

This is exactly the same as the process lined out in previous exercise. It can be seen that the constructor for reservation will need three inputs, CheckIn date, CheckOut date, and the Guest's name. The number should be procedurally generated for increased security.

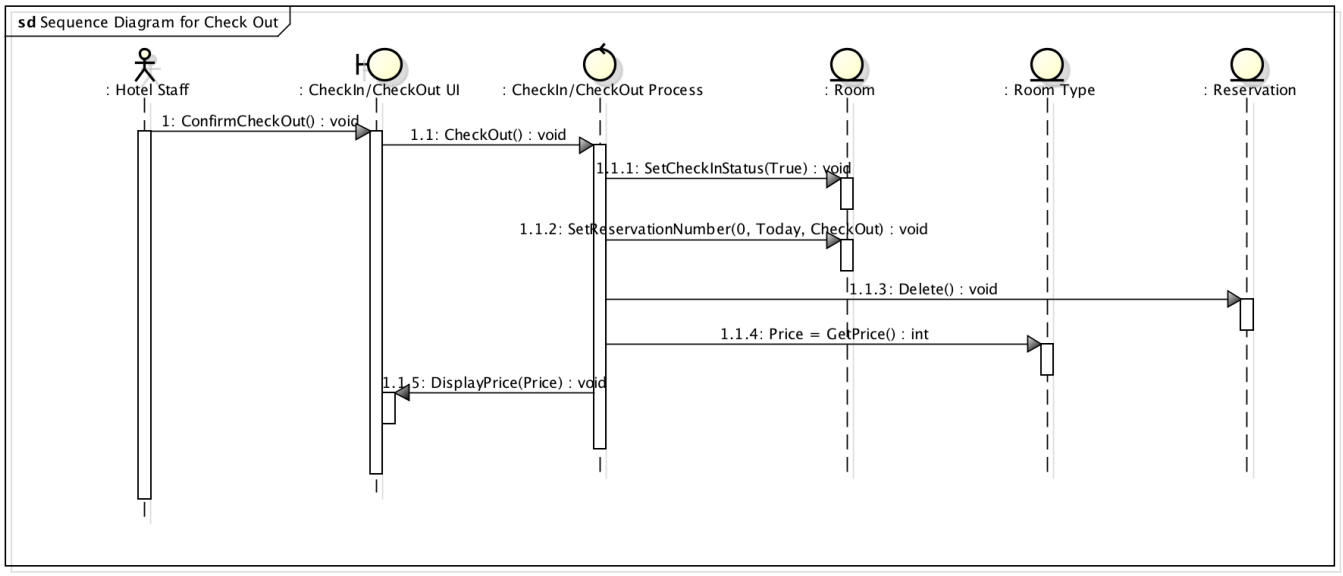

sd Sequence Diagram for Check Reservation
: Hotel Staff
: CheckIn/CheckOut UI
: CheckIn/CheckOut Process
: Reservation
: Room
1: InputReservationNumber(ReservationNumber, GuestName) : void
1.1: CheckStatus(ReservationNumber, GuestName) : void
loop [Valid = False && HasMoreReservation = True]
1.1.1: Number = GetReservationNumber() : int
1.1.2: [ReservationNumber = Number] Name = GetName() : char
alt
[GuestName = Name]
1.1.3: CheckIn = GetCheckIn() : int
1.1.4: CheckOut = GetCheckOut() : int
1.1.5: Valid = CheckDate(CheckIn, CheckOut) : boolean
1.1.6: [Valid = False] ShowWarning() : void
alt
[Valid = True]
loop [ReservationNumber != Number]
Before this loop starts, the variable "Number" must be reinitialized.
1.1.7: Number = GetReservationNumber() : int
1.1.8: [ReservationNumber = Number] CheckInStatus = GetCheckInStatus() : boolean
1.1.9: [CheckInStatus = True] PromptCheckIn() : void
1.1.10: [CheckInStatus = False] PromptCheckOut() : void

3 Sequence Diagram for Check Reservation (Previous page)

Also the most complex Sequence Diagram. It is much more easily understood than the previous communication diagram, however.

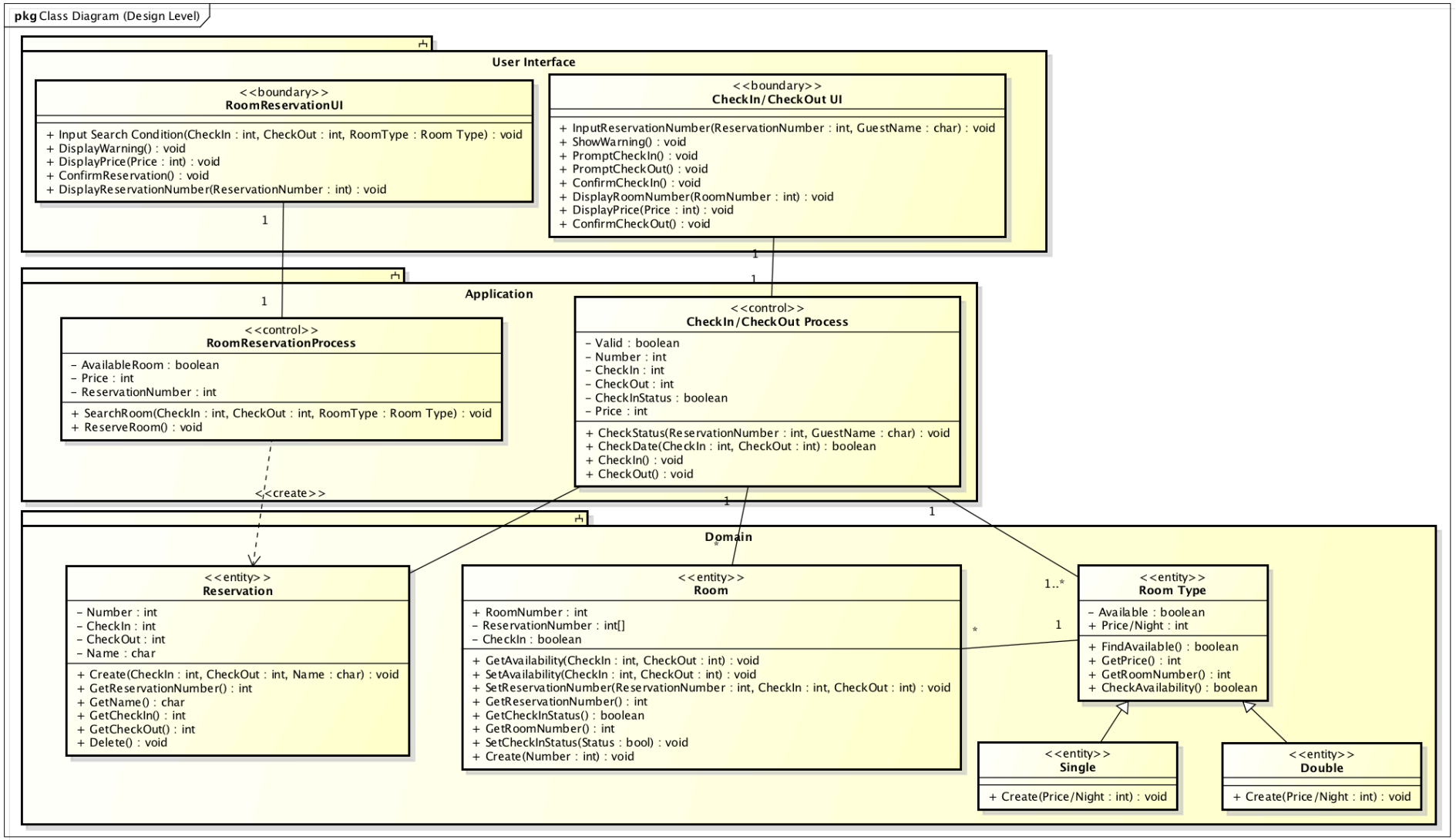


4 Sequence Diagram for Check In.



5 Sequence Diagram for Check Out.

It should be noted that, in step 1.1.2, the reason that the inputted argument is (0, Today, Checkout) is so that the function will clear out the reservation of the room, in case the guest check out early. Unreserved room are marked with the integer “0” in its reservation number attribute.


pkg Class Diagram (Design Level)
User Interface
<<boundary>>
RoomReservationUI
+ Input Search Condition(CheckIn : int, CheckOut : int, RoomType : Room Type) : void
+ DisplayWarning() : void
+ DisplayPrice(Price : int) : void
+ ConfirmReservation() : void
+ DisplayReservationNumber(ReservationNumber : int) : void
<<boundary>>
CheckIn/CheckOut UI
+ InputReservationNumber(ReservationNumber : int, GuestName : char) : void
+ ShowWarning() : void
+ PromptCheckIn() : void
+ PromptCheckOut() : void
+ ConfirmCheckIn() : void
+ DisplayRoomNumber(RoomNumber : int) : void
+ DisplayPrice(Price : int) : void
+ ConfirmCheckOut() : void
1
1
1
1
Application
<<control>>
RoomReservationProcess
- AvailableRoom : boolean
- Price : int
- ReservationNumber : int
+ SearchRoom(CheckIn : int, CheckOut : int, RoomType : Room Type) : void
+ ReserveRoom() : void
<<control>>
CheckIn/CheckOut Process
- Valid : boolean
- Number : int
- CheckIn : int
- CheckOut : int
- CheckInStatus : boolean
- Price : int
+ CheckStatus(ReservationNumber : int, GuestName : char) : void
+ CheckDate(CheckIn : int, CheckOut : int) : boolean
+ CheckIn() : void
+ CheckOut() : void
<<create>>
1
1
Domain
<<entity>>
Reservation
- Number : int
- CheckIn : int
- CheckOut : int
- Name : char
+ Create(CheckIn : int, CheckOut : int, Name : char) : void
+ GetReservationNumber() : int
+ GetName() : char
+ GetCheckIn() : int
+ GetCheckOut() : int
+ Delete() : void
<<entity>>
Room
+ RoomNumber : int
- ReservationNumber : int[]
- CheckIn : boolean
+ GetAvailability(CheckIn : int, CheckOut : int) : void
+ SetAvailability(CheckIn : int, CheckOut : int) : void
+ SetReservationNumber(ReservationNumber : int, CheckIn : int, CheckOut : int) : void
+ GetReservationNumber() : int
+ GetCheckInStatus() : boolean
+ GetRoomNumber() : int
+ SetCheckInStatus(Status : bool) : void
+ Create(Number : int) : void
*
*
1..*
1
<<entity>>
Room Type
- Available : boolean
+ Price/Night : int
+ FindAvailable() : boolean
+ GetPrice() : int
+ GetRoomNumber() : int
+ CheckAvailability() : boolean
<<entity>>
Single
+ Create(Price/Night : int) : void
<<entity>>
Double
+ Create(Price/Night : int) : void

6 Sequence Diagram for Class Diagram (Design Level) (Previous page).

Referencing the sequence diagram, we determined a more detailed function and attributes of each class.