Assignment 2

Hotel Reservation System

Ciara Power – 20072488
Kacper Wołoszyn – 20071494

Software Design and Testing
BSc in Applied Computing

Table of Contents

1. Introduction

We have chosen to base our project on a hotel booking system. The projects discovers various use cases that would be involved in making a system that is a hotel booking system. There are 11 use cases in this system with 37 scenarios.

There are 4 users of the system. a Guest, a Receptionist, a Manager and a Cleaner. The operations explored in this assignment are both that affect the guest such as booking and check in, and operations that the staff of hotel are responsible for such as modify price or clean room.

The responsibilities of the 4 users are as follows:
Manager -

- Modify prices of rooms.
- Log onto the system

Cleaner

- Clean room that has been made free or clean room daily.

Receptionist -

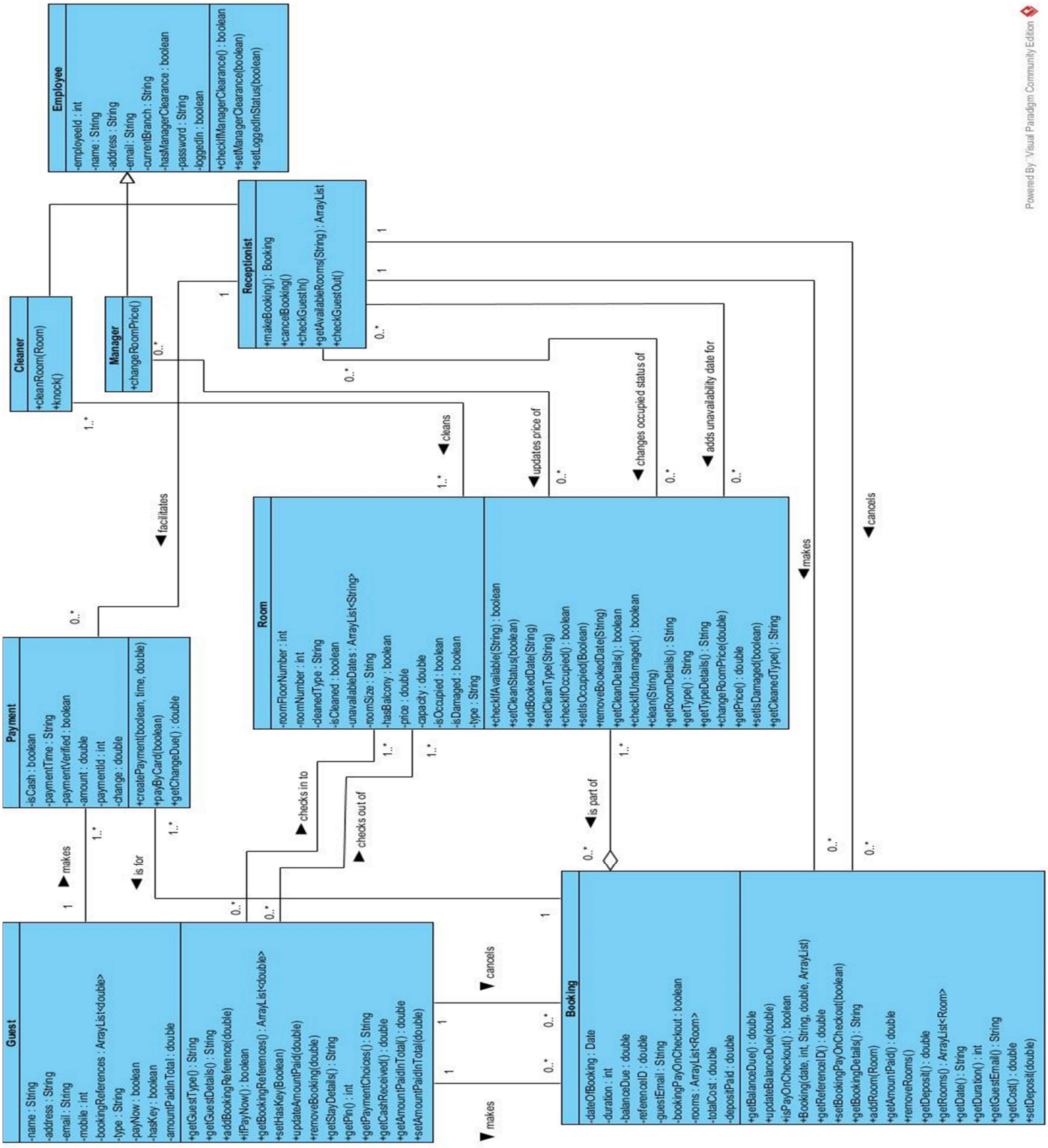
- Helps guest to check in to the hotel room.
- Helps guest to check out of the hotel room.
- Assists guest to cancel booking.
- Assists guest to make booking.
- Checks if rooms are clean.
- Checks if rooms are available.
- Helps guest to make payment.
- Gives back deposit.
- Alerts guest to check out if needs be.
- Assists to change booking.
- Logs on to the system

Guest –

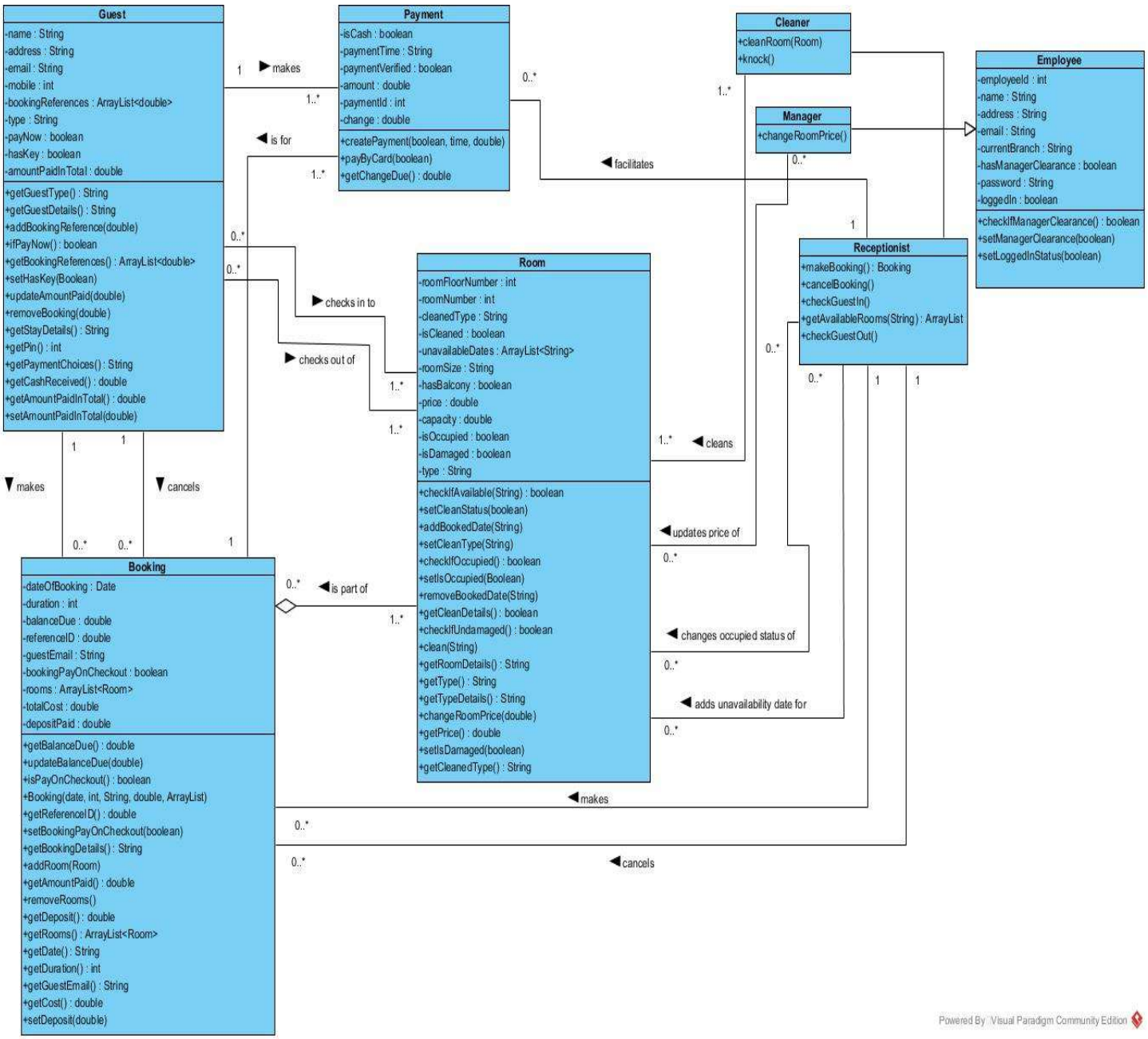
- Books a room online / phone / web app
- Pays for room during booking
- Pays when checking out
- Changes a booking
- Checks in
- Checks out
- Cancels booking
- Checks availability of room

2. Class Diagram

2.1 Landscape



2.2 Portrait


Guest
-name : String
-address : String
-email : String
-mobile : int
-bookingReferences : ArrayList<double>
-type : String
-payNow : boolean
-hasKey : boolean
-amountPaidInTotal : double
+getGuestType() : String
+getGuestDetails() : String
+addBookingReference(double)
+ifPayNow() : boolean
+getBookingReferences() : ArrayList<double>
+setHasKey(Boolean)
+updateAmountPaid(double)
+removeBooking(double)
+getStayDetails() : String
+getPin() : int
+getPaymentChoices() : String
+getCashReceived() : double
+getAmountPaidInTotal() : double
+setAmountPaidInTotal(double)
Payment
-isCash : boolean
-paymentTime : String
-paymentVerified : boolean
-amount : double
-paymentId : int
-change : double
+createPayment(boolean, time, double)
+payByCard(boolean)
+getChangeDue() : double
Cleaner
+cleanRoom(Room)
+knock()
Manager
+changeRoomPrice()
Employee
-employeeId : int
-name : String
-address : String
-email : String
-currentBranch : String
-hasManagerClearance : boolean
-password : String
-loggedIn : boolean
+checkIfManagerClearance() : boolean
+setManagerClearance(boolean)
+setLoggedInStatus(boolean)
Receptionist
+makeBooking() : Booking
+cancelBooking()
+checkGuestIn()
+getAvailableRooms(String) : ArrayList
+checkGuestOut()
Room
-roomFloorNumber : int
-roomNumber : int
-cleanedType : String
-isCleaned : boolean
-unavailableDates : ArrayList<String>
-roomSize : String
-hasBalcony : boolean
-price : double
-capacity : double
-isOccupied : boolean
-isDamaged : boolean
-type : String
+checkIfAvailable(String) : boolean
+setCleanStatus(boolean)
+addBookedDate(String)
+setCleanType(String)
+checkIfOccupied() : boolean
+setIsOccupied(Boolean)
+removeBookedDate(String)
+getCleanDetails() : boolean
+checkIfUndamaged() : boolean
+clean(String)
+getRoomDetails() : String
+getType() : String
+getTypeDetails() : String
+changeRoomPrice(double)
+getPrice() : double
+setIsDamaged(boolean)
+getCleanedType() : String
Booking
-dateOfBooking : Date
-duration : int
-balanceDue : double
-referenceID : double
-guestEmail : String
-bookingPayOnCheckout : boolean
-rooms : ArrayList<Room>
-totalCost : double
-depositPaid : double
+getBalanceDue() : double
+updateBalanceDue(double)
+isPayOnCheckout() : boolean
+Booking(date, int, String, double, ArrayList)
+getReferenceID() : double
+setBookingPayOnCheckout(boolean)
+getBookingDetails() : String
+addRoom(Room)
+getAmountPaid() : double
+removeRooms()
+getDeposit() : double
+getRooms() : ArrayList<Room>
+getDate() : String
+getDuration() : int
+getGuestEmail() : String
+getCost() : double
+setDeposit(double)
makes
is for
facilitates
checks in to
checks out of
cleans
updates price of
changes occupied status of
adds unavailability date for
is part of
makes
cancels
Powered By Visual Paradigm Community Edition

3. Use Case Descriptions

3.1 Use Case 1: Check In

3.1.1 Main Success Scenario

1. Guest presents booking reference ID if booked through email, web-app, website, or in person.
2. System implements Use Case 11 (Employee log in).
3. Receptionist chooses “Check-In” option displayed on the system.
4. System asks receptionist to input reference booking number.
5. Guest is asked for booking.
6. Receptionists enters the booking number in a box prompt.
7. System displays booking with name, and room booked.
8. Receptionist chooses “Check if Room is Ready”.
9. System displays the room is clean and ready for check in.
10. Receptionists gives key of the room chosen by the system.
11. Guest obtains a key, whether it is a card with a chip or a standard key.
12. System is updated, and the room is no longer available.
13. System returns to the welcome screen.
14. Guest has successfully checked in.

3.1.2 Extended Scenario 1: Guest lost booking page

As above (1 to 14) but with the following modifications:

Alternative: Guest has lost booking page, but knows the email he has booked with.

- At step 6,
    a) Receptionist chooses “Retrieve booking”.
    b) System displays message – “Enter Booking Email”.
    c) Receptionist asks guest for email.
    d) Receptionist inputs email into box.
    e) System retrieves Information about booking, including details like name and D.O.B
    f) Receptionists asks Guest for name and date of birth.
    g) Details provided by guest are correct.

The scenario continues at step 8.

3.1.3 Extended Scenario 2: Guest lost booking page, and gives wrong details

As above (1 to 14) but with the following modifications:

Alternative: Guest has lost booking page, but knows the email he has booked with, but when asked by receptionist for name and D.O.B to confirm identity he fails to do so.

- At step 6,

a) Receptionist chooses “Retrieve booking”.
b) System displays message – “Enter Booking Email”.
c) Receptionist asks guest for email.
d) Receptionist inputs email into box.
e) System cannot find the booking in the database.
f) Receptionist presses “Abort booking”.
g) Guest is asked to find out the email that was used to book.
h) System returns to the welcome screen menu.

Check in was not successful, but that is because of the user giving incorrect email address that was used to book. Guest is asked to return when he/she find out email address used to book.

3.1.4 Extended Scenario 3: Room is not ready

As above (1 to 14) but with the following modifications:

Alternative: System sees the room is in a “Clean state” after users have Checked out and awaits the rooms to be cleaned.

- At step 9,

a) System displays the room is not ready.
b) System displays message on screen – “Please tell guest to wait for their room to be cleaned”.
c) Receptionist clicks “OK” message.
d) System goes back to welcome screen menu.
e) Receptionist tells guest his/her check in will be available in a few minutes after the room becomes available.

The scenario continues at step 1 as some other guest may have wanted to check in into a different room that was available.

3.1.5 Extended Scenario 4: More than one room in a booking

As above (1 to 14) but with the following modifications:

Alternative: On booking, more than one room were shown, hence why more than one room must be checked in.

- At step 7,

a) System displays many rooms that were booked.
b) Receptionist chooses “Check if Rooms are Ready”.
c) System displays the rooms that are clean.
d) Receptionist chooses all rooms that are available for check in.
e) System acknowledges check in for these rooms.
f) Receptionist informs guest of the rooms they are not yet checked into, and tells them to wait until called to check into the remaining rooms.

The scenario continues at step 11 and the receptionist gives more than one key for the rooms that were made on the booking.

3.2 Use Case 2: Check Out

3.2.1 Main Success Scenario

1. Guest approaches Receptionist with the aim of checking out.
2. System implements Use Case 11 (Employee log in).
3. Receptionists asks guest for the key.
4. Guest gives key.
5. Receptionist obtains key and enters key number on the system.
6. System displays the room guest was in.
7. Receptionists asks Guest about personal details to confirm identity.
8. Guest responds with the details.
9. Receptionists enters the data in the system.
10. System confirm the details and successfully checks out the guest.
11. Receptionists presses a button "send cleaner" to check quality of room.
12. System after a short period of time, displays room quality is good and deposit can be obtained.
13. Receptionist presses give deposit.
14. System displays the cash amount owed for the deposit.
15. Receptionists gives out cash payment to the user for maintaining the rules of the hotel.
16. System prints out the receipt for the transaction.
17. The user gets his/hers deposit back and acknowledges a successful check-out.
18. System displays room map with room that was checked out to be in "Clean" state.
19. System returns to welcome screen menu after 10 seconds.

3.2.2 Extended Scenario 1: Incorrect Key Number Value Entered

As above (1 to 19) but with the following modifications:

Alternative: Wrong Room Displayed.

- At step 6,
    a) System fails to verify Room Number Information.
    b) System displays message – "Sorry, An Error Occurred in Room Number, Please Enter Again!"
    c) Receptionist chooses "OK" to continue with another attempt.

The scenario continues at step 5 with the receptionist having to input data again.

3.2.3 Extended Scenario 2: Pay When Checking-out

As above (1 to 19) but with the following modifications:

Alternative: Guest wishes to pay for some/all of the bookings at the time of checking-out.

- At step 14,
    a) System displays message – "Booking not paid for yet".
    b) Receptionist chooses "Pay Now".
    c) System implements Use Case 5 (Make Payment).

The scenario continues at step 22.

3.2.4 Extended Scenario 3: No deposit back

As above (1 to 19) but with the following modifications:

Alternative: Guest did not keep the standard of the room to an excellent quality.

- At step 12,
    a) System displays – "Room not of good quality".
    b) Receptionist informs the user that a deposit will not be obtained.

The scenario continues at step 18.

3.2.5 Extended Scenario 4: No deposit back, but guest did not pay for booking yet.

As above (1 to 19) but with the following modifications:

Alternative: Guest did not keep the standard of the room to an excellent quality, but did not pay for booking.

- At step 12,
    a) System implements Scenario 5.
    b) System displays message "Booking not yet paid for"
    c) Receptionist chooses "Pay Now".
    d) System implements Use Case 5 (Make Payment).

The scenario continues at step 18.

3.2.6 Extended Scenario 5: Late Check-Out.

As above (1 to 19) but with the following modifications:

Alternative: Guest did not check out in time specified on booking.

- At step 1,
    a) Guest did not come in time to check-out.
    b) System shows guests and room numbers that had to check-out on a specific day.

c) Receptionist presses on the room number.
d) System displays phone number to room.
e) Receptionists calls room phone.
f) User answers and is told he will pay for an extra night due to not checking out.
g) Receptionist enters booking reference.
h) System displays original date with an edit option.
i) Receptionist edits option and adds on a further date.

The scenario continues at step 1, when user once against returns to check-out.

3.2.7 Extended Scenario 6: Late Check-Out Payment.

As above (1 to 19) and scenario 5 but with the following modifications:

Alternative: Guest did not check out in time specified on booking and must pay for it

- At step 14,
  a) System implements Scenario 7.
  b) System displays – "new date, new price".
  c) Receptionist asks guest to pay for the extra night.
  d) System implements Use Case 5 (Make Payment).

The scenario continues at step 16.

3.2.8 Extended Scenario 7: Phone alert.

As above (1 to 19) but with the following modifications:

Alternative: Guest wishes to be alerted over the phone, for example not to miss a flight.

- Before step 1,
  a) User attends reception day/night before check-out.
  b) User wants to be alerted at a specific time to check-out.
  c) System is in main screen
  d) Receptionist logs onto system and chooses "Alert Resident option".
  e) System displays – "Enter Room Number".
  f) Receptionists asks user for key.
  g) User gives key.
  h) Receptionist inputs number into system.
  i) System displays clock and calendar.
  j) Receptionists asks user what time and day.
  k) User responds with time and day.
  l) Receptionist inputs data into the system.
  m) System displays – "Alert set".

The scenario continues at step 1 with a guest checking out.

3.2.9 Extended Scenario 8: Deposit on cash

As above (1 to 19) but with the following modifications:

Alternative: Guest wants the deposit on card

- At step 15,
    a) System displays – "Give Card Deposit -> insert card".
    b) Use Case 5, make payment

The scenario continues at step 16.

3.3 Use Case 3: Make Booking

3.3.1 Main Success Scenario

1. Guest approaches Receptionist with the aim of making a booking.
2. System implements Use Case 11 (Employee log in).
3. System implements Use Case 6 (Check Availability).
4. Guest tells Receptionist their choice of room.
5. Receptionist chooses room type.
6. System displays booking date and chosen room(s) details.
7. Receptionist chooses “Continue with Booking”.
8. System prompts for Guest Type.
9. Receptionist asks Guest if they are a previous customer.
10. Guest indicates they are a new customer and do not have an account.
11. Receptionist chooses “New Guest”.
12. System prompts for guest details.
13. Receptionist asks Guest for the relevant details.
14. Receptionist inputs guest name, address, mobile, and email address.
15. System displays booking and guest details.
16. Guest indicates they wish to pay full amount when checking out.
17. Receptionist chooses “Pay on Checkout” from dropdown menu.
18. Receptionist chooses “Finish Booking”.
19. System allocates Reference Number to booking.
20. System displays Reference Number.
21. System adds booking and registered guest to database.
22. Receptionist emails booking confirmation to Guest, and informs them of the Reference Number.
23. Guest acknowledges successful booking; their aim has been achieved.
24. System logs Employee out of system.
25. System returns to welcome screen menu.

3.3.2 Extended Scenario 1: Multiple Rooms in Booking

As above (1 to 25) but with the following modifications:

Alternative: Guest requires multiple rooms to be included in the booking.

- At step 7,
    a) Receptionist chooses “Add another Room”.

The scenario continues at step 4 to allow additional room to be chosen.

3.3.3 Extended Scenario 2: Registered Guest

As above (1 to 25) but with the following modifications:

Alternative: Guest is registered as a previous customer – details already in database.

- At step 10,
    a) Guest indicates they are a registered guest.
    b) Receptionist chooses "Registered Guest".
    c) System prompts Receptionist for Guest email.
    d) Receptionist asks Guest for their email address.
    e) Receptionist inputs email into system.
    f) System checks database for Guest details, using email a unique identifier.
    g) System displays message – "Guest Account Found!"

The scenario continues at step 15 to display guest details.

3.3.4 Extended Scenario 3: Pay When Booking

As above (1 to 25) but with the following modifications:

Alternative: Guest wishes to pay for some/all of the booking at the time of booking.

- At step 17,
    a) Guest indicates they would like to pay deposit/full amount.
    b) Receptionist chooses "Pay Now".
    c) System implements Use Case 5 (Make Payment).

The scenario continues at step 18.

3.3.5 Extended Scenario 4: Error Adding To Database

As above (1 to 25) but with the following modifications:

Alternative: System fails to add booking to database.

- At step 21,
    a) System cannot add booking/guest to database.
    b) System displays error message – "Error Adding to Database – Please Try Again!"

The scenario continues at step 18 to allow the Receptionist to confirm the booking and the System will retry adding the booking/guest to the database.

3.4 Use Case 4: Cancel Booking

3.4.1 Main Success Scenario

1. Guest contacts Receptionist with the aim of cancelling a booking.
2. System implements Use Case 11 (Employee log in).
3. Receptionist chooses “Cancel Booking”.
4. System continues with cancellation by prompting for Booking Reference ID number.
5. Receptionist asks Guest for their reference number.
6. Guest provides Booking Reference ID.
7. Receptionist inputs ID number into System.
8. System checks database for ID number and verifies.
9. System displays Booking Details.
10. Receptionist chooses “Delete Booking”.
11. System prompts for confirmation – “Are you sure?”
12. Receptionist confirms deletion with Guest.
13. Receptionist chooses “Yes” on System dialog.
14. System removes booking details from database.
15. System shows balance (if any) due to Guest following booking deletion.
16. System tells cash is owed to Guest.
17. System indicates cash payment was made and opens till.
18. Receptionist removes refund amount from till and gives it to the Guest.
19. Receptionist closes till.
20. Receptionist sends deletion confirmation email and refund receipt to Guest.
21. System logs Receptionist out.
22. System returns to welcome screen.

3.4.2 Extended Scenario 1: Incorrect Reference Number

As above (1 to 22) but with the following modifications:

Alternative: The reference number input did not correspond to any bookings in the database.

- At step 8,
    a) System cannot find booking in database.
    b) System displays error message – “Error – Cannot find booking for this ID, Please try again!"

The scenario continues at step 6 to allow the correct reference number be entered.

3.4.3 Extended Scenario 2: Deletion Cancelled

As above (1 to 22) but with the following modifications:

Alternative: The guest changes their mind about deleting their booking.

- At step 12,
    a) Guest indicates they would not like to proceed with cancelling the booking.
    b) Receptionist chooses "No" on system.
    c) System displays message – "Deletion Cancelled – Booking intact!"

The scenario continues at step 21.

3.4.4 Extended Scenario 3: Cannot Delete from Database

As above (1 to 22) but with the following modifications:

Alternative: The System cannot remove the booking entry from the database.

- At step 14,
    a) System cannot delete the booking from the database.
    b) System displays error message – "Error Deleting from Database – Please Try Again!"

The scenario continues at step 9 to allow the user confirm deletion and retry the booking cancellation process.

3.4.5 Extended Scenario 4: Debit/Credit Card Refund

As above (1 to 22) but with the following modifications:

Alternative: The Booking was previously paid by card payment, so refund must be put on to card.

- At step 17,
    a) System detects card payment was used originally.
    b) System responds by asking the Guest to insert debit/credit card into designated card reader.
    c) Guest inserts debit/credit card into reader in the correct position.
    d) System acknowledges card by displaying message – "Card Inserted!"
    e) System credits the refund amount to the card account.
    f) System asks the Guest to remove the card from the machine.
    g) Guest removes card from the card reader.

The scenario continues at step 20.

3.4.6 Extended Scenario 5: No Booking Reference Number

As above (1 to 22) but with the following modifications:

Alternative: Guest cannot provide reference ID number, but can provide email used for booking.

- At step 6,
-

a) Guest cannot provide Reference Number for booking.
b) Receptionist chooses “Retrieve Booking without ID”.
c) System prompts for email input by displaying message – “Enter Booking Email”.
d) Receptionist asks guest for email.
e) Receptionist inputs email into system.
f) System retrieves information about future bookings in database with this email, including Guest information details.
g) Receptionists asks Guest for name and date of birth.
h) Receptionist verifies details provided by guest are correct by matching them with booking details.
i) Receptionist chooses the relevant booking.

The scenario continues at step 9.

3.5 Use Case 5: Make Payment

3.5.1 Main Success Scenario

1. System prompts for amount to be paid.
2. Receptionist asks Guest how much they would like to pay.
3. Guest responds with amount.
4. Receptionist inputs amount due.
5. System validates amount entered.
6. System prompts for payment type.
7. Guest indicates they would like to pay with cash.
8. Receptionist chooses “Pay with Cash” option.
9. System displays current balance due and opens till drawer.
10. Receptionist prompts Guest for cash.
11. Guest provides Receptionist with cash.
12. System prompts Receptionist for cash amount received.
13. Receptionist inputs amount received.
14. System calculates and displays change due.
15. Receptionist puts received cash into till, and retrieves the correct change amount.
16. Receptionist provides Guest with change.
17. System updates balance due for booking in database to total cost minus amount paid.

3.5.2 Extended Scenario 1: Card Payment

As above (1 to 17), omitting 8-16, but with the following modifications:

Alternative: Card Payment Chosen

- At step 7,
    a) Guest indicates they would like to pay by card.
    b) Receptionist chooses “Pay by Card” option.
    c) Receptionist places card reader on desk for Guest to use.
    d) System implements Use Case 9 (Card Payment).

The scenario continues at step 17.

3.5.3 Extended Scenario 2: Cannot Validate Amount

As above (1 to 17), but with the following modifications:

Alternative: Amount entered is incorrect

- At step 5,
    a) System cannot validate amount entered.
    b) System displays message – “Please enter a valid amount”.

The scenario continues at step 2 to allow for amount to be re-entered.

3.5.4 Extended Scenario 3: Change Payment Type

As above (1 to 17), but with the following modifications:

Alternative: Guest changes their mind and requires a different payment option.

- At step 11,
    a) Guest indicates they would like to change payment option.
    b) Receptionist chooses “Change Payment Type”.

The scenario continues at step 6 to allow for payment type to be changed.

3.5.5 Extended Scenario 4: Combination of Payment Types

As above (1 to 17), but with the following modifications:

Alternative: Guest wishes to pay subtotal of the booking cost.

- At step 7,
    a) Guest indicates they would like to pay with a combination of cash and card.
    b) Receptionist chooses “Pay with Combination” option.
    c) System prompts receptionist for amount to be paid by card.
    d) Receptionist asks Guest how much they would like to charge to their debit/credit card.
    e) Guest indicates the chosen amount.
    f) Receptionist inputs amount for card payment.
    g) System implements Use Case 9 (Card payment).

Scenario continues at step 9 to carry out the cash payment portion of the transaction.

3.6 Use Case 6: Check Availability

3.6.1 Main Success Scenario

1. System prompts for booking date.
2. Receptionist asks Guest the date and duration of their stay.
3. Receptionist enters the guest's intended date of stay.
4. System verifies the date entered.
5. System prompts for the duration of stay to be entered.
6. Receptionist inputs the number of nights the booking is required for.
7. System displays the available rooms for the date.
8. Receptionist informs Guest of the available room types.
9. Guest confirms they are satisfied with the list of available rooms for their intended date of stay.

3.6.2 Extended Scenario 1: Change Date/Duration

As above (1 to 9) but with the following modifications:

Alternative: Required room type not listed as available for selected date – Guest decides to change the date/duration of stay.

- At step 9,
    a) Guest indicates they would like to change the intended date/duration of stay.
    b) Receptionist chooses "Change Date"

The scenario continues at step 1 to prompt detail entry again.

3.6.3 Extended Scenario 2: Invalid Date Entered

As above (1 to 9) but with the following modifications:

Alternative: Invalid Date Entered

- At step 4,
    a) System cannot verify date entered (incorrect format/ past date/ too far in future.
    b) System displays message to Receptionist – "Date entered is invalid, Please enter valid date"
    c) Receptionist asks Guest for suitable valid date.

The scenario continues at step 1 to prompt date entry again.

3.7 Use Case 7: Change Booking

3.7.1 Main Success Scenario

1. Guest approaches Receptionist with the aim of changing a booking.
2. System implements Use Case 11 (Employee log in).
3. System implements Use Case 4 (Cancel Booking) from step 10 - 25.
4. System implements Use Case 3 (Make Booking) from step 9 - 28.
5. System returns to welcome screen menu.

3.8 Use Case 8: Modify Prices

3.8.1 Main Success Scenario

1. System displays welcome screen menu.
2. User chooses “Modify Pricing” option.
3. System implements Use Case 11 (Employee log in).
4. System verifies logged in Employee has Managerial Clearance.
5. System prompts user to choose room type.
6. User chooses room type.
7. System displays current room details and price.
8. User chooses “Modify Price” option.
9. System prompts user to input new price.
10. User inputs new price and confirms entry.
11. System verifies price entered.
12. System displays provisioned updated room details.
13. User chooses “Confirm Changes” option.
14. System updates database for room type price.
15. System displays further options menu.
16. User chooses “Logout” option.
17. System returns to welcome screen.

3.8.2 Extended Scenario 1: Invalid Price Entered

As above (1 to 17) but with the following modifications:

Alternative: Manager attempts to set a room price that is outside the allowable range.

- At step 11,
    a) System declines new price amount entered.
    b) System displays message – “Invalid Price Entered – Too high/low! Please Try Again”
    c) User chooses “OK” to continue with retry.

The scenario continues at step 9 to prompt user to input price.

3.8.3 Extended Scenario 2: Error Updating Database

As above (1 to 17) but with the following modifications:

Alternative: System fails to update room type in database.

- At step 14,
    a) System cannot update the room type price in the database.
    b) System displays error message – "Error Updating Database – Please Try Again!"

The scenario continues at step 13 to allow the user confirm changes and retry the update process.

3.8.4 Extended Scenario 3: Make Multiple Price Changes

As above (1 to 17) but with the following modifications:

Alternative: Manager requires changing multiple room type prices.

- At step 15,
    a) User chooses "Make Additional Change" option.

The scenario continues at step 5 to allow the user to change the price for another room type.

3.8.5 Extended Scenario 4: Employee is not a Manager

As above (1 to 17) but with the following modifications:

Alternative: Employee logged in but does not have clearance to modify prices.

- At step 4,
    a) System cannot verify Employee has Managerial Clearance.
    b) System displays error message – "Sorry, You do not have clearance levels to modify prices - logging out..."
    c) System logs out user.

The scenario continues at step 17.

3.9 Use Case 9: Pay by Card

System aims to carry out a card transaction with the user. This Use Case is only internally called by the system during Make Payment.

User: Guest

3.9.1 Main Success Scenario

1) System responds by asking the user to insert debit/credit card into designated card reader.
2) User inserts debit/credit card into reader in the correct position.
3) System acknowledges card and validates correct card type inserted.
4) System prompts user for PIN number.
5) User enters PIN number on keypad provided, and presses "Enter" to indicate completion.
6) System authorises PIN number.
7) System verifies the account provided can incur the cost of the booking.
8) System charges the booking amount to the account provided.
9) System asks the user to remove the card from the machine.
10) User removes card from the card reader.

3.9.2 Extended Scenario 1: Invalid Debit/Credit Card

As above (1 to 4), with the following modifications:

Alternative: Debit/Credit card inserted is invalid.

- At step 4, System cannot validate card.
  a) System displays "Cannot validate payment card".

Scenario continues from Use Case 5 (Make Payment) – step 5, to allow for alternative payment type.

3.9.3 Extended Scenario 2: Incorrect PIN Entry

As above (1 to 7), with the following modifications:

Alternative: PIN number entered by User is incorrect.

- At step 7, System receives incorrect PIN number authorisation message from Bank services
  a) System displays "Incorrect PIN" message to user
  b) System returns to step 5 above for PIN retry.
  →This may occur at most 3 times – at which time:

i. System prompts user to remove debit/credit card from machine and displays “Max Attempts Reached – Please use another form of payment”.
ii. User removes card.
iii. System returns to payment options for alternative payment method:

➔ Use Case 5 (Make Payment) – step 5.

3.9.4 Extended Scenario 3: Insufficient Funds

As above (1 to 8), with the following modifications:

Alternative: Credit/Debit card has insufficient funds to complete transaction.

- At step 8, System is denied payment due to the account having insufficient funds.
  a) System displays “Transaction Declined” message to user.
  b) System returns to payment options for alternative payment method:

  ➔ Use Case 5 (Make Payment) – step 5.

3.10 Use Case 10: Clean Room

User: Cleaner

3.10.1 Main Success Scenario

1. System displays welcome screen.
2. User chooses “Check Room Cleaned”.
3. User enters room number into system.
4. System retrieves room details from database.
5. System displays room availability status as “Occupied”.
6. User performs daily clean.
7. User flags room as “Clean” on system.
8. System updates room status in database.

3.10.2 Extended Scenario 1: Deep Clean Room

As above (1 to 8) but with the following modifications:

Alternative: System indicates room is unoccupied and needs a deep clean.

- At step 5,
    a) System displays room availability as “Unoccupied”.
    b) User performs deep clean.

The scenario continues at step 7.

3.11 Use Case 11: Employee log in

User: Employee (Manager/Receptionist)

3.11.1 Main Success Scenario

1. System displays welcome screen menu.
2. Employee chooses “log in” option.
3. System displays user authentication screen.
4. Employee inputs Employee ID number and password.
5. System verifies Employee login information.
6. System displays Employee and relevant branch details and prompts for confirmation.
7. Employee confirms correct account displayed.
8. System goes back to home screen.

3.11.2 Extended Scenario 1: Failed Employee Login

As above (1 to 8) but with the following modifications:

Alternative: Employee Login Fails

- At step 5,
    a) System fails to verify Employee Login Information.
    a) System displays message on screen – “Invalid Login, Please Try Again!”
    b) Employee clicks “OK” to continue with login attempt.

The scenario continues at step 3 to prompt Employee Login again.

3.11.3 Extended Scenario 2: Incorrect Employee Account Retrieved

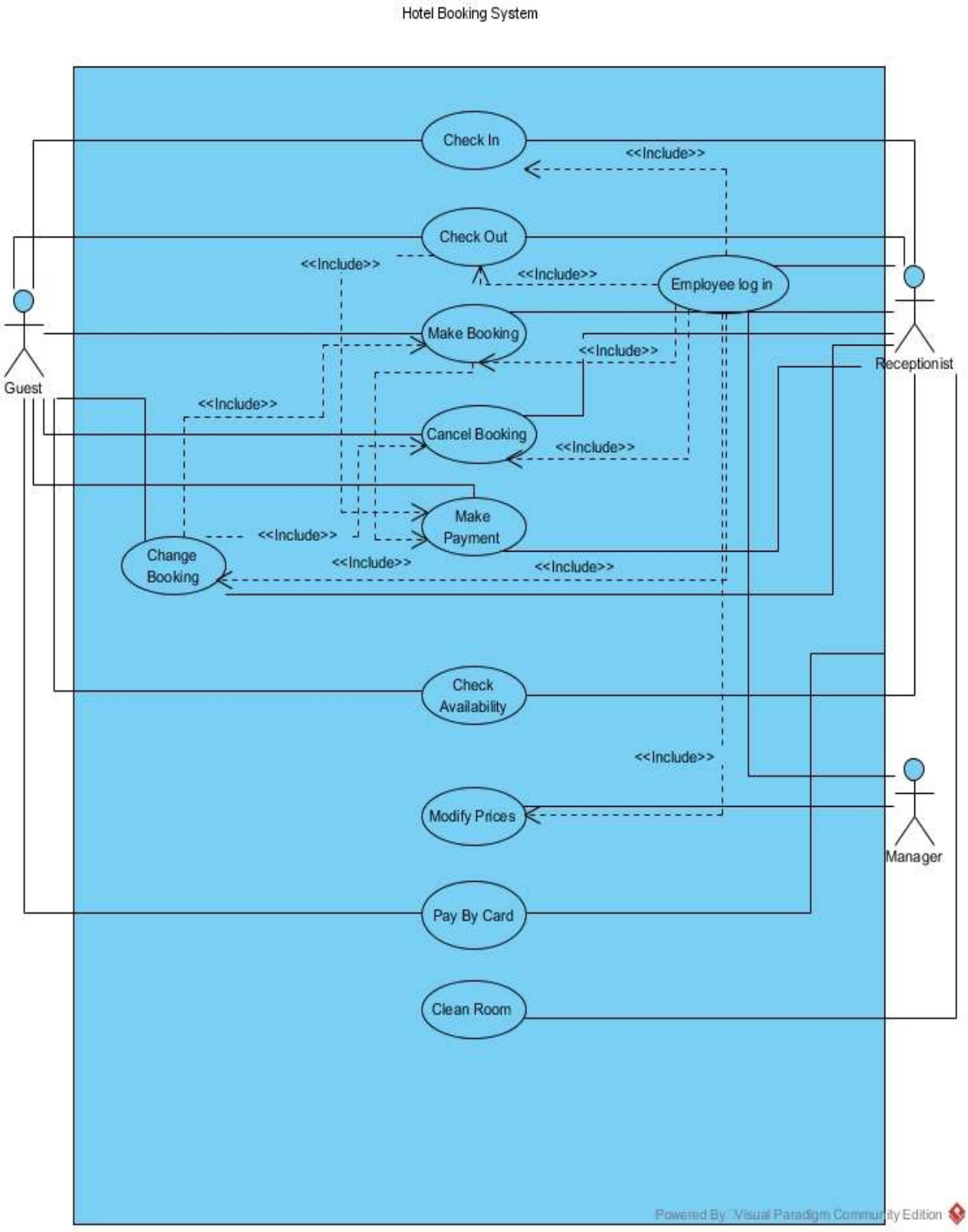
As above (1 to 8) but with the following modifications:

Alternative: Wrong Employee Account Displayed

- At step 7,
    a) Employee chooses “Incorrect Account Shown”.
    a) System displays message – “Sorry, An Error Occurred in Account Retrieval, Please Login Again!”
    b) Employee chooses “OK” to continue with another login attempt.

The scenario continues at step 3 to prompt Employee Login again.

4. Use Case Diagram



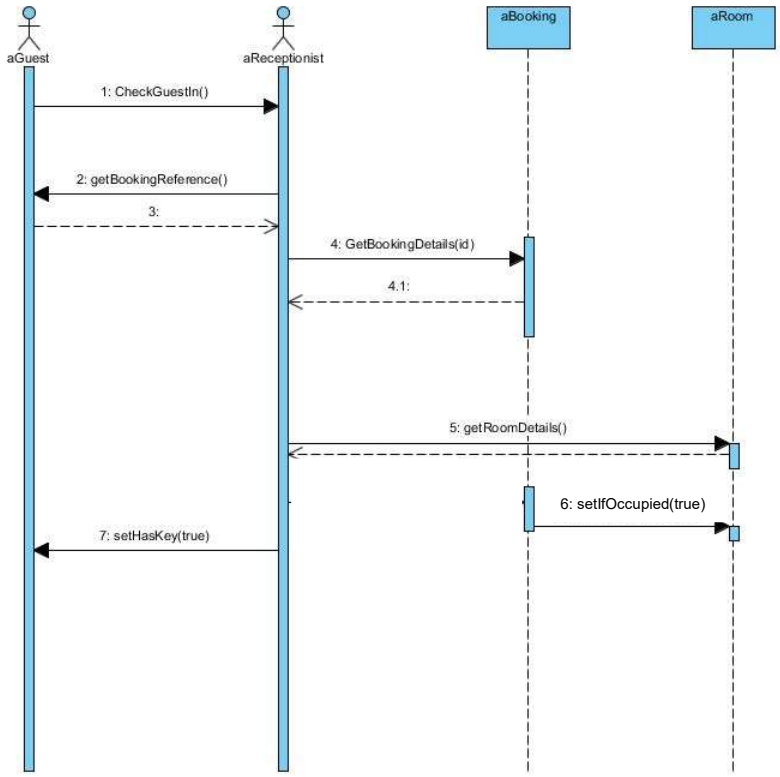
5. Sequence Diagrams.

Sequence Diagrams are for the main success scenarios of each use case, to make sure things are implemented by programmers, as if this part contains too much information, it will not be read over.

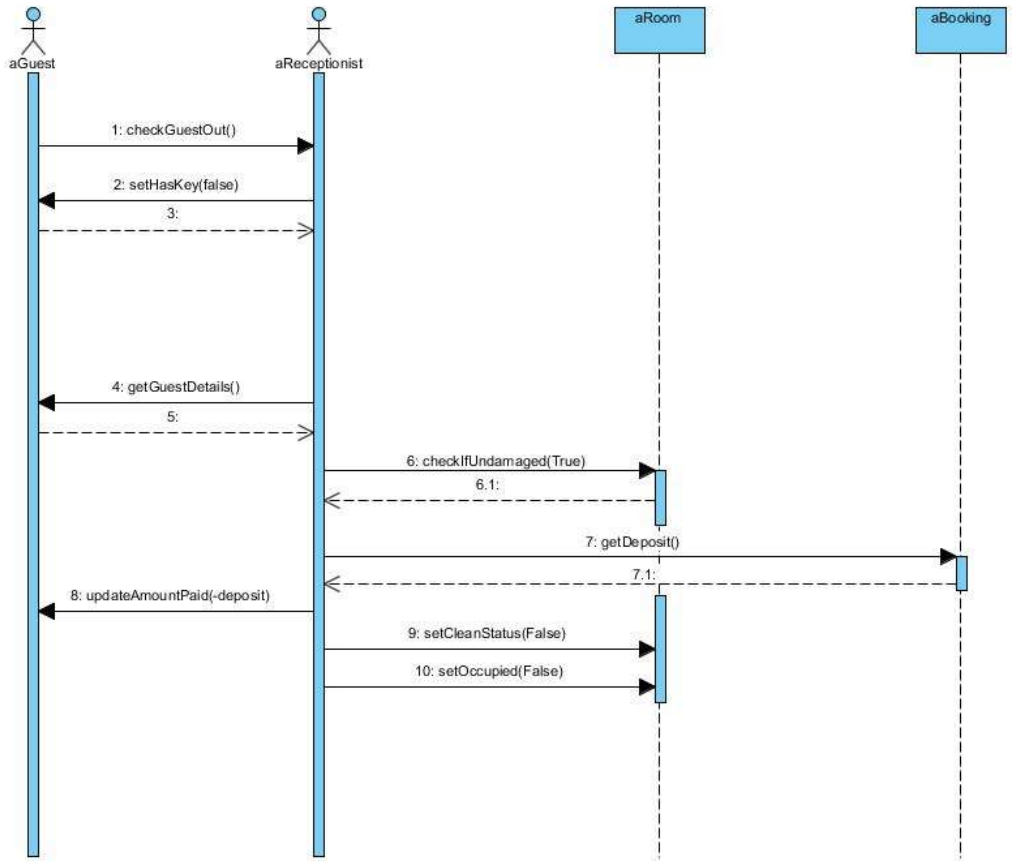
Items that are done within the system and are not included in the sequence diagrams:

- LogIn()
- LogOut()
- EmailBooking()
- GetUnnocupiedRooms()
- SearchByKey(Int)
- CheckRoomCleaned(Int)
- VerifyCard()
- VerifyPin()
- VerifyCost()
- ProcessPayment()

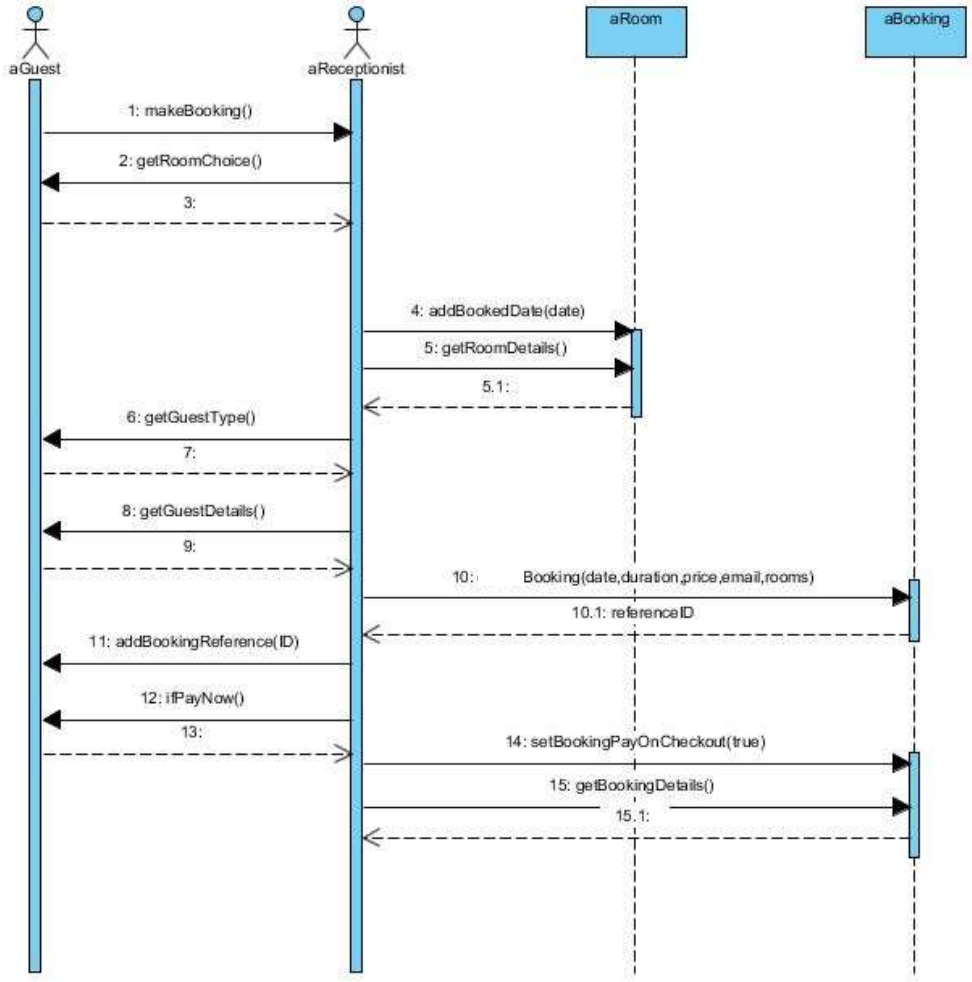
5.1 Check In:



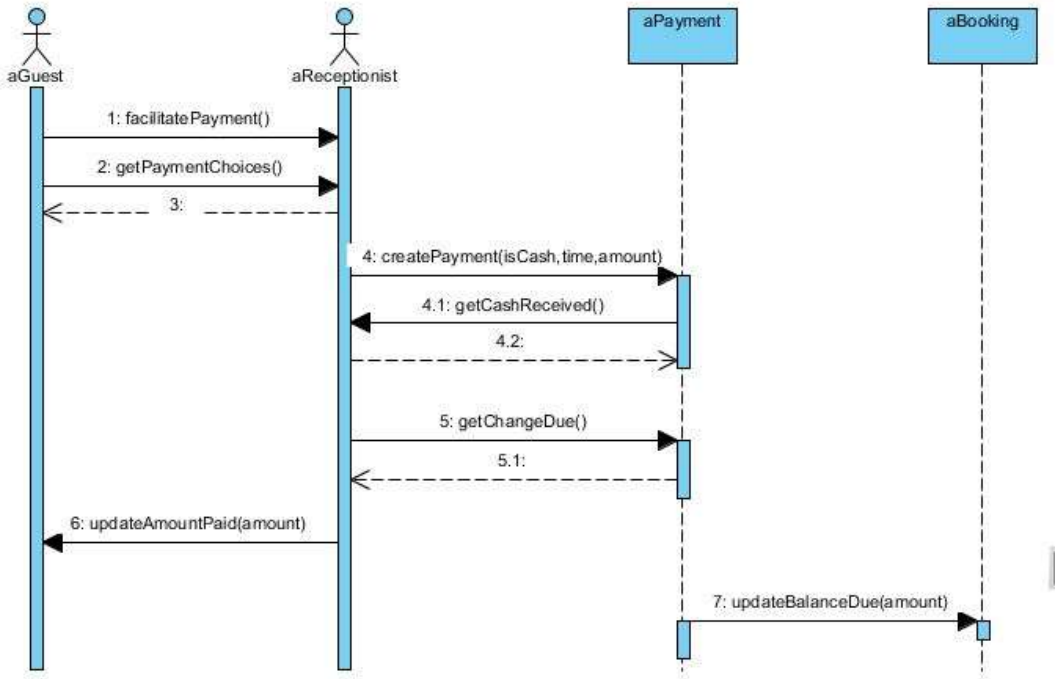
5.2 Check Out:


aGuest
aReceptionist
aRoom
aBooking
1: checkGuestOut()
2: setHasKey(false)
3:
4: getGuestDetails()
5:
6: checkIfUndamaged(True)
6.1:
7: getDeposit()
7.1:
8: updateAmountPaid(-deposit)
9: setCleanStatus(False)
10: setOccupied(False)

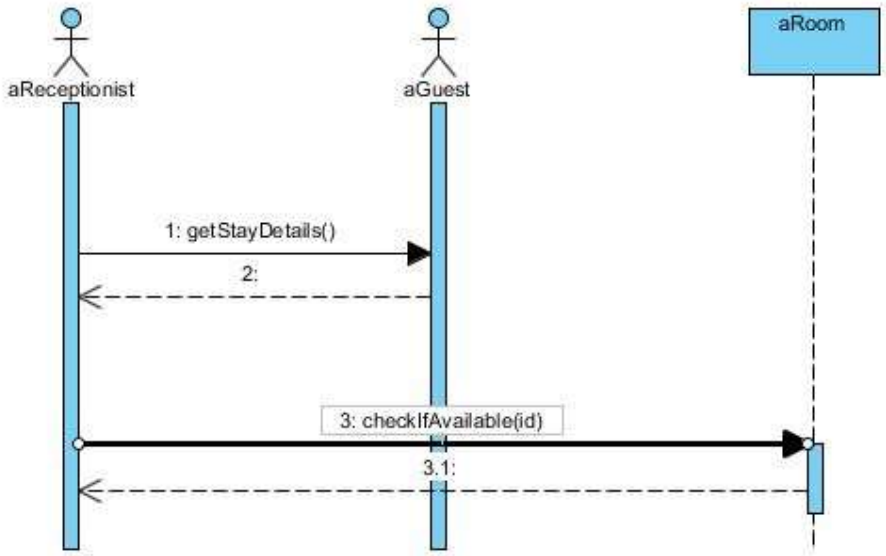
5.3 Make Booking:



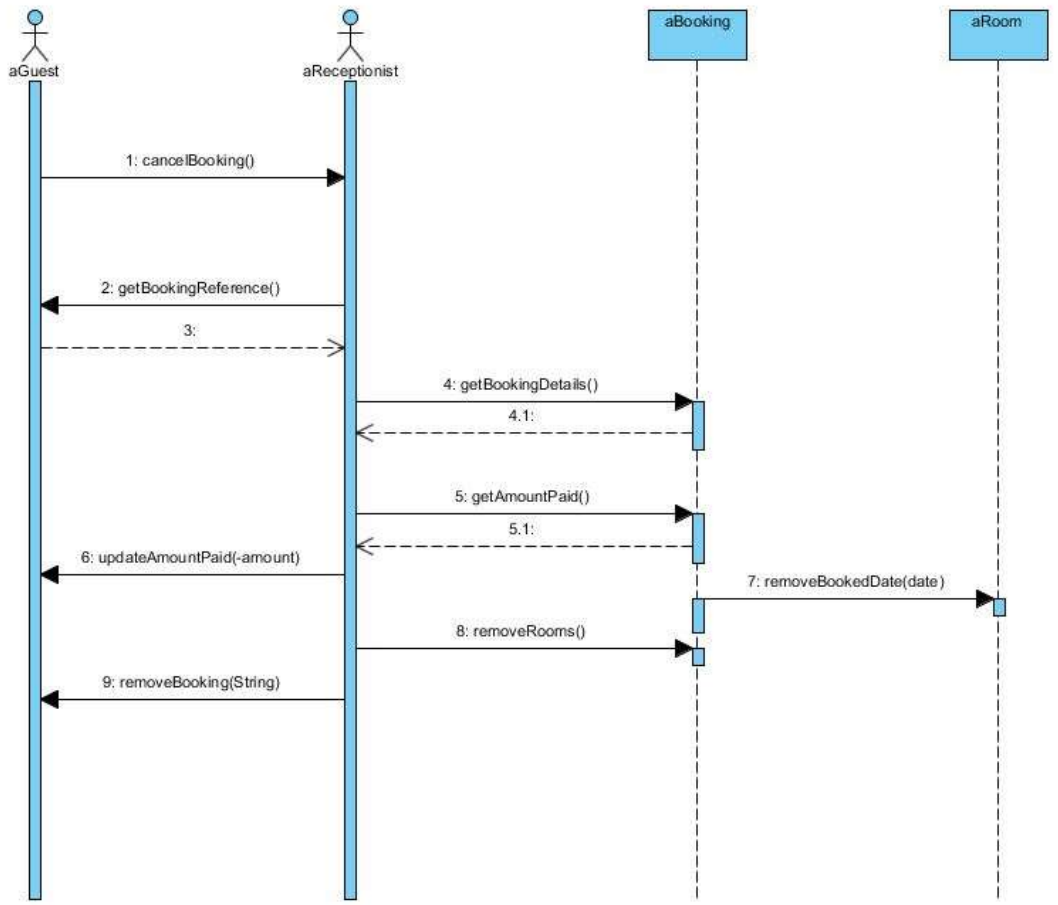
5.4 Make Payment:



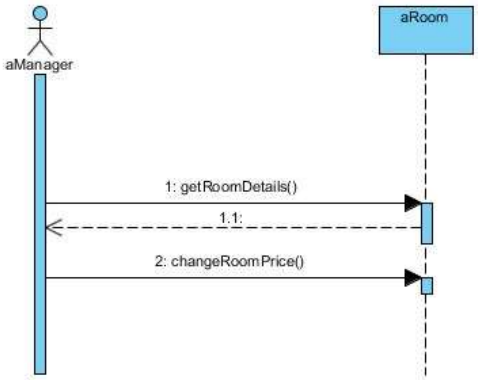
5.5 Check Availability:



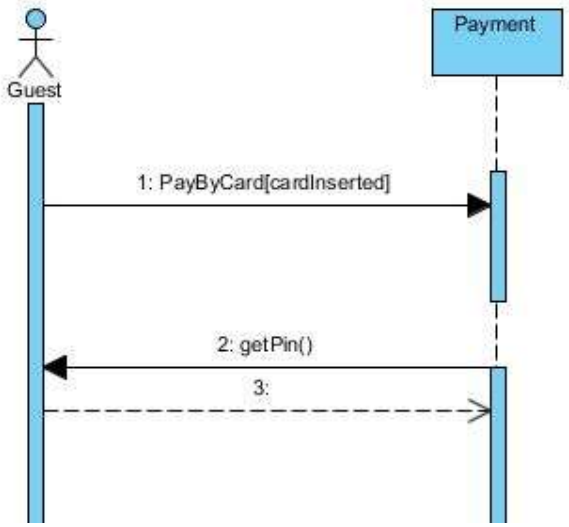
5.6 Cancel Booking:



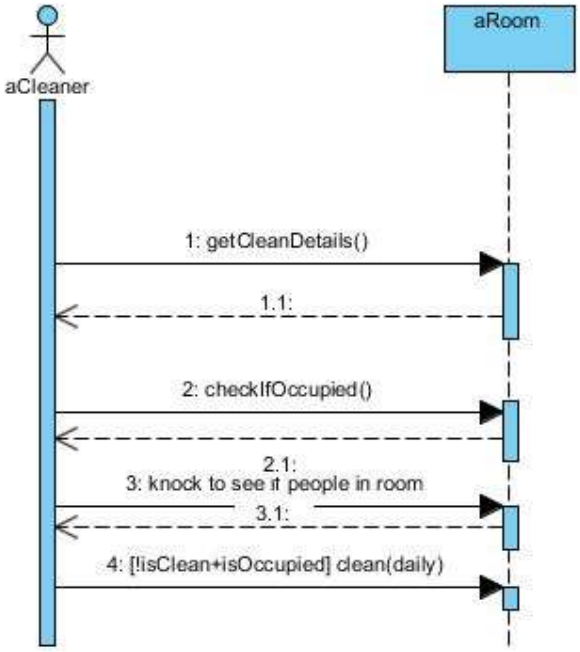
5.7 Modify Prices:



5.8 Pay By Card:

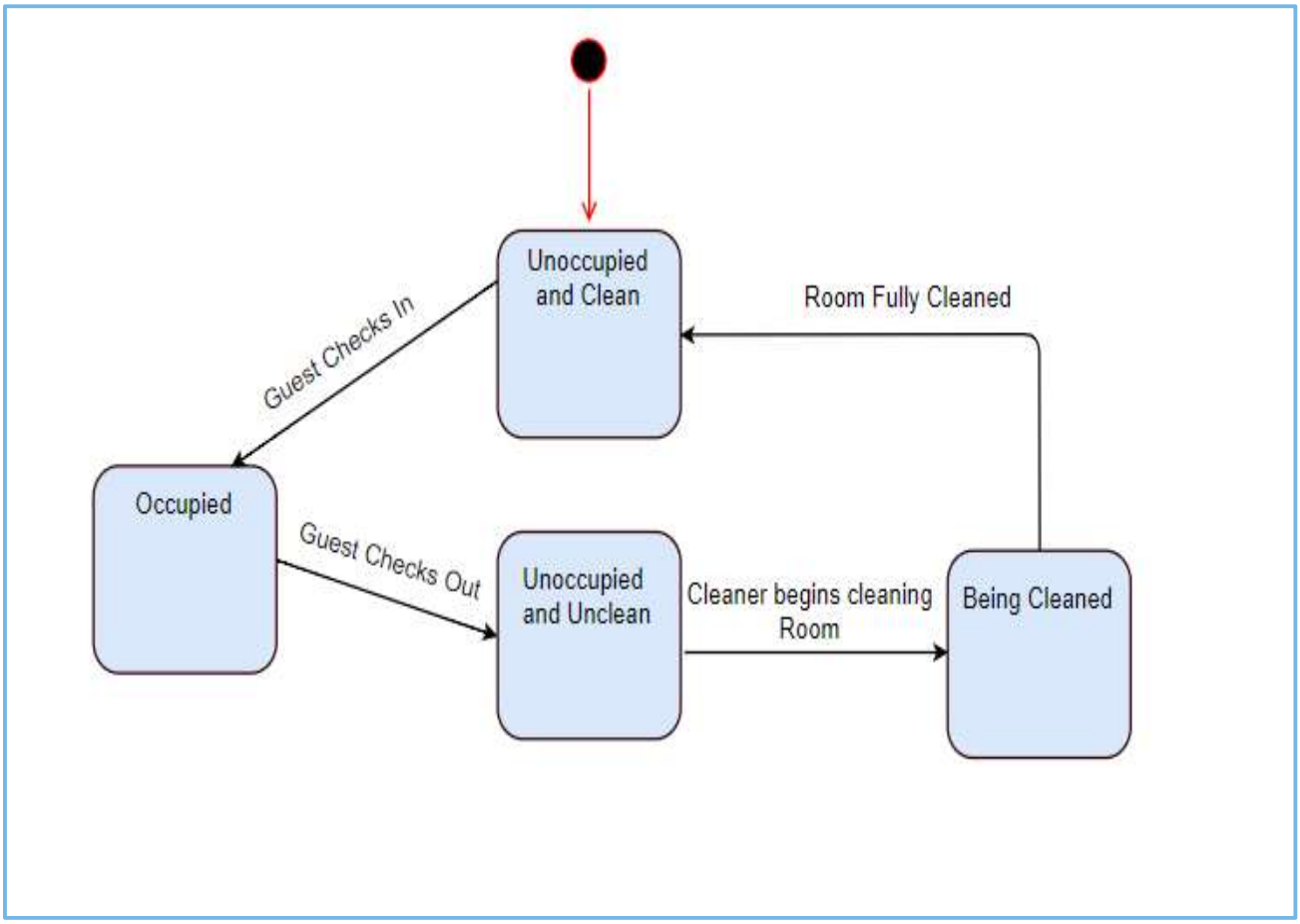


5.9 Clean Room:



6. State Diagram

6.1 Room



7. Java Code

7.1 Object Classes

7.1.1 Booking

```
import java.util.ArrayList;
import java.util.Random;


public class Booking {


	private String dateOfBooking;
	private int duration;
	private double balanceDue;
	private double referenceID;
	private String guestEmail;
	private boolean bookingPayOnCheckout;
	private ArrayList<Room> rooms;
	private double totalCost;
	private double depositPaid;



	public Booking(String dateOfBooking, int duration,  String guestEmail, double
totalCost, ArrayList<Room> rooms) {
		this.dateOfBooking = dateOfBooking;
		this.duration = duration;
		this.balanceDue = totalCost;
		this.guestEmail = guestEmail;
		this.totalCost = totalCost;
		Random rand=new Random();    // get random num for reference id (system
should ensure it is not already in database)
		this.referenceID= rand.nextInt(100000);

		for (Room room:rooms){
			room.addBookedDate(dateOfBooking);
		}

		this.bookingPayOnCheckout=true;
		this.rooms=rooms;
		this.depositPaid=0.0;

	}

	/**
	 * @return the balanceDue
	 */
	public double getBalanceDue(){
		return balanceDue;
	}

	public void updateBalanceDue(double amount){
		if(balanceDue-amount<0){
			System.out.println("Cannot pay more than is due!");
		}
		else{
		balanceDue-=amount;
		}
```

```
}

/**
 * @return the bookingPayOnCheckout
 */
public boolean isPayOnCheckout(){
return bookingPayOnCheckout;
}

/**
 * @return the referenceID
 */
public double getReferenceID() {
	return referenceID;
}

/**
 * @return the depositPaid
 */
public double getDeposit() {
	return depositPaid;
}

/**
 * @param bookingPayOnCheckout the bookingPayOnCheckout to set
 */
public void setBookingPayOnCheckout(boolean bookingPayOnCheckout) {
	this.bookingPayOnCheckout = bookingPayOnCheckout;
}

public String getBookingDetails(){
	String details="Date: "+dateOfBooking+", Duration: "+duration+
			"\nReference ID: "+referenceID+", Email: "+guestEmail+
			"\n Total: "+totalCost+", Balance Due: "+balanceDue+
			"\n Pay On Checkout? "+bookingPayOnCheckout+
			"\n Rooms: ";
	for(int i=0;i<rooms.size();i++){
		details+=rooms.get(i).getRoomDetails()+",";
	}
	return details;
}

public void addRoom(Room room){
	if(rooms.contains(room)){
		System.out.println("Room already included in booking!");
	}
	else{
	rooms.add(room);
	}
}


public double getAmountPaid(){
	return totalCost-balanceDue;
}

public void removeRooms(){
	for(Room room:rooms){
		room.removeBookedDate(dateOfBooking);
	}
	rooms.clear();
}

public ArrayList<Room> getRooms() {
	return rooms;
}
```

```
	public String getDate() {
		return dateOfBooking;
	}

	public int getDuration() {
		return duration;
	}

	public String getGuestEmail() {
		return guestEmail;
	}

	public double getCost() {
		return totalCost;
	}

	public void setDeposit(double d) {
		depositPaid=d;

	}



}
```

7.1.2 Cleaner

```
public class Cleaner extends Employee{

	public Cleaner(int employeeId, String name, String address, String email,
String currentBranch,
			boolean hasManagerClearance, String password) {
		super(employeeId, name, address, email, currentBranch, password);

	}



	public void cleanRoom(Room room){
		// Cleaner manually cleans the room
		boolean cleanedDetails=room.getCleanDetails();
		boolean occupied=room.checkIfOccupied();
		if (cleanedDetails==false){
			if(occupied==true){
				knock();  // manually done, representation here just to
show process
				room.clean("daily");
			}
			else {
				room.clean("deep clean");
			}

		}
	}

	public void knock(){  // manual knock on door, representation here just to
show process
		System.out.println("Cleaner Knocked On Door!");
	}
}
```

7.1.3 Employee

```
public class Employee {

	private int employeeId;
	private String name;
	private String address;
	private String email;
	private String currentBranch;
	private boolean hasManagerClearance;
	private String password;
	private boolean loggedIn;
	/**
	 * @param employeeId
	 * @param name
	 * @param address
	 * @param email
	 * @param startDate
	 * @param currentBranch
	 * @param password
	 */
	public Employee(int employeeId, String name, String address, String email,
String currentBranch,
			String password) {
		this.employeeId = employeeId;
		this.name = name;
		this.address = address;
		this.email = email;
		this.currentBranch = currentBranch;
		this.hasManagerClearance = false;
		this.password = password;
		this.loggedIn=false;
	}

	public boolean checkIfManagerClearance(){
		return hasManagerClearance;

	}

	public void setManagerClearance(boolean clearance){
		hasManagerClearance=clearance;
	}

	public void setLoggedInStatus(boolean status){
		loggedIn=status;
	}


}
```

7.1.4 Guest

```
import java.util.ArrayList;

public class Guest {

	private String name;
	private String address;
	private String email;
	private int mobile;
	private ArrayList<Double> bookingReferences=new ArrayList<Double>();   //
**************
	private String type;

	private Boolean payNow;
	private Boolean hasKey;
	private double amountPaidInTotal;
	private String stayDetails;
	private int pin;
	private String paymentChoices;
	private Double cashReceived;


	public Guest(String name, String address, String email, int mobile, String
type, int pin,boolean payNow) {

		this.name = name;
		this.address = address;
		this.email = email;
		this.mobile = mobile;

		if(type.equals("Registered") || type.equals("Not Registered")){
		this.type = type;
		}
		else{
			this.type="Not Registered"; //default
		}

		this.pin = pin;
		this.payNow=payNow;
		hasKey=false;
		amountPaidInTotal=0.0;

	}

	public Boolean getHasKey() {
		return hasKey;
	}

	public String getGuestType() {
		return type;
	}

	public String getGuestDetails() {
		return "Name: "+name+",Address: "+address+",Email: "+email+",Mobile:
"+mobile;
	}

	public void addBookingReference(double ref) {
```

```
			if(bookingReferences.contains(ref)){
				System.out.println("Booking Reference Already Included");
			}
			else{
			bookingReferences.add(ref);
			}
		}

		public boolean ifPayNow() {
			return payNow;
		}

		public ArrayList<Double> getBookingReferences() {
			return bookingReferences;
		}

		public void setHasKey(Boolean hasKey) {
			this.hasKey = hasKey;
		}

		public void updateAmountPaid(double amount) {
			setAmountPaidInTotal(getAmountPaidInTotal() + amount);
		}

		public void removeBooking(double booking) {
			bookingReferences.remove(booking);
		}

		public String getStayDetails() {
			return stayDetails;
		}


		public int getPin() {
			return pin;
		}


		public double getAmountPaidInTotal() {
			return amountPaidInTotal;
		}

		public void setAmountPaidInTotal(double amountPaidInTotal) {
			this.amountPaidInTotal = amountPaidInTotal;
		}


}
```

7.1.5 Manager

```
import java.util.ArrayList;

public class Manager extends Employee {

	public Manager(int employeeId, String name, String address, String email,
String currentBranch,
			boolean hasManagerClearance, String password) {
		super(employeeId, name, address, email,  currentBranch, password);
	this.setManagerClearance(true);
	}

	public void changeRoomPrice(ArrayList<Room> rooms, double price){
		// pseudo : manager enters type in system and system returns room
objects for that type
		// for this implementation I will just pass in the list of rooms of
that type as a parameter, as no system driver and database implemented
		// system will prompt for price input, for this i will pass it as
parameter

		System.out.println(rooms.get(0).getTypeDetails());  // in full
system this would be printed on system screen

			// all rooms in list have same type and price, simply print
this

		String nums="";  // print all room number details for rooms involved
in change price

		for (Room room:rooms){
			nums+=room.getRoomDetails()+",\n";
		}
		System.out.println("Rooms Involved: \n"+nums);

		for(Room room:rooms){
		 room.changeRoomPrice(price);
		}
		System.out.println("Prices Changed!\nNew Details:
"+rooms.get(0).getTypeDetails());
		nums="";
		for (Room room:rooms){
			nums+=room.getRoomDetails()+",\n";
		}
		System.out.println("Rooms Changed: \n"+nums);

	}


}
```

7.1.6 Payment

```
import java.util.Random;

public class Payment {

	private Boolean isCash;
	private String paymentTime;
	private Boolean paymentVerififed;
	private double amount;
	private int paymentId;
	private double change;

	private int  createPayment(boolean isCash,String paymentTime,double amount
) {
		this.isCash =isCash;
		this.paymentTime=paymentTime;
		this.amount=amount;
		Random rand=new Random();    // get random num for payment id
(system should ensure it is not already in database)
		this.paymentId= rand.nextInt(100000);
		this.paymentVerififed=true;
		return paymentId;
	}

	public void payByCard(boolean isCash) {
		//payment is made by card
		//can't be implemented in the system, as a seperate card system is
used.
	}

	public double getChangeDue() {
		return change;
	}

}
```

7.1.7 Receptionist

```
import java.util.ArrayList;

public class Receptionist extends Employee{

	public Receptionist(int employeeId, String name, String address, String
email, String currentBranch,
			boolean hasManagerClearance, String password) {
		super(employeeId, name, address, email,  currentBranch, password);

	}

	public Booking makeBooking(Guest guest,String date, int duration, String
email, double totalCost, String roomType,ArrayList<Room> allRooms){
		// receptionist would get details in person from guest, and
determine if they are registered
		// for this code, i pass the guest object in, this would be created
in system when details are input
		// allRooms only passed as parameter for testing as no system
database implemented
		// there would be no parameters in real system

		String type=guest.getGuestType();
		String details=guest.getGuestDetails();
		System.out.println("Type: "+type+", Details"+details);

		if(type.equals("Registered")){ // only want to create booking if
registration into database by system was successful

		ArrayList<Room> roomsAvail=getAvailableRooms(date,allRooms);
		ArrayList<Room> roomsChosen=new ArrayList<Room>();
		for(Room room: roomsAvail){
			if (room.getType().equals(roomType)){
				roomsChosen.add(room);
				break;
			}
		}

		Booking booking=new
Booking(date,duration,email,totalCost,roomsChosen);
		guest.addBookingReference(booking.getReferenceID());

		if (!guest.ifPayNow()){
			booking.setBookingPayOnCheckout(true);
		}
		else{
			booking.setBookingPayOnCheckout(false);
		}

		System.out.println(booking.getBookingDetails());
		return booking;
		}
		else{
			System.out.println("System could not register guest
previously! Booking Incomplete");
		}
		return null;
	}
```

```
	public void cancelBooking(Guest guest, Booking booking){
		// reference given by guest which system uses to search database for
booking
		// For this test code implementation i pass guest and booking as
parameters

		// system would search bookings in database for booking with
reference,
		// for this code i pass it in as parameter
		System.out.println(booking.getBookingDetails());
		double paid= booking.getAmountPaid();
		guest.updateAmountPaid(-paid);  // in real life the receptionist
gives the guest back money
		booking.removeRooms();

		guest.removeBooking(booking.getReferenceID());
	}


	public ArrayList<Room> getAvailableRooms(String date, ArrayList<Room>
rooms){
		ArrayList<Room> roomsAvail=new ArrayList<Room>();

		// Pseudo: system called to return rooms with that date available
		// for each room object , it will loop through unavailable dates,
and return the rooms in list that dont contain the chosen date as unavailable
		// for the testing of this code, all rooms are passed through as
parameter and the search is done here

		for(Room room:rooms){    // this would be done by system using
database, for test purpose i implement it here
			if(room.checkIfAvailable(date)){{
				roomsAvail.add(room);
			}
		}
		}

		return roomsAvail;

	}

	public void checkGuestIn(Booking booking,Guest guest){
		// system would find booking from ref number provided by guest, for
this the booking and guest are passed as parameters

		System.out.println(booking.getBookingDetails());
		for(Room room:booking.getRooms()){
			if(room.checkIfOccupied()){
				System.out.println("System error - room already
occupied!");
			}
			else{
			room.setIsOccupied(true);
			guest.setHasKey(true);
			}
		}
```

```
	}

	public void checkGuestOut(Booking booking,Guest guest){
		// system would find booking and room from key card reader, for this
the booking and guest are passed as parameters
		// in real system they are found in database

		guest.setHasKey(false);
		System.out.println(guest.getGuestDetails());

		boolean undamaged=true;
		for(Room room:booking.getRooms()){   // if any of the rooms are
damaged
			if(!room.checkifUndamaged()){
				undamaged=false;
				break;
			}
		}

		if(undamaged){
			guest.updateAmountPaid(- booking.getDeposit());
		}

		for(Room room:booking.getRooms()){   // set all rooms to unoccupied
and dirty
		room.setCleanStatus(false);
		room.setIsOccupied(false);
		}

	}
}
```

7.1.8 Room

```
import java.util.ArrayList;

public class Room {

	 private int roomFloorNumber;
	 private int roomNumber;
	 private String cleanedType;
	 private Boolean isCleaned;
	 private ArrayList<String> unavailableDates;
	 private String roomSize;
	 private Boolean hasBalcony;
	 private double price;
	 private double capacity;
	private Boolean isDamaged;
	 private String type;
	private Boolean isOccupied;




	public Room(int roomFloorNumber, int roomNumber, String roomSize, Boolean
hasBalcony, double price, double capacity,
			String type) {
		super();
		this.roomFloorNumber = roomFloorNumber;
		this.roomNumber = roomNumber;
		this.roomSize = roomSize;
		this.hasBalcony = hasBalcony;

		if(price>0){
		this.price = price;
		}
		else{
			this.price=50.00; //default
		}
		this.capacity = capacity;
		this.type = type;

		unavailableDates=new ArrayList<String>();
		isDamaged=false;
		isOccupied=false;
	}


	public boolean checkIfOccupied() {
		return isOccupied;
	}


	public void setIsOccupied(boolean occupied) {
		isOccupied=occupied; // sets it to occupied

	}
	public boolean checkifUndamaged() {
	   return !isDamaged;
```

```
	}

	public double getPrice() {
		return price;
	}


	public void setIsDamaged(Boolean isDamaged) {
		this.isDamaged = isDamaged;
	}


	public boolean checkIfAvailable(String date) {  // wants to go through all
// unavailable dates, if date chosen isnt there then return true
	if (unavailableDates.contains(date)){
		return false;
	}
	else{
		return true;
	}
	}

	public void setCleanStatus(boolean cleaned) {
		isCleaned=cleaned;
	}


	public String getType() {
		return type;
	}


	public String getCleanedType() {
		return cleanedType;
	}


	public ArrayList<String> getUnavailableDates() {
		return unavailableDates;
	}


	public void addBookedDate(String dateOfBooking) {
		unavailableDates.add(dateOfBooking);
	}

	public void clean(String type) {
		setCleanStatus(true);
		setCleanedType(type);
	}

	private void setCleanedType(String type) {
		this.cleanedType=type;
	}

	public boolean getCleanDetails() {
		return isCleaned;
	}
```

```
	public String getRoomDetails() {
		return "Floor: "+roomFloorNumber +",Room Number: "+
roomNumber+",Price: "+price+",Capacity: " +capacity +",Size: "+ roomSize+ ",
Balcony?: "+hasBalcony;
	}

	public void changeRoomPrice(double price) {
		if(price>0){
		this.price = price;
	}
	else{
		this.price=50.00; //default
	}
	}

	public String getTypeDetails() {
		return "Type: "+type+", Price: "+price;
	}

	public void removeBookedDate(String dateOfBooking) {
		unavailableDates.remove(dateOfBooking);
	}

}
```

7.2 Test Classes

7.2.1 CleanerTest

```
import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class CleanerTest {
    Cleaner c1;
       Room room1;
       Booking b1;

       @Before
       public void setup(){
              c1=new Cleaner(0, "name1", "address1", "email1", "branch1", false,
"pass1");

              room1=new Room(1, 101, "50x50", false, 1, 20.00,"Single");


       }

       @Test
       public void testCleanRoomOccupiedAndDirty(){
              room1.setIsOccupied(true);
              room1.setCleanStatus(false);
              c1.cleanRoom(room1);

              assertTrue(room1.getCleanDetails());
              assertEquals("daily",room1.getCleanedType());
       }

       @Test
       public void testCleanRoomNotOccupiedAndDirty(){
              room1.setIsOccupied(false);
              room1.setCleanStatus(false);
              c1.cleanRoom(room1);

              assertTrue(room1.getCleanDetails());
              assertEquals("deep clean",room1.getCleanedType());
       }
}
```

7.2.2 ConstructorsTest

```
import static org.junit.Assert.*;

import java.util.ArrayList;

import org.junit.Before;
import org.junit.Test;

public class ConstructorsTest {

	Booking b1;
	Booking b2;
	Room room1;
	Room room2;
	Room room3;
	Room room4;
	ArrayList<Room> rooms1=new ArrayList<Room>();
	ArrayList<Room> rooms2=new ArrayList<Room>();


	@Before
	public void setup(){
		room1=new Room(1, 101, "50x50", false, 20.00,1,"Single");
		room2=new Room(2, 201, "50x50", false,  30.00,3,"Double");
		room3=new Room(3, 301, "50x50", true, 40.00,1,"Single");
		room4=new Room(4, 401, "50x50", true, -90.00,4,"Family");

		rooms1.add(room1);
		b1= new Booking("01/01/2018", 1, "John@gmail.com", 20.00,rooms1 );

		rooms2.add(room1);
		rooms2.add(room2);
		b2= new Booking("01/01/2019", 1, "Joan@gmail.com", 50.00,rooms2 );
	}


	// implemented some tests for Booking and Room constructors as guideline,
same would be done for other class constructors
	@Test
	public void testBookingConstructorSetDate(){
		assertEquals("01/01/2018",b1.getDate());
	}

	@Test
	public void testBookingConstructorSetDuration(){
		assertEquals(1,b1.getDuration());
	}

	@Test
	public void testBookingConstructorSetEmail(){
		assertEquals("John@gmail.com",b1.getGuestEmail());
	}

	@Test
	public void testBookingConstructorSetPrice(){
		assertEquals(20.00,b1.getCost(),2);
	}
```

```
@Test
public void testBookingConstructorSetRooms(){
	assertEquals(rooms1,b1.getRooms());
}

@Test
public void testRoomConstructorOccupied(){
	assertFalse(room1.checkIfOccupied());
}
@Test
public void testRoomConstructorUnDamaged(){
	assertTrue(room1.checkifUndamaged());
}

@Test
public void testRoomConstructorPrice(){
	assertEquals(20.00,room1.getPrice(),2);
}
@Test
public void testRoomConstructorInvalidPrice(){
	assertEquals(50.00,room4.getPrice(),2);
}

@Test
public void testRoomConstructorType(){
	assertEquals("Single",room1.getType());
}

@Test
public void testRoomSetIsOccupied(){
	room1.setIsOccupied(true);
	assertTrue(room1.checkIfOccupied());
}

@Test
public void testRoomSetIsDamaged(){
	room1.setIsDamaged(true);
	assertFalse(room1.checkifUndamaged());
}

@Test
public void testIfAvailable(){
	room1.addBookedDate("01/01/2013");
	assertFalse(room1.checkIfAvailable("01/01/2013"));
}

@Test
public void testAddBookedDate(){
	room1.addBookedDate("01/01/2013");
	assertEquals(3,room1.getUnavailableDates().size());
}

@Test
public void testClean(){
	room1.clean("daily");
	assertTrue(room1.getCleanDetails());
	assertEquals("daily",room1.getCleanedType());
}
```

```
	@Test
	public void testRoomChangeInvalidPrice(){
		room1.changeRoomPrice(-2);
		assertEquals(50.00,room1.getPrice(),2);
	}

	@Test
	public void testRemoveBookedDate(){
		assertTrue(room1.getUnavailableDates().contains("01/01/2018"));
		room1.removeBookedDate("01/01/2018");
		assertFalse(room1.getUnavailableDates().contains("01/01/2018"));
	}
}
```

7.2.3 GuestTest

```
import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class GuestTest {

	Guest g1;
	Guest g2;

	@Before
	public void setup(){
		g1=new Guest("name1", "address1", "email1", 98765, "Registered",
1234, true);
		g2=new Guest("name2", "address2", "email2", 92234, "Invalid", 2234,
true);
	}

	@Test
	public void testConstructorKey(){
		assertFalse(g1.getHasKey());
	}
	@Test
	public void testConstructorType(){
		assertEquals("Registered",g1.getGuestType());
	}
	@Test
	public void testConstructorInvalidType(){
		assertEquals("Not Registered",g2.getGuestType());
	}

	@Test
	public void testDetails(){
		assertEquals("Name: name1,Address: address1,Email: email1,Mobile:
98765",g1.getGuestDetails());
	}

	@Test
	public void testAddBookingReference(){
		assertEquals(0,g1.getBookingReferences().size());
		g1.addBookingReference(1234);
		assertEquals(1,g1.getBookingReferences().size());
	}

	@Test
	public void testAddBookingDuplicateReference(){
		g1.addBookingReference(1234);
		assertEquals(1,g1.getBookingReferences().size());
		g1.addBookingReference(1234);
		assertEquals(1,g1.getBookingReferences().size());
	}

	@Test
	public void testConstructorIfPayNow(){
		assertTrue(g1.ifPayNow());
	}
```

```
	@Test
	public void testGetBookingReferences(){
		g1.addBookingReference(1234);
		assertEquals(1,g1.getBookingReferences().size());
		assertEquals(1234,g1.getBookingReferences().get(0),2);
	}

	@Test
	public void testSetHasKey(){
		g1.setHasKey(true);
		assertTrue(g1.getHasKey());
	}

	@Test
	public void testUpdateAmountPositive(){
		g1.setAmountPaidInTotal(100);
		g1.updateAmountPaid(20);
		assertEquals(120,g1.getAmountPaidInTotal(),2);
	}
	@Test
	public void testUpdateAmountNegative(){
		g1.setAmountPaidInTotal(100);
		g1.updateAmountPaid(-20);
		assertEquals(80,g1.getAmountPaidInTotal(),2);
	}

	@Test
	public void testRemoveBooking(){
		g1.addBookingReference(1234.0);
		assertTrue(g1.getBookingReferences().contains(1234.0));
		g1.removeBooking(1234.0);
		assertFalse(g1.getBookingReferences().contains(1234.0));

	}

	@Test
	public void testConstructorGetPin(){
		assertEquals(1234,g1.getPin());
	}

	@Test
	public void testConstructorGetAmountPaid(){
		assertEquals(0,g1.getAmountPaidInTotal(),2);
	}

	@Test
	public void testSetAmountPaid(){
		g1.setAmountPaidInTotal(10.00);
		assertEquals(10.00,g1.getAmountPaidInTotal(),2);
	}
}
```

7.2.4 ManagerTest

```
import static org.junit.Assert.*;

import java.util.ArrayList;

import org.junit.Before;
import org.junit.Test;

public class ManagerTest {

	Manager m1;
	Room room1;
	Room room2;
	Room room3;
	Room room4;
	Booking b1;
	ArrayList<Room> rooms1=new ArrayList<Room>();

	@Before
	public void setup(){
		m1=new Manager(0, "name1", "address1", "email1", "branch1", true,
"pass1");

		room1=new Room(1, 101, "50x50", false, 20.00, 1,"Single");
		room2=new Room(2, 201, "50x50", false, 30.00,3,"Double");
		room3=new Room(3, 301, "50x50", true, 40.00,1,"Single");
		room4=new Room(4, 401, "50x50", true, 50.00,4,"Family");

		rooms1.add(room1);

	}

	@Test
	public void testChangeRoomPrice(){
		m1.changeRoomPrice(rooms1, 24.00);
		assertEquals(24.00,room1.getPrice(),2);
	}
	@Test
	public void testChangeRoomPriceMultiple(){
		rooms1.add(room3);
		m1.changeRoomPrice(rooms1, 24.00);
		assertEquals(24.00,room1.getPrice(),2);
		assertEquals(24.00,room3.getPrice(),2);
	}
}
```

7.2.5 ReceptionistTest

```
import static org.junit.Assert.*;

import java.util.ArrayList;

import org.junit.Before;
import org.junit.Test;

public class ReceptionistTest {

	Receptionist r1;
	Guest g1;
	Room room1;
	Room room2;
	Room room3;
	Room room4;
	Booking b1;
	ArrayList<Room> rooms1=new ArrayList<Room>();
	ArrayList<Room> rooms2=new ArrayList<Room>();

	@Before
	public void setup(){

		g1= new Guest("name1", "address1", "email1", 34567, "Registered",
0000,true);

		room1=new Room(1, 101, "50x50", false, 20.00,1,"Single");
		room2=new Room(2, 201, "50x50", false,  30.00,3,"Double");
		room3=new Room(3, 301, "50x50", true, 40.00,1,"Single");
		room4=new Room(4, 401, "50x50", true, 50.00,4,"Family");

		rooms1.add(room1);
		rooms1.add(room2);
		rooms1.add(room3);

		 r1=new Receptionist(1, "name1", "address1", "email1", "branch1",
false, "password1");
		 b1=r1.makeBooking(g1, "01/01/2019", 1, "email1", 29.99, "Single",
rooms1);
	}

	@Test
	public void testMakeBookingDateAddedToRoom(){
		assertFalse(room1.checkIfAvailable("01/01/2019"));
	}

	@Test
	public void testMakeBookingRefAddedToGuest(){
		assertEquals(g1.getBookingReferences().size(),1);
	}

	@Test
	public void testMakeBookingPayNowBooking(){
		assertEquals(b1.isPayOnCheckout(),false);
	}

	@Test
	public void testCancelBookingAmountPaidReturned(){
```

```
		b1.updateBalanceDue(19.99);
		g1.updateAmountPaid(19.99);
		r1.cancelBooking(g1, b1);

		assertEquals(0,g1.getAmountPaidInTotal(),2);
	}

	@Test
	public void testCancelBookingGuestRemovedBookingRef(){
		r1.cancelBooking(g1, b1);


	assertFalse(g1.getBookingReferences().contains(b1.getReferenceID()));
	}

	@Test
	public void testCancelBookingRoomAvailableAgain(){
		r1.cancelBooking(g1, b1);

		assertTrue(room1.checkIfAvailable("01/01/2019"));
	}

	@Test
	public void testGetAvailableRooms(){
		ArrayList<Room> roomsReturned=r1.getAvailableRooms("01/01/2019",
rooms1);

		assertEquals(room2,roomsReturned.get(0));
		assertEquals(2,roomsReturned.size());
		}


	@Test
	public void testCheckGuestInRoomNotOccupied(){
		r1.checkGuestIn(b1, g1);
		assertTrue(b1.getRooms().get(0).checkIfOccupied());
		assertTrue(g1.getHasKey());
	}

	@Test
	public void testCheckGuestInRoomOccupiedAlready(){
		b1.getRooms().get(0).setIsOccupied(true);
		r1.checkGuestIn(b1, g1);
		assertFalse(g1.getHasKey());
	}


	@Test
	public void testCheckGuestOutKeyTaken(){
		r1.checkGuestOut(b1, g1);
		assertFalse(g1.getHasKey());
	}

	@Test
	public void testCheckGuestOutNoRoomsDamaged(){
		g1.setAmountPaidInTotal(100);
		b1.setDeposit(10.00);
		double origAmount=g1.getAmountPaidInTotal();
		r1.checkGuestOut(b1, g1);
```

```
		assertEquals(origAmount-10,g1.getAmountPaidInTotal(),2);
	}

	@Test
	public void testCheckGuestOutOneRoomDamaged(){
		g1.setAmountPaidInTotal(100);
		b1.setDeposit(10.00);
		room1.setIsDamaged(true);
		double origAmount=g1.getAmountPaidInTotal();
		r1.checkGuestOut(b1, g1);

		assertEquals(origAmount,g1.getAmountPaidInTotal(),2);
	}

	@Test
	public void testCheckGuestOutRoomCleanStatus(){
		r1.checkGuestOut(b1, g1);

		assertFalse(room1.getCleanDetails());
	}
	@Test
	public void testCheckGuestOutRoomOccupiedStatus(){
		r1.checkGuestOut(b1, g1);

		assertFalse(room1.checkIfOccupied());
	}
}
```