

|  |  |
| --- | --- |
|  | Use Case Design for Hotel Room Reservation |
|  |  |
|  | Max Larkin & David Conway  Software Engineering  1/30/19 |

Table of Contents

[Overview 2](#_Toc6335141)

[Use Case Diagram 3](#_Toc6335142)

[Use Cases 4](#_Toc6335143)

[1. Sign up 4](#_Toc6335144)

[2. Sign In 4](#_Toc6335145)

[3. Check Availability 5](#_Toc6335146)

[4. Book Room 5](#_Toc6335147)

[5. Check Balance 6](#_Toc6335148)

[6. View Service Menu 6](#_Toc6335149)

[7. Check In 6](#_Toc6335150)

[8. Check Out 6](#_Toc6335151)

[9. Cancel Booking 6](#_Toc6335152)

[10. Sign Out 6](#_Toc6335153)

[Class Diagram 6](#_Toc6335154)

Hotel Room Availability System

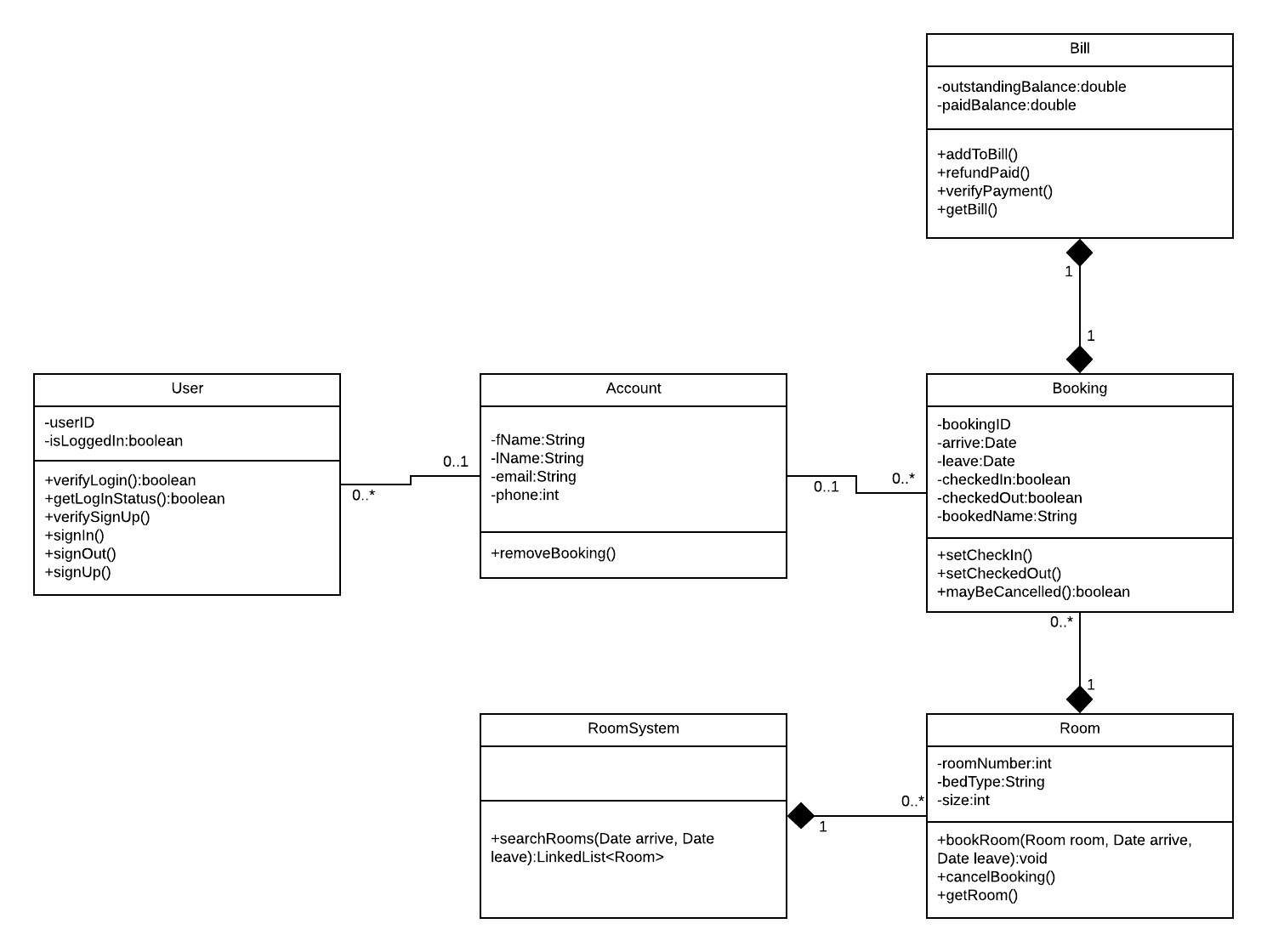
# Overview

For the purposes of demonstration, LocalHotel is an app/website designed for a single hotel which guests may use to streamline their hotel experience, allowing them to autonomously access all features of the hotel. The app may be used to book rooms, check in and out, sign up to the hotel as a repeat customer and even access room service.

For an unregistered user, basic functionality exists such as searching available rooms and booking a room, however the full amount must be paid upon booking. Checking in/out and cancellation must be done through a representative of the hotel. (This system will focus solely on functionality through the app/website however.)

As a registered user, guests may enjoy benefits such as no deposit booking and no fee cancellation. All features of the hotel may be accessed through the app while registered and most requests such as cancellation may be completed entirely in app. Features include checking in and out, which will provide a scannable QR code along with the room number to access your room.

# Class Diagram



# Use Case Diagram



# Use Cases

## Sign up

**Actor**: Unregistered User

**Trigger**: User attempts to sign up

**Post-conditions:**

•User has successfully made an account.

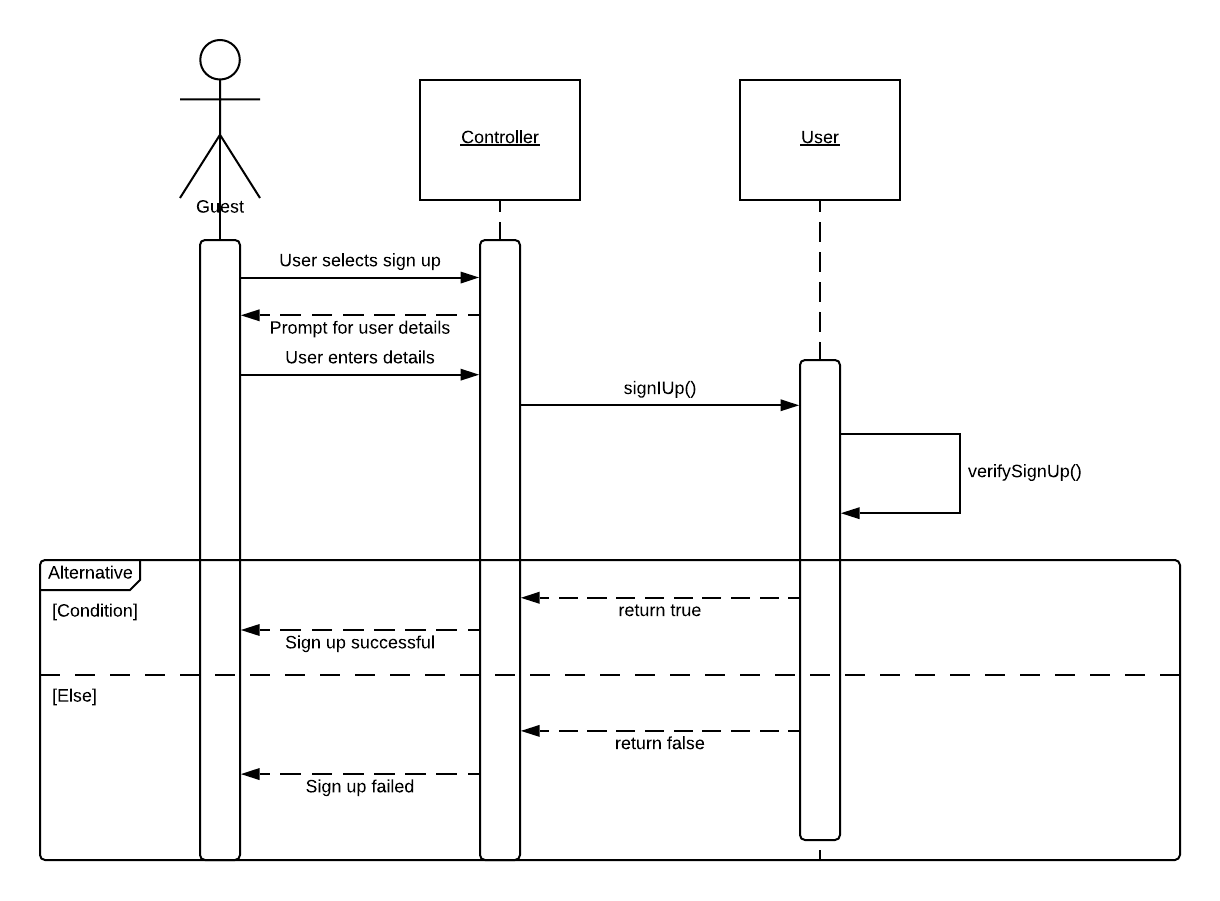
**Main Success Scenario**

1. User navigates to the sign-up button via app/website.
2. User is prompted to enter details.
3. User fills out the sign-up form.
4. Account is created.

**Failure Scenarios**

FS1: Invalid details for sign up

1. At step 3 in the main scenario, use enters invalid details.
2. Account creation fails.
3. User is notified.

**Sequence Diagram**

## Sign In

**Actor**: Registered User

**Trigger**: User attempts to sign in

**Post-conditions:**

•User has successfully signed into their account.

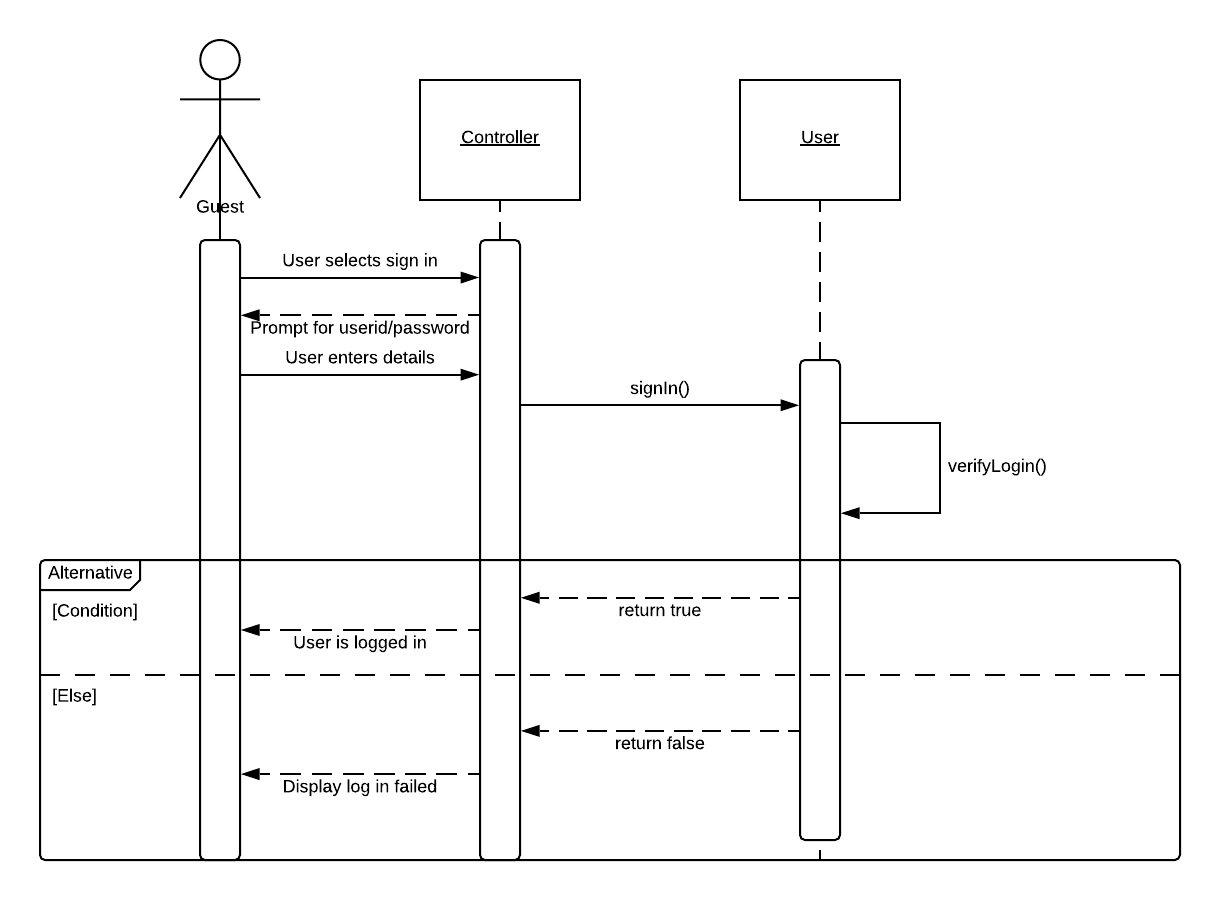
**Main Success Scenario**

1. User navigates to the sign-in button via app/website.
2. User is prompted to enter user ID and password.
3. Login information is verified by the system.
4. User is logged in.

**Failure Scenarios**

FS1: Login Verification Failure

1. At step 3, the login verification system fails.
2. The system returns a login error message.
3. The system returns the user to the main scenario at step 2.



## Check Availability

**Actor**: App/Website User

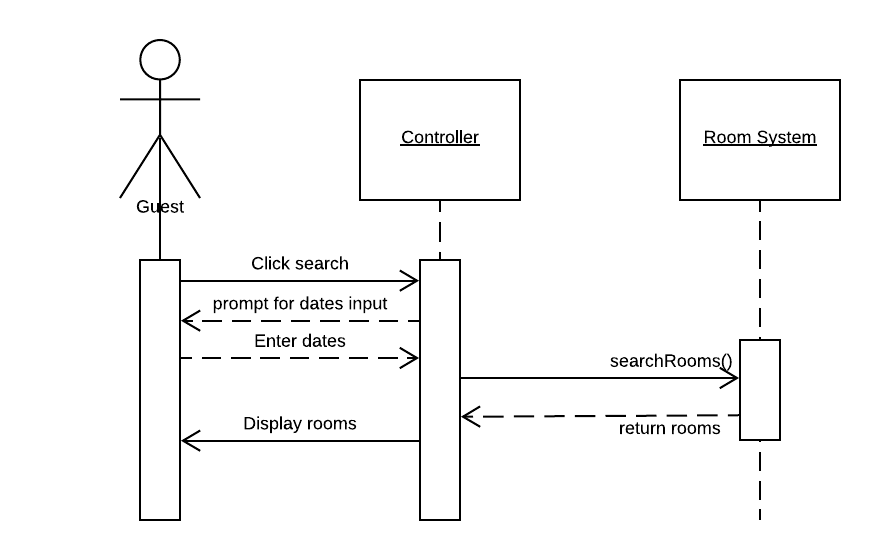
**Trigger**: User navigates to the room search button.

**Post-conditions:**

• User is displayed all available rooms for the desired dates.

**Main Success Scenario**

1. User clicks on the room search button.
2. User is prompted for an arrive and leave date.
3. User fills in form with two dates.
4. The system checks the bookings of all rooms to ensure they are free for the duration desired.
5. The app displays the available rooms.

**Sequence Diagram**

## Book Room

**Actor**: App/Website User

**Trigger**: User attempts to sign in

**Pre-conditions:**

* User has checked for room availability

**Post-conditions:**

* User has successfully booked a room.

**Main Success Scenario**

1. The user selects the preferred room from the available rooms.
2. The system verifies the users logged in status.
3. A booking is created with the users account details.

**Alternate Scenarios**

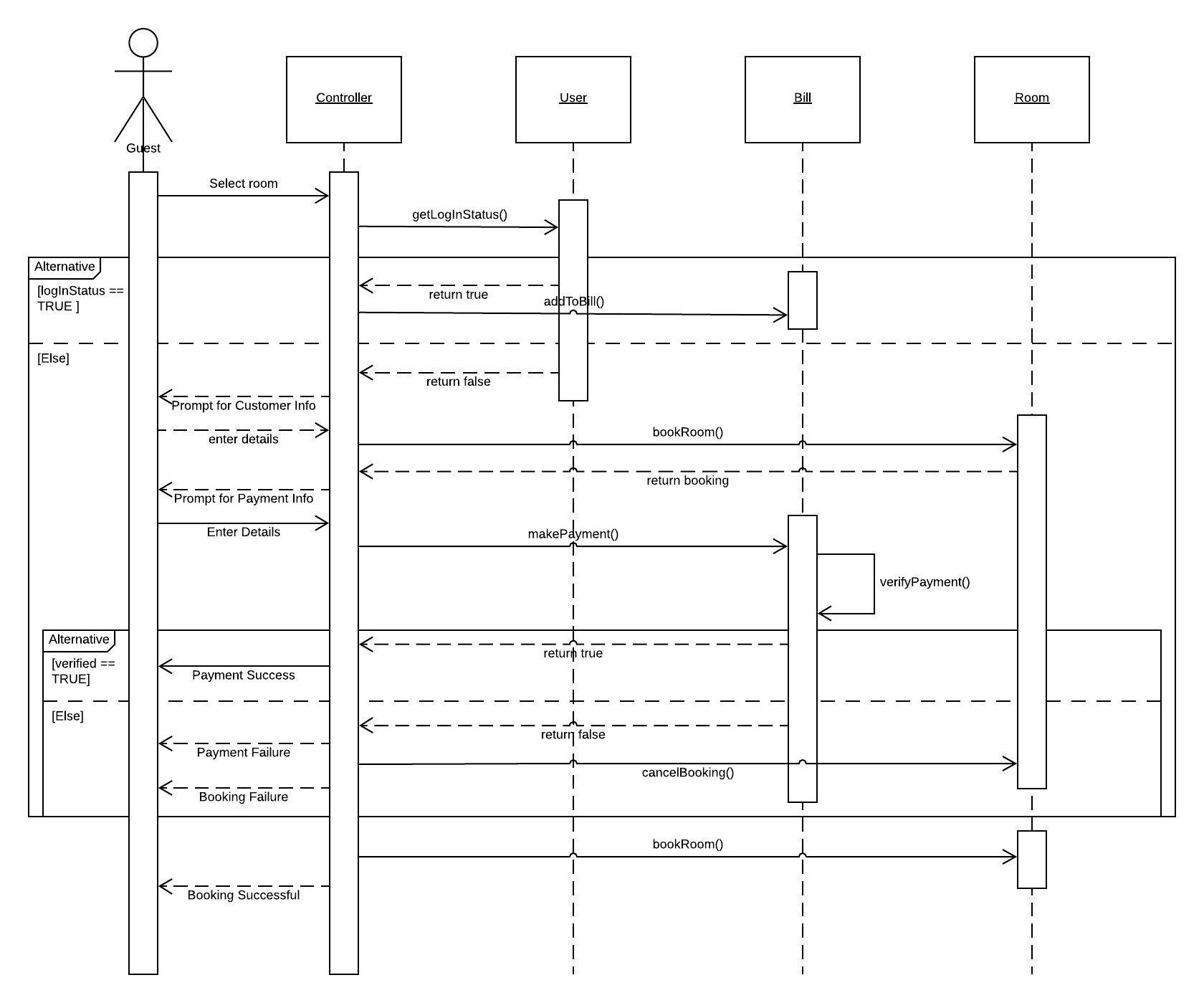
AS1: User is not logged in.

1. At step 2, the user is not logged in.
2. The system prompts the user for payment information and customer details.
3. The system creates a booking under the details.
4. The system verifies the payment information.
5. The transaction completes.
6. System notifies user of successful booking.

**Failure Scenarios**

FS1: Payment fails

1. At step 4 in AS1, the payment information is invalid, or the provider cannot process the transaction.
2. User is notified of payment failure.
3. System cancels the booking.

**Sequence Diagram**

## Check Balance

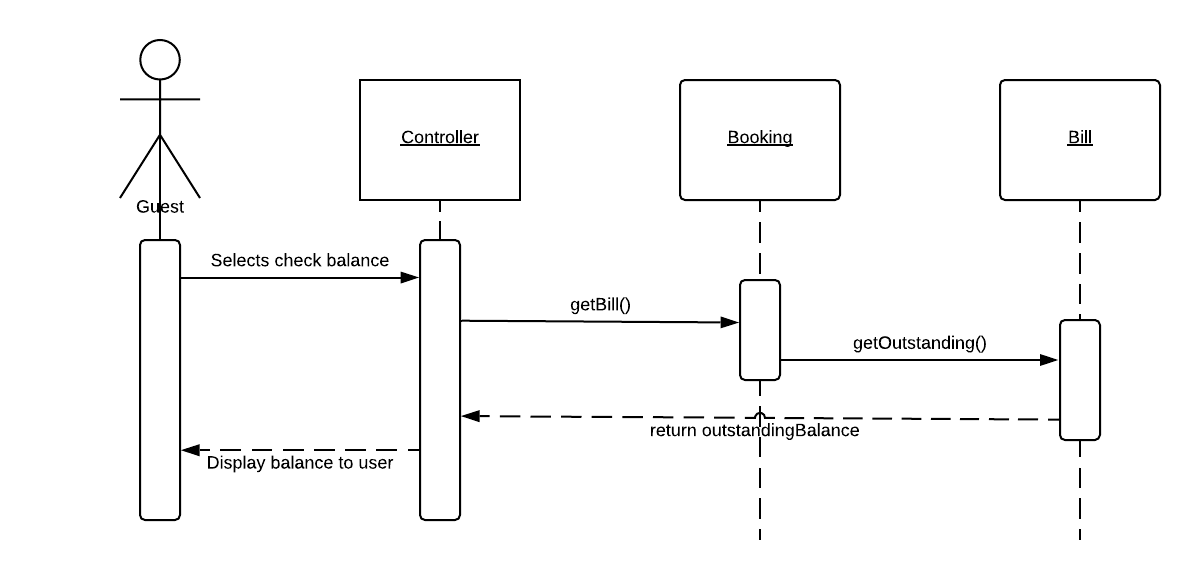
**Actor**: Registered User

**Trigger**: User selects check balance.

**Post-conditions:**

* User is displayed current outstanding balance.

**Main Success Scenario**

1. User selects check balance.
2. System returns the collective balance to be paid on all bookings.
3. System offers a prompt to make payment.

## Check In

**Actor**: Registered User

**Trigger**: User attempts to check in

**Pre-conditions:**

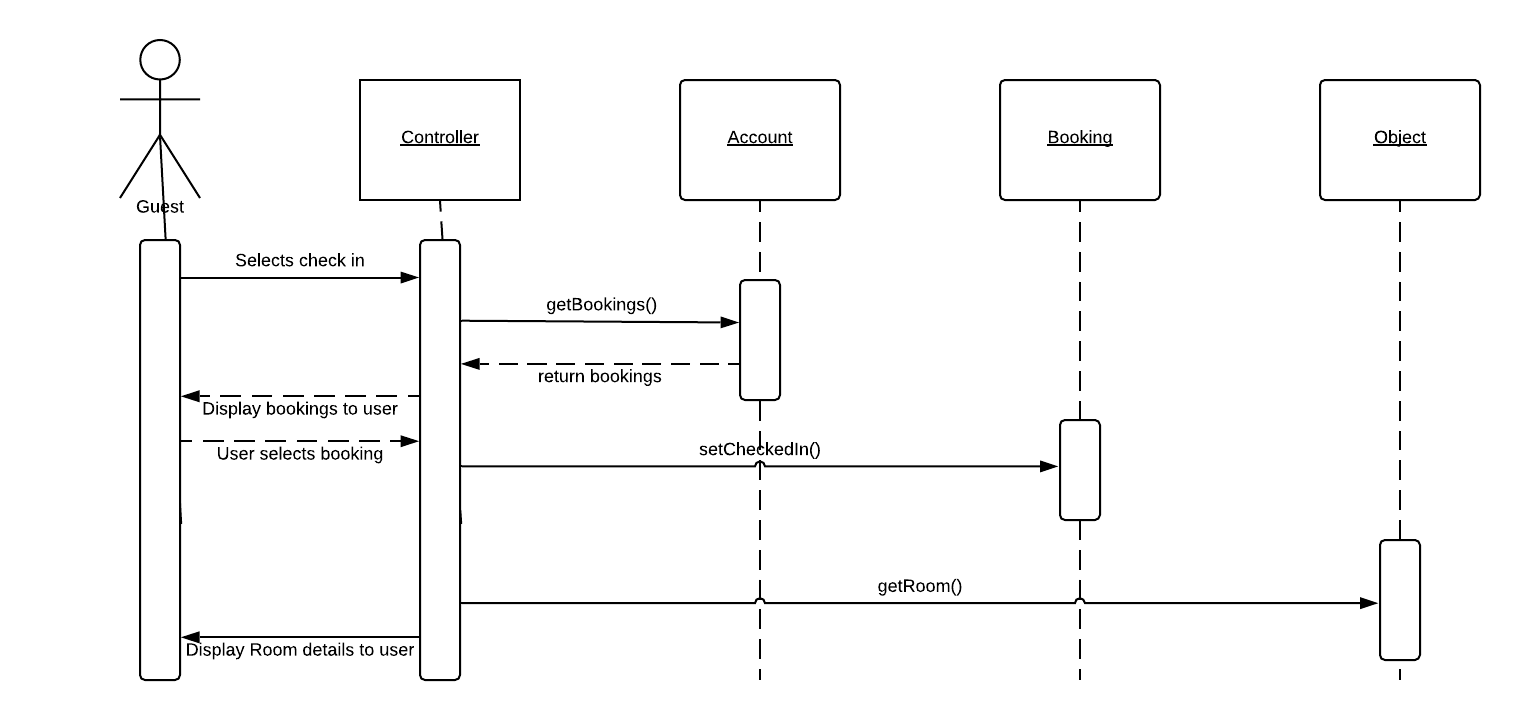
* User has a booking

**Post-conditions:**

* User has successfully checked in.

**Main Success Scenario**

1. User selects check in from home screen.
2. System returns list of booking made by the user.
3. User selects the booking to check in.
4. System checks in the booking.
5. User notified.



## Check Out

**Actor**: Registered User

**Trigger**: User attempts to check out

**Pre-conditions:**

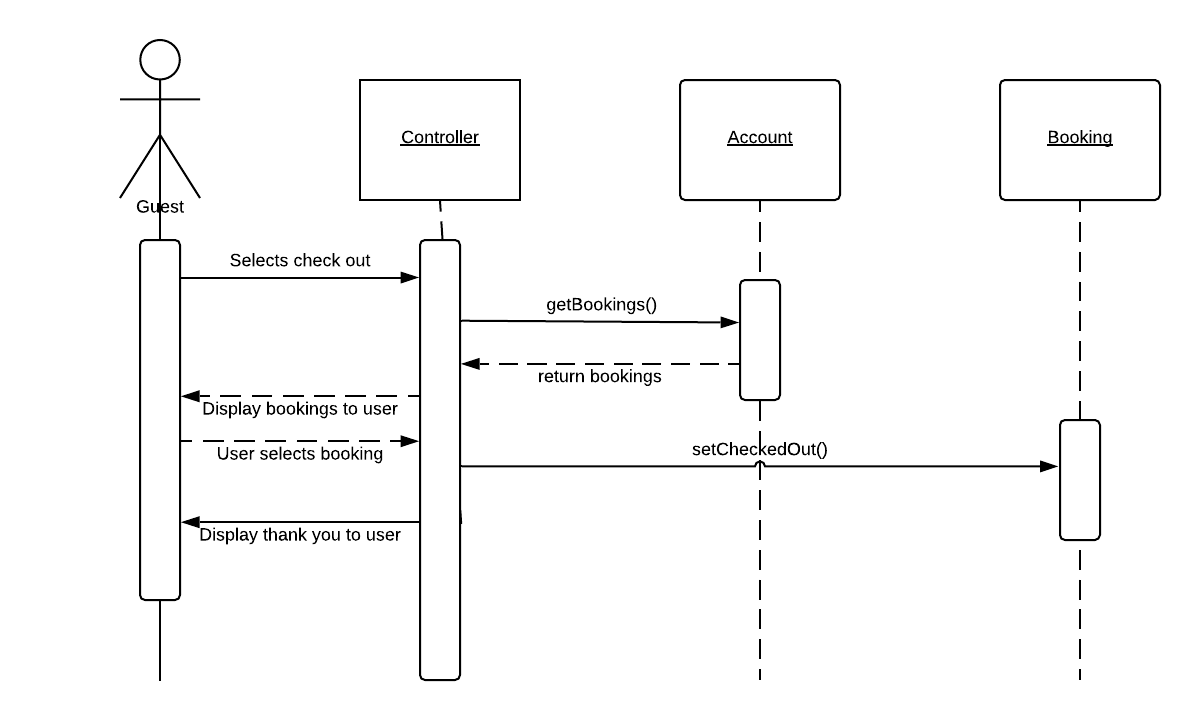
* User has a booking checked in.

**Post-conditions:**

* User has successfully checked out.

**Main Success Scenario**

1. User selects check out from home screen.
2. System returns lists of bookings to check out.
3. User selects booking.
4. System checks out the booking.
5. User notified.



## Cancel Booking

**Actor**: Registered User

**Trigger**: User selects cancel booking.

**Pre-conditions:**

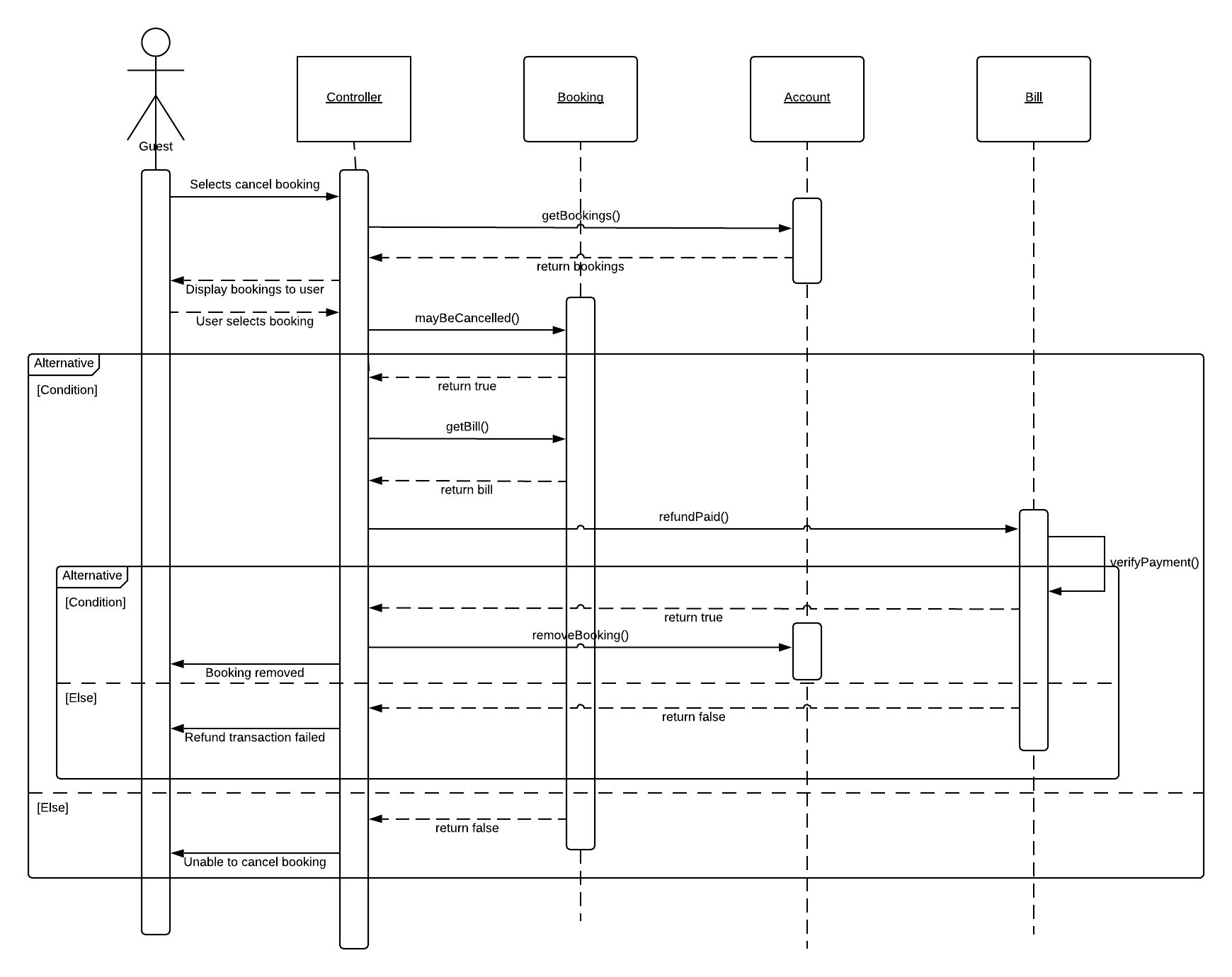
* User has booked a room.

**Post-conditions:**

* Booking has been cancelled successfully.

**Main Success Scenario**

1. System displays list of bookings.
2. User selects booking to cancel.
3. The system verifies whether this booking may be cancelled.
4. The system then attempts to refund through the customers used payment method.
5. User is refunded.
6. Booking is removed from system.



## Sign Out

**Actor**: Registered User

**Trigger**: User clicks sign out

**Pre-conditions:**

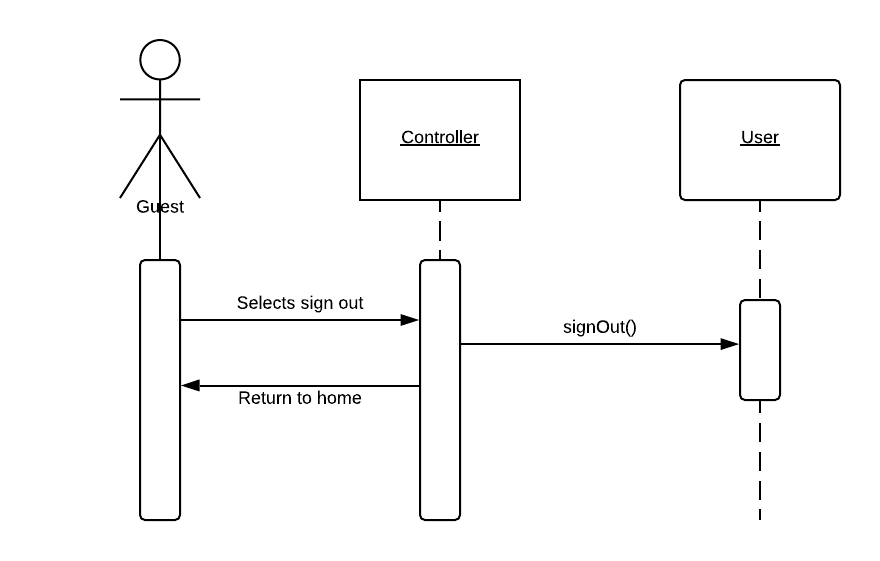
* User is signed in.

**Post-conditions:**

* User has successfully logged out.

**Main Success Scenario**

1. User selects sign out.
2. User is signed out.
3. System displays home page.



# Java Classes

## Bill

**public** **class** Bill {

**double** outstandingBalance, paidBalance;

Bill(**double** roomCost, **int** days) {

outstandingBalance = roomCost \* days;

}

**public** **void** addToBill(**double** cost) {

outstandingBalance = outstandingBalance + cost;

}

**public** **void** refundPaid() {

// Access payment servers

paidBalance = 0;

outstandingBalance = 0;

}

**public** **boolean** verifyPayment() {

// Access payment servers

**return** **true**;

}

**public** **double** getOutstanding() {

**return** outstandingBalance;

}

**public** **double** getPaidBalance() {

**return** paidBalance;

}

**public** **void** makePayment(**double** payment) {

**if** (payment > outstandingBalance) {

payment = outstandingBalance;

}

// Access pay servers

outstandingBalance = outstandingBalance - payment;

}

}

## Booking

**public** **class** Booking {

String bookedName;

Date arrive;

**int** days;

ArrayList<Date> dates = **new** ArrayList<>();

Calendar cal = Calendar.*getInstance*();

**boolean** checkedIn, checkedOut;

**public** **boolean** isCheckedIn() {

**return** checkedIn;

}

**public** **void** setCheckedIn(**boolean** checkedIn) {

**this**.checkedIn = checkedIn;

}

**public** **boolean** isCheckedOut() {

**return** checkedOut;

}

**public** **void** setCheckedOut(**boolean** checkedOut) {

**this**.checkedOut = checkedOut;

}

**public** **void** setDates(ArrayList<Date> dates) {

**this**.dates = dates;

}

Booking(Date arrive, **int** days, String bookedName) {

**this**.days = days;

**this**.arrive = arrive;

**this**.bookedName = bookedName;

dates = **new** ArrayList<>();

cal.setTime(arrive);

**for** (**int** i = 0; i < days; i++) {

dates.add(cal.getTime());

cal.add(Calendar.***DATE***, 1);

}

}

**public** ArrayList<Date> getDates() {

**return** dates;

}

**public** String getBookedName() {

**return** bookedName;

}

}

## Room

**package** application;

**import** java.util.Date;

**import** java.util.LinkedList;

**public** **class** Room {

String bedType;

**int** roomNumber, size;

**private** LinkedList<Booking> bookings = **new** LinkedList<>();

Room(String bedType, **int** size, **int** roomNumber) {

**this**.roomNumber = roomNumber;

**this**.size = size;

**this**.bedType = bedType;

}

**public** **void** addBooking(Booking booking) {

bookings.add(booking);

}

**public** **void** cancelBooking(Date date) {

**for** (**int** i = 0; i < bookings.size(); i++) {

**if** (bookings.get(i).getDates().contains(date)) {

bookings.remove(i);

}

}

}

**public** LinkedList<Booking> getBookings() {

**return** bookings;

}

}

## RoomSystem

**public** **class** RoomSystem {

**static** ArrayList<Room> *rooms* = **new** ArrayList<>();

**static** LinkedList<Room> searchRooms(LinkedList<Date> dates) {

LinkedList<Room> foundRooms = **new** LinkedList<>();

**for** (**int** i = 0; i < *rooms*.size(); i++) {

LinkedList<Booking> bookings = *rooms*.get(i).getBookings();

**for** (**int** j = 0; j < bookings.size(); j++) {

**for** (**int** k = 0; k < bookings.get(j).getDates().size(); k++) {

**if** (bookings.get(j).getDates().contains(dates.get(k))) {

**continue**;

}

foundRooms.add(*rooms*.get(i));

}

}

}

**return** foundRooms;

}

}

## User

**public** **class** User {

// Mainly interacts with server database

}

## Account

**public** **class** Account {

String fName, lName, email;

**int** phone;

Account(String fName, String lName, String email, **int** phone){

**this**.fName = fName;

**this**.lName = lName;

**this**.email = email;

**this**.phone = phone;

}

}

# JUnit Tests

## BillTest

**class** BillTest {

Bill b1;

Bill b2;

Bill b3;

@BeforeEach

**void** setUp() **throws** Exception {

b1 = **new** Bill(50, 2);

b2 = **new** Bill(50, 4);

b3 = **new** Bill(25, 4);

}

@Test

**void** testGetBalance() {

*assertEquals*(b2.getOutstanding(), 200);

}

@Test

**void** testOverPay() {

b3.makePayment(200);

*assertEquals*(b3.getOutstanding(), 0);

}

@Test

**void** testRefund() {

b3.makePayment(200);

b3.refundPaid();

*assertEquals*(b3.getPaidBalance(), 0);

}

@Test

**void** testIncreaseBill() {

b3.addToBill(200);

*assertEquals*(b3.getOutstanding(), 300);

}

}

## BookingTest

**class** BookingTest {

Booking b1;

Date arrive;

Calendar cal;

@BeforeEach

**void** setUp() **throws** Exception {

cal = Calendar.*getInstance*();

cal.set(2019, 2, 10);

arrive = cal.getTime();

b1 = **new** Booking(arrive, 5, "John Doe");

}

@Test

**void** testBookedDates() {

LinkedList<Date> dates = **new** LinkedList<>();

cal.setTime(arrive);

dates.add(arrive);

cal.add(Calendar.***DATE***, 1);

dates.add(cal.getTime());

cal.add(Calendar.***DATE***, 1);

dates.add(cal.getTime());

cal.add(Calendar.***DATE***, 1);

dates.add(cal.getTime());

cal.add(Calendar.***DATE***, 1);

dates.add(cal.getTime());

*assertEquals*(dates, b1.getDates());

}

@Test

**void** testCheckIn() {

b1.setCheckedIn(**true**);

*assertEquals*(**true**, b1.isCheckedIn());

}

@Test

**void** testCheckOut() {

b1.setCheckedOut(**true**);

*assertEquals*(**true**, b1.isCheckedOut());

}

}

## RoomTest

**class** RoomTest {

Room r1;

Booking b1;

Calendar cal;

@BeforeEach

**void** setUp() **throws** Exception {

r1 = **new** Room("Queen", 2, 10);

cal = Calendar.*getInstance*();

}

@Test

**void** testAddBooking() {

b1 = **new** Booking(cal.getTime(), 2, "James" + "Parker");

r1.addBooking(b1);

*assertTrue*(r1.getBookings().contains(b1));

}

@Test

**void** testCancelBooking() {

b1 = **new** Booking(cal.getTime(), 2, "James" + "Parker");

r1.addBooking(b1);

r1.cancelBooking(cal.getTime());

*assertFalse*(r1.getBookings().contains(b1));

}

}

## RoomSystemTest

**class** RoomSystemTest {

Room r1, r2;

Booking b1, b2;

Calendar cal;

@BeforeAll

**static** **void** setUpBeforeClass() **throws** Exception {

}

@BeforeEach

**void** setUp() **throws** Exception {

cal = Calendar.*getInstance*();

r1 = **new** Room("Queen", 2, 10);

r2 = **new** Room("King and couch", 4, 11);

b1 = **new** Booking(cal.getTime(), 5, "James Parker");

b2 = **new** Booking(cal.getTime(), 2, "Billy Bob");

r1.addBooking(b1);

r2.addBooking(b2);

}

@Test

**void** testSearchRooms() {

LinkedList<Date> dates = **new** LinkedList<>();

cal.add(Calendar.***DATE***, 4);

dates.add(cal.getTime());

LinkedList<Room> rooms = RoomSystem.*searchRooms*(dates);

*assertEquals*(rooms.size(), 0);

}

}