MARKING REPORT

**Group number: 21\_**

|  |  |
| --- | --- |
| Name | Student ID |
| 1. Lim Chen Chuen | 2300753 |
| 2. Kee Ming Xian | 2300155 |
| 3. Tiew Ri Jie | 2300782 |
| 4. Chew Sit Xiang | 2207475 |

**Marks breakdown**

Part A: Test Plan (10 marks)

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Max Mark** | **Marks Obtained** | **Remark/Comment** |
| Test objective, scope and test basis | 5 |  |  |
| Test condition, entry and exit criteria | 5 |  |  |

Part B: Test Design (20 marks)

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Max Mark** | **Marks Obtained** | **Remark/Comment** |
| Decision table | 5 |  |  |
| Appropriateness of test cases | 15 |  |  |

Part C: Java Program (application code and test code) (70 marks)

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Max Mark** | **Marks Obtained** | **Remark/Comment** |
| Setup jar file location to C:\ jar\_files | 2 |  |  |
| Source directories | 3 |  |  |
| Appropriate used of assertsXXX methods. | 10 |  |  |
| Using parameterised tests correctly | 10 |  |  |
| Invalid values are checked for in implemented code, and tests for invalid values are performed. | 10 |  |  |
| Use of mocks or stubs for testing. | 10 |  |  |
| Combining test cases into test suites | 5 |  |  |
| Setting up some tests so that test values are read from a text file instead of hardcoding into test code | 10 |  |  |
| Perform integration testing after unit tests have been completed | 10 |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A: | B: | C: | Total:  /100 | /20 |

**Application Code**

**Booking Class:**

package my.edu.utar;

import java.util.\*;

public class Booking {

    private Room room;

    private WaitingList waitingList;

    private Printer printer;

    public Booking(Room room, WaitingList waitingList, Printer printer) {

        this.room = room;

        this.waitingList = waitingList;

        this.printer = printer;

    }

    public void setBooking(User user, int numOfRooms) {

        if (user.getMemberType().equals("VIP")) {

            bookVipMember(user, numOfRooms);

        } else if (user.getMemberType().equals("Member")) {

            bookNormalMember(user, numOfRooms);

        } else {

            bookNonMember(user,numOfRooms);

        }

    }

    public void bookVipMember(User user, int numOfRooms) {

        if (numOfRooms > 3 || numOfRooms <= 0 ) {

            throw new IllegalArgumentException("VIP members can book up to 3 rooms only and negative value is invalid");

        }

        for (int i = 0; i < numOfRooms; i++) {

            boolean roomBooked = false;  // Flag to check if the room has been booked in this iteration

            if (room.checkRoom("VIP") > 0) {

                room.bookRoom(user, "VIP");

                printer.printInfo(user.getName(), user.getMemberType(), "VIP");

                roomBooked = true;

            } else if (room.checkRoom("Deluxe") > 0) {

                room.bookRoom(user, "Deluxe");

                printer.printInfo(user.getName(), user.getMemberType(), "Deluxe");

                roomBooked = true;

            } else if (room.checkRoom("Standard") > 0) {

                room.bookRoom(user, "Standard");

                printer.printInfo(user.getName(), user.getMemberType(), "Standard");

                roomBooked = true;

            }

            if (!roomBooked) {

                // All room types are fully booked, add to VIP waiting list

                waitingList.addWaiting(user, "VIP");

                printer.printInfo(user.getName(), user.getMemberType(), "Added to VIP List");

                break;  // Exit the loop as we have added the user to the waiting list

            }

        }

    }

    public void bookNormalMember(User user, int numOfRooms) {

        // Normal members can book up to 2 rooms

        boolean exclusiveRewardRedeemed = false;

        if (numOfRooms > 2 || numOfRooms <= 0) {

            throw new IllegalArgumentException("Normal members can book up to 2 rooms only and negative value is invalid");

        } else {

            for (int i = 0; i < numOfRooms; i++) {

                boolean roomBooked = false;

                if (user.getExclReward() && !exclusiveRewardRedeemed && room.checkRoom("VIP") > 0) {

                    room.bookRoom(user, "VIP");

                    printer.printInfo(user.getName(), user.getMemberType(), "VIP");

                    user.setExclReward(false); // Exclusive reward is now redeemed

                    exclusiveRewardRedeemed = true;

                    roomBooked = true;

                } else if (room.checkRoom("Deluxe") > 0) {

                    room.bookRoom(user, "Deluxe");

                    printer.printInfo(user.getName(), user.getMemberType(), "Deluxe");

                    roomBooked = true;

                } else if (room.checkRoom("Standard") > 0) {

                    room.bookRoom(user, "Standard");

                    printer.printInfo(user.getName(), user.getMemberType(), "Standard");

                    roomBooked = true;

                }

                if (!roomBooked) {

                    waitingList.addWaiting(user, "Member");

                    printer.printInfo(user.getName(), user.getMemberType(), "Added to Member List");

                    break; // Exit the loop as we have added the user to the waiting list

                }

            }

        }

    }

    public void bookNonMember(User user, int numOfRooms) {

        if (numOfRooms <= 0 || numOfRooms > 1) {

            throw new IllegalArgumentException("Non-members can book up to 1 rooms only and negative value is invalid");

        } else {

             for (int i = 1; i <= 1 ; i++) {

            // Non-members can book only one Standard room

                if (room.checkRoom("Standard") > 0) {

                    room.bookRoom(user, "Standard");

                    printer.printInfo(user.getName(), "Non-member", "Standard");

                } else {

                    // Standard room is fully booked, add to normal waiting list

                    waitingList.addWaiting(user, "Non-member");

                    printer.printInfo(user.getName(), "Non-member", "Added to Non-member List");

                }

             }

        }

    }

    public void cancelBooking(User user) {

        // Retrieve the list of rooms booked by the user

        List<String> bookedRooms = room.getBookedRoomsByUser(user);

        List<String> waitingListRooms = waitingList.getWaitingList(user.getMemberType());

        // if user booked the room and would like to cancel, then release the room

        if (!bookedRooms.isEmpty()) {

            room.releaseRoom(user);

        } else if (waitingListRooms.contains(user.getName())){

            // The user was on the waiting list and not in actual booked rooms

            waitingList.removeWaiting(user, user.getMemberType());

        } else

            throw new IllegalArgumentException("User not found in the waiting list");

        // Inform the user of the cancellation

        printer.printInfo(user.getName(), user.getMemberType(), "Booking cancelled");

    }

}

**Printer Class:**

package my.edu.utar;

public class Printer {

    public void printInfo(String name, String memberType, String roomType) {

        System.out.printf("Booking Information: Name: %s, Member Type: %s, Room Type: %s%n", name, memberType, roomType);

    }

}

**Room Class:**

package my.edu.utar;

import java.util.\*;

public class Room {

    private int vip;

    private int deluxe;

    private int standard;

    private Map<User, List<String>> userBookings;

    public Room() {

    }

    public Room(int vip, int deluxe, int standard) {

        this.vip = vip;

        this.deluxe = deluxe;

        this.standard = standard;

        this.userBookings = new HashMap<>();

    }

    public int checkRoom(String roomType) {

        switch (roomType) {

            case "VIP":

                return vip;

            case "Deluxe":

                return deluxe;

            case "Standard":

                return standard;

            default:

                return 0;

        }

    }

    public void bookRoom(User user, String roomType) {

        switch (roomType) {

            case "VIP":

                if (vip > 0) {

                    vip--;

                    addUserBooking(user, roomType);

                }

                break;

            case "Deluxe":

                if (deluxe > 0) {

                    deluxe--;

                    addUserBooking(user, roomType);

                }

                break;

            case "Standard":

                if (standard > 0) {

                    standard--;

                    addUserBooking(user, roomType);

                }

                break;

        }

    }

    public void releaseRoom(User user) {

        // Release all rooms booked by a user

        List<String> bookedRooms = getBookedRoomsByUser(user);

        if (bookedRooms != null) {

            for (String roomType : bookedRooms) {

                switch (roomType) {

                    case "VIP":

                        vip++;

                        break;

                    case "Deluxe":

                        deluxe++;

                        break;

                    case "Standard":

                        standard++;

                        break;

                }

            }

            userBookings.remove(user); // Remove the user from the booking map

        }

    }

    public void addUserBooking(User user, String roomType) {

        List<String> bookings = userBookings.containsKey(user) ? userBookings.get(user) : new ArrayList<>();

        bookings.add(roomType);

        userBookings.put(user, bookings);

    }

    public List<String> getBookedRoomsByUser(User user) {

        return userBookings.containsKey(user) ? userBookings.get(user) : new ArrayList<>();

    }

}

**User Class:**

package my.edu.utar;

public class User {

    private String name;

    private String memberType; // "VIP", "normal", or "non-member"

    private Boolean exclReward;

    public User() {

    }

    public User(String name, String memberType, Boolean exclReward) {

        this.name = name;

        this.memberType = memberType;

        this.exclReward = exclReward;

    }

    // Getters and Setters

    public String getName() {

        return name;

    }

    public String getMemberType() {

        return memberType;

    }

    public Boolean getExclReward() {

        return exclReward;

    }

    public void setName(String name) {

        if (name == null || name.matches(".\*\\d+.\*")|| name.isEmpty()) {

            throw new IllegalArgumentException("Invalid Name: " + name);

        }

        this.name = name;

    }

    public void setMemberType(String memberType) {

        if (!isValidMemberType(memberType) || memberType == null || memberType.isEmpty()|| !(memberType instanceof String)) {

            throw new IllegalArgumentException("Invalid memberType: " + memberType);

        }

        this.memberType = memberType;

    }

    public void setExclReward(Boolean exclReward) {

        if (exclReward == null) {

            throw new IllegalArgumentException("Invalid exclReward: " + exclReward);

        }

        this.exclReward = exclReward;

    }

    // Helper method to validate memberType

    private boolean isValidMemberType(String memberType) {

        return memberType != null && (memberType.equals("VIP") || memberType.equals("Member") || memberType.equals("Non-member"));

    }

}

**WaitingList Class:**

package my.edu.utar;

import java.util.\*;

public class WaitingList {

    private ArrayList<String> vipList = new ArrayList<>();

    private ArrayList<String> memberList = new ArrayList<>();

    private ArrayList<String> nonMemberList = new ArrayList<>();

    public WaitingList() {

    }

    public boolean addWaiting(User user, String listType) {

        switch (listType) {

            case "VIP":

                vipList.add(user.getName());

                return true;

            case "Member":

                memberList.add(user.getName());

                return true;

            case "Non-member":

                nonMemberList.add(user.getName());

                return true;

        }

        return false;

    }

    public boolean removeWaiting(User user, String listType) {

        switch (listType) {

            case "VIP":

                if (!vipList.contains(user.getName())) {

                    throw new IllegalArgumentException("User not found in the VIP waiting list");

                }else {

                    vipList.remove(user.getName());

                }

                return true;

            case "Member":

                if(!memberList.contains(user.getName())) {

                    throw new IllegalArgumentException("User not found in the Member waiting list");

                }else {

                    memberList.remove(user.getName());

                }

                return true;

            case "Non-member":

                if(!nonMemberList.contains(user.getName())) {

                    throw new IllegalArgumentException("User not found in the Non-member waiting list");

                }else {

                    nonMemberList.remove(user.getName());

                }

                return true;

            default:

                return false;

        }

    }

    // This method is to retrieve the waiting list, assuming it's for display or other purposes

    public ArrayList<String> getWaitingList(String listType) {

        switch (listType) {

            case "VIP":

                return vipList;

            case "Member":

                return memberList;

            case "Non-member":

                return nonMemberList;

            default:

                return null;

        }

    }

}

**Test Code**

**BookingIntegrationTest:**

package my.edu.utar;

import org.junit.\*;

import org.junit.runner.RunWith;

import junitparams.JUnitParamsRunner;

import junitparams.Parameters;

import static org.mockito.Mockito.\*;

import java.util.\*;

import static org.junit.Assert.\*;

@RunWith(JUnitParamsRunner.class)

public class BookingIntegrationTest {

    User user = new User();

    WaitingList waitingList = new WaitingList();

    Room roomMock = mock(Room.class);

    Printer printerMock = mock(Printer.class);

    Booking booking = new Booking(roomMock, waitingList, printerMock);

    @Test

        @Parameters({

            "John,VIP,true,1",

            "Jonathon,Member,false,1",

            "KeeMX,Non-member,false,1"

        })

    public void testSetBooking(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(memberType.equals("VIP") ? 5 : 0); // Example value for VIP room availability

        when(roomMock.checkRoom("Deluxe")).thenReturn(memberType.equals("Member") ? 5 : 0); // Example value for Deluxe room availability

        when(roomMock.checkRoom("Standard")).thenReturn(memberType.equals("Non-member") ? 5 : 0); // Example value for Standard room availability

        booking.setBooking(user, numberOfRooms);

        // Verify the appropriate method is called based on memberType

        if (memberType.equals("VIP")) {

            verify(roomMock, times(numberOfRooms)).bookRoom(eq(user), eq("VIP"));

        } else if (memberType.equals("Member")) {

            verify(roomMock, times(numberOfRooms)).bookRoom(eq(user), eq("Deluxe")); // Assuming Member gets Deluxe room

        } else {

            verify(roomMock, times(numberOfRooms)).bookRoom(eq(user), eq("Standard")); // Non-member gets Standard room

        }

    }

    //VIP user

    private Object[] getParamTestCharge() {

        return new Object[] {

                new Object[]{"John","VIP",true,1},

                new Object[]{"John","VIP",true,2},

                new Object[]{"John","VIP",true,3}

        };

    }

    @Test

    @Parameters(method="getParamTestCharge")

    // Test booking VIP room for a VIP user

    public void testSetBookingVipRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(4).thenReturn(3).thenReturn(2).thenReturn(1).thenReturn(0);

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock,times(numberOfRooms)).bookRoom(user, "VIP");

        verify(printerMock,times(numberOfRooms)).printInfo(user.getName(), user.getMemberType(), "VIP");

    }

    @Test

    @Parameters(method="getParamTestCharge")

    public void testSetBookingDeluxeRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(3).thenReturn(2).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, times(numberOfRooms)).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Standard"));

        // Ensure correct booking information is printed

        verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "VIP");

        verify(printerMock, times(numberOfRooms)).printInfo(user.getName(), user.getMemberType(), "Deluxe");

        verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method="getParamTestCharge")

    public void testSetBookingStandardRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(3).thenReturn(2).thenReturn(1).thenReturn(0);

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, times(numberOfRooms)).bookRoom(eq(user), eq("Standard"));

        // Ensure correct booking information is printed

        verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "VIP");

        verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Deluxe");

        verify(printerMock, times(numberOfRooms)).printInfo(user.getName(), user.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method="getParamTestCharge")

    public void testSetBookingWaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Standard"));

        waitingList.addWaiting(user, memberType);

        // Ensure correct booking information is printed

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "VIP");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Deluxe");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Standard");

       verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Added to VIP List");

    }

    private Object[] getParamTestCharge0() {

        return new Object[] {

                new Object[]{"John","VIP",true,2}

        };

    }

    @Test

    @Parameters(method = "getParamTestCharge0")

    public void testSetBooking1VipAnd1DeluxeRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0);

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Deluxe"));

        // Ensure correct booking information is printed

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "VIP");

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Deluxe");

    }

    @Test

    @Parameters(method = "getParamTestCharge0")

    public void testSetBooking1DeluxeAnd1StandardRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0);

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Standard"));

        // Ensure correct booking information is printed

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Deluxe");

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method = "getParamTestCharge0")

    public void testSetBooking1VipAnd1StandardRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0);

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Standard"));

        // Ensure correct booking information is printed

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "VIP");

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method = "getParamTestCharge0")

    public void testSetBooking1VipRoomAnd1WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Standard"));

        waitingList.addWaiting(user, memberType);

        // Ensure correct booking information is printed

       verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "VIP");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Deluxe");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Standard");

       verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method = "getParamTestCharge0")

    public void testSetBooking1DeluxeAnd1WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0); ;

        when(roomMock.checkRoom("Standard")).thenReturn(0); ;

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Standard"));

        waitingList.addWaiting(user, memberType);

        // Ensure correct booking information is printed

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "VIP");

       verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Deluxe");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Standard");

       verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method = "getParamTestCharge0")

    public void testSetBooking1StandardAnd1WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0); ;

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Standard"));

        waitingList.addWaiting(user, memberType);

        // Ensure correct booking information is printed

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "VIP");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Deluxe");

       verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Standard");

       verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Added to VIP List");

    }

    private Object[] getParamTestCharge1() {

        return new Object[] {

                new Object[]{"John","VIP",true,3}

        };

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking2VipAnd1DeluxeRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(2).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0);

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, times(2)).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Deluxe"));

        // Ensure correct booking information is printed

        verify(printerMock, times(2)).printInfo(user.getName(), user.getMemberType(), "VIP");

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Deluxe");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking2VipAnd1StandardRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(2).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0);

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, times(2)).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Standard"));

        // Ensure correct booking information is printed

        verify(printerMock, times(2)).printInfo(user.getName(), user.getMemberType(), "VIP");

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking2VipRoomAnd1WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(2).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, times(2)).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Standard"));

        waitingList.addWaiting(user, memberType);

        // Ensure correct booking information is printed

       verify(printerMock, times(2)).printInfo(user.getName(), user.getMemberType(), "VIP");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Deluxe");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Standard");

       verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking1VipAnd2DeluxeRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(2).thenReturn(1).thenReturn(0);

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, times(2)).bookRoom(eq(user), eq("Deluxe"));

        // Ensure correct booking information is printed

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "VIP");

        verify(printerMock, times(2)).printInfo(user.getName(), user.getMemberType(), "Deluxe");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking2DeluxeAnd1StandardRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("Deluxe")).thenReturn(2).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0);

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, times(2)).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Standard"));

        // Ensure correct booking information is printed

        verify(printerMock, times(2)).printInfo(user.getName(), user.getMemberType(), "Deluxe");

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking2DeluxeAnd1WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(2).thenReturn(1).thenReturn(0); ;

        when(roomMock.checkRoom("Standard")).thenReturn(0); ;

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, times(2)).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Standard"));

        waitingList.addWaiting(user, memberType);

        // Ensure correct booking information is printed

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "VIP");

       verify(printerMock, times(2)).printInfo(user.getName(), user.getMemberType(), "Deluxe");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Standard");

       verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking1VipAnd2StandardRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(2).thenReturn(1).thenReturn(0);

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, times(2)).bookRoom(eq(user), eq("Standard"));

        // Ensure correct booking information is printed

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "VIP");

        verify(printerMock, times(2)).printInfo(user.getName(), user.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking1DeluxeAnd2StandardRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(2).thenReturn(1).thenReturn(0);

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, times(2)).bookRoom(eq(user), eq("Standard"));

        // Ensure correct booking information is printed

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Deluxe");

        verify(printerMock, times(2)).printInfo(user.getName(), user.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking2StandardAnd1WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(2).thenReturn(1).thenReturn(0); ;

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, times(2)).bookRoom(eq(user), eq("Standard"));

        waitingList.addWaiting(user, memberType);

        // Ensure correct booking information is printed

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "VIP");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Deluxe");

       verify(printerMock, times(2)).printInfo(user.getName(), user.getMemberType(), "Standard");

       verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking1VipRoomAnd2WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Standard"));

        waitingList.addWaiting(user, memberType);

        // Ensure correct booking information is printed

       verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "VIP");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Deluxe");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Standard");

       verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking1DeluxeAnd2WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0); ;

        when(roomMock.checkRoom("Standard")).thenReturn(0); ;

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Standard"));

        waitingList.addWaiting(user, memberType);

        // Ensure correct booking information is printed

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "VIP");

       verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Deluxe");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Standard");

       verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking1StandardAnd2WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0); ;

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Standard"));

        waitingList.addWaiting(user, memberType);

        // Ensure correct booking information is printed

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "VIP");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Deluxe");

       verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Standard");

       verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBookingVipDeluxeAndStandardRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0);

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Standard"));

        // Ensure correct booking information is printed

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "VIP");

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Deluxe");

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking1Vip1DeluxeAnd1WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0); ;

        when(roomMock.checkRoom("Standard")).thenReturn(0); ;

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Standard"));

        waitingList.addWaiting(user, memberType);

        // Ensure correct booking information is printed

       verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "VIP");

       verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Deluxe");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Standard");

       verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking1Vip1StandardAnd1WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0); ;

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0); ;

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Standard"));

        waitingList.addWaiting(user, memberType);

        // Ensure correct booking information is printed

       verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "VIP");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Deluxe");

       verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Standard");

       verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking1Deluxe1StandardAnd1WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0); ;

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0); ;

        booking.bookVipMember(user, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Standard"));

        waitingList.addWaiting(user, memberType);

        // Ensure correct booking information is printed

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "VIP");

       verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Deluxe");

       verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Standard");

       verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Added to VIP List");

    }

    @Test(expected = IllegalArgumentException.class)

    @Parameters({

        "John,VIP,true,4",

        "Erica,VIP,false,-2",

        "Alice,Non-member,true,two"

    })

    public void illegalTestSetBookingForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        booking.bookVipMember(user, numberOfRooms);

    }

    //Member user

    private Object[] getParamTestCharge2() {

        return new Object[] {

                new Object[]{"Bob","Member",false,1},

        };

    }

    @Test

    @Parameters(method = "getParamTestCharge2")

    public void testSetBooking1DeluxeRoomForMemberUserWithoutExclReward(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0);

        booking.bookNormalMember(user, numberOfRooms);

        verify(roomMock, never()).bookRoom(user, "VIP");

        verify(roomMock, times(1)).bookRoom(user, "Deluxe");

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Deluxe");

    }

    @Test

    @Parameters({"Bob,Member,false,1"})

    public void testSetBooking1StandardRoomForMemberUserWithoutExclReward(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0);

        booking.bookNormalMember(user, numberOfRooms);

        verify(roomMock, never()).bookRoom(user, "VIP");

        verify(roomMock, never()).bookRoom(user, "Deluxe");

        verify(roomMock, times(1)).bookRoom(user, "Standard");

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Standard");

    }

    private Object[] getParamTestCharge3() {

        return new Object[] {

                new Object[]{"Bob","Member",false,2},

                new Object[]{"Bob","Member",true,2}

        };

    }

    @Test

    @Parameters(method = "getParamTestCharge3")

    public void testSetBooking2DeluxeRoomForMemberUserWithAndWithoutExclReward(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(2).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookNormalMember(user, numberOfRooms);

        verify(roomMock, never()).bookRoom(user, "VIP");

        verify(roomMock, times(2)).bookRoom(user, "Deluxe");

        verify(roomMock, never()).bookRoom(user, "Standard");

        verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "VIP");

        verify(printerMock, times(2)).printInfo(user.getName(), user.getMemberType(), "Deluxe");

        verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Standard");

        if(user.getExclReward()) {

            assertTrue(user.getExclReward());

        } else {

            assertFalse(user.getExclReward());

        }

    }

    @Test

    @Parameters(method = "getParamTestCharge3")

    public void testSetBookingDeluxeAndStandardRoomForMemberUserWithAndWithoutExclReward(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0);

        booking.bookNormalMember(user, numberOfRooms);

        verify(roomMock, never()).bookRoom(user, "VIP");

        verify(roomMock, times(1)).bookRoom(user, "Deluxe");

        verify(roomMock, times(1)).bookRoom(user, "Standard");

        verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "VIP");

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Deluxe");

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Standard");

        if(user.getExclReward()) {

            assertTrue(user.getExclReward());

        } else {

            assertFalse(user.getExclReward());

        }

    }

    @Test

    @Parameters(method = "getParamTestCharge3")

    public void testSetBooking2StandardRoomForMemberUserWithAndWithoutExclReward(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(2).thenReturn(1).thenReturn(0);

        booking.bookNormalMember(user, numberOfRooms);

        verify(roomMock, never()).bookRoom(user, "VIP");

        verify(roomMock, never()).bookRoom(user, "Deluxe");

        verify(roomMock, times(2)).bookRoom(user, "Standard");

        verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "VIP");

        verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Deluxe");

        verify(printerMock, times(2)).printInfo(user.getName(), user.getMemberType(), "Standard");

        if(user.getExclReward()) {

            assertTrue(user.getExclReward());

        } else {

            assertFalse(user.getExclReward());

        }

    }

    @Test

    @Parameters(method = "getParamTestCharge3")

    public void testSetBooking1DeluxeAnd1WaitingListForMemberUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookNormalMember(user, numberOfRooms);

        waitingList.addWaiting(user, memberType);

        verify(roomMock, never()).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Standard"));

        // Ensure correct booking information is printed

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "VIP");

       verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Deluxe");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Standard");

       verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Added to Member List");

    }

    @Test

    @Parameters(method = "getParamTestCharge3")

    public void testSetBooking1StandardAnd1WaitingListForMemberUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0);

        booking.bookNormalMember(user, numberOfRooms);

        waitingList.addWaiting(user, memberType);

        verify(roomMock, never()).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Standard"));

        // Ensure correct booking information is printed

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "VIP");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Deluxe");

       verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Standard");

       verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Added to Member List");

    }

    @Test

    @Parameters({

        "Bob,Member,false,1",

        "Bob,Member,false,2",

        "Bob,Member,true,1",

        "Bob,Member,true,2",})

    public void testSetBookingWaitingListForMemberUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookNormalMember(user, numberOfRooms);

        waitingList.addWaiting(user, memberType);

        verify(roomMock, never()).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Standard"));

        // Ensure correct booking information is printed

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "VIP");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Deluxe");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Standard");

       verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Added to Member List");

    }

    @Test

    @Parameters({"Ryan,Member,true,1"})

    public void testSetBooking1VIPRoomForMemberUserWithExclReward(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(2).thenReturn(1).thenReturn(0);

        booking.bookNormalMember(user, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(user, "VIP");

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "VIP");

    }

    private Object[] getParamTestCharge4() {

        return new Object[] {

                new Object[]{"Ryan","Member",true,2},

        };

    }

    @Test

    @Parameters(method="getParamTestCharge4")

    public void testSetBookingVIPAndStandardRoomForMemberUserWithExclReward(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0);

        booking.bookNormalMember(user, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(user, "VIP");

        verify(roomMock, never()).bookRoom(user, "Deluxe");

        verify(roomMock, times(1)).bookRoom(user, "Standard");

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "VIP");

        verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Deluxe");

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Standard");

        assertFalse(user.getExclReward());

    }

    @Test

    @Parameters(method="getParamTestCharge4")

    public void testSetBookingVIPAndDeluxeRoomForMemberUserWithExclReward(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0);

        booking.bookNormalMember(user, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(user, "VIP");

        verify(roomMock, times(1)).bookRoom(user, "Deluxe");

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "VIP");

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Deluxe");

        assertFalse(user.getExclReward());

    }

    @Test

    @Parameters(method="getParamTestCharge4")

    public void testSetBooking1VipAnd1WaitingListForMemberUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookNormalMember(user, numberOfRooms);

        waitingList.addWaiting(user, memberType);

        verify(roomMock, times(1)).bookRoom(eq(user), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(user), eq("Standard"));

        // Ensure correct booking information is printed

       verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "VIP");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Deluxe");

       verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Standard");

       verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Added to Member List");

    }

    @Test(expected = IllegalArgumentException.class)

    @Parameters({

        "Bob,Member,false,3",

        "Ryan,Member,true,-2",

        "Alice,Non-member,true,two"

    })

    public void illegalTestSetBookingForMemberUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        booking.bookNormalMember(user, numberOfRooms);

    }

    //Non-member user

    private Object[] getParamTestCharge5() {

        return new Object[] {

                new Object[]{"KeeMX","Non-member",false,1},

        };

    }

    @Test

    @Parameters(method="getParamTestCharge5")

    public void testSetBookingStandardRoomForNonMemberUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("Standard")).thenReturn(1);

        booking.bookNonMember(user, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(user), eq("Standard"));

        verify(printerMock, times(1)).printInfo(user.getName(), user.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method="getParamTestCharge5")

    public void testSetBookingWaitingListForNonMemberUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookNonMember(user, numberOfRooms);

        waitingList.addWaiting(user, memberType);

        verify(roomMock, never()).bookRoom(eq(user), eq("Standard"));

        verify(printerMock, never()).printInfo(user.getName(), user.getMemberType(), "Standard");

        verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Added to Non-member List");

    }

    @Test(expected = IllegalArgumentException.class)

    @Parameters({

        "KeeMX,Non-member,false,3",

        "Chris,Non-member,false,-2",

        "Alice,Non-member,true,two"

    })

    public void illegalTestSetBookingForNonMemberUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        User user = new User(name, memberType, exclReward);

        booking.bookNormalMember(user, numberOfRooms);

    }

    //Test cancelBooking method

    private Object[] getParamTestCharge6() {

        return new Object[] {

                new Object[]{"John","VIP",true,"VIP","Deluxe"},

                new Object[]{"Bob","Member",false,"Deluxe","Deluxe"},

                new Object[]{"KeeMX","Non-member",false,"Standard","Standard"},

        };

    }

    @Test

    @Parameters(method="getParamTestCharge6")

    public void testCancelBookingWithBookedRooms(String name, String memberType, boolean exclReward,String roomType1,String roomType2) {

        User user = new User(name, memberType, exclReward);

        List<String> rooms = Arrays.asList(roomType1, roomType2);

        when(roomMock.getBookedRoomsByUser(user)).thenReturn(rooms);

        booking.cancelBooking(user);

        verify(roomMock).releaseRoom(user);

        verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Booking cancelled");

    }

    @Test

    @Parameters(method="getParamTestCharge6")

    public void testCancelBookingWithUserOnWaitingListOnly(String name, String memberType, boolean exclReward,String roomType1,String roomType2) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.getBookedRoomsByUser(user)).thenReturn(new ArrayList<>());

        // 2 rooms added to waiting list

        waitingList.addWaiting(user,user.getMemberType());

        waitingList.addWaiting(user,user.getMemberType());

        booking.cancelBooking(user);

        waitingList.removeWaiting(user,user.getMemberType());

        verify(roomMock, never()).releaseRoom(user);

        verify(printerMock).printInfo(user.getName(), user.getMemberType(), "Booking cancelled");

    }

    @Test(expected = IllegalArgumentException.class)

    @Parameters(method="getParamTestCharge6")

    public void illegalTestCancelBookingWithNoBookingsOrWaitingList(String name, String memberType, boolean exclReward,String roomType1,String roomType2) {

        User user = new User(name, memberType, exclReward);

        when(roomMock.getBookedRoomsByUser(user)).thenReturn(new ArrayList<>());

        waitingList.removeWaiting(user,memberType);

        booking.cancelBooking(user);

    }

}

**BookingTest:**

package my.edu.utar;

import org.junit.\*;

import org.junit.runner.RunWith;

import junitparams.JUnitParamsRunner;

import junitparams.Parameters;

import static org.mockito.Mockito.\*;

import java.util.\*;

import static org.junit.Assert.\*;

@RunWith(JUnitParamsRunner.class)

public class BookingTest {

    User userMock = mock(User.class);

    WaitingList waitingListMock = mock(WaitingList.class);

    Room roomMock = mock(Room.class);

    Printer printerMock = mock(Printer.class);

    Booking booking = new Booking(roomMock, waitingListMock, printerMock);

    @Test

    @Parameters({"John,VIP,true,1","Jonathon,Member,false,1","KeeMX,Non-member,false,1"})

    public void testSetBooking(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(5); // Example value for VIP room availability

        when(roomMock.checkRoom("Deluxe")).thenReturn(5); // Example value for Deluxe room availability

        when(roomMock.checkRoom("Standard")).thenReturn(5); // Example value for Standard room availability

        booking.setBooking(userMock, numberOfRooms);

        // Verify the appropriate method is called based on memberType

        if (memberType.equals("VIP")){

            verify(roomMock, times(1)).checkRoom("VIP");

        } else if (memberType.equals("Member")) {

            verify(roomMock, times(1)).checkRoom("Deluxe");

        } else {

            verify(roomMock, times(1)).checkRoom("Standard");

        }

    }

    //VIP userMock

    private Object[] getParamTestCharge() {

        return new Object[] {

                new Object[]{"John","VIP",true,1},

                new Object[]{"John","VIP",true,2},

                new Object[]{"John","VIP",true,3}

        };

    }

    @Test

    @Parameters(method="getParamTestCharge")

    // Test booking VIP room for a VIP userMock

    public void testSetBookingVipRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(4).thenReturn(3).thenReturn(2).thenReturn(1).thenReturn(0);

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock,times(numberOfRooms)).bookRoom(userMock, "VIP");

        verify(printerMock,times(numberOfRooms)).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

    }

    @Test

    @Parameters(method="getParamTestCharge")

    public void testSetBookingDeluxeRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(3).thenReturn(2).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, times(numberOfRooms)).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, times(numberOfRooms)).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method="getParamTestCharge")

    public void testSetBookingStandardRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(3).thenReturn(2).thenReturn(1).thenReturn(0);

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, times(numberOfRooms)).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, times(numberOfRooms)).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method="getParamTestCharge")

    public void testSetBookingWaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

        verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Added to VIP List");

    }

    private Object[] getParamTestCharge0() {

        return new Object[] {

                new Object[]{"John","VIP",true,2},

        };

    }

    @Test

    @Parameters(method="getParamTestCharge0")

    public void testSetBooking1VipAnd1DeluxeRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0);

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Deluxe"));

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

    }

    @Test

    @Parameters(method="getParamTestCharge0")

    public void testSetBooking1DeluxeAnd1StandardRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0);

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method="getParamTestCharge0")

    public void testSetBooking1VipAnd1StandardRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0);

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method="getParamTestCharge0")

    public void testSetBooking1VipRoomAnd1WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

        verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method="getParamTestCharge0")

    public void testSetBooking1DeluxeAnd1WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0); ;

        when(roomMock.checkRoom("Standard")).thenReturn(0); ;

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

        verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method="getParamTestCharge0")

    public void testSetBooking1StandardAnd1WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0); ;

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

        verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Added to VIP List");

    }

    private Object[] getParamTestCharge1() {

        return new Object[] {

                new Object[]{"John","VIP",true,3}

        };

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking2VipAnd1DeluxeRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(2).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0);

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, times(2)).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Deluxe"));

        verify(printerMock, times(2)).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking2VipAnd1StandardRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(2).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0);

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, times(2)).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, times(2)).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking2VipRoomAnd1WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(2).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, times(2)).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, times(2)).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

        verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking1VipAnd2DeluxeRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(2).thenReturn(1).thenReturn(0);

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, times(2)).bookRoom(eq(userMock), eq("Deluxe"));

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, times(2)).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking2DeluxeAnd1StandardRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(2).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0);

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, times(2)).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, times(2)).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking2DeluxeAnd1WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(2).thenReturn(1).thenReturn(0); ;

        when(roomMock.checkRoom("Standard")).thenReturn(0); ;

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, times(2)).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, times(2)).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

        verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking1VipAnd2StandardRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(2).thenReturn(1).thenReturn(0);

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, times(2)).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, times(2)).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking1DeluxeAnd2StandardRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(2).thenReturn(1).thenReturn(0);

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, times(2)).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, times(2)).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking2StandardAnd1WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(2).thenReturn(1).thenReturn(0); ;

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, times(2)).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, times(2)).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

        verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking1VipRoomAnd2WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

        verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking1DeluxeAnd2WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0); ;

        when(roomMock.checkRoom("Standard")).thenReturn(0); ;

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

        verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking1StandardAnd2WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0); ;

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

        verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBookingVipDeluxeAndStandardRoomForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0);

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking1Vip1DeluxeAnd1WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0); ;

        when(roomMock.checkRoom("Standard")).thenReturn(0); ;

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

        verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking1Vip1StandardAnd1WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0); ;

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0); ;

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

        verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Added to VIP List");

    }

    @Test

    @Parameters(method="getParamTestCharge1")

    public void testSetBooking1Deluxe1StandardAnd1WaitingListForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0); ;

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0); ;

        booking.bookVipMember(userMock, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

        verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Added to VIP List");

    }

    @Test(expected = IllegalArgumentException.class)

    @Parameters({

        "John,VIP,true,4",

        "Erica,VIP,false,0",

        "Alice,VIP,true,two"

    })

    public void illegalTestSetBookingForVipUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        booking.bookVipMember(userMock, numberOfRooms);

    }

    //Member userMock

    private Object[] getParamTestCharge2() {

        return new Object[] {

                new Object[]{"Bob","Member",false,1},

        };

    }

    @Test

    @Parameters(method = "getParamTestCharge2")

    public void testSetBooking1DeluxeRoomForMemberUserWithoutExclReward(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0);

        booking.bookNormalMember(userMock, numberOfRooms);

        verify(roomMock, never()).bookRoom(userMock, "VIP");

        verify(roomMock, times(1)).bookRoom(userMock, "Deluxe");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

    }

    @Test

    @Parameters(method = "getParamTestCharge2")

    public void testSetBooking1StandardRoomForMemberUserWithoutExclReward(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0);

        booking.bookNormalMember(userMock, numberOfRooms);

        verify(roomMock, never()).bookRoom(userMock, "VIP");

        verify(roomMock, never()).bookRoom(userMock, "Deluxe");

        verify(roomMock, times(1)).bookRoom(userMock, "Standard");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

    }

    private Object[] getParamTestCharge3() {

        return new Object[] {

                new Object[]{"Bob","Member",false,2},

                new Object[]{"Bob","Member",true,2}

        };

    }

    @Test

    @Parameters(method = "getParamTestCharge3")

    public void testSetBooking2DeluxeRoomForMemberUserWithAndWithoutExclReward(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(2).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookNormalMember(userMock, numberOfRooms);

        verify(roomMock, never()).bookRoom(userMock, "VIP");

        verify(roomMock, times(2)).bookRoom(userMock, "Deluxe");

        verify(roomMock, never()).bookRoom(userMock, "Standard");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, times(2)).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

        if(userMock.getExclReward()) {

            assertTrue(userMock.getExclReward());

        } else {

            assertFalse(userMock.getExclReward());

        }

    }

    @Test

    @Parameters(method = "getParamTestCharge3")

    public void testSetBookingDeluxeAndStandardRoomForMemberUserWithAndWithoutExclReward(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0);

        booking.bookNormalMember(userMock, numberOfRooms);

        verify(roomMock, never()).bookRoom(userMock, "VIP");

        verify(roomMock, times(1)).bookRoom(userMock, "Deluxe");

        verify(roomMock, times(1)).bookRoom(userMock, "Standard");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

        if(userMock.getExclReward()) {

            assertTrue(userMock.getExclReward());

        } else {

            assertFalse(userMock.getExclReward());

        }

    }

    @Test

    @Parameters(method = "getParamTestCharge3")

    public void testSetBooking2StandardRoomForMemberUserWithandWithoutExclReward(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(2).thenReturn(1).thenReturn(0);

        booking.bookNormalMember(userMock, numberOfRooms);

        verify(roomMock, never()).bookRoom(userMock, "VIP");

        verify(roomMock, never()).bookRoom(userMock, "Deluxe");

        verify(roomMock, times(2)).bookRoom(userMock, "Standard");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, times(2)).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

        if(userMock.getExclReward()) {

            assertTrue(userMock.getExclReward());

        } else {

            assertFalse(userMock.getExclReward());

        }

    }

    @Test

    @Parameters(method = "getParamTestCharge3")

    public void testSetBooking1DeluxeAnd1WaitingListForMemberUserWithandWithoutExclReward(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookNormalMember(userMock, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Standard"));

        // Ensure correct booking information is printed

       verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

       verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

       verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

       verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Added to Member List");

    }

    @Test

    @Parameters(method = "getParamTestCharge3")

    public void testSetBooking1StandardAnd1WaitingListForMemberUserWithandWithoutExclReward(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0);

        booking.bookNormalMember(userMock, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Standard"));

        // Ensure correct booking information is printed

       verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

       verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

       verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

       verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Added to Member List");

    }

    @Test

    @Parameters({

        "Bob,Member,false,1",

        "Bob,Member,false,2",

        "Bob,Member,true,1",

        "Bob,Member,true,2",})

    public void testSetBookingWaitingListForMemberUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookNormalMember(userMock, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Standard"));

        // Ensure correct booking information is printed

       verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

       verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

       verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

       verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Added to Member List");

    }

    @Test

    @Parameters({"Ryan,Member,true,1"})

    public void testSetBooking1VIPRoomForMemberUserWithExclReward(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(2).thenReturn(1).thenReturn(0);

        booking.bookNormalMember(userMock, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(userMock, "VIP");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

    }

    private Object[] getParamTestCharge4() {

        return new Object[] {

                new Object[]{"Ryan","Member",true,2},

        };

    }

    @Test

    @Parameters(method="getParamTestCharge4")

    public void testSetBookingVIPAndStandardRoomForMemberUserWithExclReward(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(1).thenReturn(0);

        booking.bookNormalMember(userMock, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(userMock, "VIP");

        verify(roomMock, never()).bookRoom(userMock, "Deluxe");

        verify(roomMock, times(1)).bookRoom(userMock, "Standard");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method="getParamTestCharge4")

    public void testSetBookingVIPAndDeluxeRoomForMemberUserWithExclReward(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(1).thenReturn(0);

        booking.bookNormalMember(userMock, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(userMock, "VIP");

        verify(roomMock, times(1)).bookRoom(userMock, "Deluxe");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

    }

    @Test

    @Parameters({"Bob,Member,true,2"})

    public void testSetBooking1VipAnd1WaitingListForMemberUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("VIP")).thenReturn(1).thenReturn(0);

        when(roomMock.checkRoom("Deluxe")).thenReturn(0);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookNormalMember(userMock, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("VIP"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Deluxe"));

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "VIP");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Deluxe");

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

        verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Added to Member List");

    }

    @Test(expected = IllegalArgumentException.class)

    @Parameters({

        "Bob,Member,false,3",

        "Ryan,Member,true,0",

        "Alice,Member,true,two"

    })

    public void illegalTestSetBookingForMemberUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        booking.bookNormalMember(userMock, numberOfRooms);

    }

    //Non-member userMock

    private Object[] getParamTestCharge5() {

        return new Object[] {

                new Object[]{"KeeMX","Non-member",false,1},

        };

    }

    @Test

    @Parameters(method="getParamTestCharge5")

    public void testSetBookingStandardRoomForNonMemberUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("Standard")).thenReturn(1);

        booking.bookNonMember(userMock, numberOfRooms);

        verify(roomMock, times(1)).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, times(1)).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

    }

    @Test

    @Parameters(method="getParamTestCharge5")

    public void testSetBookingWaitingListForNonMemberUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.checkRoom("Standard")).thenReturn(0);

        booking.bookNonMember(userMock, numberOfRooms);

        verify(roomMock, never()).bookRoom(eq(userMock), eq("Standard"));

        verify(printerMock, never()).printInfo(userMock.getName(), userMock.getMemberType(), "Standard");

        verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Added to Non-member List");

    }

    @Test(expected = IllegalArgumentException.class)

    @Parameters({

        "KeeMX,Non-member,false,3",

        "Chris,Non-member,false,0",

        "Alice,Non-member,true,two"

    })

    public void illegalTestSetBookingForNonMemberUser(String name, String memberType, boolean exclReward,int numberOfRooms) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        booking.bookNormalMember(userMock, numberOfRooms);

    }

    //Test cancelBooking method

    private Object[] getParamTestCharge6() {

        return new Object[] {

                new Object[]{"John","VIP",true,"VIP","Deluxe"},

                new Object[]{"Bob","Member",false,"Deluxe","Deluxe"},

                new Object[]{"KeeMX","Non-member",false,"Standard","Standard"},

        };

    }

    @Test

    @Parameters(method="getParamTestCharge6")

    public void testCancelBookingWithBookedRooms(String name, String memberType, boolean exclReward,String roomType1,String roomType2) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        List<String> rooms = Arrays.asList(roomType1, roomType2);

        when(roomMock.getBookedRoomsByUser(userMock)).thenReturn(rooms);

        booking.cancelBooking(userMock);

        verify(roomMock).releaseRoom(userMock);

        verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Booking cancelled");

    }

    @Test

    @Parameters(method="getParamTestCharge6")

    public void testCancelBookingWithUserOnWaitingListOnly(String name, String memberType, boolean exclReward,String roomType1,String roomType2) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        ArrayList<String> users = new ArrayList<>();

        users.add(userMock.getName());

        users.add(userMock.getMemberType());

        users.add(String.valueOf(userMock.getExclReward()));

        users.add(roomType1);

        when(roomMock.getBookedRoomsByUser(userMock)).thenReturn(users);

        booking.cancelBooking(userMock);

        verify(printerMock).printInfo(userMock.getName(), userMock.getMemberType(), "Booking cancelled");

    }

    @Test(expected = IllegalArgumentException.class)

    @Parameters(method="getParamTestCharge6")

    public void illegalTestCancelBookingWithNoBookingsOrWaitingList(String name, String memberType, boolean exclReward,String roomType1,String roomType2) {

        when(userMock.getName()).thenReturn(name);

        when(userMock.getMemberType()).thenReturn(memberType);

        when(userMock.getExclReward()).thenReturn(exclReward);

        when(roomMock.getBookedRoomsByUser(userMock)).thenReturn(new ArrayList<>());

        booking.cancelBooking(userMock);

    }

}

**User Test:**

package my.edu.utar;

import static org.junit.Assert.\*;

import org.junit.\*;

import org.junit.runner.RunWith;

import junitparams.JUnitParamsRunner;

import junitparams.Parameters;

import java.util.\*;

@RunWith(JUnitParamsRunner.class)

public class UserTest {

    User user = new User();

    @Before

    public void setUp() {

        user = new User("DefaultName", "Non-member", false); // Example initialization

    }

    @Test

    public void testGetName() {

        User user1 = new User("John","VIP",true);

        assertEquals("Name should be " + "John", "John", user1.getName());

    }

    @Test

    public void testGetMemberType() {

        User user2 = new User("John","VIP",true);

        assertEquals("Member type should be " + "VIP", "VIP", user2.getMemberType());

    }

    @Test

    public void testGetExclReward() {

         User user3 = new User("John","VIP",true);

        assertEquals("Exclusive reward should be " + true, true, user3.getExclReward());

    }

    // Parameterized tests for setters

    private Object[] getParamTestCharge() {

        return new Object[] {

                new Object[]{"John","VIP",true},

                new Object[]{"Ryan","Member",true},

                new Object[]{"KeeMX","Non-member",false},

        };

    }

    @Test

    @Parameters(method="getParamTestCharge")

    public void testSetName(String name, String memberType, boolean exclReward) {

        user.setName(name);

        assertEquals("Name should be " + name, name, user.getName());

    }

    @Test

    @Parameters(method="getParamTestCharge")

    public void testSetMemberType(String name, String memberType, boolean exclReward) {

        user.setMemberType(memberType);

        assertEquals("Member type should be " + memberType, memberType, user.getMemberType());

    }

    @Test

    @Parameters(method="getParamTestCharge")

    public void testSetExclReward(String name, String memberType, boolean exclReward) {

        user.setExclReward(exclReward);

        assertEquals("Exclusive reward should be " + exclReward, exclReward, user.getExclReward());

    }

    // Illegal testing

    private Object[] getParamTestCharge1() {

        return new Object[] {

                new Object[]{null,"VIP",true},

                new Object[]{"","VIP",false},

                new Object[]{123,"VIP",true},

                new Object[]{123.00,"VIP",false},

        };

    }

    @Test(expected = IllegalArgumentException.class)

    @Parameters(method="getParamTestCharge1")

    public void illegalTestSetName(String name, String memberType, boolean exclReward) {

        user.setName(name);

    }

    private Object[] getParamTestCharge2() {

        return new Object[] {

                new Object[]{"John",null,true},

                new Object[]{"John","",false},

                new Object[]{"John",123,true},

                new Object[]{"John",123.00,false},

                new Object[]{"John","Ehllo",false},

        };

    }

    @Test(expected = IllegalArgumentException.class)

    @Parameters(method="getParamTestCharge2")

    public void illegalTestSetMemberType(String name, String memberType, boolean exclReward) {

        user.setMemberType(memberType);

    }

    private Object[] getParamTestCharge3() {

        return new Object[] {

                new Object[]{"John","VIP",null},

                new Object[]{"John","VIP",1233},

                new Object[]{"John","VIP",123.02},

        };

    }

    @Test(expected = IllegalArgumentException.class)

    @Parameters(method="getParamTestCharge3")

    public void illegalTestSetExclReward(String name, String memberType, boolean exclReward) {

        user.setExclReward(exclReward);

    }

}

**WaitingListTest:**

package my.edu.utar;

import static org.junit.Assert.\*;

import org.junit.\*;

import org.junit.runner.RunWith;

import org.junit.runners.MethodSorters;

import junitparams.JUnitParamsRunner;

import junitparams.Parameters;

import java.io.\*;

import java.util.\*;

@RunWith(JUnitParamsRunner.class)

@FixMethodOrder(MethodSorters.NAME\_ASCENDING)

public class WaitingListTest {

    WaitingList wl = new WaitingList();

    private static final String TEST\_FILE\_PATH = "waitingListUsers.txt";

    @BeforeClass

    public static void setUpBeforeClass() throws IOException {

        // Clear the test file before any test methods are executed

        clearTestFile(TEST\_FILE\_PATH);

    }

    private static void clearTestFile(String filename) throws IOException {

        try (PrintWriter writer = new PrintWriter(filename)) {

            writer.print("");

        }

    }

    @Before

    public void setUp() throws IOException {

        loadUsersIntoWaitingList(TEST\_FILE\_PATH);

    }

    private void loadUsersIntoWaitingList(String filename) throws IOException {

        try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {

            String line;

            while ((line = reader.readLine()) != null) {

                String[] parts = line.split(",");

                if (parts.length == 2) {

                    String name = parts[0].trim();

                    String listType = parts[1].trim();

                    wl.addWaiting(new User(name, listType, false), listType);

                }

            }

        }

    }

    private void writeUsersToFile(List<String> users, String listType) throws IOException {

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(TEST\_FILE\_PATH,true))) {

            for (String user : users) {

                writer.write(user + "," + listType);

                writer.newLine();

            }

        }

    }

    private List<String> removeUserFromFile(String user, String listType) throws IOException {

        List<String> updatedLines = new ArrayList<>();

        try (BufferedReader reader = new BufferedReader(new FileReader(TEST\_FILE\_PATH))) {

            String line;

            while ((line = reader.readLine()) != null) {

                String[] parts = line.split(",");

                if (!(parts[0].trim().equals(user) && parts[1].trim().equals(listType))) {

                    updatedLines.add(line);

                }

            }

        }

        return updatedLines;

    }

    private Object[] getParamTestCharge0() {

        return new Object[] {

                new Object[]{"Erica","David","VIP"},

                new Object[]{"Jonathon","Bob","Member"},

                new Object[]{"KeeMX","Chris","Non-member"}

        };

    }

    @Test

    @Parameters(method = "getParamTestCharge0")

    public void testAddWaiting(String user1, String user2, String listType) throws IOException {

        // Add users to the waiting list

        wl.addWaiting(new User(user1, listType, false), listType);

        wl.addWaiting(new User(user2, listType, false), listType);

        // Create the expected list

        List<String> expectedList = new ArrayList<>(Arrays.asList(user1, user2));

        // Write the entire list to file

        writeUsersToFile(expectedList, listType);

        // Check if users are added correctly

        assertEquals(listType + " list should have correct users", expectedList, wl.getWaitingList(listType));

    }

    @Test

    @Parameters(method = "getParamTestCharge0")

    public void testGetWaitingList(String user1, String user2, String listType) throws IOException {

        // Check if the correct list is retrieved

        List<String> expectedList = new ArrayList<>(Arrays.asList(user1, user2));

        assertEquals(listType + " list should have correct users", expectedList, wl.getWaitingList(listType));

    }

    @Test

    @Parameters({

        "Erica, VIP",

        "Jonathon, Member",

        "KeeMX, Non-member"

    })

    public void testRemoveWaiting(String user, String listType) throws IOException {

        // Remove the specified user from the waiting list

        wl.removeWaiting(new User(user, listType, false), listType);

        // Read the lines from the file and remove the line containing the removed user

        List<String> lines = removeUserFromFile(user, listType);

        // Write the updated lines back to the file

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(TEST\_FILE\_PATH))) {

            for (String line : lines) {

                writer.write(line);

                writer.newLine();

            }

        }

        // Create the expected list using the updated lines

        List<String> expectedList = createExpectedList(lines, listType);

        // Check if the user is removed correctly

        assertEquals(listType + " list should not contain removed user", expectedList, wl.getWaitingList(listType));

    }

    private List<String> createExpectedList(List<String> lines, String listType) {

        List<String> expectedList = new ArrayList<>();

        for (String line : lines) {

            String[] parts = line.split(",");

            if (parts[1].trim().equals(listType)) {

                expectedList.add(parts[0].trim());

            }

        }

        return expectedList;

    }

    //Not in WaitingList

    @Test(expected = IllegalArgumentException.class)

    @Parameters({

        "Erica, VIP",

        "Jonathon, Member",

        "KeeMX, Non-member"

    })

    public void illegalTestRemoveWaiting(String user, String listType) {

        // Remove the specified user from the waiting list

        wl.removeWaiting(new User(user, listType, false), listType);

    }

}

**TestSuite:**

package my.edu.utar;

import org.junit.runner.RunWith;

import org.junit.runners.Suite;

import org.junit.runners.Suite.SuiteClasses;

@RunWith(value = Suite.class)

@SuiteClasses(value =

{

        BookingTest.class,

        UserTest.class,

        WaitingListTest.class,

        BookingIntegrationTest.class

        })

public class TestSuite {

}