Name: Victoria Lee

Date: 11/26/2024

Class: CMSC 345 7380 Software Engineering Principles and Techniques.

Primary Instructor: John Lee
Week: 6 - Black-box Unit Test the Reservation Class of a Small Bed & Breakfast Reservation System

Embed here a copy of your complete Java unit test source code (e.g., TestReservation.java):

Note: I will also attach the file incase the code here has errors even though it should not when I copy and pasted it here. Refer to the TestReservation.java file if this copied code has bugs/errors. It should be the exact same code. I'm doing this just in case anything goes wrong even though I saved it on word docx.

```
// Unit test Java source code for testing

import java.text.SimpleDateFormat;

import java.util.*;


/**

 * Java TestReservation for John and Janes' B&B

 * Takes information from the tests:

 * .testConstructorAndGetters

 * .testSettersAndGetters

 * .testCalculateReservationNumberOfDays

 * .testCalculateReservationBillAmount

 * .Tests out it with the black-box techniques and checks boundaries

 * This tests out the users' reservation at JJ's B&B, errors, and

 * if the software code is written correctly

 * Note: This is tested and created through Oracle in Java coding

 * <p>

 * Course: CMSC 345 software engineering principals and techniques

 * <p>

 * Date: 11/26/2024

 * <p>
 * Project: Project 3
 *
 * @author Victoria Lee
 *
 * @version JRE17
 */
public class TestReservation {


        /**
         * datePattern: this is a string for the date when creating reservation objects.
         * It should do month, day, year.
         */
    private static String datePattern = "MMM dd, yyyy";


    /**
     * SimpleDateFormat: this helps the computer know the date format pattern using the imported
java.text.
     */
    private static SimpleDateFormat sdf = new SimpleDateFormat(datePattern);


    /**
     * This main function executes the other methods to test the constructors, getters, setters,
     * CalculateReservationNumberOfDays, and CalculateReservationBillAmount.
     * The methods should test it with black-box testing techniques.
     * @param argv  parameter argv to pass in the inputs of the methods.
     * @throws Exception Throws exceptions if the methods or functions are not correctly inputed and
other issues.
     */
    public static void main(String argv[]) throws Exception {
        testConstructorAndGetters();

        testSettersAndGetters();

        testCalculateReservationNumberOfDays();

        testCalculateReservationBillAmount();

    }


    /**

     * This method/function tests out the constructor and getters of the Reservation class for users'
reservation at JJ's B&B.

     * It should incorporate black-box testing techniques and other checks.

     */

    public static void testConstructorAndGetters() {

        System.out.println();

        System.out.println("Testing Constructor and Getters");

        System.out.println("-------------------------------");

        // Equivalence Partitioning Test Cases

        Reservation r = new Reservation(1,"RoomWBath", "Jun 16, 2022", "Jun 19, 2022");

        Reservation r2 = new Reservation(7,"RoomWBath", "Jun 16, 2022", "Jun 19, 2022");

        Assert.assertNotEqualsUUID(r.getReservationID(), r2.getReservationID());

        Assert.assertEqualsDate(r.getReservationDate(), new Date());


        Assert.assertEqualsInt(r.getGuestID(), 1);

        Assert.assertEqualsString(r.getRoomType(), "RoomWBath");

        Assert.assertEqualsString(r.getReservationStartDate(), "Jun 16, 2022");

        Assert.assertEqualsString(r.getReservationEndDate(), "Jun 19, 2022");


        System.out.println();

        System.out.println("Testing Boundary Cases for Constructor and Getters");

        System.out.println("-------------------------------");
        // Boundary case for minimal date range
        Reservation r3 = new Reservation(1, "RoomWBath", "Jan 01, 2025", "Jan 02, 2025");
        Assert.assertEqualsString(r3.getReservationStartDate(), "Jan 01, 2025");
        Assert.assertEqualsString(r3.getReservationEndDate(), "Jan 02, 2025");

        // Boundary case for crossing year boundary
        Reservation r4 = new Reservation(2, "RoomWBath", "Dec 31, 2024", "Jan 01, 2025");
        Assert.assertEqualsString(r4.getReservationStartDate(), "Dec 31, 2024");
        Assert.assertEqualsString(r4.getReservationEndDate(), "Jan 01, 2025");

        // Boundary case for leap year
        Reservation r5 = new Reservation(3, "RoomWBath", "Feb 28, 2024", "Mar 01, 2024");
        Assert.assertEqualsString(r5.getReservationStartDate(), "Feb 28, 2024");
        Assert.assertEqualsString(r5.getReservationEndDate(), "Mar 01, 2024");

        System.out.println();
        System.out.println("Testing Extra Not Equals and Equals");
        System.out.println("--------------------------------");
        //Testing Not Equals and Equals
        Assert.assertEqualsUUID(r.getReservationID(), r.getReservationID());
        Assert.assertNotEqualsUUID(r3.getReservationID(), r5.getReservationID());
        Assert.assertNotEqualsString(r3.getReservationStartDate(), "Jan 02, 2025");
        Assert.assertNotEqualsInt(r3.getGuestID(), 2);
        Assert.assertEqualsInt(r4.getGuestID(), 2);
        Assert.assertEqualsDate(r.getReservationDate(), r2.getReservationDate());
        Assert.assertNotEqualsDate(r.getReservationDate(), r5.getReservationDate());
    }
```

```
  /**
   * This method/function tests out the setters and getters of the Reservation class for users'
reservation at JJ's B&B.
   * It should incorporate black-box testing techniques and other checks.
   */
  public static void testSettersAndGetters() {
    System.out.println();
    System.out.println("Testing Setters and Getters");
    System.out.println("------------------------------");
    // Equivalence Partitioning Test Cases
    Reservation r = new Reservation(1, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022");
    r.setGuestID(2);
    r.setRoom("RoomWView");
    r.setReservationStartDate("Jul 01, 2022");
    r.setReservationEndDate("Jul 05, 2022");
    Assert.assertEqualsInt(r.getGuestID(), 2);
    Assert.assertEqualsString(r.getRoomType(), "RoomWView");
    Assert.assertEqualsString(r.getReservationStartDate(), "Jul 01, 2022");
    Assert.assertEqualsString(r.getReservationEndDate(), "Jul 05, 2022");

    Reservation r2 = new Reservation(1, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022");
    r2.setGuestID(3);
    r2.setRoom("RoomWBath");
    r2.setReservationStartDate("Aug 01, 2022");
    r2.setReservationEndDate("Aug 10, 2022");
    Assert.assertEqualsInt(r2.getGuestID(), 3);
    Assert.assertEqualsString(r2.getRoomType(), "RoomWBath");
    Assert.assertEqualsString(r2.getReservationStartDate(), "Aug 01, 2022");
    Assert.assertEqualsString(r2.getReservationEndDate(), "Aug 10, 2022");
```

```
Reservation r3 = new Reservation(1, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022");
r3.setGuestID(4);
r3.setRoom("NormalRoom");
r3.setReservationStartDate("Sep 10, 2022");
r3.setReservationEndDate("Sep 15, 2022");
Assert.assertEqualsInt(r3.getGuestID(), 4);
Assert.assertEqualsString(r3.getRoomType(), "NormalRoom");
Assert.assertEqualsString(r3.getReservationStartDate(), "Sep 10, 2022");
Assert.assertEqualsString(r3.getReservationEndDate(), "Sep 15, 2022");

System.out.println();
System.out.println("Testing Boundary Cases for Setters and Getters");
System.out.println("-------------------------------");
// Boundary Value Analysis Test Cases
Reservation r4 = new Reservation(1, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022");
r4.setReservationStartDate("Jan 01, 2025");
r4.setReservationEndDate("Jan 02, 2025");
Assert.assertEqualsString(r4.getReservationStartDate(), "Jan 01, 2025");
Assert.assertEqualsString(r4.getReservationEndDate(), "Jan 02, 2025");

Reservation r5 = new Reservation(1, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022");
r5.setReservationStartDate("Dec 31, 2024");
r5.setReservationEndDate("Jan 01, 2025");
Assert.assertEqualsString(r5.getReservationStartDate(), "Dec 31, 2024");
Assert.assertEqualsString(r5.getReservationEndDate(), "Jan 01, 2025");

Reservation r6 = new Reservation(1, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022");
r6.setReservationStartDate("Feb 28, 2024");
```

```
        r6.setReservationEndDate("Mar 01, 2024");
        Assert.assertEqualsString(r6.getReservationStartDate(), "Feb 28, 2024");
        Assert.assertEqualsString(r6.getReservationEndDate(), "Mar 01, 2024");


        System.out.println();
        System.out.println("Testing Extra Not Equals and Equals");
        System.out.println("-------------------------------");
        //Testing Not Equals and Equals
        Assert.assertNotEqualsInt(r.getGuestID(), 1);
        Assert.assertNotEqualsString(r.getRoomType(), "RoomWView");  // Fails as it equals RoomWView
        Assert.assertNotEqualsString(r.getRoomType(), "RoomWBath");
        Assert.assertNotEqualsString(r.getReservationStartDate(), "Jul 01, 2022");
        Assert.assertNotEqualsString(r.getReservationEndDate(), "Jul 05, 2022");  // Fails as it equals to Jul
05 2022
        Assert.assertNotEqualsString(r6.getReservationStartDate(), "Feb 28, 2024");
        Assert.assertNotEqualsString(r6.getReservationEndDate(), "Mar 01, 2024");  // Fails as it equals to
Mar 01 2024
    }


    /**
     * This method/function tests out CalculateReservationNumberOfDays for users' reservation at JJ's
B&B.
     * It should incorporate black-box testing techniques and other checks.
     */
    public static void testCalculateReservationNumberOfDays() throws Exception {
        System.out.println();
        System.out.println("Testing CalculateReservationNumberOfDays");
        System.out.println("----------------------------------------");
        // Equivalence Partitioning Test Cases
        Reservation r = new Reservation(1, "NormalRoom", "Jan 02, 2025", "Jan 05, 2025");
        long numberOfDays = r.calculateReversationNumberOfDays();
        Assert.assertEqualsInt((int) numberOfDays, 3);

        Reservation r2 = new Reservation(2, "RoomWBath", "Mar 01, 2025", "Mar 05, 2025");
        long numberOfDays2 = r2.calculateReversationNumberOfDays();
        Assert.assertEqualsInt((int) numberOfDays2, 4);

        Reservation r3 = new Reservation(3, "RoomWView", "Apr 15, 2025", "Apr 20, 2025");
        long numberOfDays3 = r3.calculateReversationNumberOfDays();
        Assert.assertEqualsInt((int) numberOfDays3, 5);

        System.out.println();
        System.out.println("Testing Boundary Cases for CalculateReservationNumberOfDays");
        System.out.println("-------------------------------");
        // Boundary Value Analysis Test Cases
        Reservation r4 = new Reservation(4, "RoomWBath", "Feb 28, 2024", "Mar 01, 2024"); // Leap Year
        long numberOfDays4 = r4.calculateReversationNumberOfDays();
        Assert.assertEqualsInt((int) numberOfDays4, 2);

        Reservation r5 = new Reservation(5, "NormalRoom", "Dec 31, 2024", "Jan 01, 2025"); // Year
    Boundary
        long numberOfDays5 = r5.calculateReversationNumberOfDays();
        Assert.assertEqualsInt((int) numberOfDays5, 1);

        Reservation r6 = new Reservation(6, "RoomWView", "Jan 01, 2025", "Jan 02, 2025");
        long numberOfDays6 = r6.calculateReversationNumberOfDays();
        Assert.assertEqualsInt((int) numberOfDays6, 1);
```

Reservation r7 = **new** Reservation(7, "RoomWView", "Jan 01, 2025", "Jan 01, 2025"); // Same Start and End Date

long numberOfDays7 = r7.calculateReversationNumberOfDays();

Assert.*assertEqualsInt*((**int**) numberOfDays7, 0);

Reservation r8 = **new** Reservation(8, "NormalRoom", "Feb 28, 2021", "Mar 01, 2021"); // Non-Leap Year

long numberOfDays8 = r8.calculateReversationNumberOfDays();

Assert.*assertEqualsInt*((**int**) numberOfDays8, 1);

Reservation r9 = **new** Reservation(9, "RoomWView", "Feb 29, 2020", "Mar 01, 2020"); // Leap Day on Leap Year

long numberOfDays9 = r9.calculateReversationNumberOfDays();

Assert.*assertEqualsInt*((**int**) numberOfDays9, 1);

System.***out***.println();

System.***out***.println("Testing Extra Not Equals and Equals");

System.***out***.println("------------------------------");

// Testing out if it will throw an exception if inputed invalid dates when changed or updated.

// It should test even if error will i calculate as well.

Reservation r10 = **new** Reservation(10, "RoomWView", "Feb 29, 2020", "Mar 01, 2020"); // Leap Day on Leap Year

r10.setRoom("RoomWBath");

r10.setReservationStartDate("Dec 1, 2024");

r10.setReservationEndDate("Jan 01, 2022");

long numberOfDays10 = r10.calculateReversationNumberOfDays();

// This tells me that the code has bee incorrectly counting the days.

// This is because the actual 672 is not 0 but a starting date cannot be in the future of the end date (start must be before end).

Assert.*assertEqualsInt*((**int**) numberOfDays10, 0);

```
    //Testing negative values and some floats in the days to throw exceptions.
    Assert.assertEqualsInt((int) numberOfDays9, 1);
    Assert.assertEqualsInt((int) numberOfDays9, -1);
    Assert.assertNotEqualsInt((int) numberOfDays9, -1);
    Assert.assertNotEqualsDouble(numberOfDays9, 1.1);
    Assert.assertEqualsDouble(numberOfDays9, 1.0);
    // Test invalid input of day, negative day number
    Reservation r11 = new Reservation(-10, "RoomWView", "Feb 29, 2023", "Mar 01, 2023");
    long numberOfDays11 = r11.calculateReversationNumberOfDays();
    Assert.assertNotEqualsInt((int) numberOfDays11, -1);
    Assert.assertEqualsInt(r11.getGuestID(), -10);
}

/**
 * This method/function tests out CalculateReservationBillAmount for users' reservation at JJ's B&B.
 * It should incorporate black-box testing techniques and other checks.
 */
public static void testCalculateReservationBillAmount() throws Exception {
    System.out.println();
    System.out.println("Testing calculateReservationBillAmount");
    System.out.println("--------------------------------------");
    // Equivalence Partitioning Test Cases

    // Test case for RoomWBath with many days
    Reservation r1 = new Reservation(1, "RoomWBath", "Jan 03, 2025", "Jan 06, 2025");
    double billAmount1 = r1.calculateReservationBillAmount();
    Assert.assertEqualsDouble(billAmount1, 600.0);  // 3 days * $200 = $600
    Assert.assertNotEqualsDouble(billAmount1, 600.0);
```

```
// 1 day with RoomWBath
Reservation r4 = new Reservation(1, "RoomWBath", "Jan 01, 2025", "Jan 02, 2025");
double billAmount4 = r4.calculateReservationBillAmount();
Assert.assertEqualsDouble(billAmount4, 200);  // 1 day * $200 = $200
Assert.assertNotEqualsDouble(billAmount4, 200);

// Test case for RoomWView
Reservation r2 = new Reservation(2, "RoomWView", "Feb 01, 2025", "Feb 04, 2025");
double billAmount2 = r2.calculateReservationBillAmount();
Assert.assertEqualsDouble(billAmount2, 525);  // 3 days * $175 = $525
Assert.assertNotEqualsDouble(billAmount2, 525);

// 1 day with RoomWView
Reservation r5 = new Reservation(1, "RoomWView", "Feb 01, 2025", "Feb 02, 2025");
double billAmount5 = r5.calculateReservationBillAmount();
Assert.assertEqualsDouble(billAmount5, 175);  // 1 day * $175 = $175
Assert.assertNotEqualsDouble(billAmount5, 175);

// Test case for NormalRoom
Reservation r3 = new Reservation(3, "NormalRoom", "Mar 01, 2025", "Mar 05, 2025");
double billAmount3 = r3.calculateReservationBillAmount();
Assert.assertEqualsDouble(billAmount3, 500);  // 4 days * $125 = $500
Assert.assertNotEqualsDouble(billAmount3, 500);

// 1 day with NormalRoom
Reservation r6 = new Reservation(6, "NormalRoom", "Mar 01, 2025", "Mar 02, 2025");
double billAmount6 = r6.calculateReservationBillAmount();
Assert.assertEqualsDouble(billAmount6, 125);  // 1 day * $125 = $125
Assert.assertNotEqualsDouble(billAmount6, 125);
```

```
    }


}
```

-- ------

Rubric Criteria:
Create black-box test cases to test the constructor and the getters methods of the Reservation class 8%
Your Response:

The code provided is using a combination of black-box testing techniques, primarily equivalence partitioning and boundary value analysis.

Test Case 1: Constructor & Getters

Test case #	Selected Inputs	Expected Result	Actual Result	Pass | Fail

1	Reservation r = new Reservation(1, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022");
	&
	Reservation r2 = new Reservation(7, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022");
		Unique UUID reservationIDs	Two unique UUID NOT equal to each other	Pass

2	Reservation r = new Reservation(1, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022");	Reservation date is today's date as a Java Date type with milliseconds	Today's date as a Java Date type with milliseconds	Fail Because of differences in millisecond when getting today's date.
3	Reservation r = new Reservation(1, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022");	Guest ID = 1	Guest ID = 1	Pass
4	Reservation r = new Reservation(1, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022");	Room type = "RoomWBath"	Room type = "RoomWBath"	Pass
5	Reservation r = new Reservation(1, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022");	Reservation start date = "Jun 16, 2022"	Reservation start date = "Jun 16, 2022"	Pass

6	Reservation r = new Reservation(1, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022");	Reservation end date = “Jun 19, 2022”	Reservation end date = “Jun 19, 2022”	Pass
7 (Boundary)	Reservation r = new Reservation(1, "RoomWBath", "Jan 01, 2025", "Jan 02, 2025");	Reservation date range is 1 day	Reservation date range is 1 day	Pass
8 (Boundary)	Reservation r = new Reservation(1, "RoomWBath", "Dec 31, 2024", "Jan 01, 2025");	Reservation date range crosses year boundary	Reservation date range crosses year boundary	Pass
9 (Boundary)	Reservation r = new Reservation(1, "RoomWBath", "Feb 28, 2024", "Mar 01, 2024"); // Leap Year	Reservation date range includes leap year day	Reservation date range includes leap year day	Pass

10 (Testing not equals and equals)	Assert.*assertEqualsUUID*(r.getReservationID(), r.getReservationID());
	
	Assert.*assertNotEqualsUUID*(r3.getReservationID(), r5.getReservationID());
	
	Assert.*assertNotEqualsString*(r3.getReservationStartDate(), "Jan 02, 2025");
	
	Assert.*assertNotEqualsInt*(r3.getGuestID(), 2);
	
	Assert.*assertEqualsInt*(r4.getGuestID(), 2);
	
	Assert.*assertEqualsDate*(r.getReservationDate(), r2.getReservationDate());
	
	Assert.*assertNotEqualsDate*(r.getReservationDate(), r5.getReservationDate());
		UUID (Equal): r.getReservationID() EQUALS r.getReservationID())
		UUID (Does not equal): r3.getReservationID() NOT EQUAL r5.getReservationID())
		Reservation Start (not equal): "Jan 02, 2025"
		GuestID (not equal): 2
		GuestID (equal): 2
		EqualsDate: r.getReservationDate() EQUALS r2.getReservationDate()
		NotEqualsDate: r.getReservationDate() NOT EQUALS r5.getReservationDate()
			UUID (Equal): r.getReservationID() EQUALS r.getReservationID())
			UUID (Does not equal): r3.getReservationID() NOT EQUAL r5.getReservationID())
			Reservation Start (not equal): "Jan 01, 2025"
			GuestID (not equal): 1
			GuestID (equal): 2
			EqualsDate: r.getReservationDate() EQUALS r2.getReservationDate()
			NotEqualsDate: r.getReservationDate() NOT EQUALS r5.getReservationDate()
				Pass
				Pass
				Pass
				Pass
				Pass
				Pass
				Pass

Notes for this table: The two classes provided test the cases based on the requirements and instructions, it explain that the expected results should use the following methods to check it and compare it. Then once it is compared it should show the pass or fail in the output. This note is just to explain more in-depth as the table does not include the details about the methods that are involved when comparing it.

For instance, in one of the examples the selected input was: 1, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022", and then the expected results should use the following similar methods to compare and output pass or fail:

- getReservationID(): Unique UUID
- getReservationDate(): Current date
- getGuestID(): 1
- getRoomType(): "RoomWBath"
- getReservationStartDate(): "Jun 16, 2022"
- getReservationEndDate(): "Jun 19, 2022"

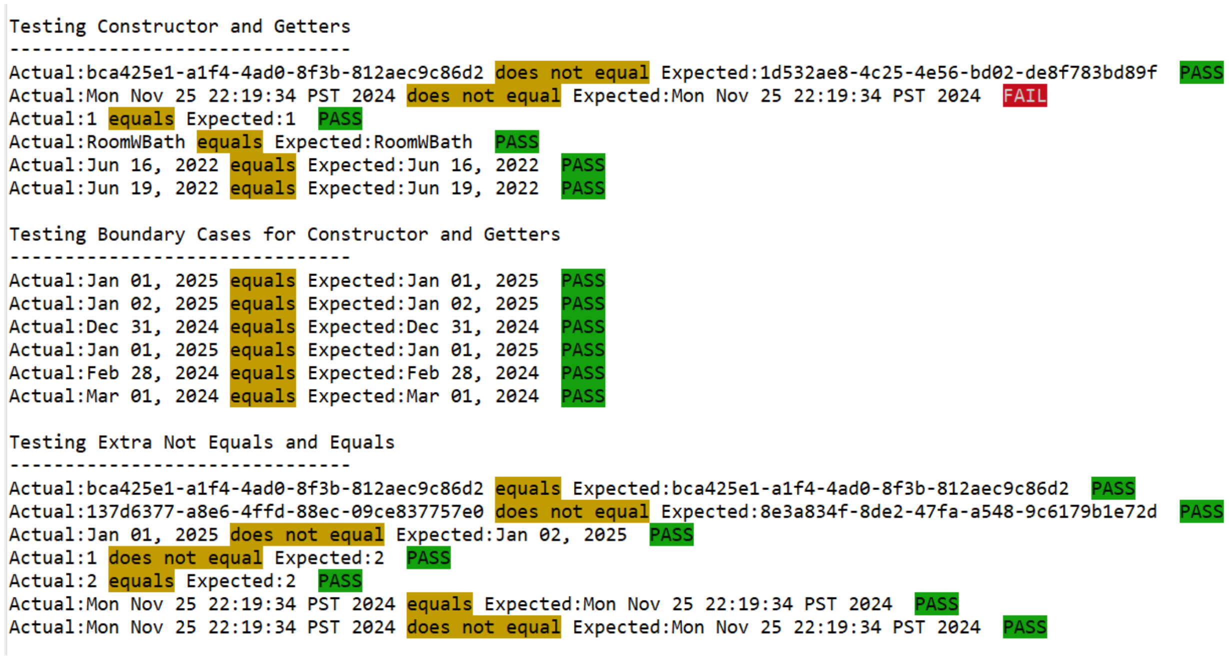
Rubric Criteria:
Execute, using w6.jar, unit tests for the constructor and the getters method of the Reservation class. Document the unit tests code and results via screenshots 10%
Your Response:

CODE:

```
  /**
   * This method/function tests out the constructor and getters of the Reservation class for users'
reservation at JJ's B&B.
   * It should incorporate black-box testing techniques and other checks.
   */
  public static void testConstructorAndGetters() {
    System.out.println();
    System.out.println("Testing Constructor and Getters");
    System.out.println("-------------------------------");
    // Equivalence Partitioning Test Cases
    Reservation r = new Reservation(1,"RoomWBath", "Jun 16, 2022", "Jun 19, 2022");
    Reservation r2 = new Reservation(7,"RoomWBath", "Jun 16, 2022", "Jun 19, 2022");
    Assert.assertNotEqualsUUID(r.getReservationID(), r2.getReservationID());
    Assert.assertEqualsDate(r.getReservationDate(), new Date());


    Assert.assertEqualsInt(r.getGuestID(), 1);
    Assert.assertEqualsString(r.getRoomType(), "RoomWBath");
    Assert.assertEqualsString(r.getReservationStartDate(), "Jun 16, 2022");
    Assert.assertEqualsString(r.getReservationEndDate(), "Jun 19, 2022");

    System.out.println();
    System.out.println("Testing Boundary Cases for Constructor and Getters");
    System.out.println("-------------------------------");

    // Boundary case for minimal date range
    Reservation r3 = new Reservation(1, "RoomWBath", "Jan 01, 2025", "Jan 02, 2025");
    Assert.assertEqualsString(r3.getReservationStartDate(), "Jan 01, 2025");
    Assert.assertEqualsString(r3.getReservationEndDate(), "Jan 02, 2025");

    // Boundary case for crossing year boundary
    Reservation r4 = new Reservation(2, "RoomWBath", "Dec 31, 2024", "Jan 01, 2025");
    Assert.assertEqualsString(r4.getReservationStartDate(), "Dec 31, 2024");
    Assert.assertEqualsString(r4.getReservationEndDate(), "Jan 01, 2025");

    // Boundary case for leap year
    Reservation r5 = new Reservation(3, "RoomWBath", "Feb 28, 2024", "Mar 01, 2024");
    Assert.assertEqualsString(r5.getReservationStartDate(), "Feb 28, 2024");
    Assert.assertEqualsString(r5.getReservationEndDate(), "Mar 01, 2024");

    System.out.println();
    System.out.println("Testing Extra Not Equals and Equals");
    System.out.println("-------------------------------");
    //Testing Not Equals and Equals
    Assert.assertEqualsUUID(r.getReservationID(), r.getReservationID());
    Assert.assertNotEqualsUUID(r3.getReservationID(), r5.getReservationID());
```

```
        Assert.assertNotEqualsString(r3.getReservationStartDate(), "Jan 02, 2025");

        Assert.assertNotEqualsInt(r3.getGuestID(), 2);

        Assert.assertEqualsInt(r4.getGuestID(), 2);

        Assert.assertEqualsDate(r.getReservationDate(), r2.getReservationDate());

        Assert.assertNotEqualsDate(r.getReservationDate(), r5.getReservationDate());

    }
```

OUTPUT TESTS:



```
Testing Constructor and Getters
-------------------------------
Actual:bca425e1-a1f4-4ad0-8f3b-812aec9c86d2 does not equal Expected:1d532ae8-4c25-4e56-bd02-de8f783bd89f  PASS
Actual:Mon Nov 25 22:19:34 PST 2024 does not equal Expected:Mon Nov 25 22:19:34 PST 2024  FAIL
Actual:1 equals Expected:1  PASS
Actual:RoomWBath equals Expected:RoomWBath  PASS
Actual:Jun 16, 2022 equals Expected:Jun 16, 2022  PASS
Actual:Jun 19, 2022 equals Expected:Jun 19, 2022  PASS

Testing Boundary Cases for Constructor and Getters
-------------------------------
Actual:Jan 01, 2025 equals Expected:Jan 01, 2025  PASS
Actual:Jan 02, 2025 equals Expected:Jan 02, 2025  PASS
Actual:Dec 31, 2024 equals Expected:Dec 31, 2024  PASS
Actual:Jan 01, 2025 equals Expected:Jan 01, 2025  PASS
Actual:Feb 28, 2024 equals Expected:Feb 28, 2024  PASS
Actual:Mar 01, 2024 equals Expected:Mar 01, 2024  PASS

Testing Extra Not Equals and Equals
-------------------------------
Actual:bca425e1-a1f4-4ad0-8f3b-812aec9c86d2 equals Expected:bca425e1-a1f4-4ad0-8f3b-812aec9c86d2  PASS
Actual:137d6377-a8e6-4ffd-88ec-09ce837757e0 does not equal Expected:8e3a834f-8de2-47fa-a548-9c6179b1e72d  PASS
Actual:Jan 01, 2025 does not equal Expected:Jan 02, 2025  PASS
Actual:1 does not equal Expected:2  PASS
Actual:2 equals Expected:2  PASS
Actual:Mon Nov 25 22:19:34 PST 2024 equals Expected:Mon Nov 25 22:19:34 PST 2024  PASS
Actual:Mon Nov 25 22:19:34 PST 2024 does not equal Expected:Mon Nov 25 22:19:34 PST 2024  PASS
```

Rubric Criteria:
Explain approach, steps, and rationale of the test cases and unit tests of testing the constructor and the getters method of the Reservation class 5%
Your Response:

Approach:

Before testing this section and creating the code, I had to read the required documents and scenarios for this case. I included the demo testing while adding more test cases after the second test case number. Then I made sure to use the black-box techniques of equivalence partitioning and boundary value analysis to create these test cases. When applying the concepts to the code, it was testing cases 1-6 for equivalence partitioning. And for boundary value analysis it was test cases 7-9. To approach this method, I tested the constructor to ensure it correctly initializes a Reservation object with the given parameters. The getter methods are tested to confirm they return the expected values set during the object construction. By using these techniques, we ensure that the Reservation class functions correctly for a variety of typical and edge case scenarios.

Steps:

1. Define the two concepts in the test cases and when creating them.
    a. Equivalence Partitioning:
        i. For inputs: Valid inputs like guest ID, room type, start date, and end date.
        ii. For Test cases: Ensure that the reservation object initializes correctly with provided inputs and verify the correctness of getter methods for valid reservations.
        iii.
    b. Boundary Value Analysis
        i. For inputs: Check boundary values for date (start and end date).
        ii. For Test cases: Test the constructor with edge dates such as the first and last possible dates of the month or year.
2. Creating Objects: Create two Reservation objects with different guest IDs and the same reservation dates.
3. Creating assertions: Use assertions to check the uniqueness of the reservation IDs and the correctness of the reservation dates.
4. Test cases creations (EP: 1-6; BVA: 7-9):
    a. Test Case #1: Verifies unique UUIDs for different reservations.
    b. Test Case #2: Verifies that the reservation date is set to today's date.
    c. Test Case #3: Verifies the correct setting of the guest ID.
    d. Test Case #4: Verifies the correct setting of the room type.
    e. Test Case #5: Verifies the correct setting of the reservation start date.
    f. Test Case #6: Verifies the correct setting of the reservation end date.
    g. Test Case #7: Verifies reservation for a minimal date range of 1 day.
    h. Test Case #8: Verifies reservation that crosses the year boundary.
    i. Test Case #9: Verifies reservation that includes the leap year day.
    j. Test Case #10: Tests out the equal and not equal that I missed.

Rationale:

First, I had to comprehend the black-box techniques for this method. According to Tsui, Karam, & Bernal (2014), GeeksforGeeks (2024), and Javapoint (n.d.), black-box testing is a methodology where the test cases are mostly derived from the requirements statements without considering the actual code content. For instance, testing based solely on specifications, without looking at the code is commonly called black-box testing. First, the user must identify equivalence classes: Divide the input domain into distinct sets of values that are likely to be processed similarly by the software. Secondly, the user must create design test cases: Select one representative value from each equivalence class to create a test case. Equivalence Partitioning testing divides the input data of the software unit into partitions of equivalent data, where each partition represents a set of values that should be treated similarly by the software. I made sure to incorporate the steps into this process. This is crucial because it reduces the total number of test cases while ensuring coverage of possible inputs. Then for Boundary value analysis, it focuses on the boundaries between partitions. I included that the test must tests at the edge of each partition, where errors are most likely to occur, including testing just inside and just outside the boundaries of input ranges. When applying the concepts to the code, it was testing cases 1-6 for equivalence partitioning. And for boundary value analysis it was test cases 7-9. By using these techniques, we ensure that the Reservation class functions correctly for a variety of typical and edge case scenarios. It also ensures that the constructor properly initializes the object and that getter methods accurately retrieve the initialized values (Tsui, Karam, & Bernal, 2014; GeeksforGeeks, 2024; Javapoint, n.d.).

Rubric Criteria:
Create black-box test cases to test the setters and the getters methods of the Reservation class 8%
Your Response:

The code provided uses a combination of black-box testing techniques, primarily equivalence partitioning and boundary value analysis.

Test Case 2: Setters & Getters

Test case #	Selected Inputs	Expected Result	Actual Result	Pass | Fail

1	r.setGuestID(2);
	r.setRoom("RoomWView");
	r.setReservationStartDate("Jul 01, 2022");
	r.setReservationEndDate("Jul 05, 2022");
		Guest ID: 2
		Room Type: "RoomWView"
		Start Date: "Jul 01, 2022"
		End Date: "Jul 05, 2022"
			Guest ID: 2
			Room Type: "RoomWView"
			Start Date: "Jun 16, 2022"
			End Date: "Jul 05, 2022"
				Pass
				Pass
				Fail
				Pass
				
				The failure demonstrates that the updated start date failed. Repeat the explanation for the other test cases.

2	r.setGuestID(3);
	r.setRoom("RoomWBath");
	r.setReservationStartDate("Aug 01, 2022");
	r.setReservationEndDate("Aug 10, 2022");
		Guest ID: 3
		Room Type: "RoomWBath"
		Start Date: "Aug 01, 2022"
		End Date: "Aug 10, 2022"
			Guest ID: 3
			Room Type: "RoomWBath"
			Start Date: "June 16, 2022"
				Pass
				Pass
				Fail
				Pass

			End Date: "Aug 10, 2022"	

3	r.setGuestID(4);
	r.setRoom("NormalRoom");
	r.setReservationStartDate("Sep 10, 2022");
	r.setReservationEndDate("Sep 15, 2022");
		Guest ID: 4
		Room Type: "NormalRoom"
		Start Date: "Sep 10, 2022"
		End Date: "Sep 15, 2022"
			Guest ID: 4
			Room Type: "NormalRoom"
			Start Date: "June 16, 2022"
			End Date: "Sep 15, 2022"
				Pass
				Pass
				Fail
				Pass

4 (boundary)	r.setReservationStartDate("Jan 01, 2025");
	r.setReservationEndDate("Jan 02, 2025");
		Start Date: "Jan 01, 2025"
		End Date: "Jan 02, 2025"
			Start Date: "Jun 16, 2022"
			End Date: "Jan 02, 2025"
				Fail
				Pass

5 (boundary)	r.setReservationStartDate("Dec 31, 2024");
	r.setReservationEndDate("Jan 01, 2025");
		Start Date: "Dec 31, 2024"
		End Date: "Jan 01, 2025"
			Start Date: "Jun 16, 2022"
			End Date: "Jan 01, 2025"
				Fail
				Pass

6 (boundary)	r.setReservationStartDate("Feb 28, 2024"); r.setReservationEndDate("Mar 01, 2024"); // Leap Year	Start Date: "Feb 28, 2024" End Date: "Mar 01, 2024"	Start Date: "Jun 16, 2022" End Date: "Mar 01, 2024"	Fail Pass

7	Assert.*assertNotEqualsInt*(r.getGuestID(), 1);
	
	Assert.*assertNotEqualsString*(r.getRoomType(), "RoomWView");
	
	Assert.*assertNotEqualsString*(r.getRoomType(), "RoomWBath");
	
	Assert.*assertNotEqualsString*(r.getReservationStartDate(), "Jul 01, 2022");
	
	Assert.*assertNotEqualsString*(r.getReservationEndDate(), "Jul 05, 2022");
	Assert.*assertNotEqualsString*(r6.getReservationStartDate(), "Feb 28, 2024");
		Int(NotEquals): 1
		String(NotEquals): RoomWView
		String(NotEquals): RoomWBath
		String(NotEquals): Jul 01, 2022
		String(NotEquals): Jul 05, 2022
		String(NotEquals): Feb 28, 2024
		String(NotEquals): Mar 01, 2024
			Int(NotEquals): 2
			String(NotEquals): RoomVWiew Fails as it equals RoomWView
			String(NotEquals): Room WView
			String(NotEquals): Jun 16, 2022
				Pass
				Fail
				Pass
				Pass
				Fail
				Pass
				Fail
				Notes:
				1st Fail: Fails as it equals RoomWView
				2nd Fail:

	Assert.*assertNotEqualsString*(r6.getReservationEndDate(), "Mar 01, 2024");		String(NotEquals): Jul 05, 2022 Fails as it equals to Jul 05 2022 String(NotEquals): Jun 16, 2022 String(NotEquals): Mar 01, 2024 Fails as it equals to Mar 01 2024	Fails as it equals to Jul 05 2022 3rd Fail: Fails as it equals to Mar 01 2024

Notes for this table: The two classes provided test the cases based on the requirements and instructions, it explain that the expected results should use the following methods to check it and compare it. Then once it is compared it should show the pass or fail in the output. This note is just to explain more in-depth as the table does not include the details about the methods that are involved when comparing it. For instance, in one of the examples the selected input was:

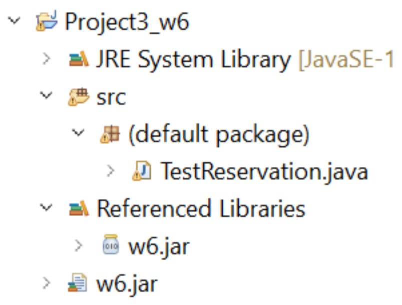
- setCustomerID(2)
- setRoom("RoomWView")
- setReservationStartDate("Jul 01, 2022")
- setReservationEndDate("Jul 05, 2022")

And then the expected results should use the following similar methods to compare and output pass or fail:

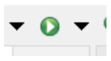
- getGuestID(): 2
- getRoomType(): "RoomWView"
- getReservationStartDate(): "Jul 01, 2022"
- getReservationEndDate(): "Jul 05, 2022"

Rubric Criteria:
Execute, using w6.jar, unit tests for the setters and the getters method of the Reservation class. Document the unit test code and results via screenshots 10%
Your Response:

Execution: Since I used Oracle, I allowed the w6.Jar as a referenced library using the properties and running it using the run debug which should automatically use the classes in that w6.jar file. All I had to do was create a main file for the test cases which is in the src file. The class is called TestReservation.java.



```
Project3_w6
  JRE System Library [JavaSE-1
  src
    (default package)
      TestReservation.java
  Referenced Libraries
    w6.jar
  w6.jar
```

← shows the w6.jar & to run the main.→ 

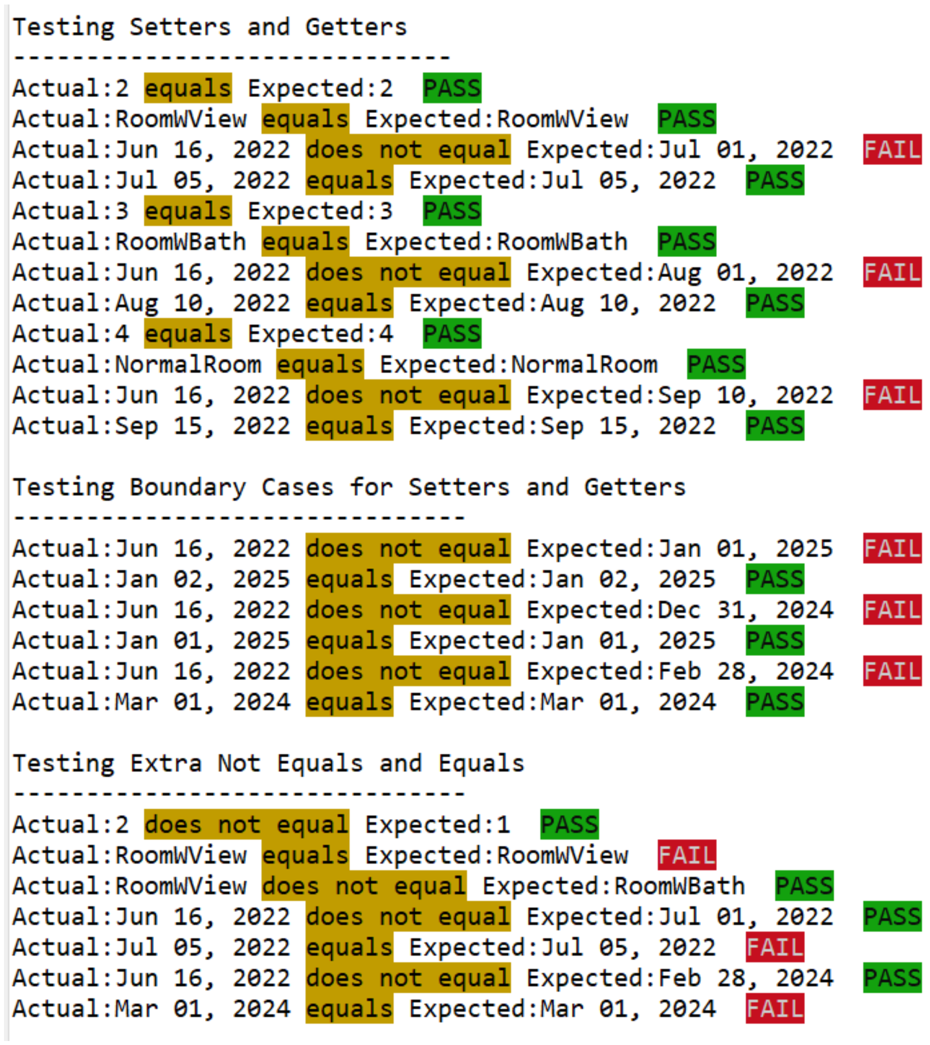
CODE:

```
  /**
   * This method/function tests out the setters and getters of the Reservation class for users' reservation at JJ's B&B.
   * It should incorporate black-box testing techniques and other checks.
   */
  public static void testSettersAndGetters() {
    System.out.println();
    System.out.println("Testing Setters and Getters");
    System.out.println("------------------------------");
    // Equivalence Partitioning Test Cases
    Reservation r = new Reservation(1, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022");
    r.setGuestID(2);
    r.setRoom("RoomWView");
    r.setReservationStartDate("Jul 01, 2022");
    r.setReservationEndDate("Jul 05, 2022");
    Assert.assertEqualsInt(r.getGuestID(), 2);
    Assert.assertEqualsString(r.getRoomType(), "RoomWView");
    Assert.assertEqualsString(r.getReservationStartDate(), "Jul 01, 2022");
    Assert.assertEqualsString(r.getReservationEndDate(), "Jul 05, 2022");

    Reservation r2 = new Reservation(1, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022");
```

```
r2.setGuestID(3);
r2.setRoom("RoomWBath");
r2.setReservationStartDate("Aug 01, 2022");
r2.setReservationEndDate("Aug 10, 2022");
Assert.assertEqualsInt(r2.getGuestID(), 3);
Assert.assertEqualsString(r2.getRoomType(), "RoomWBath");
Assert.assertEqualsString(r2.getReservationStartDate(), "Aug 01, 2022");
Assert.assertEqualsString(r2.getReservationEndDate(), "Aug 10, 2022");

Reservation r3 = new Reservation(1, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022");
r3.setGuestID(4);
r3.setRoom("NormalRoom");
r3.setReservationStartDate("Sep 10, 2022");
r3.setReservationEndDate("Sep 15, 2022");
Assert.assertEqualsInt(r3.getGuestID(), 4);
Assert.assertEqualsString(r3.getRoomType(), "NormalRoom");
Assert.assertEqualsString(r3.getReservationStartDate(), "Sep 10, 2022");
Assert.assertEqualsString(r3.getReservationEndDate(), "Sep 15, 2022");

System.out.println();
System.out.println("Testing Boundary Cases for Setters and Getters");
System.out.println("-------------------------------");
// Boundary Value Analysis Test Cases
Reservation r4 = new Reservation(1, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022");
r4.setReservationStartDate("Jan 01, 2025");
r4.setReservationEndDate("Jan 02, 2025");
Assert.assertEqualsString(r4.getReservationStartDate(), "Jan 01, 2025");
Assert.assertEqualsString(r4.getReservationEndDate(), "Jan 02, 2025");
```

```
        Reservation r5 = new Reservation(1, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022");
        r5.setReservationStartDate("Dec 31, 2024");
        r5.setReservationEndDate("Jan 01, 2025");
        Assert.assertEqualsString(r5.getReservationStartDate(), "Dec 31, 2024");
        Assert.assertEqualsString(r5.getReservationEndDate(), "Jan 01, 2025");

        Reservation r6 = new Reservation(1, "RoomWBath", "Jun 16, 2022", "Jun 19, 2022");
        r6.setReservationStartDate("Feb 28, 2024");
        r6.setReservationEndDate("Mar 01, 2024");
        Assert.assertEqualsString(r6.getReservationStartDate(), "Feb 28, 2024");
        Assert.assertEqualsString(r6.getReservationEndDate(), "Mar 01, 2024");

        System.out.println();
        System.out.println("Testing Extra Not Equals and Equals");
        System.out.println("------------------------------");
        //Testing Not Equals and Equals
        Assert.assertNotEqualsInt(r.getGuestID(), 1);
        Assert.assertNotEqualsString(r.getRoomType(), "RoomWView");  // Fails as it equals RoomWView
        Assert.assertNotEqualsString(r.getRoomType(), "RoomWBath");
        Assert.assertNotEqualsString(r.getReservationStartDate(), "Jul 01, 2022");
        Assert.assertNotEqualsString(r.getReservationEndDate(), "Jul 05, 2022");  // Fails as it equals to Jul
05 2022
        Assert.assertNotEqualsString(r6.getReservationStartDate(), "Feb 28, 2024");
        Assert.assertNotEqualsString(r6.getReservationEndDate(), "Mar 01, 2024");  // Fails as it equals to
Mar 01 2024
    }
```

OUTPUT TESTS:



```
Testing Setters and Getters
------------------------------
Actual:2 equals Expected:2  PASS
Actual:RoomWView equals Expected:RoomWView  PASS
Actual:Jun 16, 2022 does not equal Expected:Jul 01, 2022  FAIL
Actual:Jul 05, 2022 equals Expected:Jul 05, 2022  PASS
Actual:3 equals Expected:3  PASS
Actual:RoomWBath equals Expected:RoomWBath  PASS
Actual:Jun 16, 2022 does not equal Expected:Aug 01, 2022  FAIL
Actual:Aug 10, 2022 equals Expected:Aug 10, 2022  PASS
Actual:4 equals Expected:4  PASS
Actual:NormalRoom equals Expected:NormalRoom  PASS
Actual:Jun 16, 2022 does not equal Expected:Sep 10, 2022  FAIL
Actual:Sep 15, 2022 equals Expected:Sep 15, 2022  PASS

Testing Boundary Cases for Setters and Getters
-------------------------------
Actual:Jun 16, 2022 does not equal Expected:Jan 01, 2025  FAIL
Actual:Jan 02, 2025 equals Expected:Jan 02, 2025  PASS
Actual:Jun 16, 2022 does not equal Expected:Dec 31, 2024  FAIL
Actual:Jan 01, 2025 equals Expected:Jan 01, 2025  PASS
Actual:Jun 16, 2022 does not equal Expected:Feb 28, 2024  FAIL
Actual:Mar 01, 2024 equals Expected:Mar 01, 2024  PASS

Testing Extra Not Equals and Equals
-------------------------------
Actual:2 does not equal Expected:1  PASS
Actual:RoomWView equals Expected:RoomWView  FAIL
Actual:RoomWView does not equal Expected:RoomWBath  PASS
Actual:Jun 16, 2022 does not equal Expected:Jul 01, 2022  PASS
Actual:Jul 05, 2022 equals Expected:Jul 05, 2022  FAIL
Actual:Jun 16, 2022 does not equal Expected:Feb 28, 2024  PASS
Actual:Mar 01, 2024 equals Expected:Mar 01, 2024  FAIL
```

Rubric Criteria:
Explain approach, steps, and rationale of the test cases and unit tests of testing the setters and the getters method of the Reservation class 5%
Your Response:

Approach:

Before testing this section and creating the code, I had to read the required documents and scenarios for this case. I included the demo testing while adding more test cases after the second test case number. Then I made sure to use the black-box techniques of equivalence partitioning and boundary value analysis to create these test cases. When applying the concepts to the code, it was testing cases 1-3 for equivalence partitioning. And for boundary value analysis it was test cases 4-6. To approach this method, I created tests for the setter methods to ensure they correctly update the attributes of a Reservation object and then I used the getter methods to verify the updates made by the setters.

Steps:

1. Define the two concepts in the test cases and when creating them.
    a. Equivalence Partitioning:
        i. For inputs: Valid inputs for setters to update reservation details.
        ii. For Test cases: Ensure setter methods correctly update the reservation attributes and verify the correctness of getter methods after updates.
    b. Boundary Value Analysis
        i. For inputs: Check boundary values for date (start and end date and ID changes).
        ii. For Test cases: Test the constructor with edge dates.
2. Creating Objects: Create two Reservation objects. Then I created some changes to ID and the changes to the dates. Update its attributes using setter methods.
3. Creating assertions: Use assertions to check if the updated values match the expected results.
4. Test cases creations (EP: 1-3; BVA: 4-6):
    a. Test Case #1: Verifies that the setter methods correctly update the reservation attributes and that the getter methods return the updated values accurately.
    b. Test Case #2: Checks if the setters can handle a different guest ID and room type, ensuring that these changes are reflected correctly through the getters.
    c. Test Case #3: This test case ensures that the setters and getters work correctly for the "NormalRoom" type, verifying proper updates and retrievals.
    d. Test Case #4: Verifies that the methods correctly handle a minimal date range, ensuring that a 1-day reservation is accurately processed.
    e. Test Case #5: This test case ensures that the methods correctly handle reservations that span across a year boundary.
    f. Test Case #6: This test case ensures that the methods correctly handle dates around the leap year boundary, verifying correct date processing for leap years.
    g. Test Case #7: Tests out the equal and not equal that I missed.

Rationale:

First, after comprehending the black-box techniques previously, I used the concepts to then include the test cases. This time it was much shorter as it was just the reservation class. According to Tsui, Karam, & Bernal (2014), GeeksforGeeks (2024), and Javapoint (n.d.), Equivalence Partitioning testing divides the input data of the software unit into partitions of equivalent data, where each partition represents a set of values that should be treated similarly by the software. I made sure to incorporate the steps into this process. This is crucial because it reduces the total number of test cases while ensuring coverage of possible inputs. Then for Boundary value analysis, it focuses on the boundaries between partitions. I included that the test must tests at the edge of each partition, where errors are most likely to occur, including testing just inside and just outside the boundaries of input ranges. When applying the concepts to the code, it was testing cases 1-3 for equivalence partitioning. And for boundary value analysis it was test cases 4-6. The test method testSettersAndGetters initializes a Reservation object and uses setter methods to update the reservation attributes. It then uses getter methods to retrieve these attributes and compares the retrieved values with the expected results using assertions. These tests help ensure that the setters and getters in the Reservation class function as intended for typical and edge cases. By using these techniques, we ensure that setter methods properly update the object's state and that getter methods accurately retrieve the updated values (for the Reservation class) (Tsui, Karam, & Bernal, 2014; GeeksforGeeks, 2024; Javapoint, n.d.).

Rubric Criteria:
Create black-box test cases to test the calculateReservationNumberOfDays() method of the Reservation class 8%
Your Response:

The code provided is using a combination of black-box testing techniques, primarily equivalence partitioning and boundary value analysis.

Test Case 3: calculateReservationNumberOfDays()

Test case #	Selected Inputs	Expected Result	Actual Result	Pass | Fail
1	Reservation r = new Reservation(1, "NormalRoom", "Jan 02, 2025", "Jan 05, 2025");	Number of days: 3	Number of days: 3	Pass
2	Reservation r = new Reservation(2, "RoomWBath", "Mar 01, 2025", "Mar 05, 2025");	Number of days: 4	Number of days: 4	Pass
3	Reservation r = new Reservation(3, "RoomWView", "Apr 15, 2025", "Apr 20, 2025");	Number of days: 5	Number of days: 5	Pass
4 (Boundary)	Reservation r = new Reservation(4, "RoomWBath", "Feb 28, 2024", "Mar 01, 2024");	Number of days: 2	Number of days: 2	Pass
5 (Boundary)	Reservation r = new Reservation(5, "NormalRoom", "Dec 31, 2024", "Jan 01, 2025");	Number of days: 1	Number of days: 1	Pass
6 (Boundary)	Reservation r = new Reservation(6, "RoomWView", "Jan 01, 2025", "Jan 02, 2025");	Number of days: 1	Number of days: 1	Pass
7 (Boundary)	Reservation r = new Reservation(7, "RoomWView", "Jan 01, 2025", "Jan 01, 2025");	Number of days: 0	Number of days: 0	Pass
8 (Boundary)	Reservation r = new Reservation(8, "NormalRoom", "Feb 28, 2021", "Mar 01, 2021");	Number of days: 1	Number of days: 1	Pass
9 (Boundary)	Reservation r = new Reservation(9, "RoomWView", "Feb 29, 2020", "Mar 01, 2020");	Number of days: 1	Number of days: 1	Pass
10	// Testing out if it will throw an exception if input invalid dates when changed or updated and errors for calculating. Reservation r10 = **new** Reservation(10, "RoomWView", "Feb 29, 2020", "Mar 01, 2020");	Number of days: 0 Number of days: 1 Number of days: -1	Number of days: 672 Number of days: 1 Number of days: 1	Fail Pass Fail Pass Pass Pass Pass

	r10.setRoom("RoomWBath");
	r10.setReservationStartDate("Dec 1, 2024");
	r10.setReservationEndDate("Jan 01, 2022");
	long numberOfDays10 = r10.calculateReversationNumberOfDays();
	Assert.*assertEqualsInt*((**int**) numberOfDays10, 0);
	Assert.*assertEqualsInt*((**int**) numberOfDays9, 1);
	Assert.*assertEqualsInt*((**int**) numberOfDays9, -1);
	Assert.*assertNotEqualsInt*((**int**) numberOfDays9, -1);
	Assert.*assertNotEqualsDouble*(numberOfDays9, 1.1);
	Assert.*assertEqualsDouble*(numberOfDays9, 1.0);
	Reservation r11 = **new** Reservation(-10, "RoomWView", "Feb 29, 2023", "Mar 01, 2023");
	long numberOfDays11 = r11.calculateReversationNumberOfDays();
	Assert.*assertNotEqualsInt*((**int**) numberOfDays11, -1);
	Assert.*assertEqualsInt*(r11.getGuestID(), -10);
		Number of days: -1
		Number of days: 1.1
		Number of days: 1.0
		Number of days: -1
		GuestID#: -10
			Number of days: 1
			Number of days: 1.0
			Number of days: 1.0
			Number of days: 0
			GuestID#: -10
				Pass
				
				Notes:
				1st Fail: Fails as the code does not account that start should be before end and also it does not equal to 0.
				2nd Fail: Fails as it does not throw and exception when an invalid day is input.
				
				The last two Passes tells me that the code is also not effective with detecting invalid guest id number (negatives), and also it correctly assumes the number of days is 0 because the guest id is invalid. For some reason it should throw an exception.

Notes for this table: The two classes provided test the cases based on the requirements and instructions, it explain that the expected results should use the following methods to check it and compare it. Then once it is compared it should show the pass or fail in the output. This note is just to explain more in-depth as the table does not include the details about the methods that are involved when comparing it. For instance, in one of the examples the selected input was:

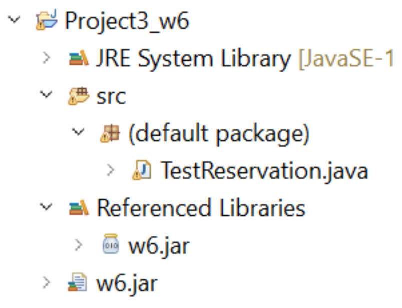
- "Jan 02, 2025", "Jan 05, 2025"

And then the expected results should use the following similar methods to compare and output pass or fail:

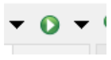
- Expected and actual should be: 3   And it's a PASS.

Rubric Criteria:
Execute, using w6.jar, unit tests for the calculateReservationNumberOfDays() method of the Reservation class. Document the unit tests code and results via screenshots 10%
Your Response:

Execution: Since I used Oracle, I allowed the w6.Jar as a referenced library using the properties and running it using the run debug, which should automatically use the classes in that w6.jar file. All I had to do was create a main file for the test cases, which is in the src file. The class is called TestReservation.java.



```
Project3_w6
  JRE System Library [JavaSE-1
  src
    (default package)
      TestReservation.java
  Referenced Libraries
    w6.jar
  w6.jar
```

← shows the w6.jar & to run the main.→ 

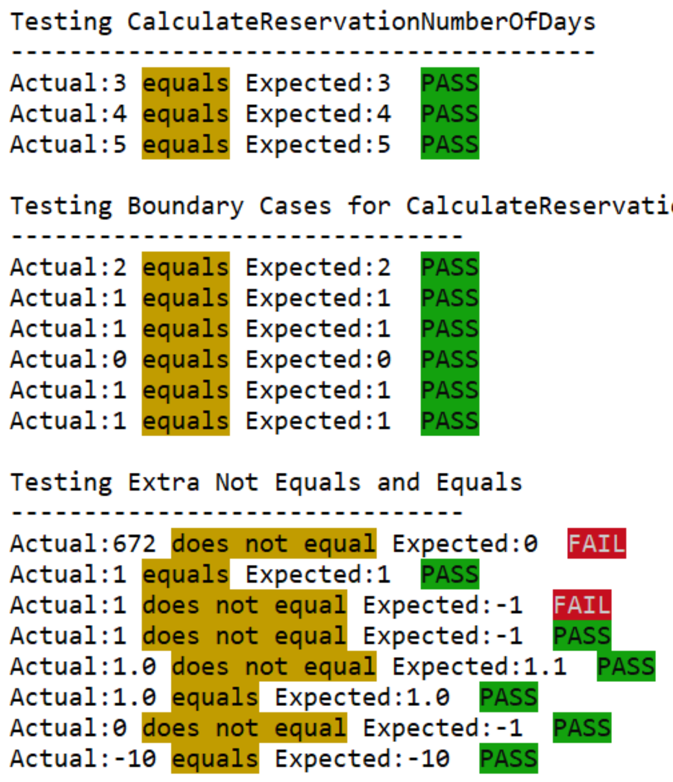
CODE:

```
/**
 * This method/function tests out CalculateReservationNumberOfDays for users' reservation at JJ's B&B.
 * It should incorporate black-box testing techniques and other checks.
 */
public static void testCalculateReservationNumberOfDays() throws Exception {
    System.out.println();
    System.out.println("Testing CalculateReservationNumberOfDays");
```

```
        System.out.println("----------------------------------------");

        // Equivalence Partitioning Test Cases

        Reservation r = new Reservation(1, "NormalRoom", "Jan 02, 2025", "Jan 05, 2025");

        long numberOfDays = r.calculateReversationNumberOfDays();

        Assert.assertEqualsInt((int) numberOfDays, 3);


        Reservation r2 = new Reservation(2, "RoomWBath", "Mar 01, 2025", "Mar 05, 2025");

        long numberOfDays2 = r2.calculateReversationNumberOfDays();

        Assert.assertEqualsInt((int) numberOfDays2, 4);


        Reservation r3 = new Reservation(3, "RoomWView", "Apr 15, 2025", "Apr 20, 2025");

        long numberOfDays3 = r3.calculateReversationNumberOfDays();

        Assert.assertEqualsInt((int) numberOfDays3, 5);


        System.out.println();

        System.out.println("Testing Boundary Cases for CalculateReservationNumberOfDays");

        System.out.println("------------------------------");

        // Boundary Value Analysis Test Cases

        Reservation r4 = new Reservation(4, "RoomWBath", "Feb 28, 2024", "Mar 01, 2024"); // Leap Year

        long numberOfDays4 = r4.calculateReversationNumberOfDays();

        Assert.assertEqualsInt((int) numberOfDays4, 2);


        Reservation r5 = new Reservation(5, "NormalRoom", "Dec 31, 2024", "Jan 01, 2025"); // Year
    Boundary

        long numberOfDays5 = r5.calculateReversationNumberOfDays();

        Assert.assertEqualsInt((int) numberOfDays5, 1);


        Reservation r6 = new Reservation(6, "RoomWView", "Jan 01, 2025", "Jan 02, 2025");

        long numberOfDays6 = r6.calculateReversationNumberOfDays();
        Assert.assertEqualsInt((int) numberOfDays6, 1);


        Reservation r7 = new Reservation(7, "RoomWView", "Jan 01, 2025", "Jan 01, 2025"); // Same Start
    and End Date
        long numberOfDays7 = r7.calculateReversationNumberOfDays();
        Assert.assertEqualsInt((int) numberOfDays7, 0);


        Reservation r8 = new Reservation(8, "NormalRoom", "Feb 28, 2021", "Mar 01, 2021"); // Non-Leap
    Year
        long numberOfDays8 = r8.calculateReversationNumberOfDays();
        Assert.assertEqualsInt((int) numberOfDays8, 1);


        Reservation r9 = new Reservation(9, "RoomWView", "Feb 29, 2020", "Mar 01, 2020"); // Leap Day
    on Leap Year
        long numberOfDays9 = r9.calculateReversationNumberOfDays();
        Assert.assertEqualsInt((int) numberOfDays9, 1);


        System.out.println();
        System.out.println("Testing Extra Not Equals and Equals");
        System.out.println("-------------------------------");
        // Testing out if it will throw an exception if inputed invalid dates when changed or updated.
        // It should test even if error will i calculate as well.
        Reservation r10 = new Reservation(10, "RoomWView", "Feb 29, 2020", "Mar 01, 2020"); // Leap
    Day on Leap Year
        r10.setRoom("RoomWBath");
        r10.setReservationStartDate("Dec 1, 2024");
        r10.setReservationEndDate("Jan 01, 2022");
        long numberOfDays10 = r10.calculateReversationNumberOfDays();
        // This tells me that the code has bee incorrectly counting the days.
```

```
        // This is because the actual 672 is not 0 but a starting date cannot be in the future of the end date
(start must be before end).

        Assert.assertEqualsInt((int) numberOfDays10, 0);

        //Testing negative values and some floats in the days to throw exceptions.

        Assert.assertEqualsInt((int) numberOfDays9, 1);

        Assert.assertEqualsInt((int) numberOfDays9, -1);

        Assert.assertNotEqualsInt((int) numberOfDays9, -1);

        Assert.assertNotEqualsDouble(numberOfDays9, 1.1);

        Assert.assertEqualsDouble(numberOfDays9, 1.0);

        // Test invalid input of day, negative day number

        Reservation r11 = new Reservation(-10, "RoomWView", "Feb 29, 2023", "Mar 01, 2023");

        long numberOfDays11 = r11.calculateReversationNumberOfDays();

        Assert.assertNotEqualsInt((int) numberOfDays11, -1);

        Assert.assertEqualsInt(r11.getGuestID(), -10);

    }
```

OUTPUT TESTING:



```
Testing CalculateReservationNumberOfDays
----------------------------------------
Actual:3 equals Expected:3  PASS
Actual:4 equals Expected:4  PASS
Actual:5 equals Expected:5  PASS

Testing Boundary Cases for CalculateReservationNumberOfDays
-------------------------------
Actual:2 equals Expected:2  PASS
Actual:1 equals Expected:1  PASS
Actual:1 equals Expected:1  PASS
Actual:0 equals Expected:0  PASS
Actual:1 equals Expected:1  PASS
Actual:1 equals Expected:1  PASS

Testing Extra Not Equals and Equals
-------------------------------
Actual:672 does not equal Expected:0  FAIL
Actual:1 equals Expected:1  PASS
Actual:1 does not equal Expected:-1  FAIL
Actual:1 does not equal Expected:-1  PASS
Actual:1.0 does not equal Expected:1.1  PASS
Actual:1.0 equals Expected:1.0  PASS
Actual:0 does not equal Expected:-1  PASS
Actual:-10 equals Expected:-10  PASS
```

Rubric Criteria:
Explain approach, steps, and rationale of the test cases and unit tests of testing the calculateReservationNumberOfDays() method of the Reservation class 5%
Your Response:

Approach:

Before testing this section and creating the code, I had to read the required documents and scenarios for this case. I included the demo testing while adding more test cases. Then I used the black-box techniques of equivalence partitioning and boundary value analysis to create these test cases. When applying the concepts to the code, it was testing cases 1-3 for equivalence partitioning. And for boundary value analysis it was test cases 4-9. The 10 were extra not equal and equal test cases. To approach this method, I created tests of the method to ensure it correctly calculates the number of reservation days between the start and end dates. I even made sure to test if it would detect invalid dates for start and end. This means it should not calculate dates if the start is the future of the end. By using these techniques, we ensure that the CalculateReservationNumberofDays() is efficient and correctly calculates the days.

Steps:

1. Define the two concepts in the test cases and when creating them.
    a. Equivalence Partitioning:
        i. For inputs: Various date ranges for reservations.
        ii. For Test cases: Verify the method correctly calculates the number of reservation days for different valid date ranges.
    b. Boundary Value Analysis
        i. For inputs: Edge cases such as end date one day after the start date.
        ii. For Test cases: Create test cases to ensure accurate calculation for minimal date differences.
2. Creating Objects: Create a Reservation object with known start and end dates.
3. Call the calculateReversationNumberOfDays() method.
4. Creating assertions: Use assertions to check if the returned number of days matches the expected result and actual.
5. Test cases creations (EP: 1-3; BVA: 4-9):
    a. Test Case #1: Verifies the method calculates the number of days for a typical 3-day reservation period.
    b. Test Case #2: Confirms that the method handles a 4-day reservation period correctly, ensuring the calculation includes all intermediate days.
    c. Test Case #3: Ensures the method accurately calculates a 5-day reservation period.
    d. Test Case #4: Verifies the method correctly accounts for leap years, including February 29th.
    e. Test Case #5: Checks the method's handling of reservations spanning the end of one year and the beginning of the next.
    f. Test Case #6: Validates the method's handling of a 1-day reservation, ensuring it calculates the difference correctly.

g. Test Case #7: Ensures the method correctly returns 0 days when the start date and end date are the same.
h. Test Case #8: Checks the method's handling of dates around February 28th in a non-leap year, ensuring proper day count.
i. Test Case #9: Verifies that the method correctly includes February 29th when it occurs in a leap year.
j. Test Case #10: Tests out the equal and not equal that I missed.

Rationale:

First, I had to comprehend the black-box techniques for this method. According to Tsui, Karam, & Bernal (2014), GeeksforGeeks (2024), and Javapoint (n.d.), black-box testing is a methodology where the test cases are mostly derived from the requirements statements without considering the actual code content. The most common specification-based technique is equivalence-class partitioning. This is where the input for the software system is divided into several equivalence classes for which the software should behave similarly, generating one test case for each class. It focuses on testing within and at the boundaries of these classes. Equivalence Partitioning testing divides the input data of the software unit into partitions of equivalent data, where each partition represents a set of values that should be treated similarly by the software. I made sure to incorporate the steps into this process. This is crucial because it reduces the total number of test cases while ensuring coverage of possible inputs. Then for Boundary value analysis, it focuses on the boundaries between partitions. I included that the test must be at the edge of each partition, where errors are most likely to occur, including testing just inside and just outside the boundaries of input ranges. When applying the concepts to the code, it was testing cases 1-6 for equivalence partitioning. And for boundary value analysis it was test cases 7-9. Other lessons learned are through the test cases defined previously. It verifies different cases, and it is important because without it the software cannot be efficient for user usage if there are errors. I made sure to go back and check the requirements of all the other documents provided. By using these techniques, we ensure and verify that the CalculateResevationNumberOfDays() accurately calculates the duration of the reservation (Tsui, Karam, & Bernal, 2014; GeeksforGeeks, 2024; Javapoint, n.d.).

Rubric Criteria:
Create black-box test cases to test the calculateReservationBillAmount() method of the Reservation class 8%
Your Response:

The code provided is using a combination of black-box testing techniques, primarily equivalence partitioning and boundary value analysis.

Test Case 4: calculateReservationBillAmount()

Test case #	Selected Inputs	Expected Result	Actual Result	Pass | Fail

1	// Test case for RoomWBath with many days
	    Reservation r1 = new Reservation(1, "RoomWBath", "Jan 03, 2025", "Jan 06, 2025");
	    double billAmount1 = r1.calculateReservationBillAmount();
	
	Assert.assertEqualsDouble(billAmount1, 600.0); // 3 days * $200 = $600
	
	Assert.assertNotEqualsDouble(billAmount1, 600.0);
		EqualsDouble: 600.0
		NotEqualsDouble: 600.0
			EqualsDouble: 0.0
			NotEqualsDouble: 0.0
				Fail
				Pass
				
				1st Fail: Fails because the code must have made a mistake that the room should be 200 *3 which is 600. One night is 200 dollars for RoomWBath.

2	// 1 day with RoomWBath
	    Reservation r4 = new Reservation(1, "RoomWBath", "Jan 01, 2025", "Jan 02, 2025");
	    double billAmount4 = r4.calculateReservationBillAmount();
	
	Assert.assertEqualsDouble(billAmount4, 200); // 1 day * $200 = $200
	
	Assert.assertNotEqualsDouble(billAmount4, 200);
		EqualsDouble: 200.0
		NotEqualsDouble: 200.0
			EqualsDouble: 0.0
			NotEqualsDouble: 0.0
				Fail
				Pass

3	// Test case for RoomWView
	    Reservation r2 = new Reservation(2, "RoomWView", "Feb 01, 2025", "Feb 04, 2025");
	    double billAmount2 = r2.calculateReservationBillAmount();
	
	Assert.assertEqualsDouble(billAmount2, 525); // 3 days * $175 = $525
	
	Assert.assertNotEqualsDouble(billAmount2, 525);
		EqualsDouble: 525.0
		NotEqualsDouble: 525.0
			EqualsDouble: 525.0
			NotEqualsDouble: 525.0
				Pass
				Fail
				
				1st Fail: Fails as 525 does indeed equal to 525.

4	// 1 day with RoomWView
	    Reservation r5 = new Reservation(1, "RoomWView", "Feb 01, 2025", "Feb 02, 2025");
	    double billAmount5 = r5.calculateReservationBillAmount();
		EqualsDouble: 175.0
		NotEqualsDouble: 175.0
			EqualsDouble: 175.0
			NotEqualsDouble: 175.0
				Pass
				Fail
				
				1st Fail: Fails as 175 does indeed equal to 175.

	Assert.assertEqualsDouble(billAmount5, 175); // 1 day * $175 = $175 Assert.assertNotEqualsDouble(billAmount5, 175);			

5	// Test case for NormalRoom
	Reservation r3 = new Reservation(3, "NormalRoom", "Mar 01, 2025", "Mar 05, 2025");
	double billAmount3 = r3.calculateReservationBillAmount();
	
	Assert.assertEqualsDouble(billAmount3, 500); // 4 days * $125 = $500
	
	Assert.assertNotEqualsDouble(billAmount3, 500);
		EqualsDouble: 500.0
		NotEqualsDouble: 500.0
			EqualsDouble: 480.0
			NotEqualsDouble: 480.0
				Fail
				Pass
				
				1st Fail: Fails because the code must have made a mistake that the room should be 125 *4 which is 500. One night is 125 dollars for NormalRoom

6	// 1 day with NormalRoom
	Reservation r6 = new Reservation(6, "NormalRoom", "Mar 01, 2025", "Mar 02, 2025");
	double billAmount6 = r6.calculateReservationBillAmount();
	
	Assert.assertEqualsDouble(billAmount6, 125); // 1 day * $125 = $125
	
	Assert.assertNotEqualsDouble(billAmount6, 125);
		EqualsDouble: 125.0
		NotEqualsDouble: 125.0
			EqualsDouble: 120.0
			NotEqualsDouble: 120.0
				Fail
				Pass
				
				1st Fail: This fails because the code incorrectly sets the NormalRoom value to 120 when it should be 125. According to the requirements, one room should be 125.

Notes for this table: The two classes provided test the cases based on the requirements and instructions, it explains that the expected results should use the following methods to check it and compare it. Then once it is compared it should show the pass or fail in the output. This note is just to explain more in-depth as the table does not include the details about the methods that are involved when comparing it. For instance, in one of the examples the selected input was:

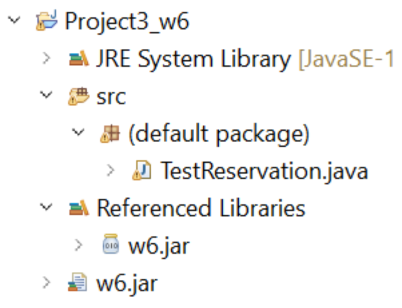
- 125, 125 comparisons.

And then the expected results should use the following similar methods to compare and output pass or fail:

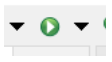
- Expected"125", Actual "120" This would fail as the coder set it to 120 when it should be 125 for one night in a NormalRoom.

Rubric Criteria:
Execute, using w6.jar, unit tests for the calculateReservationBillAmount() method of the Reservation class. Document the unit tests code and results via screenshots 10%
Your Response:

Execution: Since I used Oracle, I allowed the w6.Jar as a referenced library using the properties and running it using the run debug which should automatically use the classes in that w6.jar file. All I ha dot do was create a main file for the testcases which is in the src file. The class is called TestReservation.java.


```
Project3_w6
  JRE System Library [JavaSE-1
  src
    (default package)
      TestReservation.java
  Referenced Libraries
    w6.jar
  w6.jar
```

← shows the w6.jar & to run the main.→



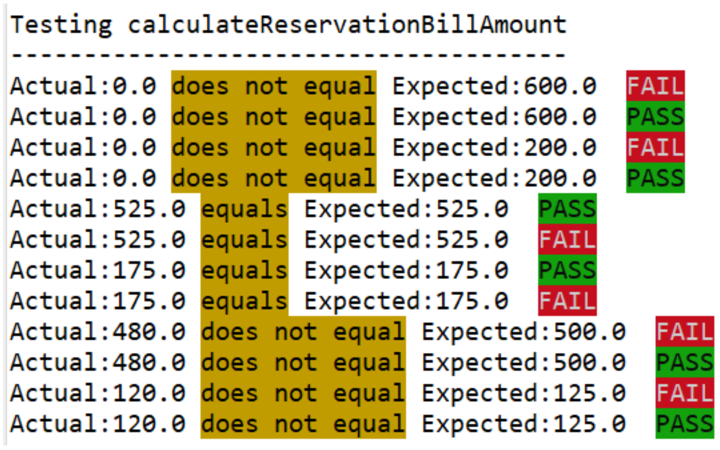
CODE:

```
/**
 * This method/function tests out CalculateReservationBillAmount for users' reservation at JJ's B&B.
 * It should incorporate black-box testing techniques and other checks.
 */
public static void testCalculateReservationBillAmount() throws Exception {
    System.out.println();
    System.out.println("Testing calculateReservationBillAmount");
    System.out.println("--------------------------------------");
    // Equivalence Partitioning Test Cases


    // Test case for RoomWBath with many days
```

```
Reservation r1 = new Reservation(1, "RoomWBath", "Jan 03, 2025", "Jan 06, 2025");

double billAmount1 = r1.calculateReservationBillAmount();

Assert.assertEqualsDouble(billAmount1, 600.0);  // 3 days * $200 = $600

Assert.assertNotEqualsDouble(billAmount1, 600.0);


// 1 day with RoomWBath

Reservation r4 = new Reservation(1, "RoomWBath", "Jan 01, 2025", "Jan 02, 2025");

double billAmount4 = r4.calculateReservationBillAmount();

Assert.assertEqualsDouble(billAmount4, 200);  // 1 day * $200 = $200

Assert.assertNotEqualsDouble(billAmount4, 200);


// Test case for RoomWView

Reservation r2 = new Reservation(2, "RoomWView", "Feb 01, 2025", "Feb 04, 2025");

double billAmount2 = r2.calculateReservationBillAmount();

Assert.assertEqualsDouble(billAmount2, 525);  // 3 days * $175 = $525

Assert.assertNotEqualsDouble(billAmount2, 525);


// 1 day with RoomWView

Reservation r5 = new Reservation(1, "RoomWView", "Feb 01, 2025", "Feb 02, 2025");

double billAmount5 = r5.calculateReservationBillAmount();

Assert.assertEqualsDouble(billAmount5, 175);  // 1 day * $175 = $175

Assert.assertNotEqualsDouble(billAmount5, 175);


// Test case for NormalRoom

Reservation r3 = new Reservation(3, "NormalRoom", "Mar 01, 2025", "Mar 05, 2025");

double billAmount3 = r3.calculateReservationBillAmount();

Assert.assertEqualsDouble(billAmount3, 500);  // 4 days * $125 = $500

Assert.assertNotEqualsDouble(billAmount3, 500);
```

```
        // 1 day with NormalRoom

        Reservation r6 = new Reservation(6, "NormalRoom", "Mar 01, 2025", "Mar 02, 2025");

        double billAmount6 = r6.calculateReservationBillAmount();

        Assert.assertEqualsDouble(billAmount6, 125);  // 1 day * $125 = $125

        Assert.assertNotEqualsDouble(billAmount6, 125);

    }
```

OUTPUT TESTING:



```
Testing calculateReservationBillAmount
--------------------------------------
Actual:0.0 does not equal Expected:600.0  FAIL
Actual:0.0 does not equal Expected:600.0  PASS
Actual:0.0 does not equal Expected:200.0  FAIL
Actual:0.0 does not equal Expected:200.0  PASS
Actual:525.0 equals Expected:525.0  PASS
Actual:525.0 equals Expected:525.0  FAIL
Actual:175.0 equals Expected:175.0  PASS
Actual:175.0 equals Expected:175.0  FAIL
Actual:480.0 does not equal Expected:500.0  FAIL
Actual:480.0 does not equal Expected:500.0  PASS
Actual:120.0 does not equal Expected:125.0  FAIL
Actual:120.0 does not equal Expected:125.0  PASS
```

Rubric Criteria:
Explain approach, steps, and rationale of the test cases and unit tests of testing the calculateReservationBillAmount() method of the Reservation class 5%
Your Response:

Approach:

Before testing this section and creating the code, I had to read the required documents and scenarios for this case. I included the demo testing while adding more test cases. Then I used the black-box techniques of equivalence partitioning and boundary value analysis to create these test cases. When applying the concepts to the code, it was testing cases 1-6 for equivalence partitioning and boundary value analysis. To approach this method, I created tests of the method to ensure it correctly calculates the number of reservation days between the start and end dates. I even made sure to test if it would detect invalid dates for start and end. This means it should not calculate dates if the start is the future of the end. By using these techniques, we ensure that the CalculateReservation BillAmount() is efficient and correctly calculates the days.

Steps:

1. Define the two concepts in the test cases and when creating them.
    a. Equivalence Partitioning:
        i. For inputs: Different combinations of room types and reservation durations.

        ii. For Test cases: Verify the method calculates the correct bill amount based on the daily rate and number of days.
    b. Boundary Value Analysis
        i. For inputs: Edge values for the shortest and longest possible stays.
        ii. For Test cases: Check calculations for minimal and maximal valid reservation periods.
2. Creating Objects: Create a Reservation object with known start and end dates.
3. Call the calculateReversationBillAmount() method.
4. Creating assertions: Use assertions to check if the returned bill amount matches the expected result and actual.
5. Test cases creations (EP: 1-3; BVA: 4-9):
    a. Test Case #1: Verifies that 200 * 3 is 600 which is the correct calculation.
    b. Test Case #2: Verifies that 200 per night is RoomWBath which is the correct pricing.
    c. Test Case #3: Verifies that 175 * 3 is 525 which is the correct calculation.
    d. Test Case #4: Verifies that 175 per night is RoomWView which is the correct pricing.
    e. Test Case #5: Verifies that 125 * 4 is 480 which is the correct calculation.
    f. Test Case #6: Verifies that 125 per night is NormalRoom which is the correct pricing.

Rationale:

First, I had to comprehend the black-box techniques for this method. According to
Tsui, Karam, & Bernal (2014), GeeksforGeeks (2024), and Javapoint (n.d.),

black-box testing is a methodology where the test cases are mostly derived from the requirements statements without considering the actual code content. Equivalence Partitioning testing divides the input data of the software unit into partitions of equivalent data, where each partition represents a set of values that should be treated similarly by the software. Again, the main purpose of this method is to divide the input domain into a smaller number of equivalence classes and to design test cases that cover each equivalence class. I made sure to incorporate the steps into this process. This is crucial because it reduces the total number of test cases while ensuring coverage of possible inputs. Then for Boundary value analysis, it focuses on the boundaries between partitions. I included that the test must tests at the edge of each partition, where errors are most likely to occur, including testing just inside and just outside the boundaries of input ranges. When applying the concepts to the code, it was testing cases 1-6 for equivalence partitioning and boundary value analysis. I learned that the coding must be somewhat incorrect as the pricing for the rooms does not match the calculations. Only one passed by the other two had been set up incorrectly which caused the total pricing to not calculate it accurately. I also double-checked the requirements and made sure that the test cases mentioned previously verified all scenarios. By using these techniques, we ensure and verify that the CalculateResevationBillAmount() accurately calculates the bill amount of the reservation including the total amount by the days spent there (Tsui, Karam, & Bernal, 2014; GeeksforGeeks, 2024; Javapoint, n.d.).

Rubric Criteria:
Reflect on the learning experience and lessons learned 8%
Your Response:

I learned that black-box functional unit test cases can help ensure that the software is efficient and does not have errors. The tests provided use a combination of black-box testing techniques, primarily equivalence partitioning and boundary value analysis. According to Tsui, Karam, & Bernal (2014), GeeksforGeeks (2024), and Javapoint (n.d.), black-box testing is a methodology where the test cases are mostly derived from the requirements statements without considering the actual code content. For instance, testing based solely on specifications, without looking at the code is commonly called black-box testing. The most common specification-based technique is equivalence-class partitioning. Equivalence Partitioning divides the input data of a software unit into partitions of equivalent data from which test cases can be derived. The goal is to ensure that the test cases cover all possible inputs, reducing the number of test cases while still providing adequate coverage. Meanwhile, Boundary Value Analysis focuses on the boundaries between partitions. It involves testing at the edge of each partition, which is where errors are most likely to occur. This project taught me to use table testing as well to represent the input conditions and actions. These black-box testing techniques (equivalence partitioning and boundary value analysis) help cover a wide range of input scenarios while ensuring the Reservation class methods work as expected. Moreover, comments are important for programmers because it affect the readability and maintainability of the code. Programmers must comprehend this concept because incorrect comments can lead to complexity, wrong comprehension of the code, outdated, and hard for other programmers to understand the code when dealing with complex and larger projects. By methodically testing various partitions and boundaries, I can validate that the implementation handles all relevant cases correctly without accessing the internal code structure. This helps create test cases similar to real-world usage scenarios for the reservation system. This assignment offers valuable lessons in software development through black-box unit testing of the Reservation class. Additionally, understanding the specifications of both the Reservation class and the Assert class is crucial for creating effective test cases. The documentation clarifies assumptions made by the Reservation class (e.g., valid input, date order), influencing the test case design. And each test case defines the Selected Inputs which are the data provided to the class methods, expected result, actual result, and pass/fail. The Expected Result is the anticipated output based on the input. The Actual Result is the outcome obtained by executing the method the Pass/Fail verifies if the actual result matches the expected outcome. In this project, I learned to do a testing design case and learn to test different functionalities for this software. I noticed that the software code has some issues that should be fixed through these tests and incorporating the NotEquals and Equals to compare the results. For instance, testing covers various aspects of the Reservation class of the Constructor and getters, setters and getters, calculateReservationNumberOfDays(), and calculateReservationBillAmount(). The first section ensures proper object creation and data retrieval. The second section verifies setting and retrieving values correctly. The third section tests the calculation of the reservation duration. The fourth section confirms the accurate bill amount based on room type and duration. By identifying partitions, I am allowed to reduce the number of test cases needed to achieve comprehensive coverage. Instead of testing every possible input, you only need to test one value from each partition. Boundary value analysis helps identify and focus on critical areas where errors are most likely to occur. I learned that the benefits of unit testing allow me to identify potential errors in isolated class functionality, code reliability, and maintain correct behavior with future modifications. I learned that this exercise demonstrates the importance of test-driven development and detecting errors to ensure an efficient final software product. Furthermore, this project allowed me to create and understand different test case types and apply the concepts in software engineering. Therefore, through the textbook and online, black-box testing techniques are essential in software engineering for creating reliable and user-friendly software products, by detecting errors early to improve the quality and reliability of the software, especially in projects like reservation systems where accuracy and reliability are paramount. By learning these valuable lessons, developers can improve their software development practices through black-box unit testing and ensure high-quality, reliable applications (Tsui, Karam, & Bernal, 2014; GeeksforGeeks, 2024; Javapoint, n.d.).

References

GeeksforGeeks. (2024). *Boundary Value Analysis vs Equivalence Partitioning*. GeeksforGeeks. https://www.geeksforgeeks.org/software-testing-boundary-value-analysis-vs-equivalence-partitioning/

Javatpoint. (n.d.). *Black box testing*. Javatpoint. https://www.javatpoint.com/black-box-testing

Tsui, F., Karam, O., & Bernal, B. (2014). *Essentials of software engineering: Chapter 9* (3rd ed.). Jones and Bartlett Learning. https://library-books24x7-com.ezproxy.umgc.edu/toc.aspx?site=VGX8U&bookID=51648