Name: Jade Pearl  
Date: 2/17/2024  
Course Code – CMSC 345 6832 – Chuck Anyanso  
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):

//Unit test Java Source code for testing the Constructor & getters methods of the Reservation Class

//Used from the example given by UMGC in the assignment 3 instructions and edited by Jade Pearl

//for further program testing.

//CMSC 345 Assignment 3

//Last updated: 2/20/2024

import java.util.\*;

import java.lang.module.ResolutionException;

import java.text.SimpleDateFormat;

public class TestReservation {

private static String datePattern = "MMM dd, yyyy";

private static SimpleDateFormat sdf = new SimpleDateFormat(datePattern);

public static void main(String argv[]) throws Exception {

testContructorAndGetters();

System.out.println();

testSettersAndGetters();

System.out.println();

testNumofDays();

System.out.println();

testBillAmount();

}

public static void testContructorAndGetters() {

System.out.println();

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

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

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(), r2.getGuestID());

Assert.assertEqualsString(r.getRoomType(), r2.getRoomType());

Assert.assertNotEqualsString(r.getReservationStartDate(), r2.getReservationStartDate());

Assert.assertEqualsString(r.getReservationEndDate(), r2.getReservationEndDate());

}

public static void testSettersAndGetters() {

System.out.println();

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

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

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

//change the values with the setters for testing

r.setGuestID(3);

r.setRoom("Normal Room");

r.setReservationStartDate("Feb 29, 2024");

r.setReservationEndDate("Mar 7, 2024");

//Make a separate reservation for comparison. I used the constructor for convenience

Reservation r2 = new Reservation(2, "RoomWView", "Feb 29, 2024", "Mar 3, 2024");

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

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

Assert.assertNotEqualsInt(r.getGuestID(), r2.getGuestID());

Assert.assertNotEqualsString(r.getRoomType(), r2.getRoomType());

Assert.assertEqualsString(r.getReservationStartDate(), r2.getReservationStartDate());

Assert.assertNotEqualsString(r.getReservationEndDate(), r2.getReservationEndDate());

}

public static void testNumofDays() {

System.out.println();

System.out.println("Testing calculateReservationNumOfDays()");

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

Reservation r = new Reservation(1, "RoomWView", "Feb 29, 2024", "Mar 7, 2024");

Reservation r2 = new Reservation(2, "RoomWBath", "Feb 29, 2024", "Mar 3, 2024");

//make variables to store the number of days in the reservations.

try {

Assert.assertNotEqualsLong(r.calculateReversationNumberOfDays(), r2.calculateReversationNumberOfDays());

Assert.assertEqualsLong(r.calculateReversationNumberOfDays(), r.calculateReversationNumberOfDays());

} catch (Exception e) {

e.printStackTrace();

}

}

public static void testBillAmount() {

System.out.println();

System.out.println("Testing calculateReservationNumOfDays()");

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

Reservation r = new Reservation(1, "RoomWView", "Feb 29, 2024", "Mar 7, 2024");

Reservation r2 = new Reservation(2, "RoomWBath", "Feb 29, 2024", "Mar 3, 2024");

Reservation r3 = new Reservation(3, "NormalRoom", "Feb 29, 2024", "Mar 7, 2024");

//make variables to store the number of days in the reservations.

try {

Assert.assertNotEqualsDouble(r.calculateReservationBillAmount(), r2.calculateReservationBillAmount());

Assert.assertEqualsDouble(r.calculateReservationBillAmount(), r.calculateReservationBillAmount());

Assert.assertNotEqualsDouble(r.calculateReservationBillAmount(), r3.calculateReservationBillAmount());

} catch (Exception e) {

e.printStackTrace();

}

}

}

-----------------------------------------------------------------------------------------------------------------------------------

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case # | Selected Input  custID,  roomType  startDate  endDate | Expected Results | Actual Results | Pass | Fail |
| 1: Taken from the example given in the assignment instructions | 1, “RoomWBath”, “Jun 16, 2022”, “Jun 19, 2022”  &  7, “RoomWBath,”, “Jun 16, 2022”, “Jun 19, 2022” | Two unique UUID guest IDs | Two UUID guest IDs that are not equal to each other | PASS |
| 2: Taken from the example given in the assignment instructions | 1, “RoomWBath”, “Jun 16, 2022”, “Jun 19, 2022” | Today’s date as a Java date type with milliseconds | Today’s date as a Java date type with milliseconds | FAIL due to difference in milliseconds  PASS when the opposite is used (assertNotEqualsDate) |
| 3 | 1, “RoomWBath”, “Jun 16, 2022”, “Jun 19, 2022”  &  7, “RoomWBath,”, “Jun 16, 2022”, “Jun 19, 2022” | Detects two reservations with the different Guest IDs | Two reservations with unique Guest IDs not equal to each other | FAIL due to the IDs not being equal. This is ***intentional*** because it is comparing the two customer IDs |
| 4 | 1, “RoomWBath”, “Jun 16, 2022”, “Jun 19, 2022”  &  7, “RoomWBath,”, “Jun 16, 2022”, “Jun 19, 2022” | Two of the same room type is set | Two room types that are EQUAL | PASS |
| 5 | 1, “RoomWBath”, “Jun 16, 2022”, “Jun 19, 2022”  &  7, “RoomWBath,”, “Jun 16, 2022”, “Jun 19, 2022” | Two of the same start dates for reservation | Reservation start dates that are EQUAL | FAIL. This is shows that the reservations are being placed at the same start date. This is an intentional fail because we are testing in the testing program if the start dates are NOT EQUAL which is clearly untrue because the start dates are the same! |
| 6 | 1, “RoomWBath”, “Jun 16, 2022”, “Jun 19, 2022”  &  7, “RoomWBath,”, “Jun 16, 2022”, “Jun 19, 2022” | Two of the same end dates for reservation | Reservation end dates that EQUAL each other | PASS – means the reservations end at the same date. |

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:

A computer screen shot of a program code

Description automatically generated

A computer screen with text and images

Description automatically generated

The first 20 lines were given as an example in the assignment instructions. Lines 21 – 24 were written by me to continue testing the getters. These results show that the constructors and getter functions of the Reservation class indeed work as intended.

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:

Black-box testing was a very new concept for me, so I tried to approach it in a careful manner. I carefully read over the specifications of both the Assert and Reservation classes before reading more of the assignment instructions to get examples of what a black-box test would look like for this scenario. I ended up taking the two examples given for the assignment and testing them to see how they worked and what they were testing. I then proceeded to make similar test cases that would ensure all getters would be tested from the Reservation class. The two given examples tested the getReservationID() and getReservationDate() functions which worked for the most part other than the millisecond issue that was found in the getReservationDate() function.

I knew the constructor was functional because of the data being stored when the constructor was called. It stored all information accurately for the getters to use. I intentionally threw in two calls that I knew would fail but still proved that the getters worked properly. In my example for the Guest IDs, I tried to see if they were EQUAL in my code. And since they were not equal, the test “FAILed”. This still proved the getGuestID worked because the Guest IDs were 1 and 7, unique numbers. I had a similar approach to the getReservationStartDate() function. I tested for them to be NOT EQUAL (which is untrue based on the reservation start date that was set for both Reservation items in the constructor) and the test “FAILed” again. The getter worked because the start dates for both Reservations are the same dates so saying they are NOT EQUAL is supposed to fail by standard. I figured this would be an interesting way to test whether the getters work.

-----------------------------------------------------------------------------------------------------------------------------------

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case # | Selected Input  custID,  roomType  startDate  endDate | Expected Results | Actual Results | Pass | Fail |
| 1 | 3, “Normal Room”, “Feb 29, 2024”, “Mar 7, 2024” (made with setters)  &  2, “RoomWView”, “Feb 29, 2024”, “Mar 3, 2024” | Two unique UUID reservation IDs | Two UUIDs NOT EQUAL | PASS |
| 2 | 3, “Normal Room”, “Feb 29, 2024”, “Mar 7, 2024” (made with setters) | Today’s date as a Java date type with milliseconds | Today’s date as a Java date type with milliseconds | FAIL due to millisecond problem as explained in the first test case set |
| 3 | 3, “Normal Room”, “Feb 29, 2024”, “Mar 7, 2024” (made with setters)  &  2, “RoomWView”, “Feb 29, 2024”, “Mar 3, 2024” | Two unique guest IDs | Two guest ID’s NOT EQUAL | PASS |
| 4 | 3, “Normal Room”, “Feb 29, 2024”, “Mar 7, 2024” (made with setters)  &  2, “RoomWView”, “Feb 29, 2024”, “Mar 3, 2024” | Two unique room types | Two room types NOT EQUAL | PASS |
| 5 | 3, “Normal Room”, “Feb 29, 2024”, “Mar 7, 2024” (made with setters)  &  2, “RoomWView”, “Feb 29, 2024”, “Mar 3, 2024” | Two of the same reservation start date | Two reservation start dates NOT EQUAL | FAIL, setter might not work. Different start date detected for reservation 1 |
| 6 | 3, “Normal Room”, “Feb 29, 2024”, “Mar 7, 2024” (made with setters)  &  2, “RoomWView”, “Feb 29, 2024”, “Mar 3, 2024” | Two unique reservation end dates | Two reservation end dates NOT EQUAL | PASS |

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

A computer screen shot of a program code

Description automatically generatedA computer screen with text and images

Description automatically generated

According to the test and results, the program kept the start date initialized with the constructor for Reservation r. The setReservationStartDate() function does not work properly because the value set did not overwrite the value set beforehand by the Constructor.

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:

I approached this very similarly to how I tested the Constructor and Getters in the first part of the assignment. To use the setters, I had to initialize the first reservation item using the Constructor since there is no parameter-less Constructor in the Reservation class. After I set “dummy” values (they are the values used in the previous input set. After that, I used the setters to change the values of the first reservation. I then made a second reservation item using the constructor for comparison since I did not need to use the setters a second time and I know that everything in the constructor works. As I tested, I made sure to test things that are true (i.e. Guest IDs are not equal, start and end dates are the same, and room types are different). If that test failed, I knew that there was something wrong with the setter or getter associated with that variable. I assumed there was something wrong with the setter for the reservation start date because the getters worked in the previous test case set.

NOTE: The setGuestID was written in the specifications as setCustomerID which does not exist if put into the code. It should be setGuestID instead.

-----------------------------------------------------------------------------------------------------------------------------------

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case # | Selected Input  custID,  roomType  startDate  endDate | Expected Results | Actual Results | Pass | Fail |
| 1 | 1, “RoomWView”, “Feb 29, 2024”, “Mar 7, 2024”  &  2, “RoomWBath”, “Feb 29, 2024”, “Mar 3, 2024” | Two unique numbers of days per reservation | Two numbers of days NOT EQUAL | PASS |
| 2 | 1, “RoomWView”, “Feb 29, 2024”, “Mar 7, 2024”  &  1, “RoomWView”, “Feb 29, 2024”, “Mar 7, 2024” | Two identical numbers of days per reservation | Two numbers of days EQUAL | 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:

A computer screen with colorful text

Description automatically generatedA screen shot of a computer

Description automatically generated

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:

This test was a lot shorter than the other two test case sets since it was only testing one method. I first wanted to make two reservations with start and end dates that made it easy to calculate the number of days each reservation had. I also made sure to include a try/catch statement in the testing code for the method because it throws an Exception if not used. I first compared the two different reservations, r and r2, since they did not have the same number of reservation dates. I saw that the program calculated the dates correctly and it passed when I was seeing that they were NOT EQUAL. To double check for the sake of covering all bases, I also included a statement that compared the r Reservation item to itself in the number of days method. Since it is comparing to itself, I knew that if it worked, it should test if it is EQUAL. This passed, so this test affirmed that it worked both ways, hence me drawing to a conclusion for this part of the test.

-----------------------------------------------------------------------------------------------------------------------------------

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case # | Selected Input  custID,  roomType  startDate  endDate | Expected Results | Actual Results | Pass | Fail |
| 1 | 1, “RoomWView”, “Feb 29, 2024”, “Mar 7, 2024”  &  2, “RoomWBath”, “Feb 29, 2024”, “Mar 3, 2024” | Two unique bill amounts | Two bill amounts NOT EQUAL | PASS – There is one issue though, the second room type is listed as a bill amount of 0 which should not be the case. |
| 2 | 1, “RoomWView”, “Feb 29, 2024”, “Mar 7, 2024”  &  1, “RoomWView”, “Feb 29, 2024”, “Mar 7, 2024” | Two identical bill amounts | Two bill amounts EQUAL | PASS |
| 3 | 1, “RoomWView”, “Feb 29, 2024”, “Mar 7, 2024”  &  3, “NormalRoom”, “Feb 29, 2024”, “Mar 7, 2024 | Two unique bill amounts | Two bill amounts NOT EQUAL | PASS |

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:

A computer screen shot of a program code

Description automatically generatedA screenshot of a computer screen

Description automatically generated

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:

After handling the exceptions from the previous test case set, this was a lot easier to approach for the bill amount method. For this case, I made three Reservation type items because there are three room types, all priced differently depending on the rate of days. We already know that the calculation of number of days is accurate, so we do not need to worry about calculating that before testing the bill amount seeing as the program uses the number of days automatically when calculating the bill. The program test passed in all three cases which you would think is a good sign. However, it seems that the program does NOT correctly calculate the bill for all three room types. The r2 item with type RoomWBath is written to the class specification guidelines and has a number of days of 3 days, yet the rate shows up as 0.0 in the test results. This value is incorrect as it should be 600.0 (200/day). Even though the testing passes the test no matter the comparison, the value is still inaccurate. The values calculated for the other two room types are correct and require no fixing. This is how I concluded that this method is only partly functional.

-----------------------------------------------------------------------------------------------------------------------------------

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

Through this assignment, I learned about the true essence and purpose of black-box testing. I understood that black-box testing implied no access to the implemented code itself, but testing from the specifications of the code that they are trying to analyze. Even though I had understanding of what it is, I had not fully understood HOW it was performed. There are different techniques for it, but making the actual test cases for it was a challenge at first because I had to become familiar through this assignment what the test cases would look like and how those cases are used to test the implemented code without knowing what is written in the code itself other than the how the methods are defined.

I learned that when you are testing and something fails, it is safe to assume that it is the fault of the program first. For example, when I was testing the setters and getters, I tested something that should have passed but it failed. When I made a reservation item r, I had to initialize it with the constructor, so I chose a start date of Jun 16, 2022 and then overwrote that value with the setReservationStartDate method. When it tested, I was shown that the start date did not get changed by that method, so I then knew that it was not something I had implemented incorrectly in my testing code, but the method being tested had a mistake in it somewhere.

I think this learning experience helped me increase my understanding of the vital role that testing has in software engineering, especially if someone is working on a bigger team where different members have different roles. You need to have decent knowledge on how testing software works in order for software development to be successful because it will give helpful feedback to the programmer on where their values might be incorrect. It will also help narrow down where an essential operation might be missing. In my previous example, the setter that failed may have needed to make sure to assign the new value being passed in to the value of the target member in that class that needs to be changed.