Comprehensive Test Pass Report

Preethi Balasubramaniam [balasubramaniampree@cityuniversity.edu](mailto:balasubramaniampree@cityuniversity.edu)

Jayna Mehta [mehtajayna@cityuniversity.edu](mailto:mehtajayna@cityuniversity.edu)

Yu-Che Liu [liuyuche@cityuniversity.edu](mailto:liuyuche@cityuniversity.edu)

Ashley Schonberg [schonbergashley@cityuniversity.edu](mailto:schonbergashley@cityuniversity.edu)

Team 4, CS750 Software Testing, Computer Science,

City University of Seattle

**Abstract**

In this report, we will use automated test tools to test the source code of four unique Java programs: Cal.java, Thermostat.java, TriType.java, and TmcTestMojo.java. For three of the programs, we will perform unit tests and document our results. We will use either Branch Coverage or Statement Coverage while executing these tests and report our findings in the test execution summary. Using JUnit, a framework that allows the user to write repeatable tests, we will then create automated tests that have three assertions and two faults for each program (Bechtold et al.). Finally, we will report any bugs found while testing and list them in a reporting tool we have created.

**Keywords:** Software testing, Java, unit tests, Branch Coverage, Junit, bugs

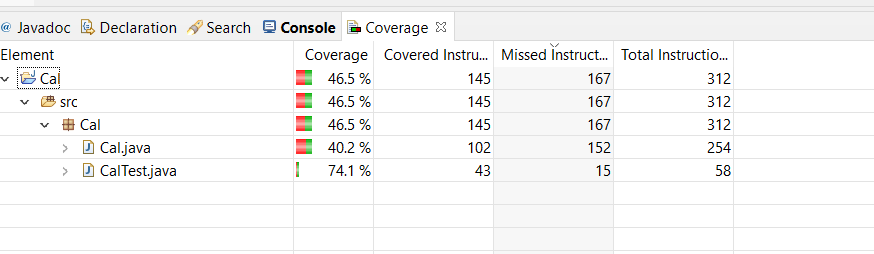
1. **CAL.JAVA**

**Code purpose**

The Cal.java program calculates the number of days between two user-provided dates within the same year. When a user runs this program, they start by entering the first month and day. Next, they enter the second month and day. After entering the dates, the user needs to provide the year as input. Finally, the program calculates the number of days between the two dates.

**Code coverage**

We will be using the statement coverage because the program is simple, and it does not require much more time to write the test statements. Following is the code coverage result. The program was able to achieve 46.5 %.



**Testing scope**

This program includes a main method, which requires the user input, and a class called Cal, which works the actual calculation. Therefore, we decided to automate all test cases. We choose not to perform any manual test cases because automated testing covers all the cases.

**Test cases**

We have set up the following test cases:

1. Test\_PositiveFlow()

This test case finds the number of the days between two months. (input 11,1,12,31,2020)

public void test\_PositiveFlow() {

        int numberOfDays = Cal.cal(11, 1, 12, 31, 2020);

        assertEquals(60, numberOfDays);

1. Test\_withLeapYear()

This test case finds the leap year using the given inputs. (input 2,1,12,31,2020)

public void test\_withLeapYear() {

        int numberOfDays = Cal.cal(2, 1, 12, 31,2020);

        assertEquals(334, numberOfDays);

    }

1. Test\_WithoutLeapYear()

This test case finds the number of the days without leap year. (input 2,1,12,31,2020)

public void test\_withoutLeapYear() {

        int numberOfDays = Cal.cal(2, 1, 12, 31, 2019);

        assertEquals(333, numberOfDays);

    }

1. Test\_wrongvaluewithdate()

This test case finds the wrong date means invalid value. (input 1,1,6,31,2020)

public void test\_wrongvaluewithdate(){

        int numberOfDays = Cal.cal(1,1,6,31,2020);

        assertEquals(181, numberOfDays);

    }

1. Test\_wrongvalue()

This test case finds the wrong month means invalid value. (input 12,1,12,31,2020)

public void test\_wrongvalue() {

        int numberOfDays = Cal.cal(13, 1, 12, 31, 2020);

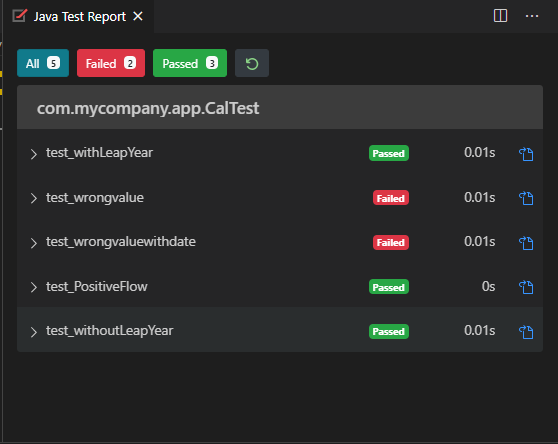
        assertEquals(365, numberOfDays);

    }

These tests fill the assignment requirements that we test at least three assertions (tests 1, 2, and 3) and two faults (tests 4 & 5).

**Test results**

The test results show that the three test cases passed, and two test cases fail because the invalid value.



**Bugs found**

There are some bugs in Cal.java. If each selected input is same, then the max date for that month is not considered. There is no check for February 31st. For these reasons, the 4th and 5th test cases fail. Following is the bug report:

Cal. Java bug report

|  |  |  |
| --- | --- | --- |
| Executed | Passed | 3 |
| Failed | 2 |
| Test executed (passed + failed) | 5 |
| Pending | 0 | |
| In progress | 0 | |
| Blocked | 0 | |
| Test planned | 5 | |

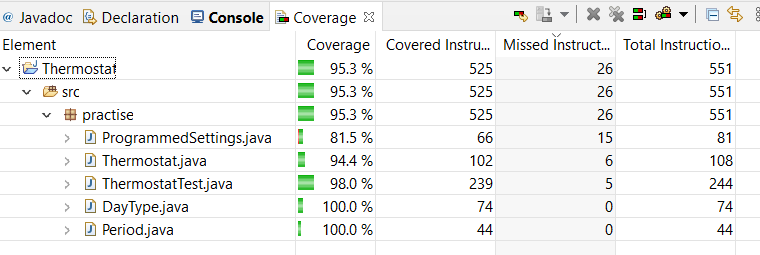
1. **THERMOSTAT.JAVA**

**Code purpose**

Thermostat.java is a class that controls a heating device. The user can set the temperature to change based on the day of the week. The class also provides some settings that allow the temperature to be overridden. The user can set the thermostat to have different, scheduled temperatures throughout a given day. If needed, they can also change the temperature settings to override any scheduled temperatures and immediately turn on the heater.

## Code Coverage

Following is the code coverage result. The program was able to achieve 95.3%. code coverage on Thermostat.java class under test.



**Testing scope**

This program appears to be complete enough to test the entire thing, so we will not be setting any parts of this code as out-of-scope. We plan to automate the testing using JUnit and Eclipse IDE.

**Test cases**

The following test cases will test that the heater turns on based on days planned:

1. **test1()**

setPeriod(Period.MORNING);

setDay(DayType.WEDNESDAY);

setCurrentTemp(50);

setThresholdDiff(2);

setOverride(true);

setOverTemp(80);

setTimeSinceLastRun(4);

setMinLag(2);

1. **test2()**

setPeriod(Period.NIGHT);

setDay(DayType.WEDNESDAY);

setCurrentTemp(50);

setThresholdDiff(2);

setOverride(true);

setOverTemp(80);

setTimeSinceLastRun(10);

setMinLag(2);

1. **test3()**

t.setPeriod(Period.MORNING);

t.setDay(DayType.WEDNESDAY);

t.setCurrentTemp(65);

t.setThresholdDiff(2);

t.setOverride(false);

t.setTimeSinceLastRun(4);

t.setMinLag(2);

1. **test4()**

t.setPeriod(Period.MORNING);

t.setDay(DayType.THURSDAY);

t.setCurrentTemp(50);

t.setThresholdDiff(2);

t.setTimeSinceLastRun(10);

t.setMinLag(5);

1. **test5()**

t.setPeriod(Period.NIGHT);

t.setDay(DayType.THURSDAY);

t.setCurrentTemp(50);

t.setThresholdDiff(2);

t.setTimeSinceLastRun(10);

t.setMinLag(5);

1. **test6()**

t.setPeriod(Period.NIGHT);

t.setDay(DayType.FRIDAY);

t.setCurrentTemp(60);

t.setThresholdDiff(2);

t.setOverride(true);

t.setOverTemp(80);

t.setTimeSinceLastRun(2);

t.setMinLag(10);

assertFalse(t.turnHeaterOn(pSet));

1. **testBug()**

t.setPeriod(Period.EVENING);

t.setDay(DayType.FRIDAY);

t.setCurrentTemp(60);

t.setThresholdDiff(2);

t.setOverride(true);

t.setOverTemp(80);

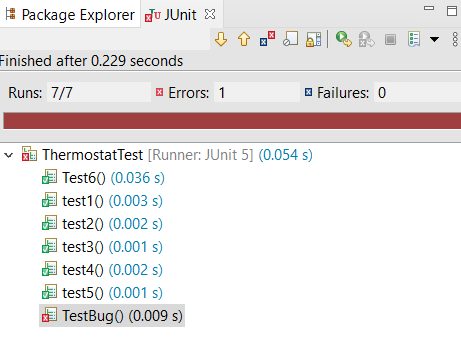
t.setTimeSinceLastRun(2);

t.setMinLag(4);

assertFalse(t.turnHeaterOn(null));

## Test Execution Result

We tried to execute the test cases by manually walking through the code and check to see if expected result would match the returned value. This was done due to the problem we had with automating test cases. Here are the test case execution results.



**Bugs found**

There is bug in Thermostat.java program. We also found that there is no error handling if the program setting is null. Test case testbug() failed due to this issue. The programmer would need to add error handling for these cases and gracefully handle them and rather than simply turning the heater on in these situations. Following is the bug report:

Thermostat Test Report

|  |  |  |
| --- | --- | --- |
| Executed | PASSED | 6 |
| FAILED | 1 |
| TESTS EXCEUTED (PASSED+FAILED) | 7 |
| PENDING | 0 | |
| In PROGESS | 0 | |
| BLOCKED | 0 | |
| TESTS PLANNED | 7 | |

**3.** **TRITYP.JAVA**

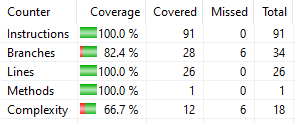
**Code purpose**

TriTyp.java is a program that asks a user to enter in the lengths of three sides of a triangle and uses that data to categorize the triangle as either a scalene, isosceles, equilateral, or an invalid triangle. It first prints the instructions and requests the inputs from the user. It takes the entered string and parses it to an integer. If the program is unable to read the input, it catches the exception and choses the number 1 as the input. It then begins to classify the type of triangle.

If any of the lengths entered are less than zero, the program determines that the shape entered is not a valid triangle. The program also checks to see if two sides added together are less than the length of the third side. If this happens, the program determines this is also not a triangle. If all sides are equal lengths, the program determines the triangle is equilateral. If only two lengths are the same, the triangle is isosceles, and if all lengths are different, the triangle is scalene.

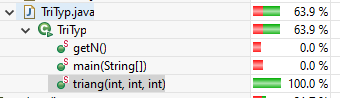
**Code coverage**

We used statement coverage in our JUnit tests for TriTyp.java.



**Testing scope**

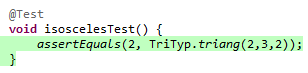
We have chosen to focus our testing on the triang() method even though the class has two additional methods. The main() method is the driver of the program but only has a few print statement lines of code to test. The getN() method catches exceptions, but the triang() method deals with those exceptions by returning an “invalid” response when provided these inputs. Therefore, we’ve focused our testing on the triang() method only and left the other two out of scope. We have automated the testing using JUnit and Eclipse.



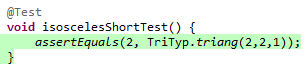
**Test cases**

We have set up the following test cases:

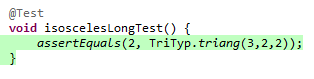
1. isoscelesTest()
   1. Tests to make sure isosceles triangle inputs result in an isosceles triangle result (inputs: 2, 3, 2)



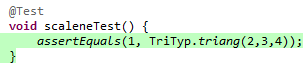
1. isoscelesShortTest()
   1. Tests a different order of lengths for an isosceles triangle. This one has two long sides and one short. (inputs: 2, 2, 1)



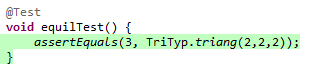
1. isoscelesLongTest()
   1. Tests a different order of lengths for an isosceles triangle in order to get full code coverage. This one has one long side and two shorter sides. (inputs: 3, 2, 2)



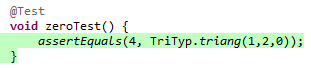
1. scaleneTest()
   1. Tests to make sure scalene triangle inputs result in a scalene triangle result (inputs: 2, 3, 4)



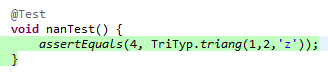
1. equilTest()
   1. Tests to make sure equilateral triangle inputs result in an equilateral triangle result (inputs: 2, 2, 2)



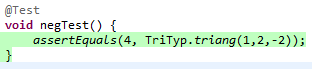
1. zeroTest()
   1. Tests to make sure that an invalid triangle is returned if the user enters a zero (inputs: 1, 2, 0)



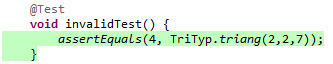
1. nanTest()
   1. Tests to make sure the program returns an invalid triangle if an input is not a number. (inputs: 1, 2, z)



1. negTest()
   1. Tests to make sure the program returns an invalid triangle if an input is a negative number. (inputs: 1, 2, -2)



1. invalidTest()
   1. Tests to make sure invalid inputs result in an invalid triangle result. We tested 2, 2, 7, because these are impossible side lengths for a valid triangle.

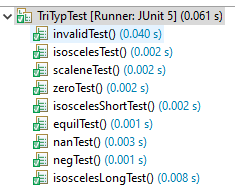


These tests fill the assignment requirements that we test at least three assertions (tests 2, 3, and 5) and two faults (tests 1 & 4).

During the setup of these tests, I received an error that the program’s triang() method was not a public method allowed for testing, so I did add “public” to line 30 of the original program for the purposes of testing.

**Test results**

Our test results show that all automated JUnit tests passed. In our initial round of testing, the nanTest() was throwing an error, but we determined this was due to isolating the test to the triang() method. In the previous report submission, we called the triang() method expecting to test the getN() method. It is important when conducting unit tests to make sure you are testing for each individual unit at a time, otherwise errors like this can occur. It was corrected for our final submission and the test passed as expected.



**Bugs found**

We did not encounter any bugs while testing the TriTyp.java class. Any initial bugs encountered were a result of testing errors.

If testing were to be expanded, one might decide to include edge testing for equilateral, isosceles, and scalene triangle values. However, the testing we completed covers the basics for the TriTyp.java class.

**4.** **TMCTESTMOJO.JAVA**

**Code purpose**

TmcTestMojo.java is one of the files inside the TestMyCode project. TestMyCode project is to provide a solution for instructor and student learning coding in a useful way. The instructors’ perspective can save time for students to face the tests as automated guidance, and the instructor can have more time to help students understand more difficult concepts. The students’ perspective can record what students do for the assignment and give feedback based on automatic testing results. In this session, it will focus on TmcTestMojo.java. The purpose of this file is run the test using TMC test runner. (Social Media and the Encryption Challenge 2016)

**Code coverage**

The test will use statement coverage. This involves execution of all statements of the source code at least once.

**Testing scope**

This program contains several different classes, testing scope will cover execute function, getUserJvmOpts function, getTestRunnerClassPath function, getTestRunnerVersion function, runInForkedVM function, and other functions will be set to out of scope. We plan to automate the testing using JUnit and Eclipse IDE.

**Test cases**

1. getTestRunnerClassPath(): This function is to allow user to use either choose an existing Maven or coordinate version to process a POM dependencies. In this test case, it will test if the Test Runner path is not found and it throws some exceptions rather than allowing the program to crash. Another security issue needs to test is file system permission. In different working environments, you may have different group policy settings on your computer. The test case will focus on if the system permission is being blocked and if it should remind the user to change their filesystem permission for this program.

2. getUserJvmOpts(). JVM is one of the key elements for executing Java programs. It helps the Java complier to translate our code into machine readable code to execute. Before the user runs this program, we need to check if the user has the correct JVM. If there is not JVM, it should return an empty string to the program, and another program will know this user does not have the right JVM to run the Java code. If we find that the user installed the correct JVM, we need to check if there is Maven dependency running on the machine. In this test case, JVM and Maven should all be installed so that the function can run the correct value. If one of the elements is missing, the function should return empty string.

3. execute () is the main function for this Java file. The main task is to run Maven and execute JVM to execute the testing and run the testing file. In this test case, it will test if the user is in the wrong configuration environment. If so, the program should stop and return failure complier. When the program starts to run, it will check the target testing file. If the test case inside the test case is not larger than 0, it will return error message for Failed to run the code and give the exit code. When the program cannot run the test case file, it also logs the error message for future analysis.

4. getTestRunnerVersion(). In this function, it will test the tmcJunitRunnerVersion on the machine. If the tmcJunitRunner is not installed on the machine, it will return the string text “tmcjunitrunner.version”

5. runInForkedVM() function is to create the new environment of the Java running environment. It builds a new commandline tool for user to run testing file. If the user’s working environment does not allow user to use commandline tool, it may be blocked by the program. In this testing case, it will check the number of the testing cases inside the file if it is correct or not.

**Test results**

Based on testing result, all four test cases have passed. However, if the user does not have the run environment or the user’s JVM is not installed on the right location. User needs to manually configure to right location. The GetRunningClassPath function may give the error message if the path is empty or path is not found.

**Bugs found**

We do not find any bugs in this file. However, there are several things we suggest improving in the future. First, when this program runs, it should prompt user to select the correct destination of the program instead using the default one.

**9.** **REFERENCES**

Amman, P., & Offutt, J. (2017). *Introduction to software testing.* Cambridge, Unites Kingdom: Cambridge University Press.

Bechtold, S., Brannen, S., Link, J., Merdes, M., Philipp, M., De Rancourt, J., &amp; Stein, C. (n.d.). JUnit 5 User Guide. Retrieved October 16, 2020, from https://junit.org/junit5/docs/current/user-guide/

McAlexander, M. [McProgramming]. (2014, May 24). *Java – JUnit testing in Eclipse* [Video]. YouTube. https://youtu.be/I8XXfgF9GSc

Nygren, H. (n.d.). Testmycode/tmc-maven-plugin. Retrieved October 16, 2020, from https://github.com/testmycode/tmc-maven-plugin/tree/cbb1c8bb36e3d7c12005ec9591a695c6cfbe7068

TestMyCode. (n.d.). Retrieved December 08, 2020, from https://tmc.mooc.fi