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**

Cal.java is a program that does…

**Code coverage**

Are we using Branch Coverage, Statement Coverage, or both?

**Testing scope**

We decided to test these parts of the program…  
We did not test these parts of the program…

We made these decisions because…

**Test cases**

List of test cases (at least 3 assertions and 2 faults)…

**Test results**

List of test results…

**Bugs found**

Summary of bugs found through testing…

**2. THERMOSTAT.JAVA**

**Code purpose**

This is a thermostat class which controls a heater on based on planned weekdays temperature and some override parameters. The thermostat can receive a planned day’s which has desire temperature for different times of day per day of week. User can also override the temperature goal and cause the heater to turn on.

**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 for Turn heater on based on days planned program:

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);

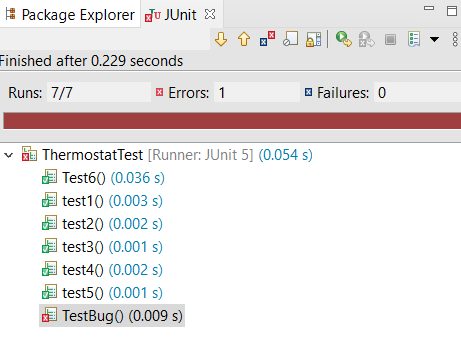
t.setTimeSinceLastRun(2);

t.setMinLag(4);

assertFalse(t.turnHeaterOn(null));

## Test Execution Result

While trying 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 had with automating test cases. Here are the test case execution results.



**Bugs found**

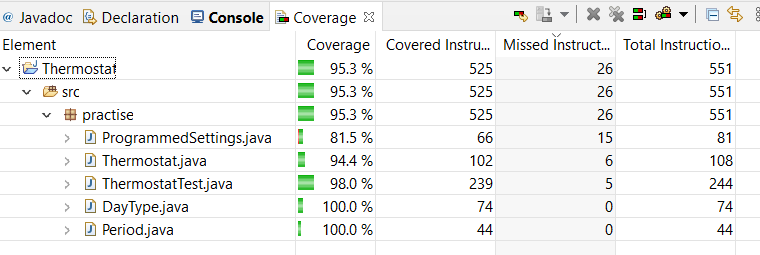
There is bug in thermostat program. As Well As in the fact that there is no error handling for the case that program setting is null. Test cases testbug() failed due to this issue. It would need to add error handling for these cases and gracefully handle them and just do not turn the heater on in these 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 | 11 | |

## Code Coverage

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



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

(Ashley Schonberg)

**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 will be using statement coverage in our test because the program is simple, and it will not require much more time or effort to test each statement in the program. Coverage details will be provided in the next turn-in.

**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**

We have set up the following test cases:

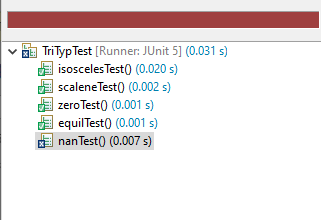
1. scaleneTest()
   1. Tests to make sure scalene triangle inputs result in a scalene triangle result (inputs: 2, 3, 4)
2. isoscelesTest()
   1. Tests to make sure isosceles triangle inputs result in an isosceles triangle result (inputs: 2, 2, 1)
3. equilTest()
   1. Tests to make sure equilateral triangle inputs result in an equilateral triangle result (inputs: 2, 2, 2)
4. zeroTest()
   1. Tests to make sure that an invalid triangle is returned if the user enters a zero (inputs: 1, 2, 0)
5. nanTest()
   1. Tests to make sure the program converts an invalid string to the number 1 (inputs: 1, 2, o)

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

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

**Test results**

Preliminary test results show that four of our tests passed. However, the last test, checking to make sure the letter ‘o’ is converted to the number 1 resulted in a failure. We will review the test case and the program more thoroughly in the following weeks to determine the best way to test this part of the code for the final turn-in.



**Bugs found**

Summary of bugs found through testing provided in next turn-in…

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

(Yu Che Liu)

**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.

**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 test does the function can detect the correct TestRunner path for testing. If there is any missing path, it will throw the error message.

**Test results**

List of test results…

**Bugs found**

Summary of bugs found through testing…

**9.** **REFERENCES**

Amman, P., & Offutt, J. (2017). *Introduction to softwate 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/

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

**Appendices and Annexures**

Here is where we will include our referenced code and any screenshots that do not fit into the two-column format requested. These will be added at the final turn-in during Week 10.