Comprehensive Test Report of

Four Java Classes

Derek Bugg, Anil Erturk, Mu Li, Eugenio Cesar Lorenzo

Team 9, CS570 Software Testing

Master of Science in Computer Science

City University of Seattle

buggderek@cityuniversity.edu, erturkanil@cityuniversity.edu, limu@cityuniverstiy.edu, cesarlorenzoeugenio@cityuniversity.edu

**Abstract**

The purpose of this project is to provide a comprehensive analysis of the implementation of unit tests using automated test tools. Four Java classes will be tested - three of which were provided to the authors, and one of which was selected by the authors. This paper documents the test cases, automated tests, code coverage, test execution scope, and bug reports. Areas not tested will be discussed along with rationale regarding trade-offs and priority testing scope decisions.

The first provided source code module is “Cal.Java”. It has the cal, getN, and main class methods*.* cal method two dates and calculates the number of days between those in the same year. getN method has no parameters and returns an int that is the next input from the user. The main method tests the functionality of the class. The second source code module is “TriTyp.java”. It has triang, getN and main methods. triang method takes three ints that are lengths of sides of a triangle and returns whether these values make a valid triangle or not***.*** GetN and main do the same job as the Cal class.The third file is “Thermostat.java”. Apart from its set methods for its instance variables, it has the turnHeaterOn method that takes a ProgrammedSettings object and returns a boolean that shows whether the heater has been turned on or not. Finally, the team-chosen source code file is IntegerSet.java. It has a boolean array integerSet as a field. Its methods allow it to do calculations that involve it’s boolean array and boolean array of another IntegerSet object. The methods are union, intersection, insertElement, deleteElement, toString, isEqualTo.

Unit testing will be performed on the three instructor-provided modules and documented herein. Code coverage will be developed and performed on all four modules. Automated testing will be created using Junit that includes at least 3 assertions and 2 faults for each module. Though bugs may be insignificant, a full report of all bugs is provided to show full scope of work.

**Keywords:** Software Testing, Java, Junit, Automated Testing, Code Coverage, Test Execution Scope

**1.** **TriTyp class**

**Code Purpose**

The triang method’s gets three integers that represent the lengths of sides of a triangle. Categorizes those side lengths as either scalene triangle, isosceles triangle, equilateral triangle, or invalid. Returns the result that is kept in the triOut variable. triOut variable’s value is an index for triTypes string array that tells us what kind of a triangle those side lengths give us.

triang method was modified to make it more readable and more understandable.

**Test Scope**

JUnit tests were written to test the triang method. triang method has four different groups of outputs. At least one test case that returns each of these output groups were tested. There are more than four unique paths for this method. At least one test case that executes each of the unique paths were tested. Table 1 describes four different output groups and a test value of each group.

|  |  |
| --- | --- |
| Arguments | Expected Result |
| 2, 3, 4 | 1 (scalene) |
| 3, 3, 4 | 2 (isosceles) |
| 3, 3, 3 | 3 (equilateral) |
| 1, 2, 3 | 4 (invalid) |

Table 1

Tests for the triang method are shown in Table 2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test name | Test Type | Arguments | Test Case Description | Expected Result |
| testTriang | Assert | -3, 4, 5 | negative side length | 4 |
| testTriang2 | Assert | 3, 3, 4 | side1 == side2 | 2 |
| testTriang3 | Assert | 4, 3, 3 | side2 == side3 | 2 |
| testTriang4 | Assert | 3, 4, 3 | side1 == side3 | 2 |
| testTriang5 | Assert | 3, 3, 3 | equilateral triangle | 3 |
| testTriang6 | Assert | 1, 2, 3 | invalid triangle | 4 |
| testTriang7 | Assert | 2, 3, 4 | scalene triangle | 1 |

Table 2

**Tests**

In order to test paths related to each code block, we need test that test each of those paths.

Following code block executes if any of the sides are negative.

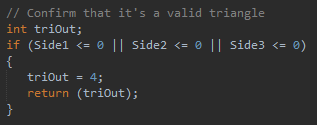


Fig. 1.1

Test where at least one of them are negative.

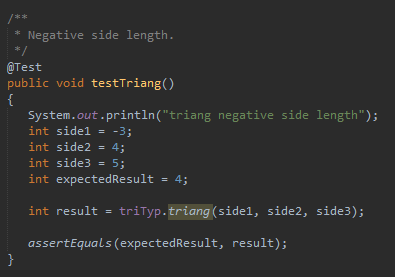


Fig. 1.2

Following code block determines how many and which sides are equal.

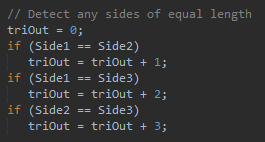


Fig. 2.1

Test where side1 equals side2.

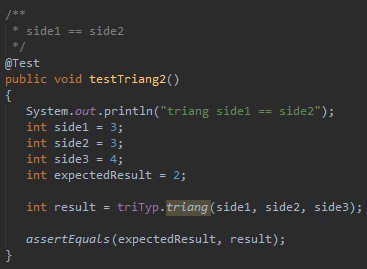


Fig 2.2

Test where side2 equals side3.

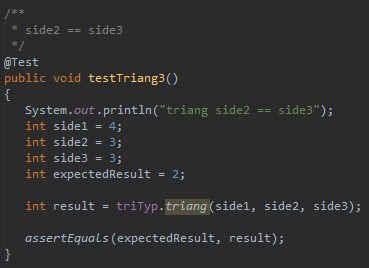


Fig. 2.3

Test where side1 equals side3.

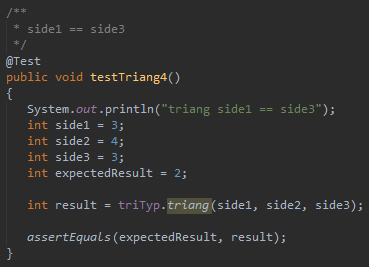


Fig. 2.4

Test where side, side2, and side3 are equal.

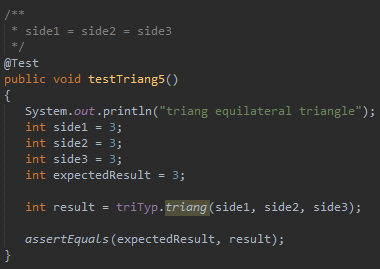


Fig. 2.5

Following code block determines whether the side lengths make a valid triangle or not.

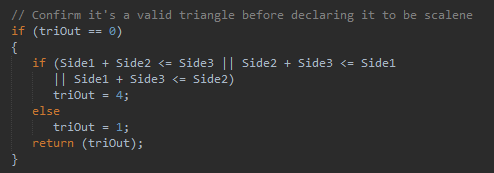


Fig 3.1

Test where side lengths don’t create a valid triangle.

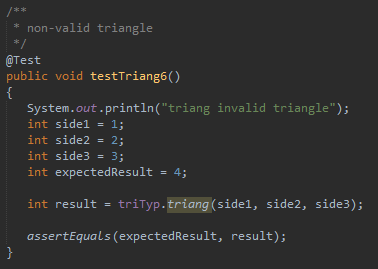


Fig. 3.2

Test where side lengths create a valid scalene triangle.

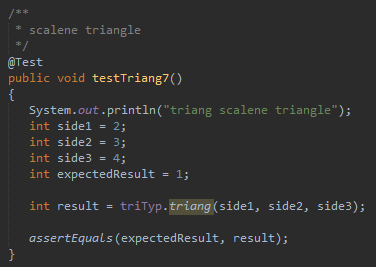


Fig. 3.3

The last code block in triang method doesn’t require any more test cases because the test cases we have created so far include the test cases required here.

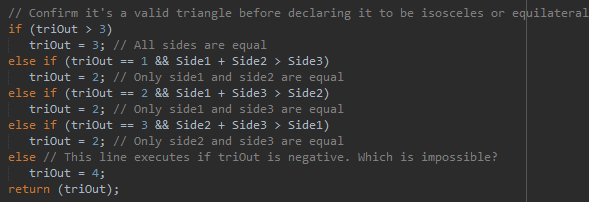


Fig 4.1

**Bugs**

The method acted expectedly for all test cases.

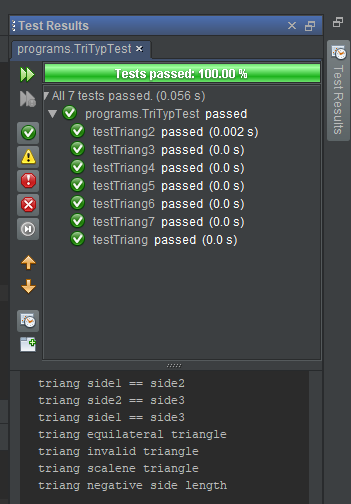


Fig. 5.1

**2.** **Cal class**

**Code Purpose**

cal method of class Cal calculates the number of days between two dates in the same year. If the two dates are in the same month, it returns day2 – day1. If they are not in the same month, it first calculated whether we are in a leap year or not and sets the number of days in February accordingly. Then it returns day2 + remaining days in month1 + days in months between two months.

**Test Scope**

There are multiple paths for execution. Test cases have been developed to cover all unique paths. JUnit tests have been written for all test cases.

Tests for the cal method are shown in Table 3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test name | Test Type | Arguments | Test Case Description | Expected Result |
| testCal | Assert | 1, 8, 1, 15, 2020 | month1 == month2 | 7 |
| testCal2 | Assert | 1, 8, 5, 15, 2020 | month1 != month2. leap year | 128 |
| testCal3 | Assert | 1, 8, 5, 15, 2019 | month1 != month2. Non leap year | 127 |

Table 3

**Tests**

Following code block executes if both dates are in the same month.



Fig 6.1

Test where both months are same.

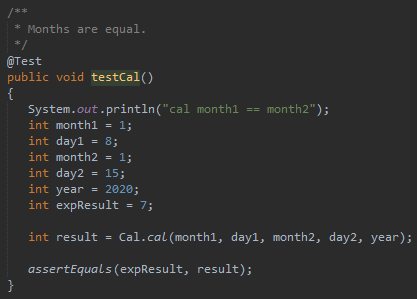


Fig 6.2

Following code block executes if two dates are in different months. There are two different paths based on whether the year is a leap year or not.

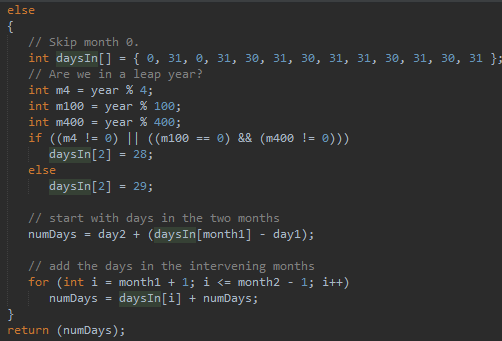


Fig 7.1

Test where the year is a leap year.

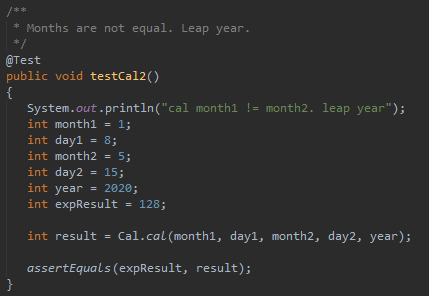


Fig 7.2

Test where the year is not a leap year.

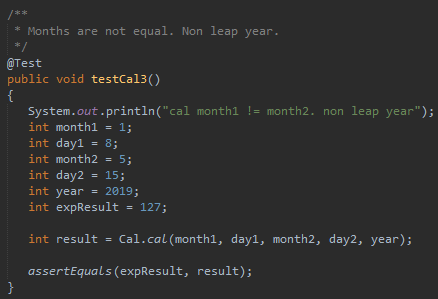


Fig 7.3

**Bugs**

The method acted expectedly for all test cases.

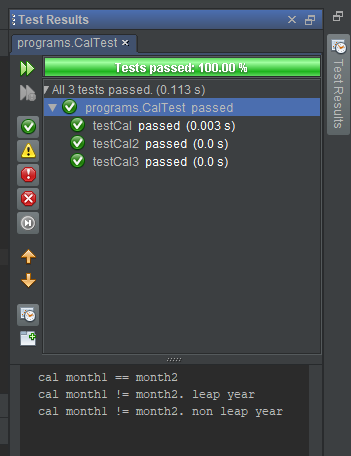


Fig 8.1

**3.** **Thermostat class**

**Code Purpose**

TurnHeaterOn method of class Thermostat determines whether to turn on the thermostat or not based on the variables it has and based on variables of its ProgrammedSettings parameter called pSet.

**Test Scope**

There are multiple paths for execution. Test cases have been developed to cover all unique paths. JUnit tests have been written for all test cases.

Tests for the turnHeaterOn method are shown in Table 4.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test name | Test Type | Arguments | Test Case Description | Expected Result |
| testTurnHeaterOn | Assert | thermostat, pSet | Heater should turn on. | true |
| testTurnHeaterOn2 | Assert | thermostat, pSet | Heater should not turn on. Override = true. | true |
| testTurnHeaterOn3 | Assert | thermostat, pSet | Heater should not turn on. | false |
| testTurnHeaterOn4 | Assert | thermostat, pSet | Heater should not turn on. | false |

Table 4

**Tests**

The turnHeaterOn method decides whether to turn on the heater based on many variables in the Thermostat class and ProgrammetSettings type pSet parameter. The following tests, test the different execution paths of this method.

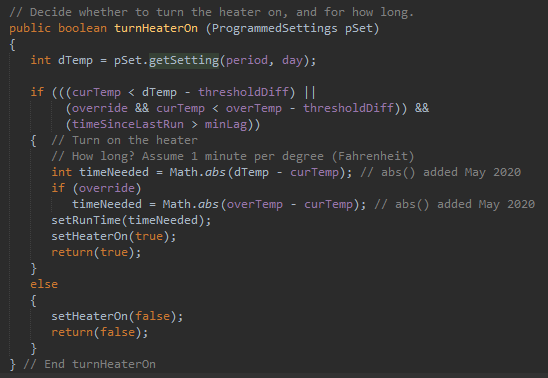


Fig 9.1

Using these parameters, the heater should turn on.

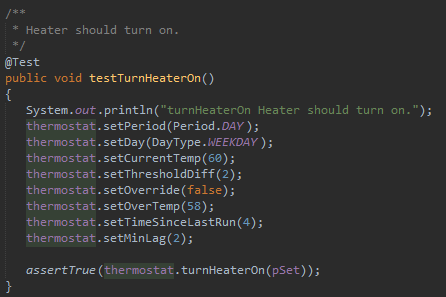


Fig 9.2

Using these parameters, the heater should not turn on. But we have the override variable set to true.

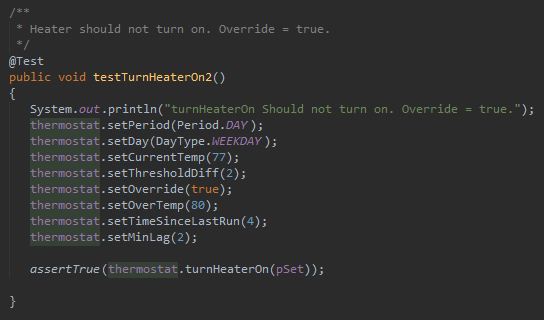


Fig 9.3

Using these parameters, the heater should not turn on.

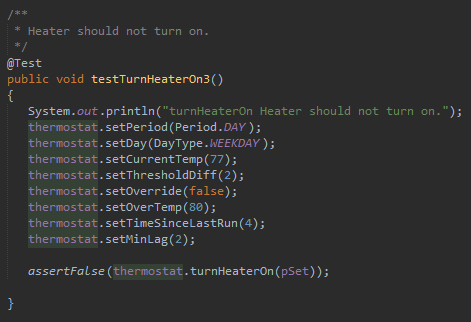


Fig 9.4

Using these parameters, the heater should not turn on.

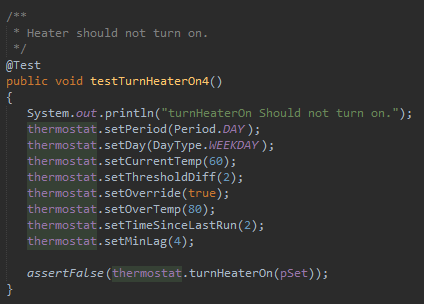


Fig 9.5

**Bugs**

The method acted expectedly for all test cases.

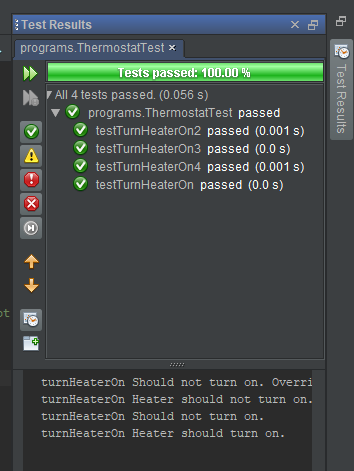


Fig 10.1

**4.** **IntegerSet class**

**Code Purpose**

The IntegerSet class has methods that allow it to simulate bit behaviour. insertElement and deleteElement methods allow us to insert or delete an element from a certain index. union and intersection methods allow us to do calculations with these simulated bits. toString method prints the simulated bits. isEqualTo tests whether two simulated bit sets are equal.

**Test Scope**

There are multiple paths for execution. Test cases have been developed to cover all unique paths. JUnit tests have been written for all test cases.

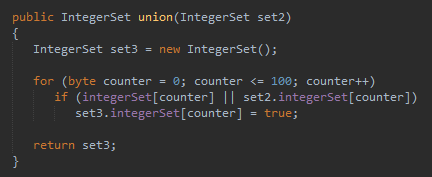
Tests for the cal method are shown in Table 5.

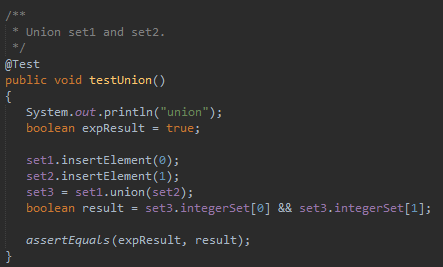
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test name | Test Type | Arguments | Test Case Description | Expected Result |
| testUnion | Assert | 1, 8, 1, 15, 2020 | month1 == month2 | 7 |
| testIntersection | Assert | 1, 8, 5, 15, 2020 | month1 != month2. leap year | 128 |
| testInsertElement | Assert | 1, 8, 5, 15, 2019 | month1 != month2. Non leap year | 127 |
| testDeleteElement |  |  |  |  |
| testToString |  |  |  |  |
| testIsEqualTo |  |  |  |  |

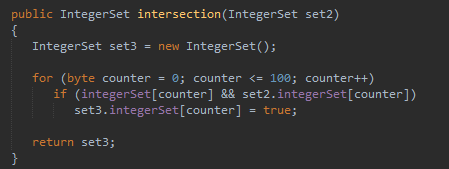
Table 5

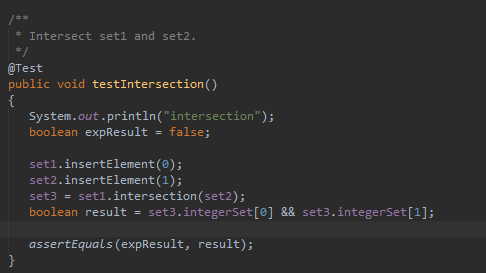
**Tests**

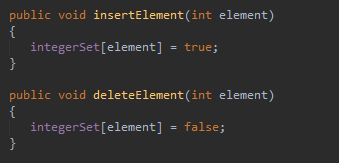
Following code block executes if both dates are in the same month.

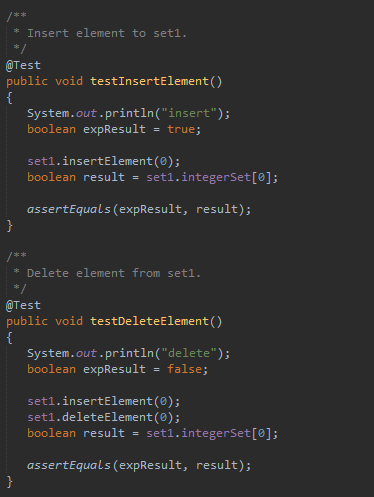


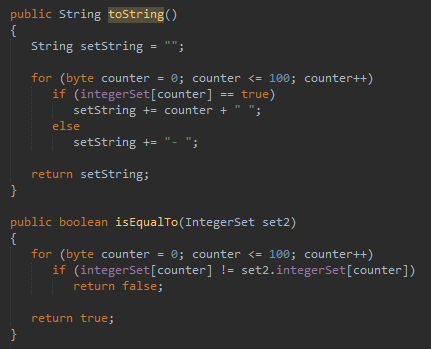














**Bugs**

We had a problem with testToString

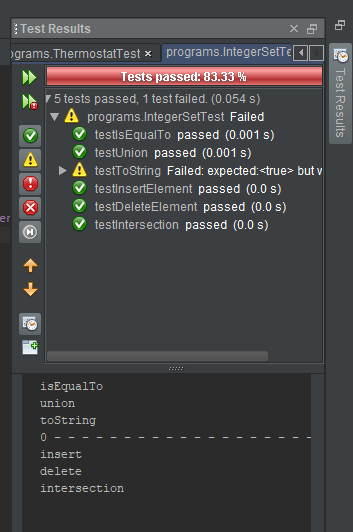


Fig 11.1

**9.** **REFERENCES**

Ammann, P., & Offutt, J. (2017). Introduction to Software Testing (2nd ed.). Cambridge, UK: Cambridge University Press.

Axelrod, A. (2018). Complete Guide to Test Automation: Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects. Berkeley, CA: Apress.

Deitel, P. J., Deitel, H. M. (2015). Java: How to program: Early objects (10th ed.). Harlow, Essex: Pearson Education Limited.

Myers, G. J., Badgett, T., Thomas, T. M., &amp; Sandler, C. (2004). The Art of Software Testing (2nd ed.). Hoboken, NJ: John Wiley & Sons.