ISO2-2023-A04-Testing-P3

Implementation

Alejandro Del Hoyo Abad
Sergio Pozuelo Martín-Consuegra

Variable definition

Luis Eduardo Fernández-Medina Cimas
Anatoli Zournatz
Öykü Sedef Öztürk

Testing

Juan Alcázar Morales
Adrián Gómez del Moral Rodríguez Madridejos

1) Write, at least, the pseudocode of the identified method or methods. However, it will be desirable that you fully implement the code to have a better idea of the final code to be tested.

```
public class GravitationalForceCalculator {

    private double mass1;
    private double mass2;
    private double distance;

    public GravitationalForceCalculator(double mass1, double
mass2, double distance) {
      this.mass1 = mass1;
      this.mass2 = mass2;
      this.distance = distance;
    }

    public double getMass1() {
      return mass1;

    }

    public double getMass2() {
      return mass2;
    }

    public double getDistance() {
      return distance;
    }

    public double calculateForce() throws
NonPositiveDistanceOrMassException {
if (mass1 > 0 && mass2 > 0 && distance > 0)
           return Utils.GRAVITATIONAL_CONSTANT * (mass1 * mass2) /
Math.pow(distance, 2);
      else
           throw new NonPositiveDistanceOrMassException("Value
must be greater than 0.");
    }
}
```

2. Identify the variables that should be considered to test the method of interest.

The variables that we had to consider to test the method of interest are the three variables needed in order to calculate the gravitational force. In this case, we have the first mass, the second mass and the distance between these masses.

3. Identify the test values for each previously identified variables mentioned above, specifying the technique used to obtain each of those values).

In this case, we are going to apply the equivalence class approach. In this case, in the following table, we are able to see the data related to the statement if the exercise:

Input parameter	Equivalence class	Test error guessing values
Mass 1	(-∞, 0) U [0] U [1, ∞)	{-14000, -200, 0, 75, 245, 5000, 100000}
Mass 2	(-∞, 0) U [0] U [1, ∞)	{-1200, -100, 0, 85, 350, 4670, 56600}
Distance	(-∞, 0) U [0] U [1, ∞)	{-25445,-54, 0, 45, 1500, 11000}

4. Calculate the maximum possible number of test cases that could be generated from the test values (combinatorics).

The total number of tests cases for this case would be the cartesian product of all the possible values that each variable can take. In this case, for mass 1 and mass 2, the number of test cases are 7 and for the distance 6. Therefore, the following formula is used:

*n.º mass 1 values * n.º mass 2 values * n.º mass 2 values = max test cases*

Then, we obtain 7 * 7 * 6 = **294** possible test cases.

5. Define a set of test cases to fulfill each use (each value once)

In this case, “each use” means that the relevant test values chosen in section 4 appears at least in one test case, resulting in as many test cases as there are values in the variable with the most components.

Thus, the set of test cases will be given in the following way: (mass 1, mass 2, distance)

- SC1: {0, 0, -54}
- SC2: {-200, 350, 11000}
- SC3: {5000,-100, 45}
- SC4: {245,4670,0}
- SC5: {100000, 56600, 1500}
- SC6: {-14000, 0, -25445}
- SC7: {75, 85, 1500}

6. Define test suitea to achieve pairwise coverage using the proposed algorithm explained in the Lectures

We're dealing with variables that can hold different values. Our goal is to cover all possible pairs of these values, known as achieving "pairwise coverage." Given the complexity arising from the number of variables and their potential values, we've employed a software tool called PICT1. Using this tool, we've generated a comprehensive set of 294 test cases. As a result, we've added these test cases into the file named *pairwise-result.md*.

7. For code segments that include decisions, propose a set of test cases to achieve coverage of decisions.

In this case, the only section of code in which decisions are included is in the *calculate force function.* For this case, it is required the following:

mass1 > 0 and mass2 > 0 and distance > 0

A: mass1 > 0
B: mass2 > 0
C: distance > 0

A	B	C	A and B and C
T	T	T	T
T	T	F	F
T	F	T	F
T	F	F	F
F	T	T	F
F	T	F	F
F	F	T	F
F	F	F	F

Test cases

mass1	mass2	distance	Result
20	30	3	T
5	42	0	F

8. For code segments that include decisions, propose a test case suite to achieve MC/DC coverage.

In this case, the only section of code in which decisions are included is in the *calculate force function.* For this case, it is required the following:

mass1 > 0 and mass2 > 0 and distance > 0

A: mass1 > 0
B: mass2 > 0
C: distance > 0

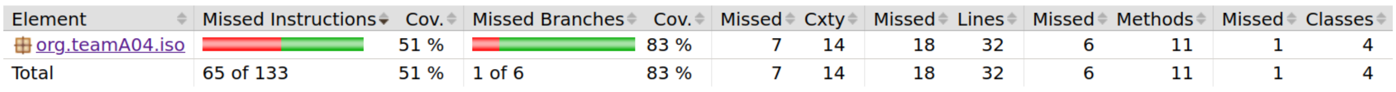
A	B	C	A and B and C	Dominant
T	T	T	T	A,B,C
T	T	F	F	C
T	F	T	F	B
T	F	F	F	B,C
F	T	T	F	A
F	T	F	F	A,C
F	F	T	F	A,B
F	F	F	F	A,B,C

Test cases

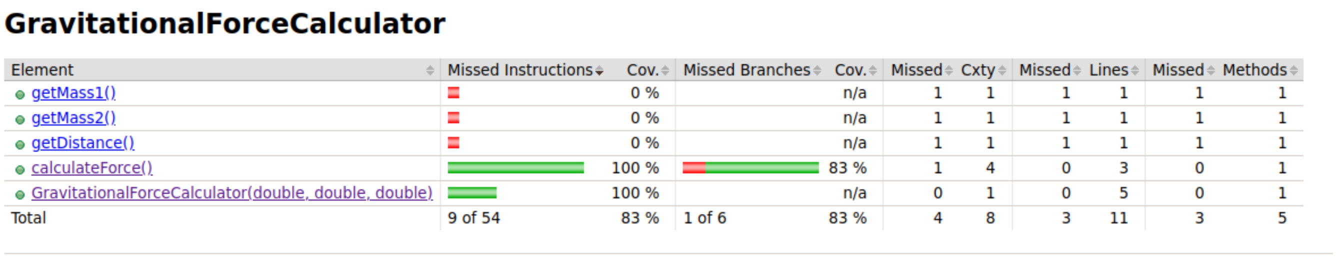
mass1	mass2	distance	Result
5	10	3	T
5	10	0	F
10	0	3	F
-5	10	3	F

9. Comment on the results of the number of test cases obtained in sections 4, 5, and 6, as well as the execution of the oracles: what can be said about the coverage achieved?

In this case the coverage achieved is not quite big because the class App has not been tested correctly. Moreover for the class GravitatiationalForceCalculator the coverage is not 100% because the getters are never called by the test code.

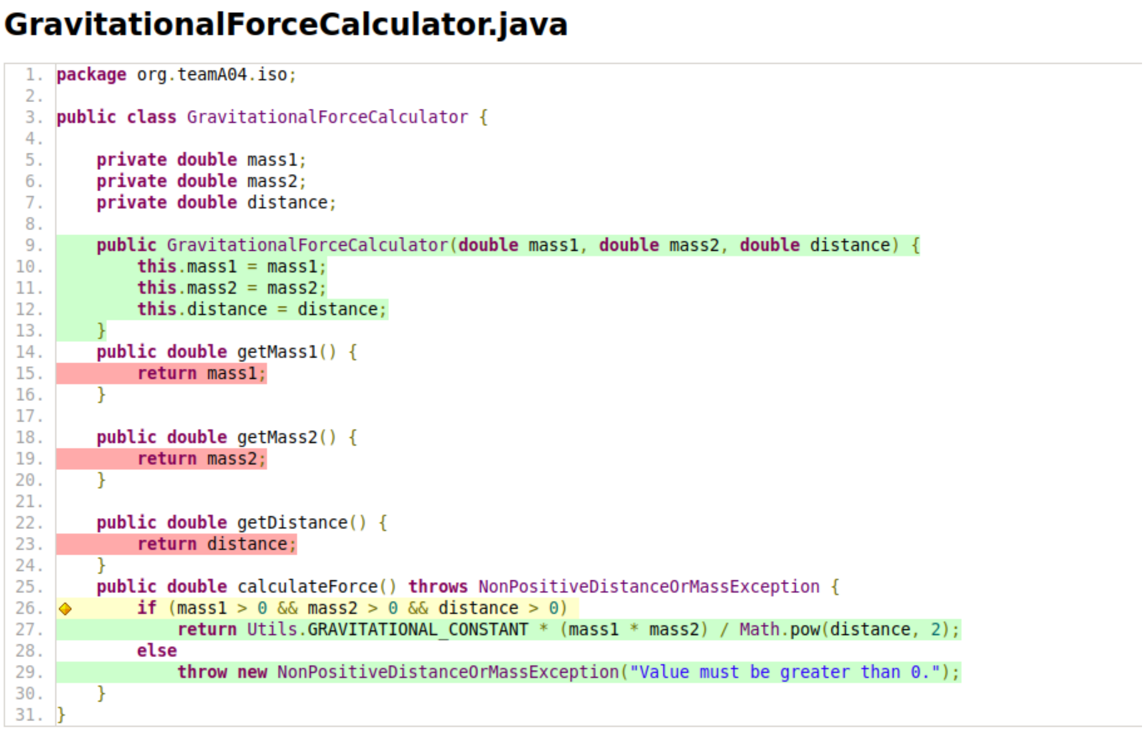


Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.teamA04.iso	🟥🟩	51 %	🟥🟩	83 %	7	14	18	32	6	11	1	4
Total	65 of 133	51 %	1 of 6	83 %	7	14	18	32	6	11	1	4



GravitationalForceCalculator

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
getMass1()	🟥	0 %		n/a	1	1	1	1	1	1
getMass2()	🟥	0 %		n/a	1	1	1	1	1	1
getDistance()	🟥	0 %		n/a	1	1	1	1	1	1
calculateForce()	🟩	100 %	🟥🟩	83 %	1	4	0	3	0	1
GravitationalForceCalculator(double, double, double)	🟩	100 %		n/a	0	1	0	5	0	1
Total	9 of 54	83 %	1 of 6	83 %	4	8	3	11	3	5



GravitationalForceCalculator.java

```java
package org.teamA04.iso;

public class GravitationalForceCalculator {

    private double mass1;
    private double mass2;
    private double distance;

    public GravitationalForceCalculator(double mass1, double mass2, double distance) {
        this.mass1 = mass1;
        this.mass2 = mass2;
        this.distance = distance;
    }
    public double getMass1() {
        return mass1;
    }

    public double getMass2() {
        return mass2;
    }

    public double getDistance() {
        return distance;
    }
    public double calculateForce() throws NonPositiveDistanceOrMassException {
        if (mass1 > 0 && mass2 > 0 && distance > 0)
            return Utils.GRAVITATIONAL_CONSTANT * (mass1 * mass2) / Math.pow(distance, 2);
        else
            throw new NonPositiveDistanceOrMassException("Value must be greater than 0.");
    }
}
```