Report for lab 1, TDDD04

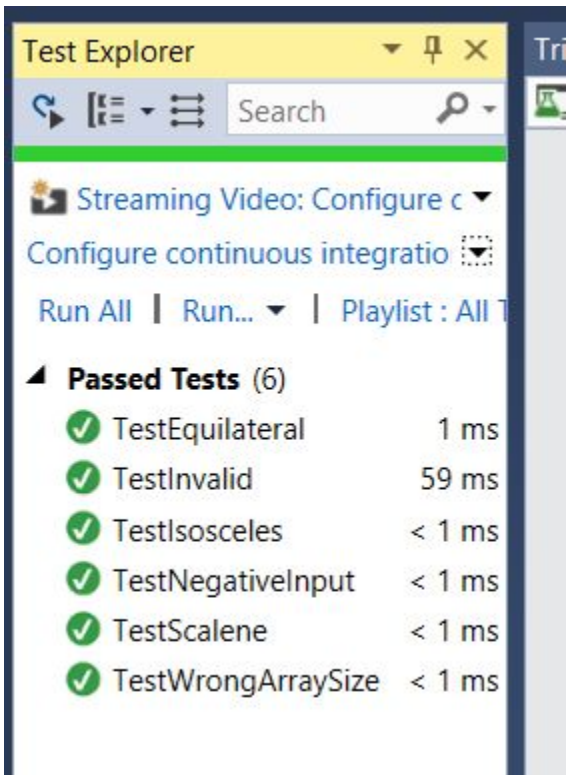
All code needed for this lab is available on gitlab, group tddd04-2016-c3-1.
https://gitlab.ida.liu.se/groups/tddd04-2016-c3-1

Part 1

The test cases that we used to test the program was

- 2 2 1 (isosceles)
- 4 5 6 (scalene)
- 1 1 1 (equilateral)
- 1 1 2 (non-triangle)
- 1 3 (non-triangle)
- -1 1 1 (negative length of one side)

These inputs were converted to unit tests. In the first three of these cases the program behaved as expected, but in the last three the program crashed.
The code was later fixed to handle these types of input and all the test cases from before now passes, as seen in the figure below.



Part 2

In the factory method code example we managed to get close to 100% line and mutation coverage as shown in the screenshot below. The only missed mutations are in the class PizzaStore.

Problems @ Javadoc Declaration Search Console Coverage PIT Mutations PIT Summary

Pit Test Coverage Report

Package Summary

default

Number of Classes	Line Coverage		Mutation Coverage	
9	94%	50/53	96%	25/26

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
CheesePizza.java	100%	2/2	100%	1/1
ClamPizza.java	100%	2/2	100%	1/1
NYCheesePizza.java	100%	2/2	100%	1/1
NYPizzaStore.java	100%	11/11	100%	5/5
PepperoniPizza.java	100%	2/2	100%	1/1
Pizza.java	100%	10/10	100%	5/5
PizzaStore.java	73%	8/11	83%	5/6
SthlmPizzaStore.java	100%	11/11	100%	5/5
VeggiePizza.java	100%	2/2	100%	1/1

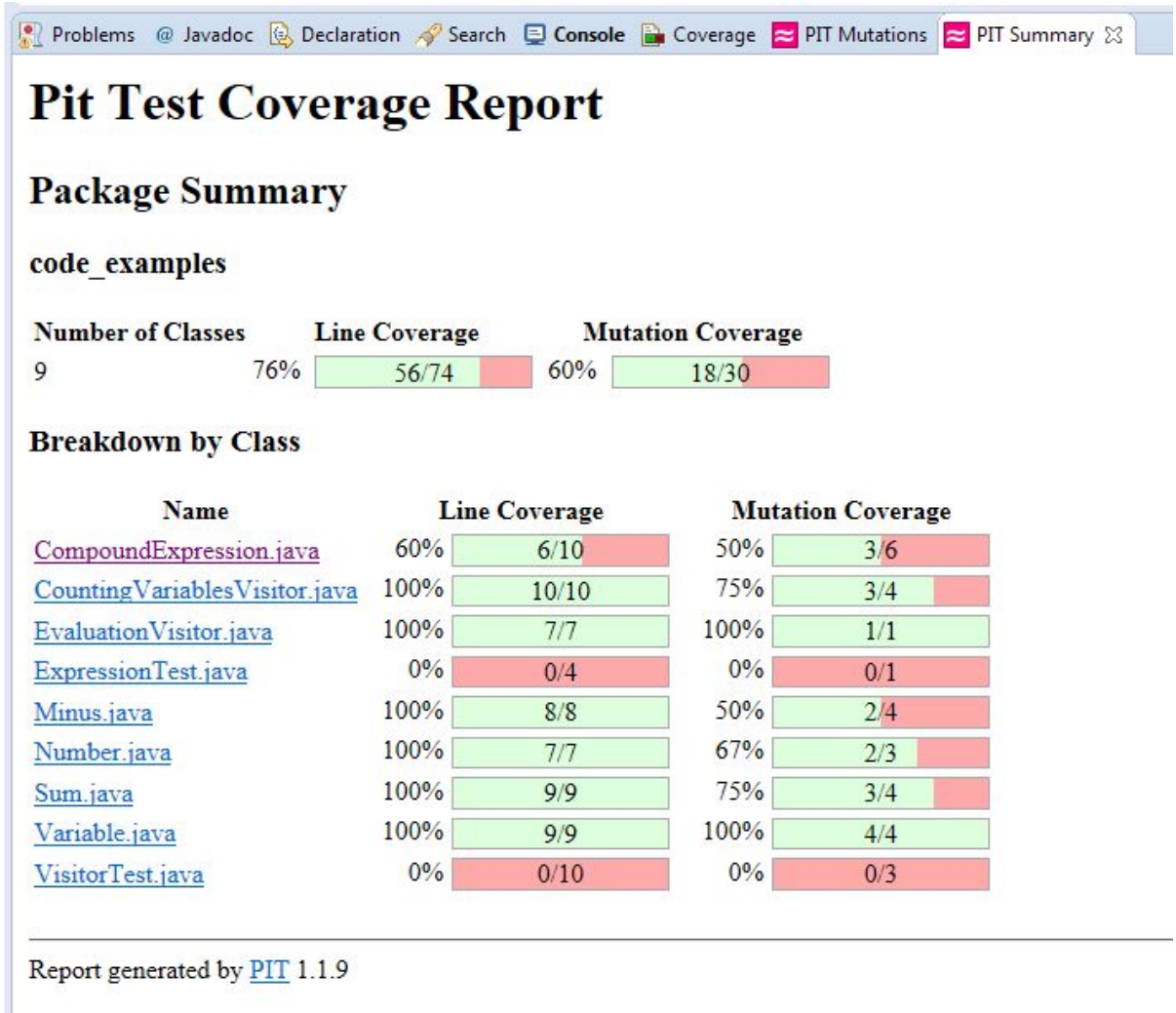
Report generated by PIT 1.1.9

Problems @ Javadoc Declaration Console Coverage

PizzaStoreTest (2016-sep-01 14:57:58)

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
code_examples	29,3 %	251	606	857
Visitor/src	0,0 %	0	284	284
Builder/src	0,0 %	0	238	238
Decorator/src	0,0 %	0	79	79
Factory Method/src	98,0 %	251	5	256
(default package)	98,0 %	251	5	256
PizzaStore.java	85,7 %	18	3	21
Pizza.java	90,5 %	19	2	21
CheesePizza.java	100,0 %	5	0	5
ClamPizza.java	100,0 %	5	0	5
NYCheesePizza.java	100,0 %	5	0	5
NYClamPizza.java	100,0 %	3	0	3
NYPepperoniPizza.java	100,0 %	3	0	3
NYPizzaStore.java	100,0 %	42	0	42
NYVeggiePizza.java	100,0 %	3	0	3
PepperoniPizza.java	100,0 %	5	0	5
PizzaStoreTest.java	100,0 %	84	0	84
SthlmCheesePizza.java	100,0 %	3	0	3
SthlmClamPizza.java	100,0 %	3	0	3
SthlmPepperoniPizza.java	100,0 %	3	0	3
SthlmPizzaStore.java	100,0 %	42	0	42
SthlmVeggiePizza.java	100,0 %	3	0	3
VeggiePizza.java	100,0 %	5	0	5

For the Visitor example we were only able to reach 60 % mutation coverage. When inspecting the missed mutations we could see that they all were in unreachable parts of the program. For example, mutations in the two classes ExpressionTest and VisitorTest were missed since these classes are never used. We also missed mutations in the accept method in the class AbstractExpression. This method will never be executed since all subclasses of this abstract class will override this method.

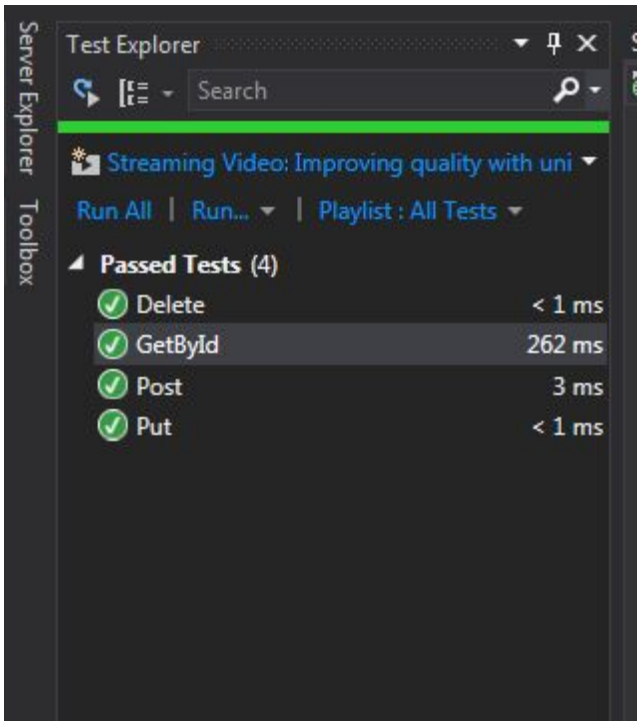


Problems | Javadoc | Declaration | Search | Console | Coverage | PIT Mutations | PIT Summary

VisitorTestCase (2017-jan-10 09:15:17)

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
code_examples	44,4 %	324	405	729
Factory Method/src	0,0 %	0	300	300
Visitor/src	75,5 %	324	105	429
code_examples	75,5 %	324	105	429
VisitorTest.java	0,0 %	0	55	55
ExpressionTest.java	0,0 %	0	35	35
CompoundExpression.java	80,0 %	48	12	60
VisitorAllTests.java	0,0 %	0	3	3
AbstractExpression.java	100,0 %	3	0	3
CountingVariablesVisitor.java	100,0 %	26	0	26
EvaluationVisitor.java	100,0 %	11	0	11
Minus.java	100,0 %	23	0	23
Number.java	100,0 %	20	0	20
SimpleExpression.java	100,0 %	3	0	3
Sum.java	100,0 %	23	0	23
Variable.java	100,0 %	22	0	22
Visitor.java	100,0 %	3	0	3
VisitorTestCase.java	100,0 %	142	0	142

Part 3



Above we have included the succeeded test cases, all stubs are commented in the code for the lab attached, after implementing the actual storage.

Part 4

All statements in the tests are not covered. The test cases that are not fully covered contains a lot of if statements (branches), some of which are missed when running the tests. This is likely due to the fact that the program i highly coupled which means that it is hard to test

small parts of the program (proper unit testing). When the classes are dependent on each other to work, it is hard to test them separately. Therefore we have to use long test cases with many if statements to test as much of the code as possible, which is also why we miss some branches of the test cases.

From the UML class diagram below it is clear that the program is highly coupled due to the many dependencies in both directions from and to the Colony Class to other classes. To enable more thorough testing without creating other concrete objects one could implement mock objects. This allows the tester to decide the behaviour of moc objects to create different scenarios and by so covering all branches of possible executions.

The danger of testing high coupled programmes is the high factor of branching possible; when one class behaviour changes, it creates many different scenarios in other classes as well. This creates the illusion of coverage when in fact lots of branches may have been missed in the tests. Same goes for path coverage.