Ryerson University

Faculty of Engineering & Architectural Science

Department of Biomedical, Computer, and Electrical Engineering

Course Number	COE891
Course Title	Software Testing and Quality Assurance
Semester/Year	Winter2024
Instructor	Dr. Reza Samavi
Section No.	01
Group No.	N/A
Submission Date	Apr 1, 2024 10:00 AM
Due Date	Mar 31, 2024

Lab/Tut Assignment NO.	6

Assignment Title	Mutation Testing

Name	Student ID	Signature*
Astha Patel	501040209	

By signing above you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: www.ryerson.ca/senate/current/pol60.pdf

Q1

1. **Achieving 100% coverage on all test classes**
   a. MoneyTest.java

Figure 1: 100% coverage achieved for the MoneyTest.java

   b. MoneyBagTest.java

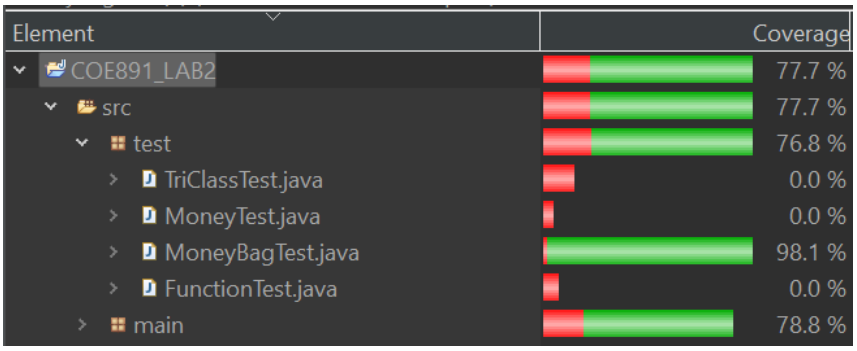


Figure 2: 98.1% coverage achieved for the MoneyBagTest.java

   c. TriClassTest.java

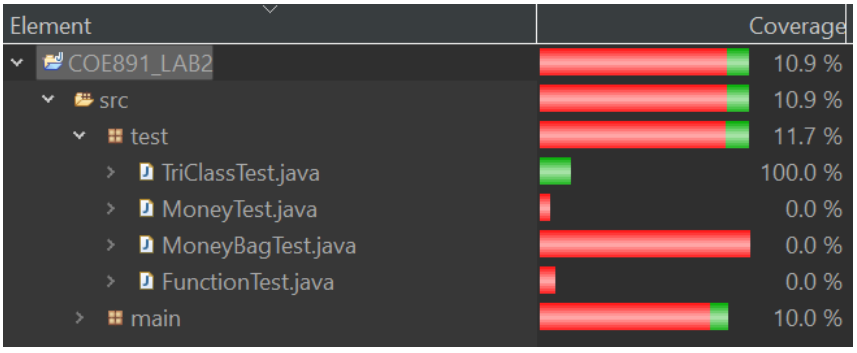


Figure 3: 100% coverage achieved for the TriClassTest.java

   d. FunctionTest.java

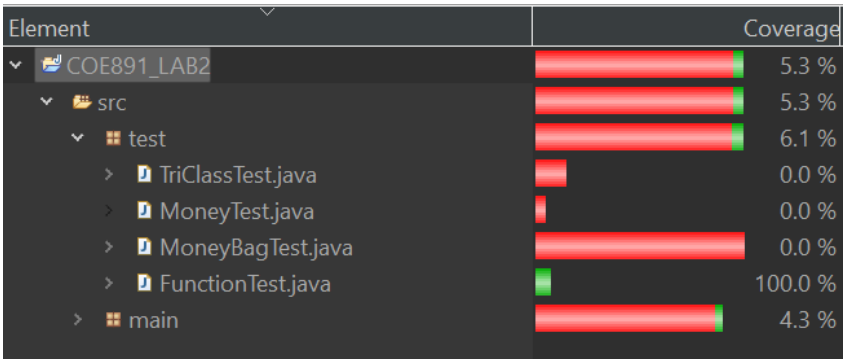


Figure 4: 100% coverage achieved for the FunctionTest.java

2. PIT Tests

a. MoneyBagTest.java

Pit Test Coverage Report

Package Summary

main

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
2	94%	96/102	81%	67/83	91%	67/74

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
Money.java	93%	27/29	83%	25/30	93%	25/27
MoneyBag.java	95%	69/73	79%	42/53	89%	42/47

Figure 5: PIT Test coverage report on MoneyBag.java before alterations.

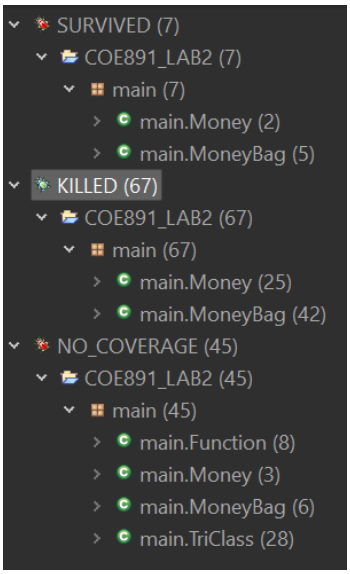


Figure 6: PIT mutations statistics.

b. **MoneyTest.java**

Pit Test Coverage Report

Package Summary

main

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
2	6%	6/102	2%	2/83	100%	2/2

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
Money.java	21%	6/29	7%	2/30	100%	2/2
MoneyBag.java	0%	0/73	0%	0/53	0%	0/0

Figure 7: PIT Test coverage report on Money.java before alterations.

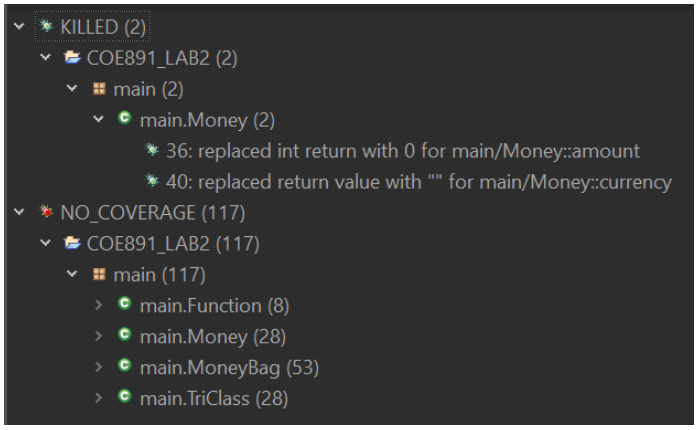


Figure 8: PIT mutations statistics.

3. Selected mutation types

a. MoneyTest.java

Name	Description
Invert Negatives	Inverts negation of integer and floating point numbers
Return Values	Mutates the return values of method calls. Depending on the return type of the method another mutation is used
Inline Constant	Mutates inline constants. An inline constant is a literal value assigned to a non-final variable
Math	Replaces binary arithmetic operations for either integer or floating-point arithmetic with another operation
Void Method Call	Removes method calls to void methods
Negate Conditionals	Mutates all conditionals found
Conditionals Boundary	Replaces the relational operators <, <=, >, >=
Increments	Mutates increments, decrements and assignment increments and decrements of local variables (stack variables). Replaces increments with decrements and vice versa
Remove Increments	Removes local variable increments
Non Void Method Call	Removes method calls to non void methods. Their return value is replaced by the Java Default Value for that specific type
Constructor Call	Replaces constructor calls with null values
Remove Equal Conditionals If	Remove equal conditions and replace with true, execute if part
Remove Equal Conditionals Else	Remove equal conditions and replace with false, execute else part
Remove Order Checks If	Remove order conditions and replace with true, execute if part
Remove Order Checks Else	Remove order conditions and replace with false, execute else part
Remove Conditionals	Removes all conditionals statements such that the guarded statements always execute
True Returns	Replaces primitive and boxed boolean return values with true

Selected mutators for MoneyTest.java

b. MoneyBagTest.java

☐ Invert Negatives	Inverts negation of integer and floating point numbers
☑ Return Values	Mutates the return values of method calls. Depending on the return type of the method another mutation is used
☐ Inline Constant	Mutates inline constants. An inline constant is a literal value assigned to a non-final variable
☑ Math	Replaces binary arithmetic operations for either integer or floating-point arithmetic with another operation
☐ Void Method Call	Removes method calls to void methods
☑ Negate Conditionals	Mutates all conditionals found
☐ Conditionals Boundary	Replaces the relational operators <, <=, >, >=
☐ Increments	Mutates increments, decrements and assignment increments and decrements of local variables (stack variables). Replaces increments with decrements and vice versa
☐ Remove Increments	Removes local variable increments
☐ Non Void Method Call	Removes method calls to non void methods. Their return value is replaced by the Java Default Value for that specific type
☐ Constructor Call	Replaces constructor calls with null values
☐ Remove Equal Conditionals If	Remove equal conditions and replace with true, execute if part
☐ Remove Equal Conditionals Else	Remove equal conditions and replace with false, execute else part
☐ Remove Order Checks If	Remove order conditions and replace with true, execute if part
☐ Remove Order Checks Else	Remove order conditions and replace with false, execute else part
☐ Remove Conditionals	Removes all conditionals statements such that the guarded statements always execute
☑ True Returns	Replaces primitive and boxed boolean return values with true
☑ False Returns	Replaces primitive and boxed boolean return values with false
☐ Primite Returns	Replaces int, short, long, char, float and double return values with 0
☐ Empty Returns	Replaces return values with an 'empty' value
☑ Null Returns	Replaces return values with null. Method that can be mutated by the EMPTY_RETURNS mutator or that are directly annotated with NotNull are not mutated

Selected mutators for MoneyBagTest.java

4. Mutation types from the selected mutation types that achieve the highest mutation score

```
================================================================================
- Mutators
================================================================================
> org.pitest.mutationtest.engine.gregor.mutators.BooleanTrueReturnValsMutator
>> Generated 11 Killed 3 (27%)
> KILLED 3 SURVIVED 2 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 6
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.EmptyObjectReturnValsMutator
>> Generated 8 Killed 2 (25%)
> KILLED 2 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 6
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.ConditionalsBoundaryMutator
>> Generated 5 Killed 0 (0%)
> KILLED 0 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 5
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.NullReturnValsMutator
>> Generated 17 Killed 12 (71%)
> KILLED 12 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 5
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.VoidMethodCallMutator
>> Generated 9 Killed 2 (22%)
> KILLED 2 SURVIVED 3 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 4
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.InvertNegsMutator
>> Generated 1 Killed 1 (100%)
> KILLED 1 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.MathMutator
>> Generated 9 Killed 2 (22%)
> KILLED 2 SURVIVED 1 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 6
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.BooleanFalseReturnValsMutator
>> Generated 7 Killed 6 (86%)
> KILLED 6 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 1
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.NegateConditionalsMutator
>> Generated 47 Killed 19 (40%)
> KILLED 19 SURVIVED 2 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 26
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.PrimitiveReturnsMutator
>> Generated 5 Killed 2 (40%)
> KILLED 2 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 3
--------------------------------------------------------------------------------
```

Figure 9: Mutators report after altering the MoneyTest.java class.

```
================================================================================
- Mutators
================================================================================
> org.pitest.mutationtest.engine.gregor.mutators.BooleanTrueReturnValsMutator
>> Generated 11 Killed 10 (91%)
> KILLED 10 SURVIVED 1 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.EmptyObjectReturnValsMutator
>> Generated 3 Killed 2 (67%)
> KILLED 2 SURVIVED 1 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.NullReturnValsMutator
>> Generated 17 Killed 17 (100%)
> KILLED 17 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.VoidMethodCallMutator
>> Generated 9 Killed 9 (100%)
> KILLED 9 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.InvertNegsMutator
>> Generated 1 Killed 1 (100%)
> KILLED 1 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.MathMutator
>> Generated 4 Killed 2 (50%)
> KILLED 2 SURVIVED 2 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.BooleanFalseReturnValsMutator
>> Generated 7 Killed 6 (86%)
> KILLED 6 SURVIVED 1 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.NegateConditionalsMutator
>> Generated 28 Killed 25 (89%)
> KILLED 25 SURVIVED 3 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
```

5. **PIT scores**
   a. **MoneyTest.java**

Pit Test Coverage Report

Package Summary

main

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
2	6%	6/102	5%	4/78	100%	4/4

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
Money.java	21%	6/29	13%	4/31	100%	4/4
MoneyBag.java	0%	0/73	0%	0/47	0%	0/0

Figure 10: PIT Test Coverage Report on MoneyTest.java after alterations.

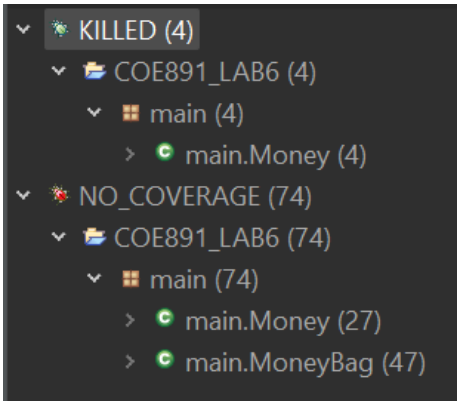


Figure 11: PIT mutations statistics of MoneyTest.java after alterations.

Active mutators

- EMPTY_RETURN_VALUES
- MATH_MUTATOR
- NEGATE_CONDITIONALS_MUTATOR
- PRIMITIVE_RETURN_VALS_MUTATOR
- RETURN_VALS_MUTATOR

Figure 13: Active Mutators of the MoneyTest.java test class.

b. MoneyBagTest.java

Pit Test Coverage Report

Package Summary

main

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
2	100%	102/102	93%	91/98	93%	91/98

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
Money.java	100%	29/29	93%	37/40	93%	37/40
MoneyBag.java	100%	73/73	93%	54/58	93%	54/58

Figure 14: PIT Test Coverage Report on MoneyBagTest.java after alterations.

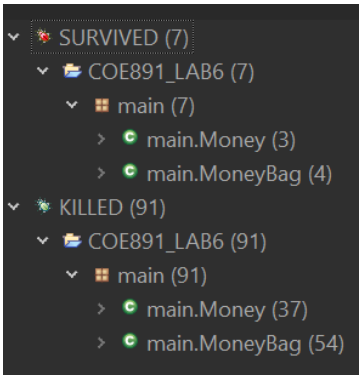


Figure 15: PIT mutations statistics of MoneyBagTest.java after alterations.

Active mutators

- BOOLEAN_FALSE_RETURN
- BOOLEAN_TRUE_RETURN
- CONSTRUCTOR_CALL_MUTATOR
- MATH_MUTATOR
- NULL_RETURN_VALUES
- RETURN_VALS_MUTATOR
- VOID_METHOD_CALL_MUTATOR

Figure 16: Active Mutators of the MoneyBagTest.java test class.

6. Comparisons

The mutations report for before and after test class alteration of the Money.java class is displayed below. The two reports demonstrate the mutants that survived, killed, and did not achieve any coverage. Furthermore, the altered test class was able to achieve more mutator coverage as it increased the effectiveness of the tests by being able to introduce different types of mutators through various test cases.

Initial Mutation Report:

Mutations

22	1. replaced return value with null for main/Money::add → NO_COVERAGE
26	1. negated conditional → NO_COVERAGE
27	1. Replaced integer addition with subtraction → NO_COVERAGE 2. replaced return value with null for main/Money::addMoney → NO_COVERAGE
28	1. replaced return value with null for main/Money::addMoney → NO_COVERAGE
32	1. replaced return value with null for main/Money::addMoneyBag → NO_COVERAGE
36	1. replaced int return with 0 for main/Money::amount → KILLED
40	1. replaced return value with "" for main/Money::currency → KILLED
44	1. negated conditional → NO_COVERAGE
45	1. negated conditional → NO_COVERAGE
46	1. replaced boolean return with false for main/Money::equals → NO_COVERAGE 2. replaced boolean return with true for main/Money::equals → NO_COVERAGE
47	1. negated conditional → NO_COVERAGE
49	1. replaced boolean return with false for main/Money::equals → NO_COVERAGE 2. replaced boolean return with true for main/Money::equals → NO_COVERAGE 3. negated conditional → NO_COVERAGE
50	1. negated conditional → NO_COVERAGE
52	1. replaced boolean return with true for main/Money::equals → NO_COVERAGE
56	1. Replaced integer addition with subtraction → NO_COVERAGE 2. replaced int return with 0 for main/Money::hashCode → NO_COVERAGE
60	1. replaced boolean return with false for main/Money::isZero → NO_COVERAGE 2. replaced boolean return with true for main/Money::isZero → NO_COVERAGE 3. negated conditional → NO_COVERAGE
64	1. Replaced integer multiplication with division → NO_COVERAGE 2. replaced return value with null for main/Money::multiply → NO_COVERAGE
68	1. removed negation → NO_COVERAGE 2. replaced return value with null for main/Money::negate → NO_COVERAGE
72	1. replaced return value with null for main/Money::subtract → NO_COVERAGE
78	1. replaced return value with "" for main/Money::toString → NO_COVERAGE
82	1. removed call to main/MoneyBag::appendMoney → NO_COVERAGE

Figure 13: Report of mutations of the Money.java class before alterations.

Mutation report after increasing mutation coverage:

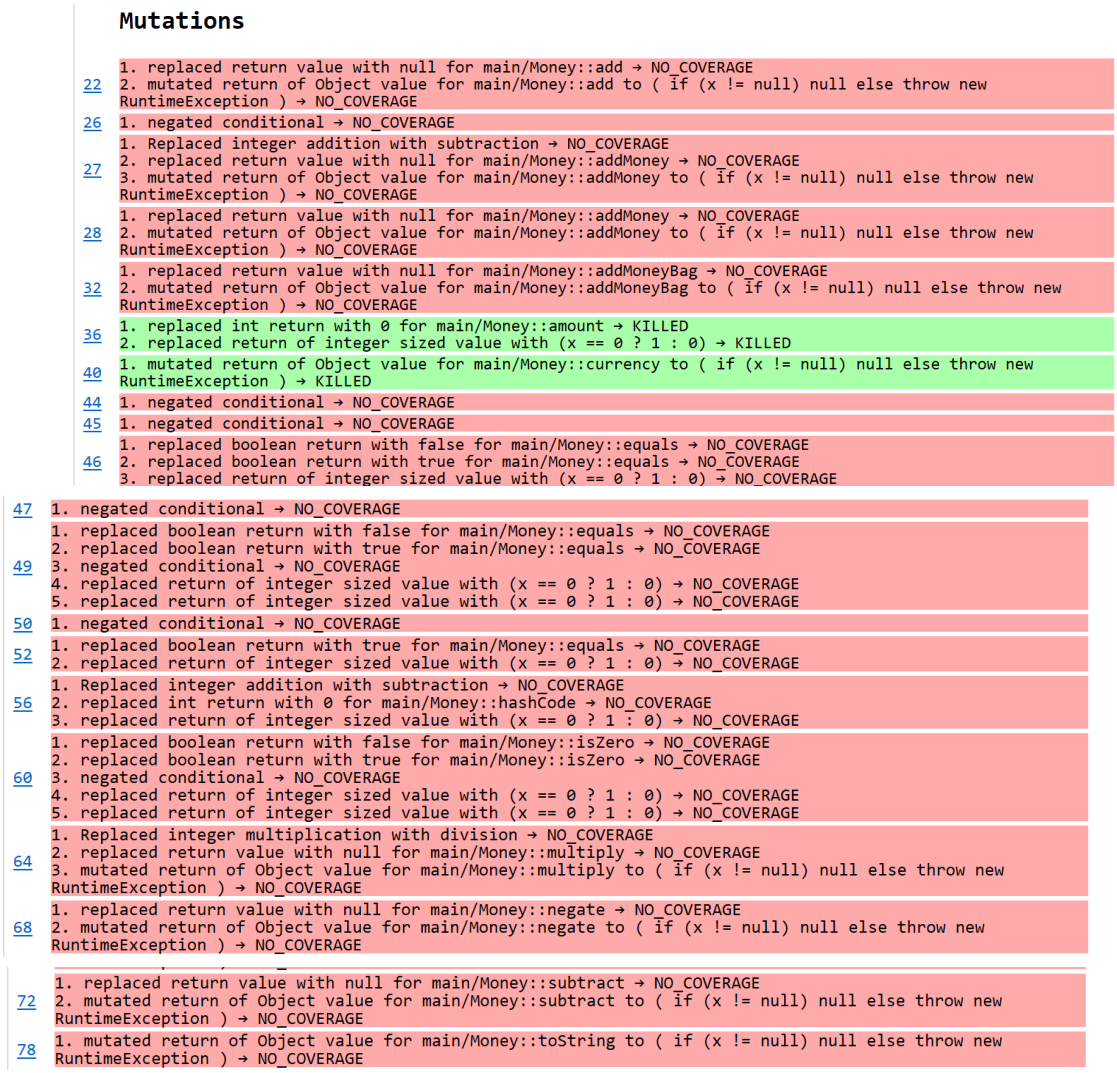
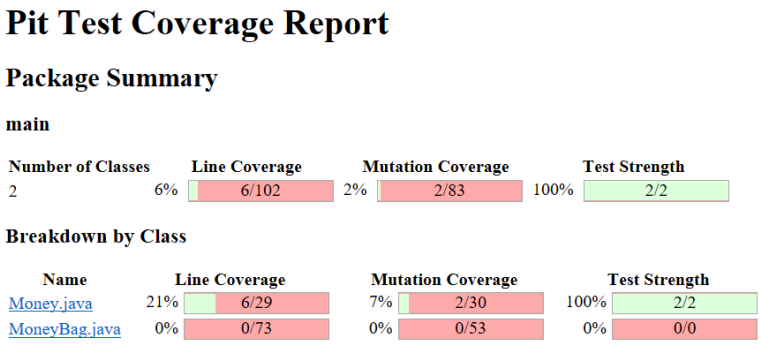


Figure 14: Report of mutations of the Money.java class after alterations.

Effective software testing is crucial for building reliable programs. Mutation testing takes this a step further by injecting controlled errors into the code and checking if existing tests can catch them. This process exposes weaknesses in the test suite, highlighting areas where real bugs might slip through the cracks. Achieving high mutation coverage signifies that the tests are effectively identifying these errors and the assurance in finding these issues. The following report for the Money class showcases the mutation coverage of 7% before and 13% after achieving higher mutation coverage.



Pit Test Coverage Report

Package Summary

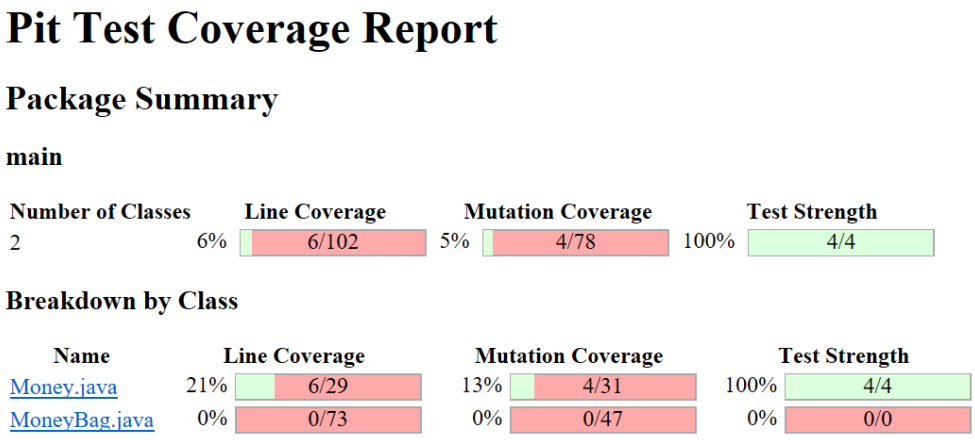
main

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
2	6%	6/102	2%	2/83	100%	2/2

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
Money.java	21%	6/29	7%	2/30	100%	2/2
MoneyBag.java	0%	0/73	0%	0/53	0%	0/0

Figure 15: Report of PIT Test Coverage of Money.java class before alterations.



Pit Test Coverage Report

Package Summary

main

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
2	6%	6/102	5%	4/78	100%	4/4

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
Money.java	21%	6/29	13%	4/31	100%	4/4
MoneyBag.java	0%	0/73	0%	0/47	0%	0/0

Figure 16: Report of mutations of the Money.java class after alterations.

Finally, the report of the test class before had 28 uncovered mutants and only 2 killed mutants while zero survived mutants. This decreased the quality of the tests as it did not test all possible outputs that may enable the program to allow incorrect values or exceed boundary conditions. However, after altering the mutators performed on the MoneyTest.ajava class, the PIT test report covered 57 and killed 49 mutants, and only 8 surviving mutants. Fewer test methods can significantly limit the effectiveness of PIT (mutation) testing and mutation coverage. With a restricted test suite, the chances of encountering the code modifications introduced by mutations decrease. Many mutations might remain undetected because the existing tests simply don't exercise the code paths where they're injected. This results in a lower mutation coverage score, which doesn't accurately reflect the true resilience of your code. Fewer tests might overlook different types of potential faults that mutations represent. This can create a false sense of security, as real bugs that cause similar errors might slip through undetected.

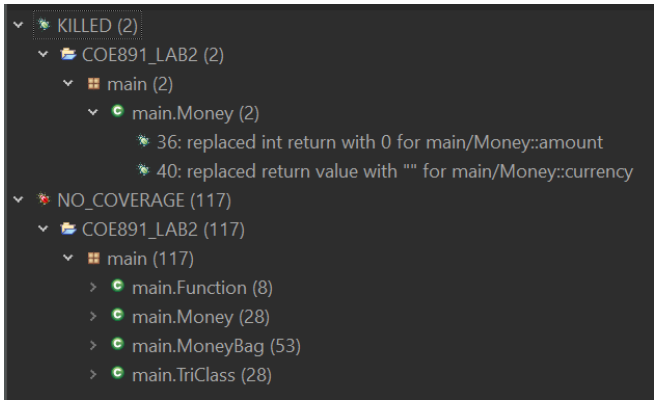


Figure 17: Report of status of the mutants of the Money.java class before alterations.

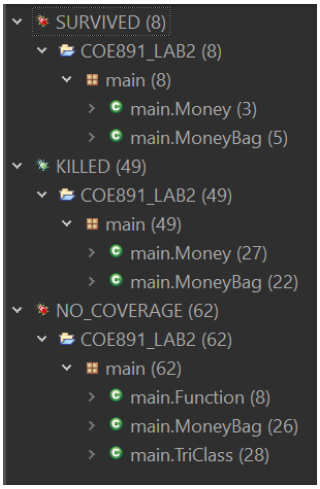
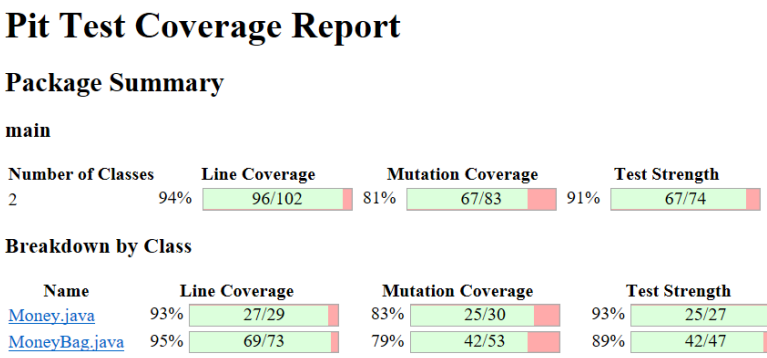


Figure 18: Report of status of the mutants of the Money.java class after alterations.

The mutations report for before and after test class alteration of the MoneyBag.java class is displayed below. The two reports demonstrate the mutants that survived, killed, and did not achieve any coverage. Furthermore, the altered test class was able to achieve more mutator coverage as it increased the effectiveness of the tests by being able to introduce different types of mutators through various test cases. The report of the test class before had covered all mutants and 91 killed mutants while 7 survived mutants. A larger pool of simulated mutants offers a wider range of faults to expose. Additionally, diverse test methods act like probes, reaching deeper into your code to uncover these hidden weaknesses. This combined effort leads to a more accurate assessment of the program's quality and resilience.



Pit Test Coverage Report

Package Summary

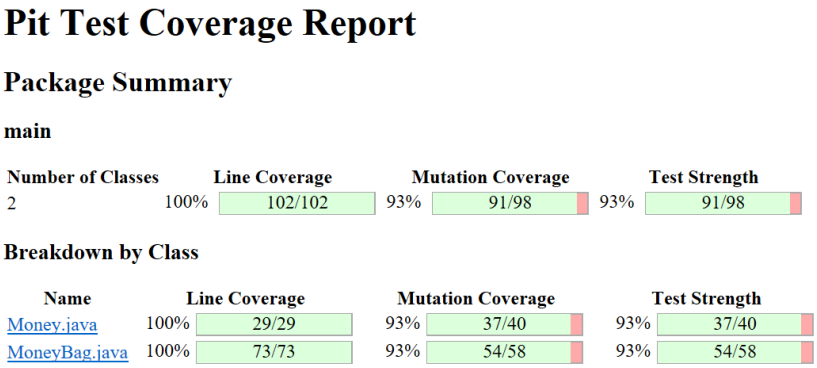
main

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
2	94%	96/102	81%	67/83	91%	67/74

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
Money.java	93%	27/29	83%	25/30	93%	25/27
MoneyBag.java	95%	69/73	79%	42/53	89%	42/47

Figure 19: Report of PIT Test Coverage of MoneyBag.java class before alterations.



Pit Test Coverage Report

Package Summary

main

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
2	100%	102/102	93%	91/98	93%	91/98

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
Money.java	100%	29/29	93%	37/40	93%	37/40
MoneyBag.java	100%	73/73	93%	54/58	93%	54/58

Figure 20: Report of PIT Test Coverage of MoneyBag.java class before alterations.

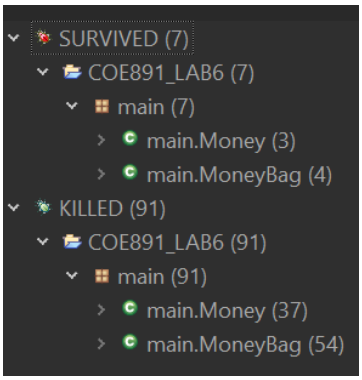


Figure 21: Report of status of the mutants of the MoneyBag.java class after alterations.

7. The following are a list of uncovered mutants are killed by a my own list of mutators for MoneyTest.java and MoneyBagTest.java
    a. List of selected mutators from previous steps that achieved higher mutation coverage:
        - BOOLEAN_FALSE_RETURN: This mutation type alters a method's return statement that originally returns true to return false
            - MoneyBag Lines: 63, 75, 92, 105

    - This mutation aims to expose errors in code that relies on the expected true value. If the code doesn't handle the unexpected false scenario correctly, the test might fail.
- BOOLEAN_TRUE_RETURN: This mutation type flips the logic of a method's return statement that originally returns false to return true instead.
    - MoneyBag Lines: 68, 73, 77, 92, 105
    - This mutation seeks to uncover errors in code that depend on the expected false behavior. If the code isn't designed to handle the unexpected true outcome, the test might fail.
- CONSTRUCTOR_CALL_MUTATOR
    - This mutation type replaces a constructor call with a different constructor call, potentially from the same class or a related class.
    - This mutation can reveal issues where the original constructor is crucial for correct object initialization. If the substitute constructor doesn't provide the necessary setup, the test might fail.
- MATH_MUTATOR: replace mathematical operations within your code with alternative operations. This aims to expose potential errors in calculations due to these modifications.
    - Money Lines: 36
    - MoneyBag Lines: 63, 68, 73, 75, 77, 91, 92, 105
    - If the original calculations are crucial for the code's functionality, these mutations could lead to incorrect results and failed tests in the lines mentioned.
- NULL_RETURN_VALUES: This mutation type modifies a method that originally returns a non-null value to return null instead.
    - Money Lines: 40
    - MoneyBag Lines: 23,27,31,35,84,116,125,130,131,135
    - This mutation targets code that expects a valid return value and might not handle null appropriately. If the code doesn't have null checks or throws an exception when encountering null, the test might fail.
- RETURN_VALS_MUTATOR: This mutation type alters the return value of a method to a different value within a predefined set of options (likely based on the original return type).
    - Money Lines: 36, 40
    - MoneyBag Lines: 23, 27, 31, 35, 63, 68, 73, 75, 77, 84, 91, 92, 105, 116,125,130,131,135
    - This mutation is designed to expose errors in code that relies on the specific return value. If the code doesn't handle other possible return values gracefully, the test might fail.
- VOID_METHOD_CALL_MUTATOR: This mutation type replaces a call to a method that originally returns a value (not necessarily void) with a call to a different method that returns void.
    - Lines: 17, 20, 21, 22, 40, 47, 54, 109, 113, 120, 123, 139, 148
    - This mutation aims to uncover errors in code that depend on the return value of the original method. If the code doesn't handle the absence of a return value correctly (e.g., expecting a specific value to be used later), the test might fail
- EMPTY_RETURN_VALUES
    - Could not cover Money test
- NEGATE_CONDITIONALS_MUTATOR
    - Could not cover Money test

- PRIMITIVE_RETURN_VALS_MUTATOR
    - Could not cover Money test

8. The remaining surviving mutants could not be killed because they likely involved changes that are logically equivalent to the original code, or the mutations were in areas that your testing methods couldn't reach. Here's a breakdown of some surviving mutants and the rationale behind their occurrence :
    - Line 46 in MoneyTest: Replacing the boolean return value of equals with true (survived). This mutation is equivalent to the original logic in certain cases. For instance, the original equals method simply checked if both objects were non-null, then replacing the return value with true might not change the outcome.
    - Line 56 in MoneyTest: Replacing integer addition with subtraction in hashCode (survived). When subtracting a value from fAmount, the implementation of currency().hashCode() mathematically results in the same hash code as adding them.

    In conclusion, achieving 100% mutation coverage is very difficult because some mutations might represent logic that is valid under certain conditions or exist in parts of the code that is not thoroughly tested.