COMPGS03 Validation & Verification
Coursework #1

Antony Hwang
Antony.Hwang.14@ucl.ac.uk

March 8, 2018

1.1 Automation

The library files has been added to the folder and the build.xml script file has been modified to perform mutation testing using PITest. The mutation test can be ran by typing ant pit in the terminal. For evidence please refer to the source code folder.

1.2 Killed Mutants

```
115        private final int getFreq(final Object obj, final Map<?, Integer> freqMap) {
116            final Integer count = freqMap.get(obj);
117  1         if (count != null) {
118  1             return count.intValue();
119            }
120  1         return 0;
```

Figure 1: Mutation Testing Result for CollectionUtils.java

The negate conditional mutator mutates all condition found in this piece of code, by replacing the $!=$ with $==$ on line 117. The mutant is killed, because the mutated condition has caused the test case to fail. For example if the variable count is not null, it will satisfy the original condition and fail the mutated condition, which then will lead to an unexpected output and kill the mutant.

1.3 Improving Mutation Score

Figure 2 below, the lines that are highlighted in red shows the mutants that have not been killed by any of the unit tests. The Figure 3 shows the result, after improvements have been made to the unit tests. The code shown in the figures can be found in LazyList.java located in at `\src\main\java\org\apache\commons\collections4\list\LazyList.java`.

```
111     public E get(final int index) {
112         final int size = decorated().size();
113 2       if (index < size) {
114             // within bounds, get the object
115             E object = decorated().get(index);
116 1           if (object == null) {
117                 // item is a place holder, create new one, set and return
118                 object = factory.create();
119                 decorated().set(index, object);
120 1               return object;
121             }
122             // good and ready to go
123 1           return object;
124         }
125         // we have to grow the list
126 3       for (int i = size; i < index; i++) {
127             decorated().add(null);
128         }
129         // create our last object, set and return
130         final E object = factory.create();
131         decorated().add(object);
132 1       return object;
133     }
```

Figure 2: Before Improvement

```
111     public E get(final int index) {
112         final int size = decorated().size();
113 2       if (index < size) {
114             // within bounds, get the object
115             E object = decorated().get(index);
116 1           if (object == null) {
117                 // item is a place holder, create new one, set and return
118                 object = factory.create();
119                 decorated().set(index, object);
120 1               return object;
121             }
122             // good and ready to go
123 1           return object;
124         }
125         // we have to grow the list
126 3       for (int i = size; i < index; i++) {
127             decorated().add(null);
128         }
129         // create our last object, set and return
130         final E object = factory.create();
131         decorated().add(object);
132 1       return object;
133     }
```

Figure 3: After Improvement

As can been seen in the results, the improvements made to the tests has managed to kill all the mutants that existed in this piece of code. The get(final int index) function accepts an index number as parameter and returns the element in the list at the given index.

113	1. changed conditional boundary → SURVIVED
	2. negated conditional → KILLED
116	1. negated conditional → SURVIVED
120	1. mutated return of Object value for org/apache/commons/collections4/list/LazyList::get to (if (x != null) null else throw new RuntimeException) → NO_COVERAGE
123	1. mutated return of Object value for org/apache/commons/collections4/list/LazyList::get to (if (x != null) null else throw new RuntimeException) → KILLED
126	1. changed conditional boundary → KILLED
	2. Changed increment from 1 to -1 → TIMED_OUT
	3. negated conditional → KILLED
132	1. mutated return of Object value for org/apache/commons/collections4/list/LazyList::get to (if (x != null) null else throw new RuntimeException) → KILLED

Figure 4: Mutations

The mutant that survived on line 113 was the changed conditional boundary, this mutator replaced the condition (index $<$ size) with (index $<=$ size). The unit tests did not test the function with the index same as the size of the list. Hence, the conditional boundary mutant survived. Another mutant that survived on line 116 was the negated conditional, this mutator replaced the (object $==$ null) with (object $!=$ null). The lines that were highlighted in red from 118 to 120 was not covered by any of the unit tests. The unit tests did not test the function with the index that points to a null element in the list. Hence, the negated conditional mutant survived and did not cover the rest of the lines form 118 to 120.

The following Figures shows the modified unit test code in the file located at `\src\test\java\org\apache\commons\collections4\ListUtilsTest.java`.

```java
@Test
public void testLazyListGet() {
    final List<Integer> list = ListUtils.lazyList(new ArrayList<Integer>(), new
        Factory<Integer>() {

        private int index;

        public Integer create() {
            index = 1;
            return 1;
        }
    });
    Integer result = list.get(0);
    assertSame(1, result);
}
```

Figure 5: Improved Unit Test

The test shown in Figure 5 has been added to kill the conditional boundary mutant. The unit test code first initializes an empty list with size of 0, then it retrieves the element at index 0 with the get function. This kills the mutant as the the index is equal to the size of the list. It will fail the condition on line 113 and satisfy the condition on the mutated condition. Hence, it will get a different output with the mutated condition and fail the test.

```java
@Test
public void testLazyListGetNull() {
    final List<Integer> list = ListUtils.lazyList(new ArrayList<Integer>(), new
        Factory<Integer>() {

        private int index;

        public Integer create() {
            index = 1;
            return 1;
        }
    });
    list.add(null);
    Integer result = list.get(0);
    assertSame(1, result);
}
```

Figure 6: Improved Unit Test

The test shown in Figure 6 has been added to kill the negated conditional mutant. The unit test code first initializes an list with a null element at index 0, then it retrieves the element at index 0 with the get function. This kills the mutant as the object is equal to null, it will satisfy the condition on line 116 and fail on the mutated condition. Hence, it will get a different output with the mutated condition and fail the test.

Number of Classes	Line Coverage		Mutation Coverage	
269	45%	5975/13251	41%	3522/8549

Figure 7: Before Improvement

Number of Classes	Line Coverage		Mutation Coverage	
269	45%	5978/13251	41%	3525/8549

Figure 8: After Improvement

The Figures above shows the mutation score of the entire project before and after the improvements. As can been seen the line coverage and mutation coverage, both has been increased by 3. For the break down of the report please refer to the pit report located in the pitReports folder.

END OF COURSEWORK