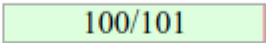
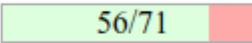
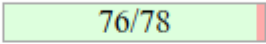
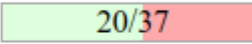
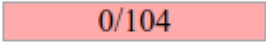
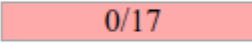
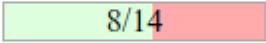
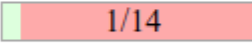
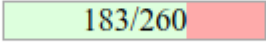
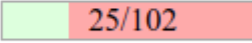
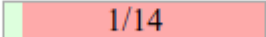
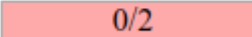


CS 362 – Software Engineering 2  
Professor Aburas Ali  
Assignment 3 – Mutation Testing and Coverage  
Khuong Luu (luukh)

## 1. Screenshots

### Original

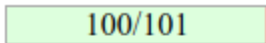
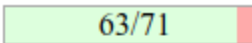
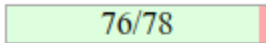
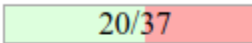
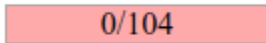
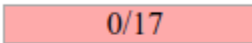
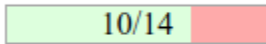
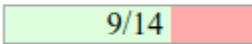
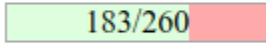
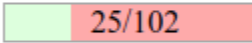
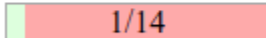
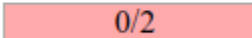
#### Breakdown by Class

Name	Line Coverage	Mutation Coverage
<a href="#">Appt.java</a>	99% 	79% 
<a href="#">CalDay.java</a>	97% 	54% 
<a href="#">CalendarMain.java</a>	0% 	0% 
<a href="#">CalendarUtil.java</a>	57% 	7% 
<a href="#">DataHandler.java</a>	70% 	25% 
<a href="#">XmlParserErrorHandler.java</a>	7% 	0% 

---

Report generated by [PIT](#) 1.2.0

### Final

Name	Line Coverage	Mutation Coverage
<a href="#">Appt.java</a>	99% 	89% 
<a href="#">CalDay.java</a>	97% 	54% 
<a href="#">CalendarMain.java</a>	0% 	0% 
<a href="#">CalendarUtil.java</a>	71% 	64% 
<a href="#">DataHandler.java</a>	70% 	25% 
<a href="#">XmlParserErrorHandler.java</a>	7% 	0% 

---

Report generated by [PIT](#) 1.2.0

## 2. Did you take less than 90% mutation coverage for each class?

Yes. Explain: It's because the program actually has many bugs, both from which I intentionally introduced, from which I unintentionally introduced, and also a new *huge* bug that I found. These bug prevented my test cases from passing, which in turn prevented Pitest from generating the mutation coverage report. Therefore, I had to comment out those assertion statement in order to let my failed test cases pass. For those test cases whose assertions statement are commented out, the results from these

was that my tests even though did cover lines, but the missing of assertion statements made my mutation coverage go way down since my test didn't actually "test" it.

### 3. Point out some code, if any, that your "final" test cases do not kill/cover mutants (i.e., survived). Why?

Here are some of the code that my tests could not cover. I hereby include all types of scenarios that happen for many reasons: untestable, unfeasible to test, unpractical to test, and actual bugs

- Could not cover the setXmlElement(null) method;

```
115 1 setRecurrence(recurringDays, REC
116
117 //Leave XML Element null
118 1 setXmlElement(null);
119
120 //Sets valid to true - this is n
121
```

because it's a private attribute thus no accessors method to get value; and thus it's impossible to test

- Could not the setRecurrence method and the methods it calls:

```
292 */
293 public void setRecurrence(int[] recurDays, int recurBy, int recurIncrement, int recurNumber) {
294 1 setRecurDays(recurDays);
295 1 setRecurBy(recurBy);
296 1 setRecurIncrement(recurIncrement);
297 1 setRecurNumber(recurNumber);
298 }
299 private void setRecurDays(int[] recurDays) {
300 1 if (recurDays == null) {
301     this.recurDays = new int[0];
```

This is because there are actual bugs I that I introduced there and I had to comment out assert statement so that pitest report can be generated; otherwise pitest will not generate report

```
public void testSetRecurrence() throws Throwable {
    Appt appt = new Appt(15, 30, 9, 12, 2018, "Birthday Party", "This is my birthday party",
"xyz@gmail.com");
    appt.setRecurrence(null, 0, 0, 0);

    int[] expectedRecurDays = new int[0];
    int[] recurDays = appt.getRecurDays();
    //assertEquals(expectedRecurDays, recurDays);
    int recurBy = appt.getRecurBy();
    //assertEquals(0, recurBy);
    int recurIncrement = appt.getRecurIncrement();
    //assertEquals(0, recurIncrement);
    int recurNumber = appt.getRecurNumber();
    //assertEquals(0, recurNumber);
}
```

- Could not cover the rest of the branch in getApptRange in DataHandler

```

244         int daysDifference = 0;
245         nextDay = (GregorianCalendar)firstDay.clone();
246         Iterator itr = apptOccursOnDays.iterator();
247         while (itr.hasNext()) {
248             GregorianCalendar apptOccursOn = (GregorianCalendar)itr.next();
249
250             if (diagnose) {
251                 System.out.println("\t" + apptOccursOn);
252             }
253
254             while(nextDay.before(apptOccursOn)) {
255                 daysDifference++;
256                 nextDay.add(nextDay.DAY_OF_MONTH, 1);
257             }
258
259             CalDay calDayOfAppt = (CalDay)calDays.get(daysDifference);
260             calDayOfAppt.addAppt(appt);
261
262         }
263
264         //This appointment has been added to all CalDays
265         if (diagnose) {
266             System.out.println("This appointment is done.");
267         }
268     } //for nodelist
269     return calDays;
270 }

```

This is because of a huge bug I could not find. The bug may be caused by me or by the original author and so far I have not been able to fix this bug. This bug make the rest of the code file unreachable because there is a condition in which no matter the case we put in, it always return false and return out of the method. However, it's a good thing that the mutation coverage discover this bug while the line coverage could not.

#### 4. Did you find any “new” bugs?, if yes, describe the bug and the test case that detected it; and if no, explain why your test suites did not detect any “new” bugs?

The mutation coverage has discovered a bug that the line coverage could not. The bug is described in the last section of my answer for question (3) (just above this answer).

#### 5. Overall experience

- What problems did you face in applying mutation analysis?, and how did overcome them?

I encounter 2 major problems:

(1) At first, I couldn't even get the Pitest running and generate report. I searched the error but also could not find the solution. So I email the instructor and came to office hour. He pointed out that the it was because my test has failed and Pitest is only able to generate report once all the tests are “green,”

which basically mean all the tests have passed. I was the first person to find out, ask, and address this problem because I started this assignment right in the night it was announced.

(2) The most irritated and annoying about Pitest is that even though it generated report well and nicely, the details are not much deep nor helpfully specific. For example, it reported that one of the mutants for one of my if-statements was able to survive my test. There were two mutations and 1 of them survived, but Pitest didn't generate **how and why** that mutation was able to survived while the similar one which was similar to it was killed. Which value Pitest has mutated to those 2 mutations and which one survived? Pitest didn't tell me. So, I have to turned back to the source code and write more and more test that cover every single the combination of that if-statement's condition. In the end, my additional tests were able to kill that survived mutation; but the experience wasn't good because I think I should be able to see the reason that mutation survived so I can address that issue faster and more efficient.

- **Discuss your overall experience with using the mutation tool.**

My second point (2) in the above section also describe my overall experience using Pitest in this assignment. It's helpful and can do its jobs. However, I wish it could be more specific so the user can spend less time investigate and make "guess work" about how did the/those mutation(s) was/were able to survive and address them in less time.

Another complain I would have is that Pitest generate a new report folder every single time I run it instead of having an option or config that can allow the new report override the previous one so that I don't have to turn on my file browser and open the new report in a new folder instead of just refreshing the web page like I conveniently did with other line and branch overage tool