Nitin Nagavel

Assignment #1 Software Testing

Com S/SE 417 Spring 2022

Handed out Feb 1st, 2022

Due at 11:59 PM, Feb 10t, as a pdf uploaded to Canvas

Homework Policy

Homework Policy: The homework assignment should be done individually. You may talk to classmates about the problems in general, and you can help each other with getting the tools running, but you must complete the homework on your own. You are not permitted to use published answers from websites, etc. Assistance by others must be specifically credited in the solution to the problem that is turned in, describing what the contribution was (e.g., "Thanks to [name] for explaining X in Problem Y", not "Thanks to [name] for help with HW1."). The Dean of Students Office offers several good resources on how to avoid plagiarism.

The goal is for each of you to learn the material sufficiently well to use it productively, to think innovatively, and to develop confidence in your problem-solving abilities. Feel free to talk to me individually about this if you have any questions.

Late policy: *10% penalty per day* (or part of day) for late homework. ***Assignments will not be accepted after Feb 14th***, unless otherwise arranged/discussed with me.

Some homework problems are adapted from the course textbook, "Introduction to Software Testing", 2nd edition, Ammann & Offutt, 2017. There is a Student Solution Manuals available online with answers to other practice questions https://cs.gmu.edu/~offutt/softwaretest/exer-student.pdf.

In this assignment we will use a code coverage tool for Java, Jacoco (version 0.8.7)

https://www.eclemma.org/jacoco/

I have put a version in Canvas with the triangle program and demoed this in class. Please see directions for running Jacoco with that Program. You can use that infrastructure to start this assignment. All you will need to do is to add a new directory inside of the examples folder with the files for this assignment.

I have put a zipped file of a directory called "**power**" containing a program that computes the exponents of numbers (with a maven pom.xml file and the necessary file structure). You need to copy this to the "examples" directory in Jacoco to make this work. Jacoco will run out of the box on pyrite. You can also run this locally on your own machines (if you want to set it up). You will need Maven and Java to run

this on your home machine. I have provided directions for running on Pyrite if you choose to use that version.

Program Specifications: Power.java

This class has two functions that compute the power of two numbers. The first called power is defined as: **power(left,right) = left^{right}** . Any number to the zero power is equal to 1.

Negative exponents (e.g. 3^{-2}) return -1 (the program does not calculate fractions).

The power function limits the size of the left and right inputs to keep the program from overflowing.

The second function computes the power but inverses the left and right. **Inverse(left,right)=right^{left}**. This program does not limit the size of the inputs. Ignore that for now and only use test inputs on inverse that will create a return within a valid integer range.

For each numbered item (except #1), answer the questions, and provide screen shots demonstrating the resulting reports from Jacoco. This should include the Junit test details, the coverage report and the visualization of the source code showing coverage. Create a .pdf of the final report and submit via canvas. Make sure to clearly label the question #s.

1. Download the Jacoco distribution from their website or from Canvas. Download the 'power' directory from canvas and place the contents inside of the examples directory as well. You will see I have given you a couple of Junit tests to start.

2. Set up the Jacoco program to run either on pyrite and/or your local machine. Run the initial test cases using the "mvn" commands as shown in class (and provided on the lecture notes). This is run inside of the **power** main directory

   Go to: directory … jacoco-0.8.7/examples/power
   >mvn clean
   >mvn test (will create junit report)
   >mvn jacoco:restore-instrumented-classes
   >mvn jacoco:report. (will create the coverage reports)

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 01:29 min
[INFO] Finished at: 2022-02-07T20:03:46-06:00
[INFO] ------------------------------------------------------------------------
home4nit@pyrite-n1 power]$ mvn jacoco:restore-instrumented-classes
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------< org.jacoco:org.jacoco.examples.maven >----------------
[INFO] Building JaCoCo Maven plug-in example with Offline Instrumentation 0.8.7
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- jacoco-maven-plugin:0.8.7:restore-instrumented-classes (default-cli) @ org.jacoco.examples.maven ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 24.388 s
[INFO] Finished at: 2022-02-07T20:06:17-06:00
[INFO] ------------------------------------------------------------------------
home4nit@pyrite-n1 power]$ mvn jacoco:report
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------< org.jacoco:org.jacoco.examples.maven >----------------
[INFO] Building JaCoCo Maven plug-in example with Offline Instrumentation 0.8.7
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- jacoco-maven-plugin:0.8.7:report (default-cli) @ org.jacoco.examples.maven ---
```

3. Look at the initial reports (in the target directory) for JUnit (under surefire) and for Code Coverage (under sites). You should keep these (to hand in). ***Write a short (sentence or two) summary of what these reports tell you and attach the reports to the assignment.***

PowerTest.txt - Notepad

File Edit Format View Help

```
-------------------------------------------------------------------------------
Test set: PowerTest
-------------------------------------------------------------------------------
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 4.021 sec
```

JaCoCo Maven plug-in example with Offline Instrumentation

JaCoCo Maven plug-in example with Offline Instrumentation

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
default		47%		35%	10	13	9	17	1	3	0	1
Total	34 of 65	47%	13 of 20	35%	10	13	9	17	1	3	0	1

According to the Junit Report, the tests have successfully passed. However, with the Code Coverage report the coverage of code tested is 35 percent meaning that all the code hasn't been successfully tested.

4. Now add additional tests to increase both the line and branch coverage as high as you can. Aim for 100 percent coverage, however this may not be possible – do not change the program to achieve higher coverage. You must do this only with test cases.

- **List the test cases you have added and for each state:**
  (a) The input, the oracle, and the reason you added it (i.e. what condition/statement are you trying to cover – give line numbers from the Jacoco report)

```
@Test public void PowTest3(){
assertEquals(myPow.power(10,5),100000);

}
```

In line 24, I am covering the max value of the first if condition for the left and right values

```
@Test public void PowTest4(){
 assertEquals(myPow.power(5,10),9765625);

}
```

In line 24, I am covering the max value of the second if condition for the left and right values

```
@Test public void PowTest5(){
 assertEquals(myPow.power(0,5),0);

}
```

In line 24, I am covering the lowest values of the first if conditon

```
@Test public void PowTest6(){
 assertEquals(myPow.power(10,0),-1);

}
```

For lines 26 – 29, the first if condition sets the result to -1 if the right value is 0 and this test satisfies that

```
@Test public void InvTest1(){
 assertEquals(myPow.inverse(5,0), 1);
}
```

For lines 59 – 62, I chose a random left number but set the right number to 0 to satisfy the condition.

```
@Test public void InvTest2(){
 assertEquals(myPow.inverse(3,2), 8);
}
```

For lines 65 – 67 I am satisfying the for loop by setting the left number greater than 2 and checking if the inverse is correct.

```
@Test public void InvTest3(){
 assertEquals(myPow.inverse(2,4), 16);
}
```

For the final test case, I chose left = 2 and right = 4 as a test case that won't fall within the lines of code specified from the first condition at lines 57 – 59 and had multiple iterations of the loop

- **Provide screen shots of the reports demonstrating the resulting coverage.**

5. What is the **maximum line and branch coverage** for each method you have reached? If it is not 100 percent, explain why you were not able to obtain that goal.

JaCoCo Maven plug-in example with Offline Instrumentation

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
default		92%		75%	5	13	2	17	0	3	0	1
Total	5 of 65	92%	5 of 20	75%	5	13	2	17	0	3	0	1

I was not able to reach 100 percent on both methods because the code for each method didn't include all of the conditions to satisfy the function.

6. Identify at least two **faults *not due to numeric overflow in the program*.** For each, state if they lead to failures (and if so provide a test case). If not explain **why not using the RIPR model**.

Fault 1: Line 61 where if the right value = 0, it results in 1 which is wrong.
Test Case: $0^1 = 0$ and not 1

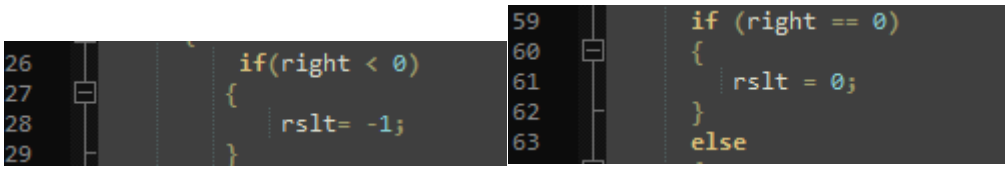
Fault 2: Line 26 where the right value <= 0, then it will return a -1. This is wrong and it should be right < 0 for any negative exponents/value.

```
@Test public void PowTest7(){
 assertEquals(myPow.inverse(1,0),0);
}

@Test public void PowTest8(){
 assertEquals(myPow.inverse(2,0),1);
}
```

7. **Try fixing the faults you have found** and see if you can increase the code coverage – explain what you discover and state how you fix the faults (provide line numbers and fixes – or screenshots)

I will change line 26 to right < 0. This covers all of the negative values only. On line 61 I changed rslt = 0. This satisfies the inverse



8. The first method (power) has some checks to prevent numeric overflow, however it misses one case. Find a test case that will cause an overflow (and fail). **Does adding this test change the code coverage at all**? Discuss why or why not and the implications for testing.

```
@Test public void PowTest8(){
        assertEquals(myPow.inverse(2,11),2048);
  }
}
```

JaCoCo Maven plug-in example with Offline Instrumentation

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
default		92%		75%	5	13	2	17	0	3	0	1
Total	5 of 65	92%	5 of 20	75%	5	13	2	17	0	3	0	1

This test case doesn't change the coverage because some of the previous tests may have performed the coverage due to altered code.