CSE 4321/5321 Homework 5

Spring 2019

Question Weighting:

Question 1-4 - 25% each

Use the following approach for each problem **(all test must use the JUnitParams runner and read values from csv files except as noted).**

1. Develop CFG (reduced) and cyclomatic complexity.
2. Develop basis path set.
3. Determine significance on each variable.
4. Develop inputs and expected outputs from requirements, not code.
5. Add tests for missing Boundary Values not tested, including extreme range values.
6. Add tests for extreme range values for each variable that has a BV.
7. Add MCDC test cases for Multiple Condition Decision statements.
8. Add test cases to verify all table data.

**Submit the following in the PDF file - this is the evidence file**

1. Test case table snapshot
   1. Basis Path test cases (for problem 1 only)
      1. Use the line numbers in Eclipse for your basis path line.
      2. where tests are addition to basis path set use a "-" to indicate the basis path.
      3. Make sure all true is the first BP and tests are in correct order
      4. You do NOT need to submit the CFG with this homework
   2. Indicate tests for MCDC coverage with a "Statement 11 FFT" - put this in the comment column.
2. JUnit pass indicator (green bar expanded)
3. JaCoCo statement green source line annotations (not summary)
4. Make sure to include the time stamp on your screen shots.

**Include in ZIP file**

1. Your evidence file (above)
2. JUnit test files (make sure problem number is referenced in the file name)
3. csv files used (make sure problem number is referenced in the file name)
4. For Problem 1 corrected source code file (if required)
5. For Problem 2 see the list of submission items
6. For Problem 4 your Problem4ServerData.class file

**Problem 1**) Develop the test cases for Problem 1 source code (in the attached zip file). The specification (requirements) for the code are described below. The specification is correct but may not be optimal. If there are any errors (in the code only) or optimizations (requirements and/or code) submit the updated code with your solution set. Your tests must fully test the code (or updated code) to the requirements. Significance is to the Cent. Please do not use truncation on the calculation of total.

**Problem Specification**

The code shall compute two outputs: total and specialOrderStatus as described by the following requirements.

**total** shall be computed based on the value of ordAmount as shown in the following table



**specialOrderStatus** shall be calculated as follows

specialOrderStatus =

(deepDish) && ((ordAmount > 50.0) || (numPizzas > 4) ) ||

(!deepDish) && ((ordAmount <= 50.0) || (numPizzas > 4) || (ordAmount > 50.0)&&(numPizzas <= 4) ) ﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿

simplifies to: specialOrderStatus = (!deepDish) || (ordAmount > 50.0) || (numPizzas > 4);

**input ranges**:

$0.00 <= ordAmount <= $5,000.00

0 <= numPizzas <= 1,000

**Recode**

If you are forced to re-write the code - arrange boundary conditions starting with the largest boundary condition to smallest in the same style and manner as the original code.

**Test**

Use the following template for the test case table. Document how the multiple condition expression is tested using MCDC in the Comments column (see the Test case table below) and as described above. The comments column should also describe the basis path set. The comparison threshold for total is 0.01



**Problem 2**) Run JUnit, JaCoCo, and PIT (PITclipse) against the Blackboard provided test cases for the Problem 2 code. Develop your test cases to cause PIT to be **all green**. You may ONLY change the **test case table** NOT the source code.

For this problem submit the following:

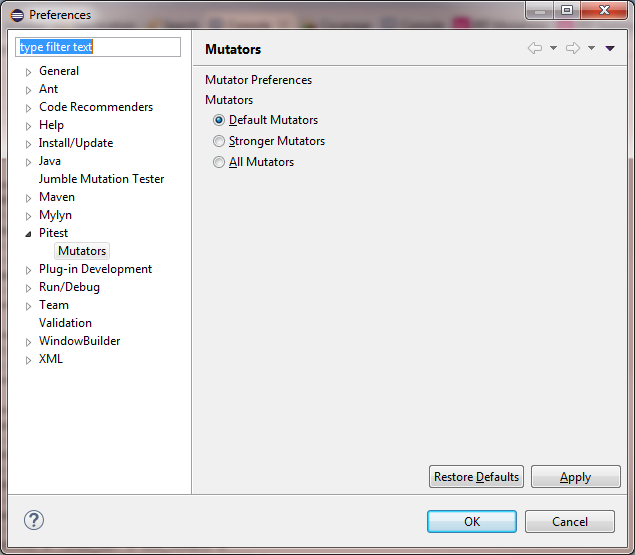
1. Take snapshots of the JaCoCo coverage of the **provided** test cases.
2. PIT results of the **provided** test cases. Explain why PIT is not all green when JaCoCo is.
3. Take snapshots of the JaCoCo coverage of **your** test cases.
4. PIT results of **your** test test cases.
5. Submit your test case table as an .xlsx file. Make sure in the comments column to document the MCDC test cases for the source code.

**PIT setup instructions**

Load PITClipse as follows:

* In Eclipse, go to Help -> Eclipse Marketplace... type "Pitclipse" in the search bar. Once found click Install and follow the instructions from there.

InEclipse select Window -> Preferences... then Pitest and expand as follows:, **select the All Mutator option of "Mutators"**



expand

**NOTE: You run PIT against your JUnit test file - NOT the Java source file**

The comparison threshold for calcTotal is 0.01.

0 <= memberPoints <= 1,000

$0.00 <= total <= $5,000.00

**Problem 3**) Develop the test cases for Problem 3 source code (in the attached zip file). This code is used to assist a small business in keeping count of boxes shipped. They use the railroad exclusively to ship their merchandise.



A shipment is a set of 10 railroad cars each of which may hold either 180, 190, 200, 210, or 215 boxes. A shipment is a total of 2,000 boxes. The railroad cycles through the 10 cars as represented by the following box capacity schedule:

Schedule 1: 190, 200, 210, 215, 190, 200, 215, 210, 190, 180

Schedule 2: 190, 210, 200, 215, 190, 200, 215, 210, 190, 180

Schedule 1 is used normally. Schedule 2 is only used when the shipment number is divisible by 25 or when the shipment number is divisible by 5 but not by 10.

Given the box number in the car, the railroad car number (1-10) and the shipment number the software calculates the **absolute** box number of the 2,000 box shipment as follows:



Box In Car Number, RailRoad Car number, and Shipment Number are all positive integers. Assume Shipment Number has a maximum of 100,000 for testing purposes.

Use the following format for your test case table.



**Problem 4**) Develop the test cases for Problem 4 source code (in the attached zip file). You will be able to use the exact same test case table as Problem 2. Use EasyMock to mock the call to the server to get the memberPoints.

EasyMock instructions

* + 1. Download EasyMock from the M14 Blackboard files
    2. InstallEasyMock in your project (add the EasyMock.jar to your Java Build path)

Execute the JUnit test. Create **Problem4ServerData.java** to define the signature for the getMemberPoints() method (see M14 for how to do this). Remember to follow the steps shown in slide 35 of M14 to get EasyMock to work in your test environment.