# File Content

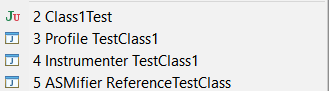
|  |  |
| --- | --- |
| **Src\** |  |
| Instrumenter.java | A 20-line code to instrument a code given it’s class name |
| InstrumenterClassVisitor.java | Contains both classVisitor and methodVisitor. The students will be filling this mainly. |
| Profiler.java | The main profiler class |
| CoverageHelper.java | A helper class that stores all coverage stored during profiling. |
| ReferenceTestClass.java | A reference class that the student can ASMify to show how to construct it using ASM visit calls only |
| TestClass1.java | A test java class that the student will be profiling. Many others can be created. |

|  |  |
| --- | --- |
| **test\** |  |
| Class1Test.java | A test class that instruments and profiles TestClass1, and tests if its profile matches several criteria based on a reference profile. So far, I included 3 tests for method coverage and method pair coverage |

|  |  |
| --- | --- |
| **output\** |  |
|  | A temporary directory that contains instrumented classes. |

# Predefined Run Configurations

For the convenience of students, the following run configurations are provided.



**Import the following files Run configurations** from ***#RunConfigurations\***

|  |  |
| --- | --- |
| ASMifier ReferenceTestClass : | Run ASMifier on ReferenceTestClass |
| Instrumenter TestClass1: | Instrument TestClass1 |
| Profile TestClass1: | Run the instrumented TestClass1 |
| Class1Test: | Check if the profile meets several criteria. |

# Steps

## Step 0

Re Introduce how events occur in ASM.

Students should change the code to **save** the class name and method name in the Class and Method visitors. The locations of these changes should be marked in the java file.

## Step 1

To profile executed methods, we would like to call the function **Profiler.**handleMethodEntry every time a method executes. To achieve this goal, we must determine:

1. The bytecode that executes such a call
2. The corresponding ASM visitor calls to the classWriter

For the purpose of this step, you are provided with a ReferenceTestClass that you can use to determine the corresponding ASM visit calls. For the following input, you can use the predefined Run configuration ASMifier ReferenceTestClass to determine the code below.

|  |
| --- |
| Profiler.*handeMethodEntry*("TestClass1", "main", "x"); |

|  |
| --- |
| ...  methodVisitor.visitLdcInsn("TestClass1");  methodVisitor.visitLdcInsn("main");  methodVisitor.visitLdcInsn("x");  methodVisitor.visitMethodInsn(INVOKESTATIC, "Profiler", "handeMethodEntry", "(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;)V", false);  ... |

The final step is to understand where to place this code. The sequence of events that can occur when parsing a method is displayed below. visitCode is the first event to occur before further instructions are visited. Go to InstrumenterClassVisitor and modify the visitCode method to add the instructions above.

|  |
| --- |
| visitAnnotationDefault?  ( visitAnnotation | visitParameterAnnotation | visitAttribute )\*  ( **visitCode**  ( **visitTryCatchBlock** | visitLabel | visitFrame | **visitXxxInsn** |  **visitLocalVariable** | **visitLineNumber** )\*  visitMaxs  )?  **visitEnd** |

**To test your code:**

Run the configurations in that order:

1. Instrument TestClass1
2. Profile TestClass1

and observe the output displaying which methods were called.

For your convenience, a series of unit tests are provided to make sure that your code works. Run Junit Class1Test. The following tests should pass: testCoveredMethods and testNonCoveredMethods

## Step 2 – Method Pairs