CS472/672: Software Product Design and Development I

Howard R. Hughes College of Engineering
University of Nevada, Las Vegas
Assignment 2 – Software Testing (Dynamic Analysis)

Daniel Ogenrwot
NSHE: 2002476057
ogenrwot@unlv.nevada.edu

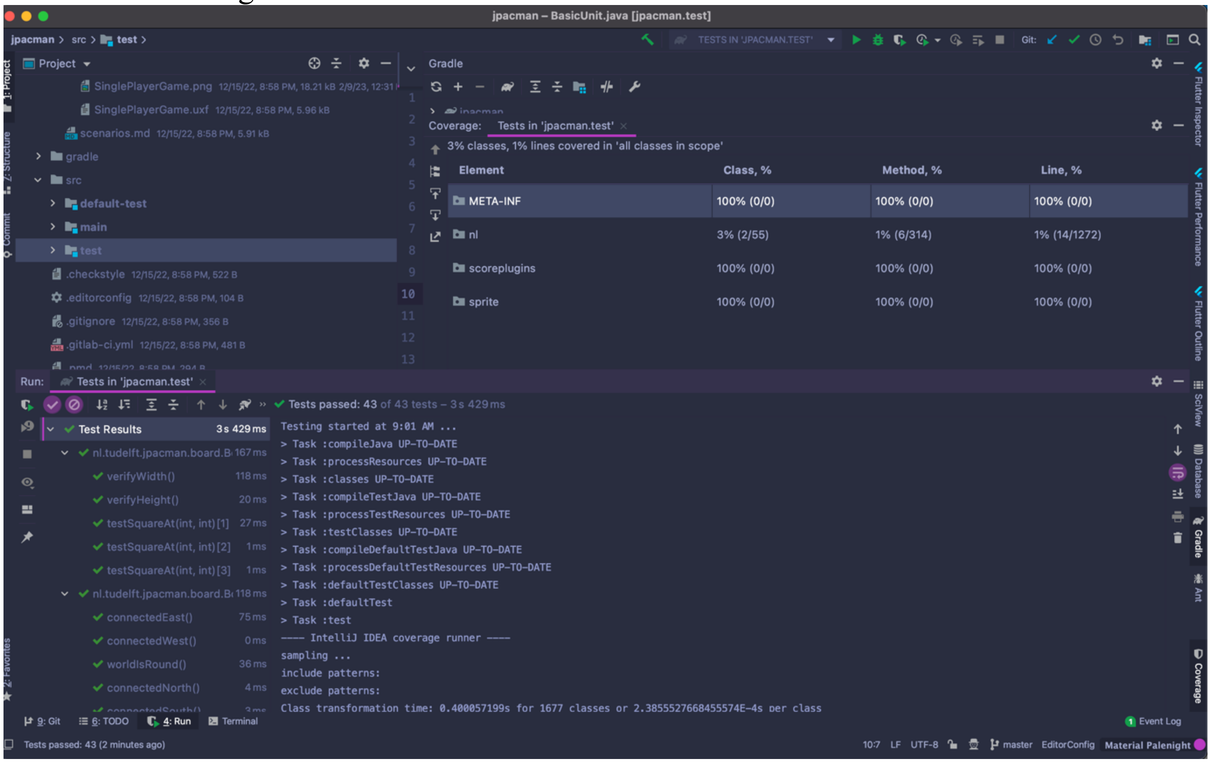
September 25, 2023

Report:

Link to Repository: https://github.com/danielogen/CS-472-2023-GROUP-2

Task 1 – JPacman Test Coverage

The figure below shows the screenshot of successfully running tests with coverage. The default test coverage runner is IntelliJ IDEA.


jpacman – BasicUnit.java [jpacman.test]
jpacman > src > test >
Project
SinglePlayerGame.png 12/15/22, 8:58 PM, 18.21 kB 2/9/23, 12:31
SinglePlayerGame.uxf 12/15/22, 8:58 PM, 5.96 kB
scenarios.md 12/15/22, 8:58 PM, 5.91 kB
gradle
src
default-test
main
test
.checkstyle 12/15/22, 8:58 PM, 522 B
.editorconfig 12/15/22, 8:58 PM, 104 B
.gitignore 12/15/22, 8:58 PM, 356 B
.gitlab-ci.yml 12/15/22, 8:58 PM, 481 B
Gradle
Coverage: Tests in 'jpacman.test'
Run: Tests in 'jpacman.test'
9: Git
6: TODO
4: Run
Terminal
Tests passed: 43 (2 minutes ago)

3% classes, 1% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
META-INF	100% (0/0)	100% (0/0)	100% (0/0)
nl	3% (2/55)	1% (6/314)	1% (14/1272)
scoreplugins	100% (0/0)	100% (0/0)	100% (0/0)
sprite	100% (0/0)	100% (0/0)	100% (0/0)

```
✓ Tests passed: 43 of 43 tests – 3s 429ms

✓ Test Results                          3s 429ms
  ✓ nl.tudelft.jpacman.board.B          167ms
      ✓ verifyWidth()                   118ms
      ✓ verifyHeight()                  20ms
      ✓ testSquareAt(int, int)[1]       27ms
      ✓ testSquareAt(int, int)[2]       1ms
      ✓ testSquareAt(int, int)[3]       1ms
  ✓ nl.tudelft.jpacman.board.B          118ms
      ✓ connectedEast()                 75ms
      ✓ connectedWest()                 0ms
      ✓ worldIsRound()                  36ms
      ✓ connectedNorth()                4ms
```

```
Testing started at 9:01 AM ...
> Task :compileJava UP-TO-DATE
> Task :processResources UP-TO-DATE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources UP-TO-DATE
> Task :testClasses UP-TO-DATE
> Task :compileDefaultTestJava UP-TO-DATE
> Task :processDefaultTestResources UP-TO-DATE
> Task :defaultTestClasses UP-TO-DATE
> Task :defaultTest
> Task :test
---- IntelliJ IDEA coverage runner ----
sampling ...
include patterns:
exclude patterns:
Class transformation time: 0.400057199s for 1677 classes or 2.3855527668455574E-4s per class
```

Figure 1: Showing initial tests running with coverage on IntelliJ

Question: Is the coverage good enough?

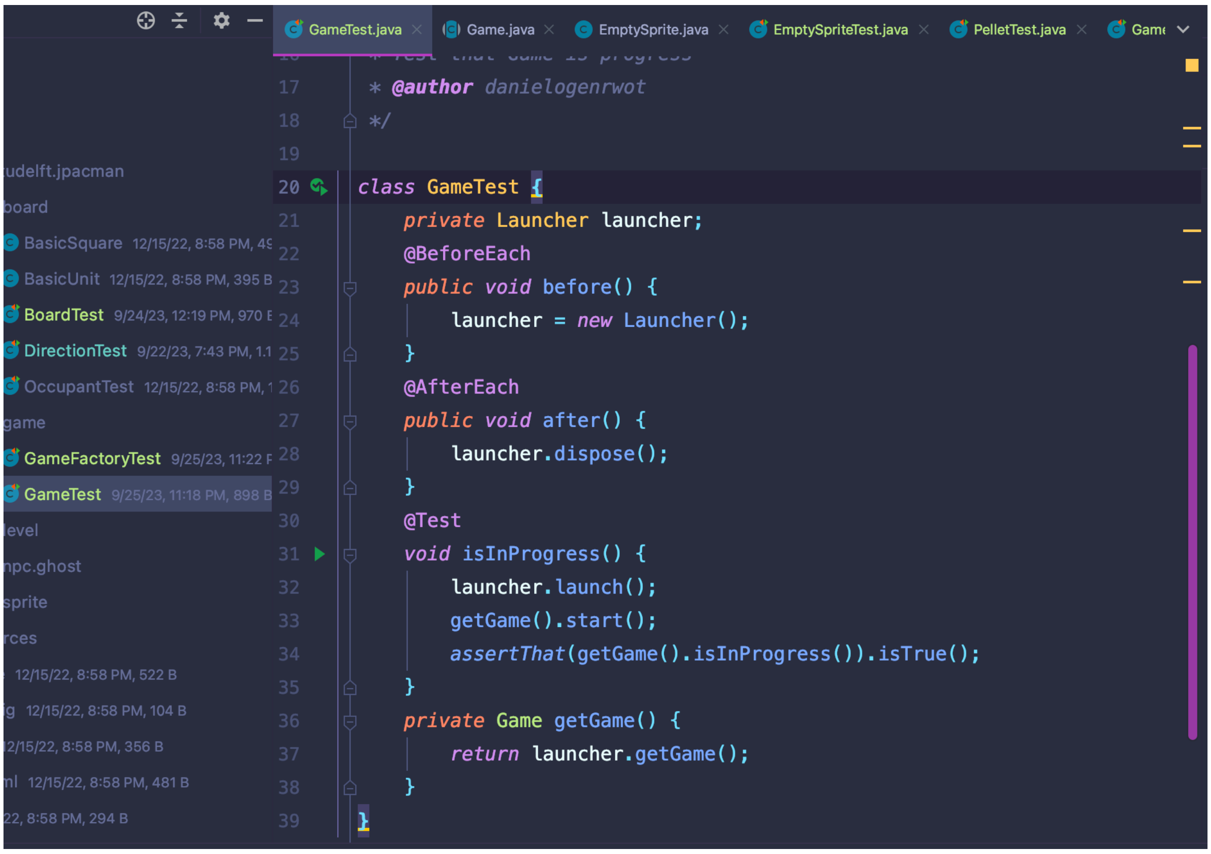
- This coverage is NOT good enough since only 3% (2/55) classes, 1% (6/314) methods and 1% (14/1272) lines of code is covered by the initial test cases. You need to write more tests to increase coverage.

Task 2.5 – Increasing Coverage on JPacman

The screenshots below show test cases written for the following classes:

Fully Qualified Class Name
src/main/java/nl/tudelft/jpacman/game/Game.isInProgress
src/main/java/nl/tudelft/jpacman/game/GameFactory.getPlayerFactory
src/main/java/nl/tudelft/jpacman/board/Board.getHeight
src/main/java/nl/tudelft/jpacman/board/Board.getWidth
src/main/java/nl/tudelft/jpacman/sprite/EmptySprite.splite

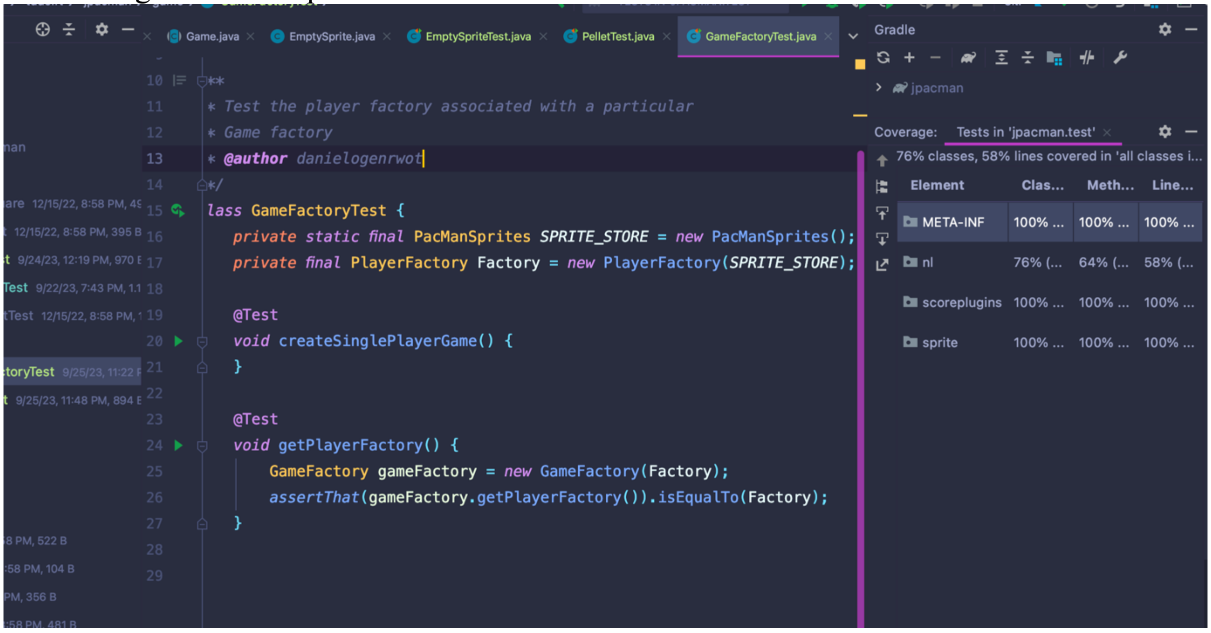
To test that the game is in progress, we create a launcher object and start the game. We create two methods to setup the test with the decorators @BeforeEach and @AfterEach



```java
 * Test that Game is progress
 * @author danielogenrwot
 */

class GameTest {
    private Launcher launcher;
    @BeforeEach
    public void before() {
        launcher = new Launcher();
    }
    @AfterEach
    public void after() {
        launcher.dispose();
    }
    @Test
    void isInProgress() {
        launcher.launch();
        getGame().start();
        assertThat(getGame().isInProgress()).isTrue();
    }
    private Game getGame() {
        return launcher.getGame();
    }
}
```

Figure 2: Screenshot: Testing isInProgress method in Game class

Figure 3 below shows the process of testing player factory associated with a particular game factory. As shown, we used to the getPlayerFactory() to check that a playerFactory was created with the right PacManSprites.



```java
/**
 * Test the player factory associated with a particular
 * Game factory
 * @author danielogenrwot
 */
lass GameFactoryTest {
    private static final PacManSprites SPRITE_STORE = new PacManSprites();
    private final PlayerFactory Factory = new PlayerFactory(SPRITE_STORE);

    @Test
    void createSinglePlayerGame() {
    }

    @Test
    void getPlayerFactory() {
        GameFactory gameFactory = new GameFactory(Factory);
        assertThat(gameFactory.getPlayerFactory()).isEqualTo(Factory);
    }
```

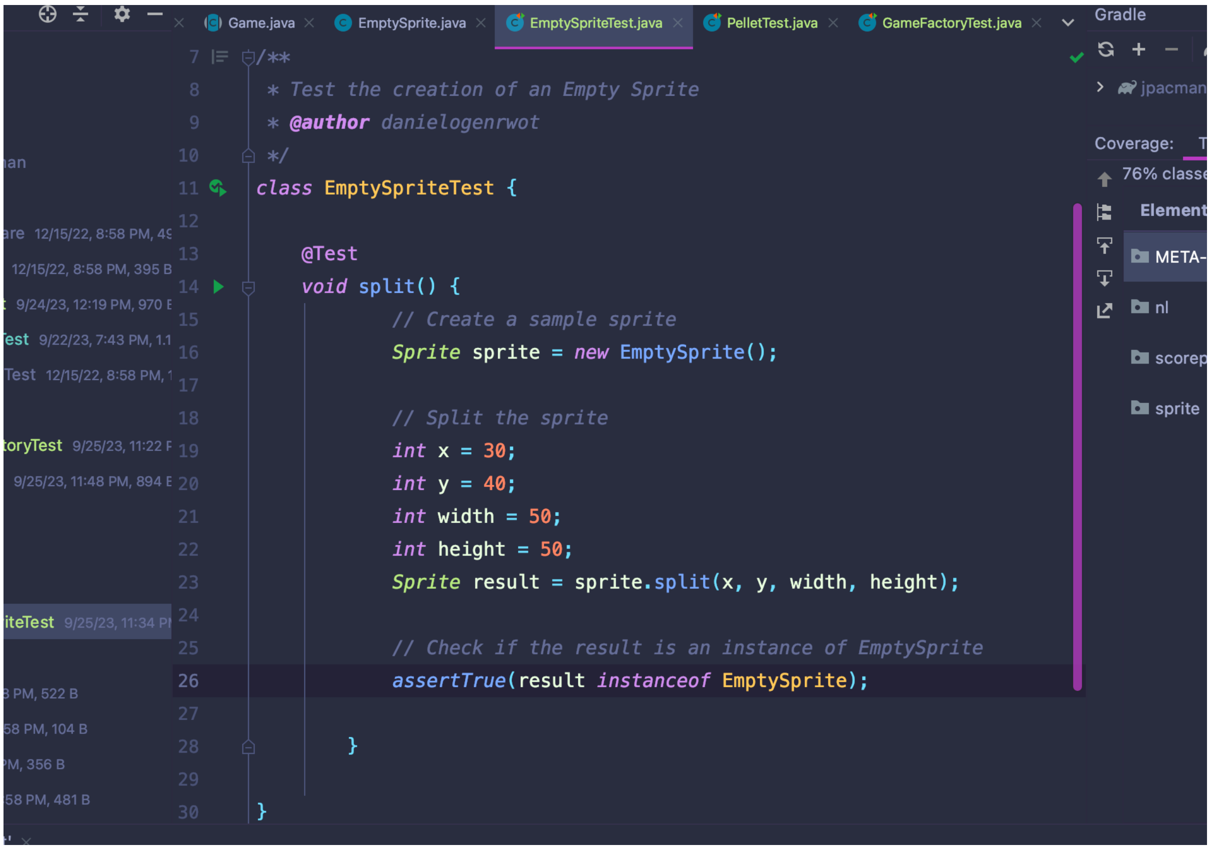
Coverage: Tests in 'jpacman.test'

76% classes, 58% lines covered in 'all classes i...

Element	Clas...	Meth...	Line...
META-INF	100% ...	100% ...	100% ...
nl	76% (...	64% (...	58% (...
scoreplugins	100% ...	100% ...	100% ...
sprite	100% ...	100% ...	100% ...

Figure 3: Screenshot: Testing getPlayerFactory method in GameFactory class

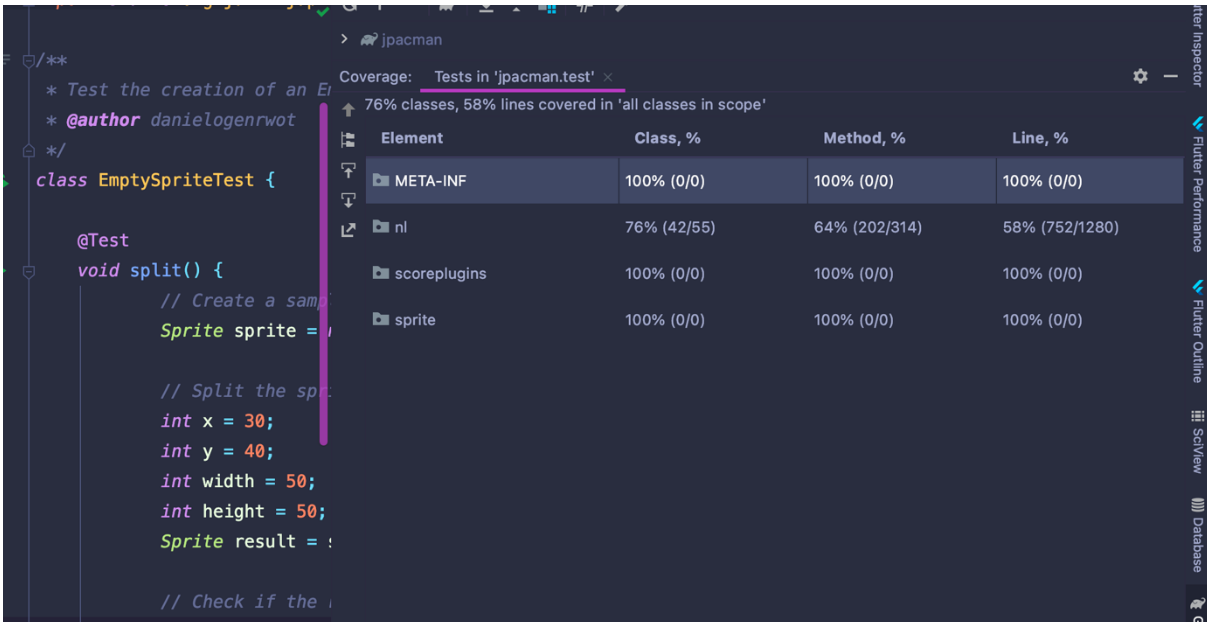
To test the creation of EmptySprite object, we use *instanceOf* method to determined that the object created using the the sprite.split method of an instance of EmptySprite.



```java
/**
 * Test the creation of an Empty Sprite
 * @author danielogenrwot
 */
class EmptySpriteTest {

    @Test
    void split() {
        // Create a sample sprite
        Sprite sprite = new EmptySprite();

        // Split the sprite
        int x = 30;
        int y = 40;
        int width = 50;
        int height = 50;
        Sprite result = sprite.split(x, y, width, height);

        // Check if the result is an instance of EmptySprite
        assertTrue(result instanceof EmptySprite);

    }

}
```

Figure 4: Screenshot: Testing creation of EmptySprite object using the split method

This screenshot shows the overall increase in test coverage as the result of writing the above test cases.



Coverage: Tests in 'jpacman.test'

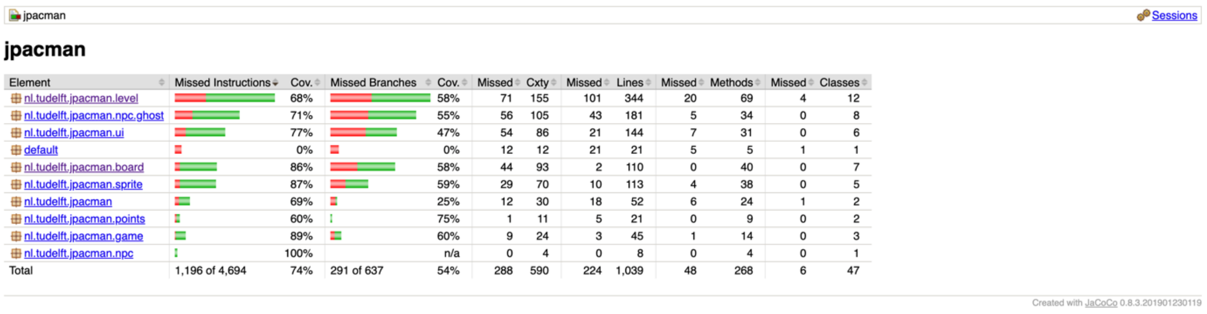
76% classes, 58% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
META-INF	100% (0/0)	100% (0/0)	100% (0/0)
nl	76% (42/55)	64% (202/314)	58% (752/1280)
scoreplugins	100% (0/0)	100% (0/0)	100% (0/0)
sprite	100% (0/0)	100% (0/0)	100% (0/0)

Figure 5: Overall coverage increased to 76%

Overall, I was able to write 13 more test cases and ultimately increase the coverage from 16% to 76%. As shown in Figure 5 above.

Task 3: JCoco Report on JPacman



jpacman

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
nl.tudelft.jpacman.level		68%		58%	71	155	101	344	20	69	4	12
nl.tudelft.jpacman.npc.ghost		71%		55%	56	105	43	181	5	34	0	8
nl.tudelft.jpacman.ui		77%		47%	54	86	21	144	7	31	0	6
default		0%		0%	12	12	21	21	5	5	1	1
nl.tudelft.jpacman.board		86%		58%	44	93	2	110	0	40	0	7
nl.tudelft.jpacman.sprite		87%		59%	29	70	10	113	4	38	0	5
nl.tudelft.jpacman		69%		25%	12	30	18	52	6	24	1	2
nl.tudelft.jpacman.points		60%		75%	1	11	5	21	0	9	0	2
nl.tudelft.jpacman.game		89%		60%	9	24	3	45	1	14	0	3
nl.tudelft.jpacman.npc		100%		n/a	0	4	0	8	0	4	0	1
Total	1,196 of 4,694	74%	291 of 637	54%	288	590	224	1,039	48	268	6	47

Created with JaCoCo 0.8.3.201901230119

Figure 5: Showing JCoco report visualization

Questions:

1. Are the coverage results from JaCoCo similar to the ones you got from IntelliJ in the last task? Why so or why not?

   - Both JaCoCo and IntelliJ coverage gives similar (but NOT the same) coverage results with variations in details. For example, IntelliJ record total coverage of 76% but JaCoCo is showing 74% only. In my opinion, JaCoCo provides more details than IntelliJ

2. Did you find helpful the source code visualization from JaCoCo on uncovered branches?
   - Yes, JaCoCo is very helpful in providing information at a glance
3. Which visualization did you prefer and why? IntelliJ's coverage window or JaCoCo's report?
   - I would prefer IntelliJ when am coding but if I want to dig deeper into the branches and lines missed by the test cases, I would prefer JaCoCo