Tracking coverage metrics ...

Jacoco

Objectives

- Running code coverage tools
- Applying white-box coverage techniques
- Refactoring in systematic steps
- Continue practice of Git/GitHub

Using seg3103_playground

- Create **/lab03** directory
  - Extract **computation.zip** and **date.zip**
  - Make sure you can
    - compile code
    - Run tests
    - Run jacoco

JaCoCo Java Code Coverage Library

JaCoCo is a free code coverage library for Java, which has been created by the EclEmma team based on the lessons learned from using and integration existing libraries for many years.

JaCoCo > org.jacoco.report

org.jacoco.report

Element	Instruction Coverage		Missed Classes	Missed Methods	Missed Blocks	Missed Lines
org.jacoco.report.html		63%	10 / 20	54 / 128	84 / 214	117 / 385
org.jacoco.report.csv		20%	8 / 9	35 / 43	52 / 68	100 / 126
org.jacoco.report		75%	3 / 7	6 / 25	12 / 69	26 / 98
org.jacoco.report.xml		90%	2 / 10	9 / 42	13 / 88	17 / 146
org.jacoco.report.html.resources		87%	1 / 3	1 / 7	9 / 40	2 / 35
Total		64%	24 / 49	105 / 245	170 / 479	262 / 790

Code Coverage Report for JaCoCo 0.1.0.20091027174426 — Created with JaCoCo 0.1.0.20091027174426

http://www.eclemma.org/jacoco/

In this lab, you will experiment with Java Code Coverage Library (JoCoCo)

- Instruction coverage – amount of bytecode executed.
- Branches coverage – amount of branches executed. JoCoCo indicates a branch as partially covered covered when only parts of a branches controlled by a decision have been executed.
- Lines coverage – amount of lines with at least one instruction executed.
- Methods coverage - amount of methods with at least one instruction executed.
- Classes coverage - amount of classes with at least one instruction executed.

Run Coverage Metrics on **Computation**

```
13:07 /tmp/computation $ ./bin/jacoco
Note: src/Computation.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

Thanks for using JUnit! Support its development at https://junit.org/sponsoring

.
├─ JUnit Jupiter ✔
│  └─ ComputationTest ✔
│     ├─ divide() ✔
│     ├─ add() ✔
│     ├─ substract() ✔
│     ├─ multiply() ✔
│     ├─ catchesException() ✔
│     └─ justALoop() ✔
└─ JUnit Vintage ✔

Test run finished after 48 ms
[         3 containers found      ]
[         0 containers skipped    ]
[         3 containers started    ]
[         0 containers aborted    ]
[         3 containers successful ]
[         0 containers failed     ]
[         6 tests found           ]
[         0 tests skipped         ]
[         6 tests started         ]
[         0 tests aborted         ]
[         6 tests successful      ]
[         0 tests failed          ]

[INFO] Loading execution data file /private/tmp/computation/jacoco.exec.
[INFO] Analyzing 2 classes.
```

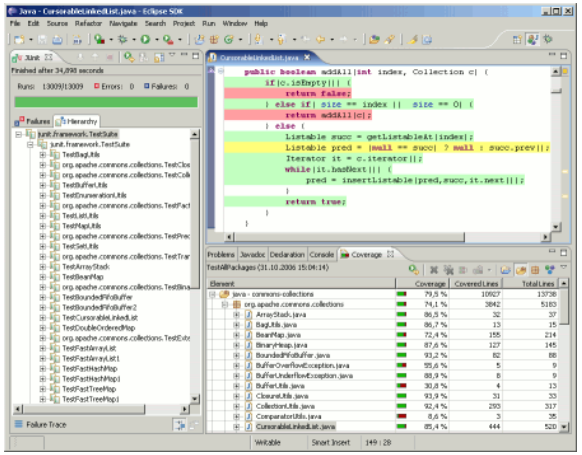
```
# Run the agent
java -javaagent:lib/jacocoagent.jar -jar lib/junit-platform-libconsole-standalone-1.7.1.jar --class-path dist --scan-class-path

# Generate a report
java -jar/jacococli.jar report jacoco.exec --classfiles dist --sourcefiles src --html report
```


JaCoCo includes a command line tool for generating reports.

Computation.java

```
1.  public class Computation {
2.
3.      public int add(int a, int b) {
4.          int result = a + b;
5.          int zero = 0;
6.          int result2 = result;
7.          if (a == Integer.MIN_VALUE) {
8.              new Integer(result);
9.          }
10.         int result3 = result2;
11.         return result + zero;
12.     }
13.
14.     public int multiply(int n, int m) {
15.     int result = 0;
16.     for (int j = 0; j < m; j++) {
17.         result += n;
18.       }
19.     return result;
20.     }
21.
22.     public int substract(int a, int b) {
23.         int result = a - b;
24.         return result;
25.     }
```

Analyze results in the browser (report/index.html)



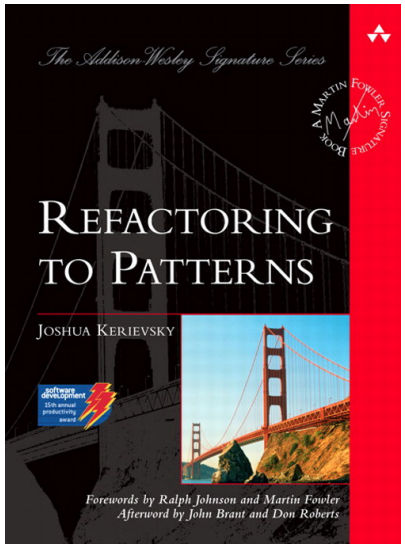
EclEmma

(If using Eclipse)

https://www.eclemma.org

http://www.eclemma.org/installation.html

If you wish to use Eclipse, then you can also install EclEmma



Refactor code one change at a time

Exercise

- Derive a test suite for 100% X coverage for class Date
  - Statement coverage
  - Branch coverage
  - Condition coverage
  - Condition/branch coverage

DATE.JAVA CODE COVERAGE

- Run Jacoco against the existing Date tests and observe the coverage
- Add additional tests to try and achieve 100% coverage
  - Is it possible? How do you know

REFACTOR DATE.JAVA

- Clean up Date.java
  - Small commits
  - No changes to tests should be required
  - Re-run tests after each small change (you don't need to re-run coverage)
  - Run coverage after refactoring
    - Did it improve? Degrade? Why?

Submission

- All work should written under
  - **seg3103_playground/lab03**
- Create **README.md** to summarize your work
  - Summarize your solution here (make it *easy* to grade)
  - Embed screenshots where appropriate, for example ![description](assets/date_coverage_before_refactoring.png)
- Update JUnit code to improve coverage
- Refactor Java code to improve readability
- Share your repository with the teacher and TA (s)
  - Submissions to BrightSpace should clearly reference your GitHub repository