

# Métriques de couverture ...

## Jacoco

# OBJECTIFS

- Exécution des outils de couverture de code
- Application de techniques de couverture en boîte blanche
- Refactoring petits étapes par étapes
- Continuer la pratique de Git / GitHub

# AVEC SEG3103\_PLAYGROUND






- Créer le répertoire **/lab03**
  - Unzip **computation.zip** et **date.zip**
  - Assurez-vous que vous pouvez
    - Compiler le code
    - Exécuter des tests
    - Exécuter Jacoco
  - Exécuter l'app user-registration-app
- Validez votre environnement et commit le code AVANT de appliquer vos modifications

# 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 / 68	26 / 98
org.jacoco.report.xml	 90%	2 / 10	9 / 42	13 / 68	17 / 146
org.jacoco.report.html.resources	 87%	1 / 3	1 / 7	9 / 40	2 / 35
<b>Total</b>	<b>64%</b>	<b>24 / 49</b>	<b>105 / 245</b>	<b>170 / 479</b>	<b>262 / 790</b>

Code Coverage Report for JaCoCo 0.1.0 20091027174426 Created with JaCoCo 0.1.0 20091027174426

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

# Exécuter des mesures de couverture sur **Calculation**

```
[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() ✓
      ├── subtract() ✓
      ├── 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
```

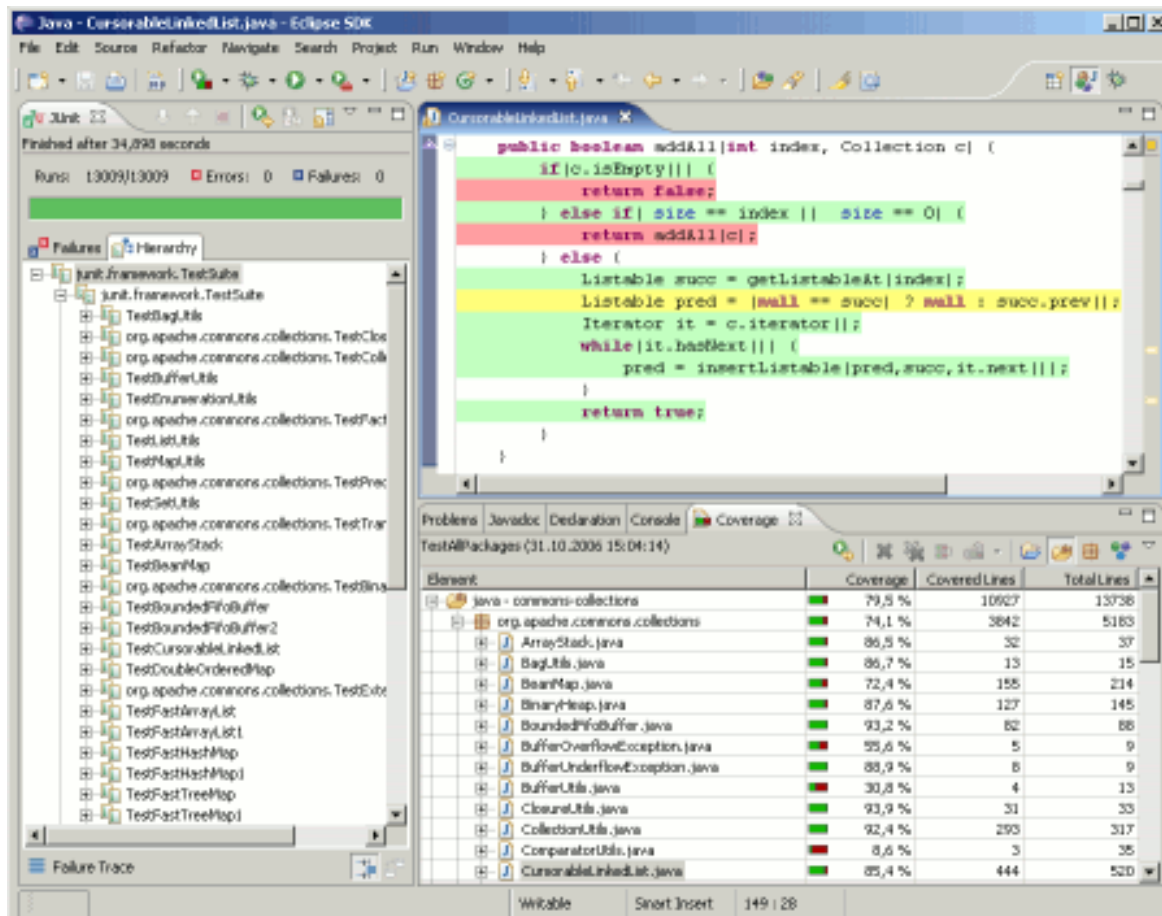
# Generate a report

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

## 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 subtract(int a, int b) {  
23.        int result = a - b;  
24.        return result;  
25.    }  
26. }
```

Analyser les  
résultats dans le  
navigateur (report/  
index.html)

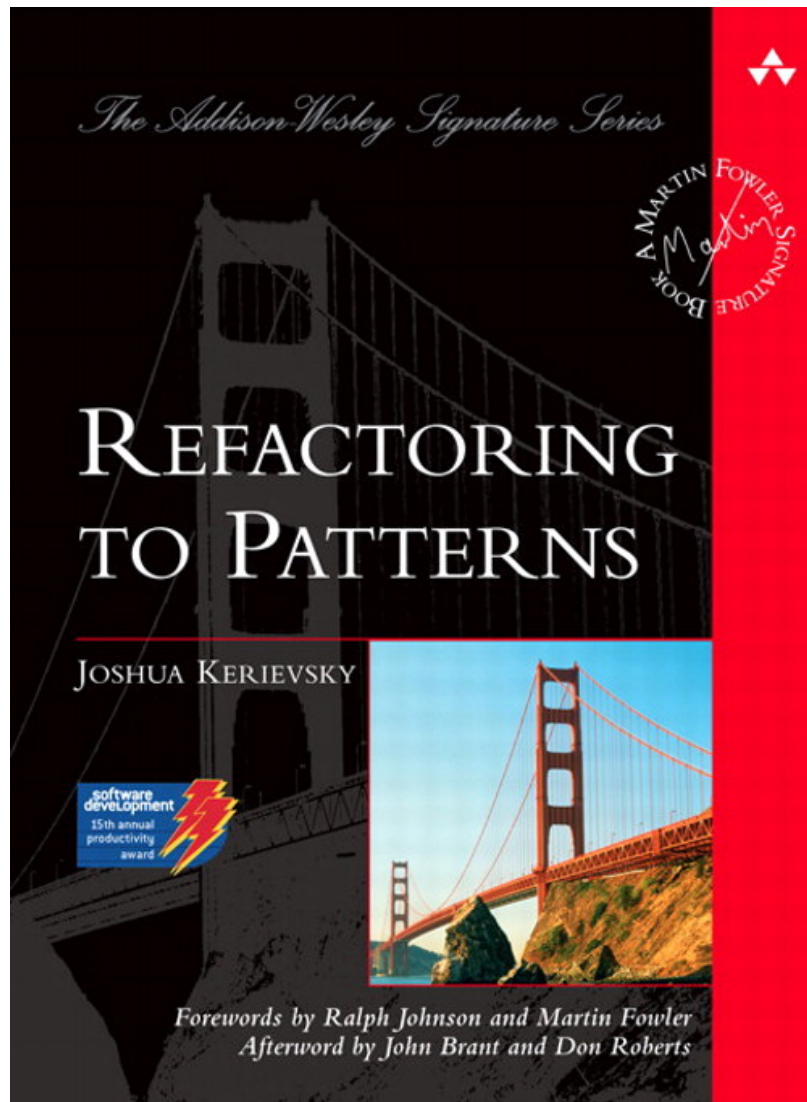


# EclEmma

(Si vous utilisez Eclipse)

<https://www.eclemma.org>

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



# Refactoriser le code un changement à la fois



# EXERCICE

- Dériver une suite de tests pour une couverture X à 100% pour la classe Date
  - Des énoncés
  - Des branches
  - Des conditions
  - Des conditions/branches

# DATE.JAVA COUVERTURE DE CODE

- Exécutez Jacoco sur les tests existants de Date et observez la couverture
- Ajoutez un test supplémentaire pour essayer d'atteindre une couverture de 100%
  - C'est possible? Comment savez-vous

# REFACTORISER DATE.JAVA

- Nettoyer Date.java
  - Petits commits
  - Modification des tests ne devrait pas être nécessaire
  - Ré-exécutez les tests après chaque petit changement (vous n'avez pas besoin de ré-exécuter la couverture)
- Exécuter la couverture après la refactorisation
  - La couverture s'est-il amélioré? Dégrader? Pourquoi?

# SOUSSION

- Tous les travaux doivent être dans
  - **seg3503\_playground/lab02**
- Créez **README.md** pour résumer votre travail
  - Résumez votre solution (faciliter le marquage)
  - Intégrez des screenshots, par exemple! [Description] (assets/date\_coverage\_before\_refactoring.png)
- Editez le code JUnit pour améliorer la couverture
- Partagez votre répo avec l'enseignant et les AEs
  - Les soumissions à BrightSpace doivent clairement référencer votre référentiel GitHub