GUIDE TO JACOCO

Author: *VAMOS Team*
Date: January 19, 2024

I. Introduction

What is Jacoco?

JaCoCo, short for Java Code Coverage, is an open-source toolkit for measuring and reporting code coverage in Java applications.

In this KISS doc, we will install Jacoco into JakartaEE project, using Maven plugin. We will also go into details and understand the information displayed in Jacoco.

II. Installation

If you haven't installed Jacoco into your project yet, please follow from step 01.

However, in the Agile Course project, I've already installed it. So you can go straight to step 2.2 and run Jacoco.

Step 1: Go to this site to get the most recent update of Jacoco plugin:

https://www.eclemma.org/jacoco/trunk/doc/examples/build/pom-it.xml

On this site, you will see a <plugin> like this:

Step 2.1: Copy the Jacoco Plugin and add it into the Pom.xml file, inside the <plugins> tag:

You can change the version of the plugin based on the its version on Maven Central.

For example, I want to use **Jacoco 0.8.11** as mentioned on [Maven Central Repository](). Therefore, I changed the version of the plugin in my pom.xml to 0.8.11.

Home » org.jacoco » jacoco-maven-plugin » 0.8.11

JaCoCo :: Maven Plugin » 0.8.11

The JaCoCo Maven Plugin provides the JaCoCo runtime agent to your tests and allows basic report creation.

You may read the comments inside the plugin for more details on each tag.

```
<!-- JaCoCo Maven Plugin Configuration -->
<plugin>
   <groupId>org.jacoco</groupId>
   <artifactId>jacoco-maven-plugin</artifactId>
   <version>0.8.11</version>
   <!-- Execution to prepare the JaCoCo agent -->
   <executions>
       <execution>
           <id>default-prepare-agent</id>
           <goals>
               <goal>prepare-agent</goal>
           </goals>
       </execution>
       <!-- Execution to generate the JaCoCo coverage report
after running tests -->
       <execution>
           <id>default-report</id>
           <phase>test</phase>
           <goals>
               <goal>report</goal>
           </goals>
       </execution>
       <!-- Execution to generate JaCoCo coverage report for
integration tests -->
       <execution>
           <id>default-report-integration</id>
           <goals>
               <goal>report-integration</goal>
           </goals>
       </execution>
       <!-- Execution to check coverage against specified rules
-->
       <execution>
```

```
            <id>default-check</id>
            <goals>
                <goal>check</goal>
            </goals>
            <!-- Configuration for coverage rules -->
            <configuration>
                <rules>
                    <rule>
                        <!-- Specify the element (in this case,
PACKAGE) to apply the coverage rule -->
                        <element>PACKAGE</element>
                        <!-- Define coverage limits for the
specified element -->
                        <limits>
                            <limit>
                                <!-- Specify the counter type
(LINE) -->
                                <counter>LINE</counter>
                                <!-- Specify the coverage metric
(COVEREDRATIO) -->
                                <value>COVEREDRATIO</value>
                                <!-- Set the minimum acceptable
coverage ratio to 80% -->
                                <minimum>0.80</minimum>
                            </limit>
                        </limits>
                    </rule>
                </rules>
            </configuration>
        </execution>
    </executions>
</plugin>
```

One important thing in the plugin is the <limit> tag. It specifies the minimum acceptable coverage ratio.

In the Agile Course project, the minimum coverage is set to a minimum of 0.80 = 80% according to our DOD.

Step 2.2: Exclude the classes and packages you don't want to display in the test coverage

A/ Exclude packages:

Add the packages you want to exclude in the **<configuration><excludes></excludes></configuration>**

In Agile Course Project, I want to exclude Base package and all Entity packages.

```
<groupId>org.jacoco</groupId>
<artifactId>jacoco-maven-plugin</artifactId>
<configuration>
   <includes>
       <include>com/axonactive/agilecourse/**</include>
   </includes>
   <excludes>
       <exclude>com/axonactive/agilecourse/base/**</exclude>
       <exclude>**/entity/**</exclude>
   </excludes>
</configuration>
```

When we run Jacoco later, we won't see these packages again.

B/ Excludes classes and methods:

Starting from **JaCoCo 0.8.2**, we can exclude classes and methods by annotating them with a custom annotation with the following properties:

- The name of the annotation should include *Generated*.
- The retention policy of annotation should be *runtime* or *class.*

Create **@Generate** annotation:

```
/* Classes or Methods that are annotated with @Generated
will be excluded from Jacoco Test Coverage
 */
2 usages
@Documented
@Retention(RUNTIME)
@Target({TYPE, METHOD, CONSTRUCTOR})
public @interface Generated {
}
```

Then we can apply this annotation on both classes and methods.

```
@Generated
public class Course extends BaseEntity {

    private String title;
```

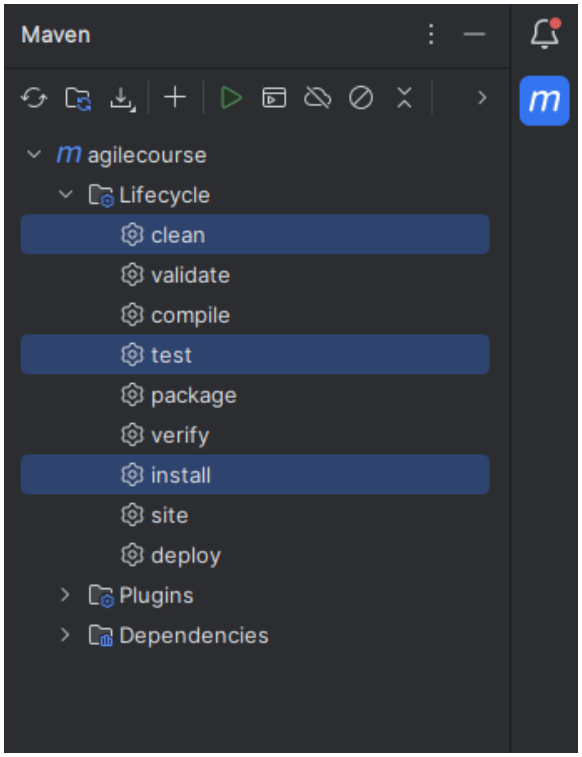
```
AAVN\ntkhoa +1
@DELETE
@Path("/{courseId}")
@Generated
public Response removeCourse(@PathParam("courseId")

AAVN\ntkhoa
```

Step 3: Write some tests

Prepare some test cases to use Jacoco.

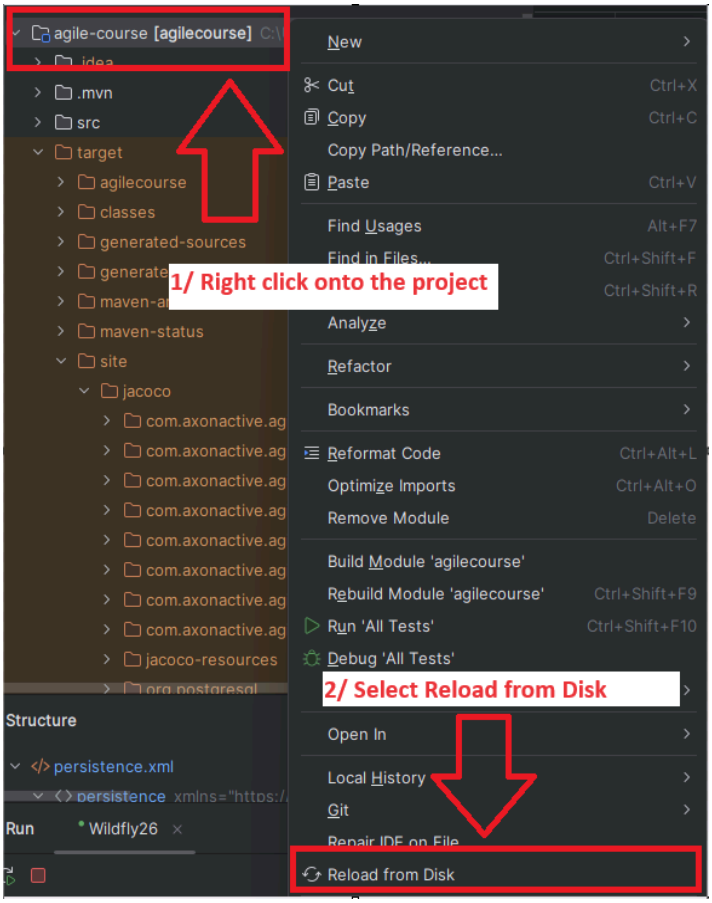
Step 4: Run MVN Clean Test Install

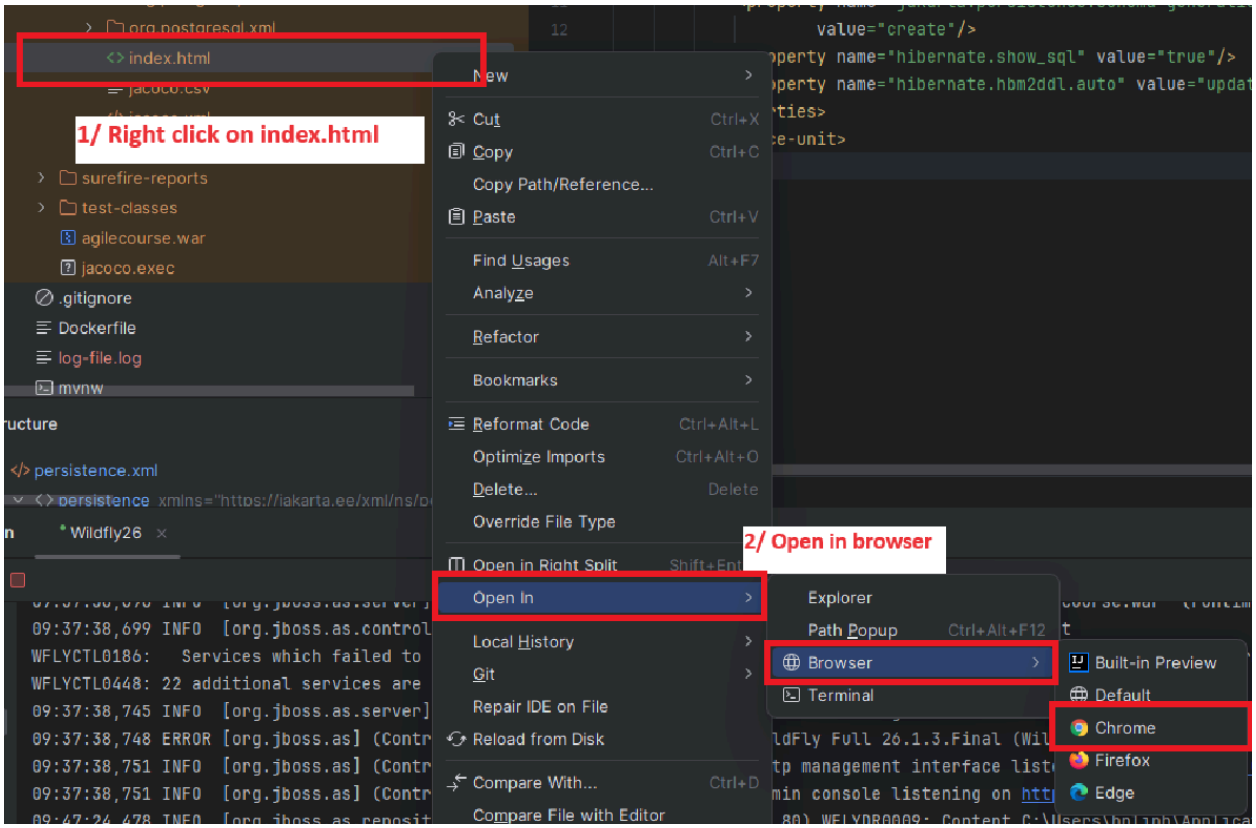


Step 5: Reload the disk

Step 6: You will see [target] folder below [src] folder.

Open **target => site => jacoco**, then open the index.html file on the browser.





Voilà! Now you can see your test coverage.

agilecourse

agilecourse

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.axonactive.agilecourse.course.rest		0%		n/a	3	3	4	4	3	3	1	1
com.axonactive.agilecourse.course.dao		0%		n/a	1	1	2	2	1	1	1	1
com.axonactive.agilecourse.course.service		100%		n/a	0	3	0	3	0	3	0	1
Total	24 of 37	35%	0 of 0	n/a	4	7	6	9	4	7	2	3

III. Understand the Coverage Counters

1/ Instruction:

Instruction indicates certain parts of your code were not executed during testing.

agilecourse

Element	Missed Instructions	Cov.
com.axonactive.agilecourse.course.rest		0%
com.axonactive.agilecourse.test		100%
com.axonactive.agilecourse.course.service		0%
com.axonactive.agilecourse.course.dao		0%
Total	137 of 163	15%

You can go into any **packages => class => methods** to see your code. If your code is highlighted red like the example below, it means that the code was not executed to be tested.

```
    public Optional<Course> findCourseById(Long id) {
        var c = courseDAO.findById(id);
        return c;
    }
```

2/ Branches:

This number counts the number of branches that were not executed during the test run.

A branch, in this context, represents a decision point in the code, such as an `if`, `else`, or `switch` statement.

agilecourse

agilecourse

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Mis
com.axonactive.agilecourse.course.rest		0%		0%	
com.axonactive.agilecourse.test		100%		100%	
com.axonactive.agilecourse.course.service		0%		n/a	
com.axonactive.agilecourse.course.dao		0%		n/a	
Total	137 of 163	15%	4 of 8	50%	

```
60.     @GET
61.     @Path("/{courseId}")
62.     @Produces({MediaType.APPLICATION_JSON})
63.     public Response findCourseById(@PathParam("courseId") Long courseId) {
64.         var c = courseService.findCourseById(courseId);
65.         if(c.isPresent()){
66.             return Response.ok().entity(c).build();
67.         }
68.
69.                                 tus(400).entity(ResponseMessage.builder().success(false).
70.                         statusCode(Response.Status.BAD_REQUEST.getStatusCode())
71.                         .message("Invalid course id").build()).build();
72.         }
73.     }
74. }
```

Any Missed Branch will be marked with this red dot

3/ Other figures:

Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
8	8	21	21	6	6	1	1
0	4	0	6	0	2	0	1
6	6	8	8	6	6	1	1
1	1	2	2	1	1	1	1
15	19	31	37	13	15	3	4

Created with JaCoCo 0.8.11.202310140853

Cyclomatic Complexity (*Cxty* Column):

Cxty indicates the number of test cases required to fully cover a given piece of code, like class or a method.

Lines:

This counter informs us about the number of byte code instructions, not each line of code, since - depending on the source formatting, one line of our source code can refer to multiple classes, methods

Methods: indicates a number of non-abstract methods.

Classes: finally, classes indicates whether at least one method from the given class has been executed

For deeper understanding of the counters, please check this site:
https://codersee.com/jacoco-with-spring-boot-gradle-and-kotlin/

Note: While low coverage is not a good sign, high coverage doesn't necessarily mean high-quality tests either.

References:

Download link:
https://www.jacoco.org/jacoco/

Jacoco Plugin source:
https://www.eclemma.org/jacoco/trunk/doc/maven.html
https://www.eclemma.org/jacoco/trunk/doc/report-mojo.html

Tutorial - Add Jacoco Maven Plugin (uploaded Nov 11th 2023) :
https://www.youtube.com/watch?v=YKCsEtqsuGc

Jacoco report exclusions:
https://www.baeldung.com/jacoco-report-exclude