# Implementation plan

This tool is already installed to everyone's machine as it is built-in to IntelliJ. After understanding how to use it (see my demo) you can start using it right away.
It is expected from you to show the test coverage report of the lines you added in the video evidence for every ticket.

# Configuring and using JaCoCo

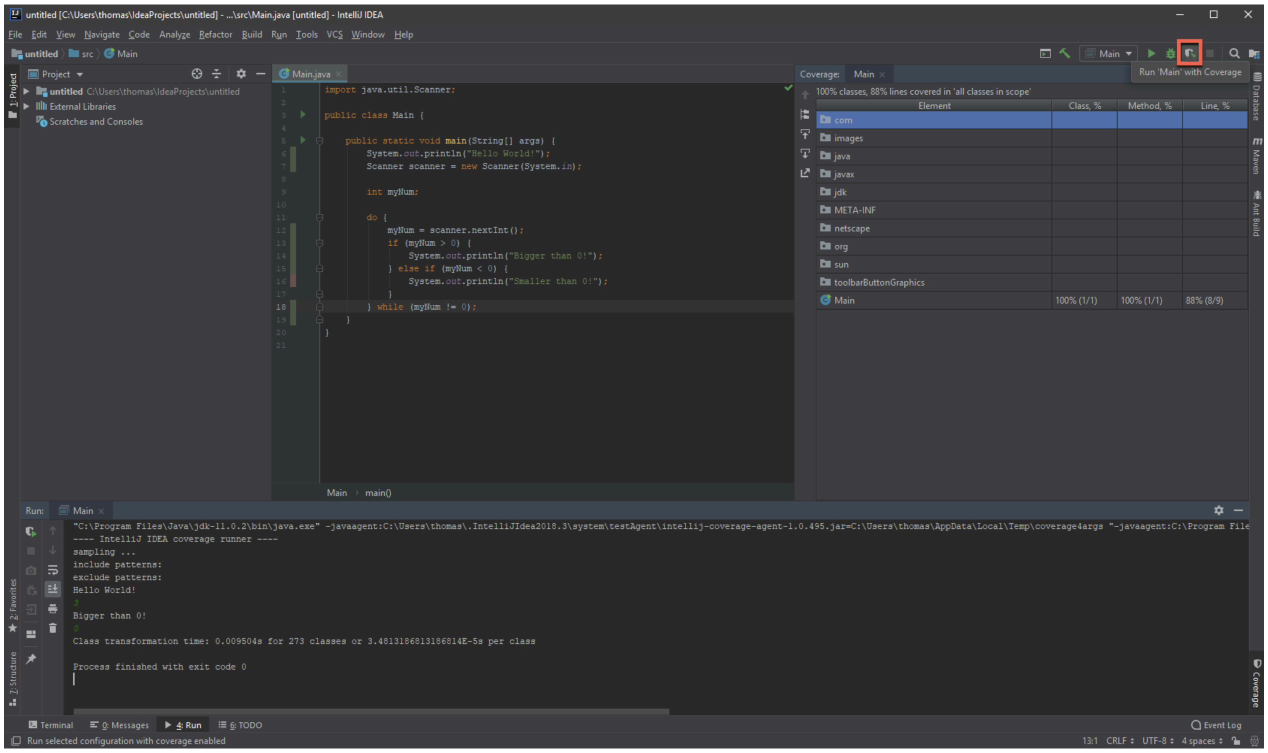
You can use JaCoCo to do either an automated unit/integration test, or just a manual test coverage report. JaCoCo can do either or a combination of all.

## Using IntelliJ

Creating a coverage report with IntelliJ is very simple. There is a separate button for it in the toolbar.

Steps:

1. Click “Run with Coverage”


untitled [C:\Users\thomas\IdeaProjects\untitled] - ...\src\Main.java [untitled] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
untitled › src › Main
Run 'Main' with Coverage
Project
untitled C:\Users\thomas\IdeaProjects\untitled
External Libraries
Scratches and Consoles
Main.java
Main › main()

```java
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        System.out.println("Hello World!");
        Scanner scanner = new Scanner(System.in);

        int myNum;

        do {
            myNum = scanner.nextInt();
            if (myNum > 0) {
                System.out.println("Bigger than 0!");
            } else if (myNum < 0) {
                System.out.println("Smaller than 0!");
            }
        } while (myNum != 0);
    }
}
```

Coverage: Main

100% classes, 88% lines covered in 'all classes in scope'

| Element | Class, % | Method, % | Line, % |
|---|---|---|---|
| com | | | |
| images | | | |
| java | | | |
| javax | | | |
| jdk | | | |
| META-INF | | | |
| netscape | | | |
| org | | | |
| sun | | | |
| toolbarButtonGraphics | | | |
| Main | 100% (1/1) | 100% (1/1) | 88% (8/9) |

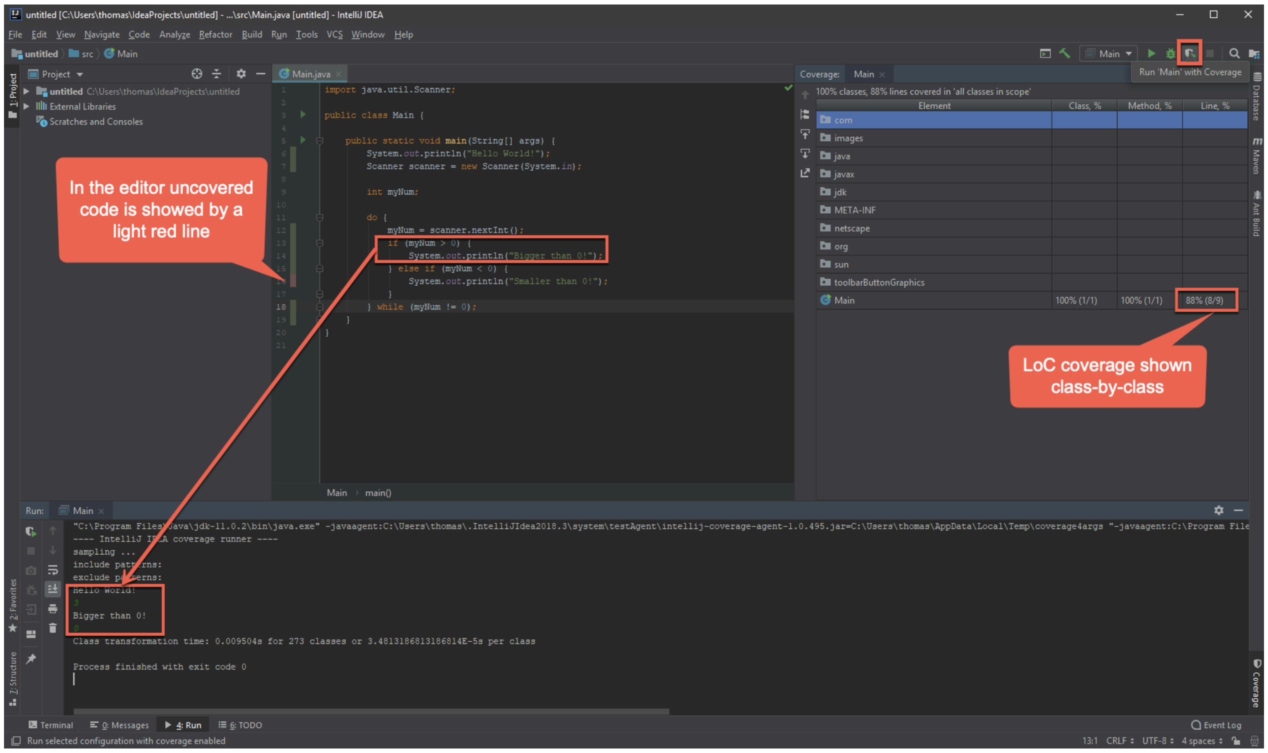
Run: Main

```
"C:\Program Files\Java\jdk-11.0.2\bin\java.exe" -javaagent:C:\Users\thomas\.IntelliJIdea2018.3\system\testAgent\intellij-coverage-agent-1.0.495.jar=C:\Users\thomas\AppData\Local\Temp\coverage4args "-javaagent:C:\Program File
---- IntelliJ IDEA coverage runner ----
sampling ...
include patterns:
exclude patterns:
Hello World!
3
Bigger than 0!
0
Class transformation time: 0.009504s for 273 classes or 3.4813186813186814E-5s per class

Process finished with exit code 0
```

2. This will start “recording” the coverage, so every line you cover during your manual or automated tests will be shown in the IDE.
   In this example I have only 1 branching logic in my code and during my manual test I only tested one of them, which shows in the covered LoC, but also next to

each line of code in the editor.


untitled [C:\Users\thomas\IdeaProjects\untitled] - ...\src\Main.java [untitled] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
untitled src Main
Project
untitled C:\Users\thomas\IdeaProjects\untitled
External Libraries
Scratches and Consoles
Main.java
In the editor uncovered code is showed by a light red line
Run 'Main' with Coverage
LoC coverage shown class-by-class
Main main()
Terminal
0: Messages
4: Run
6: TODO
Event Log
Run selected configuration with coverage enabled
13:1 CRLF UTF-8 4 spaces

```java
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        System.out.println("Hello World!");
        Scanner scanner = new Scanner(System.in);

        int myNum;

        do {
            myNum = scanner.nextInt();
            if (myNum > 0) {
                System.out.println("Bigger than 0!");
            } else if (myNum < 0) {
                System.out.println("Smaller than 0!");
            }
        } while (myNum != 0);
    }
}
```

Coverage: Main

100% classes, 88% lines covered in 'all classes in scope'

| Element | Class, % | Method, % | Line, % |
|---|---|---|---|
| com | | | |
| images | | | |
| java | | | |
| javax | | | |
| jdk | | | |
| META-INF | | | |
| netscape | | | |
| org | | | |
| sun | | | |
| toolbarButtonGraphics | | | |
| Main | 100% (1/1) | 100% (1/1) | 88% (8/9) |

```
"C:\Program Files\Java\jdk-11.0.2\bin\java.exe" -javaagent:C:\Users\thomas\.IntelliJIdea2018.3\system\testAgent\intellij-coverage-agent-1.0.495.jar=C:\Users\thomas\AppData\Local\Temp\coverage4args "-javaagent:C:\Program File
---- IntelliJ IDEA coverage runner ----
sampling ...
include patterns:
exclude patterns:
Hello World!
3
Bigger than 0!
0
Class transformation time: 0.009504s for 273 classes or 3.4813186813186814E-5s per class

Process finished with exit code 0
```

# If you don't use IntelliJ

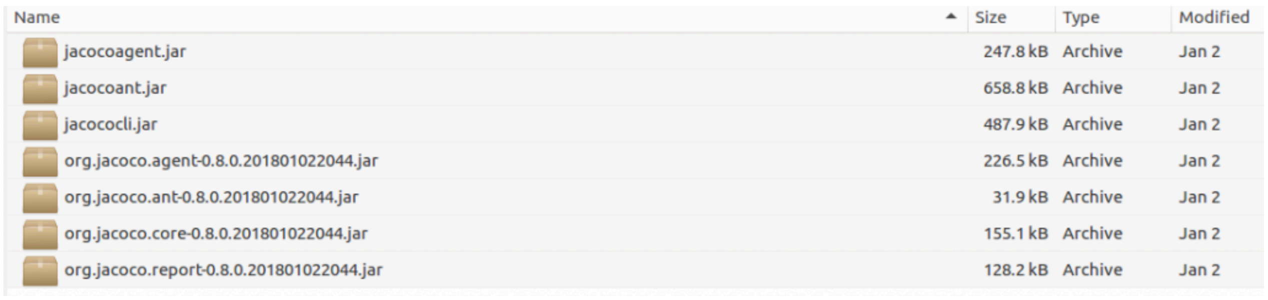
Prerequisites:

1. Tomcat is installed on your machine
2. You know how to deploy your application on Tomcat. if not, learn here: https://stackoverflow.com/questions/5109112/how-to-deploy-a-war-file-in-tomcat-7

Steps:

1. Download the latest Jacoco jars from here: https://oss.sonatype.org/service/local/artifact/maven/redirect?r=snapshots&g=org.jacoco&a=jacoco&e=zip&v=LATEST

2. Extract the downloaded zip folder. You will see different Jacoco jars in /lib folder.

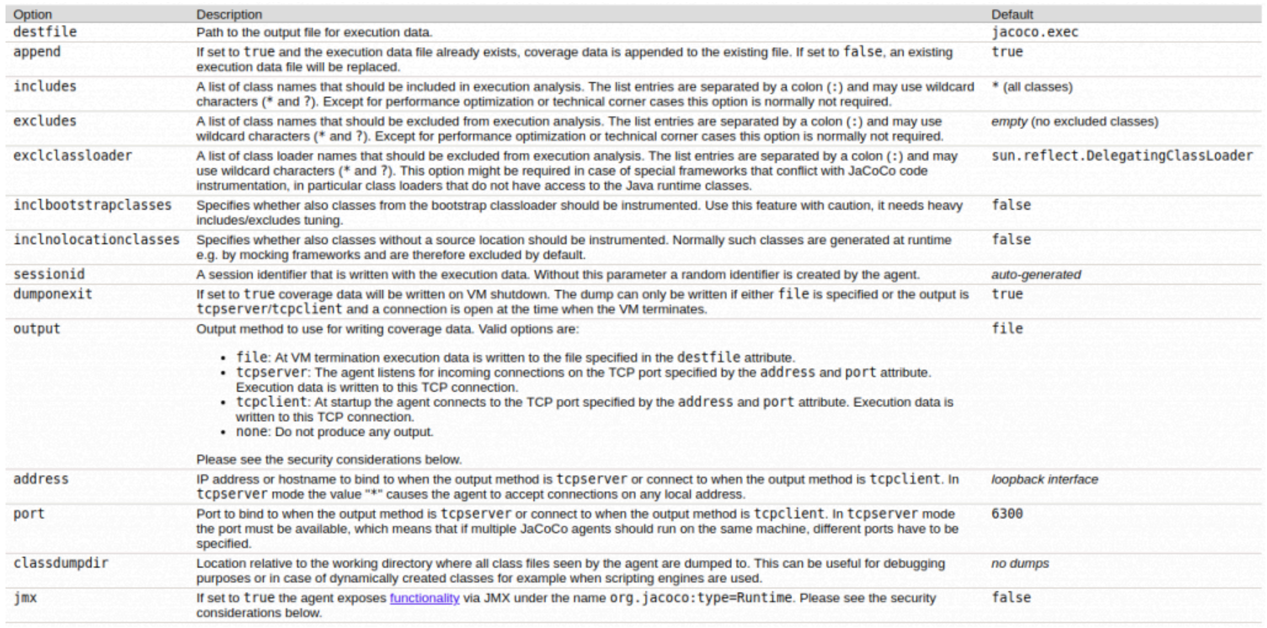


| Name | Size | Type | Modified |
|---|---|---|---|
| jacocoagent.jar | 247.8 kB | Archive | Jan 2 |
| jacocoant.jar | 658.8 kB | Archive | Jan 2 |
| jacococli.jar | 487.9 kB | Archive | Jan 2 |
| org.jacoco.agent-0.8.0.201801022044.jar | 226.5 kB | Archive | Jan 2 |
| org.jacoco.ant-0.8.0.201801022044.jar | 31.9 kB | Archive | Jan 2 |
| org.jacoco.core-0.8.0.201801022044.jar | 155.1 kB | Archive | Jan 2 |
| org.jacoco.report-0.8.0.201801022044.jar | 128.2 kB | Archive | Jan 2 |

3. Go the /bin folder of your tomcat. Create a *setenv.sh* file, add the below contents and save it.

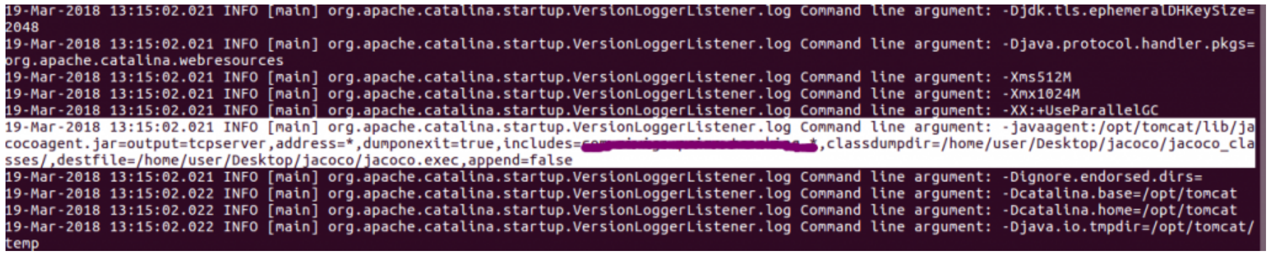
```
CATALINA_OPTS=”$CATALINA_OPTS
-javaagent:/path/to/jacocoagent.jar=output=tcpserver,address
=*,dumponexit=true,includes=classes/to/be/included,classdump
dir=/path/to/save,destfile=/path/to/save/jacoco.exec,append=
false”
```

Below are the different configuration parameters with their explanations



| Option | Description | Default |
|---|---|---|
| `destfile` | Path to the output file for execution data. | `jacoco.exec` |
| `append` | If set to `true` and the execution data file already exists, coverage data is appended to the existing file. If set to `false`, an existing execution data file will be replaced. | `true` |
| `includes` | A list of class names that should be included in execution analysis. The list entries are separated by a colon (`:`) and may use wildcard characters (`*` and `?`). Except for performance optimization or technical corner cases this option is normally not required. | * (all classes) |
| `excludes` | A list of class names that should be excluded from execution analysis. The list entries are separated by a colon (`:`) and may use wildcard characters (`*` and `?`). Except for performance optimization or technical corner cases this option is normally not required. | *empty* (no excluded classes) |
| `exclclassloader` | A list of class loader names that should be excluded from execution analysis. The list entries are separated by a colon (`:`) and may use wildcard characters (`*` and `?`). This option might be required in case of special frameworks that conflict with JaCoCo code instrumentation, in particular class loaders that do not have access to the Java runtime classes. | `sun.reflect.DelegatingClassLoader` |
| `inclbootstrapclasses` | Specifies whether also classes from the bootstrap classloader should be instrumented. Use this feature with caution, it needs heavy includes/excludes tuning. | `false` |
| `inclnolocationclasses` | Specifies whether also classes without a source location should be instrumented. Normally such classes are generated at runtime e.g. by mocking frameworks and are therefore excluded by default. | `false` |
| `sessionid` | A session identifier that is written with the execution data. Without this parameter a random identifier is created by the agent. | *auto-generated* |
| `dumponexit` | If set to `true` coverage data will be written on VM shutdown. The dump can only be written if either `file` is specified or the output is `tcpserver/tcpclient` and a connection is open at the time when the VM terminates. | `true` |
| `output` | Output method to use for writing coverage data. Valid options are: <br>• `file`: At VM termination execution data is written to the file specified in the `destfile` attribute. <br>• `tcpserver`: The agent listens for incoming connections on the TCP port specified by the `address` and `port` attribute. Execution data is written to this TCP connection. <br>• `tcpclient`: At startup the agent connects to the TCP port specified by the `address` and `port` attribute. Execution data is written to this TCP connection. <br>• `none`: Do not produce any output. <br><br>Please see the security considerations below. | `file` |
| `address` | IP address or hostname to bind to when the output method is `tcpserver` or connect to when the output method is `tcpclient`. In `tcpserver` mode the value "`*`" causes the agent to accept connections on any local address. | *loopback interface* |
| `port` | Port to bind to when the output method is `tcpserver` or connect to when the output method is `tcpclient`. In `tcpserver` mode the port must be available, which means that if multiple JaCoCo agents should run on the same machine, different ports have to be specified. | `6300` |
| `classdumpdir` | Location relative to the working directory where all class files seen by the agent are dumped to. This can be useful for debugging purposes or in case of dynamically created classes for example when scripting engines are used. | *no dumps* |
| `jmx` | If set to `true` the agent exposes functionality via JMX under the name `org.jacoco:type=Runtime`. Please see the security considerations below. | `false` |

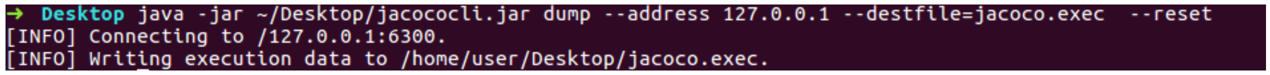
4. Restart your tomcat and do  UI/API testing (manual or integration tests) of your application. You should see command line argument property in the logs while tomcat is deploying the application.



```
19-Mar-2018 13:15:02.021 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Djdk.tls.ephemeralDHKeySize=
2048
19-Mar-2018 13:15:02.021 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Djava.protocol.handler.pkgs=
org.apache.catalina.webresources
19-Mar-2018 13:15:02.021 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Xms512M
19-Mar-2018 13:15:02.021 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Xmx1024M
19-Mar-2018 13:15:02.021 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -XX:+UseParallelGC
19-Mar-2018 13:15:02.021 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -javaagent:/opt/tomcat/lib/ja
cocoagent.jar=output=tcpserver,address=*,dumponexit=true,includes=[redacted],classdumpdir=/home/user/Desktop/jacoco/jacoco_cla
sses/,destfile=/home/user/Desktop/jacoco/jacoco.exec,append=false
19-Mar-2018 13:15:02.021 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Dignore.endorsed.dirs=
19-Mar-2018 13:15:02.022 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Dcatalina.base=/opt/tomcat
19-Mar-2018 13:15:02.022 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Dcatalina.home=/opt/tomcat
19-Mar-2018 13:15:02.022 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Djava.io.tmpdir=/opt/tomcat/
temp
```

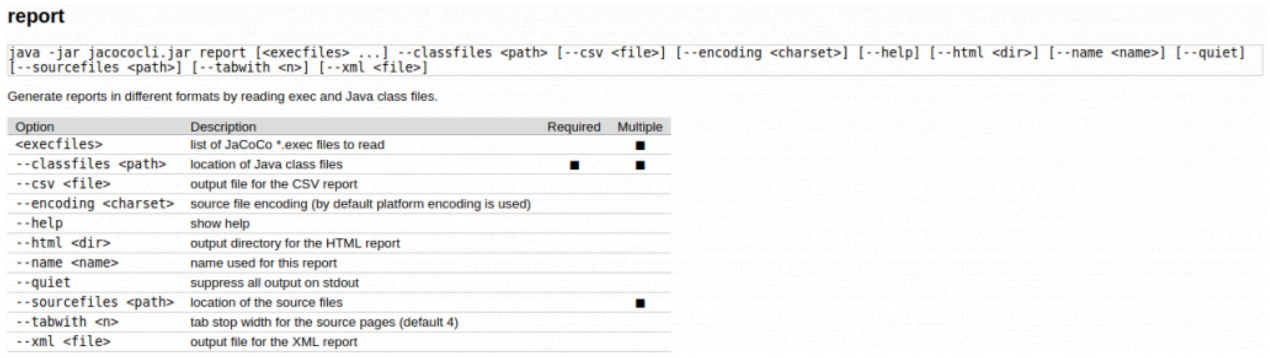
5. Fetch coverage data using *dump* command:

```
java -jar /path/to/jacococli.jar dump –address 127.0.0.1
–destfile=jacoco.exec –reset
```



```
➜  Desktop java -jar ~/Desktop/jacococli.jar dump --address 127.0.0.1 --destfile=jacoco.exec  --reset
[INFO] Connecting to /127.0.0.1:6300.
[INFO] Writing execution data to /home/user/Desktop/jacoco.exec.
```

Below are the different parameters for report command



**report**

```
java -jar jacococli.jar report [<execfiles> ...] --classfiles <path> [--csv <file>] [--encoding <charset>] [--help] [--html <dir>] [--name <name>] [--quiet]
[--sourcefiles <path>] [--tabwith <n>] [--xml <file>]
```

Generate reports in different formats by reading exec and Java class files.

| Option | Description | Required | Multiple |
|---|---|---|---|
| `<execfiles>` | list of JaCoCo *.exec files to read | | ■ |
| `--classfiles <path>` | location of Java class files | ■ | ■ |
| `--csv <file>` | output file for the CSV report | | |
| `--encoding <charset>` | source file encoding (by default platform encoding is used) | | |
| `--help` | show help | | |
| `--html <dir>` | output directory for the HTML report | | |
| `--name <name>` | name used for this report | | |
| `--quiet` | suppress all output on stdout | | |
| `--sourcefiles <path>` | location of the source files | | ■ |
| `--tabwith <n>` | tab stop width for the source pages (default 4) | | |
| `--xml <file>` | output file for the XML report | | |