

Viewing the Code Coverage

We'll cover the following ^

- Code coverage

Code coverage







“What’s a good coverage number?” is a question that can start a bar fight among drunken developers: “If we write a test and then write the minimum code necessary to make the test pass...” Irrespective of what your team believes to be the appropriate coverage percentage, you’ll need a way to view the coverage. Jacoco, which is a coverage tool that works for Java, works nicely with Kotlin as well. Let’s see how we did with code coverage so far in the sample we created in this chapter.

Whether you’ve been using Gradle or Maven to build the examples so far, you’ve been generating the code coverage all along. If you’re using Gradle, take a look at the configuration for Jacoco in the `build.gradle.kts` file and take note of the version number of the library in particular. Likewise, if you’re using Maven, take a look at the `pom.xml` file and look for the configuration of the Jacoco plugin. After running the build, using Gradle or Maven, take a peek at the Jacoco report file. If you’re using Gradle, you’ll find the report `index.html` under `build/reports/jacoco/test/html`, and if you’re using Maven look for it under `target/site/jacoco`.

Here’s the coverage report generated by Jacoco for the Kotlin code we’ve created so far in this chapter.

 [airportapp](#) >  com.agiledeveloper.airportstatus

com.agiledeveloper.airportstatus

Element	Missed Instructions	Cov.	Missed Branches
 AirportStatusKt.getAirportStatus.new Function2()_{...}	<div><div></div></div>	100%	
 Airport.Companion	<div><div></div></div>	100%	
 Airport	<div><div></div></div>	100%	
 AirportStatusKt	<div><div></div></div>	100%	
 AirportStatusKt.getAirportStatus.2.1.new Function2()_{...}	<div><div></div></div>	100%	
 Airport.Companion.sort..inlined.sortedBy.new Comparator()_{...}	<div><div></div></div>	100%	
Total	0 of 166	100%	0 of 0

We've driven the design and implementation of the code using tests; as a last step, we'll write a driver program to run the code in the next lesson.