

## How to start developing Spark applications in Eclipse

By Marko Bonaci

In this article, you will learn to write Spark applications using Eclipse, the most widely used development environment for JVM-based languages.

### ***Application prerequisites***

In the following article, we describe:

- Downloading and extracting Eclipse IDE
- Configuring Eclipse
- Starting Eclipse
- Installing ScalaIDE Eclipse plugin (used to work with Scala in Eclipse)

### ***Installing and configuring Eclipse***

Hmm, what could be the first prerequisite for application development in Eclipse? Well, it's usually the Eclipse itself, so unless you already have the latest *Eclipse IDE for Java developers*, you should download it from here: [www.eclipse.org/downloads](http://www.eclipse.org/downloads). Accept the suggested, default download location, `Downloads` folder inside your *home*.

Once the download completes execute the following script in your terminal. The script extracts *Eclipse* from the archive and moves it to the newly created `bin` folder in your home directory (if you already created `bin` directory in the previous chapter, chapter 2, you can skip the `mkdir` command):

```
$ cd $HOME/Downloads
$ tar -xvf eclipse*.tar.gz
$ mkdir ../bin
$ mv eclipse ../bin/
```

The default Eclipse memory settings are too restrictive, so let's give your IDE some breathing room. It'll work much faster (Eclipse will use all memory at its disposal before falling back to disk for swap space) and you'll avoid potential `OutOfMemoryErrors`:

For source code, sample chapters, the Online Author Forum, and other resources, go to <http://www.manning.com/bonaci/>

```
$ cd ../bin/eclipse
$ gedit eclipse.ini
```

Replace the following JVM parameters<sup>1</sup> and, if needed, adjust the amount of memory that you are willing to give to Eclipse (`Xms` denotes how much memory Eclipse will take initially, as it loads and `Xmx` is the maximum amount of memory Eclipse is allowed to use):

```
-XX:MaxPermSize=384m
-Xms128m
-Xmx2g
```

Let's start Eclipse. In your terminal, still positioned in the `$HOME/bin/eclipse` folder, issue the following command:

```
$ ./eclipse
```

When prompted to choose your workspace folder just click OK, since your home directory, suggested by Eclipse, is a good option.

All good? We have a launch? Great! Close the *Welcome* tab. When it closes, to minimize the noise, also close *Task list* tab.

Next, you need to install two Eclipse plugins: *Scala IDE* and *Scala Maven Plugin*. These are Eclipse plugins that are used to integrate the Scala programming language and the accompanying tools into Eclipse.

---

<sup>1</sup> eclipse.ini in the book's GitHub repo <https://github.com/spark-in-action/first-edition/blob/master/ch03/eclipse.ini>

## Installing Eclipse plugins

Go to *Help*<sup>2</sup> > *Install new software...* and click *Add* in the upper right corner. Once the *Add repository* window appears, enter `scala-ide` in the *Name* field, <http://download.scala-ide.org/sdk/lithium/e44/scala211/stable/site> in the *Location* field and confirm with *OK* (figure 1).

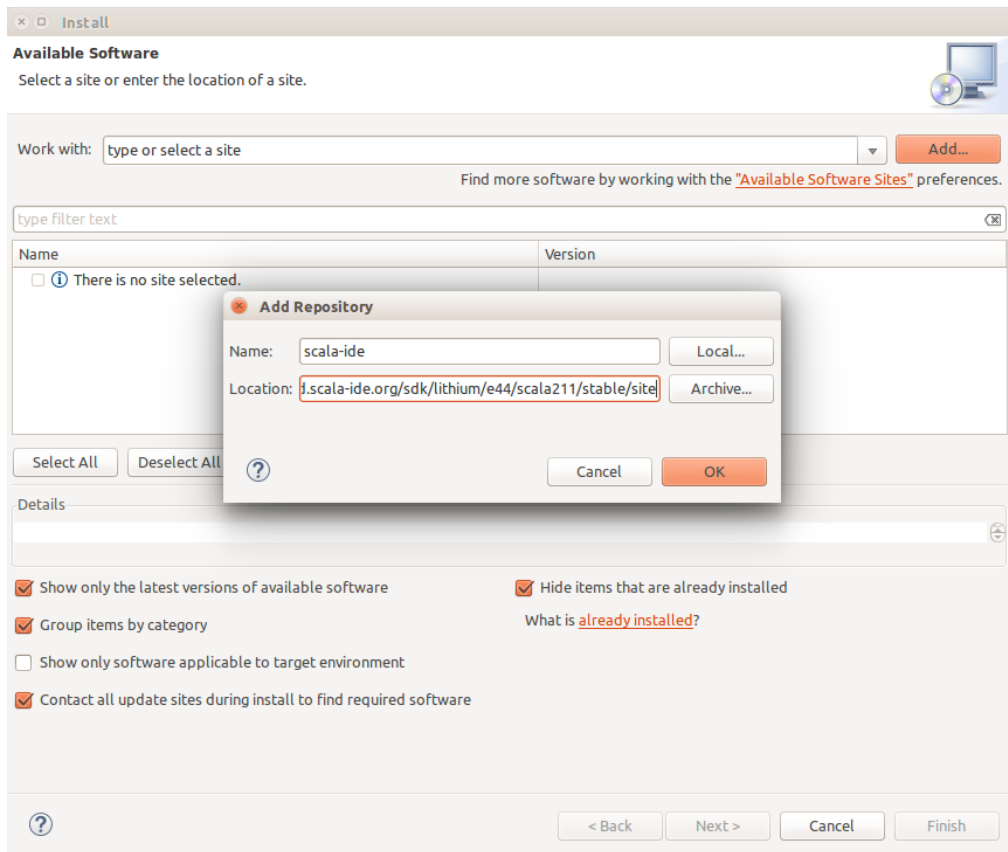


Figure 1 Scala IDE Eclipse plugin installation – Adding remote software repository

<sup>2</sup> Readers new to Ubuntu might not know that toolbar of the currently active window is always located at the top of the screen, and it's revealed once you hover mouse cursor near the top.

Eclipse will look up the URL you entered and display the available software it found there. Select only the topmost entry and all its subentries, as shown in the *figure 2*.

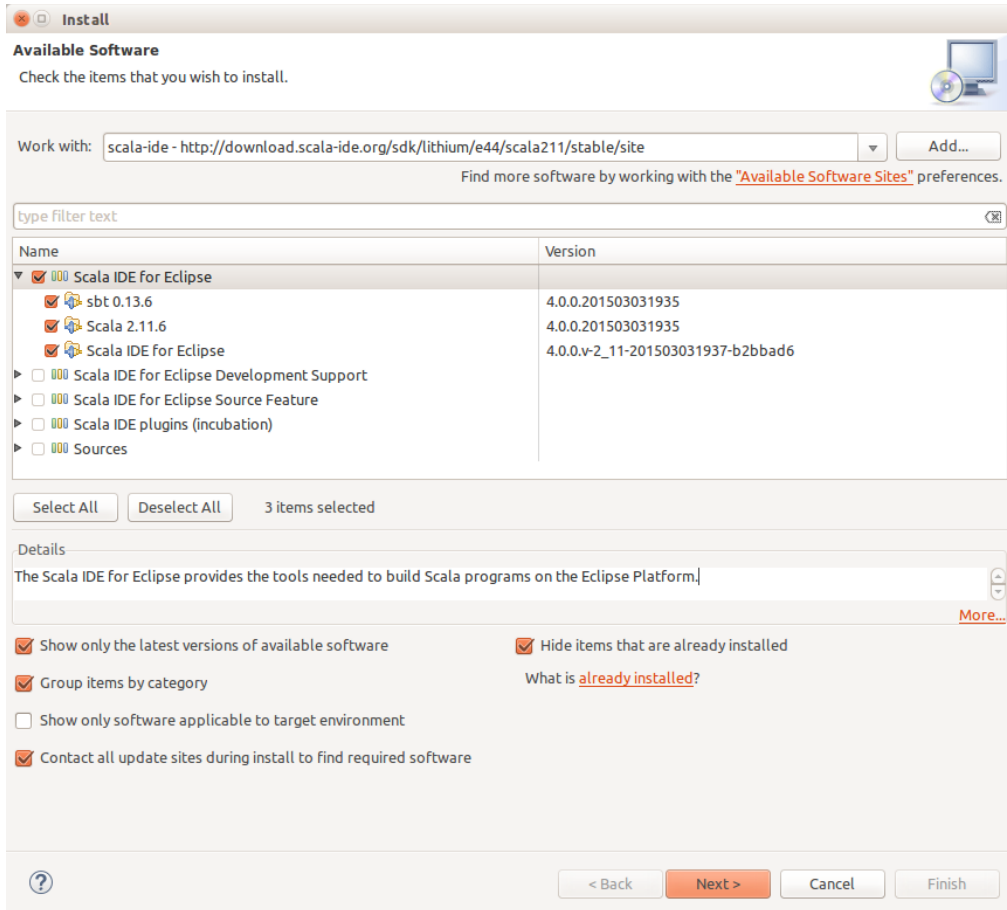


Figure 2 Scala IDE Eclipse plugin installation – Choosing items to install

Confirm the selection on the next screen and accept license on the one after that. Restart Eclipse when prompted.

Good stuff! Next, you will install *Scala Maven integration for Eclipse* plugin<sup>3</sup>, the same way you just installed *Scala IDE*.

So, once again, go to *Help > Install new software...* and add the following repository:

- Name: m2eclipse-scala
- Location: <http://alchim31.free.fr/m2e-scala/update-site>

Make sure that all the items are checked and then follow the installation wizard. Confirm the security warning when prompted and finish the installation by accepting Eclipse restart.

Assuming everything went well<sup>4</sup>, you should now have full Scala and Maven support in Eclipse. Still, there is one more thing you need to do. Maven has this notion of *archetype*, which is basically a project scaffolding mechanism that creates a new Maven project based on a provided project template. This is useful because you usually don't want to start your project with an empty folder. It basically bootstraps a new project with all dependencies and default configuration parameters defined up front, so you don't have to do that manually, every time you start a new project.

To simplify setting up new projects, we have prepared an archetype called `scala-archetype-sparkinaction` which is used to create a starter Spark project where versions and dependencies have already been taken care of.

## **Generating a new Spark project in Eclipse**

Now that you have installed and configured Eclipse, you are ready to start a new Eclipse project that will host your application. We prepared a Maven archetype for bootstrapping new Spark Scala projects which boils down the whole process to "just a few clicks".

To create the new project in Eclipse, on your toolbar menu, select:

---

<sup>3</sup> Official install docs: <http://scala-ide.org/docs/tutorials/m2eclipse/>

<sup>4</sup> If you got stuck with Eclipse setup, you can alternatively download the *ScalaIDE* (<http://scala-ide.org/download/sdk.html>) and unpack the downloaded bundle the same way as you did with Eclipse, but without the need to install any plugins. *ScalaIDE* is basically a slightly visually altered Eclipse distribution with all important Scala plugins already installed (which includes both previously mentioned plugins). We did not recommend *ScalaIDE* as the first option since we wanted to introduce you to Eclipse plugin installation procedure, as you're bound to need it in the future, if you choose Eclipse as your IDE.

File > New... > Project... > Maven > Maven Project

Do not make any changes on the first screen of the new project wizard, simply click *Next*.

On the second screen click *Configure...* (which opens: *Maven > Archetypes* section of the Eclipse Preferences). Click on *Add Remote Catalog...* button and fill in the following values in the dialog that pops up (*figure 3*):

- **Catalog File:** `https://github.com/spark-in-action/scala-archetype-sparkinaction/raw/master/archetype-catalog.xml`
- **Description:** Spark in Action

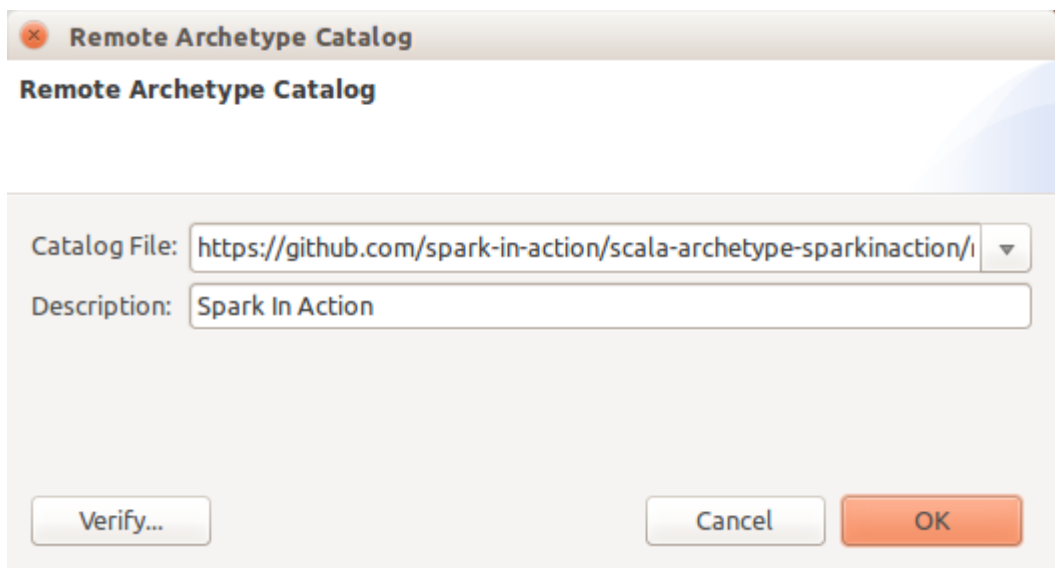


Figure 3 Adding Maven remote archetype catalog to Eclipse preferences

Confirm with *OK* and then close the Preferences window. You should see a progress bar in the lower right corner, which is how Eclipse notifies you that it went to look up the catalog that you just added.

You are now back into *New Maven Project* wizard. Select *Spark in Action* in the *Catalog* drop-down field (*figure 4*).

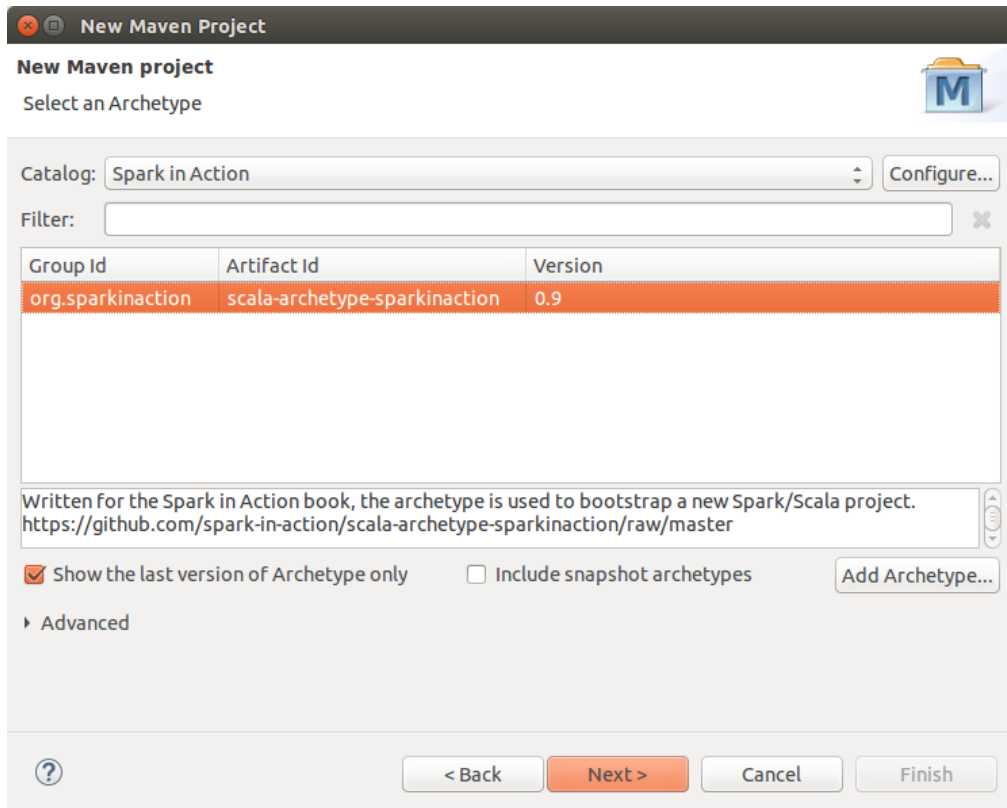


Figure 4 Choosing the Maven archetype that you want to use as the new project's template

The next screen prompts you to enter your project's parameters. Notice (in the *figure 3.5*) that your root package would be comprised of *groupId* and *artifactId*.

### Group ID and Artifact ID

If you are new to Maven, perhaps it might be easier for you to look at *artifactId* as you project name and *groupId* as its full qualified organization name. For example, Spark has *groupId* `org.apache` and *artifactId* `spark`. Play framework has `com.typesafe` as its *groupId* and `play` as its *artifactId*.

You can specify whichever values you like for *groupId* and *artifactId*, but it might be easier for you to follow if you choose the same values as we did (*figure 5*). Confirm by clicking Finish, of course.

**New Maven Project**  
Specify Archetype parameters

Group Id:

Artifact Id:

Version:

Package:

Properties available from archetype:

Name	Value
------	-------

Advanced

Figure 5 Creating a new Maven project – Specifying project parameters

Your project, once generated, should look like the one in the *figure 6*.



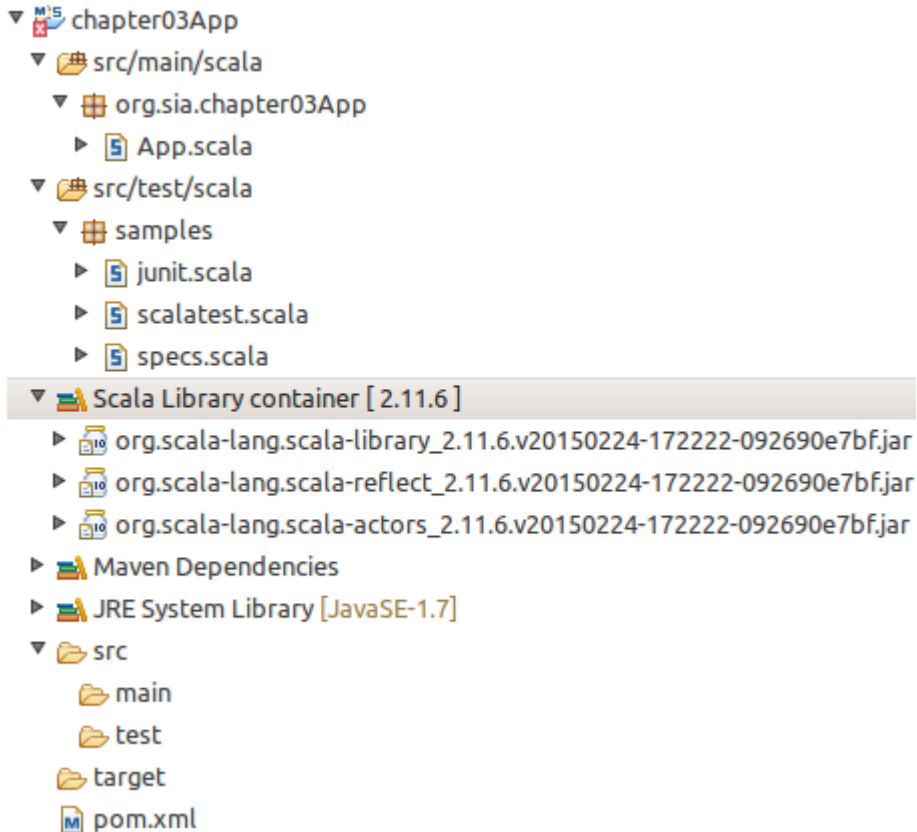



Figure 6 Generated project in Eclipse's Package Explorer window

We don't know what kind of operation you're running there, but we see that you have some errors (that is what the red  mark on the project root signifies) and that is not cool at all.

Let's fix that. To see the error listing, go to *Problems tab*, just below the main editor window (if you cannot find it, you can go to `Window > Show View > Other... > Problems`). The cause of errors is the version incompatibility between your project's libraries (that came with the archetype) and the *Scala Library container* that *ScalaIDE* plugin automatically placed into your project (notice the version [2.11.6] besides the *Scala Library container*, in *figure 6*).

To change the *Scala Library container* version, right click on the project's root and (figure 3.7) select: Scala > Set the Scala Installation > Fixed Scala Installation: 2.10.5(bundled). Note that the *Fixed* version will never change "under you", while the *Latest* version is going to float up, to the latest 2.10.x version with upgrades of ScalaIDE, as it brings new maintenance<sup>5</sup> releases in.

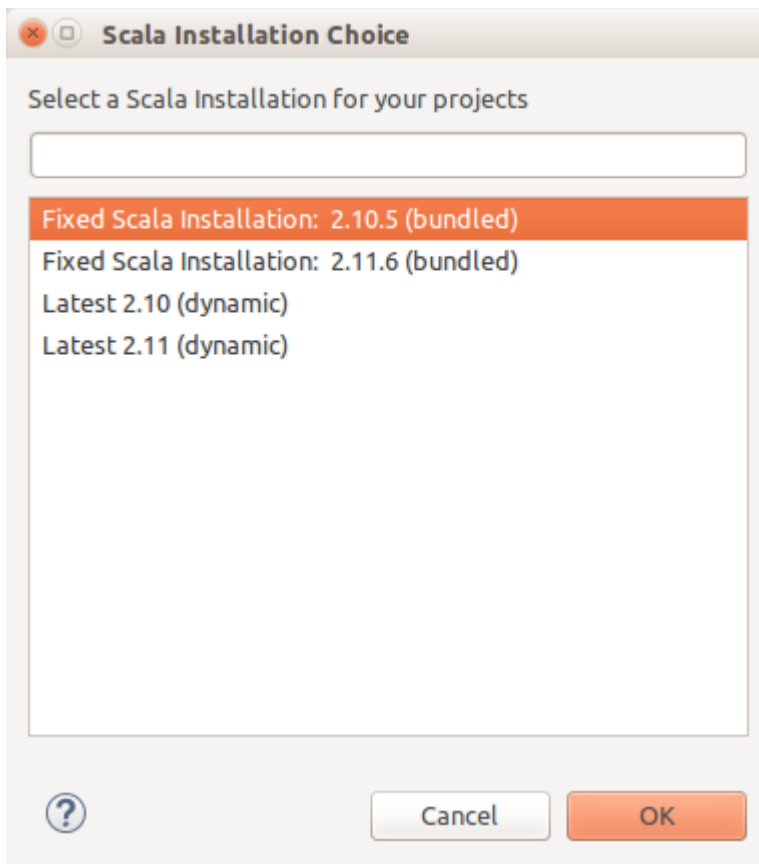



Figure 7 Set the compatible Scala version project-wide

---

<sup>5</sup> E.g. v2.10.5 is comprised of the following release levels [2 – major].[10 – minor].[5 – maintenance].

At this point the red  mark disappeared and your project is free of errors. Nice! Now your Eclipse Package Explorer window should look like figure 8.

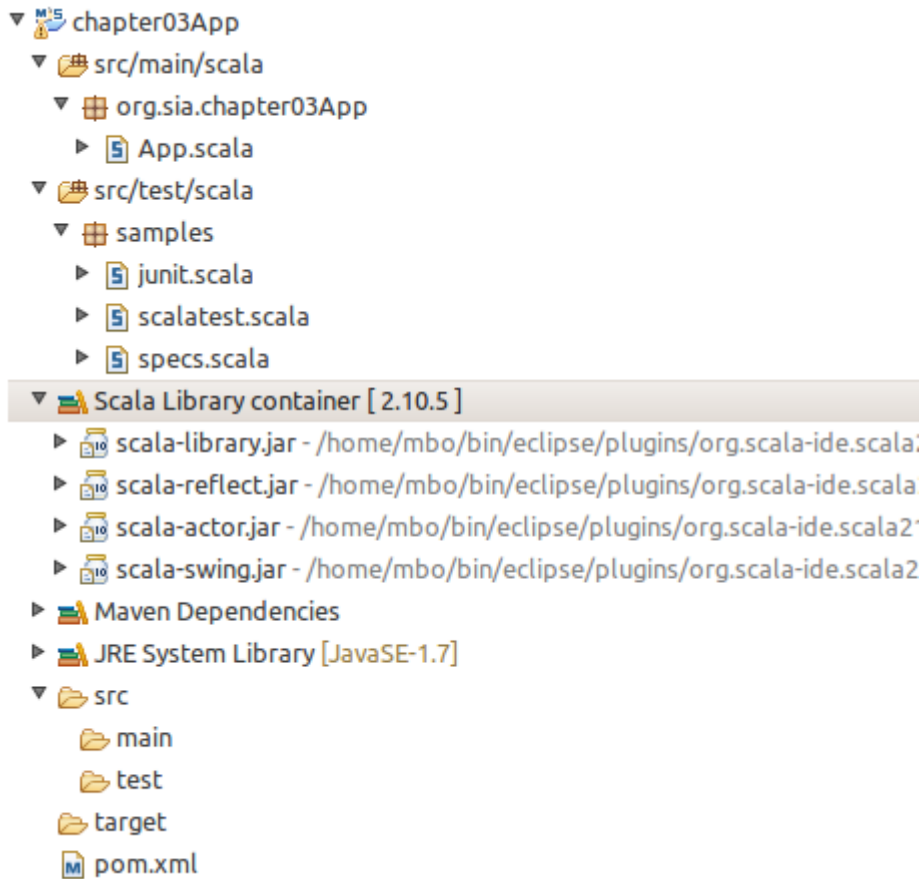


Figure 8 Package Explorer window after adjusting the Scala library version

Let's examine the structure of the generated project. Looking from the top, the first, root entry, is the project's main folder, always named the same way as the project. We call this folder **project's root** (or *project root*, interchangeably).

`src/main/scala` is your main Scala source folder (if you were to add some Java code into your project, you would create `src/main/java` source folder). Inside the `scala` folder, you have the root package<sup>6</sup>, `org.sia.chapter03App`, with a single Scala source file inside, `App.scala`. This is the Scala file where you'll start writing your application.

Next comes the main test folder with prepared samples for various test types, followed by the container for your Scala library. You are using Scala that came with ScalaIDE Eclipse plugin. Next, Maven Dependencies. If you open the `pom.xml` from the *project root* and switch to *Dependency Hierarchy* tab (figure 9), you'll have a much better view of the project's dependencies and their causality.

---

<sup>6</sup> On the file system level, `org.sia.chapter03App` is comprised of 3 folders, i.e. this is the path to the `App.scala` file: `chapter03App/src/main/scala/org/sia/chapter03App/App.scala`

## Dependency Hierarchy [test]

### Dependency Hierarchy



Figure 9 Project's libraries dependency hierarchy (in `pom.xml`)

There are only six libraries at the top level (all explicitly listed in your `pom.xml`). Each of those libraries brought with it its own dependencies, and many of those dependencies, in turn, have their own dependencies, and so on...

Below *Maven Dependencies* is **JDK** (Eclipse refers to JRE and JDK in the same way, as *JRE System Library*).

Further down in the *Package Explorer* window is again the `src` folder, but this time in a role of an ordinary folder (non-source, or more precisely, non-jvm-source folder), in case you want to add other types of resources to your project, such as images, JavaScript, HTML, CSS, anything that you don't want to be processed by the Eclipse JVM tooling.

Then there is the `target` folder, which is where compiled resources go (like `.class` or `.jar` files).

Finally, there is the all-encompassing `pom.xml`, which is the project's Maven specification.

You can get more information on my book *Spark in Action* at the book's [GitHub repo](#).