

O'REILLY®

Setting up Scala Projects

Daniel Hinojosa

About Me...

Daniel Hinojosa

Programmer, Developer, Consultant,
Instructor, and Speaker

Notable Content:

Testing in Scala (Book)

Beginning Scala Programming (Video)

Java & TDD (Video Training)

Scala Beyond The Basics

Scala Programming Fundamentals Series

Speaker:

OSCON

No Fluff Just Stuff Tour

DevNexus

dhinojosa@evolutionnext.com

@dhinojosa



Other Scala Classes

- Learn the Basics of Scala in 3 hours
- **Setting Up Scala Projects**
- Scala Beyond the Basics
 - Implicits
 - Pattern matching
- Scala Programming Fundamentals
 - Classes, Methods, Traits
- Scala Programming Fundamentals
 - Sealed, Collections, and Functions

Structure of the Class

- Mixture of discussion and labs
- Performed in IntelliJ, Eclipse, and VSCode
- Heavily SBT, but information on Maven, Gradle, Mill
- Definitely Ask Questions

Checklist before we proceed...

- We provided pre-setup configuration already.
- Be sure `JAVA_HOME` is set and that the following works:
 - `javac -version`
 - `java -version`

sbt

Basic SBT

SBT

- Simple Build Tool
 - Commonly used build tool among Scala projects
 - Parallel Processing
 - Tab Completed Tasks
 - Scoped Based
 - Java and Scala Support out of the box
 - Cross Compiling Support
 - Plugin Support
 - Embedded console with project class loading

SBT Advantages

- Built to run Scala and Java projects very well
- No separate build files like Maven for multi-module projects
- Large Community
- Large Amount of Plugins
- Uses JVM based common folder structure
- Rapid project scaffolding from git repositories

SBT Disadvantages

- Much higher learning curve
- Mature project builds hard to read
- Two repositories, one for SBT boot level, one for dependencies
- Repository separate from other build tools

SBT Installation with Brew

```
% brew update  
% brew cask install java  
% brew install scala  
% brew install sbt@1
```



SBT Manual Installation

- Visit `http://scala-sbt.org`
- Click the Download Button
- Download the appropriate binary for your system:
- Mac and Linux will load a `.tgz`, or a `.zip` file
- Windows will download an `.msi` executable
- For Mac and Linux you can expand with

```
tar -xvf sbt-1.0.2.tgz
```

sdkman.io



```
$ sdk install sbt
```

Create a manual build

Creating a Manual Build

- Create a project folder with whatever name you like
- In the directory, touch `build.sbt`
- `mkdir -p src/{main,test}/{scala,java}`
- Edit `build.sbt` with the following replacing
 - `<<project-name>>`
 - `<<version>>`
 - `<<scala-version>>`

```
name := "<<project-name>>"  
  
version := "<<version>>"  
  
scalaVersion := "<<scalaVersion>>"
```

Initializing the Project

- Reload will reload any changes from `build.sbt`
- Do so when after editing your build, to trigger changes
- Update will download any and all dependencies into
`~/.ivy2/cache`

```
% sbt reload update
```


In case you are behind a proxy...

```
export JAVA_OPTS="$JAVA_OPTS  
    -Dhttp.proxyHost=yourserver  
    -Dhttp.proxyPort=8080  
    -Dhttp.proxyUser=username  
    -Dhttp.proxyPassword=password"
```

- Don't include "http://" in the `yourserver` value
- Don't include the port in the `yourserver` value
- You probably also want to include `https.proxyHost` and `https.proxyPort` since a lot of stuff works over https

<https://stackoverflow.com/questions/13803459/how-to-use-sbt-from-behind-proxy>

Initializing the Project

- Reload will reload any changes from `build.sbt`
- Do so when after editing your build, to trigger changes
- Update will download any and all dependencies into
`~/.ivy2/cache`

```
% sbt reload update
```

Lab: Creating a Manual Build

- Create a project folder called `scala_rocks`
- In the directory, touch `build.sbt`
- `mkdir -p src/{main,test}/{scala,java}`
- Edit `build.sbt` with the following and reload and update the project

```
name := "scala_rocks"  
  
version := "1.0-SNAPSHOT"  
  
scalaVersion := "2.12.6"
```

Create a Giter8 build

Starting a Giter8 Project

- Giter8 : <https://github.com/foundweekends/giter8>
- Command line tool to generate files and directories from templates published on Github or any other Git repository
- Can be called using `sbt new`
- Visit <https://github.com/foundweekends/giter8/wiki/giter8-templates> for a list of templates

```
% sbt new scala/hello-world.g8
```

Lab: Create a build using Giter8 templates

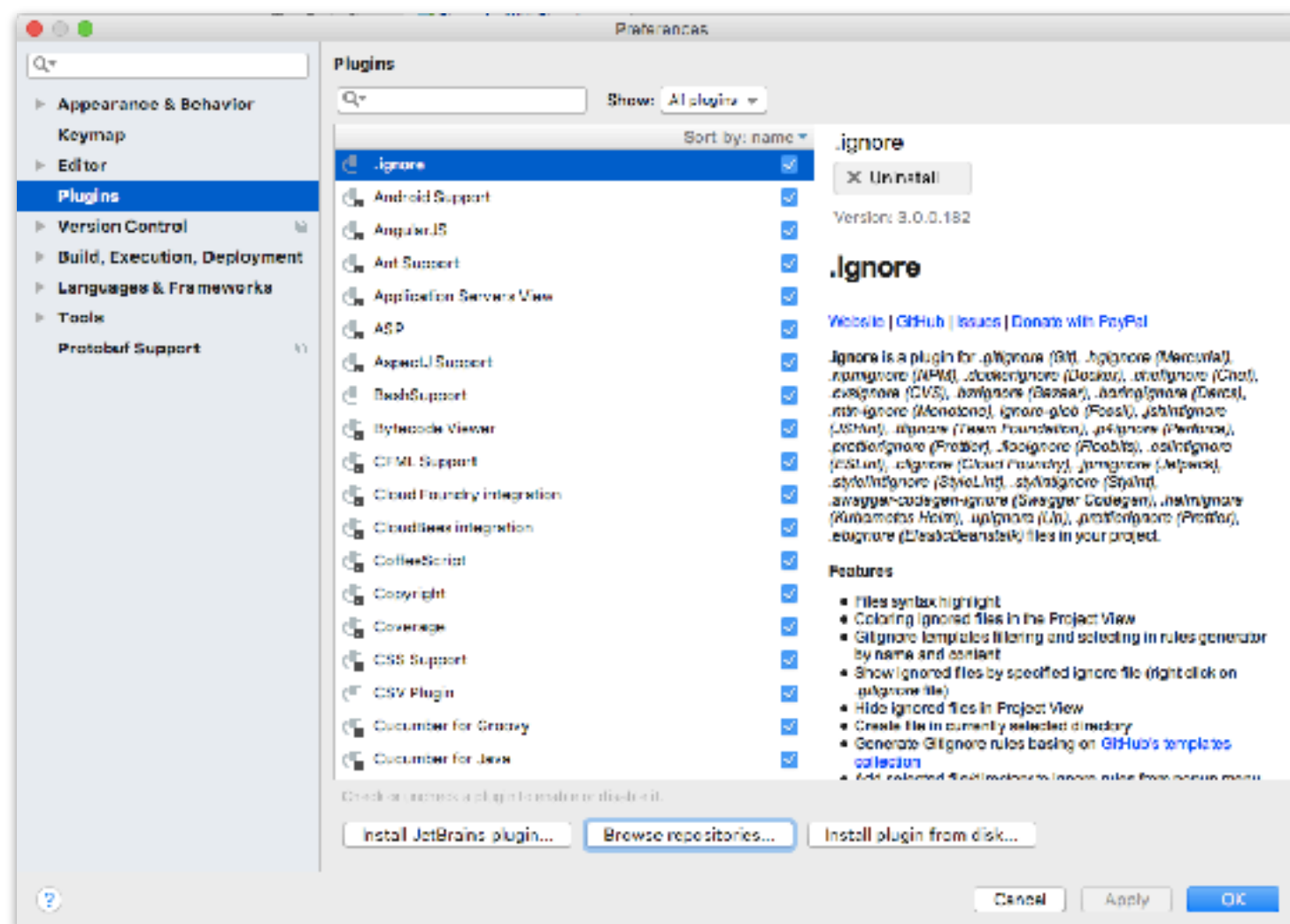
- Locate the `scala-seed.g8` template from the catalog
- Create a scala-seed project in it's own folder, please don't create inside the previous project
- Call the project `scala_giter_8`
- Inspect the layout using `tree` or `find` .
- Go into the `build.sbt` file and call your project `scala_giter_8`. It will be `hello_world` by default.



Importing your Project into IntelliJ

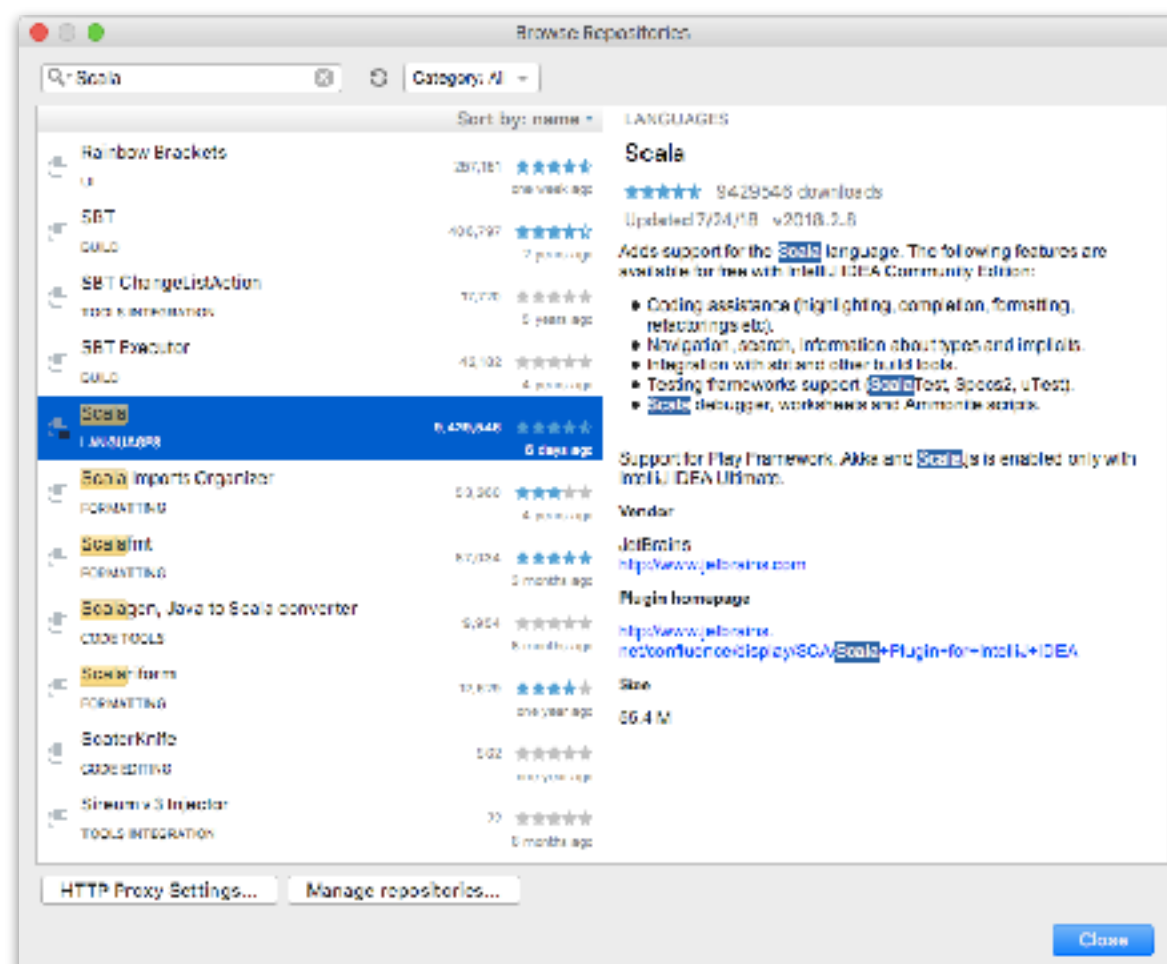
Importing Project into IntelliJ

- Go to Preferences | Plugins | Browse Repositories



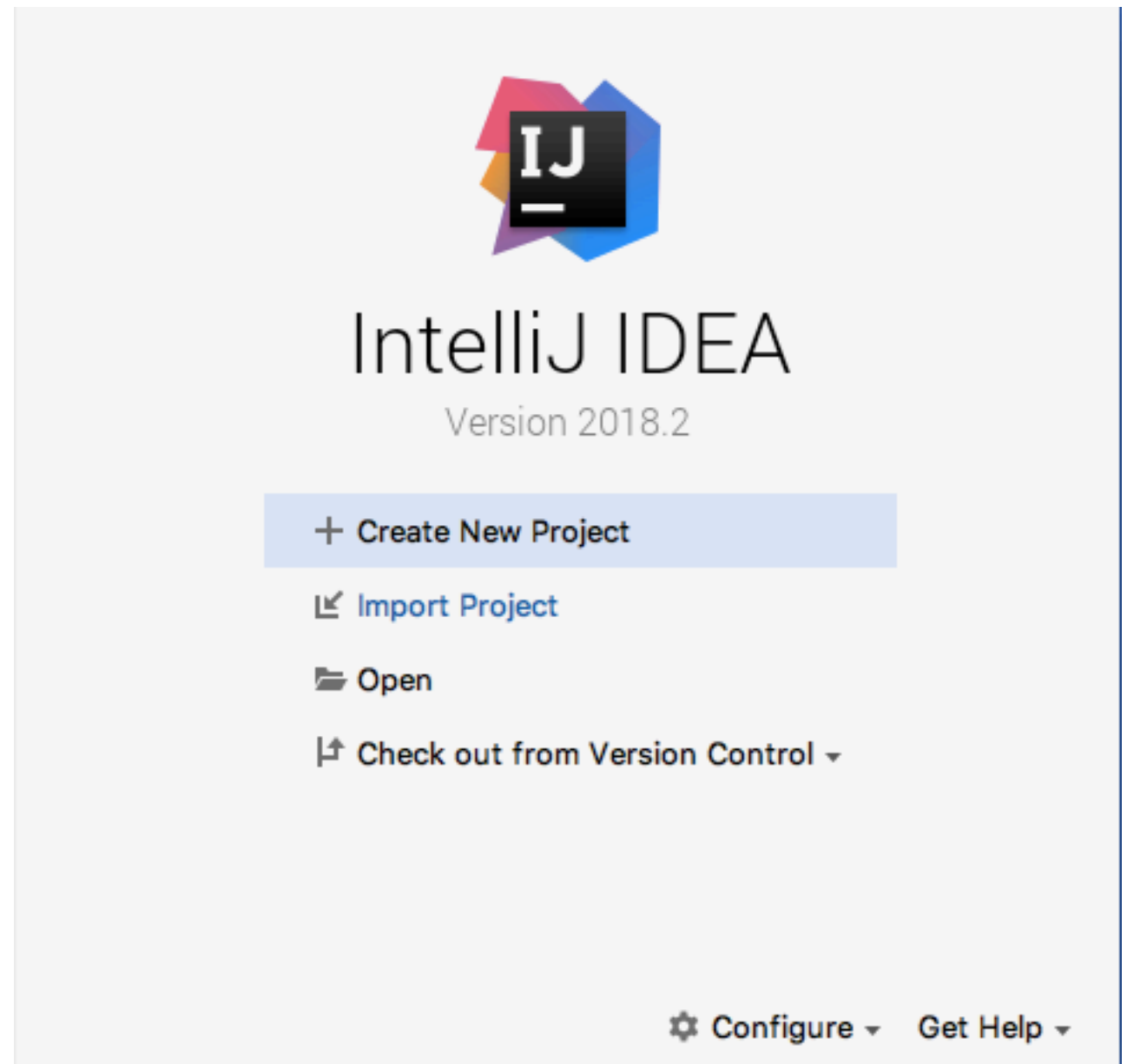
Importing Project into IntelliJ

- Find the Scala Plugin
- **Do not install the SBT Plugin**, the Scala Plugin has SBT included
- Click on Install
- Restart IntelliJ



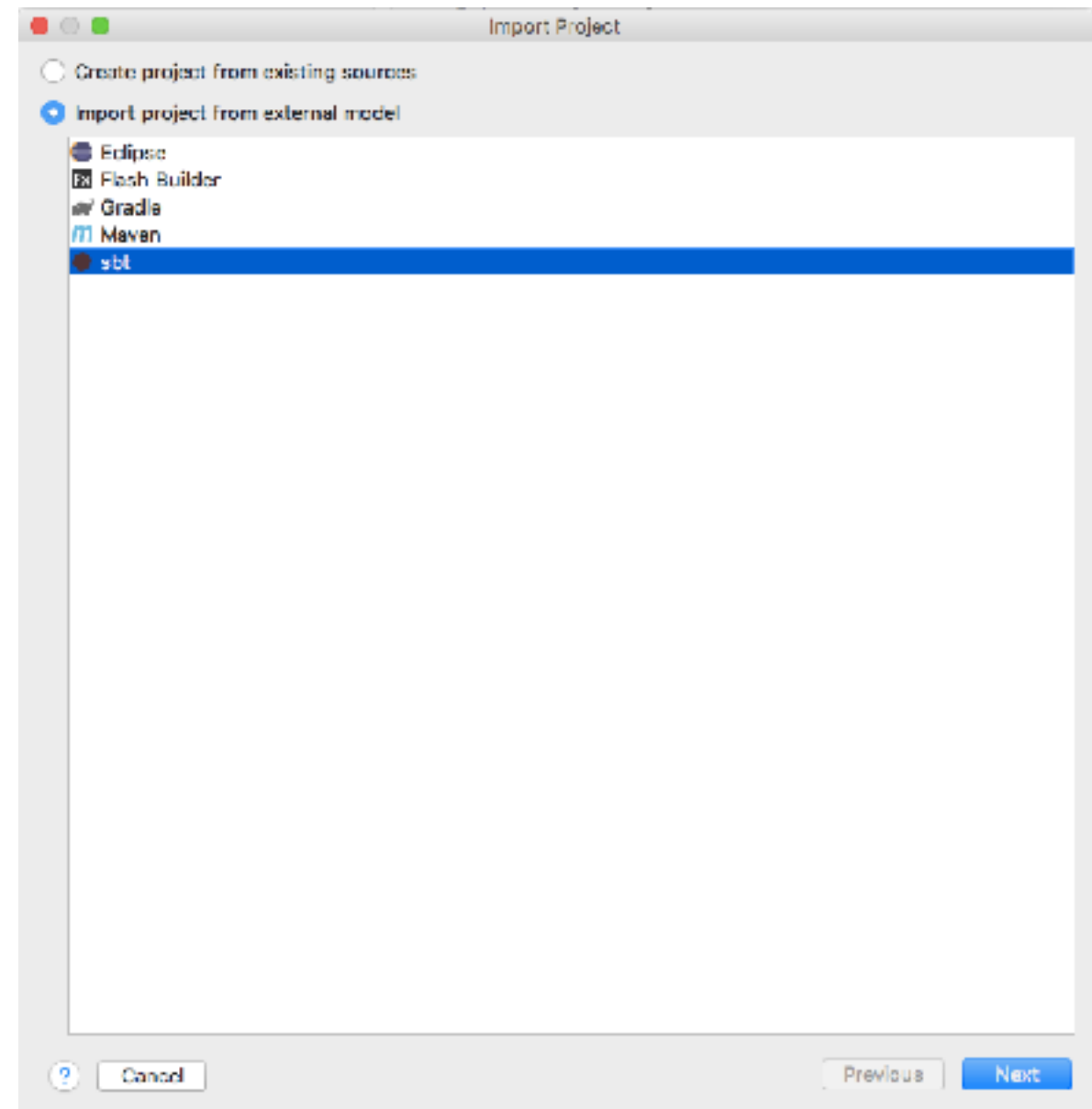
Importing Project into IntelliJ

- If you are in the splash screen, by closing all projects, you can select “Import Project”



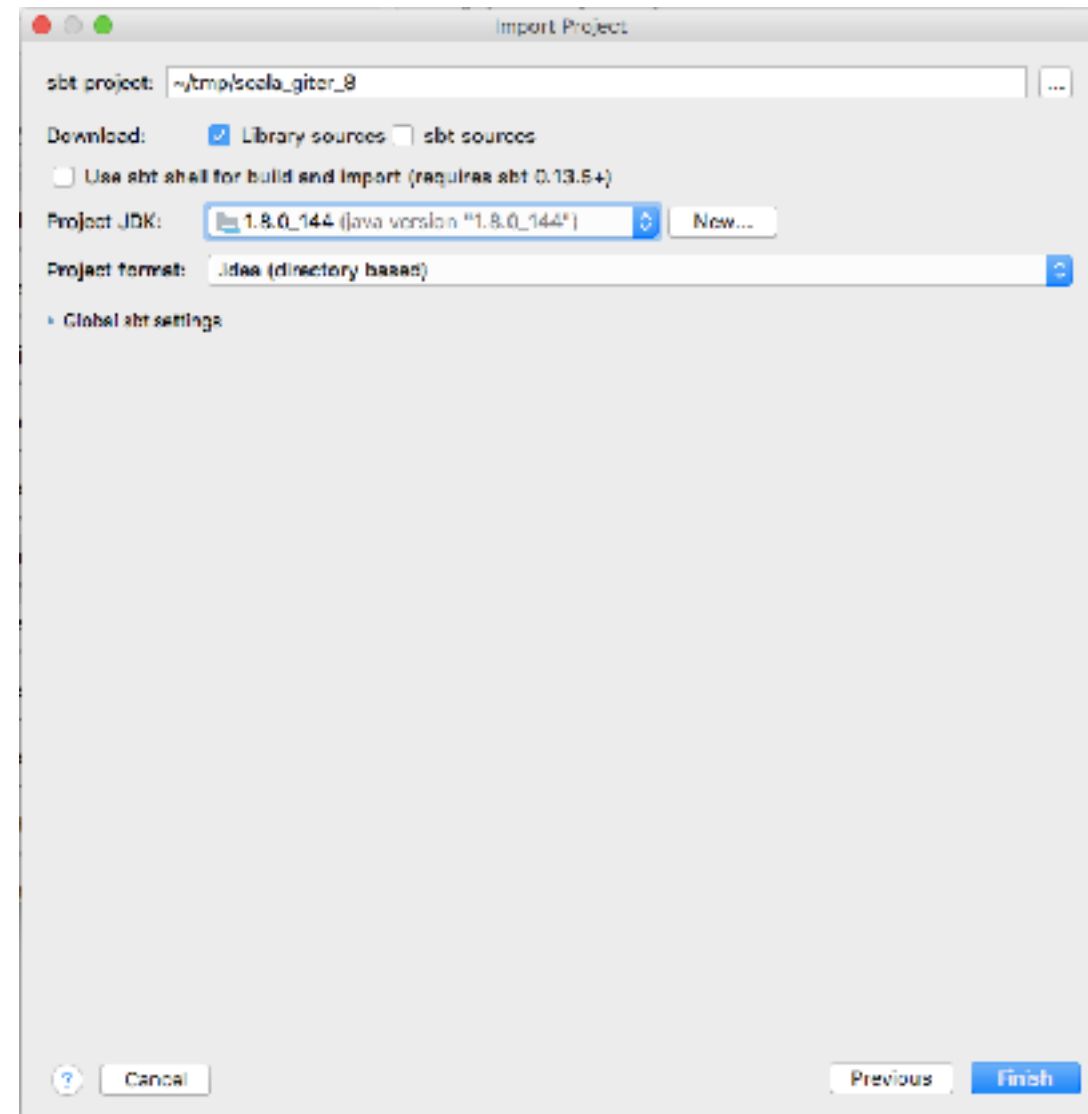
Importing Project into IntelliJ

- Select “sbt” as your project

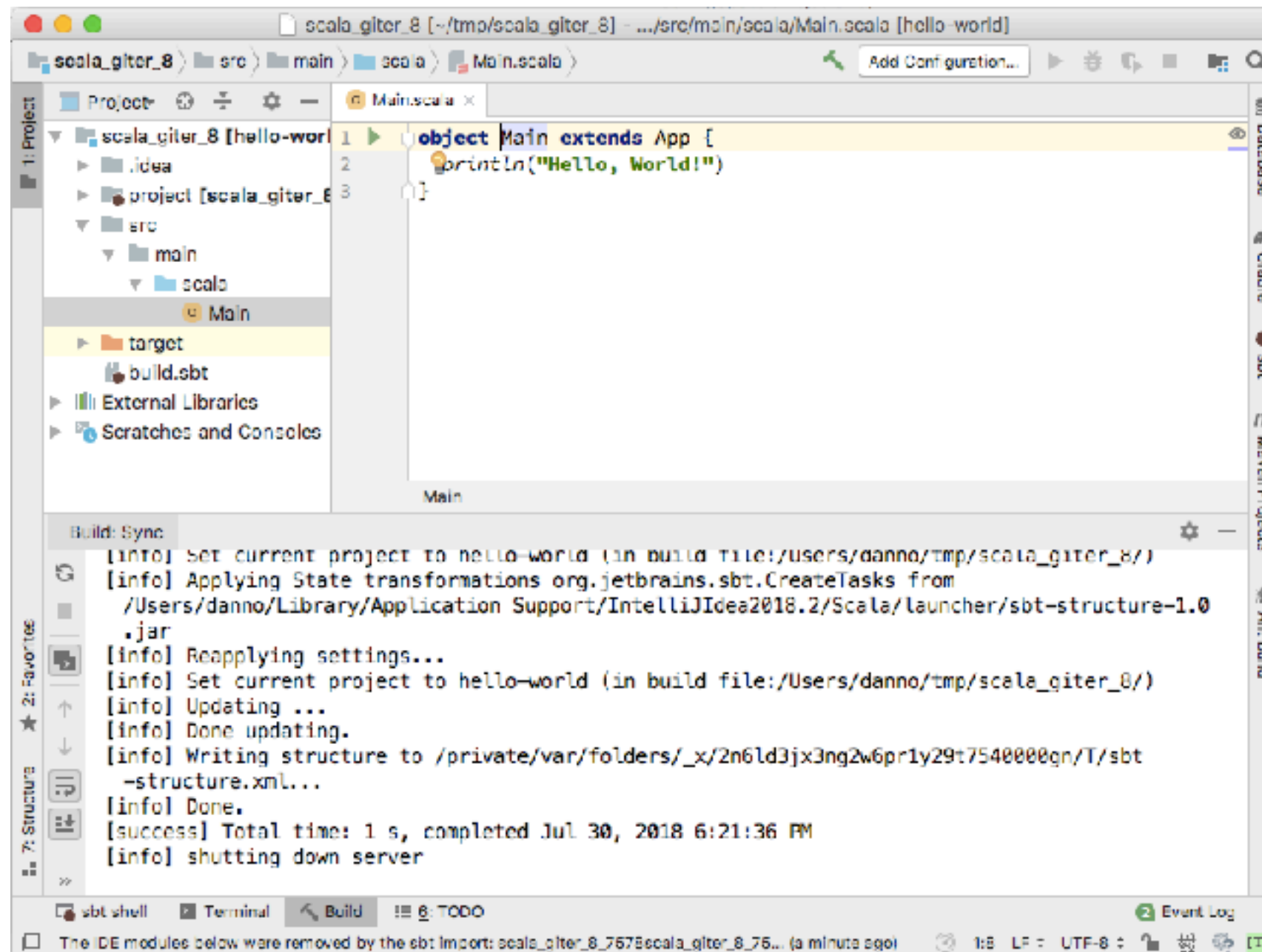


Importing Project into IntelliJ

- Select your JDK Preference, and select “Finish”



Importing Project into IntelliJ





Importing your Project into Eclipse

Setting up the Eclipse Plugin

- Either:
 - Create a file in the project directory of your sbt project called `plugins.sbt`
 - Open `plugins.sbt` if it is already created
- Or:
 - Create a global file `~/ .sbt/1.0/plugins/plugins.sbt`
 - Open `plugins.sbt` if it is already created

Setting up the Eclipse Plugin

- Add the following to either your global `plugins.sbt` file or your project `plugins.sbt` file
- Restart your sbt if you are in a shell

```
addSbtPlugin("com.typesafe.sbteclipse" % "sbteclipse-plugin" % "5.2.4")
```

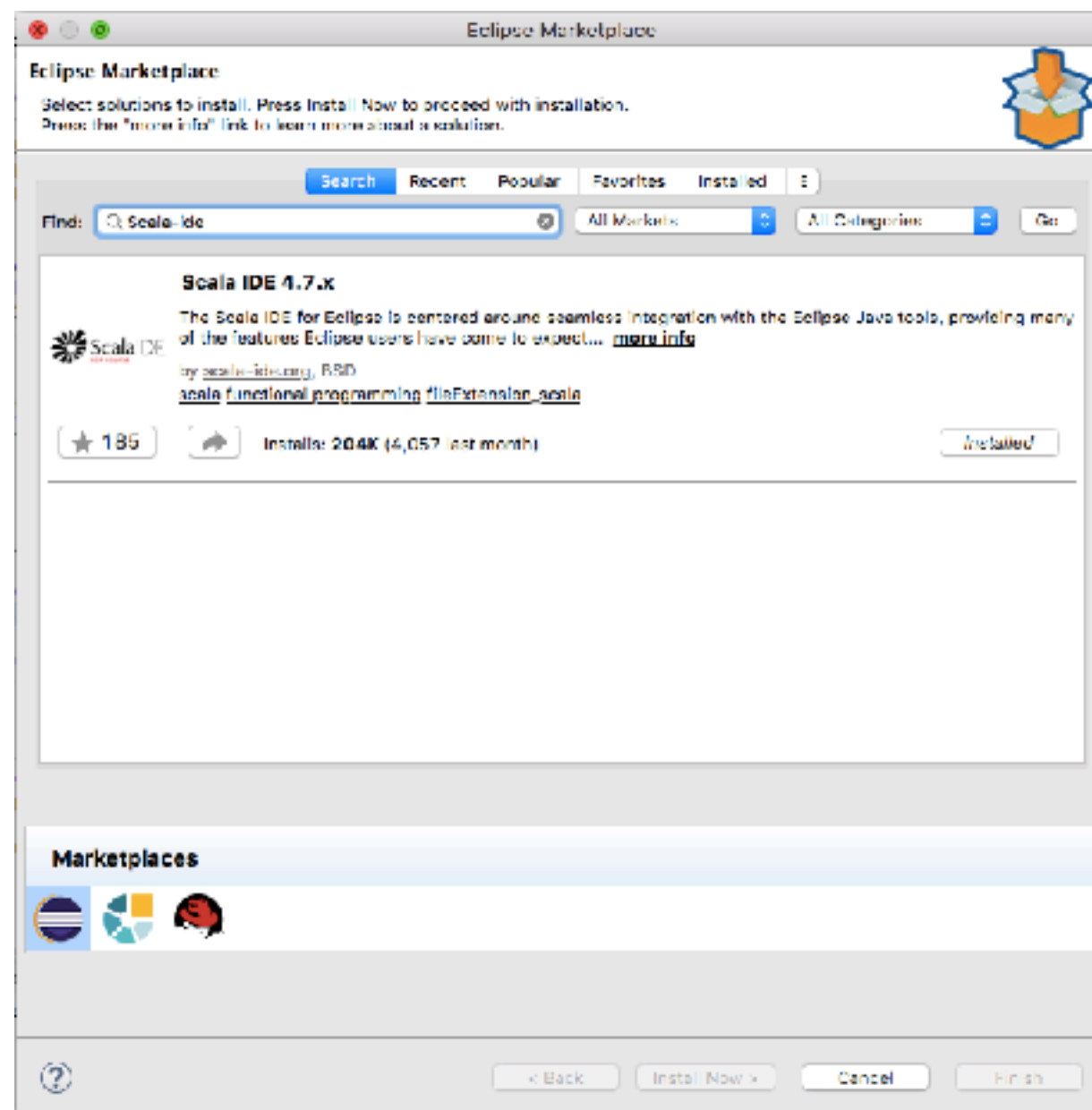

Run the eclipse command

- Run the eclipse command to generate Eclipse project file

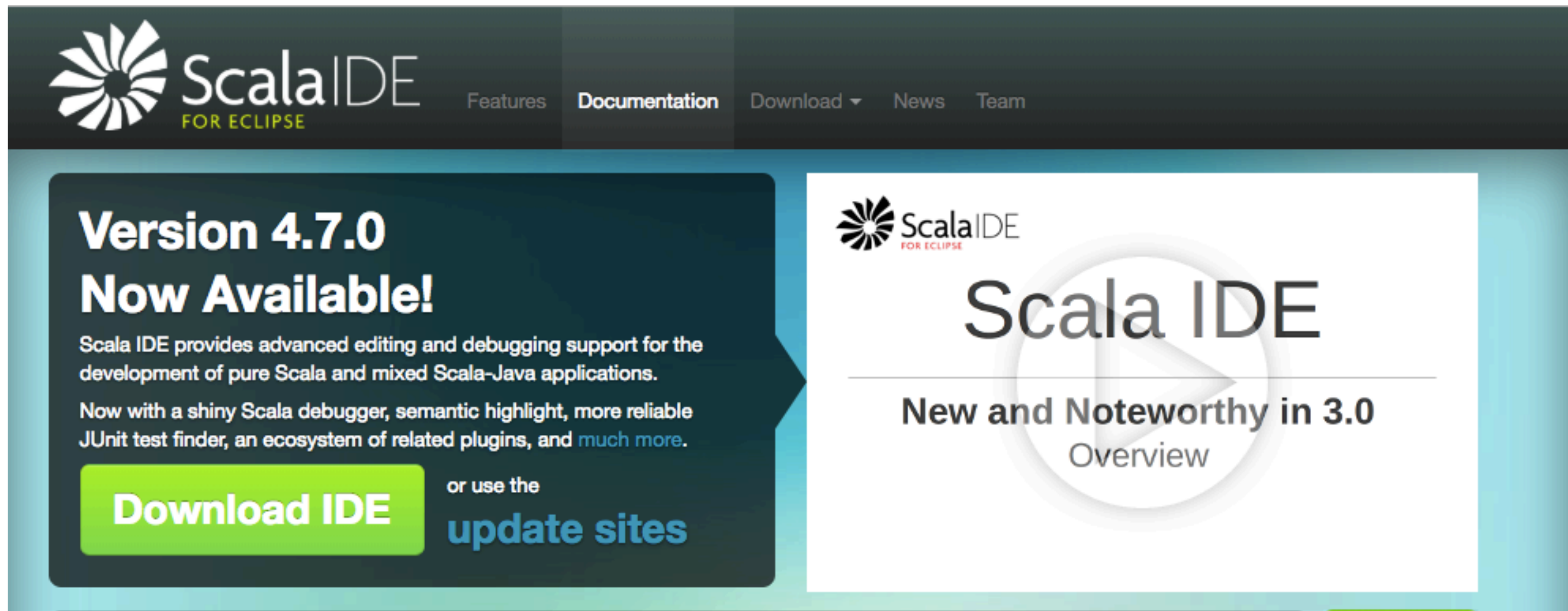
```
% sbt
sbt:scala_giter8> eclipse
[info] About to create Eclipse project files for your project(s).
[info] Updating ...
[info] Done updating.
[info] Successfully created Eclipse project files for project(s):
[info] scala_giter_8
```

Importing Project into Eclipse

- Go to Help | Eclipse Marketplace
- Search for Scala-IDE
- Install by accepting all defaults



Or, you can also download a full Scala-IDE



The screenshot shows the Scala IDE website. The top navigation bar includes the Scala IDE logo, 'FOR ECLIPSE', and links for Features, Documentation, Download, News, and Team. The main content area is split into two sections. The left section, on a dark blue background, announces 'Version 4.7.0 Now Available!' and describes the IDE's capabilities. It features a prominent green 'Download IDE' button and a link to 'update sites'. The right section, on a white background, features the Scala IDE logo and a large play button icon, with the text 'Scala IDE' and 'New and Noteworthy in 3.0 Overview'.

**Version 4.7.0
Now Available!**

Scala IDE provides advanced editing and debugging support for the development of pure Scala and mixed Scala-Java applications.

Now with a shiny Scala debugger, semantic highlight, more reliable JUnit test finder, an ecosystem of related plugins, and [much more](#).

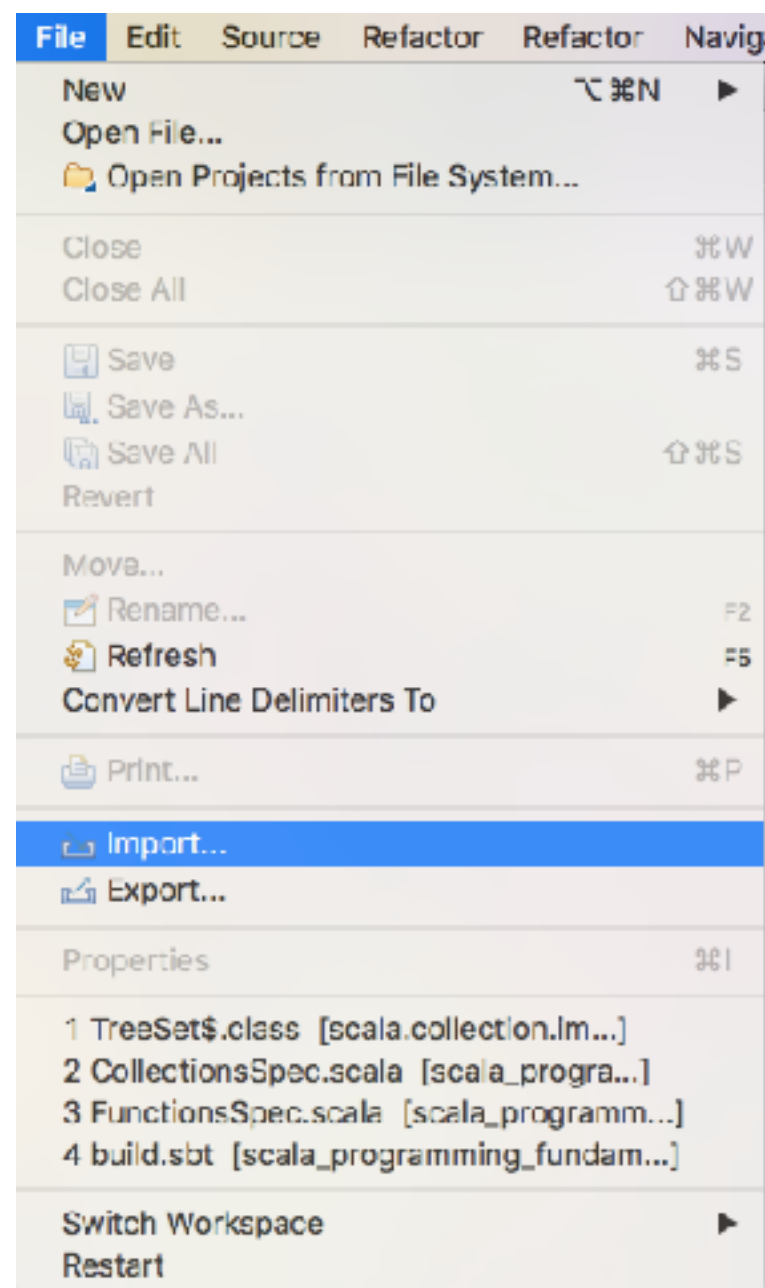
Download IDE or use the [update sites](#)

Scala IDE

New and Noteworthy in 3.0
Overview

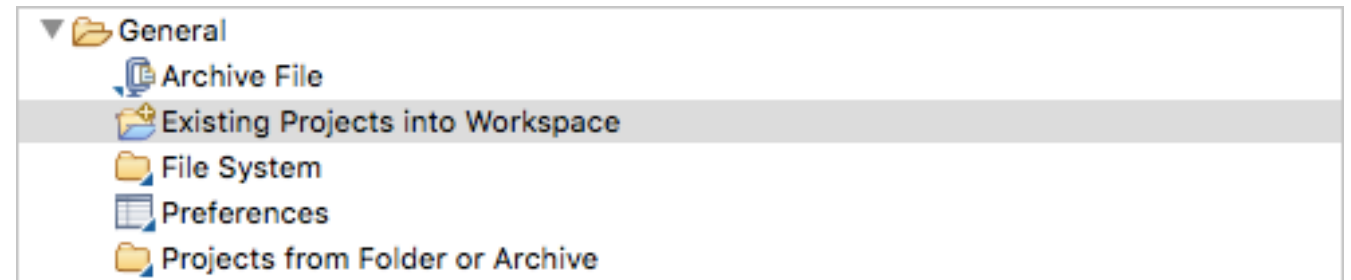
Importing Project into Eclipse

- Go to File | Import



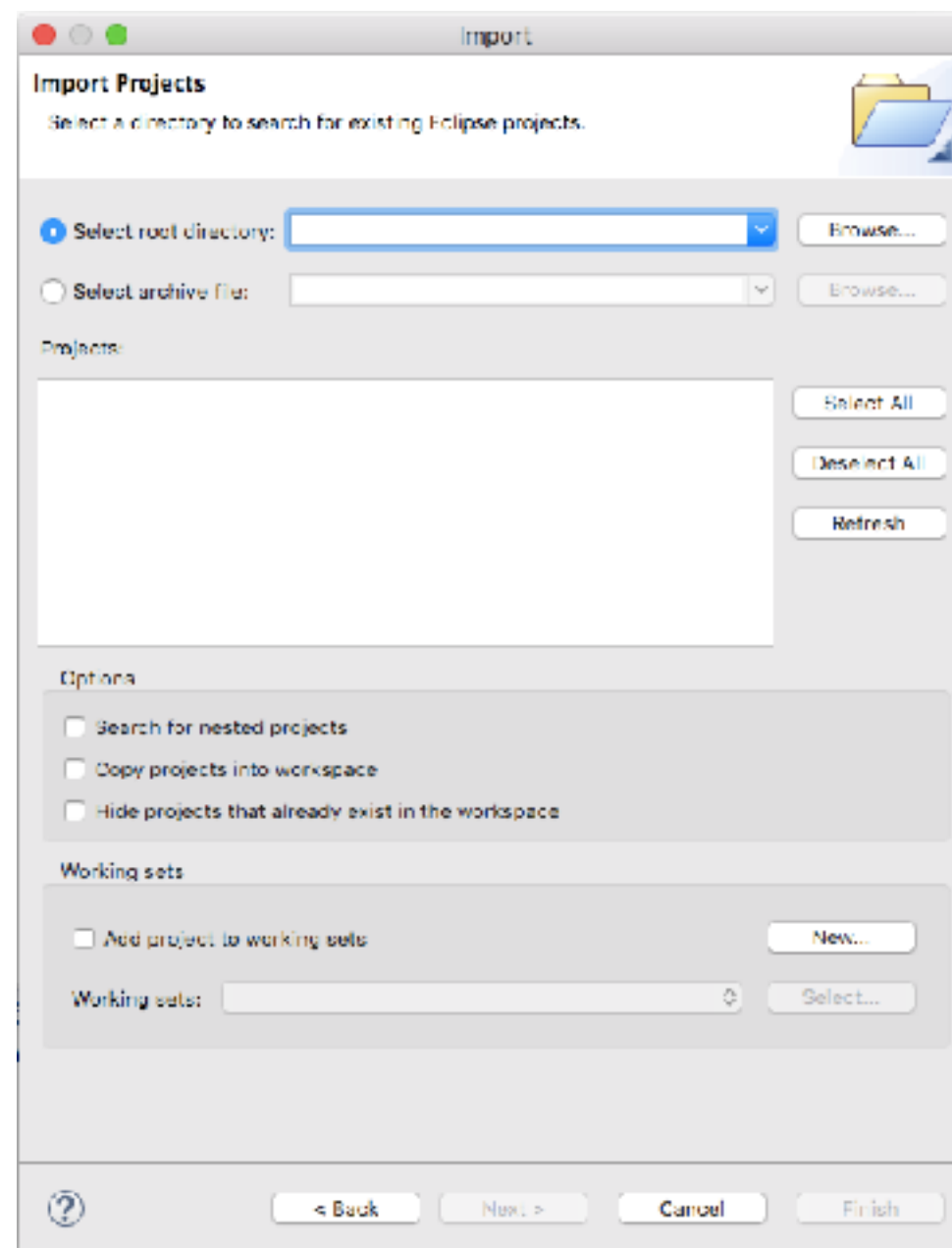
Select Import Existing Projects

- In the next window, select “Existing Projects into Workspace”
- Hit Next



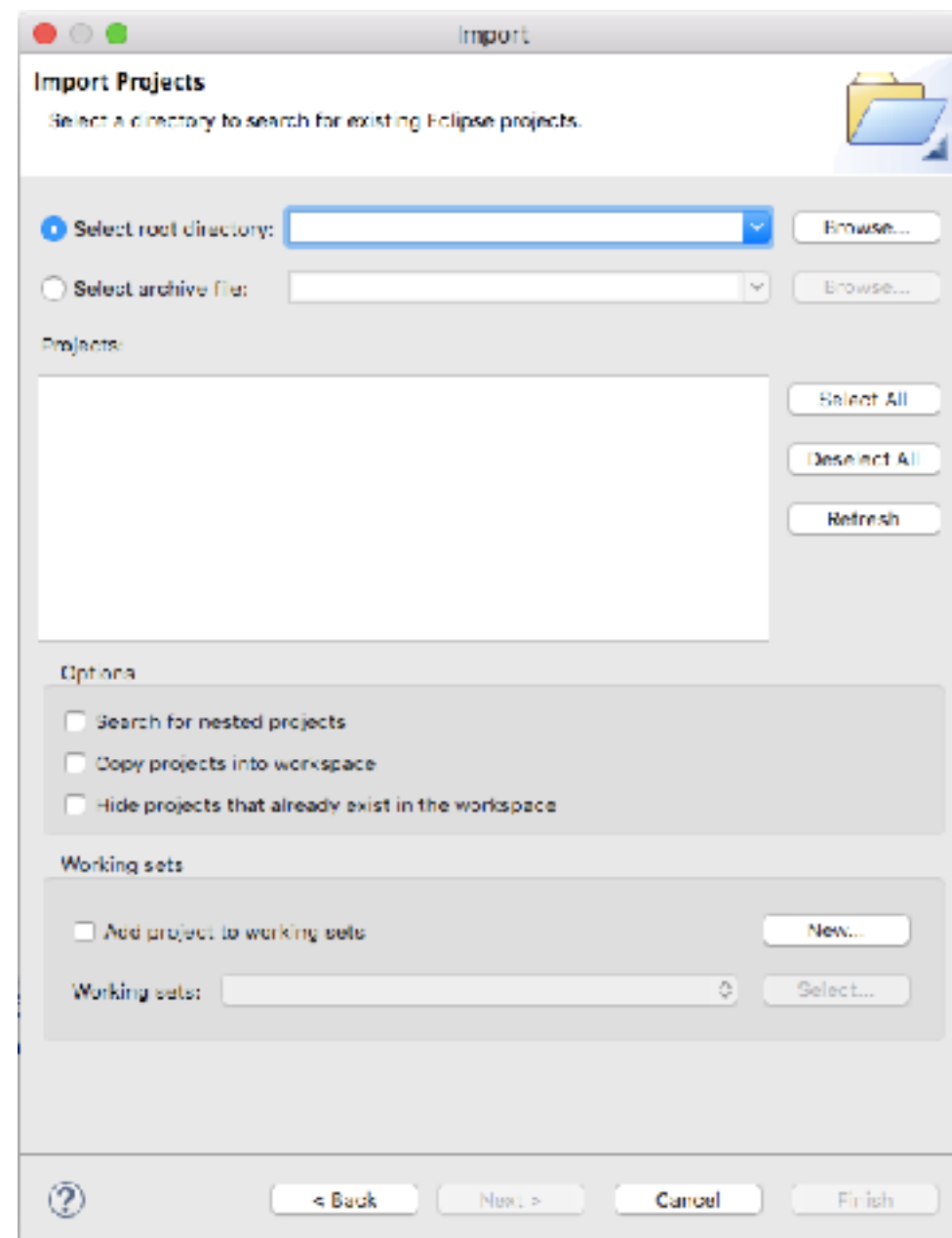
Locate and Open your project

- Select your project location.
- Hit Finish

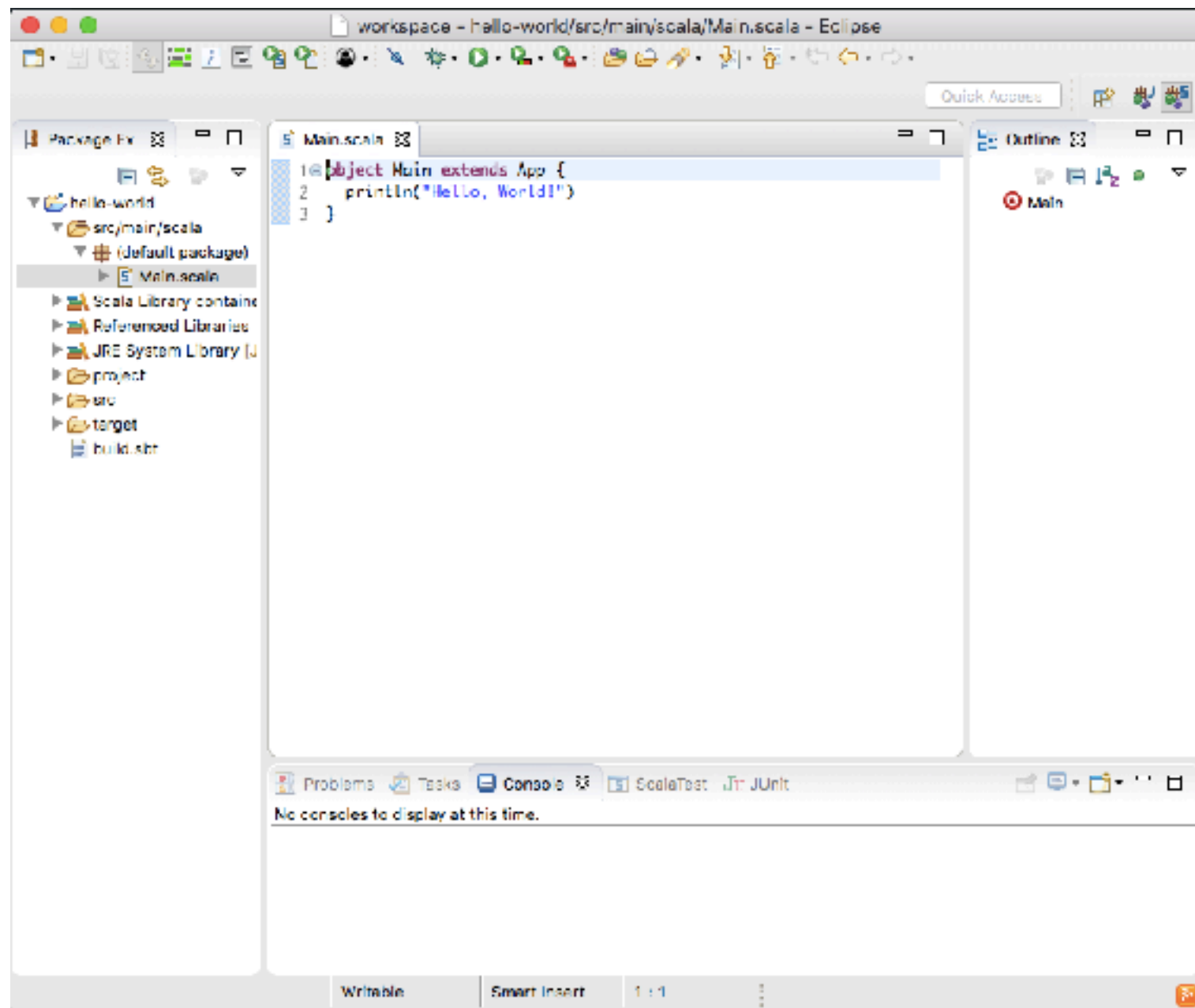


Locate and Open your project

- In the next window, select “Existing Projects into Workspace”
- Hit Next



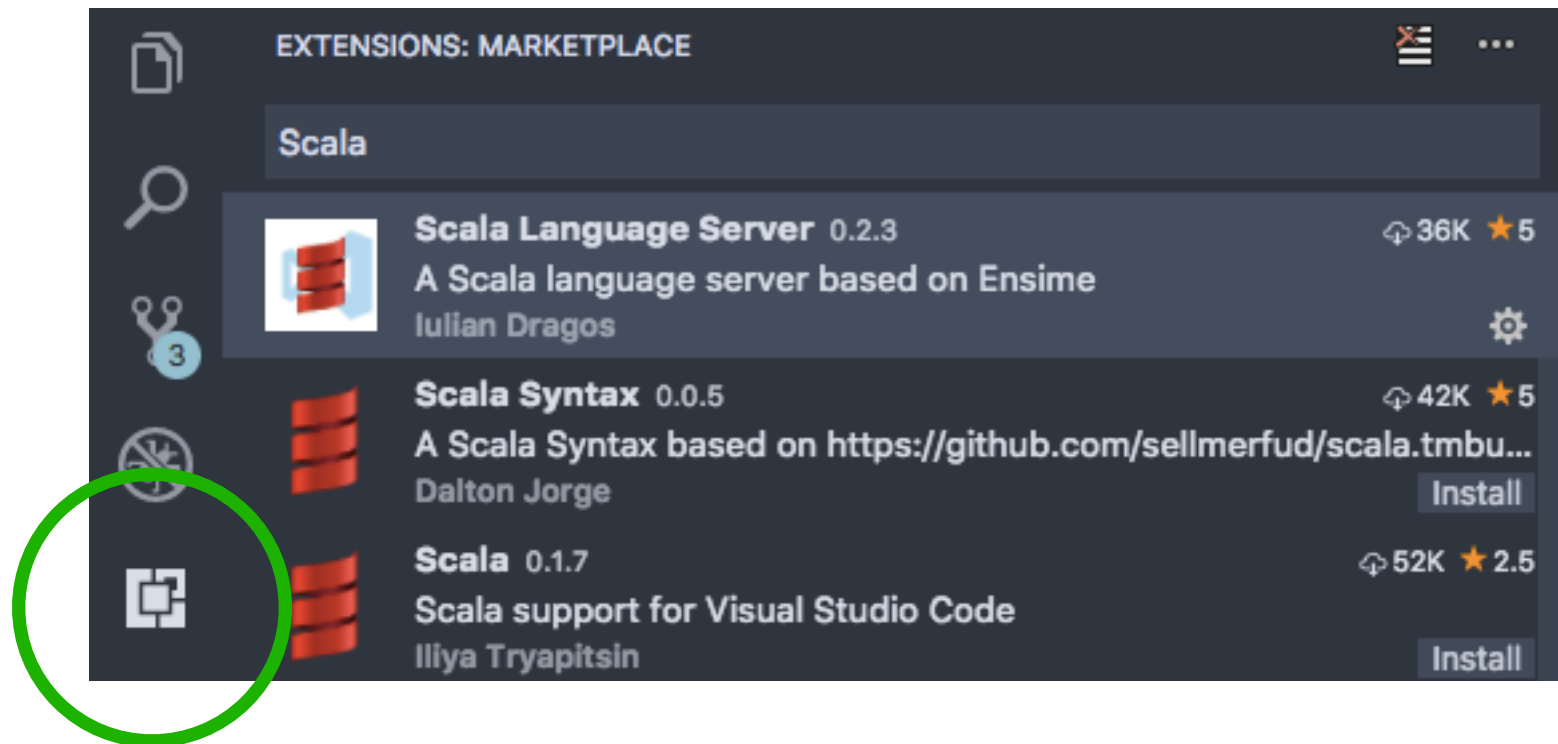
Importing Project into Eclipse





Importing your Project into VSCode

Install the Extension for VSCode



- Select Extensions on the left of VSCode
- Do a search for “Scala” and install Scala Language Server
- Select Install
- Will likely require a reload



<http://ensime.github.io/>

About Enzyme

- Brings full-fledged IDE experience to Editors
- Show inferred types.
- Contextual completion.
- Semantic highlighting with implicit expansions.
- Jump to source code or documentation.
- Refactorings (rename, organise imports).
- Errors and warnings in your code (i.e. “red squiggles”)
- Appeals to hackers, minimalists and connoisseurs

Supported Text Editors

- Emacs
- Atom
- Vim
- VSCode (What we will be using)
- Sublime

For more specifics on your favorite editor: <http://ensime.github.io/editors/>

Setting up the Enzyme in SBT

- Create a global file `~/ .sbt/1.0/plugins/ plugins.sbt`
- Or open `plugins.sbt` if it is already created
- Add the following to `plugins.sbt`

```
addSbtPlugin("org.ensime" % "sbt-ensime" % "2.5.1")
```

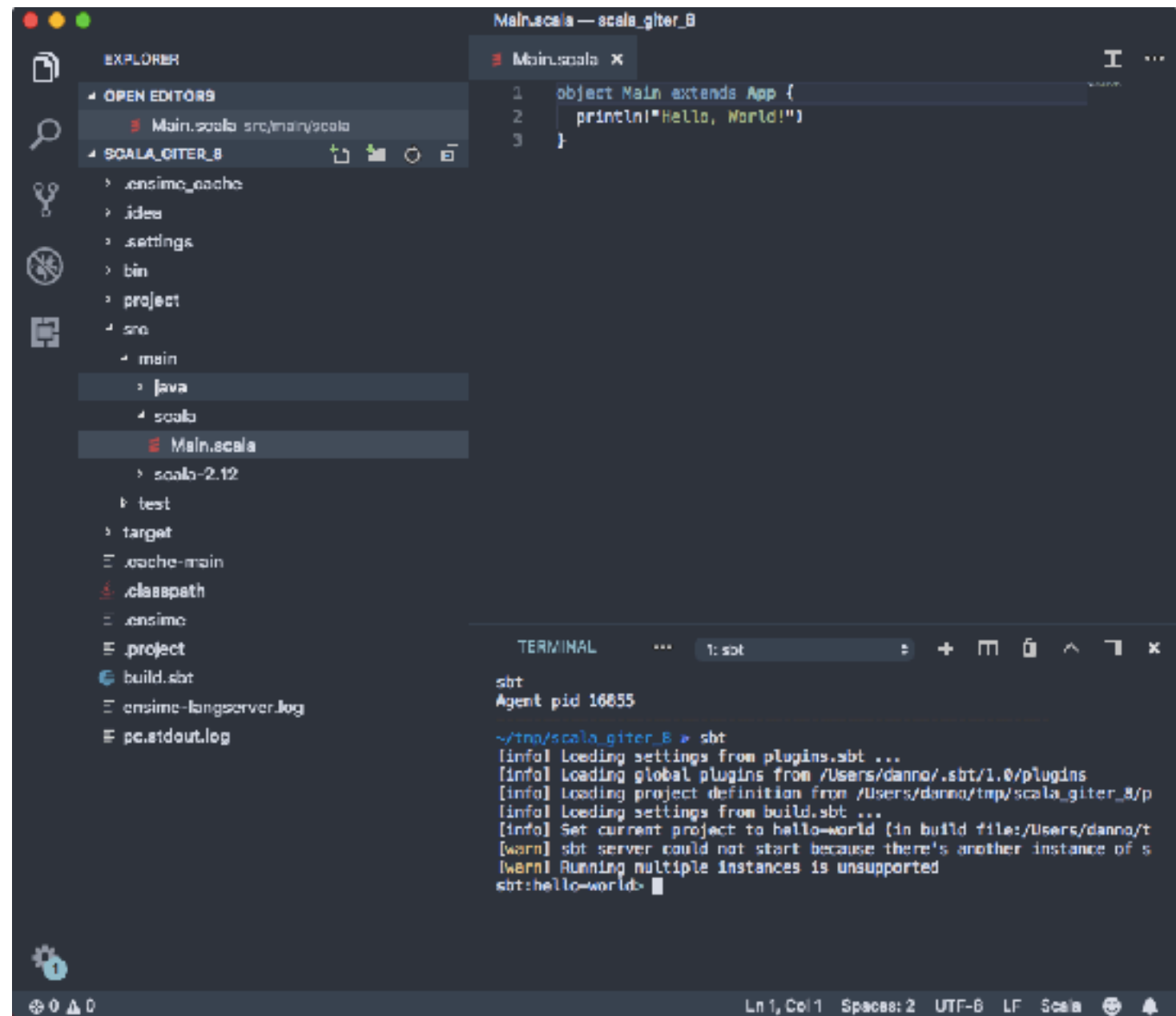
Build an Enzyme Config and Project Config

- In your project run an SBT command called, `enzymeConfig`
- Then run `enzymeProjectConfig` within SBT
- This will require a multiple downloads

```
% sbt enzymeConfig  
% sbt enzymeProjectConfig
```

Bringing your project into VSCode

- Go to File | Open
- Locate your project that you built an `ensimeConfig`
- Select Open



Lab: Import your Giter8 project

- Import your `scala_giter_8` project into your favorite IDE or editor.
- Unless you are awesome and want to try all of the main IDEs, do them all!
- You can also go for your own favorite editor using Enzyme
 - Compatibility may be an issue
 - See compatibility list of possible issues
- ***Please ask questions! I can help with most but not all editors though.***

Basic SBT Commands

<code>clean</code>	Deletes all generated files (in the target directory).
<code>compile</code>	Compiles the main sources (in <code>src/main/scala</code> and <code>src/main/java</code> directories).
<code>test</code>	Compiles and runs all tests.
<code>testOnly</code>	Compiles and runs one test, requires class name after the command.
<code>console</code>	Starts the Scala interpreter with a classpath including the compiled sources and all dependencies.

<code>run <argument>*</code>	Runs the main class for the project in the same virtual machine as sbt.
<code>package</code>	Creates a jar file containing the files in <code>src/main/resources</code> and the classes compiled from <code>src/main/scala</code> and <code>src/main/java</code>
<code>help <command></code>	Displays detailed help for the specified command. If no command is provided, displays brief descriptions of all commands
<code>reload</code>	Reloads the build definition (<code>build.sbt</code> , <code>project/.scala</code> , <code>project/.sbt</code> files). Needed if you change the build definition.
<code>update</code>	Download any of the dependencies that are required by your project

Running from your shell

```
% sbt clean

[info] Loading settings from plugins.sbt ...
[info] Loading global plugins from /Users/danno/.sbt/1.0/plugins
[info] Updating ProjectRef(uri("file:/Users/danno/.sbt/1.0/
plugins/"), "global-plugins")...
[info] Done updating.
[info] Loading settings from plugins.sbt ...
[info] Loading project definition from /Users/danno/Development/
scala_rules/project
[info] Updating ProjectRef(uri("file:/Users/danno/Development/
scala_rules/project/"), "scala_rules-build")...
[info] Done updating.
[info] Loading settings from build.sbt ...
[info] Set current project to scala_rules (in build file:/Users/
danno/Development/scala_rules/)
[success] Total time: 1 s, completed Jul 12, 2018, 9:31:00 PM
```

Running from SBT's shell

```
% sbt
[info] Loading settings from plugins.sbt ...
[info] Loading global plugins from /Users/danno/.sbt/1.0/plugins
[info] Loading settings from plugins.sbt ...
[info] Loading project definition from /Users/danno/Development/scala_rules/project
[info] Loading settings from build.sbt ...
[info] Set current project to scala_bootcamp (in build file:/Users/danno/Development/scala_rules/)
[info] sbt server started at local:///Users/danno/.sbt/1.0/server/51b90af7ed1c80f5a6ad/sock
sbt:scala_rules> clean
```

Testing

>ScalaTestTM

simply productive

ScalaTest is designed to increase your team's productivity through simple, clear tests and executable specifications that improve both code and communication.

<http://www.scalatest.org/>

Adding Scala Test Dependency

GroupId	ArtifactId	Version	Updated
org.scalatest	scalatest_2.12	3.0.6-SNAP1	15-Jun-2018
org.scalatest	scalatest_2.12	3.0.5-M1	12-Feb-2018
org.scalatest	scalatest_2.12	3.0.5	29-Jan-2018
org.scalatest	scalatest_2.12	3.2.0-SNAP10	25-Jan-2018
org.scalatest	scalatest_2.12	3.1.0-SNAP6	24-Jan-2018
org.scalatest	scalatest_2.12	3.0.4	17-Aug-2017
org.scalatest	scalatest_2.12	3.2.0-SNAP9	28-Jun-2017
org.scalatest	scalatest_2.12	3.2.0-SNAP8	21-Jun-2017
org.scalatest	scalatest_2.12	3.2.0-SNAP7	19-Jun-2017
org.scalatest	scalatest_2.12	3.0.3	20-Apr-2017
org.scalatest	scalatest_2.12	3.0.2	18-Apr-2017

Adding Scala Test Dependency

GroupId	ArtifactId	Version	Updated
org.scalatest	scalatest_2.12	3.0.6-SNAP1	15-Jun-2018
org.scalatest	scalatest_2.12	3.0.5-M1	12-Feb-2018
org.scalatest	scalatest_2.12	3.0.5	29-Jan-2018
org.scalatest	scalatest_2.12	3.2.0-SNAP10	25-Jan-2018
org.scalatest	scalatest_2.12	3.1.0-SNAP6	24-Jan-2018
org.scalatest	scalatest_2.12	3.0.4	17-Aug-2017
org.scalatest	scalatest_2.12	3.2.0-SNAP9	28-Jun-2017
org.scalatest	scalatest_2.12	3.2.0-SNAP8	21-Jun-2017
org.scalatest	scalatest_2.12	3.2.0-SNAP7	19-Jun-2017
org.scalatest	scalatest_2.12	3.0.3	20-Apr-2017
org.scalatest	scalatest_2.12	3.0.2	18-Apr-2017

Adding Scala Test Dependency

GroupId	ArtifactId	Version	Updated
org.scalatest	scalatest_2.12	3.0.6-SNAP1	15-Jun-2018
org.scalatest	scalatest_2.12	3.0.5-M1	12-Feb-2018
org.scalatest	scalatest_2.12	3.0.5	29-Jan-2018
org.scalatest	scalatest_2.12	3.2.0-SNAP10	25-Jan-2018
org.scalatest	scalatest_2.12	3.1.0-SNAP6	24-Jan-2018
org.scalatest	scalatest_2.12	3.0.4	18-Apr-2017
org.scalatest	scalatest_2.12	3.0.3	18-Apr-2017
org.scalatest	scalatest_2.12	3.0.2	18-Apr-2017

```
name := "<<project-name>>"  
version := "<<version>>"  
scalaVersion := "<<scalaVersion>>"
```

Adding Scala Test Dependency

```
name := "<<project-name>>"
```

```
version := "<<version>>"
```

```
scalaVersion := "<<scalaVersion>>"
```

```
libraryDependencies += "<groupId>" % "<artifactID>" % "<version>"
```

Adding Scala Test Dependency

```
name := "<<project-name>>"
```

```
version := "<<version>>"
```

```
scalaVersion := "<<scalaVersion>>"
```

```
libraryDependencies += "<groupId>" % "<artifactID>" % "<version>"
```

Adding Scala Test Dependency

```
name := "<<project-name>>"
```

```
version := "<<version>>"
```

```
scalaVersion := "<<scalaVersion>>"
```

```
libraryDependencies += "org.scalatest" % "scalatest_2.12" % "3.0.5"
```

Adding Scala Test Dependency

We will be based off of the version

```
name := "<<project-name>>"
```

```
version := "<<version>>"
```

```
scalaVersion := "2.12"
```

```
libraryDependencies += "org.scalatest" %% "scalatest" % "3.0.5"
```

Selecting a Flavor of Test

- ScalaTest has different flavors
- Depending on your testing style and what you require
- http://www.scalatest.org/user_guide/selecting_a_style

FunSuite

```
import org.scalatest.FunSuite

class SetSuite extends FunSuite {

  test("An empty Set should have size 0") {
    assert(Set.empty.size == 0)
  }

  test("Invoking head on an empty Set should produce NoSuchElementException") {
    assertThrows[NoSuchElementException] {
      Set.empty.head
    }
  }
}
```

FlatSpec

```
import org.scalatest.FlatSpec

class SetSpec extends FlatSpec {

  "An empty Set" should "have size 0" in {
    assert(Set.empty.size == 0)
  }

  it should "produce NoSuchElementException when head is invoked" in {
    assertThrows[NoSuchElementException] {
      Set.empty.head
    }
  }
}
```

Fun Spec

```
import org.scalatest.FunSpec

class SetSpec extends FunSpec {

  describe("A Set") {
    describe("when empty") {
      it("should have size 0") {
        assert(Set.empty.size == 0)
      }

      it("should produce NoSuchElementException when head is invoked") {
        assertThrows[NoSuchElementException] {
          Set.empty.head
        }
      }
    }
  }
}
```

WordSpec

```
import org.scalatest.WordSpec

class SetSpec extends WordSpec {

  "A Set" when {
    "empty" should {
      "have size 0" in {
        assert(Set.empty.size == 0)
      }

      "produce NoSuchElementException when head is invoked" in {
        assertThrows[NoSuchElementException] {
          Set.empty.head
        }
      }
    }
  }
}
```

FreeSpec

```
import org.scalatest.FreeSpec

class SetSpec extends FreeSpec {

  "A Set" - {
    "when empty" - {
      "should have size 0" in {
        assert(Set.empty.size == 0)
      }

      "should produce NoSuchElementException when head is invoked" in {
        assertThrows[NoSuchElementException] {
          Set.empty.head
        }
      }
    }
  }
}
```

Creating a Test

- Be sure to put testing data in `src/test/scala` (Scala) or `src/test/java` (Java)
- Be sure to put production data in `src/main/scala` (Scala) or `src/test/scala` (Scala)

Pending

```
import org.scalatest.FunSpec
class ExampleSpec extends FunSpec {
  describe("A Stack") {
    it("should pop values in last-in-first-out order") (pending)
    it("should throw NoSuchElementException if an empty stack is popped") (pending)
  }
}
```

- Be sure to put pending at the top of all code
- This is useful in case you working on some test and you are not done with it

Matchers

- A Matcher is a trait that offers linguistic syntax for assertions
- http://www.scalatest.org/user_guide/using_matchers

```
import org.scalatest._  
class ExampleSpec extends FlatSpec with Matchers { ...
```

Example Matcher

```
List(1, 2, 3, 4, 5) should contain oneOf (5, 7, 9)  
Some(7) should contain oneOf (5, 7, 9)  
"howdy" should contain oneOf ('a', 'b', 'c', 'd')
```


Testing All Classes in SBT

```
% sbt test
```

Testing A Specific Class in SBT

```
% sbt testOnly com.xyzcorp.EmployeeSpec
```

Testing A Specific Test in a class in SBT

```
% sbt testOnly com.xyzcorp.EmployeeSpec -- -z "This test"
```

Lab: Testing your code

- Create a test called `EmployeeSpec`
- Choose whatever flavor test you like
- Mix in the `Matchers` trait
- Create a test spec that will:
 - Create an `Employee` that accepts a `firstName` and `lastName`
 - Calls the `Employee` instance's `fullName` method (that you will create)
 - Ensure that it is the combination of `firstName` and `lastName` with a space in between

SBT Console

- What's great about console is the ability use a REPL with your code from `src/main/scala`

```
% sbt
sbt:scala_giter_8> console

[info] Starting scala interpreter...
Welcome to Scala 2.12.6 (Java HotSpot(TM) 64-Bit
Server VM, Java 10.0.1).
Type in expressions for evaluation. Or try :help.

scala>
```

Lab: Using Console

- Run console from within your project sbt
- Create an `Employee` object from within your console

Other Build Tools

maven

```
<project>
...
<build>
  <sourceDirectory>src/main/scala</sourceDirectory>
  <testSourceDirectory>src/test/scala</testSourceDirectory>
  <plugins>
    <plugin>
      <groupId>net.alchim31.maven</groupId>
      <artifactId>scala-maven-plugin</artifactId>
      <version>3.4.2</version>
    </plugin>
  </plugins>
</build>
</project>
```


maven

Demo



- Gradle comes pre-installed with a plugin
- It is a matter of turning it on in your build.gradle file

```
apply plugin: 'scala'
```

- Be sure to place content in src/main/scala and src/test/scala directories
- Additional tasks: compileScala, compileTestScala
- https://docs.gradle.org/current/userguide/scala_plugin.html





- A vastly different take on builds
- Aims for simplicity
- All build files are objects extending either `JavaModule` or `ScalaModule`
- <https://www.lihaoyi.com/mill/>



Demo

Thank You