and OSS but is maintained by Google, not JetBrains. If you're doing only Android development and want to adopt Google's release cycle, use Android Studio. If you're doing other types of development, be it JVM, Web, etc. then you're best bet is to use IntelliJ IDEA.

WebStorm, PyCharm, RubyMine, PhpStorm, AppCode or IntelliJ IDEA?

JetBrains provides specialized IDE's which are more focused around a specific language and/or technology. These are:

- **WebStorm**: For JavaScript and general Web
- **PyCharm**: Python, Django, etc.
- **RubyMine**: Ruby and Rails
- **PhpStorm**: PHP
- **AppCode**: Objective-C

All of them are based on the same platform: IntelliJ IDEA Platform. If you are doing JVM Development, you need IntelliJ IDEA. Otherwise pick the IDE that's most appropriate for you. All the functionality included in each of these individual IDE's (with the exception of AppCode), are also available in IntellIJ IDEA, some ship out of the box, others you can download as plugins (http://plugins.jetbrains.com).
The release cycles for IntelliJ IDEA and other IDE's vary though.

Project Formats

When working with IntelliJ IDEA you have various options on how you want to deal with projects. You can use IntelliJ IDEA's own format or use Maven/Gradle projects.

IntelliJ IDEA Projects

Modules and Libraries

An IntelliJ IDEA Project is similar to a Visual Studio Solution and can consist of modules and libraries:

- **Module**: A discrete unit of functionality that can be compiled, debugged, tested independently. Think of them as Projects in Visual Studio. Configuration information for a module is stored in a file with extensions *.iml*. Each module inside the project has its own *.iml* file.
- **Libraries**: An archive of compiled code (i.e. JAR) that a module can depend on.

A single IntellIJ project can consist of one or more modules and/or libraries.

Storage Format

IntelliJ Projects can be persisted in two ways:

- **Project File** (Legacy): Project information is stored in a file with extension *.ipr*.
- **Directory Based** (Recommended): Project information is stored in a folder called .idea in the main root of the project folder.

More information about projects and storage formats (http://www.jetbrains.com/idea/webhelp/project.html)

Maven Projects

IntelliJ IDEA can also work with Maven modules. You can either import existing Maven modules or create a new one.

Gradle Projects

IntelliJ IDEA can also work with Gradle projects. Much like Maven, you can also create a new one or import an existing one.

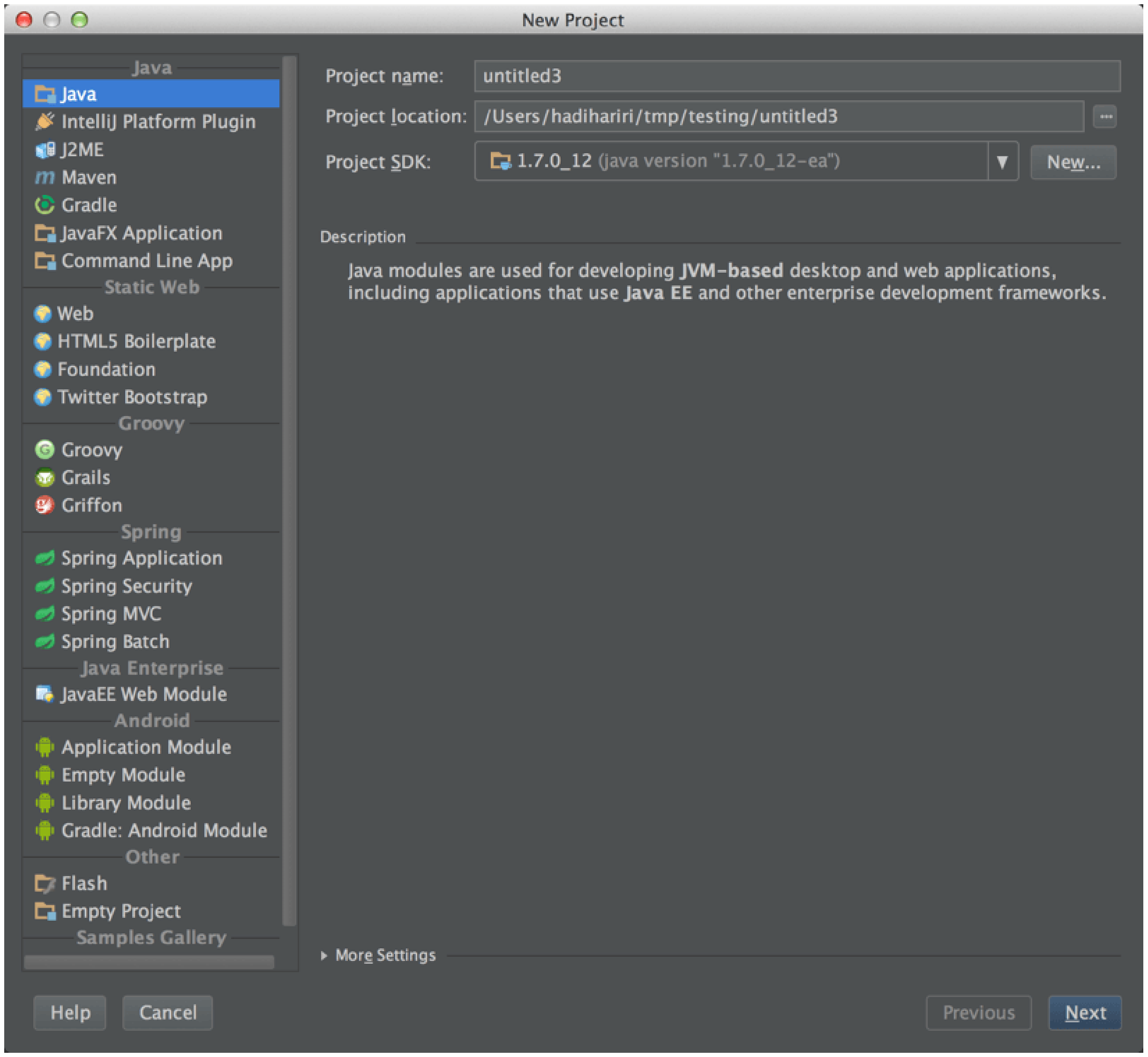
Files to check-in to Version Control

When working with Version Control, you should check in all files in the project folder, including the .idea folder (when using directory based approach), except for the *workspace.xml* and *tasks.xml* files. See Serge's explanation for more information (https://intellij-support.jetbrains.com/entries/23393067)

Creating a new Project

1. Click on **File** | **New Project**

2. Select the type of module. The most basic one is *Java*, independently of the language you're going to use. Remember, IntelliJ IDEA is multi-language so in essence you can have multiple languages in the same module. The other options are technology specific. So for instance, Android will create an Android application, Spring will create a Spring application, et al.



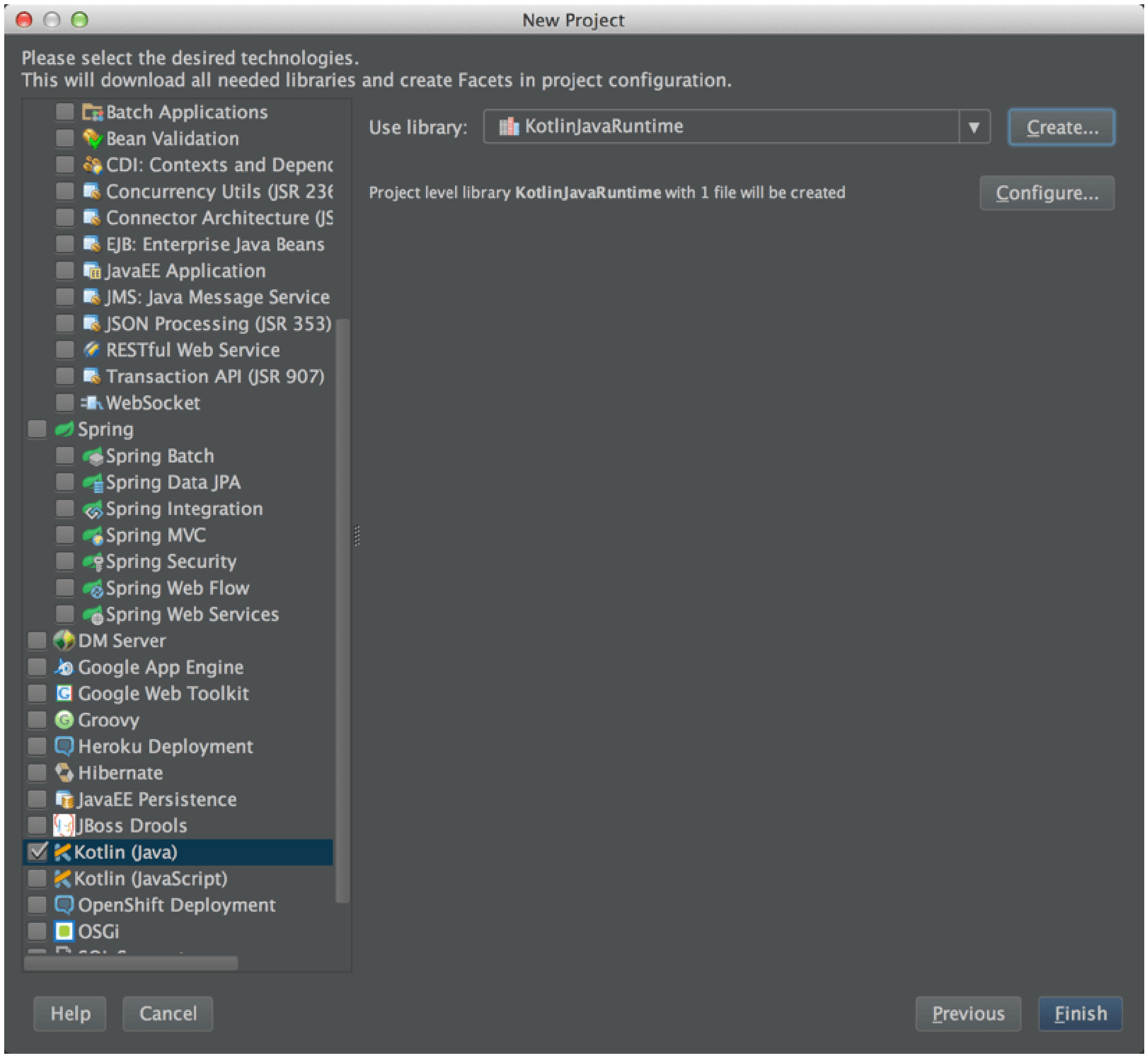
Project name: untitled3
Project location: /Users/hadihariri/tmp/testing/untitled3
Project SDK: 1.7.0_12 (java version "1.7.0_12-ea")

Description

Java modules are used for developing **JVM-based** desktop and web applications, including applications that use **Java EE** and other enterprise development frameworks.

3. Select the SDK. Usually you'd pick a JDK here (or Android SDK). This is equivalent to selecting the target .NET framework in Visual Studio. If you don't have anything available in the dropdown, it's because you don't have a JDK IntelliJ IDEA knows about. Click New and add one, which means: point to the installation folder of the JDK. You can also add other types of SDK's such as Android or IntelliJ IDEA Plugin SDK.

4. On clicking *Next*, you have the chance to pick the technologies you want included in the module. This means ultimately adding library references or support for specific languages. For instance, when working with Kotlin, I pick Kotlin at this stage (albeit IntelliJ IDEA can also detect new Kotlin files and add support automatically). You can pick things like Java EE support, Heroku Development, etc.


New Project

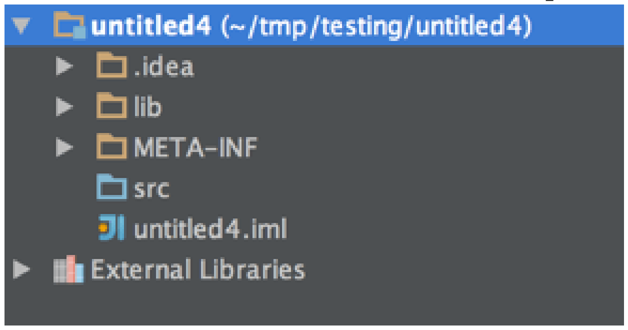
Please select the desired technologies.
This will download all needed libraries and create Facets in project configuration.

Kotlin (Java) ✓

Use library: KotlinJavaRuntime

Project level library **KotlinJavaRuntime** with 1 file will be created

5. Click Finish. You should now end up with a new Project and module



Adding new Modules to existing projects

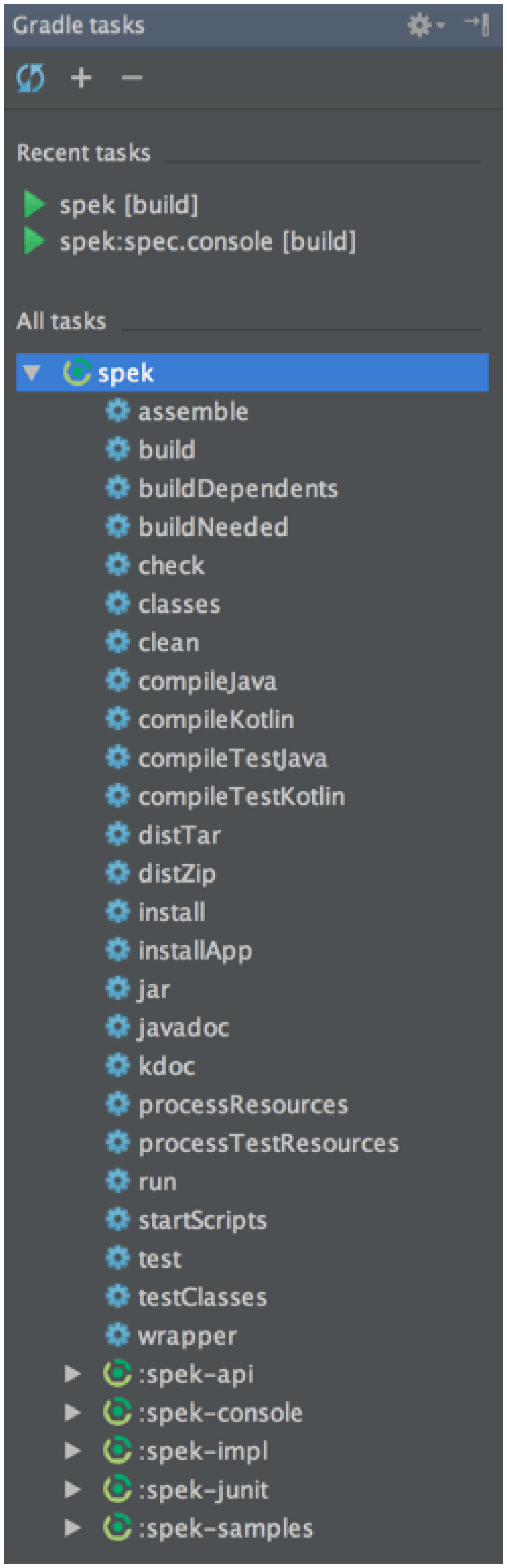
If you want to add a second module (think C#/VB project) to the project:

1. Click on **File** | **New Module**

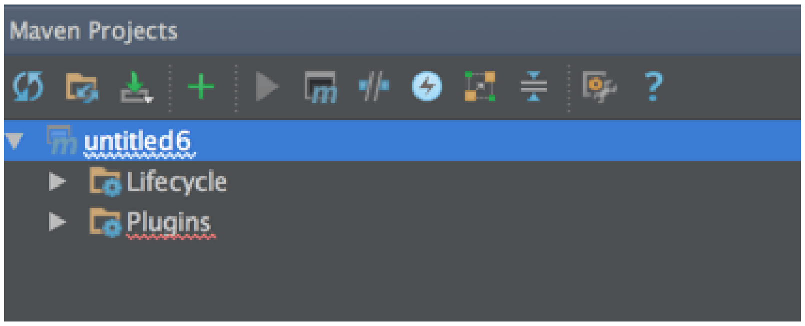
2. Repeat steps 2 to 5 from Creating a new Project

Working with Maven and Gradle Projects

You can create and import existing Maven or Gradle projects into IntelliJ IDEA. If it’s a new project, select Maven/Gradle in Step 2 of Creating a new Project. If it’s an existing project, select **File | Import Module** or **Import Module** from the main welcome window and point to the corresponding *pom.xml* file for Maven or *build.gradle* for Gradle.
Once you do this, IntelliJ IDEA will automatically provide you with a tool window to handle Maven/Gradle.


Gradle tasks
Recent tasks
spek [build]
spek:spec.console [build]
All tasks
spek
assemble
build
buildDependents
buildNeeded
check
classes
clean
compileJava
compileKotlin
compileTestJava
compileTestKotlin
distTar
distZip
install
installApp
jar
javadoc
kdoc
processResources
processTestResources
run
startScripts
test
testClasses
wrapper
:spek-api
:spek-console
:spek-impl
:spek-junit
:spek-samples

Gradle



Maven

Sharing Projects with Eclipse

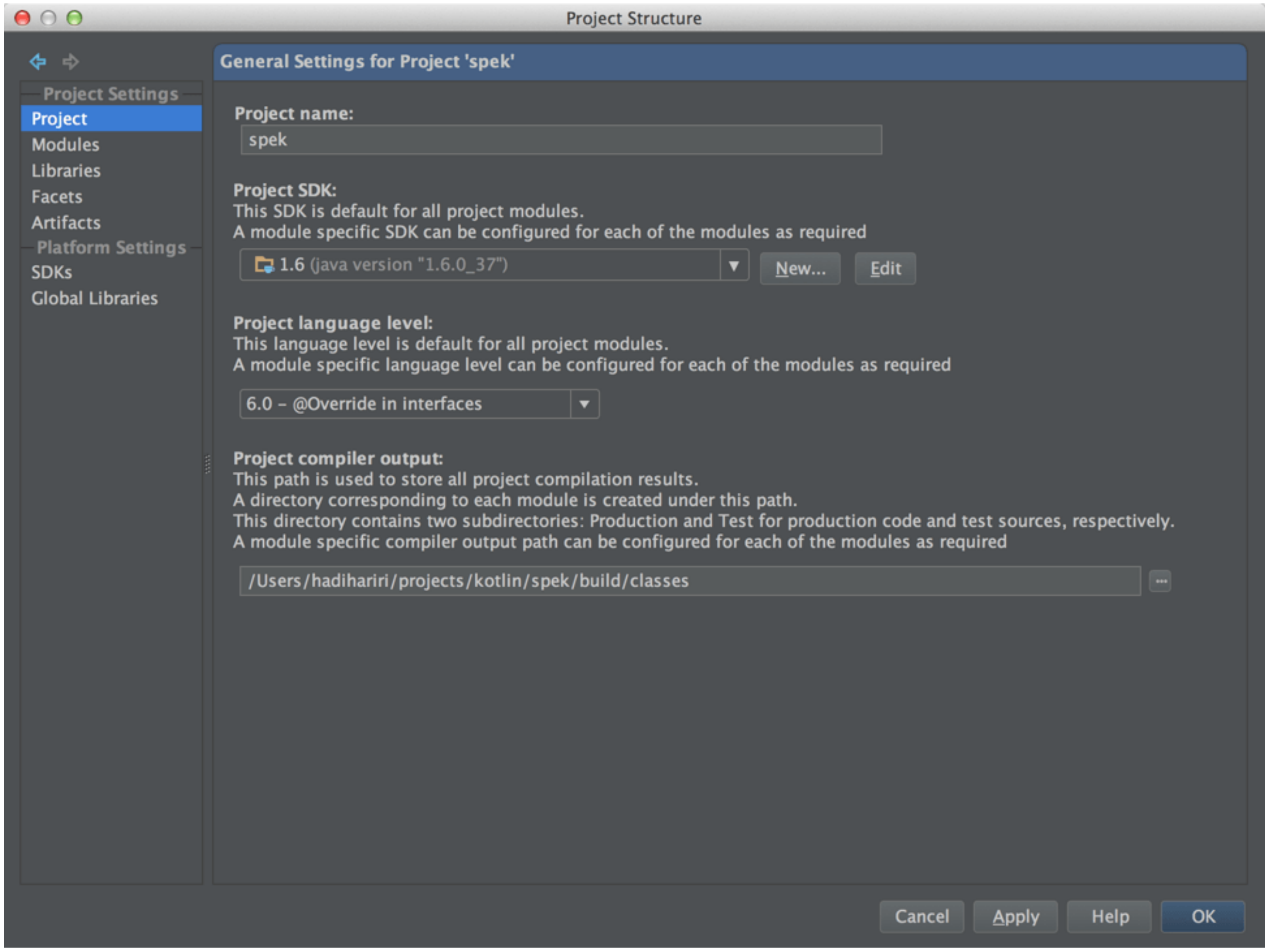
If some members of your team are using Eclipse, you can still use IntelliJ IDEA and have two-sync. See the Eclipse FAQ for IntelliJ IDEA users (http://www.jetbrains.com/idea/documentation/migration_faq.html).

Project Settings

This one can be a bit overwhelming when you first open it.

Project Setting Tab

This contains information global to all modules, that is the project name, SDK and if applicable language level. In addition you can set the output compilation folder.


Project Structure

Project Settings
- Project
- Modules
- Libraries
- Facets
- Artifacts

Platform Settings
- SDKs
- Global Libraries

General Settings for Project 'spek'

Project name: spek

Project SDK:
This SDK is default for all project modules.
A module specific SDK can be configured for each of the modules as required

1.6 (java version "1.6.0_37")

Project language level:
This language level is default for all project modules.
A module specific language level can be configured for each of the modules as required

6.0 - @Override in interfaces

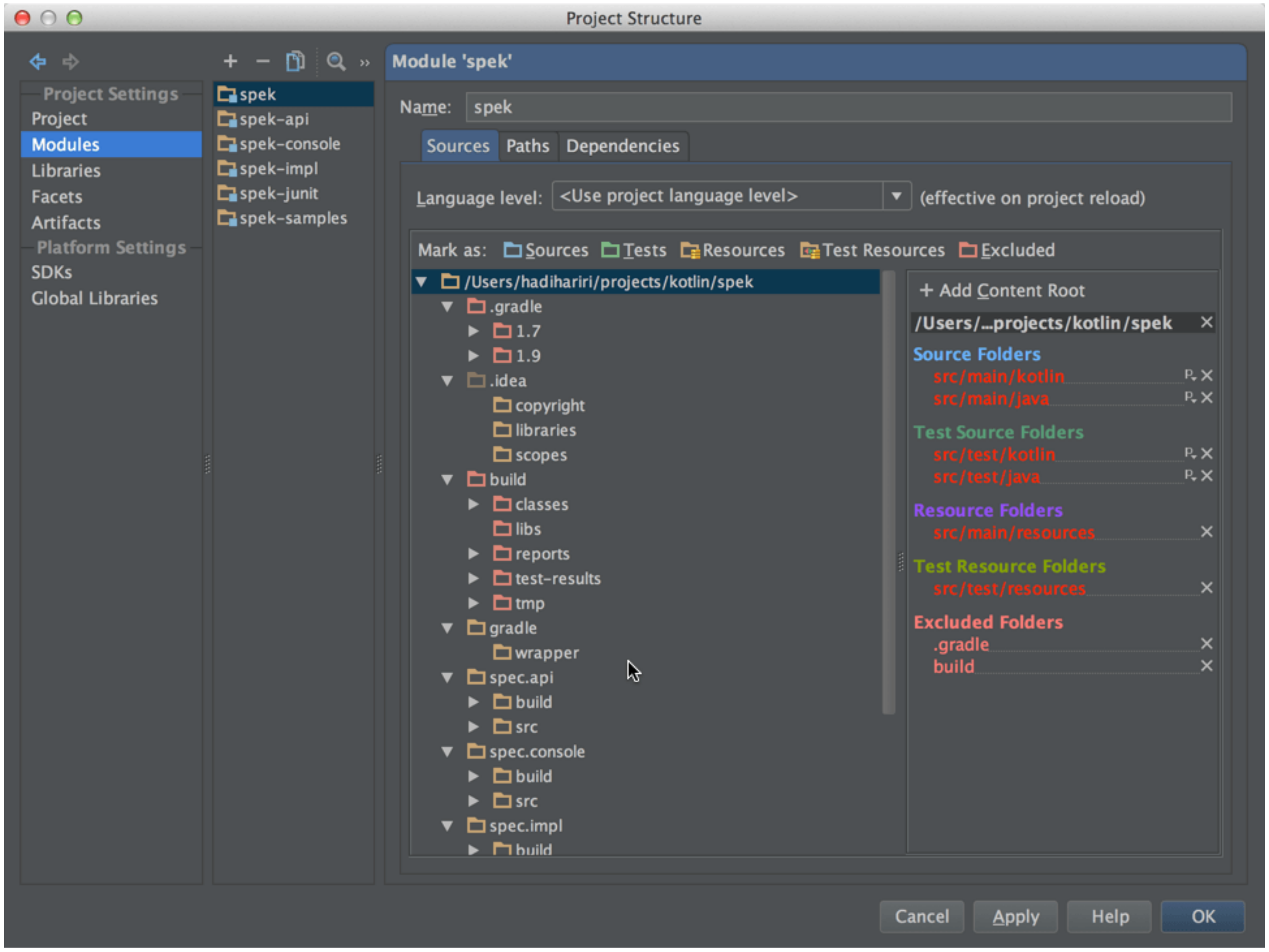
Project compiler output:
This path is used to store all project compilation results.
A directory corresponding to each module is created under this path.
This directory contains two subdirectories: Production and Test for production code and test sources, respectively.
A module specific compiler output path can be configured for each of the modules as required

/Users/hadihariri/projects/kotlin/spek/build/classes

- **SDK**: As mention previously, this is like the .NET Framework we’re targeting.
- **Language Level**: In IntelliJ IDEA you can pick the language version you want to target. This is different to Visual Studio whereby each version has its own associated version of the compiler.

Module Entries

Each module has its own entry with its own settings. Clicking on Modules we can see each entry on the right pane.


Project Structure

Module 'spek'

Name: spek

Language level: <Use project language level> (effective on project reload)

/Users/hadihariri/projects/kotlin/spek

/Users/...projects/kotlin/spek

Source Folders
src/main/kotlin
src/main/java

Test Source Folders
src/test/kotlin
src/test/java

Resource Folders
src/main/resources

Test Resource Folders
src/test/resources

Excluded Folders
.gradle
build

The buttons at the top (+/-/Copy) allow you to add/remove and copy modules directly from this page.

Sources

There are some folders in IntelliJ IDEA that have a special meaning, namely

- Source Folders
- Test Folders
- Resources
- Test Resources

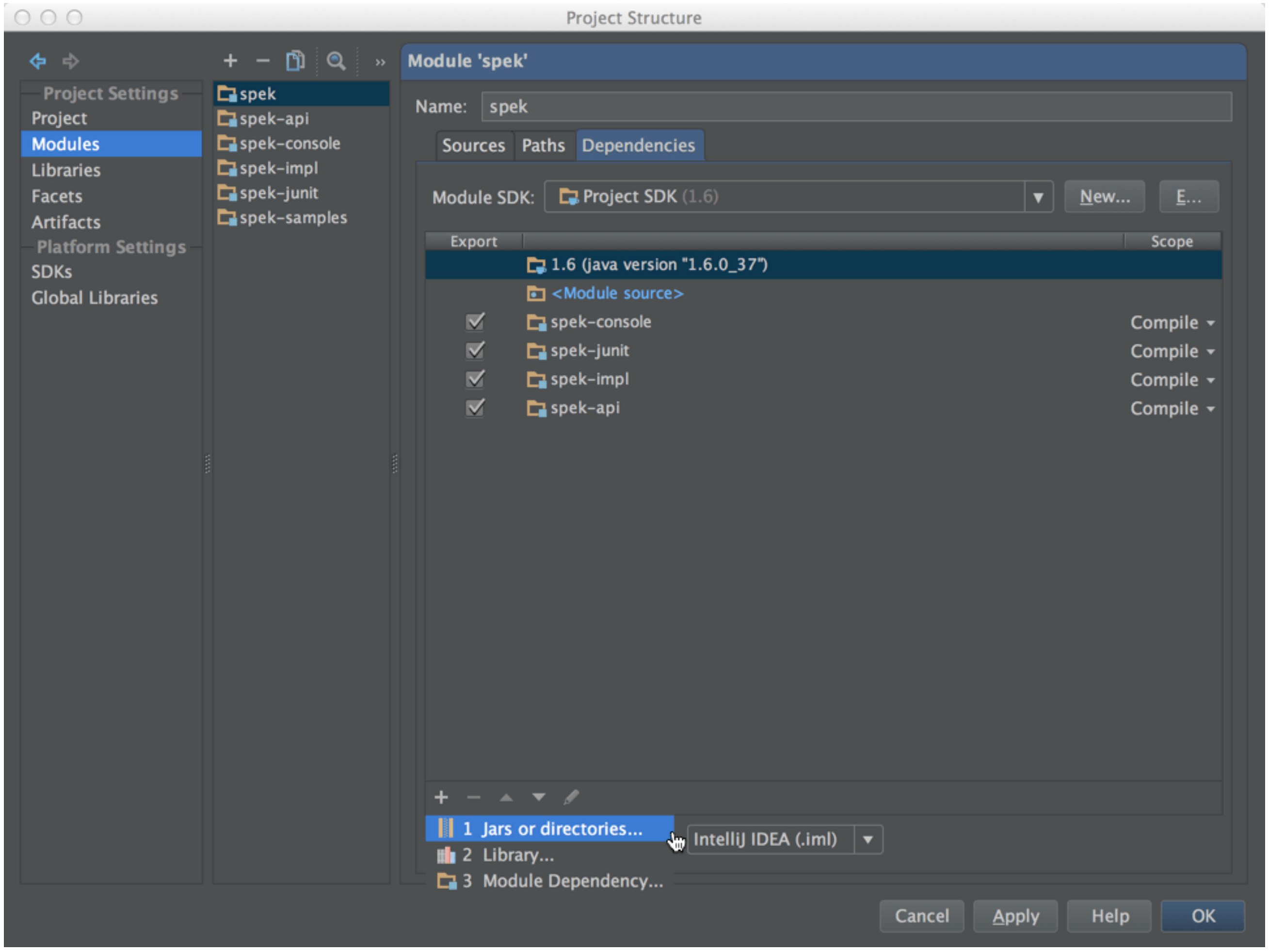
then there's Excluded Folders.
This is where you can mark what is what. Right-click and select. You can also do this from the Project Window.
Usually when creating a new project, *src* is automatically marked as a *src* folder. Folders such as *out* are excluded.
See Sources Tab for more detailed information (http://www.jetbrains.com/idea/webhelp/sources-tab.html)

Paths

This one is pretty straight-forward. You can override the default project settings and specify different paths. You can also manage JavaDoc lookups and Annotations (http://www.jetbrains.com/idea/webhelp/paths-tab.html).

Dependencies

This is where you specify your module dependencies (i.e. other things you rely on). A dependency can be:



Name: spek

Module SDK: Project SDK (1.6)

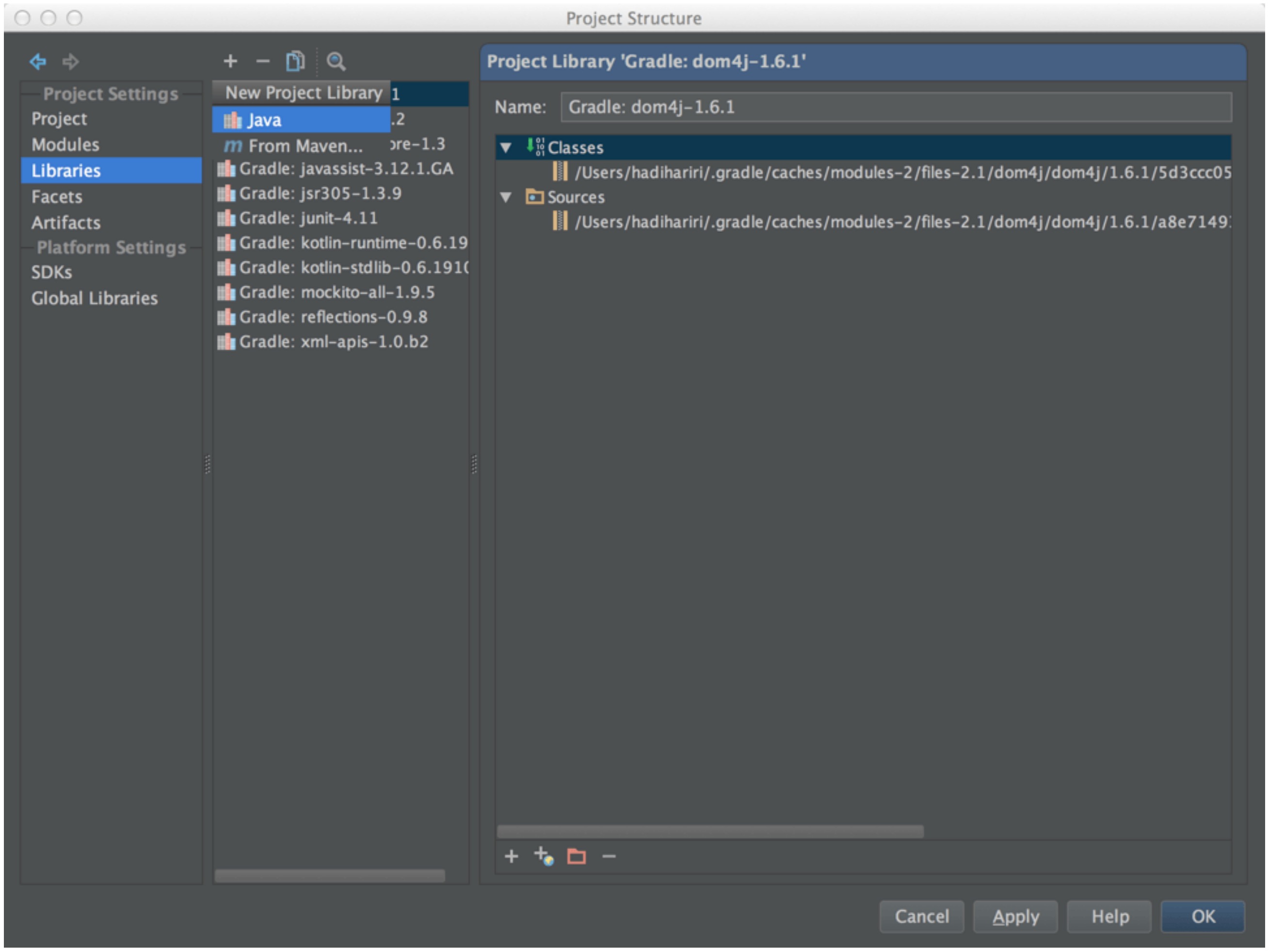
Export		Scope
	1.6 (java version "1.6.0_37")	
	<Module source>	
✓	spek-console	Compile
✓	spek-junit	Compile
✓	spek-impl	Compile
✓	spek-api	Compile

- A JAR or directory containing classes
- A Library
- Another module in the project

Wondering about Maven dependencies? Wait till we get to Libraries, which is next!

Libraries

This is where you can reference external modules. IntelliJ IDEA defines these as libraries. A library can be a Maven library or a JAR directory with its class files, sources and docs.


Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
New Project Library
Java
From Maven...
Gradle: javassist-3.12.1.GA
Gradle: jsr305-1.3.9
Gradle: junit-4.11
Gradle: kotlin-runtime-0.6.19
Gradle: kotlin-stdlib-0.6.191(
Gradle: mockito-all-1.9.5
Gradle: reflections-0.9.8
Gradle: xml-apis-1.0.b2
Project Library 'Gradle: dom4j-1.6.1'
Name: Gradle: dom4j-1.6.1
Classes
/Users/hadihariri/.gradle/caches/modules-2/files-2.1/dom4j/dom4j/1.6.1/5d3ccc05
Sources
/Users/hadihariri/.gradle/caches/modules-2/files-2.1/dom4j/dom4j/1.6.1/a8e7149
Cancel
Apply
Help
OK

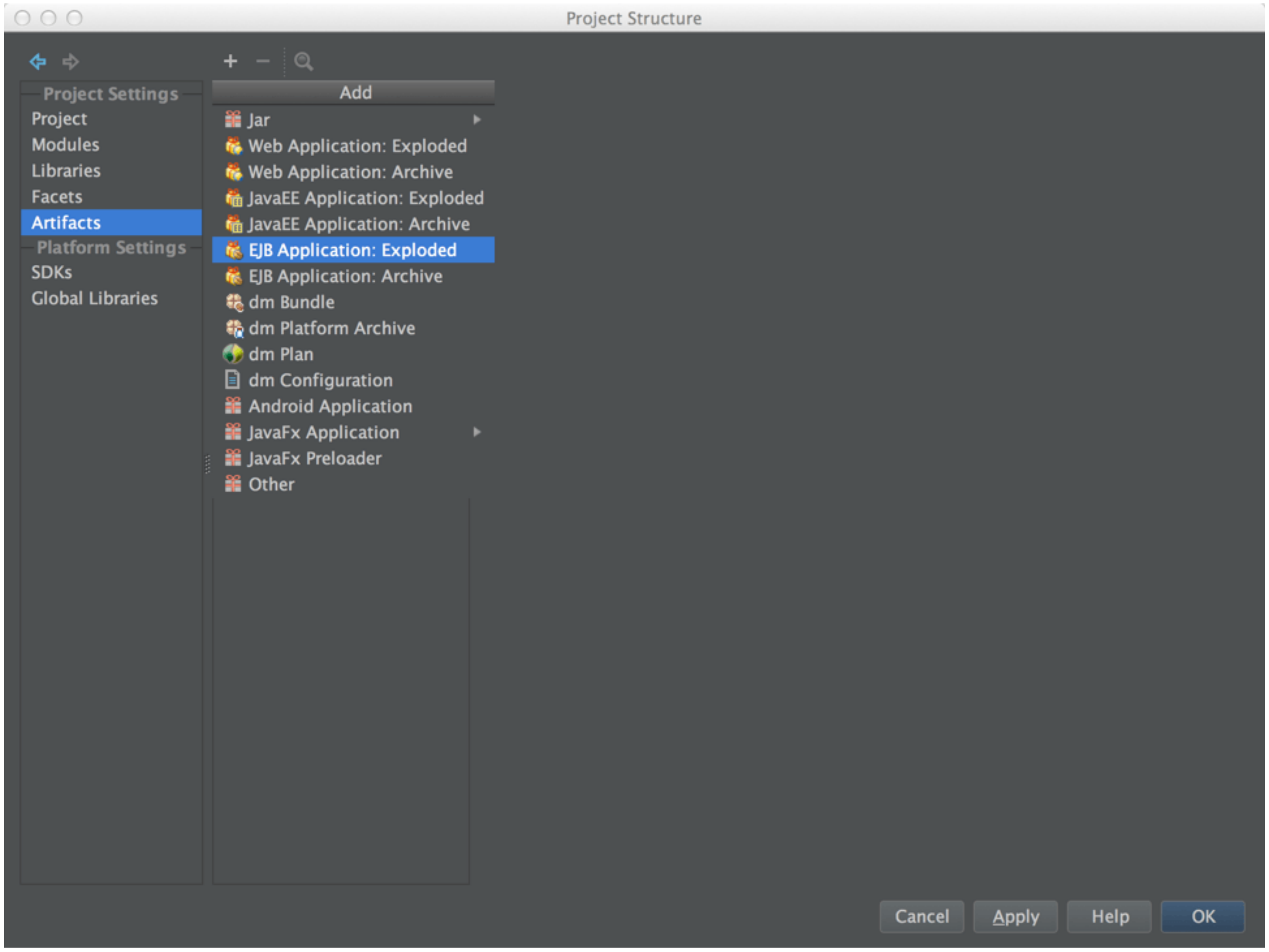
You can also add additional directories (or remove them) from existing libraries (right pane).
See Configuring Module Dependencies and Libraries (http://www.jetbrains.com/idea/webhelp/configuring-module-dependencies-and-libraries.html).

Facets

Frameworks such as Spring require certain configurations, libraries, dependencies and technologies. A Facet is a way to group all this into a manageable component that is then used by IntelliJ IDEA. You can add Facets here. For instance, when creating a Spring module with a new project, it adds the Spring Facet.
For more information about Facets see the documentation (http://www.jetbrains.com/idea/webhelp/facet.html)

Artifacts

Artifacts, or otherwise known as output generated by your project. This is where you tell IntelliJ IDEA to create JAR's, WAR's and other types of artifacts for you. Once you define them you can then access them from the **Build** menu.



SDK’s

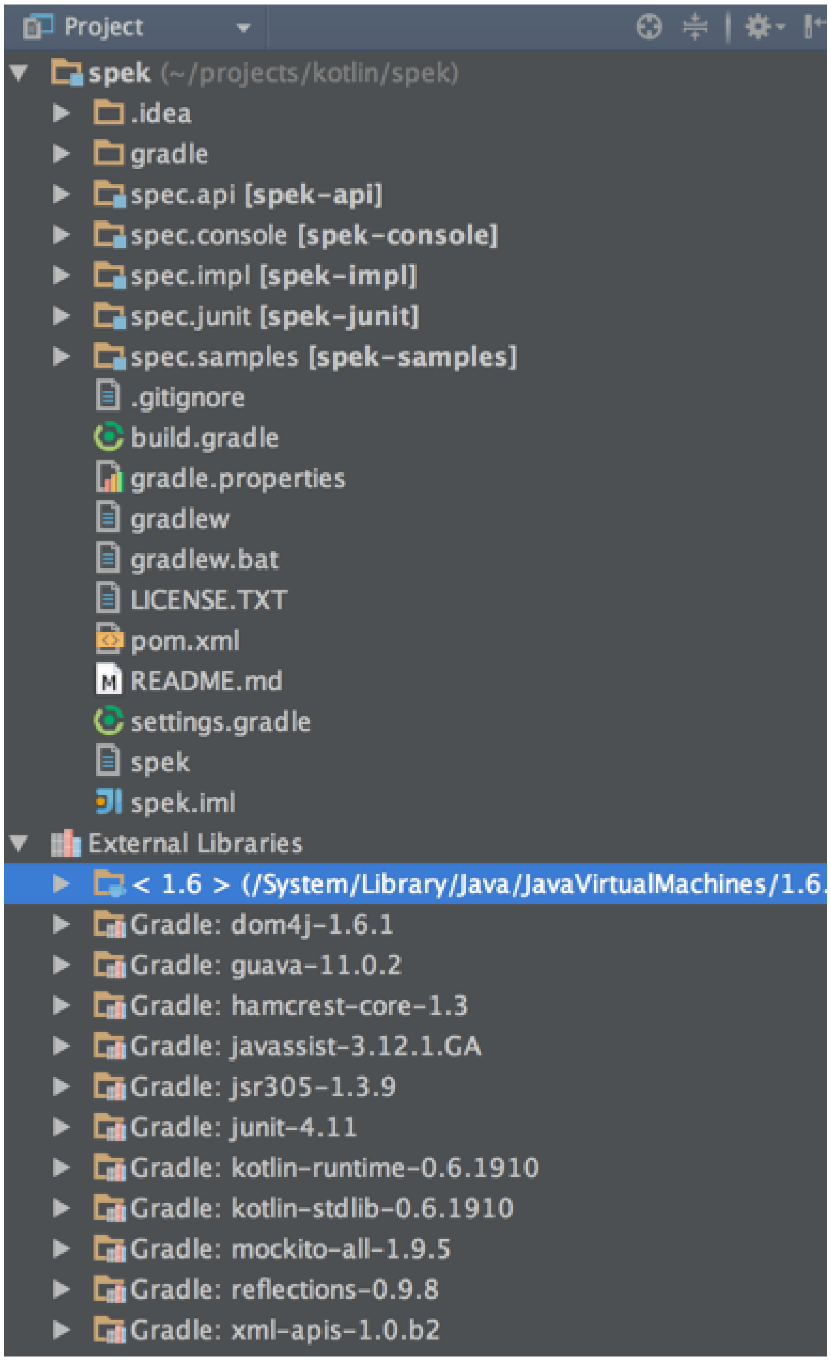
These are globally available SDK’s for IntelliJ IDEA. Add and remove at will. Obviously don’t remove those required!

Global Libraries

Libraries as we saw are libraries you can use in your project, but they’re only available to your project. If you want libraries to be available globally for all projects, define them here.

External Libraries

In the Project Window you always see an entry named *External Libraries*. These are libraries that are not internal to your project (i.e. other modules).



```
Project
▼ spek (~/projects/kotlin/spek)
    ▶ .idea
    ▶ gradle
    ▶ spec.api [spek-api]
    ▶ spec.console [spek-console]
    ▶ spec.impl [spek-impl]
    ▶ spec.junit [spek-junit]
    ▶ spec.samples [spek-samples]
      .gitignore
      build.gradle
      gradle.properties
      gradlew
      gradlew.bat
      LICENSE.TXT
      pom.xml
      README.md
      settings.gradle
      spek
      spek.iml
▼ External Libraries
    ▶ < 1.6 > (/System/Library/Java/JavaVirtualMachines/1.6.
    ▶ Gradle: dom4j-1.6.1
    ▶ Gradle: guava-11.0.2
    ▶ Gradle: hamcrest-core-1.3
    ▶ Gradle: javassist-3.12.1.GA
    ▶ Gradle: jsr305-1.3.9
    ▶ Gradle: junit-4.11
    ▶ Gradle: kotlin-runtime-0.6.1910
    ▶ Gradle: kotlin-stdlib-0.6.1910
    ▶ Gradle: mockito-all-1.9.5
    ▶ Gradle: reflections-0.9.8
    ▶ Gradle: xml-apis-1.0.b2
```

Building Projects

IntelliJ IDEA is aware of the build system you use, to the point that if you're using Ant, Maven or Gradle you can actually have these invoke project builds. In fact, with Maven and Gradle, your project is continuously being built when changes occur.
In the case of Ant, Maven or Gradle, you can use the corresponding tool windows
However, there's also a *Build* menu that has a few options:

- **Compile**: Compiles files in the selected scope, whether they've changed or not
- **Make Module**: Compiles module, and only recompiles files if they have changed
- **Make Project**: Same as Make Module but for the entire project.

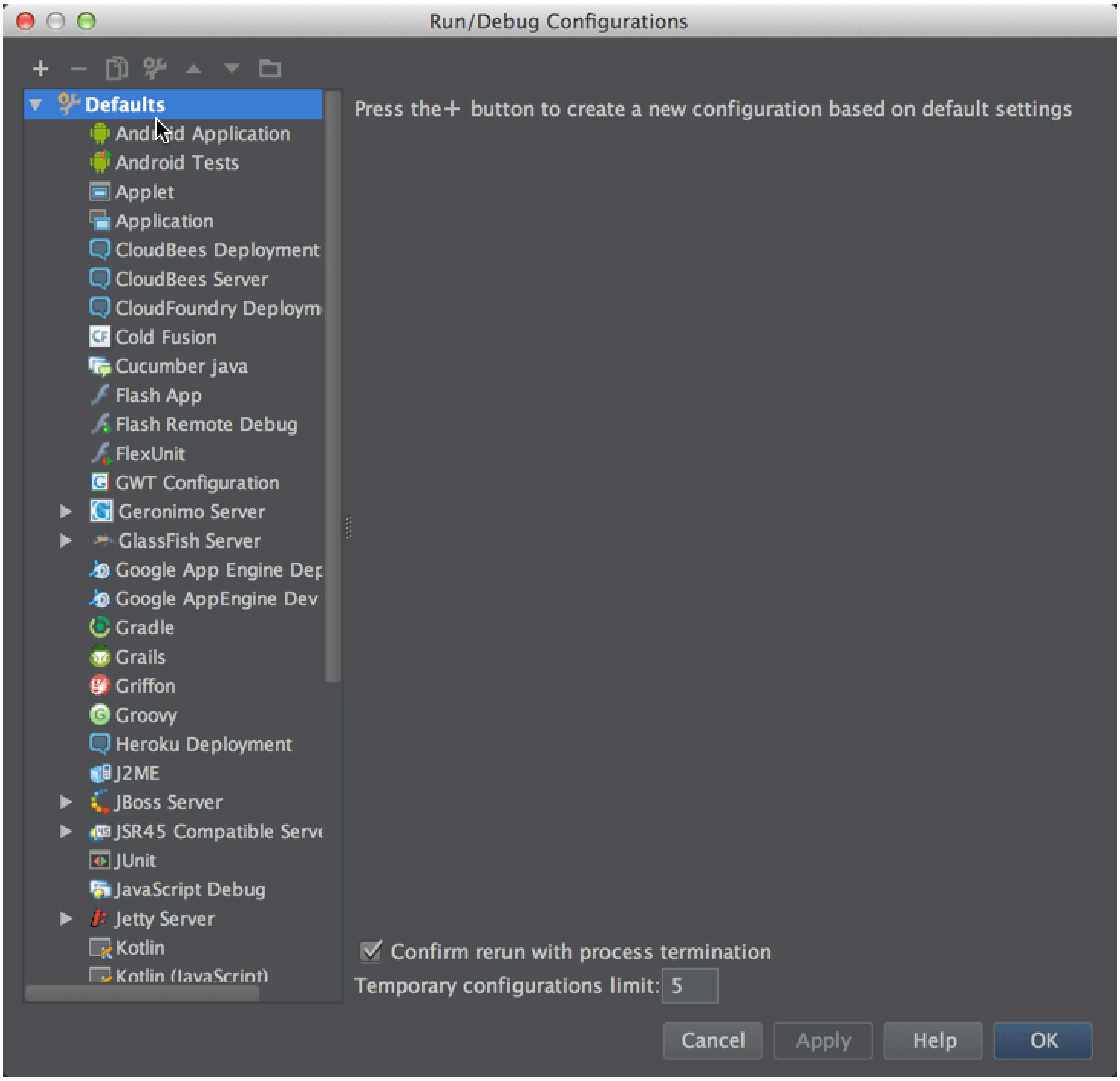
- **Rebuild Project**: Forces a recompile of the entire project.

For more details see documentation (http://www.jetbrains.com/idea/webhelp/compilation-types.html) Note: these still work even if you’re using Ant, Maven or Gradle.
I usually use Make Project (Cmd+F9 on OSX).

Running Applications

Run Configurations

IntelliJ IDEA has the concept of Run Configurations (http://www.jetbrains.com/idea/webhelp/running.html). What this means is that before you can run an app, you need to create a run configuration for it. Fortunately this is quite easy. In fact with certain types of applications, such as command line apps, you can just right-click on the main class and hit Run. IntelliJ IDEA will automatically create a run configuration for you.
Alternatively, click on **Run | Edit Configurations** to create a new entry


Run/Debug Configurations

Press the + button to create a new configuration based on default settings

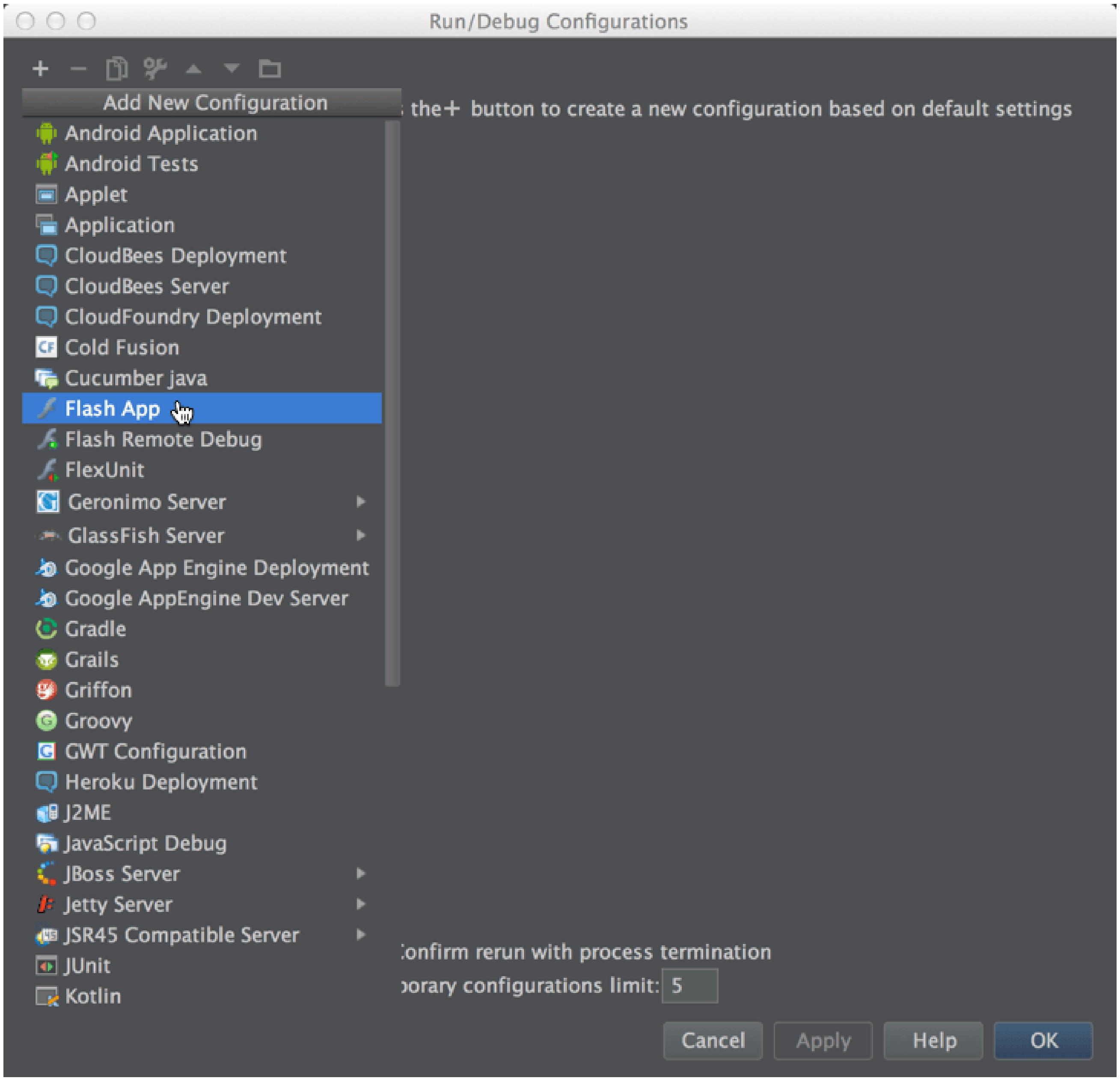
- Defaults
  - Android Application
  - Android Tests
  - Applet
  - Application
  - CloudBees Deployment
  - CloudBees Server
  - CloudFoundry Deploym
  - Cold Fusion
  - Cucumber java
  - Flash App
  - Flash Remote Debug
  - FlexUnit
  - GWT Configuration
  - Geronimo Server
  - GlassFish Server
  - Google App Engine Dep
  - Google AppEngine Dev
  - Gradle
  - Grails
  - Griffon
  - Groovy
  - Heroku Deployment
  - J2ME
  - JBoss Server
  - JSR45 Compatible Serv
  - JUnit
  - JavaScript Debug
  - Jetty Server
  - Kotlin
  - Kotlin (JavaScript)

☑ Confirm rerun with process termination

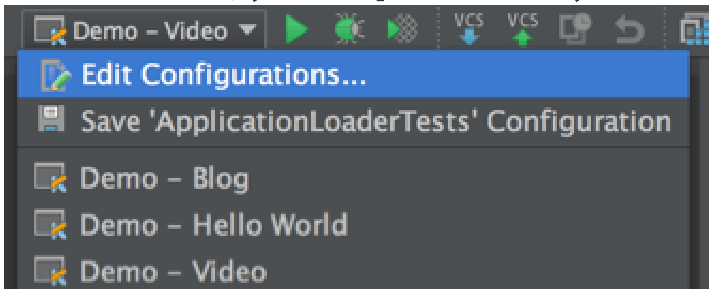
Temporary configurations limit: 5

Do not modify the defaults

The entries you see are defaults. You don’t usually modify these. What you need to do is add a new configuration:



Once you have a Run Configuration, the Run button will be enabled. If you have more than one, you can pick the one you want from the dropdown



Debugging Applications

Just like Run Configurations, IntelliJ IDEA also provides configurations for debugging. Same principle, except you provide parameters required for debugging. See Creating and Editing Run/Debug Configurations (http://www.jetbrains.com/idea/webhelp/creating-and-editing-run-debug-configurations.html) for more information.

Much like any decent IDE, IntelliJ IDEA provides a full debugger with breakpoints, conditional breakpoints, watches, evaluations, etc. In addition it provides JavaScript debugging too (and for most languages that have a plugin available)
See Debugging (http://www.jetbrains.com/idea/webhelp/debugging.html) for more information.

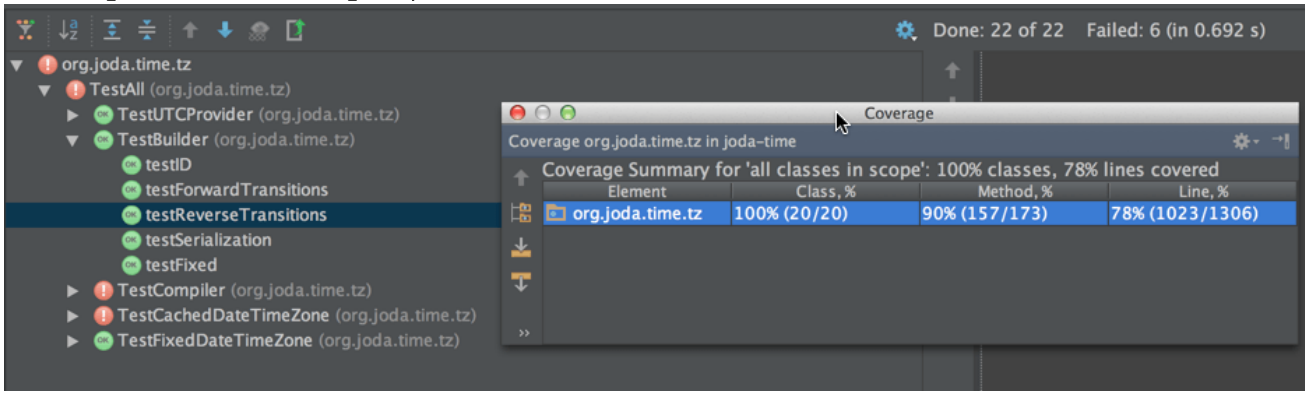
Temporary Configurations

In case you were wondering, yes, you can have temporary configurations too (http://www.jetbrains.com/idea/webhelp/creating-and-saving-temporary-run-debug-configurations.html)

Unit Testing

Running, Debugging and Code Coverage

IntelliJ IDEA ships with support for JUnit, TestNG, Groovy JUnit, FlexUnit, JSTestDriver, Jasmine out of the box, and allows you to run, debug and get code coverage for tests using any of these frameworks.


Done: 22 of 22 Failed: 6 (in 0.692 s)

```
▼ org.joda.time.tz
  ▼ TestAll (org.joda.time.tz)
    ▶ TestUTCProvider (org.joda.time.tz)
    ▼ TestBuilder (org.joda.time.tz)
        testID
        testForwardTransitions
        testReverseTransitions
        testSerialization
        testFixed
    ▶ TestCompiler (org.joda.time.tz)
    ▶ TestCachedDateTimeZone (org.joda.time.tz)
    ▶ TestFixedDateTimeZone (org.joda.time.tz)
```

Coverage

Coverage org.joda.time.tz in joda-time

Coverage Summary for 'all classes in scope': 100% classes, 78% lines covered

Element	Class, %	Method, %	Line, %
org.joda.time.tz	100% (20/20)	90% (157/173)	78% (1023/1306)

Similar to ReSharper Unit Test Runner, you can also create test sessions, run the current tests, run a scope of tests, etc.
See the Unit Testing and Code Coverage (http://confluence.jetbrains.com/display/IntelliJIDEA/Unit+Testing+and+Code+Coverage) tutorial for more details

Integrating with TeamCity

// TODO

Completion

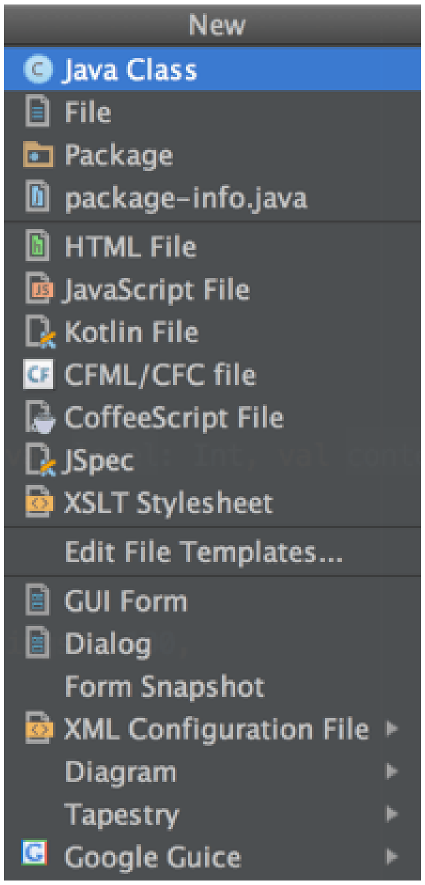
IntelliJ IDEA offers code completion, making you type less and providing you with insight into API's.
Check out the Completion Tutorial for more information (http://confluence.jetbrains.com/display/IntelliJIDEA/Code+Completion)

Documentation and Parameter Info

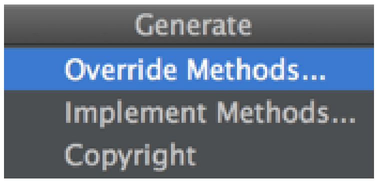
IntelliJ IDEA supports in-place documentation and parameter information (http://confluence.jetbrains.com/display/IntelliJIDEA/Quick+Popups)

Code Generation

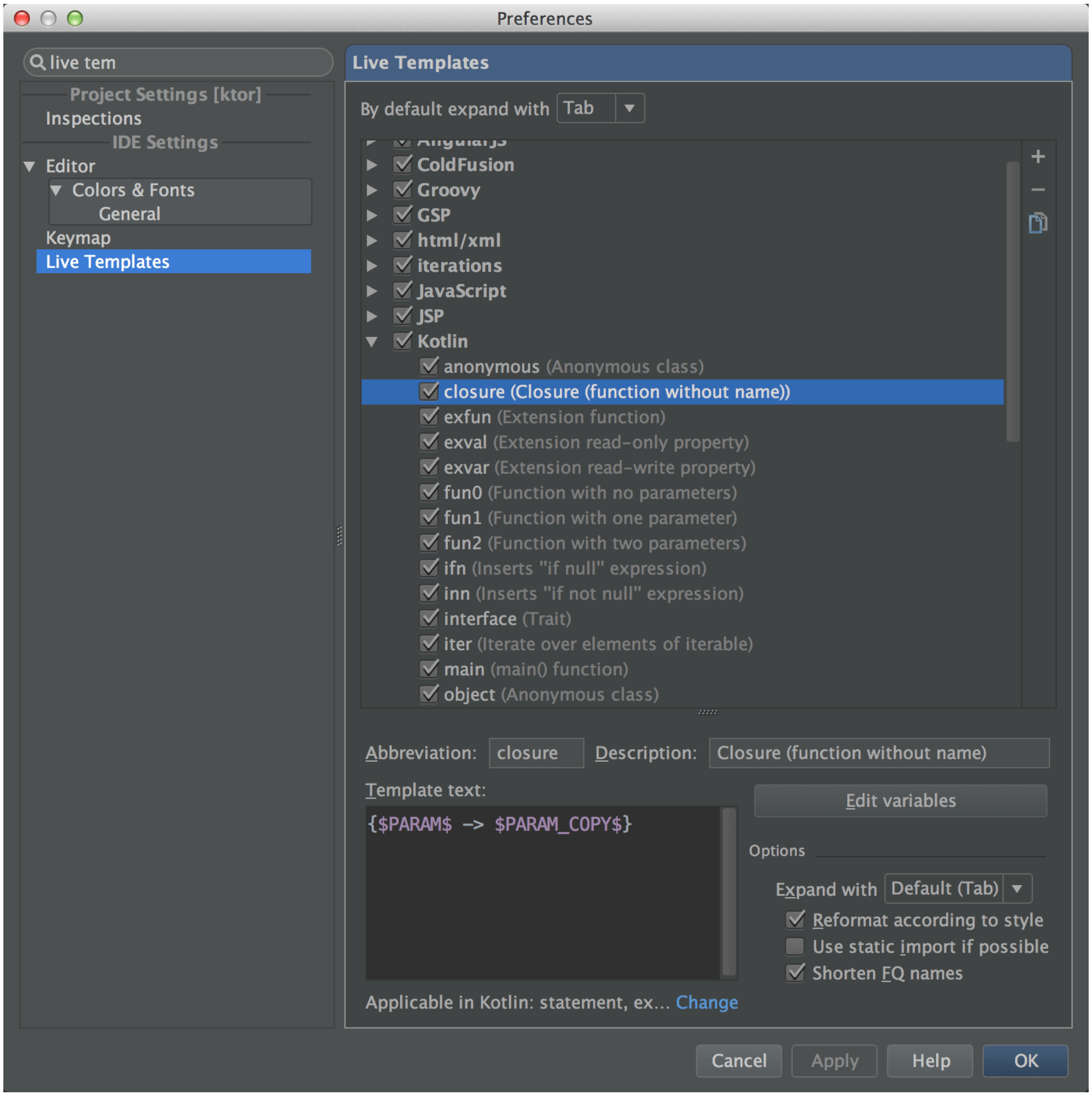
File Templates are the fastest way to create new files (Cmd+Ctrl+N on OSX when inside a file or Cmd+N when in Project Window).



Cmd+N on a class allows you to override methods, implement methods, etc. It's context sensitive



Along with File templates, you also have Live Templates which are interactive snippets. You can access them via **Preferences | Live Templates**



Live Templates

By default expand with: Tab

- AngularJS
- ColdFusion
- Groovy
- GSP
- html/xml
- iterations
- JavaScript
- JSP
- Kotlin
  - anonymous (Anonymous class)
  - closure (Closure (function without name))
  - exfun (Extension function)
  - exval (Extension read-only property)
  - exvar (Extension read-write property)
  - fun0 (Function with no parameters)
  - fun1 (Function with one parameter)
  - fun2 (Function with two parameters)
  - ifn (Inserts "if null" expression)
  - inn (Inserts "if not null" expression)
  - interface (Trait)
  - iter (Iterate over elements of iterable)
  - main (main() function)
  - object (Anonymous class)

Abbreviation: closure Description: Closure (function without name)

Template text:

```
{$PARAM$ -> $PARAM_COPY$}
```

Edit variables

Options

Expand with: Default (Tab)

- [✓] Reformat according to style
- [ ] Use static import if possible
- [✓] Shorten FQ names

Applicable in Kotlin: statement, ex... Change

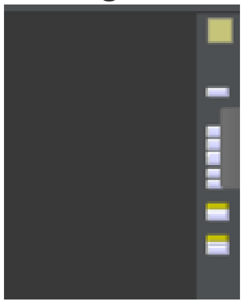
Navigation

Very rich navigation and pretty much aligned feature-wise with ReSharper. If there's one key to remember that's Double-Shift (IntelliJ IDEA 13+): *Search Everywhere* Check out the Navigation tutorial (http://confluence.jetbrains.com/display/IntelliJIDEA/Navigation) for more information.

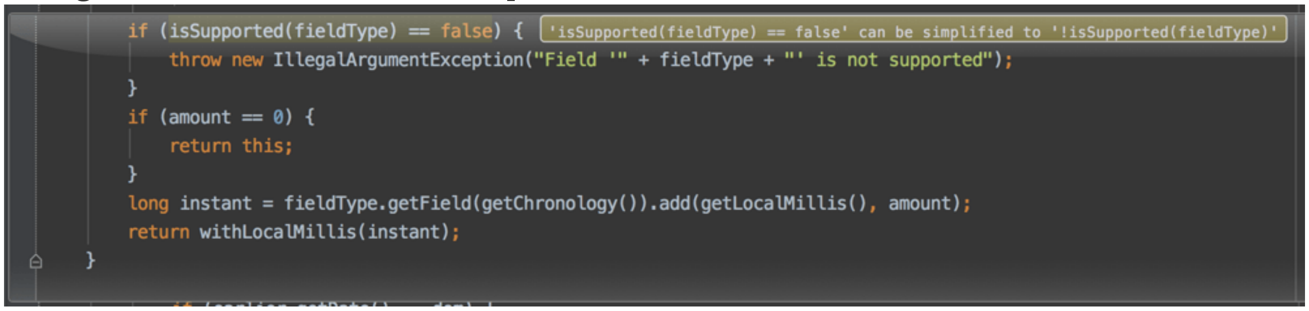
Code Analysis

IntelliJ IDEA sees Dead Code, a.k.a. the famous T-Shirt (http://www.ptxstore.com/jetbrains/product_info.php?products_id=442&osCsid=6ab235f88dd658808603e500d77331f1). It detects issues

in your code (via Inspections) and offers you quick fixes (Intentions). That's the famous Alt+Enter menu shortcut. Which btw, has its sister ReSharper T-Shirt (http://www.ptxstore.com/jetbrains/product_info.php?products_id=1870).
The Red marks you see in the gutter are all these inspections that IntelliJ IDEA does. Don't get scared. If you don't agree, you can turn them off.

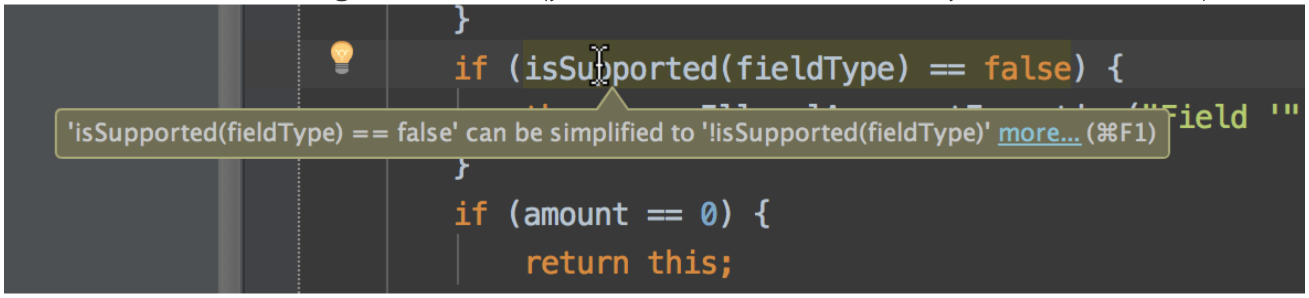


If the top square is Green, you're all good. If it's Yellow, there are some warnings and hints. If Red, you got a problem!
Navigate over the marks to see a preview



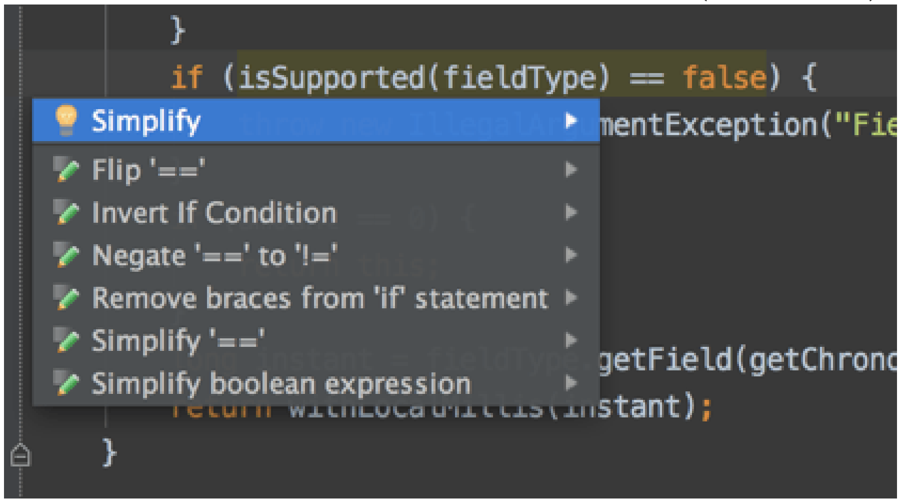
```
if (isSupported(fieldType) == false) {   'isSupported(fieldType) == false' can be simplified to '!isSupported(fieldType)'
    throw new IllegalArgumentException("Field '" + fieldType + "' is not supported");
}
if (amount == 0) {
    return this;
}
long instant = fieldType.getField(getChronology()).add(getLocalMillis(), amount);
return withLocalMillis(instant);
}
```

Click on them to navigate to them (you can also do this via keyboard shortcuts):



```
}
if (isSupported(fieldType) == false) {
'isSupported(fieldType) == false' can be simplified to '!isSupported(fieldType)' more... (⌘F1)
}
if (amount == 0) {
    return this;
```

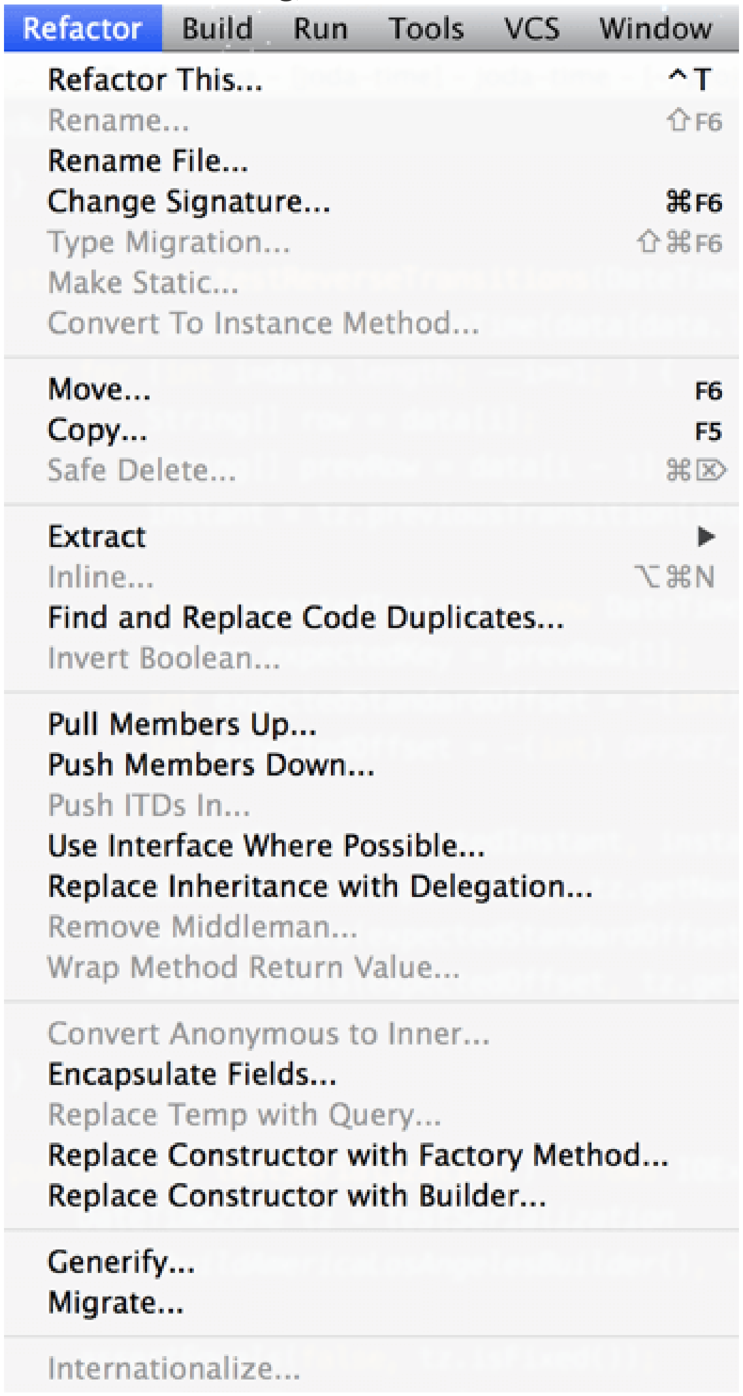
Hit Alt+Enter to see what IntelliJ IDEA offers (Intentions):



```
}
if (isSupported(fieldType) == false) {
```

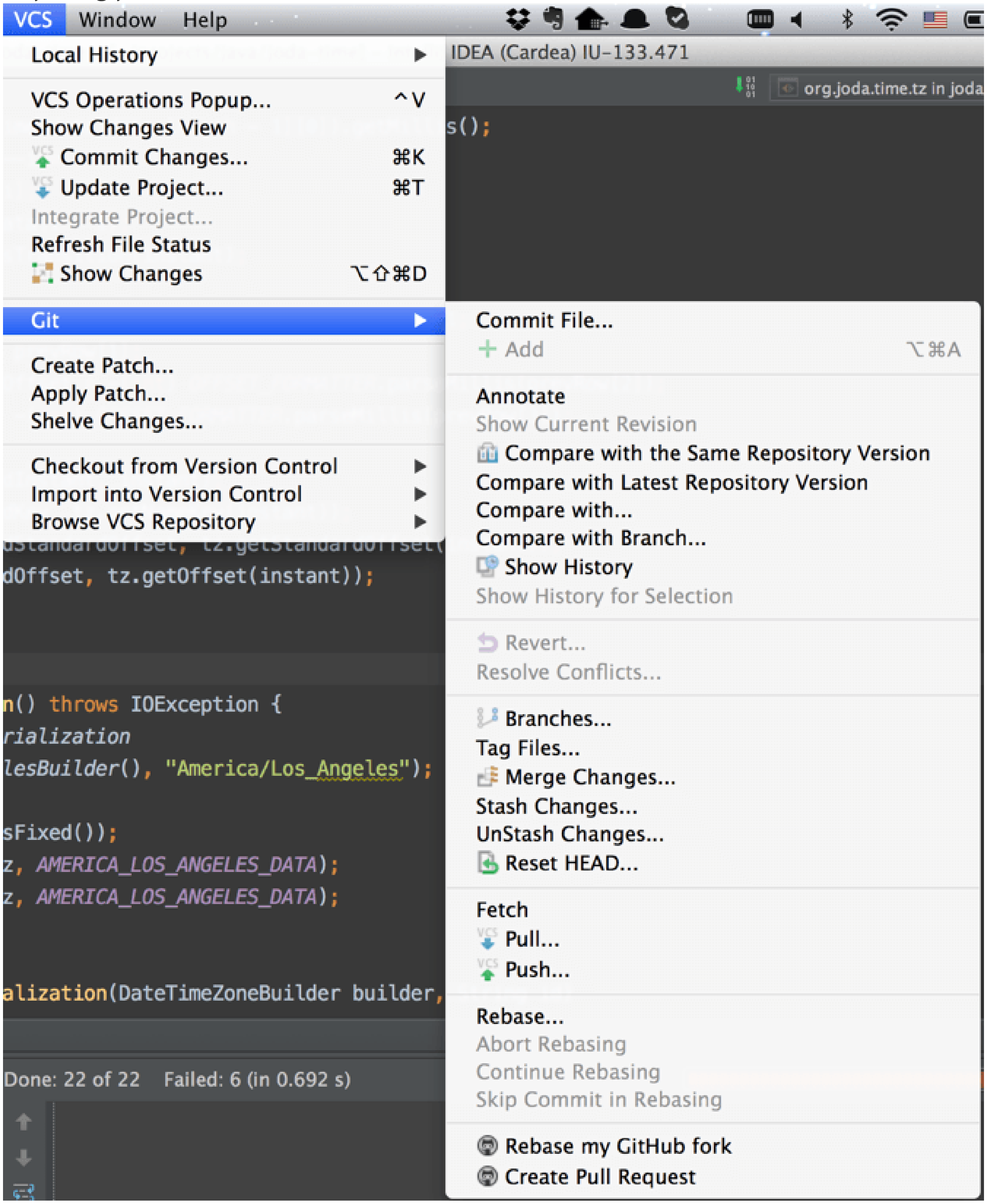
For more information see the Inspections (http://confluence.jetbrains.com/display/IntelliJIDEA/Inspections) tutorial as well as the Intentions one (http://confluence.jetbrains.com/display/IntelliJIDEA/Intentions)

Refactoring

IntelliJ IDEA was initially named Renamer, so refactoring is in the core of the product! There are too many refactorings to cover in detail, but before doing any manual refactoring, check to make sure IntelliJ IDEA doesn't already support it

Refactor Build Run Tools VCS Window
Refactor This... ^T
Rename... ⇧F6
Rename File...
Change Signature... ⌘F6
Type Migration... ⇧⌘F6
Make Static...
Convert To Instance Method...
Move... F6
Copy... F5
Safe Delete... ⌘⌦
Extract ▶
Inline... ⌥⌘N
Find and Replace Code Duplicates...
Invert Boolean...
Pull Members Up...
Push Members Down...
Push ITDs In...
Use Interface Where Possible...
Replace Inheritance with Delegation...
Remove Middleman...
Wrap Method Return Value...
Convert Anonymous to Inner...
Encapsulate Fields...
Replace Temp with Query...
Replace Constructor with Factory Method...
Replace Constructor with Builder...
Generify...
Migrate...
Internationalize...
Also, note that Refactoring, much like Inspections and Navigation are language-dependent. So if some entries are missing, they're not implemented yet for your language of choice.

Version Control

IntelliJ IDEA supports a variety of version control systems including Git, Mercurial, Subversion, Perforce and even TFS.
You can add, delete, commit, revert, branch, push, pull, and pretty much do anything you want from inside the IDE.

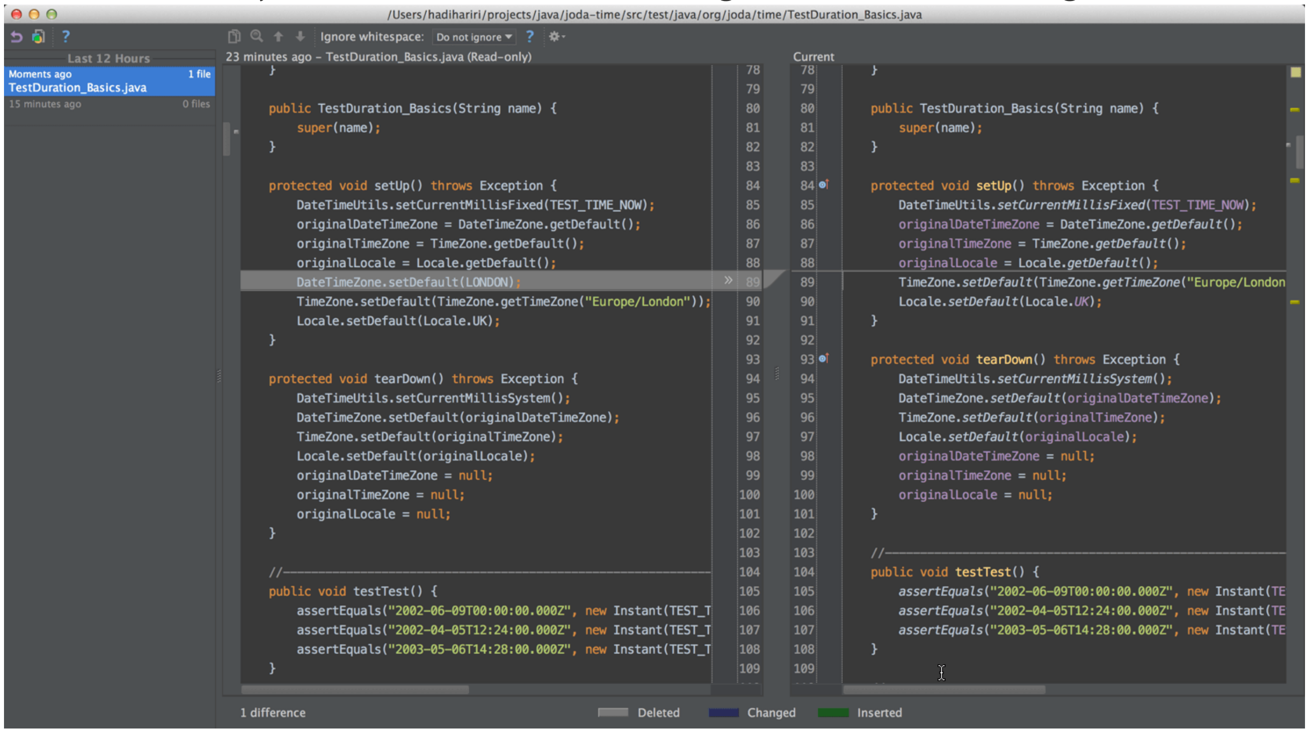


```
s();
dOffset, tz.getOffset(instant));
n() throws IOException {
rialization
lesBuilder(), "America/Los_Angeles");
sFixed());
z, AMERICA_LOS_ANGELES_DATA);
z, AMERICA_LOS_ANGELES_DATA);
alization(DateTimeZoneBuilder builder,
```

```
Done: 22 of 22    Failed: 6 (in 0.692 s)
```

Local History

Something that can save you often is **Local History**. IntelliJ IDEA keeps a local history of your changes, so even if you haven't committed or attached the project to version control, you can still benefit from being able to rollback changes.



/Users/hadihariri/projects/java/joda-time/src/test/java/org/joda/time/TestDuration_Basics.java

Ignore whitespace: Do not ignore

Last 12 Hours

Moments ago 1 file
TestDuration_Basics.java
15 minutes ago 0 files

23 minutes ago - TestDuration_Basics.java (Read-only)

```
    }

    public TestDuration_Basics(String name) {
        super(name);
    }

    protected void setUp() throws Exception {
        DateTimeUtils.setCurrentMillisFixed(TEST_TIME_NOW);
        originalDateTimeZone = DateTimeZone.getDefault();
        originalTimeZone = TimeZone.getDefault();
        originalLocale = Locale.getDefault();
        DateTimeZone.setDefault(LONDON);
        TimeZone.setDefault(TimeZone.getTimeZone("Europe/London"));
        Locale.setDefault(Locale.UK);
    }

    protected void tearDown() throws Exception {
        DateTimeUtils.setCurrentMillisSystem();
        DateTimeZone.setDefault(originalDateTimeZone);
        TimeZone.setDefault(originalTimeZone);
        Locale.setDefault(originalLocale);
        originalDateTimeZone = null;
        originalTimeZone = null;
        originalLocale = null;
    }

    //-----------------------------------------------------------------------
    public void testTest() {
        assertEquals("2002-06-09T00:00:00.000Z", new Instant(TEST_T
        assertEquals("2002-04-05T12:24:00.000Z", new Instant(TEST_T
        assertEquals("2003-05-06T14:28:00.000Z", new Instant(TEST_T
    }
```

Current

```
    }

    public TestDuration_Basics(String name) {
        super(name);
    }

    protected void setUp() throws Exception {
        DateTimeUtils.setCurrentMillisFixed(TEST_TIME_NOW);
        originalDateTimeZone = DateTimeZone.getDefault();
        originalTimeZone = TimeZone.getDefault();
        originalLocale = Locale.getDefault();
        TimeZone.setDefault(TimeZone.getTimeZone("Europe/London
        Locale.setDefault(Locale.UK);
    }

    protected void tearDown() throws Exception {
        DateTimeUtils.setCurrentMillisSystem();
        DateTimeZone.setDefault(originalDateTimeZone);
        TimeZone.setDefault(originalTimeZone);
        Locale.setDefault(originalLocale);
        originalDateTimeZone = null;
        originalTimeZone = null;
        originalLocale = null;
    }

    //-----------------------------------------------------------------------
    public void testTest() {
        assertEquals("2002-06-09T00:00:00.000Z", new Instant(TE
        assertEquals("2002-04-05T12:24:00.000Z", new Instant(TE
        assertEquals("2003-05-06T14:28:00.000Z", new Instant(TE
    }
```

1 difference ▭ Deleted ▭ Changed ▭ Inserted

Tools

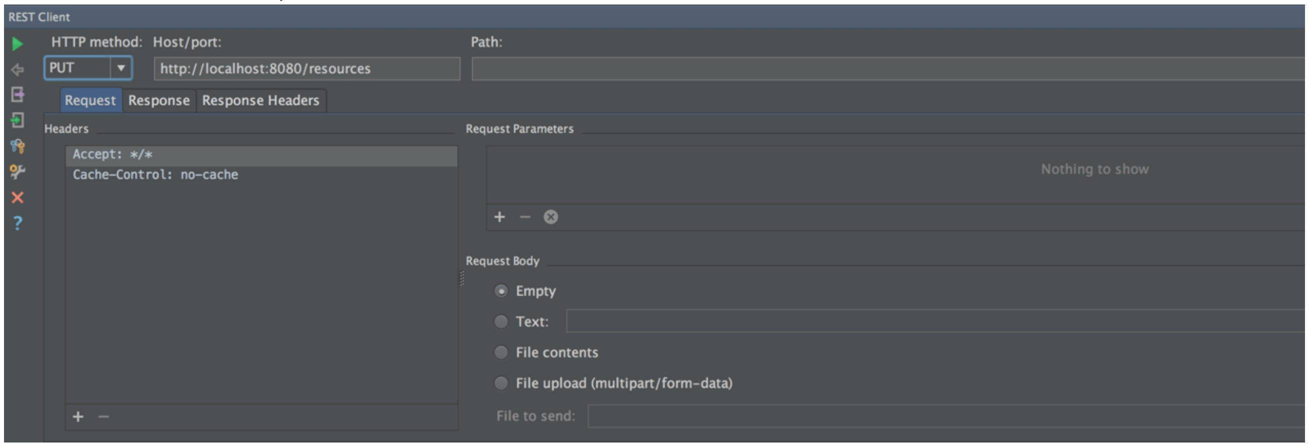
IntelliJ IDEA comes with a series of frequently used tools. Too many to cover in this guide in one go but here are some important ones you might need:

Terminal

Just launch the **Tools | Terminal** and you have complete shell access from inside the IDE

REST Client

Want to talk to your HTTP servers? Issue POST/PUT/DELETE or JSON even? Use the built-in *Test RESTful Web Service*.



REST Client

HTTP method: PUT
Host/port: http://localhost:8080/resources
Path:

Request | Response | Response Headers

Headers

```
Accept: */*
Cache-Control: no-cache
```

Request Parameters

Nothing to show

Request Body

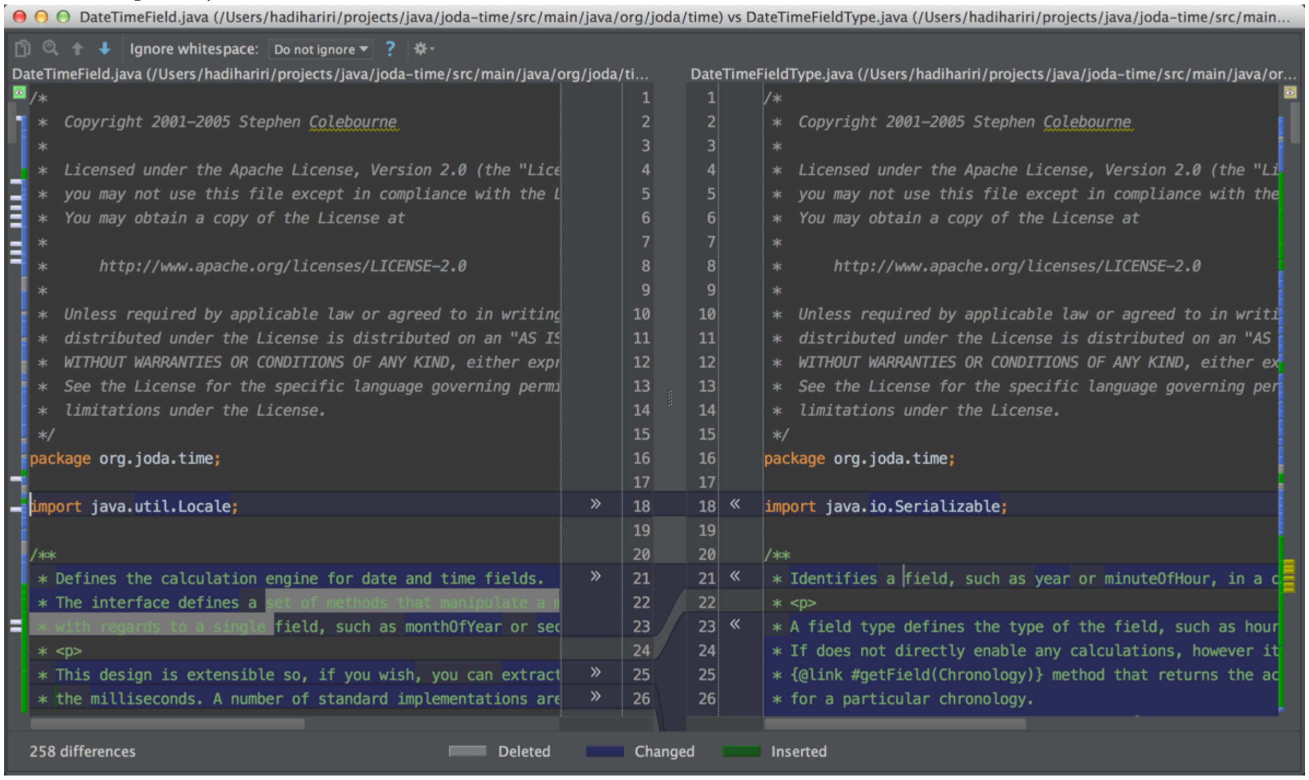
- (●) Empty
- ( ) Text:
- ( ) File contents
- ( ) File upload (multipart/form-data)

File to send:

See this tutorial from PhpStorm (http://www.jetbrains.com/phpstorm/webhelp/testing-restful-web-services.html) for more information. Or watch this video (http://www.youtube.com/watch?v=fpNhaMRaLgY).
Note: This, and many other features and tools of IntelliJ IDEA are part of the core platform so they are available in all the different JetBrains IDE products.

File Comparisons

Same as you can do with Local History, you can select two files and compare them, as well as edit in place. Just select the two files in the Project Window, right-click and *Compare files*



DateTimeField.java (/Users/hadihariri/projects/java/joda-time/src/main/java/org/joda/time) vs DateTimeFieldType.java (/Users/hadihariri/projects/java/joda-time/src/main...

Ignore whitespace: Do not ignore

DateTimeField.java (/Users/hadihariri/projects/java/joda-time/src/main/java/org/joda/ti...

```
/*
 *  Copyright 2001-2005 Stephen Colebourne
 *
 *  Licensed under the Apache License, Version 2.0 (the "Lice
 *  you may not use this file except in compliance with the L
 *  You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 *  Unless required by applicable law or agreed to in writing
 *  distributed under the License is distributed on an "AS IS
 *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either expr
 *  See the License for the specific language governing permi
 *  limitations under the License.
 */
package org.joda.time;

import java.util.Locale;

/**
 * Defines the calculation engine for date and time fields.
 * The interface defines a set of methods that manipulate a m
 * with regards to a single field, such as monthOfYear or sec
 * <p>
 * This design is extensible so, if you wish, you can extract
 * the milliseconds. A number of standard implementations are
```

DateTimeFieldType.java (/Users/hadihariri/projects/java/joda-time/src/main/java/or...

```
/*
 *  Copyright 2001-2005 Stephen Colebourne
 *
 *  Licensed under the Apache License, Version 2.0 (the "Li
 *  you may not use this file except in compliance with the
 *  You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 *  Unless required by applicable law or agreed to in writi
 *  distributed under the License is distributed on an "AS
 *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either ex
 *  See the License for the specific language governing per
 *  limitations under the License.
 */
package org.joda.time;

import java.io.Serializable;

/**
 * Identifies a field, such as year or minuteOfHour, in a c
 * <p>
 * A field type defines the type of the field, such as hour
 * If does not directly enable any calculations, however it
 * {@link #getField(Chronology)} method that returns the ac
 * for a particular chronology.
```

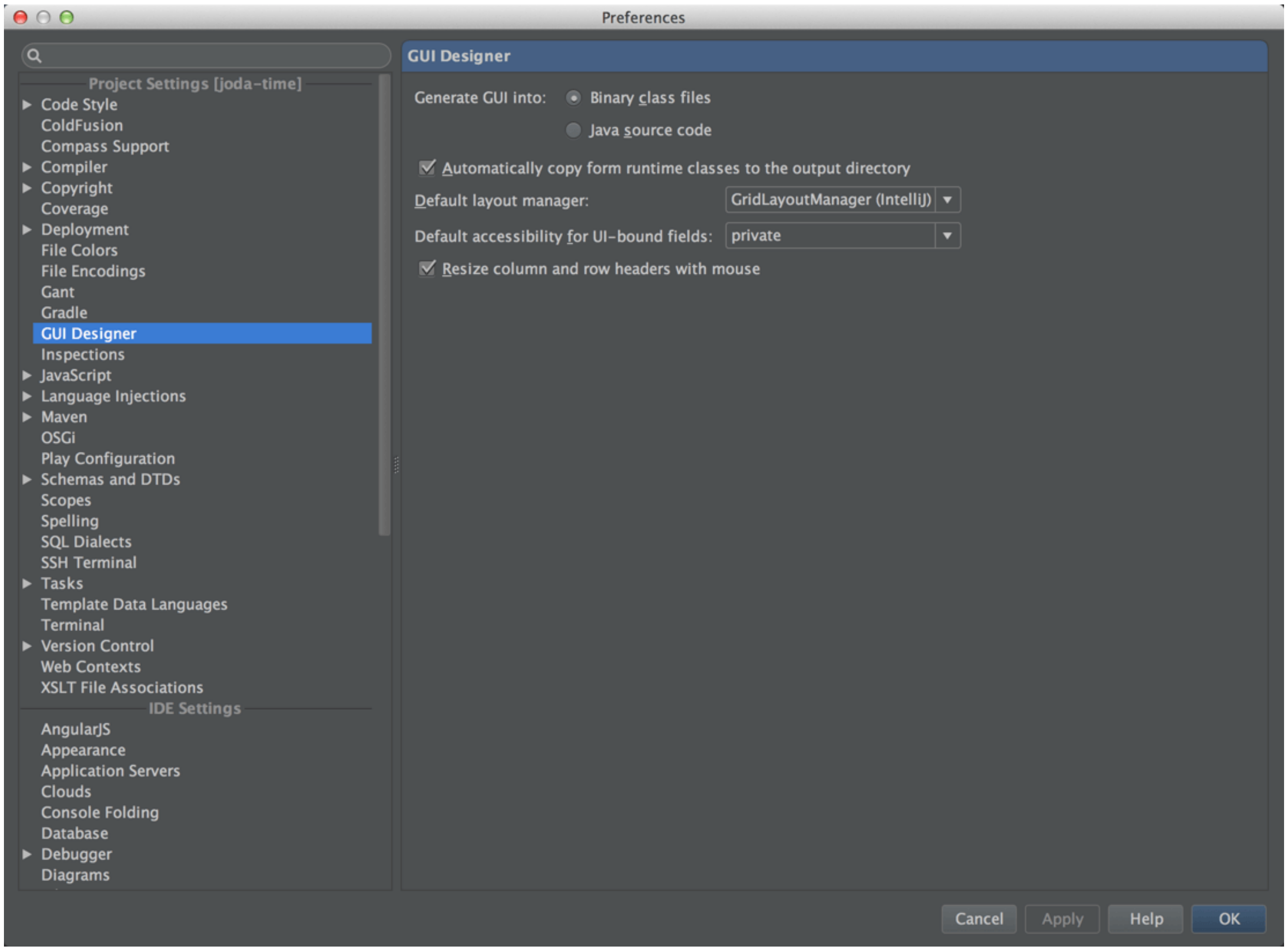
258 differences   ▭ Deleted   ▬ Changed   🟩 Inserted

Database Tools

Want to talk to a SQL Database? Just use the built-in database tools (http://blog.jetbrains.com/phpstorm/2013/10/database-tools-and-sql-editor-changes-in-phpstorm-7/). (this post is a bit outdated as it's now much easier to set up new connections. No need to specify JDBC string anymore).

Configuration

IntelliJ IDEA is extremely configurable and you'll discover features as you use it more. It would be impossible to cover every aspect in this guide, but a couple of hints:


Preferences

GUI Designer

Generate GUI into: [✓ Binary class files] Java source code

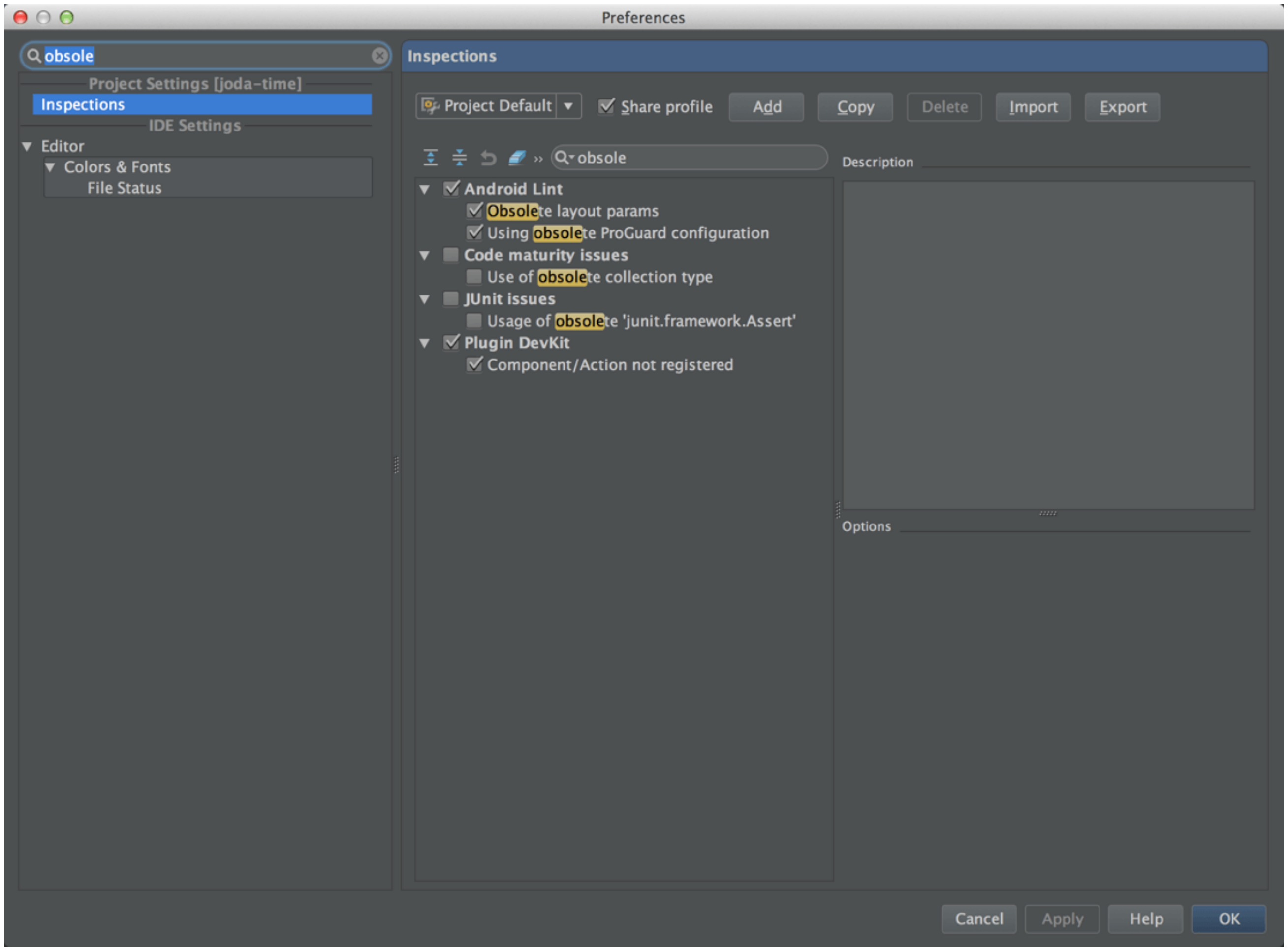
✓ Automatically copy form runtime classes to the output directory

Default layout manager: GridLayoutManager (IntelliJ)

Default accessibility for UI-bound fields: private

✓ Resize column and row headers with mouse

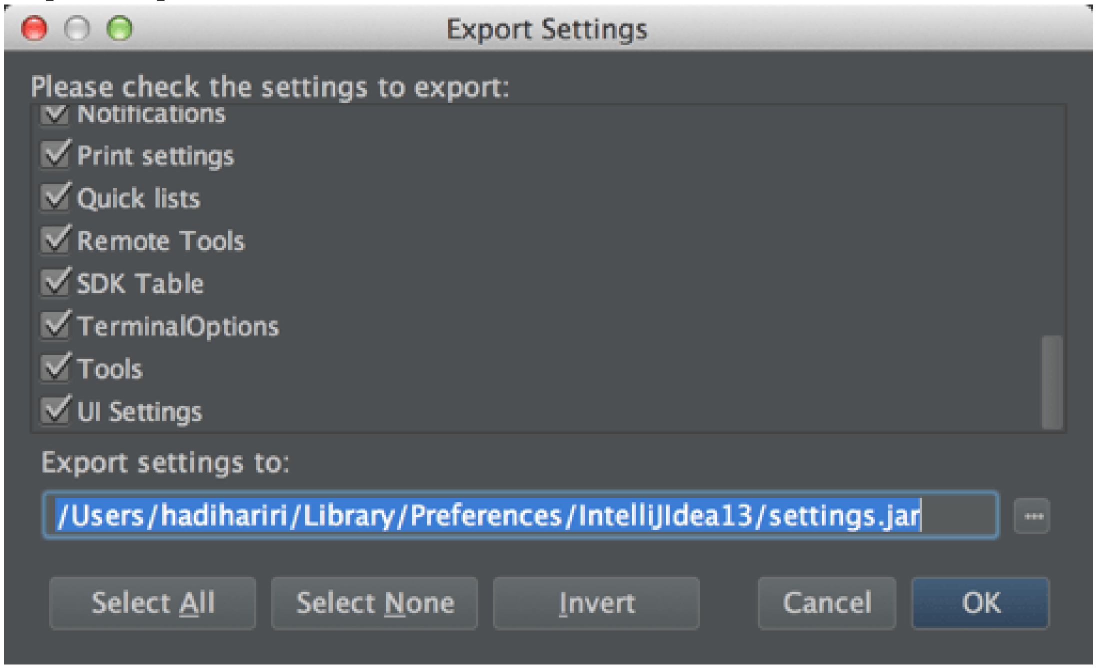
Do not sit looking through pages and tabs of configuration, just open up the dialog box and start typing. There’s a quick find. For instance, want to go to change inspection settings for obsolete code, start typing ‘obsol…’



Android Lint
Obsolete layout params
Using obsolete ProGuard configuration
Code maturity issues
Use of obsolete collection type
JUnit issues
Usage of obsolete 'junit.framework.Assert'
Plugin DevKit
Component/Action not registered

Importing and Exporting Settings

To import and export settings use **File | Import Settings** and **File | Export Settings** respectively. Settings are saved as *.jar* files. All settings, be it keyboard mappings, templates, fonts, colors are all saved in the same file. You can pick what you want to import/export.



Please check the settings to export:

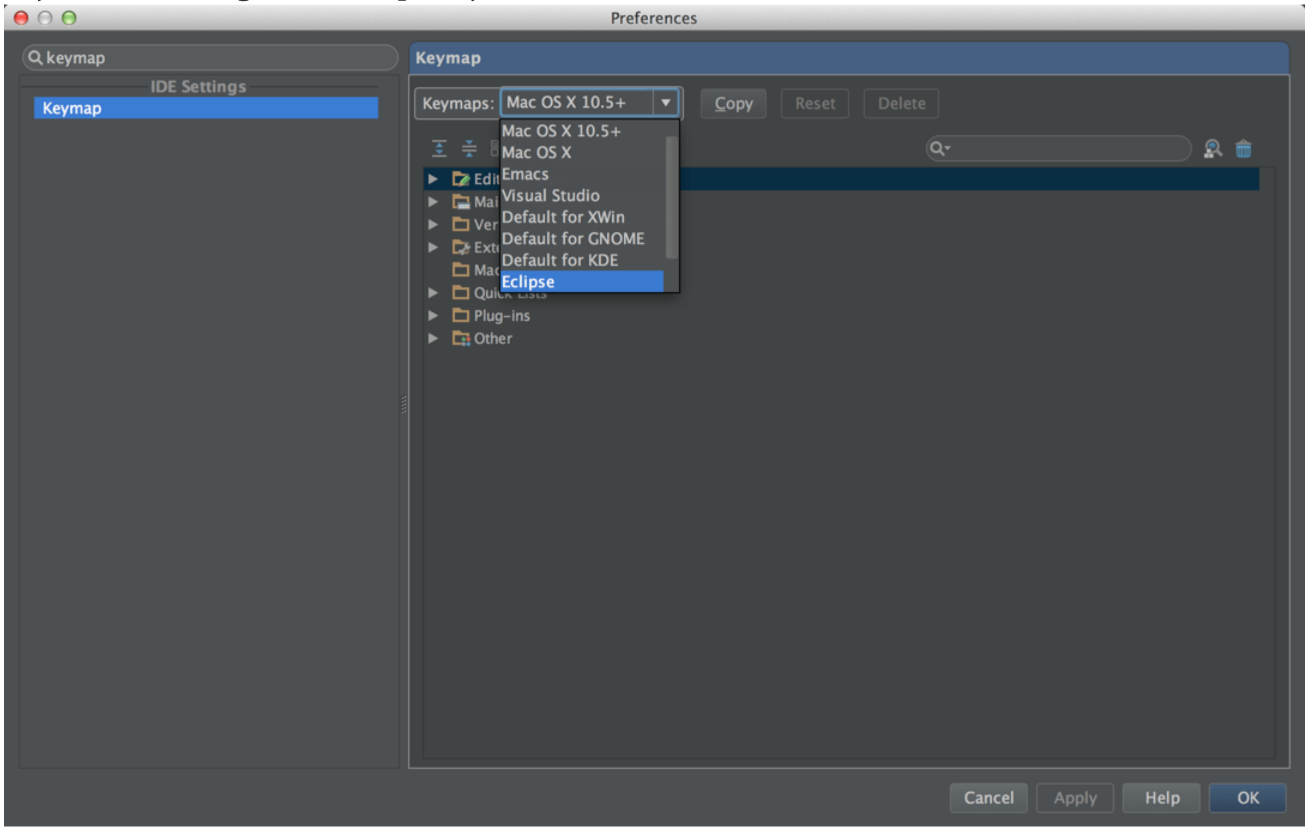
- [x] Notifications
- [x] Print settings
- [x] Quick lists
- [x] Remote Tools
- [x] SDK Table
- [x] TerminalOptions
- [x] Tools
- [x] UI Settings

Export settings to:

/Users/hadihariri/Library/Preferences/IntelliJIdea13/settings.jar

Key Bindings and Schemes

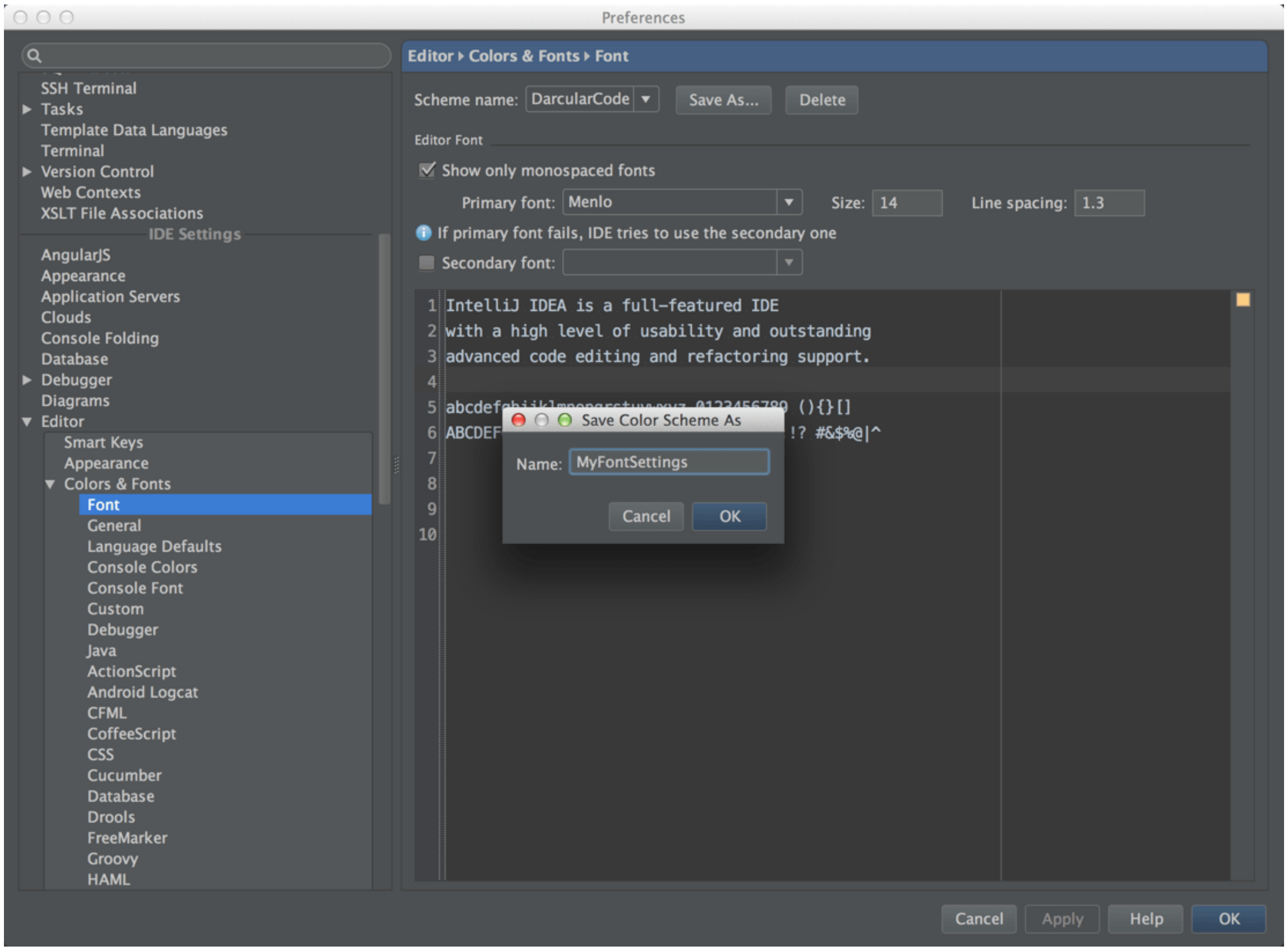
Configure the key bindings to best adapt to your needs. If you are coming from Visual Studio and using ReSharper with IntelliJ IDEA or ReSharper 2.x key mapping, there's an option available for you. If you prefer ReSharper settings, I've created a file (although I no longer maintain) with these mappings (https://github.com/hhariri/Tidbits/blob/master/resharper.jar).
If you're coming from Eclipse, you have one too.



Personally I try and adapt to a new environment learning the settings for that one. On Mac I use the default Mac OSX+ keymap scheme.

Fonts, Colors and Theme

I use Darcula theme with my own font size. You can change it to your needs. But remember, you need to first Save the scheme to a new name before you can change it.



Scheme name: DarcularCode

✓ Show only monospaced fonts

Primary font: Menlo   Size: 14   Line spacing: 1.3

If primary font fails, IDE tries to use the secondary one

Secondary font:

```
IntelliJ IDEA is a full-featured IDE
with a high level of usability and outstanding
advanced code editing and refactoring support.

abcdef                                   (){}[]
ABCDEF                                   !? #&$%@|^
```

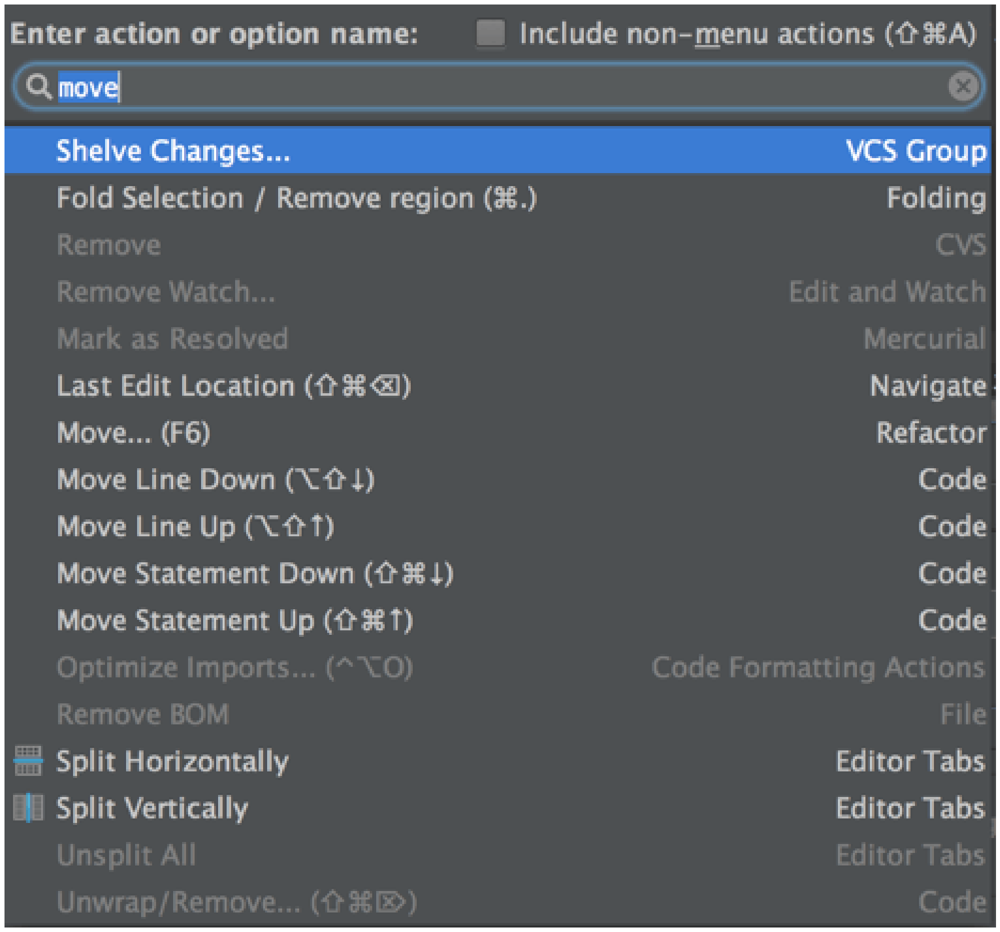
Save Color Scheme As

Name: MyFontSettings

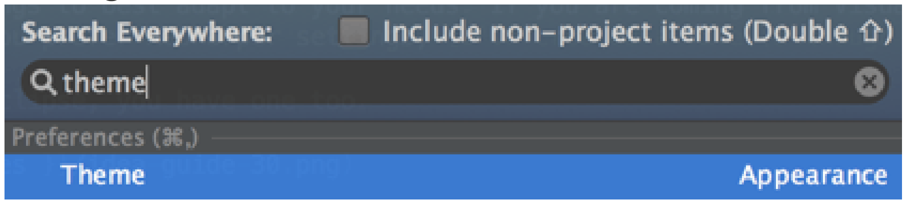
Remember these two keyboard shortcuts

If all else fails remember:

- **Double Shift**: Search everywhere. And I really mean everywhere. Even settings.


Enter action or option name:
Include non-menu actions (⇧⌘A)
move
Shelve Changes... VCS Group
Fold Selection / Remove region (⌘.) Folding
Remove CVS
Remove Watch... Edit and Watch
Mark as Resolved Mercurial
Last Edit Location (⇧⌘⌫) Navigate
Move... (F6) Refactor
Move Line Down (⌥⇧↓) Code
Move Line Up (⌥⇧↑) Code
Move Statement Down (⇧⌘↓) Code
Move Statement Up (⇧⌘↑) Code
Optimize Imports... (^⌥O) Code Formatting Actions
Remove BOM File
Split Horizontally Editor Tabs
Split Vertically Editor Tabs
Unsplit All Editor Tabs
Unwrap/Remove... (⇧⌘⌦) Code

- **Cmd+Shift+A**: Can’t remember a keyboard shortcut or menu entry? Start looking for it



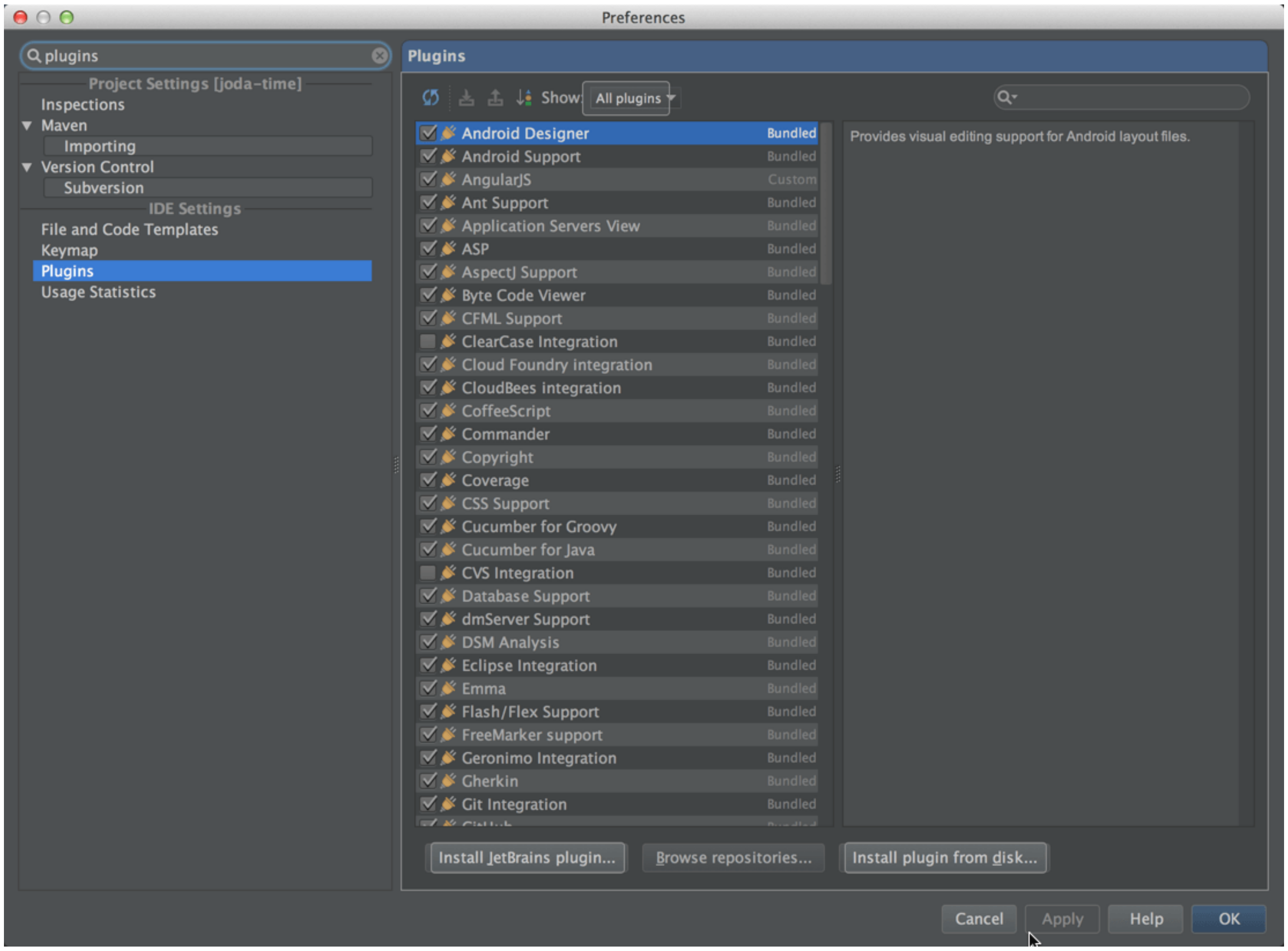
Editing

No Save

You don’t need to save files in IntelliJ IDEA. It automatically does so.

Plugins

Over 1000 plugins so too many to even consider mentioning here. You can either browse or search for them via the browser on the JetBrains plugin site (http://plugins.jetbrains.com) or directly from inside IntelliJ IDEA



✓	Android Designer	Bundled
✓	Android Support	Bundled
✓	AngularJS	Custom
✓	Ant Support	Bundled
✓	Application Servers View	Bundled
✓	ASP	Bundled
✓	AspectJ Support	Bundled
✓	Byte Code Viewer	Bundled
✓	CFML Support	Bundled
	ClearCase Integration	Bundled
✓	Cloud Foundry integration	Bundled
✓	CloudBees integration	Bundled
✓	CoffeeScript	Bundled
✓	Commander	Bundled
✓	Copyright	Bundled
✓	Coverage	Bundled
✓	CSS Support	Bundled
✓	Cucumber for Groovy	Bundled
✓	Cucumber for Java	Bundled
	CVS Integration	Bundled
✓	Database Support	Bundled
✓	dmServer Support	Bundled
✓	DSM Analysis	Bundled
✓	Eclipse Integration	Bundled
✓	Emma	Bundled
✓	Flash/Flex Support	Bundled
✓	FreeMarker support	Bundled
✓	Geronimo Integration	Bundled
✓	Gherkin	Bundled
✓	Git Integration	Bundled

Provides visual editing support for Android layout files.

You can install plugins from:

- **JetBrains Plugins**: These are plugins created and maintained by JetBrains (i.e. official)
- **Browse Repositories**: By default points to plugins.jetbrains.com but you can point to other repositories too.
- **Install plugin from disk**: Point to a zip file (plugins ship as zip files)

You can also of course create your own plugins (http://confluence.jetbrains.com/display/IntelliJIDEA/Plugin+Development). And while I can't mention any because there are way too many cool ones, for VIM lovers, there's IDEAVim which gives you a somewhat Vim experience inside IntelliJ IDEA.

IntelliJ IDEA For Eclipse Users

Anton from Zeroturnaround has written an awesome and complete guide for IntelliJ IDEA users coming from Eclipse (http://zeroturnaround.com/rebellabs/getting-started-with-intellij-idea-as-an-eclipse-user/). Highly recommended if it's your case. And still recommended even if you're not coming from Eclipse as it contains a ton of tips and tricks.

Change Log

Tracks changes made to this guide.

Date	Change
06th Jan 2014	First Draft Published
07th Jan 2014	Fixed typo's
12th Feb 2014	Added IntelliJ for Eclipse Users

Twitter (http://twitter.com/hhariri) Google+ (http://plus.google.com/+HadiHariri1)

LinkedIn (http://linkedin.com/pub/hadi-hariri/83/68a/b03)

Github (http://github.com/hhariri)

IntelliJ IDEA Minimal Survival Guide was published on January 06, 2014 by Hadi Hariri (/about).

YOU MIGHT ALSO ENJOY (VIEW ALL POSTS (/POSTS/))

- Build Tools - Make, no more (/2014/04/21/build-make-no-more/)
- The Vendor Session (/2014/04/18/the-vendor-session/)
- Getting started with a Kotlin OSS Project (/2014/04/17/getting-started-with-kotlin-oss-project/)
