GETTING STARTED WITH INTELLIJ IDEA AS AN ECLIPSE USER

BY ANTON ARHIPOV

REBELLABS

TABLE OF CONTENTS

Click to go to section

INTRODUCTION:
WHY INTELLIJ IDEA?

Coffee or Tea? Mac vs. PC? iPhone OS or Android? Eclipse or IntelliJ IDEA (or NetBeans)? Believe it or not, software developers argue as much about their IDE as any of these other hotly-debated topics. And for a good reason...

Confessions of an IntelliJ addict

My name is Anton, and I'm an IntelliJ IDEA addict. Whew, it feels good to say it out loud. The choice of IDE for developers is one of the most contentious debates in the software game. Unless you have a darn good reason, you don't often use more than one IDE, let alone all three of the major IDEs in the Java world: Eclipse, IntelliJ IDEA and NetBeans.

The truth is that I have been a happy IntelliJ IDEA (for the sake of brevity and laziness, you'll see IntelliJ IDEA referred to in full, as well as simply "IntelliJ" and "IDEA" as well from here on) user since 2004—a full decade. Over the last 10 years, I have always been discovering the new aspects of the IDE and I'm always waiting for the new version release with an excitement.

But why? After all, aren't all IDEs more or less the same? (BOOO! HISSSSS!) Was there something that happened in my past to make me so dedicated to this particular IDE?

Well, before settling on IDEA, I had worked with Eclipse and NetBeans, even championing Eclipse at one point in my former job. But by using IDEA and Eclipse almost side by side for some time I realized that IDEA supports the work I was doing much better. Eventually, I dodged other IDEs and decided to become a full-time IDEA user. When I joined ZeroTurnaround, my new role required me to learn all IDEs in-depth due to the fact that our flagship product, JRebel, ships as an IDE plugin for Eclipse, NetBeans, IDEA, and JDeveloper. My primary IDE is IntelliJ IDEA and my teammates think of me as of an IntelliJ IDEA addict.

Now it might be only my impression, but back then IDEA had the best refactoring capabilities. These days, I think it still is more consistent in terms of UX, and there are shortcuts for almost every action. Not to mention it understands the source code context better than others. Subjectively, it made me more productive, but this kind of depends on the set of features that you, as a developer, need and want. If the IDE matches what you actually use then it is probably the best IDE for you :)

The Java IDE market landscape

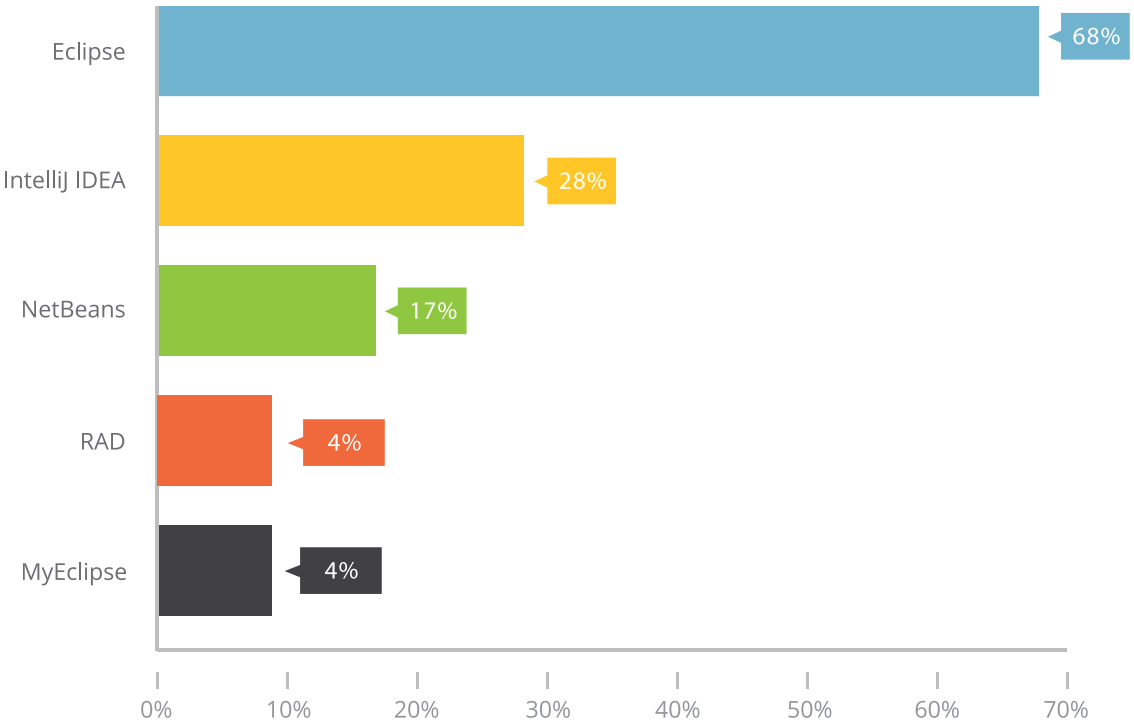
Here you can see a recent breakdown of the IDE landscape (as of 2012). We are planning to survey developers again in 2014 for new stats, but most likely, things haven't changed that drastically. Note: Answers were non-exclusive and asked developers to select IDEs in use, not single selections only).

As you can see, I'm not alone. Close to 1/3 of Java developers that we surveyed in 2012 use IDEA on at least some projects. However, neither IDEA nor third-place NetBeans compare to Eclipse, which clearly dominates the scene. Understandably, Eclipse is an advanced tool that is completely free and open source (as is NetBeans), whereas you have to pay to get the most out of IDEA. Wait, what?

Yep. What makes IDEA different is that was a commercial-only tool until 2009, when IntelliJ IDEA 9 was released. Now we have both Ultimate and Community editions. Community Edition is free and open source and Ultimate Edition is a commercial product and builds on top of Community Edition and offers more features and support.

A number of features are provided only in the commercial distribution **and in this guide we assume that you will try Ultimate Edition when evaluating the IDE**. The community edition is enough for Java and even for Android development, but you will miss a lot of cool features. See the full comparison of the two IDEA versions at the official product page: http://www.jetbrains.com/idea/features/editions_comparison_matrix.html

The Java IDE market landscape (2012)



REBELLABS

What you'll see in this report

My main purpose in writing this report is to show Eclipse users, specifically, how to get started using IDEA faster and with less headaches.

Perhaps you are Eclipse users who are interested in trying out IntelliJ IDEA, or perhaps considering the migration. Moving from Eclipse to IDEA can be quite overwhelming. People usually neither have patience nor time to learn new IDE with its own set of shortcuts. Even more - in some cases it requires changing your mindset about how you use your IDE. With this guide, my goal is to point out the main differences when moving from Eclipse to IDEA so that the path wouldn't be as painful.

I have seen a lot of my colleagues going from Eclipse to IntelliJ IDEA. They struggled initially, but survived. In a few days or weeks they didn't want to come back to their previous IDE. Naturally, I have tried to help a lot of my friends to migrate to IDEA and with this guide I'd like to help YOU as well.

So, I've broken it down into a few chapters. In Chapter I, we'll talk about getting your project set up right, Chapter II looks at keymap, navigation and editor settings, and Chapter III covers more advanced things like testing, deploying and artifacts. So, let's get started!

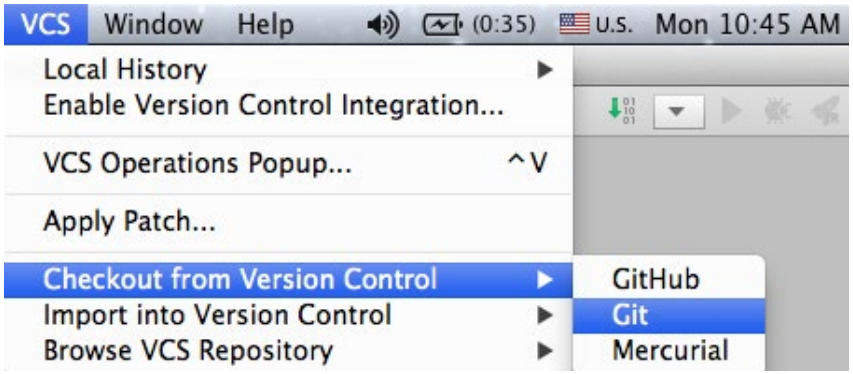
CHAPTER I:
GETTING YOUR FIRST INTELLIJ IDEA PROJECT SET UP

Let's start the guide with getting your first project set up. You wouldn't be able to give IDEA a try if you don't have a project opened, right? Instead of creating a new project, you most likely will need to open—or import—an existing one.

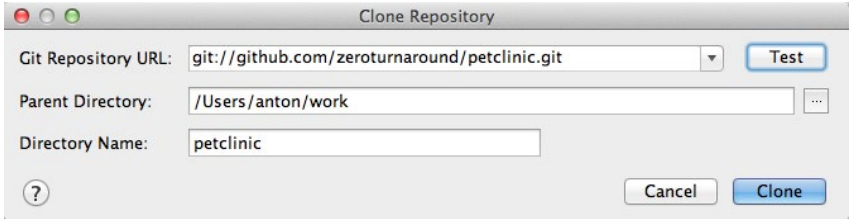
Cloning and importing the project from a Git repository

Let's clone a sample project from https://github.com/zeroturnaround/petclinic. This is a Petclinic web application based on the Spring framework.

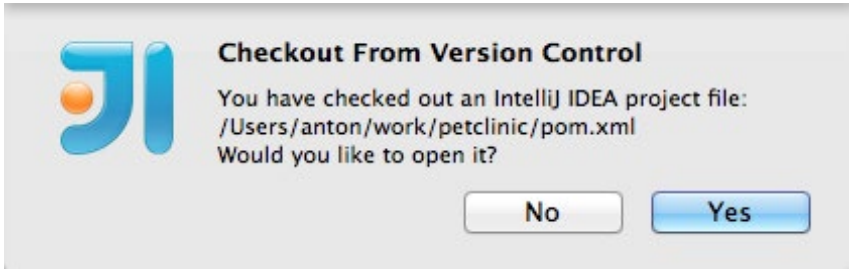
1. VCS → Checkout from Version Control → Git



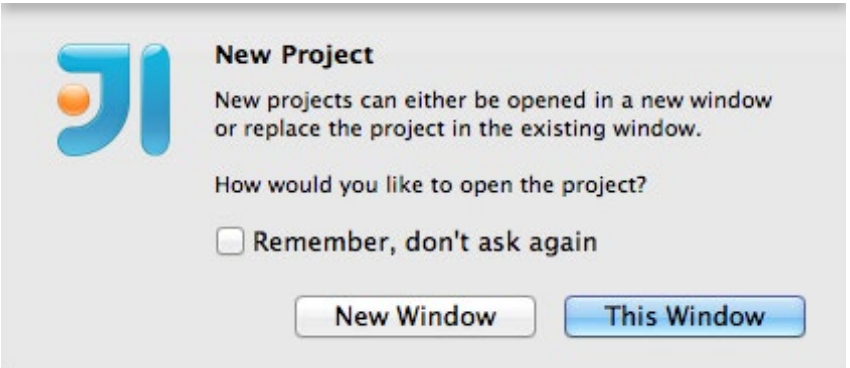
2. In the dialog, enter the repository URL and the destination folder. Click **Clone** to import the project.



3. Once the project sources are cloned, the IDE will ask if you want to open the project because it discovered Maven's **pom.xml**. In addition to Maven, IDEA also detects Gradle and Eclipse configuration files.



If you had another project opened, the IDE will ask if you want to open it in a separate window.



As an Eclipse user, you might be confused here—why would you need to open a project in another IDE window? This is not how the things work in Eclipse. With IntelliJ IDEA, you can have an IDE window per project. We will see more details about this a bit later in the **Projects vs Modules** part.

4. Voila! You're done! As you have confirmed to open the project at the previous step, the project imports automagically and is ready to be hacked on. Keep in mind that after the project import, it will take some time for IDEA to index your project files. Later you'll get this time back using smart code assistance.

> **Note:** If you're an Eclipse user, for now just assume that "Project" in IntelliJ IDEA is the same as "Workspace" in Eclipse, and "Module" in IntelliJ IDEA is the same as "Project" in Eclipse.

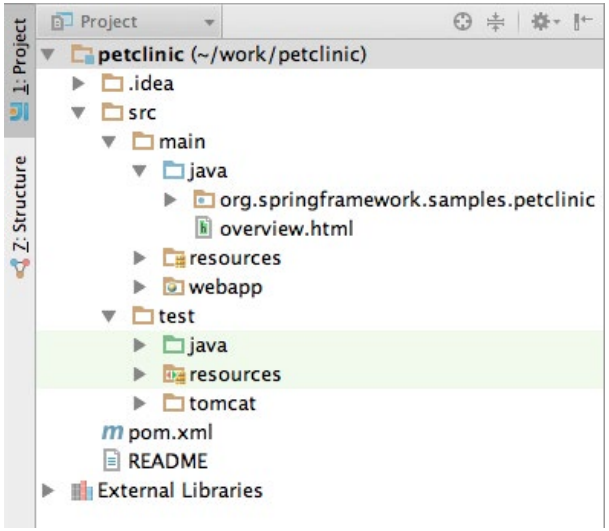
Project Structure and Configuration

Project structure and configuration in IntelliJ IDEA is very different from Eclipse. Almost everything—from project structure, folder management, how the packages are represented in the project tree, and this terminology issue with **projects vs modules—**looks really weird to an experienced Eclipse user. In this part, we will highlight the important aspects of the project structure and configuration.

FOLDERS

Every project module can include a number of folders that can be handled differently.

When importing the project from a model (e.g. in the case of a Maven project) IDEA automatically discovers the source, test and resources folders. You can also set the folder type manually by selecting the appropriate option via the *Mark Directory As* context menu by right-clicking on the folder in project view.



The project tool window can be accessed with **Alt+1** / **Cmd+1** shortcut. Here is a quick key for defining the meaning of folder labels:

Source - marks the folder as a source root

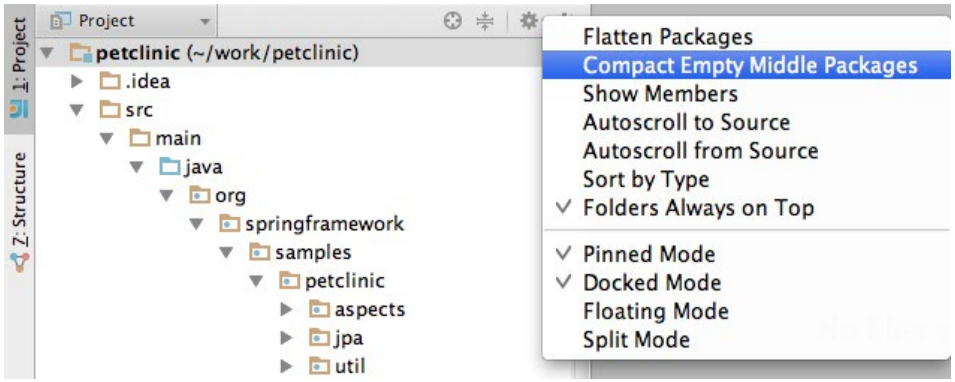
Test - marks the sources as tests

Excluded - If you mark a folder as ignored it won't display in the project tree and its contents will not be indexed. Note that you might want to exclude a folder if there are a lot of big files, otherwise it can cause a bit of performance penalty for the indexer.

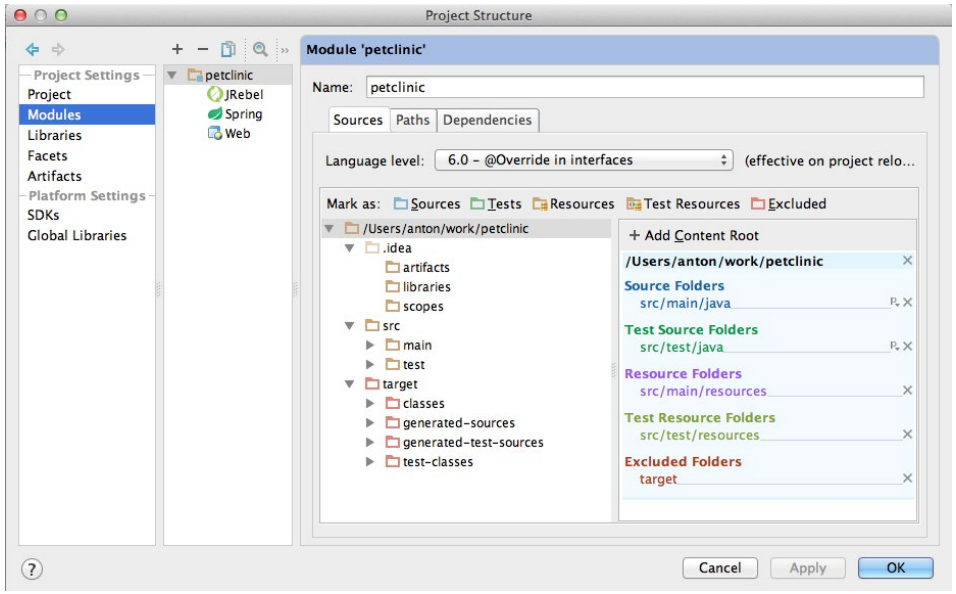
Resources - The new type of folders introduced in IDEA 13 - the resources folder. The contents of the resource folder are automatically copied to the compiler's target folder.

Under *External Libraries* you will find all the dependencies for the module.

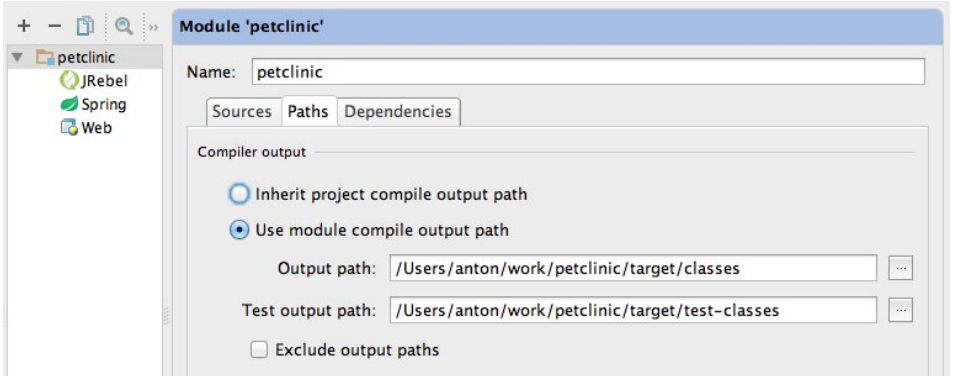
The project view can be adjusted according to your preferences.
For instance, the way the packages are displayed, or if you would like to autoscroll to the source when selecting a resource in the project view, etc.



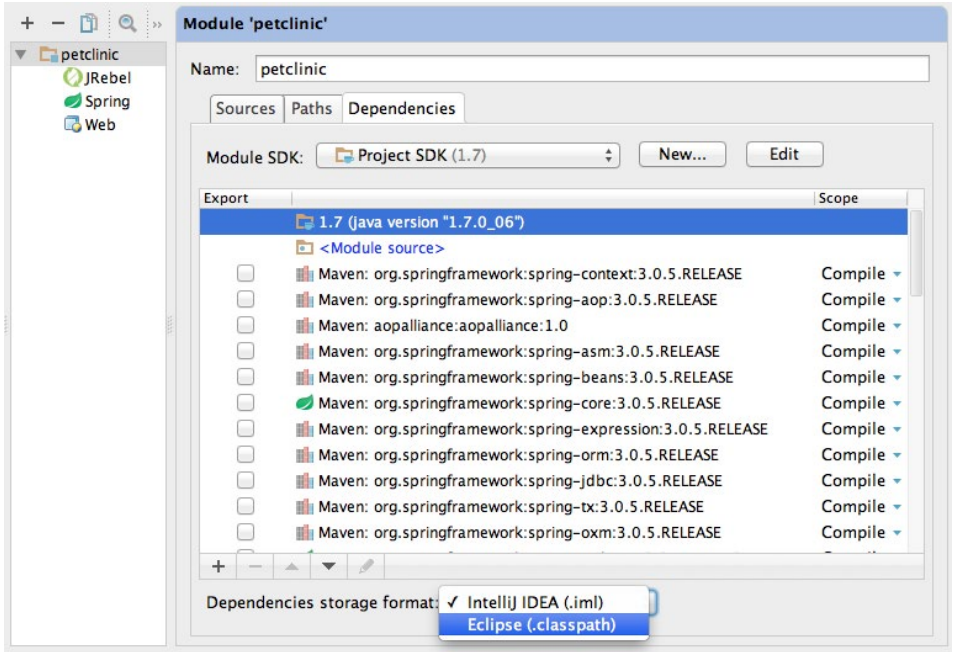
To get the detailed overview of all the modules, libraries and other settings related to the project, open the *Project Structure* dialog **Ctrl+Shift+Alt+S / Cmd+**;



The modules view in the *Project Structure* dialog provides a full overview of the folder types in the selected module. Note that the Java language level can also be configured per module, i.e. one module could use Java 5 syntax, but you may want to set it to Java 7 for the other one



For defining the compilation target folder there are two options: either put every module's compiled classes to the same location, or use a module-specific location, which can be overridden.



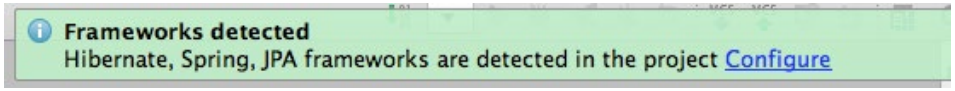
Dependencies can be specified in a variety of ways. For a Maven project, IDEA just automatically imports the dependencies specified in **pom.xml** and stores the list in an ***.iml** file in the root of the module. Alternatively, it is possible to tell IDEA to save the list in Eclipse-specific format.

The official IntelliJ IDEA documentation explains what are the specific configuration files for a project — http://www.jetbrains.com/idea/webhelp/project.html, and a module — http://www.jetbrains.com/idea/webhelp/module.html

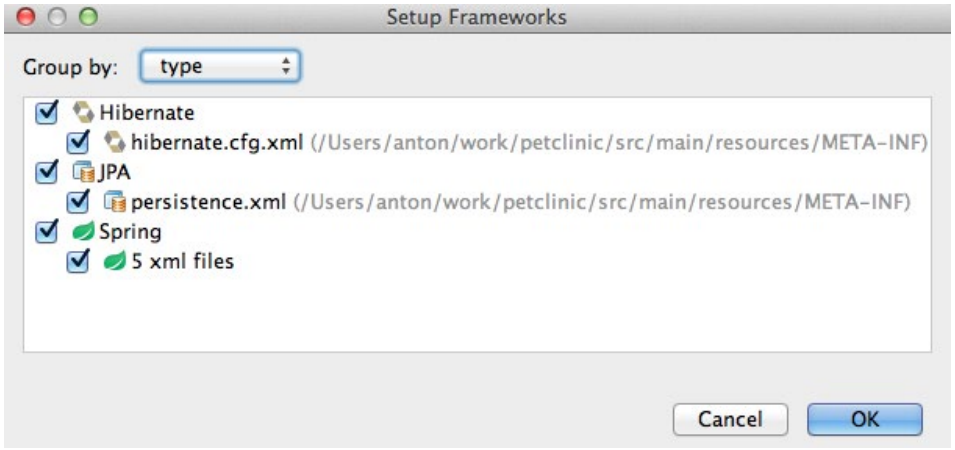
Modules may have the references to other modules as well. It is similar to specifying the required project dependency in the Java Build Path in Eclipse.

FACETS

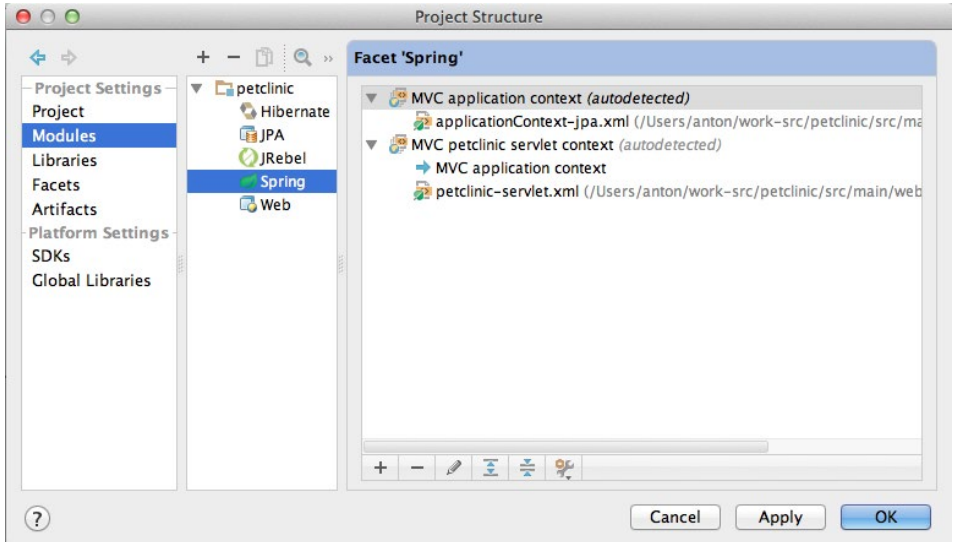
Once the project is opened and after the initial scanning, IDEA will tell you that it has detected some artifacts and will suggest that you set some *facets* for the module, which are like little configuration assistants.



By clicking the Configure link in the popup, it will bring out a dialog with the list of configurations IDEA suggests for you to configure (i.e. Hibernate, Spring, JPA, etc).



Facets are used to streamline the project configuration in IDEA, and they encapsulate the support for a variety of frameworks, technologies and languages. For instance, a tool—let's use JRebel as an example of shameless plug—can be set up to add its own facet to configure specific properties for a module. You can see below the JRebel facet there, just like you have for Spring. Now you can configure web-specific properties for the module.



Interestingly, facets exist in Eclipse too, but for some reason this configuration method isn't adopted very widely. In contrast to Eclipse, IDEA users depend on facets as a really common way of configuring modules.

MODULES VS PROJECTS

Now, back to the Modules vs Projects dilemma. Before, I wrote that you should think of IDEA *projects* as of Eclipse *workspaces*. This analogy is good enough for the beginning, but it isn't exactly correct, as projects in IDEA are not generally used in the same way as Eclipse workspaces.

In Eclipse, you would usually put a lot of different projects into the same workspace and then filter them with *Working Sets* or by closing the projects that aren't relevant at the moment. That said, in Eclipse workspaces are rarely used for grouping arbitrary projects. Workspaces are used rather as a common root and the projects are added to or removed from the workspace. The average Eclipse user would have a limited number of workspaces in the file system.

In contrast, IDEA's project is an umbrella for *logically connected modules*, whereas the modules are rather similar to Eclipse projects with the only difference that IDEA modules can be nested within the project tree.

> A module is a part of the project that can be built independently.

Normally, you wouldn't put the disconnected modules under the same project. Technically, you can, but that's not idiomatic. You would rather open a new IDE window with the second project than importing another non-related module into the existing project.

CHAPTER II:
GETTING COMFORTABLE WITH IDEA'S KEYMAP, NAVIGATION AND SETTINGS

Get ready to make a list! In this chapter, we go over in detail all the shortcuts you'll need to effectively get started with IDEA, and highlight the major differences that might befuddle Eclipse users at first...also, we ***may*** have a cheatsheet for you.

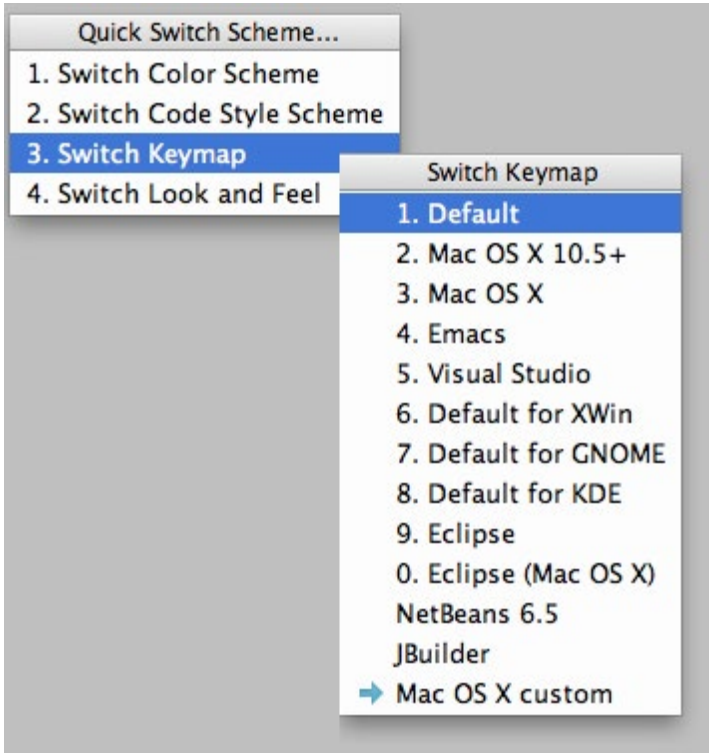
eclipse			IntelliJIDEA		
ACTION	**WINDOWS**	**OS X**	**ACTION**	**WINDOWS**	**OS X**
Code completion	Ctrl+Space	^Space	Basic completion	Ctrl+Space	^Space
-	-	-	**Smart completion**	Ctrl+Shift+Space	⇧^Space
Quick access	Ctrl+3	⌘3	**Search everywhere**	Shift × 2	⇧ x 2
Maximize active view or editor	Ctrl+M	^M	**Hide all tool windows**	Ctrl+Shift+F12	⇧⌘F12
Open type	Ctrl+Shift+T	⇧⌘ T	Navigate to class	Ctrl+N	⌘N
Next view	Ctrl+F7	⌘F7	-	-	-
-	-	-	**Recent files**	Ctrl+E	⌘E
-	-	-	**Switcher**	Ctrl+Tab	^ →⇥
Quick outline	Ctrl+O	⌘O	**File structure**	Ctrl+F12	⌘F12
Move lines	Alt + Up/Down	⌥ ⇅	Move lines	Alt+Shift+Up/Down	⇧⌥ ⇅
Delete lines	Ctrl+D	⌘D	Delete lines	Ctrl+Y	⌘Y
Quick fix	Ctrl+1	⌘1	**Show intention action**	Alt+Enter	⌥ ⏎
Quick switch editor	Ctrl+E	⌘E	**Switcher**	Ctrl+Tab	^→⇥
-	-	-	**Recent files**	Ctrl+E	⌘E
Quick hierarchy	Ctrl+T	⌘T	Navigate to type hierarchy	Ctrl+H	^H
-	-	-	**Navigate to method hierarchy**	Ctrl+Shift+H	⇧⌘H
-	-	-	**Show UML popup**	Ctrl+Alt+U	⌥⌘U
Last edit location	Ctrl+Q	^Q	Last edit location	Ctrl+Shift+Backspace	⇧^⌫
Next editor	Ctrl+F6	⌘F6	**Select next tab**	Ctrl+Left	^ ⟶
-	-	-	**Switcher**	Ctrl+Tab	^ →⇥
-	-	-	**Recent files**	Ctrl+E	⌘E
Run	Ctrl+Shift+F11	⇧⌘F11	Run	F10	F10

eclipse			IntelliJIDEA		
ACTION	**WINDOWS**	**OS X**	**ACTION**	**WINDOWS**	**OS X**
Correct indentation	Ctrl+I	⌘I	Auto-indent lines	Ctrl+Alt+I	⌥⌘I
Format	Ctrl+Shift+F	⇧⌘F	Reformat code	Ctrl+Alt+L	⌥⌘L
Open resource	Ctrl+Shift+R	⇧⌘R	Navigate to file	Ctrl+Shift+N	⇧⌘N
Surround with	Ctrl+Alt+Z	⌥⌘Z	Surround with	Ctrl+Alt+T	⌥⌘T
-	-	-	Surround with live template	Ctrl+Alt+J	⌥⌘J
Open declaration	F3	F3	Navigate to declaration	Ctrl+B	⌘B
-	-	-	**Quick definition**	Ctrl+Shift+I	⇧⌘I
Open type hierarchy	F4	F4	Navigate to type hierarchy	Ctrl+H	-
-	-	-	**Show UML popup**	Ctrl+Alt+U	⌥⌘U
References in workspace	Ctrl+Shift+G	⇧⌘G	**Find usages**	Alt+F7	⌥F7
-	-	-	**Show usages**	Ctrl+Alt+F7	⌥⌘F7
-	-	-	**Find usages settings**	Ctrl+Alt+Shift+F7	⇧⌥⌘F7
Open search dialog	Ctrl+H	^H	**Find in path**	Ctrl+Shift+F	⇧^F
Copy lines	Ctrl+Alt+Down	⌥⌘↓	**Duplicate lines**	Ctrl+D	⌘D
Extract local variable	Ctrl+Alt+L	⌥⌘L	Extract variable	Ctrl+Alt+V	⌥⌘V
Assign to field	Ctrl+2,F	⌘2,7	Extract field	Ctrl+Alt+F	⌥⌘F
Show refactor quick menu	Ctrl+Alt+T	⌥⌘T	**Refactor this**	Ctrl+Shift+Alt+T	⇧⌥⌘T
Rename	Ctrl+Alt+R	⌥⌘R	Rename	Shift+F6	⇧F6
Go to line	Ctrl+L	⌘L	Navigate to line	Ctrl+G	⌘G
Structured selection	Alt+Shift+Up/Down	⇧⌥↑↓	**Select word at caret**	Ctrl+W/Ctrl+Shift+W	⌘W/ ⇧⌘W
Find next	Ctrl+J	⌘K	Find next	F3	F3
Show in	Ctrl+Alt+W	⌥⌘W	Select in	Alt+F1	⌥F1
Back	Ctrl+[	⌘[	Back	Ctrl+Alt+Left	⌥⌘←
Forward	Ctrl+]	⌘]	Forward	Ctrl+Alt+Right	⌥⌘ →

The Keymap

Key bindings is something that gets welded into the backbone of developers—this is probably one of the hardest parts about migrating to one IDE from another. The most painful thing is when you hit a shortcut that performs a completely different (and often harmful) action than what you expect. If you are Eclipse user, try hitting **Ctrl+D** / **Cmd+D** in IntelliJ IDEA: you will realize the issue immediately!

Of course, you can change the bindings of your keymap in IDEA. This is as simple as pressing **Ctrl+`** and selecting a preferred keymap. You can select between Eclipse, NetBeans, Emacs key bindings and some more.



Even though you can change the key bindings to the ones from the IDE that you're familiar with, it doesn't seem to be a good idea.

"What I screwed up when trying IntelliJ IDEA is that I changed to Eclipse shortcuts and then I didn't know what the key for Alt+Enter is."
ERKKI LINDPERE - Typist

"I went over keymap and assigned the most wired key-bindings from my Eclipse days. Especially if they didn't collide with anything important."
OLEG SHELAJEV - Product Engineer

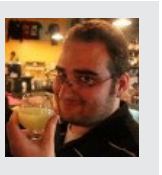
As we figured, the best strategy is to stick with the default key bindings and only change those handful of shortcuts that are really welded into your muscle memory and you can't help yourself but hitting the wrong keys.

There are many shortcuts in IntelliJ IDEA, for navigation, editing, refactoring and more. In the following section, we'll go over the most important key bindings that you most likely will need.

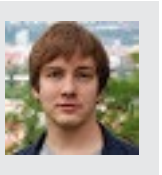
Before we proceed to the main part, let's bring up the shortcuts that cause the most annoyance to ex-Eclipse users.

CTRL+W / CMD+W

One single shortcut that probably annoys Eclipse users the most is the *block selection* shortcut. **Ctrl+W** / **Cmd+W** is used to gradually expand the selection of code.



"I wish Ctrl+W did what I expect it to do :("
ADAM KOBLENTZ - Senior Pre-Sales Engineer



"You should be aware of the auto-save feature and know what are you doing to enjoy it.
For example, our small Play 1 application tries to reload everything when IDEA saves file (because it works on the source code level). In the same way I suspect any interpreted language behaviour somewhat cripples the usefulness of autosave."
OLEG SHELAJEV - Product Engineer

"We also are now trying to promote OSX 10.5+ keyboard binding on OSX to make OSX users feel more at home."
HADI HARIRI - Developer

Yeah, indeed, **Ctrl+W** is usually used to close a tab or a window of an application. And IDEA decided to reserve this shortcut for something else: code selection.

But hey! The reason you want to use **Ctrl+W** / **Cmd+W** is actually irrelevant if you learn the zen of not closing the tabs — we'll cover that in the **Dealing with Tab**s part later. If you develop a habit when you don't have to close the tabs, you can then use **Ctrl+W** / **Cmd+W** for something else, i.e. for code selection.

CTRL+D / CMD+D

This is, arguably, the second most-annoying shortcut. In Eclipse, this shortcut is used for deleting the line. In IntelliJ IDEA, 'D' stands for 'duplicate'. To delete a line, use **Ctrl+Y** / **Cmd+Y** instead.

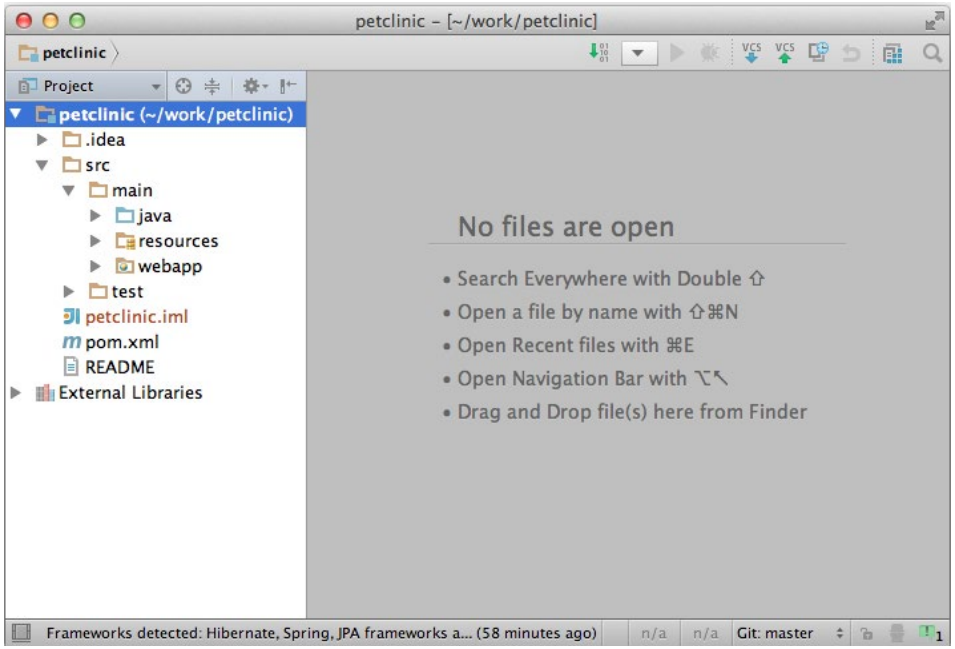
CTRL+S / CMD+S

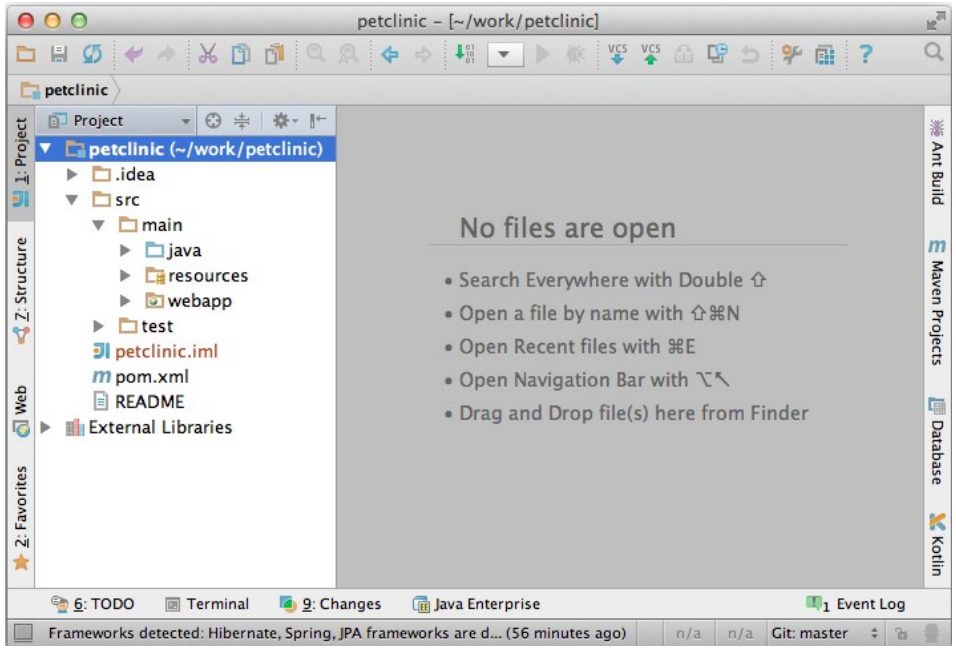
You do not have to save changes in IDEA: autosave works incredibly well. Just forget about saving the files as if it is a bad habit. **Stop. Hitting. Ctrl+S.**

Navigation

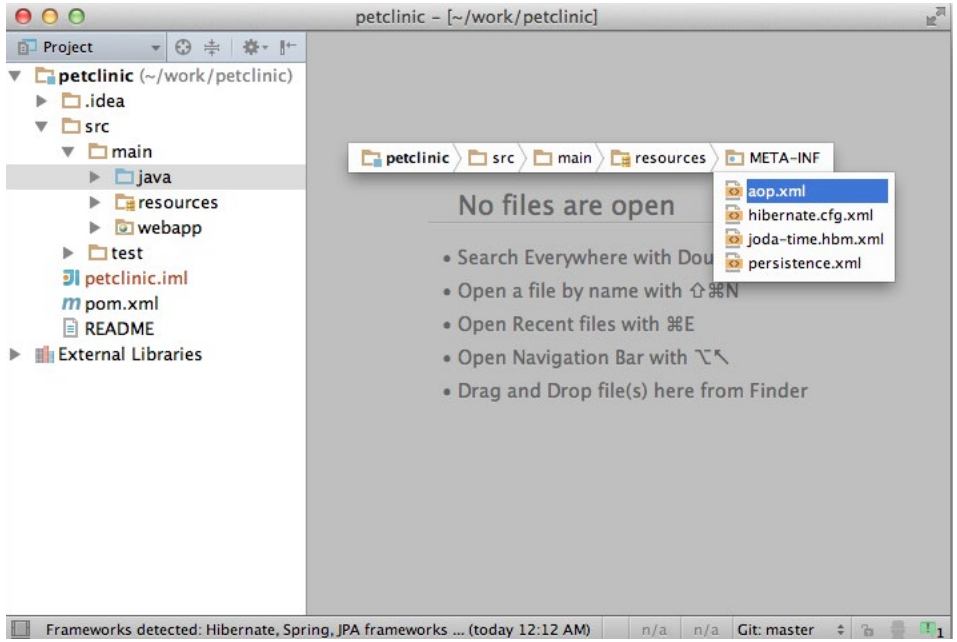
The default look of IntelliJ IDEA is quite minimalistic. Starting from version 13 it doesn't even include the toolbar in the default UI configuration.

If you're a new user and just starting to learn the IDE, you would probably want to **switch on the toolbar** as well as the **tool window** buttons. However, these control elements take up some space from the working area and as you learn the navigation shortcuts, you will most likely hide these elements after a while.





Even more: the navigation bar with the launch buttons that is enabled by default could be hidden too and accessed later with shortcuts.



*Navigation bar can be invoked via **Alt + Home***

Obviously, with all the tool controls closed you need to have access with shortcuts and that's what makes navigation in IntelliJ IDEA quite pleasant.

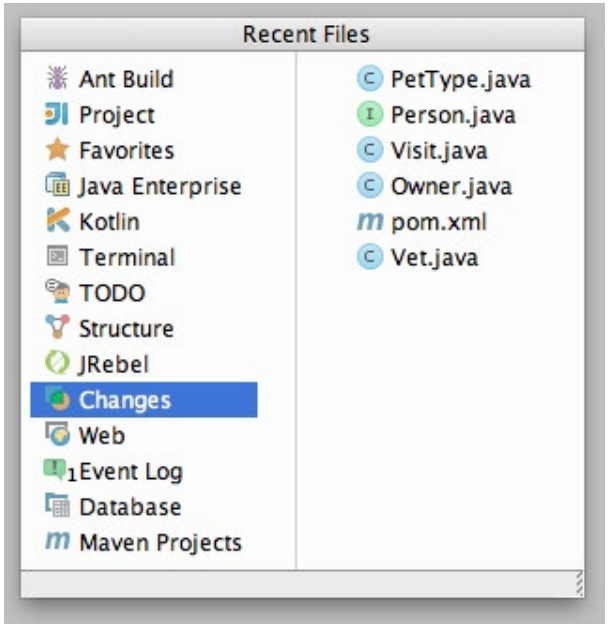
Now let's get our hands dirty. Assume that the our IDE workbench is in default setup and start discovering the navigation tricks.

TOOL WINDOWS

Tool windows provide an insight to all kinds of project aspects. For instance, there's the Project view, *Structure view, Maven Projects view*, etc. So, tool windows are instrumental to the "I" in "IDE".

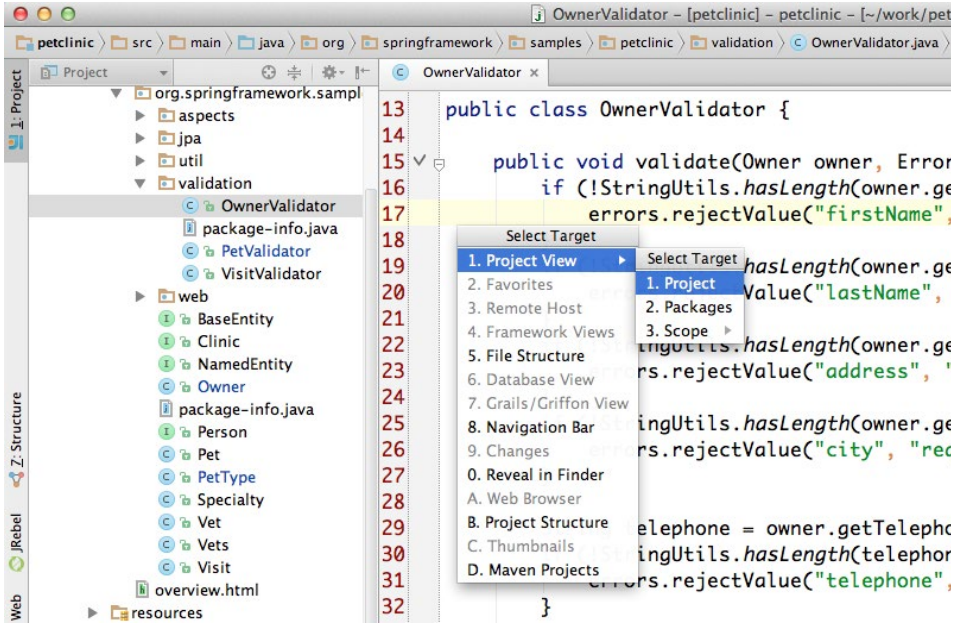
First, to quickly access these views, some of the tool window buttons have numbers and can be accessed with **Alt+NUMBER** / **Cmd+NUMBER** shortcuts. As project view is probably the most important one, then **Alt+1** / **Cmd+1** is a reasonable shortcut for this matter.

Recent Files popup, invoked via **Ctrl+E** / **Cmd+E**, is also handy for navigating between tool windows.



Also the *Switcher* popup, invoked with **Ctrl+Tab**, could be used for the same purpose. However, the mechanics of navigation are a bit quirky in *Switcher*, thus navigating via the *Recent Files* popup is more convenient.

Sometimes, I need to navigate from the file in editor to a tool window related to the current file. **Alt+F1** opens a special popup that can be used to jump to any of the tool windows that are relevant. Probably the most common case is when you want to find out where the current file is located in the project structure. For that, **Alt+F1** and hit **Enter—**and the currently open file will be highlighted in the project tree.

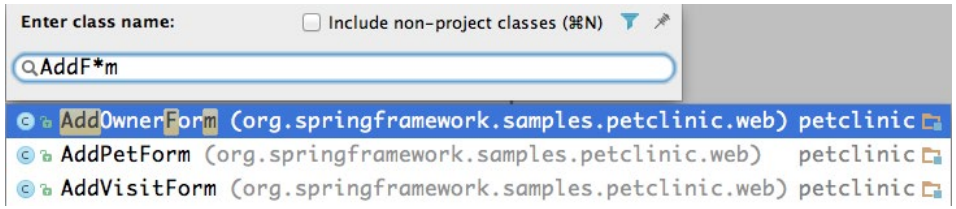


The official documentation provides sophisticated information about the settings for the tool windows in IntelliJ IDEA
http://www.jetbrains.com/idea/webhelp/intellij-idea-tool-windows.html

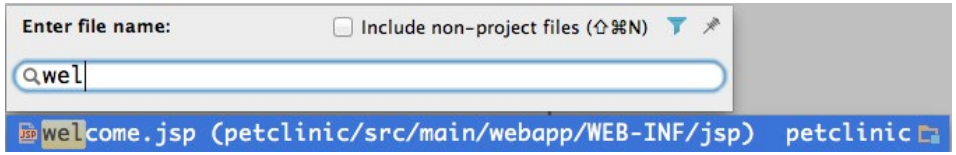
BASIC NAVIGATION

Navigation through the source code is one of the activities that we programmers execute on a regular basis. Let's see what the navigation features are in IntelliJ IDEA.

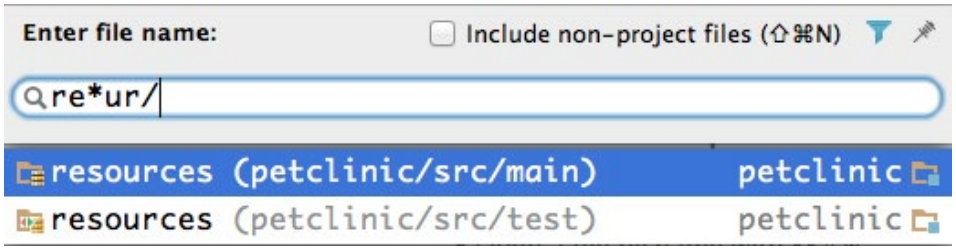
Probably the most popular navigation feature is the ability to navigate to a particular class, invoked with **Ctrl+N** / **Cmd+N**.



Similarly, the *Navigate to File* action is accessible with **Ctrl+Shift+N** / **Cmd+Shift+N**.



This action can actually be used to navigate to a folder in the project tree by adding a slash to the end of the search pattern.



Next, you can also *Navigate to Symbol*, **Ctrl+Shift+Alt+N** / **Cmd+Shift+Alt+N**, to navigate to a specific class, method or a field. I find this action to be a bit slow however, and therefore I don't use it as often.

You might have noticed a pattern in the shortcut definitions by now:

Action	Windows	Mac (Mac OS X layout)
Navigate to Class	Ctrl+N	Cmd+N
Navigate to File	Ctrl+Shift+N	Cmd+Shift+N
Navigate to Symbol	Ctrl+Shift+Alt+N	Cmd+Shift+Alt+N

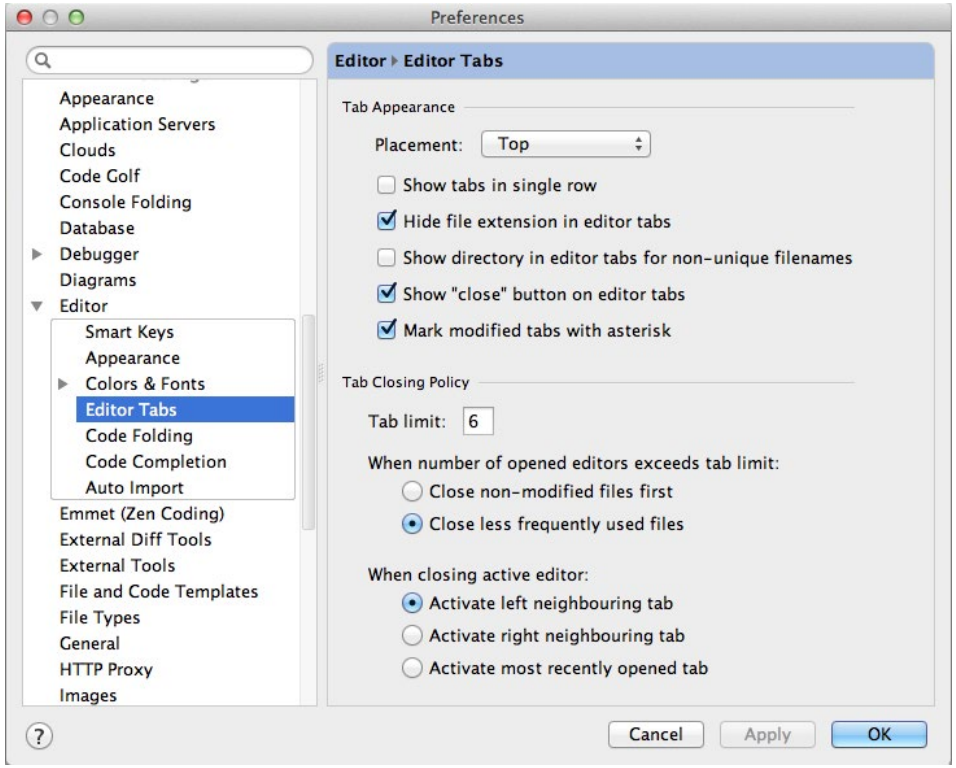
> **Protip:** Note that for Mac there are two keymaps available: **Mac OS X** keymap, and **Mac OS X 10.5+** keymap, which is more like the Eclipse keymap. So, you might want to switch to it.

Arguably, the most useful shortcut is the most accessible one, with via the **Ctrl** / **Cmd** key. Then, by adding the **Shift** key to the mix you access the second useful shortcut. And by adding more keys to the combination—**Alt**, in the case above—you proceed to the advanced actions.

DEALING WITH EDITOR TABS

Before getting to more interesting navigation, it is a good idea to talk a look on the editor tabs behavior.

Let's go to **Settings** → **Editor** → **Editor Tabs**.



My zen for the editor tabs is that there shouldn't be more than one row of tabs in the editor view, but at the same time I do not want the tabs to start hiding once they don't fit in one line. So I usually set the tab limit to a fairly small number, depending on the screen size, and leave the tabs dismissing policies to whatever is default.

By the way, the behavior of the tabs closing is somewhat different from Eclipse, so you might want to change that.

"Default strategy for closing tabs in idea is peculiar at least. 10 open tabs with average java class names, that are not arranged into a single row, take half of the screen on my laptop and annoy me enough to be the first thing I change after a fresh install. Other than that, settings are flexible enough to allow tweaking for any sensible taste."
OLEG SHELAJEV - Product Engineer

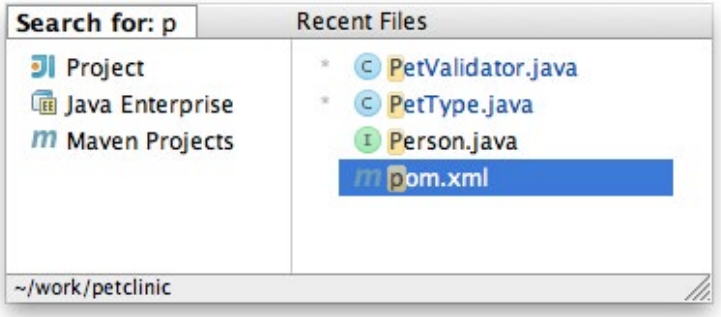
In general, navigation between editors rarely involves clicking on tabs. So I prefer a minimal in-your-face appearance: no package names, no ugly stars for unsaved files, no close buttons, single row, close unused tabs often, open most recent tab to preserve linearity of movements history.

MORE NAVIGATION TO FILES

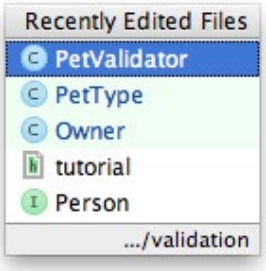
You may be confused by now about why you would restrict yourself on the number of tabs opened. In big projects, you might need to navigate through a lot of files and by closing the tabs would would have to find and open them again. So that seems inconvenient.

But have no fear! There is a better solution for navigation that I suggest you get familiar with...

IDEA has some actions that are very handy for navigation when you have to keep yourself in the context of many files. **Ctrl+E** / **Cmd+E** pops up the *Recent Files* list. The default number of files kept in the buffer is 50.



BTW, you can do a quick search in the *Recent Files* popup as well! Just start typing and the search will filter out the irrelevant entries.



If you know that you have edited the file recently, there's a bit better option that you can use - *Recently Edited Files,* accessed with **Ctrl+Shift+E** / **Cmd+Shift+E**.

Arrows can be used for navigation between tabs, but also for navigating the cursor position history.

Ctrl+⟶ and **Ctrl+⟵** allows you to navigate between tabs as they are opened in the editor. That's a pretty basic navigation feature. However, IDEA maintains the history of the cursor positions within the files as we have been browsing through the code. So it is possible to get back to the position in the code even if we don't remember which exact file it was. Use **Ctrl+Alt+⟶** and **Ctrl+Alt+⟵** for navigating over the cursor positions history.

Über Actions

Learning all the shortcuts at once can be quite overwhelming. You have to exercise them a lot to get comfortable with the new keymap and bindings. However, there are a few shortcuts that you **absolutely have to remember!**

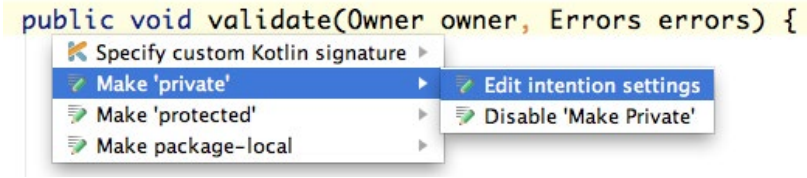
ALT+ENTER

Alt+Enter in IntelliJ IDEA is as universal as **Ctrl+1** / **Cmd+1** in Eclipse. If fixes things, but it can also suggest intentions "out of the blue"!

Whenever something is highlighted as a warning or error, use **Alt+Enter** and IDEA will help you out. But you can also hit **Alt+Enter** in almost any part of the code and a little popup will suggest actions that you can perform within the current context.

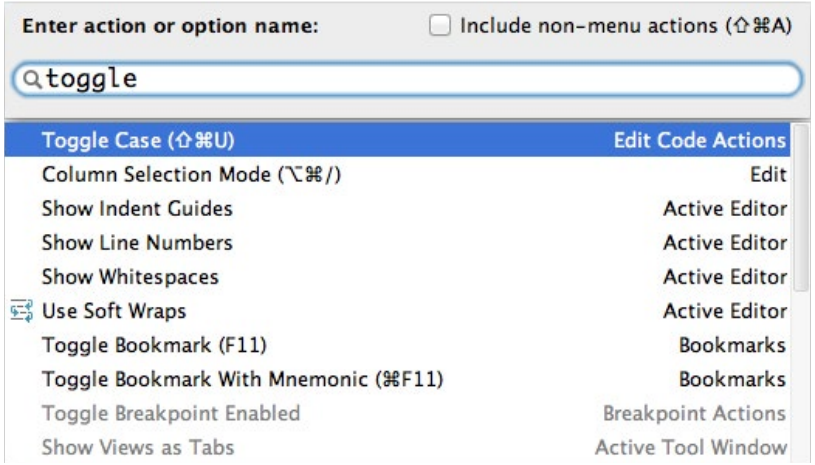
"My recommendation,
try hitting Alt+Enter more often."
OLEG SHELAJEV - Product Engineer

Another purpose of **Alt+Enter** is to invoke *intentional actions*. For instance, place the a cursor to a method modifier, hit **Alt+Enter**, and the popup will suggest you to convert the modifier from *public* to *private*, *protected* or make it *package-local*. Plugins can also contribute to intentions and add the specific actions to the context.



Read more about the intention actions in the official documentation:
http://confluence.jetbrains.com/display/IntelliJIDEA/Intentions

But what if you simply cannot remember all the shortcuts? Have no fear! You can find the actions by name as well as the settings, using the **Ctrl+Shift+A** / **Cmd+Shift+A**. This popup can be quite useful because you probably don't have all the actions assigned with shortcuts.

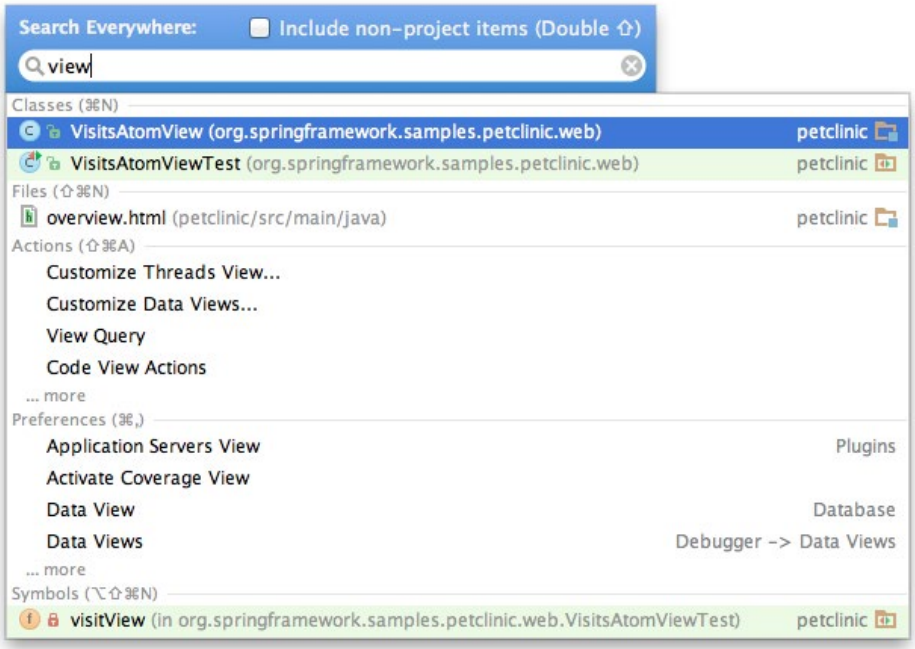


SEARCH EVERYWHERE

The newest addition to IntelliJ IDEA super-shortcuts is *Search Everywhere*. Double clicking the **Shift** button will invoke the relevant popup. *Search Everywhere* combines several shortcuts in one:

1. At the very start, it shows the list of recently opened files, same as **Ctrl+E** / **Cmd+E**.

2. Next, when you start typing, it starts displaying the results for classes, files, members, actions and IDE settings.

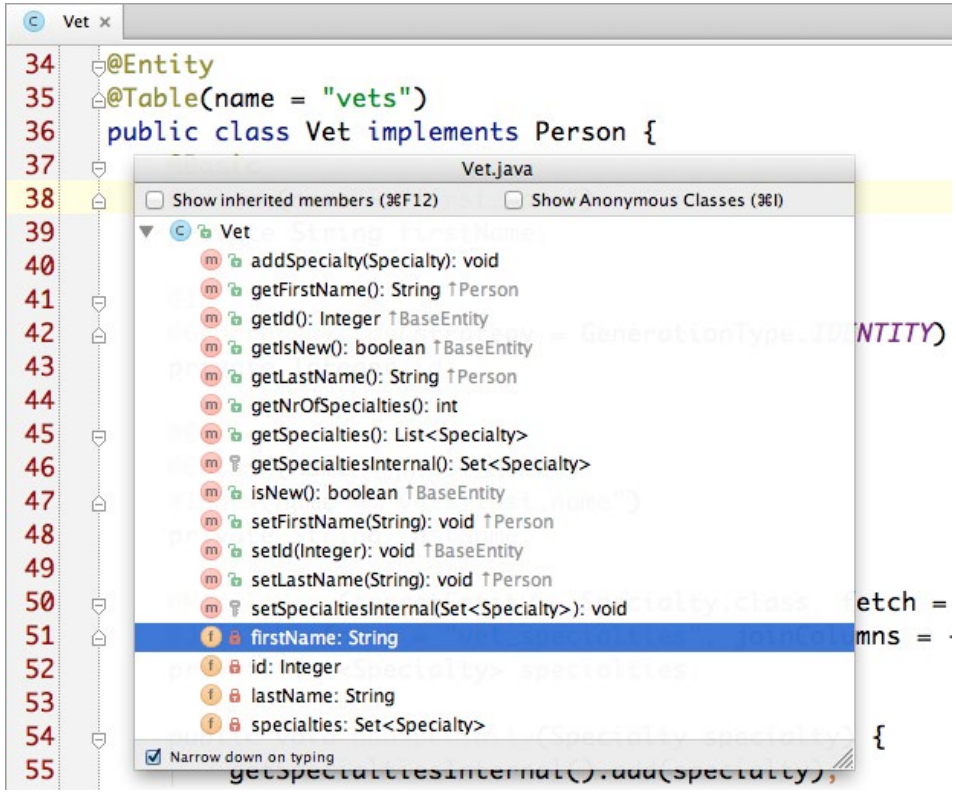


For you Eclipse users out there, *Search Everywhere* reminds me of **Ctrl+3** / **Cmd+3** in Eclipse, however it behaves a little bit differently.

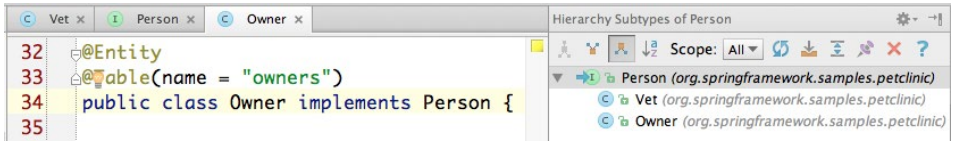
BROWSING THE CODE

There are plenty of shortcuts for navigating over the source code in IntelliJ IDEA. There are shortcuts used for file navigation, but they are definitely not enough for a professional software developer.

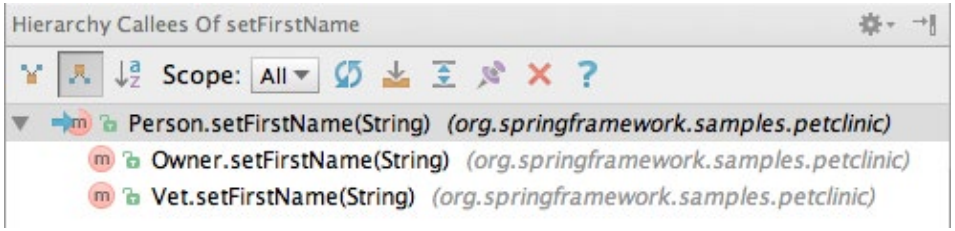
Within the opened Java class, you can easily navigate to the specific method using the **class outline** popup, invoked via **Ctrl+F12** / **Cmd+F12**.



To inspect the type hierarchy of the type/class that is under the cursor, hit **Ctrl+H** to open the *type hierarchy* tool window with the relevant class graph:



Ctrl+Alt+H does the same for the *method call hierarchy*.



"Compared to Eclipse, I'm actually missing the ability to see the call hierarchy for the fields. In IntelliJ IDEA I can view the call hierarchy if I invoke the action on a getter to the field, but not if I try doing the same on a field itself."

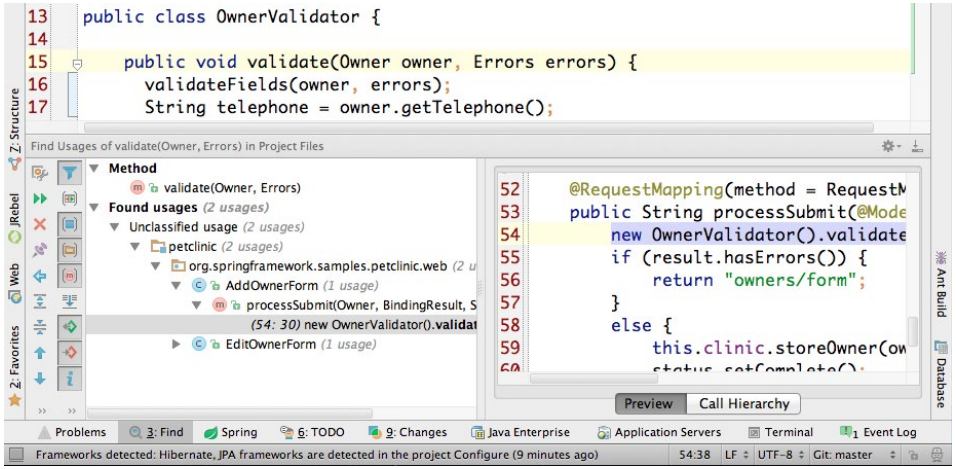
PAVEL GRIGORENKO - Research Engineer

So, let there be more shortcuts!

One of the most common ways to navigate the code is to follow the references. **Ctrl+B** / **Cmd+B** will probably be your beloved shortcut for this purpose; it's same as **F3** in Eclipse.

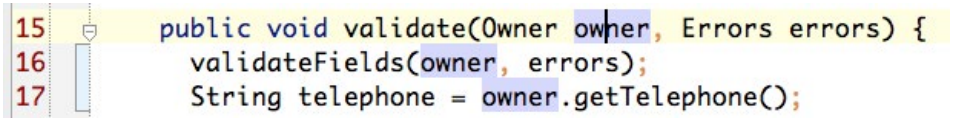
Finding the usages for a field, class, method or a local variable is another very common activity while browsing the source code. In IDEA, there are several shortcuts that are used for this task:

Use **Alt+F7** to find all usages and display the results in a *Find tool window*.

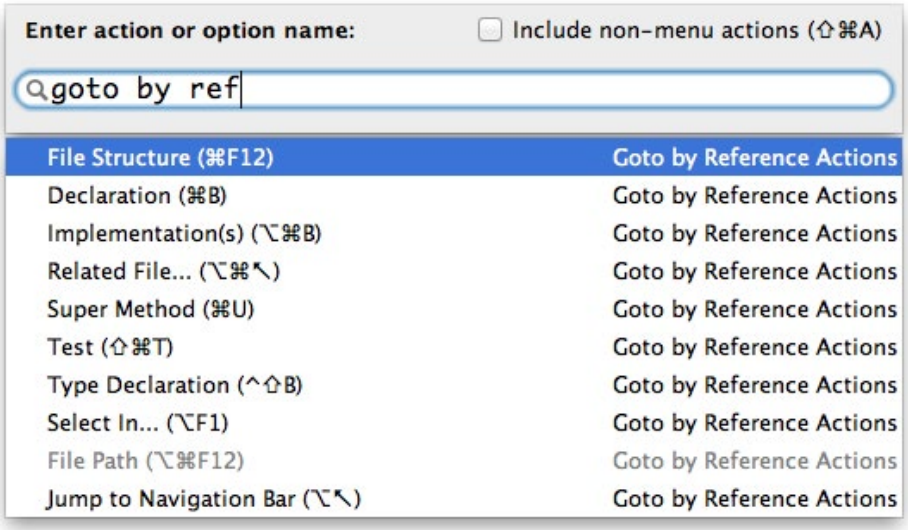


Ctrl+Alt+F7 / Cmd+Alt+F7 finds all usages and displays the results in a popup:

By using **Ctrl+Shift+F7 / Cmd+Shift+F7** you will highlight the usages within the current file:



To learn about all possible reference navigation shortcuts, hit **Ctrl+Shift+A / Cmd+Shift+A**, and type "goto by ref" and you will see a whole list of navigation actions.

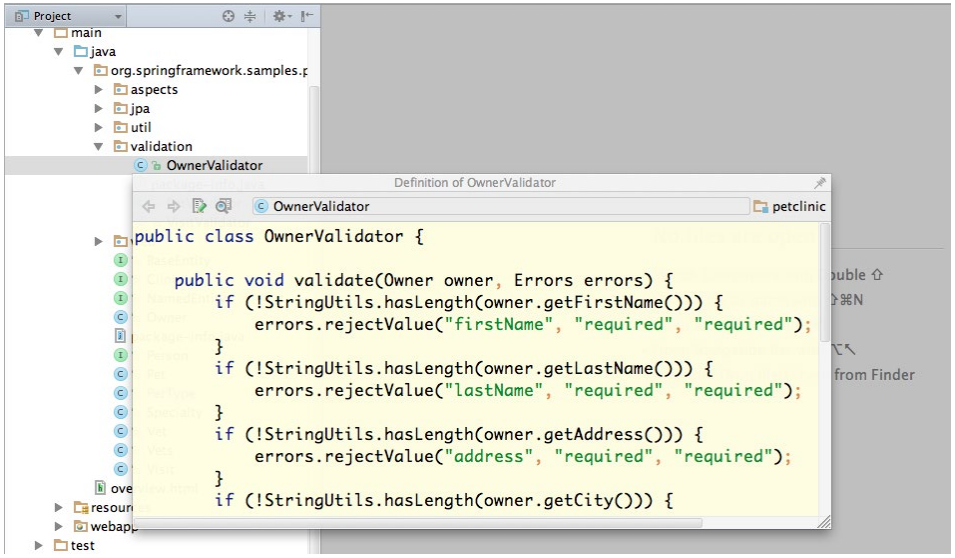


While typing into the search action popup, you will notice more navigation action types: *Goto by Name Actions*, which we covered previously, and *Goto Error/Bookmark Actions*, are for navigating to the warnings or errors in the currently opened file.

Often you don't really want to navigate anywhere, but just quickly lookup the method definition, or maybe Javadoc?

Ctrl+Shift+I allows you to lookup the source code almost from any place in the IDE. Obviously, you can lookup the code from the editor:

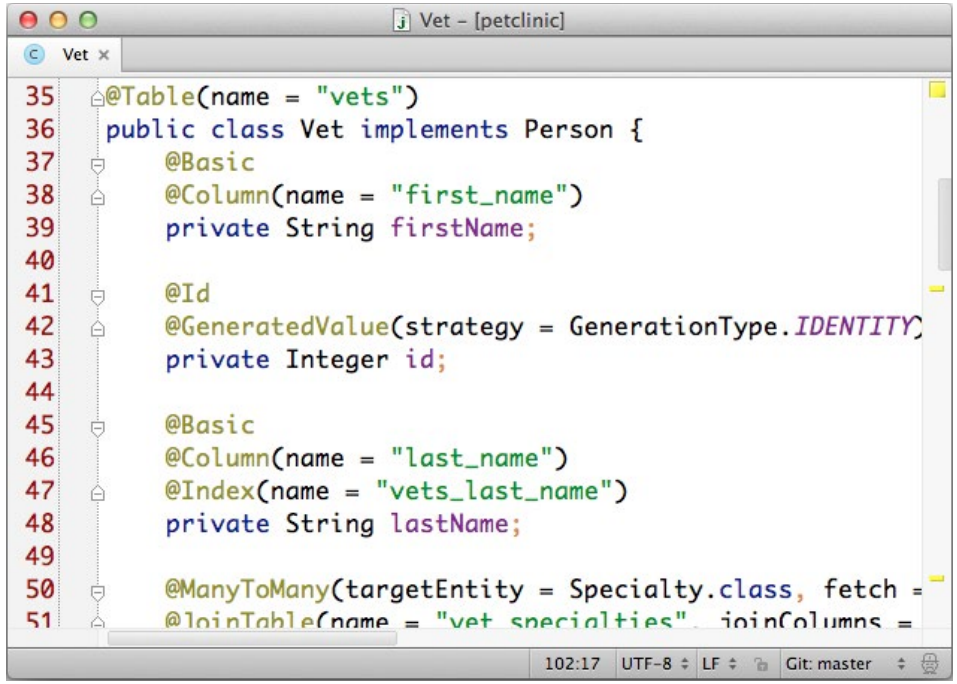
But you can also lookup the code from the *Navigation Bar* or even from the *Project tool window* without opening the file.



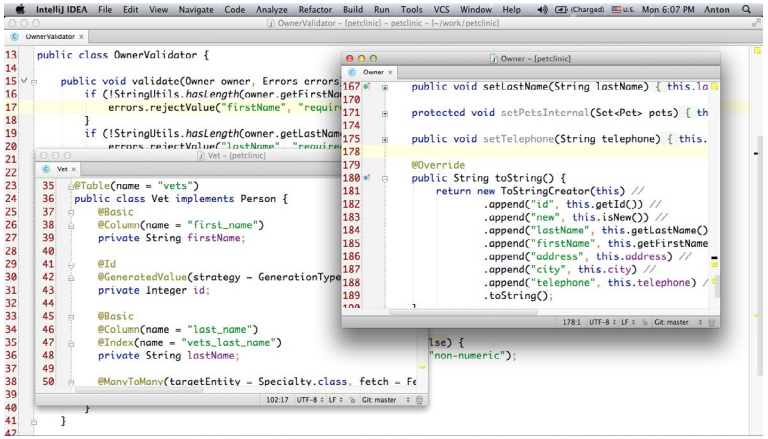
It previews *Javadoc* for the code under the cursor, use **Ctrl+Q** / **Ctrl+J**.

The Editor

The Editor is a central feature in IntelliJ IDEA. **Ctrl+Shift+F12 /Cmd+Shift+F12** hides all the tool windows and leaves only the editor area in front of you. If you have closed all the toolbars, it will look very ascetic and clean:



In addition, it is possible to detach the individual tabs of the editor which makes it convenient to use with multiple monitors.



SETTINGS

One of the things that most commonly surprises Eclipse users in IDEA's editor is the fact that you can place the cursor caret after the end of the line. This is default behavior, but you can switch it off: **Editor → Virtual Space → Allow placing caret after end of line**. You're welcome! ;)

Also, soft wraps are switched off by default, and it might be that you want to enabled these as well.

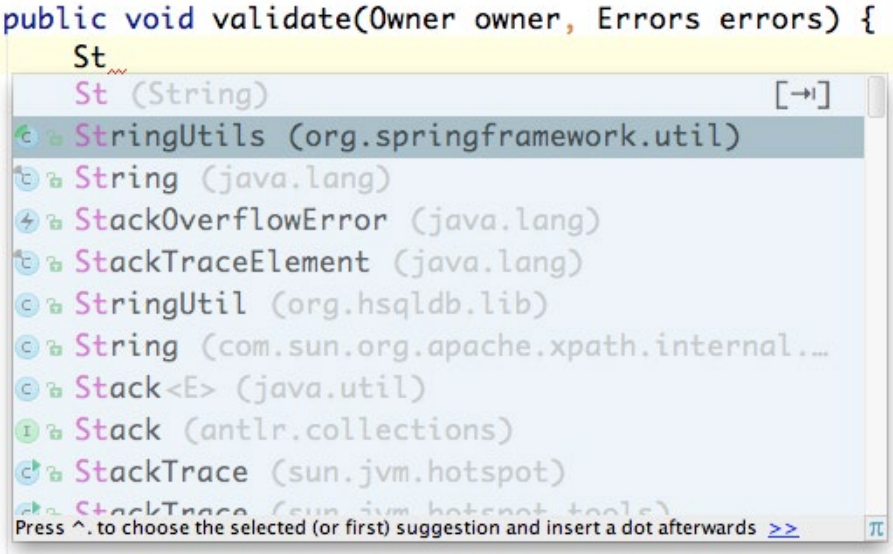
One of the interesting features that is also disabled by default is the use of "CamelHump" words (**Editor → Virtual Space → Use "CamelHump" words**). CamelHump is a feature that identifies parts of compound names composed according to CamelCase, where each part starts with a capital letter. So you will be able to navigate by the CamelCase, instead of the whole word.

There's a lot of different settings for the editor, so we won't be listing all of them here. Just take your time to adjust the settings if you feel like doing so.

AUTO-COMPLETION

I think that IDEA stands out with the number of auto-completions available. This is also something that you have to get used to after using Eclipse. With Eclipse, only **Ctrl+Space** is your main shortcut for auto-completion, but in IDEA there are multiple options for this.

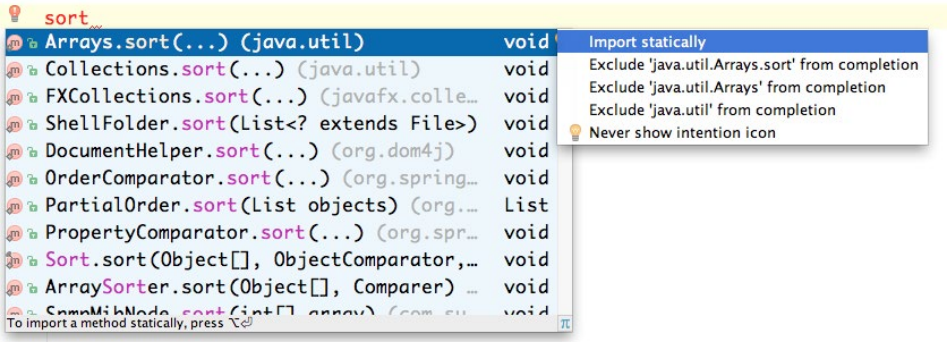
As soon as you type something, IDEA will start to suggest the options for completion. This is **instant completion**.



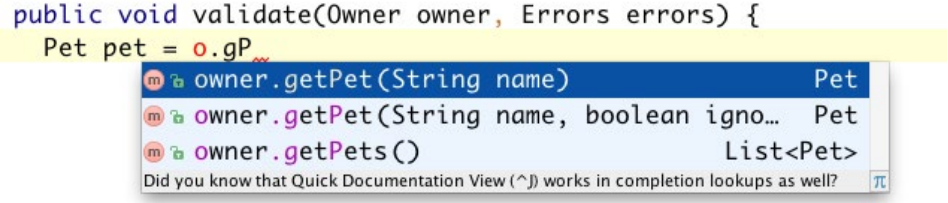
Instant completion appeared in IDEA relatively late, since it had existed in Eclipse and NetBeans for some time before it was finally implemented in IntelliJ IDEA. I'll be the first to admit that it seemed weird in the beginning, but now I wouldn't go back. Instant completion is a must!

Basic completion is something that all Eclipse users already know using **Ctrl+Space**. But since instant completion appeared, I don't remember using basic completion as often any more.

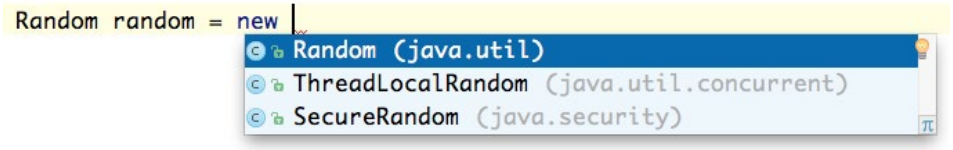
Basic completion can be used to suggest static methods. If you start typing the method name and invoke the basic completion twice, you get a list of the possible static methods that match the name. By hitting **Alt+Enter**, you can perform a static import for the selected method.



A very nifty feature of IntelliJ IDEA's completion is the ability to find candidates for **completing the incomplete symbols**. One may start typing a name of the variable, put dot, and continue typing the member name - the IDE will provide corrected completion for both - variable and its member.



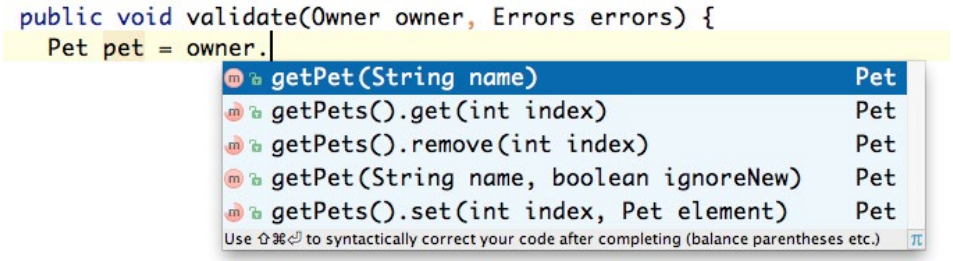
IDEA also features type-aware completion, which is often referred to as **smart completion**. **Ctrl+Shift+Space** filters the suggestion list and shows only classes, variables, fields and methods of the type—you know, the stuff you might expect in this context.



"It is actually confusing to have the separate shortcuts for completion. I tried assigning both the completions under the same shortcut, however, I'm not seeing the difference, but it seems to be doing what I need. Perhaps, this is just basic autocompletion which is good enough."
OLEG SHELAJEV - Product Engineer

CHAINED COMPLETION

If you need to autocomplete a type that is obtained via a chain of calls, you can call the smart completion twice. To satisfy chained completion, IDEA will scan all the methods, fields and classes that are available from the context and provide the list of options according to the expected type.



PROTIP FOR MAC OS X USERS

Mac OS X users will face a little challenge with the **smart auto-completion** shortcut. Namely, it appears that there is some bad interaction between the Chinese trackpad input method, and the default keyboard shortcut for smart auto-completion, which is **Ctrl+Shift+Space**.
OS X captures **Ctrl+Shift+Space** even when the Chinese trackpad input method is disabled. Here's the way to mitigate this problem:

- **System Preferences → Language & Text → Input Sources → Keyboard Shortcuts...**
- *Enable Chinese - Traditional*. This will enable the button. If this isn't enabled, the shortcut will not show up in the Keyboard pref pane, so you wouldn't be able to edit it.
- *Click the Keyboard Shortcuts...* button
- At the *Keyboard* preferences pane, change the *"Show Hide Trackpad Hand..."* shortcut to something improbable.
- Go back to *Language & Text* and disable *Chinese - Traditional*.

Done! **Ctrl+Shift+Space** will now perform as it should!

COMPLETING STATEMENTS & BLOCKS

Statement completion is often undervalued, but you might be really surprised how much time can it save! **Ctrl+Shift+Enter** / **Cmd+Shift+Enter** helps you complete the rest of that part of a statement. It automatically adds missing parentheses, brackets, braces and applies necessary formatting.

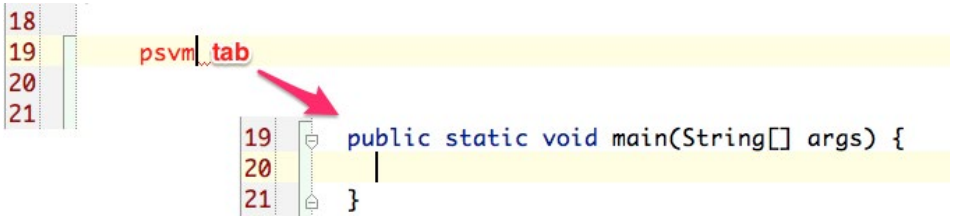
One complaint from the team was that statement completion should be invoked **Ctrl+Enter / Cmd+Enter** rather than incorporating **Shift** key into the combination. The argument is that statement completion is used more often than the *split line* action that occupies the Shift-less shortcut.

Above, we listed a number of completions available in IntelliJ IDEA. There are more! Depending on the context, you can make the autocompletes to work even more efficiently! Note that autocompletion works not only in the editor, but also in the other places of the IDE, i.e. in debugger views, and even in file dialogs

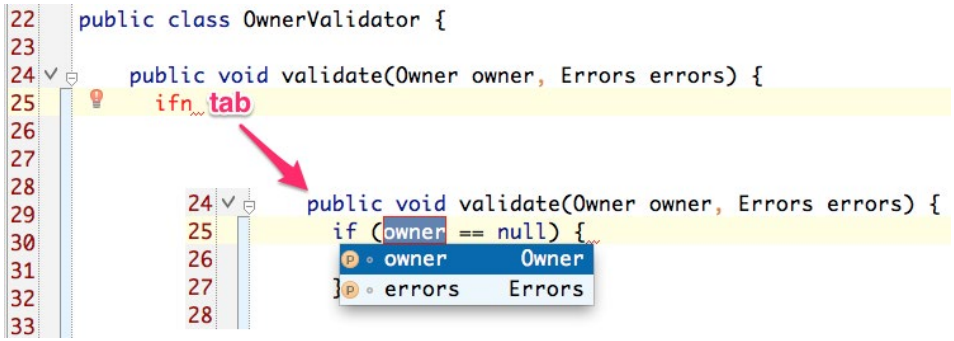
LIVE TEMPLATES

Templates help you write code faster. Usually, your IDE will provide you a set of predefined templates, but you're also allowed to add new templates.

Here's an example: type **psvm** and then hit **Tab**: the *public static void main(...)* method will be generated.

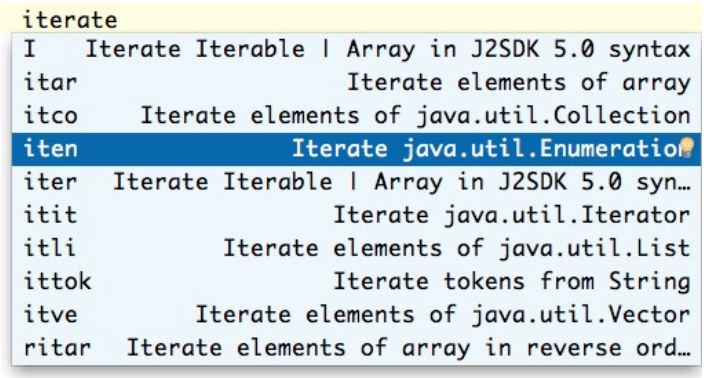


Here's another use case: check the method parameter for null equality. The corresponding if-statement can be generated with ***ifn*** template. So by typing "ifn" in the editor and pressing the **Tab** key the null-check-if-statement is generated.



Every template is associated with the context in which it can be applied. This is done so that the user doesn't generate statements that do not belong to the context. For instance, *psfs* would not expand in the middle of a method, since it is only applicable to the *declaration* scope.

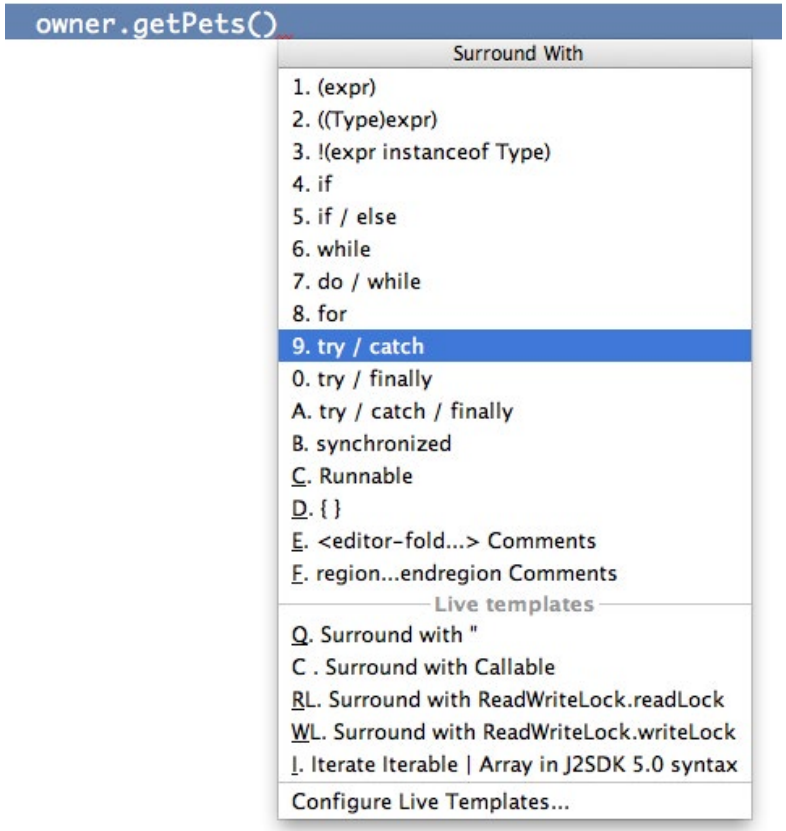
Ctrl+J will popup a list of all the templates. The list can be filtered based on the template name but also the by the description.



Templates exist in Eclipse too. So, what's different? The unique feature of the templates in IDEA is that the templates are backed with the available features from the IDE, e.g smart code completion and other useful functions. For more details, check out the **official doc**: http://www.jetbrains.com/idea/webhelp/edit-template-variables-dialog.html

SURROUND WITH LIVE TEMPLATE

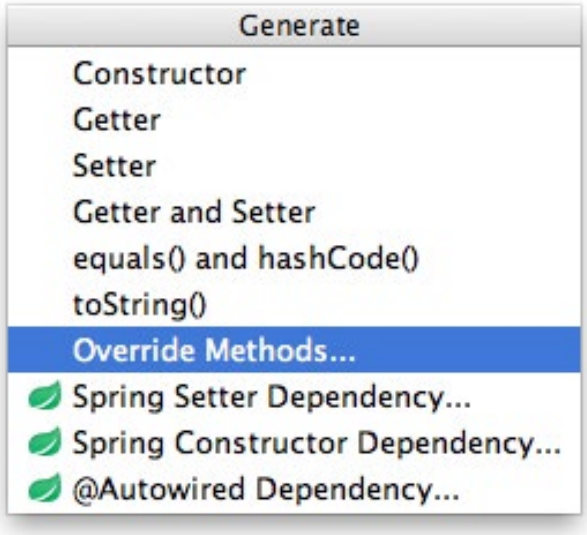
Templates can be used not only to generate code, but also for surrounding the existing statements. **Ctrl+Alt+T** / **Cmd+Alt+T** does the trick.



For instance, you can wrap the code with a try-catch block; this is probably the most common use case.

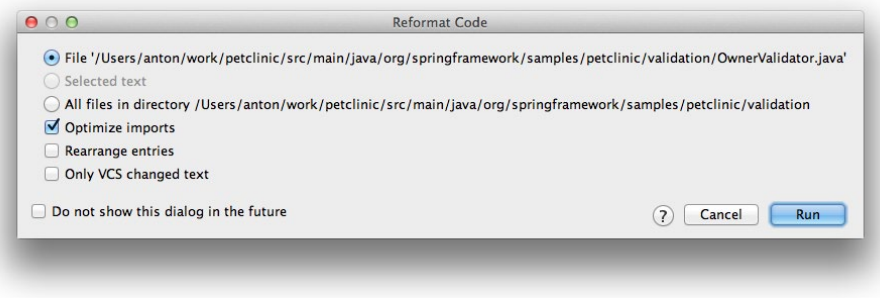
GENERATING CODE

Alt+Insert / **Ctrl+N** will be one of your beloved shortcuts if you are used to generating constructors, setters and getters, *toString()* method, etc. The same shortcut is used to create new entities, i.e. files, in the project tool window.



Code Formatting

To reformat the code, press **Ctrl+Alt+L / Cmd+Alt+L** and the appropriate dialog will appear. You can agree with the default options and choose not to show the dialog the next time.



Code style is very important when working in teams. If your team members are using different IDEs, there might be some issues. Even if the team members try to adjust the settings as much as possible to match each other, still chances are high that something will be different in the way the different IDEs format code.

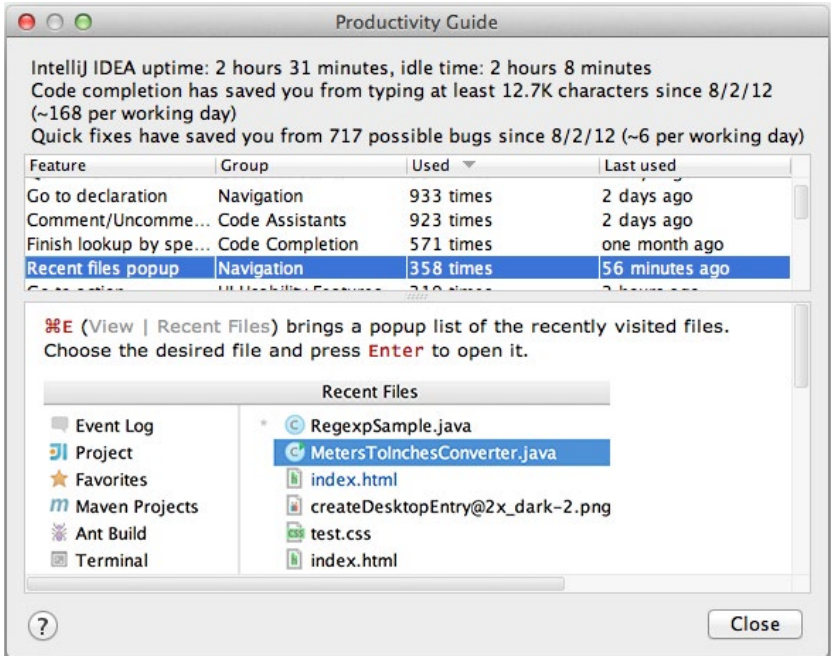
IMPORTING CODE FORMATTING SETTINGS FROM ECLIPSE

IntelliJ IDEA is capable of importing code formatter settings from Eclipse without the use of any plugins. All you need to do is export settings from Eclipse: go to Eclipse's **Preferences → Java → Code Style→ Formatter** and export the settings to an XML file via the Export All button. Then open IntelliJ IDEA **Settings → Code Style → Java**, click **Manage**, and import that XML file by simply clicking Import. Here is some more info on **formatter capabilities:** http://blog.jetbrains.com/idea/2014/01/intellij-idea-13-importing-code-formatter-settings-from-eclipse/.

PRODUCTIVITY GUIDE

That was a lot of information and a lot of shortcuts, but don't fear: there's even more!

Would you like to know if you're using IntelliJ IDEA effectively enough, and even get tips for being even more effective? Well, IDEA keeps track of your actions and collects the statistics that is turned into a nice report for your convenience.



Help → Productivity Guide

From the Productivity Guide report you can learn, which shortcuts you love and which you've never used before. Nice insight for improvement!

COMPILATION

Historically, IntelliJ IDEA did not compile code automatically after save operation. Instead, the user was supposed to press **Ctrl+F9** / **Cmd+F9** to run make process for the project. It does not mean that the whole project is rebuilt. *The compilation is incremental: IntelliJ IDEA keeps track of dependencies between source files and recompiles only if a file has been changed.*

Since IDEA v12, a function called off-process compilation mode was added. This can be enabled in **Settings → Compiler → Make project automatically**. With this, IDEA will save the file automatically, as well as compile it.

Note: *The save operation delay is 1.5 seconds by default, and compilation delay is 0.3 seconds after the save operation is complete. These settings can be altered, but not a subject of this guide.*

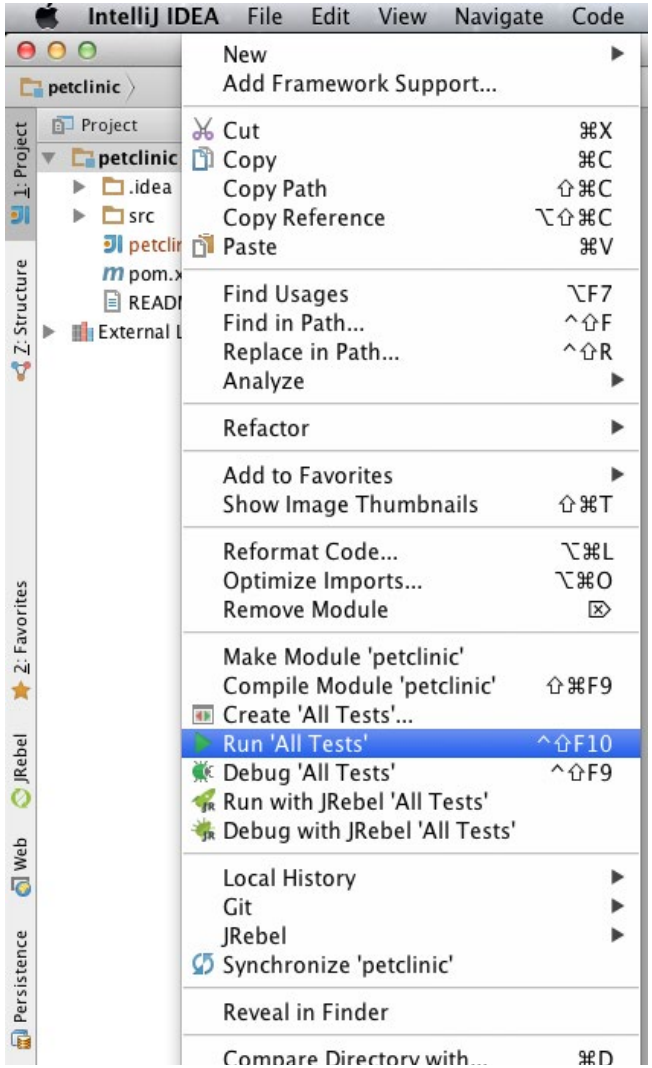
Here is the end of this chapter, but the next one goes into running tests, deployment to your application server and a bit about artifacts! Moving on...

CHAPTER III:
GETTING PRODUCTIVE WITH TESTS, DEPLOYMENTS AND ARTIFACTS

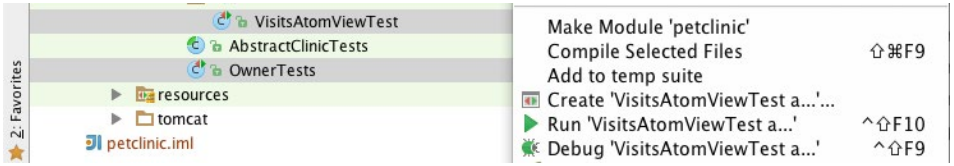
Now that we've covered project setup and a bazillion pages of keymap, navigation and editor tips/shortcuts, let's see if you're brave enough to get into testing and deployment ;-)

Running Unit Tests in IntelliJ IDEA

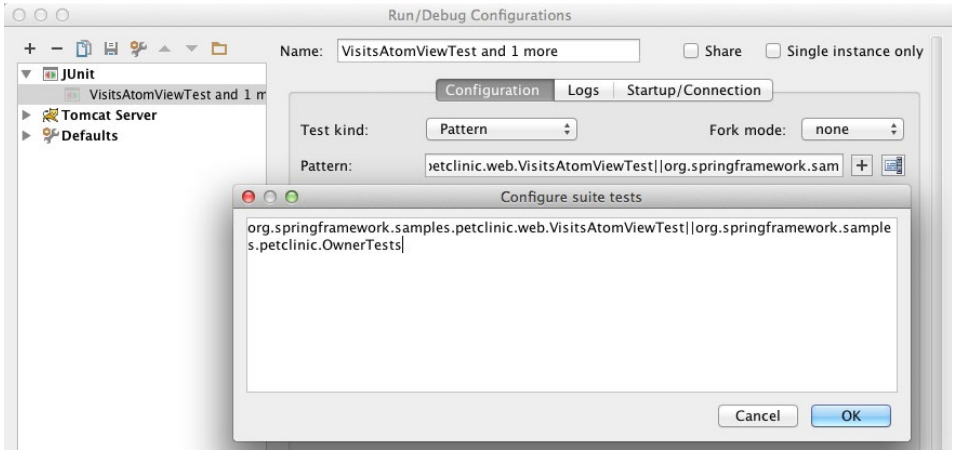
Executing unit tests in IDEA isn't very different from how you would do it in Eclipse. Right-click on a module, from the context menu select Run 'All Tests'. IntelliJ IDEA will execute all the tests that are located under the test folders.



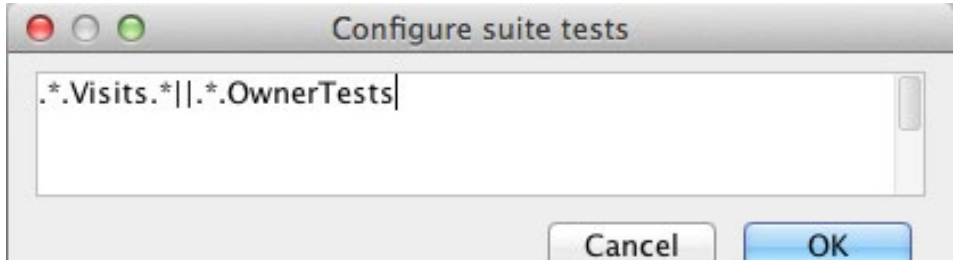
In the same way it is possible to select a single package, or a few test classes from different packages. This is especially handy when you would only like to run a limited set of tests locally.



For instance, in Petclinic project, one could select only *VisitsAtomViewTest* and *OwnerTests* classes in the project tree to execute the respective test set. Check out the run configuration that was created for the case when the tests were executed for the two test classes.

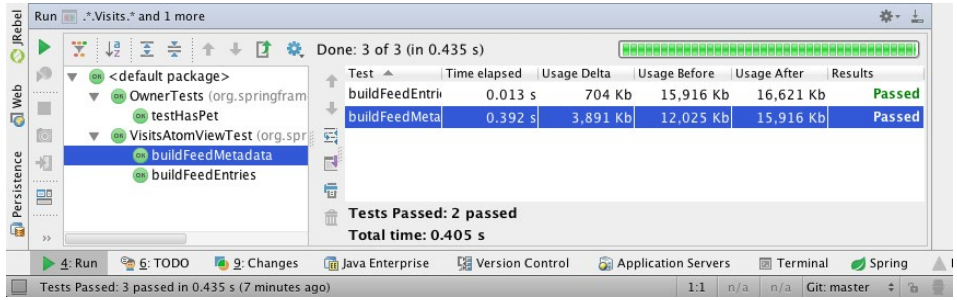


The Test *kind* combo-box allows you to specify what kind of test execution we would like to define for the particular configuration. The 'Pattern' value for the type of test would allow to specify an expression to match the names of test classes for execution. No need to specify exact names of the classes—just write a regular expression.



Of course, there are all the other *Test kind* options available for running the tests for a particular package, test class, or just a single method in a test class.

Test results are observed in the *Run tool window*.



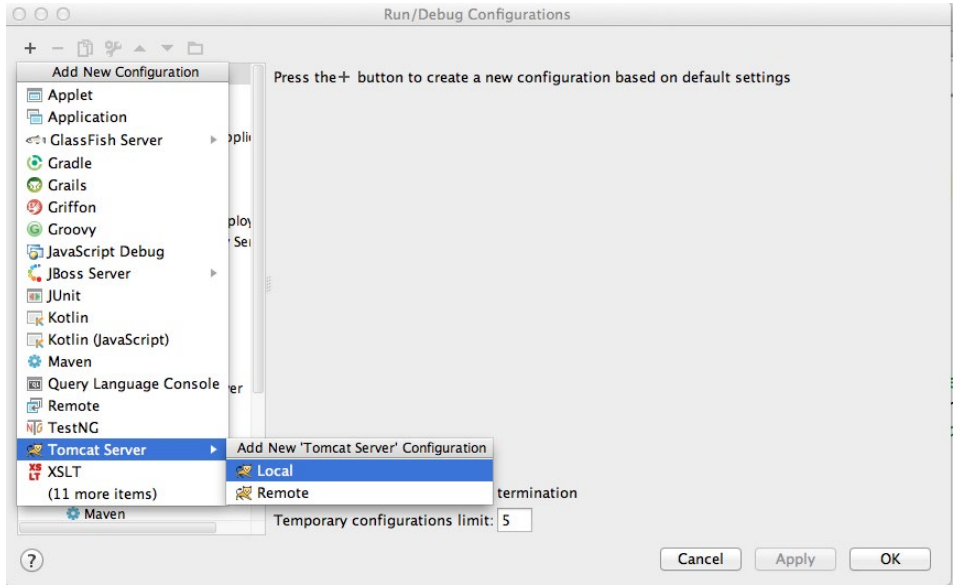
Deployment to Application Server

Let's now deploy the *Petclinic* app to a Tomcat container. First, we will need to configure the application server and then tell what do we actually want to deploy.

RUN CONFIGURATION

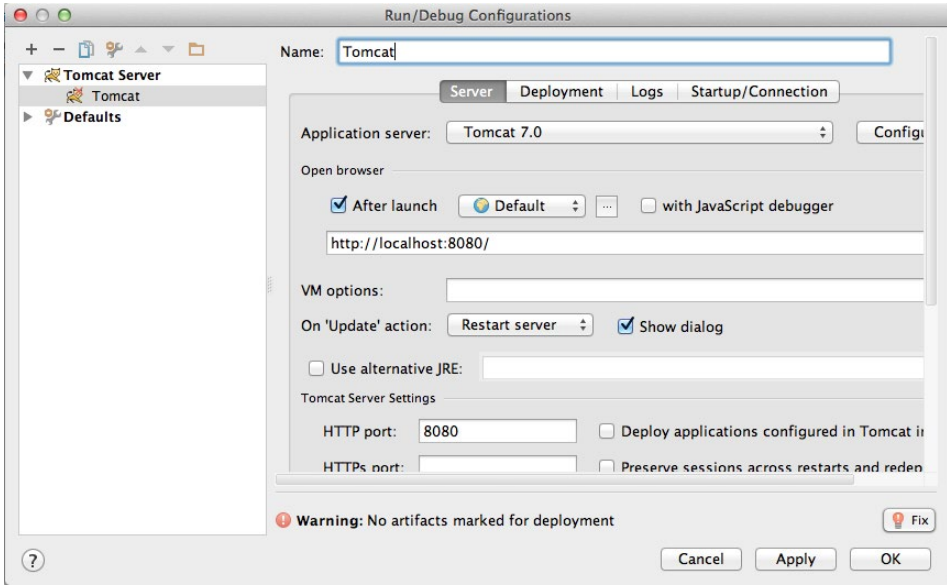
IntelliJ IDEA has the concept of Run Configurations, just like with Eclipse. Before you can run an application, you need to create a run configuration for it.

Open the Run Configurations dialog via **Run** → **Edit Configurations**.

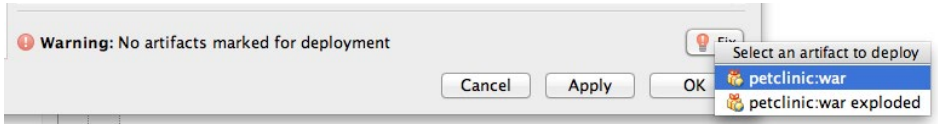


For every application server you can either create a *Local* configuration, or *Remote*. The *Local* configuration can be used if you intend to start the application server from the IDE. The *Remote* configuration type is good when the application server is started externally, either on the same machine or remotely.

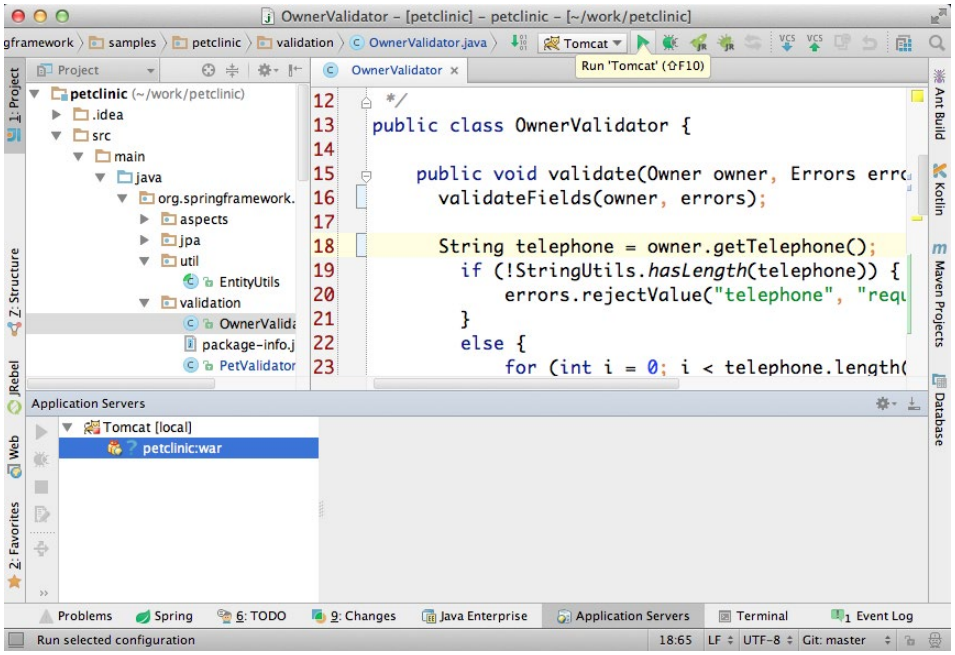
Once a configuration is created, the dialog will notify you that the deployment artifact is missing; notice the **Fix** button in the bottom right.



By pressing the **Fix** button, IDEA will automatically derive what kind of artifacts are possible to create and deploy from the project structure.



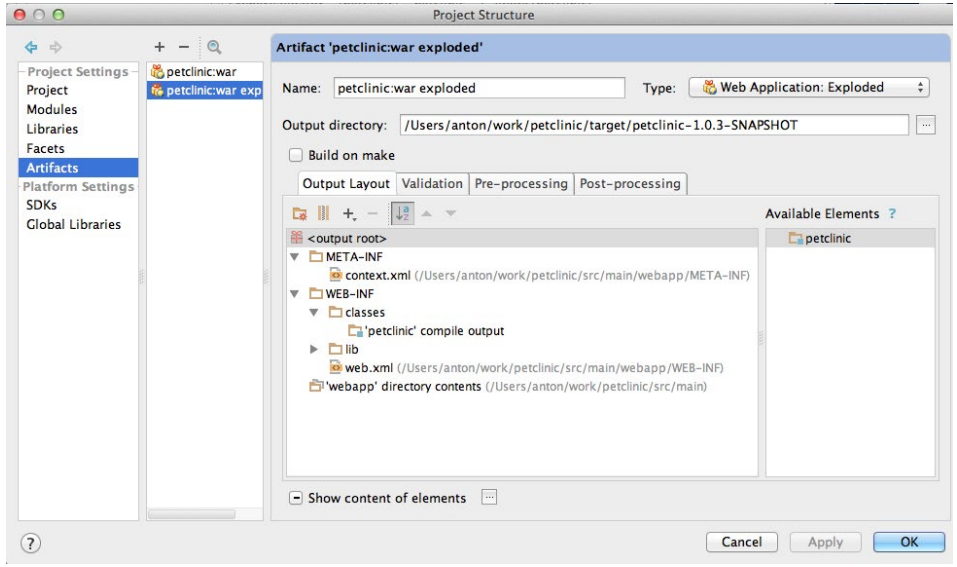
Once the deployment is configured, you will see the Application Server view in the main IDE window. It lists the servers and the artifacts that should be deployed. Now you can just hit **Shift+F10** to start the currently selected Run Configuration.



Artifacts

We have mentioned artifacts a lot, but haven't explained what exactly they are. When you compile your module, the compilation results are placed into the module output folder. As a result, you get the ability to run, test and debug your application. However, as soon as you want something additional, you have to define an artifact.

An artifact may be as simple as your module compilation results packed in a .jar, and it may also include compilation results from more than one module (in different formats like .war, .ear, etc).



For the "happy-path" scenario, as in our case with the *Petclinic* application, artifacts can be created semi-automatically; you just push the **Fix** button while creating the *Run Configuration*. This is because the sample application makes use of Maven and IDEA integrates well with that.

However, in case of more complicated scenarios, such as when the artifact structure doesn't match the project structure, then you have the ability to define exactly what should be the final result of the make process.

CHAPTER IV:
SUMMARY, CONCLUSION AND GOODBYE COMIC ;-)

Here is a nice TL;DR section for you lazy devs out there...complete with a comic that I made personally about how IDE users see other IDEs.

How do IDE users see each other?

Source: http://twitpic.com/dseptp

Final thoughts

I'd like to think of this guide as a resource you can return to repeatedly, including our little shortcut cheat sheet, courtesy of Andrey Cheptsov from JetBrains. There is so much more I could have written into this report—after all, there are a lot more interesting features that we did not cover, such as support for databases and web technologies, specific capabilities for Spring & Java EE, and more...

It's not an easy thing to switch IDEs, which is why I hope I covered all the most important aspects when migrating from Eclipse to IntelliJ IDEA. As a recap, here is what we covered:

Chapter I - Project structure and configuration, including folders, facets, and the terminology gap between IDEA and Eclipse regarding Modules and Projects.

Chapter II - This was the big boy of the report, covering keymap and key bindings, loads of navigation and editor settings, compilation, and a whole list of awesome shortcuts in a printable cheat sheet.

Chapter III - This is where we get truly practical with running tests and deploying your app to your application server, along with artifact definition.

When it comes to something as personal as choice of IDE, it's probably best to stick with what you know. But we developers are a curious type, and it's always cool to learn about new methods, tools and technologies. IDEs are one of those tools that we use more or less constantly throughout the coding day, and even an incremental increase in productivity or feature availability has a big effect. This guide is also designed to increase your productivity by highlighting the biggest differences in features and shortcuts that Eclipse users will need to get used to in IntelliJ IDEA. Now, go forth and be awesome!

Contact Us

Twitter: @RebelLabs

Web: http://zeroturnaround.com/rebellabs

Email: labs@zeroturnaround.com

Estonia
Ülikooli 2, 4th floor
Tartu, Estonia, 51003
Phone: +372 653 6099

USA
399 Boylston Street,
Suite 300, Boston,
MA, USA, 02116
Phone: 1(857)277-1199

Czech Republic
Osadní 35 - Building B
Prague, Czech Republic 170 00
Phone: +372 740 4533

This report is brought to you by:
Anton Arhipov, Oleg Shelajev, Oliver White,
Ladislava Bohacova