

USING ECLIPSE FOR JAVA DEVELOPMENT

A HELPFUL INTRODUCTION TO THE WORLD'S
MOST-USED IDE



From noob to ninja in 40 pages



TABLE OF CONTENTS

INTRODUCTION

INTRO AND HISTORY TO THE ECLIPSE IDE **1-2**

PART I

GETTING STARTED WITH INSTALLATION AND MAINTENANCE **3-15**

PART II

MAKING ECLIPSE YOUR OWN **16-28**

PART III

TIPS AND TRICKS FOR USING ECLIPSE LIKE A SUPER-NINJA BAD@\$ \$ **29-41**

SUMMARY

CONCLUSION AND A GOODBYE COMIC ;-) **42-46**

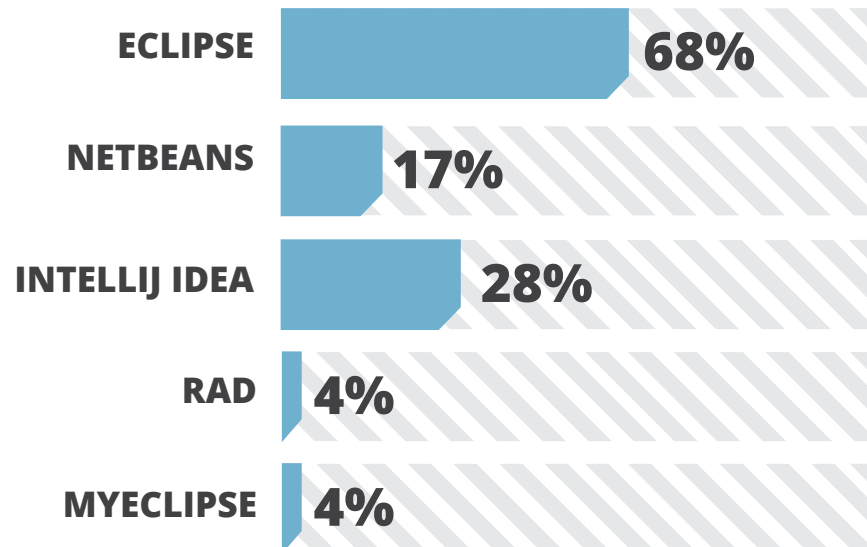
INTRODUCTION:

INTRO AND HISTORY TO THE ECLIPSE IDE

This report focuses on the most commonly used Java IDE currently available to mankind. Under the spotlight is Eclipse, the most widely selected, praised and complained about IDE.

Intro and history to the Eclipse IDE

The Eclipse IDE, according to a recent report on developer productivity, is used by roughly two-thirds of the 1800+ Java developers we talked to, making it a significant player in the IDE market. With the largest user base and a vast number of plugins and integrations into the development world, Eclipse is where most developers start off when it comes to writing code.



Eclipse is a self-described universal toolset for development, defined as a platform for building integrated development environments and tools for various languages. However that is a very broad description and sometimes it's nice to see the actual trees in the forest, so here's our take on it: *Eclipse is an extremely customizable Java IDE which supports several other languages and development platforms.*

Eclipse started off as an IBM Canada project back in 2001, later rolled into an open-source program with a consortium of stewards from leading companies. Since 2004, it is supported and maintained by the Eclipse Foundation, which is a non-profit organisation that is backed (i.e. funded by annual dues) by top industry companies, like Oracle, IBM, Red Hat, SAP, Google and ZeroTurnaround ;-)

The Eclipse Foundation not only keeps the infrastructure of the Eclipse IDE project running and helps set up transparent & maintainable development for projects that are willing to join the umbrella of Eclipse, but they also prioritize the care of the Eclipse ecosystem and community. The Eclipse Foundation actively markets all kinds of projects based on or using Eclipse which, combined with the availability of educational materials, makes Eclipse a solid choice when it comes to determining your next project's platform.



PART I:

GETTING STARTED WITH INSTALLATION AND MAINTENANCE

Read the manual! The first question about any particular kind of software in such a large and vague category as IDEs is about what it does that others don't. We will talk about that a bit later, but for now, let us be coherent and start with a more down-to-earth approach, like how Eclipse is distributed, what bundles to download and in general how to get started.

Eclipse bundles: Java EE, Java, C/C++ and more

Eclipse provides a platform to create powerful applications and has a huge ecosystem of plugins; therefore, it's easy for them to create archives that generically fit some problem space very well. So the easiest and most straightforward way to get started with Eclipse is to download a so-called 'packaged solution', which is basically a bundled archive that includes, is an archive that includes an Eclipse Runtime with preinstalled tools. As we can see, the Eclipse bundle for Java EE developers is the most frequently downloaded.



Eclipse IDE for Java EE Developers, 244 MB

Downloaded 1,032,013 Times

Tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn...



Eclipse IDE for Java Developers, 150 MB

Downloaded 458,831 Times

The essential tools for any Java developer, including a Java IDE, a CVS client, Git client, XML Editor, Mylyn, Maven integration...



Eclipse IDE for C/C++ Developers, 148 MB

Downloaded 271,736 Times

An IDE for C/C++ developers with Mylyn integration.



Eclipse IDE for Java and Report Developers, 275 MB

Downloaded 81,266 Times

Java EE tools and BIRT reporting tool for Java developers to create Java EE and Web applications that also have reporting...



Eclipse Modeling Tools, 289 MB

Downloaded 77,482 Times

This package contains framework and tools to leverage models : an Ecore graphical modeler (class-like diagram), Java code generation utility for.



Eclipse IDE for Java and DSL Developers, 266 MB

Downloaded 74,790 Times

The essential tools for Java and DSL developers, including a Java & Xtend IDE, a DSL Framework (Xtext), a Git client.

If you go after Eclipse for Java Developers, you'll get some more things, like Maven build tool integration and XML editor support. A bundle for Java EE developers adds around 100MB of additional features, and this installation will be aware of web application architectures and JPA, JSF, etc. It even comes with Mylyn support, which allows you to connect your Eclipse installation to task-tracking solutions like Jira, Github Issue Tracker, Bugzilla and more so you can resolve issues and work on other tasks without leaving the same working window.

There is also a mobile dev platform called [Android Developer Tools](#), or ADT. It can be installed to your existing Eclipse installation as a plugin, but it is rarely useful to have regular old Java projects mixed together with Android stuff, so the ADT bundle is more convenient.

ADT is a powerful bundle, and it includes tools to test and debug Android apps both in a simulator and on devices, plus it contains UI builders and supports native development too.



Platform runtime binaries and a quick sidenote on OSGi

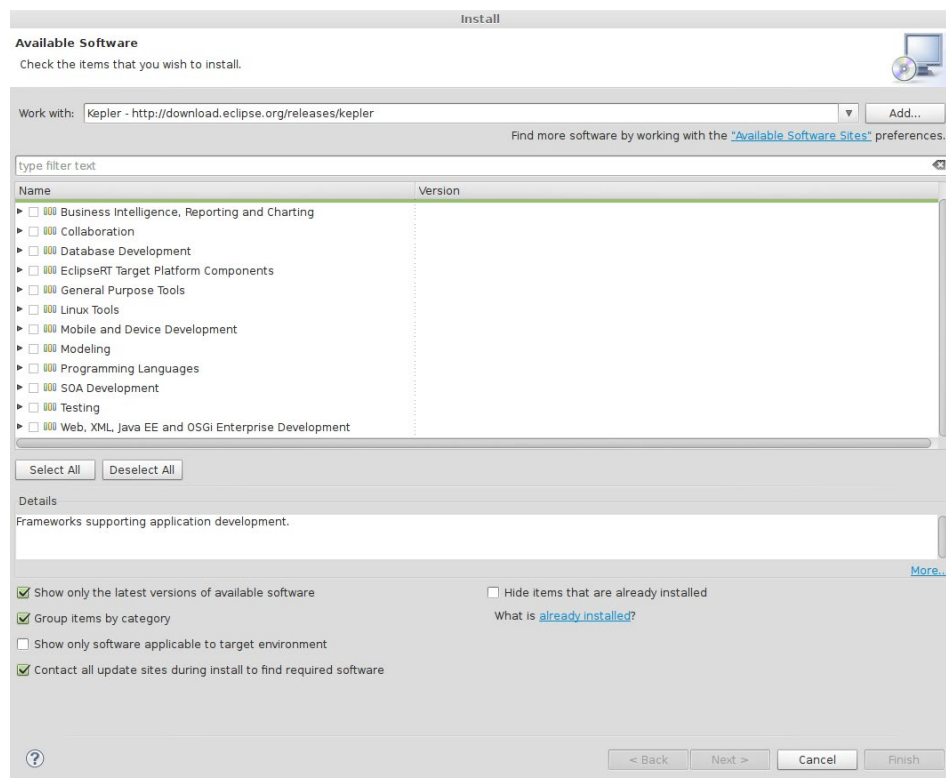
Platform runtime binaries are another way to get around Eclipse. If you don't want a pre-packaged solution and think that Eclipse maintainers will likely stuff it with things that you don't use anyway, you can get the bare-bones binary, which is just the platform to which you can add your own toppings as you like. The Platform Binary archive is just about 60 megabytes and it's pretty darn fast because it comes stripped of anything that can slow it down.

Since we all like to be fast in development, let's try simple and not-at-all scientific experiment: measuring start-up times of Eclipse Runtime binary vs a machine with Windows 7, JDK 7u40, Eclipse JEE. We timed this on an iPhone stopwatch with a steady hand, so don't judge us too harshly ;-)

As you can see, there is a difference in startup speed, however we are not sure if it's significant. Also note that the workspace was empty, but more often than not you'll have several open editors and files to index, so the difference could be more recognizable.

Eclipse	Cold Start		Warmed up	
	Until workspace select (s)	Until it is ready (s)	Until workspace select (s)	Until it is ready (s)
Eclipse JEE	16	36.5	4, 3.7, 3.8	17, 15.5, 15.7
Platform binary	7.8	17.4	3.3, 3.4, 3.2	11.5, 11.5, 11.3

If you decide to give Platform Binary a try, just go to **Help -> Install New Software...**, pick the release/update site for your version of Eclipse and you'll get a list of things that you can add to your Eclipse installation: programming languages packs, web development tools, modeling tools, all kinds of connectors and adaptors to work with external applications and so forth.



Real quick, a note about OSGi. OSGi is a module system and a dynamic runtime where modules (also called bundles) can come and go, but your code has to conform to the requirements of the module system — it is perhaps the only widely-used framework for the JVM that enforces real modularity.

Increased modularity will be really useful when you have:

- a complex set of dependencies
- lots of reusable pieces
- customizations of your software in different contexts
- a chaotic mess of a system that needs some order introduced into it
- the need to install and uninstall services or plugins into a running system

Luckily, this is exactly a case for your Eclipse IDE, where most of the functionality deals with some kind of text editors, or structured text editors like we use for XML—or where most of the features can be treated as optional and come from plugins.

OSGi has a dynamic runtime, which creates a different classloader for every bundle. These classloaders are isolated and have references to other bundle classloaders, so importing and dependencies mechanisms work. However this approach allows OSGi to drop an entire module along with its classloader and recreate it again at a whim. This makes possible limited code reloading practices and greatly benefits the upgradeability of the application employing OSGi.

Other Eclipse-based IDEs

Needless to say that with a starting point as good as the Eclipse platform, any additional effort should likely result fairly impressively. Most of the IDEs based on Eclipse are fully compliant with open standards from the Java development world, and fully charged to help developing an application with EJBs, JPA, JSF, etc.



Probably the best-known Eclipse-based IDE is Genuitec's MyEclipse, which positions itself as the most comprehensive Java/Jave EE IDE on the market. It is a commercial, proprietary solution that, in addition to the obvious toolsets we use to develop Java and web applications, also includes preconfigured components to develop HTML5 and mobile apps, and several application servers like TomEE. The package comes fully ready to be utilized and you probably can save yourself some stress and time with MyEclipse's quick installation and easy configuration for dozens of plugins.



Next up is IBM's Rational Application Developer, or RAD, which focuses on easing the way of designing, developing and testing web applications utilizing the full Jave EE spectrum and Service Component Architecture. It integrates best with IBM WebSphere Application Server and includes profilers, data visualization tools and products to work with the IBM Middleware line.



Red Hat's JBoss Developer Studio is another example of an IDE focused on a specific set of technologies provided by a large company with a wide array of tools. Here you'll see integration with the JBoss Enterprise platform, and on top of the latest Eclipse and Web Tools Project includes number of excellent features from JBoss Tools.

How to migrate to new versions of Eclipse

Think it's time to try the latest version of Eclipse? Here are a few notes about migrating, including compatibility issues and how to install via pointing to previous locations.

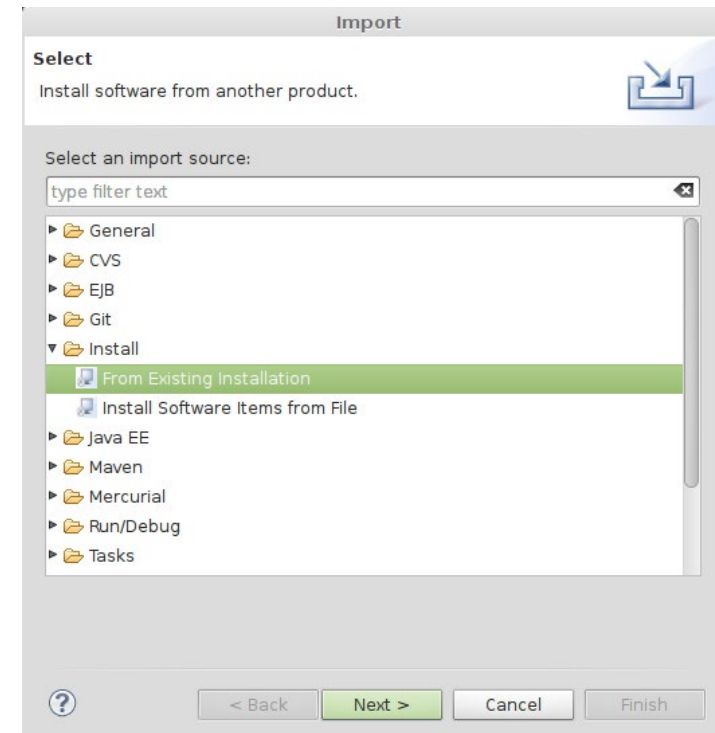
As we write this report, the latest stable version of Eclipse is version 4.3, aka 'Kepler'. However we're sure that a lot of people are still using older branches, such as Indigo (v3.8) or Juno (v4.2). As a developer, you know that your IDE settings are a very personal thing; you can configure it in a million ways and once all the formatters, validators, plugins are properly set, it's really annoying to reconfigure a new instance of the IDE.

However, Eclipse includes a very cool feature that let's you import everything from another installation. So when you think it's time to try a newer version of Eclipse, you don't need to risk corrupting your current instance.

Just download and extract a new archive somewhere and start it. Then simply go to **File -> Import...**, select **From Existing Installation**, feed it the path to your current Eclipse installation and voila! This will pick up your previous settings, create a list of installed plugins and go download them in a batch. After a restart, your new Eclipse version will feel just like home!

You can move directories with Eclipses around so all your filesystem links and shortcuts will point to the new one. Then you can enjoy a [hopefully] more powerful, improved version of Eclipse and continue developing awesome apps.

Now, if you are very eager to upgrade you can meet some compatibility issues, especially with plugins. They might require some older versions of a



core component installed, which would resist installation because a newer version of it is already available. In this case we usually recommend that you wait a bit. If a stable release of Eclipse is scheduled for June, then you can assume that all related projects like plugins and other tools will do their best to release soon after that. Then upgrading will probably work like a charm, unless you use some rare plugin.

The main point is that this installation method can save you some headaches.

Tips for increasing performance

At this point, you should have yourself a running instance of Eclipse and can dive into developing your favorite programs. However, there is one thing that we like to do when dealing with fresh Eclipse installation.

That's right: let's make it faster!

Performance is a critical property of an IDE, and if it feels clumsy and slow it doesn't matter how many awesome features it has, you're likely to look at a different IDE next time. So here are some hints about making Eclipse a bit more productive.

Point number 0. If you don't have it yet, get an SSD drive. Eclipse makes tons of file operations, so fast I/O is essential. The performance difference can be huge and it grows with the number of projects you have currently open.

First of all, Eclipse is a Java-based application, so all tuning advice that you know about usual Java programs applies here as well. Use the latest JDK and preferably Oracle/OpenJDK--Eclipse performs best on them. There also is an eclipse.ini file that specifies options for Java processes. A typical ini file will specify several arguments to the Java Virtual Machine, and below you'll see an example of our ini files.

First, we don't want to verify class files, and this makes starting up faster. Also, let's make use of G1 garbage collector and set some properties for the JIT compiler--it is usually smart to specify memory arguments too.

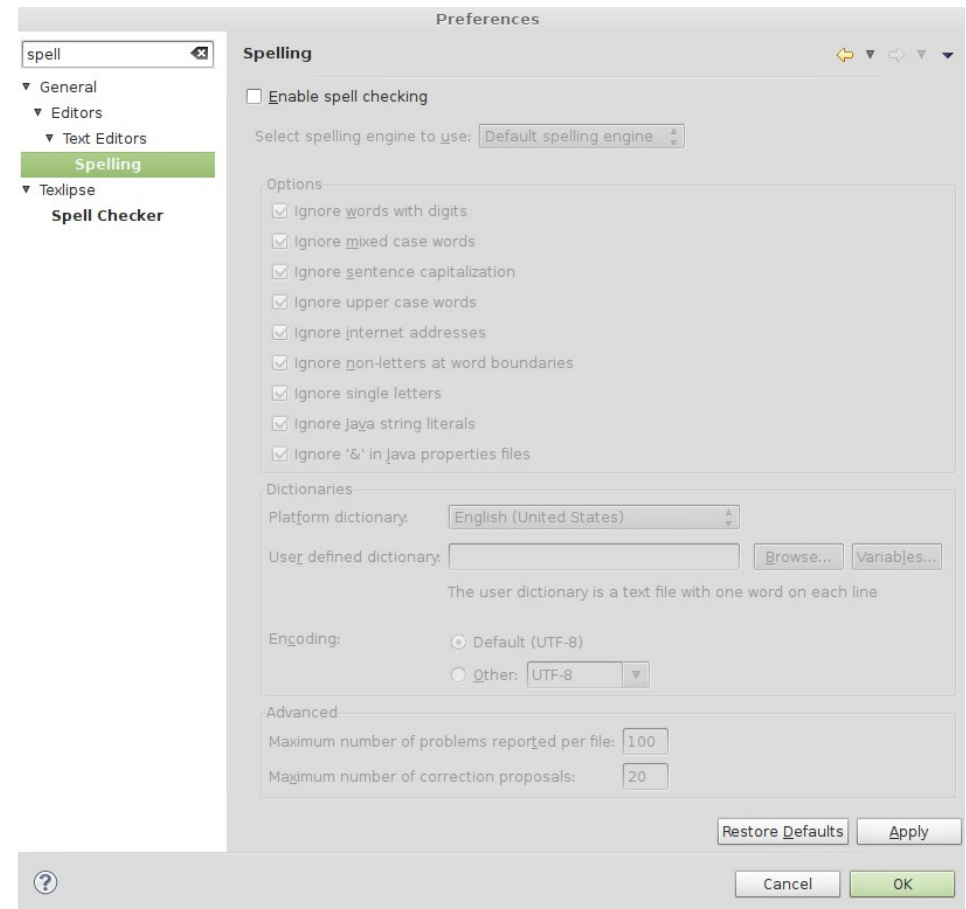
```
-vmargs  
-Xverify:none  
-XX:+UseG1GC  
  
-XX:-DontCompileHugeMethods  
-XX:MaxInlineSize=1024  
-XX:FreqInlineSize=1024  
  
-Xss1m  
-XX:MaxPermSize=256m  
-Xms512m  
-Xmx512m
```

Play with those parameters to see how Eclipse reacts. It might be a good idea to import your projects first, so you'll see a real-world usage performance.

Now let's tweak in-Eclipse properties to make life even better. Usually there are several things that Eclipse does, but you don't use every day or even not at all and can be disabled to gain some responsiveness in return.

In our humble opinion, Spellchecker, all kinds of validators and startup plugins are among the first candidates to be disabled. Luckily, the Eclipse preferences menu has good search functionality so you won't need to traverse it all to find relevant bits. Go to **Preferences**, type "spell" into search and disable it.

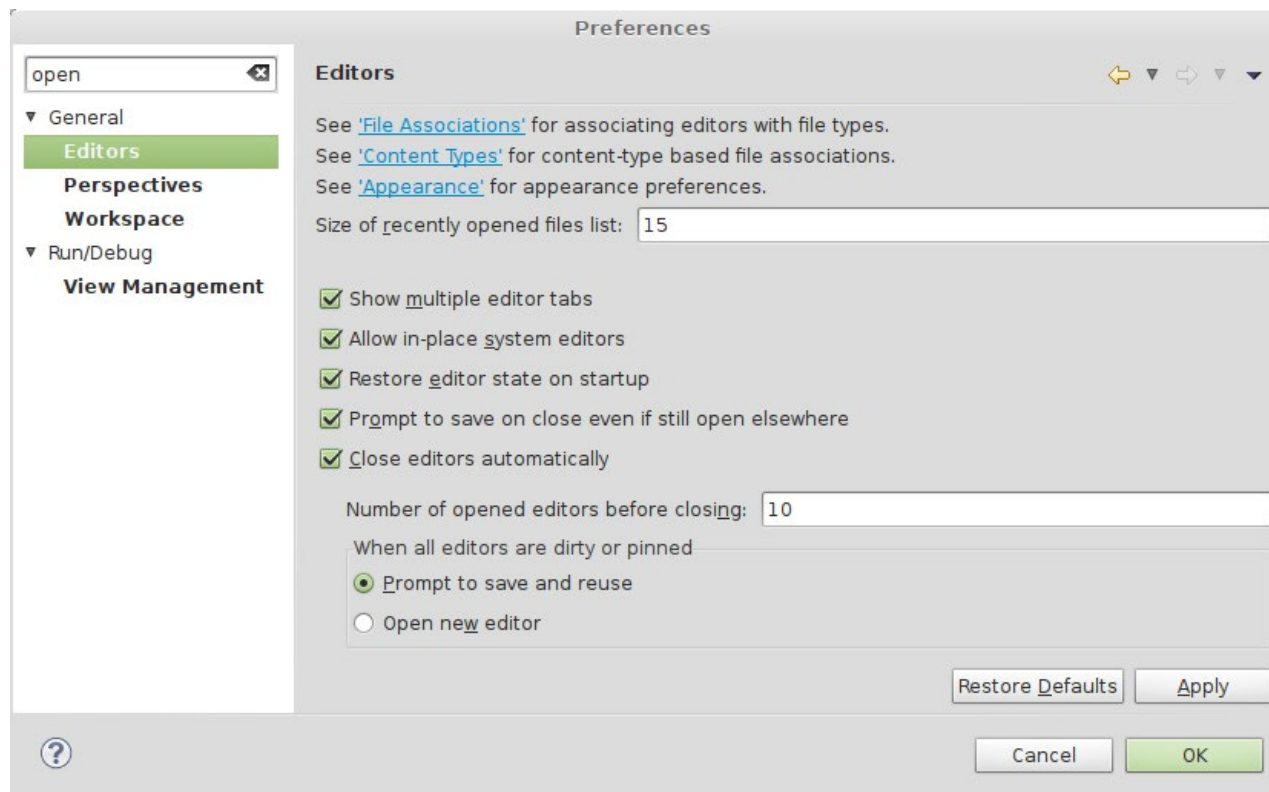
Continue with searching for "startup" and review the list of plugins under "Startup and Shutdown". Are you sure that you need that "m2e marketplace"? Disable it. If you are not sure, it's almost always a good idea to disable a plugin. If you realize later that it was a bad idea and something doesn't work, you can always turn it back on.



Another tip is to restrict the amount of editors that Eclipse will hold open while you're playing with open or new files. Find the **Editors** options in the **Preferences** menu and set it to some lower number. Remember, every opened editor consumes memory and takes precious CPU cycles to parse, validate, etc.

Naturally, there are many more settings that can be changed to improve performance without compromising your user experience. These first few are our simple suggestions to improve the responsiveness of your Eclipse installation by a noticeable margin.

Remember, Linus Torvalds thinks that performance almost always matters. What about you?

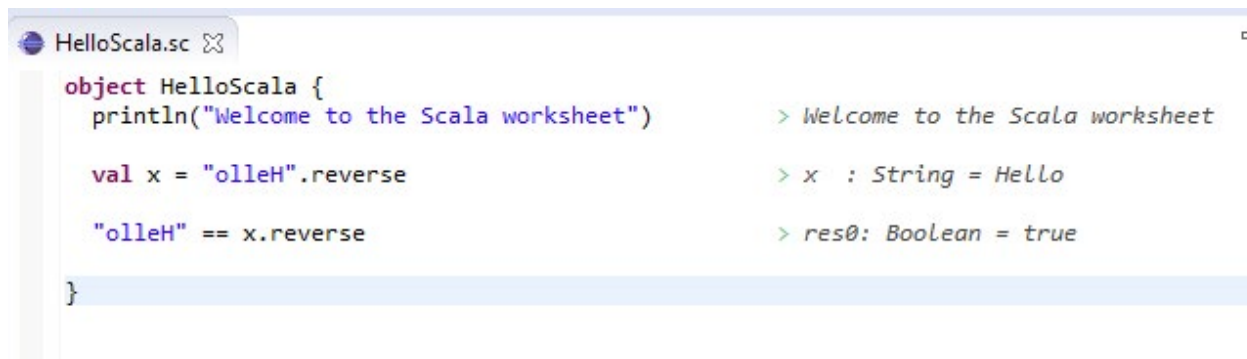


Other language packs for Eclipse – Scala, Python and Xtend

SCALA

Support for Scala programming with your Eclipse IDE is provided by Typesafe and other contributors, and it's available from <http://scala-ide.org/> (either as a bundle or from an update site). Support for Eclipse 4.x versions is currently experimental, but it's designed to hook into Java Development Tools via runtime weaving and makes mixed Scala/Java projects work rather seamlessly. Unfortunately, it also means that if there is a bug in the Scala IDE, the bugs may surface in your Java projects as well. It used to be quite problematic, but has been rather stable since version 2.0 (at the time of this writing it's v3.0.1).

Besides the expected language pack features, such as perspective, editor with code completion, formatter, project nature, refactoring support and more, there are some awesome optional features, such as the ScalaTest plugin for running unit tests and Scala Worksheet, which combines a code editor and REPL into one, evaluating each line when you save the file:



```
object HelloScala {  
  println("Welcome to the Scala worksheet")  
  
  val x = "olleH".reverse  
  
  "olleH" == x.reverse  
}
```

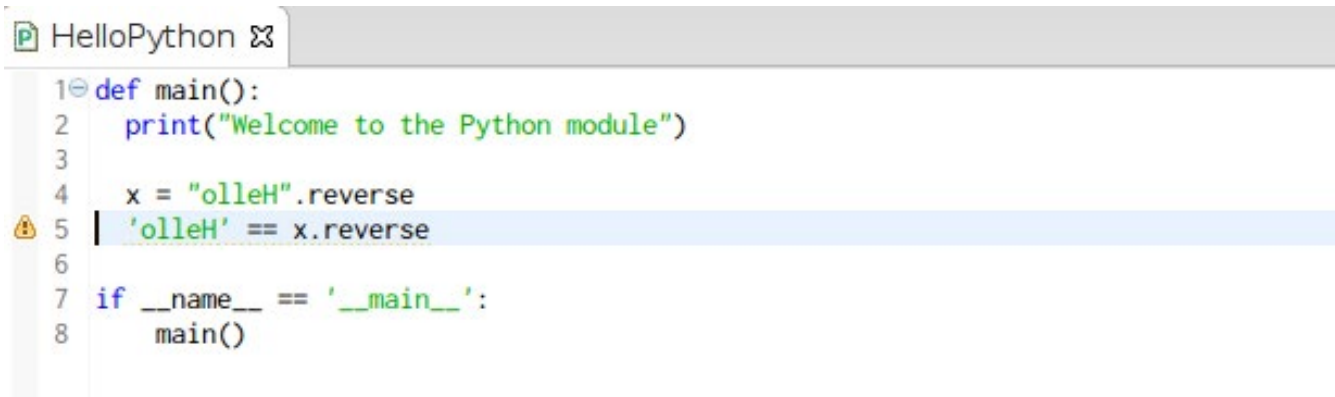
> Welcome to the Scala worksheet
> x : String = Hello
> res0: Boolean = true

PYTHON

Projects often require some glue scripting and if you're using Python for that you can benefit from having a proper Python IDE inside your Eclipse. [PyDev](#), an open-source Eclipse plugin developed by Fabio Zadrozny, can help you with it.

PyDev makes debugging Python scripts much easier, provides content assist for the interactive Python console and in general helps you work with Python code better. Other features of PyDev include integrations with *Django* and *unittest*, including code-coverage support and the ability to quickly browse a project's global tokens.

Refactoring isn't as top-notch as a Java developer would hope for, but the most used things are there; like extracting, inlining & renaming variables, and extract methods. Naturally, you can use PyDev with any flavor of Python you want: normal, Jython or IronPython.

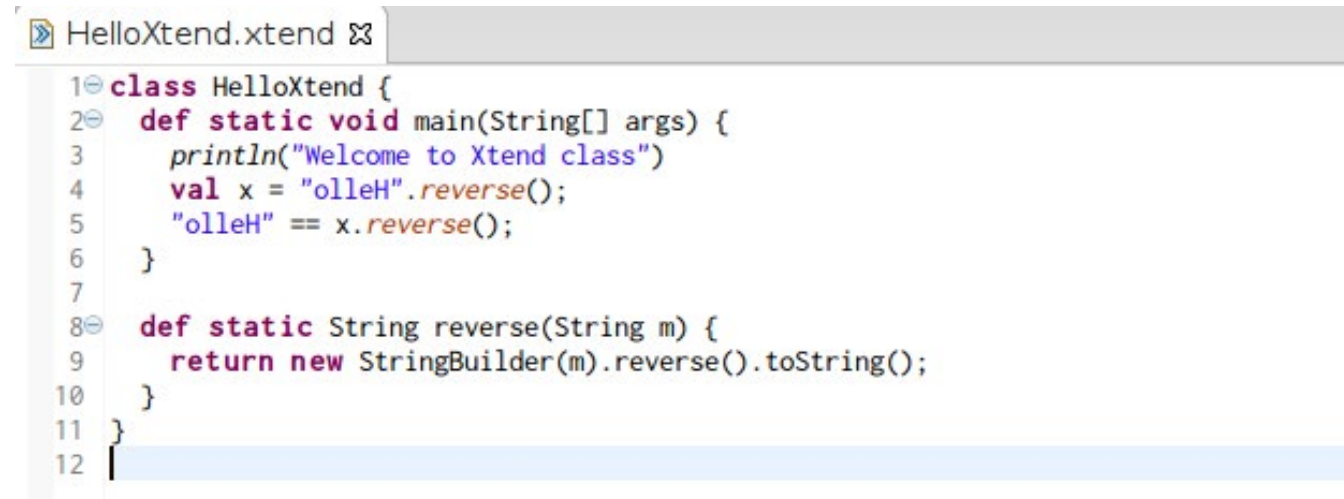


```
1 def main():
2     print("Welcome to the Python module")
3
4     x = "olleH".reverse
5     'olleH' == x.reverse
6
7 if __name__ == '__main__':
8     main()
```

XTEND

Xtend is a language that grew out of Eclipse's effort to build a framework for language creation, so it is expected that IDE support is excellent. Indeed, you can install an Xtend plugin into your Eclipse installation and run a full bundle with Xtend support. The fact that the IDE and the language itself were designed together makes a difference.

Xtend compiles to Java source code and is completely compatible with Java syntax, so you just need to add the Xtend library as a dependency in order to use Xtend in a usual Java project. The plugin makes it easy to configure the location where you want to generate intermediate Java source files and go through this generated code when you want to see what Xtend does under the hood.



```
1 class HelloXtend {
2     def static void main(String[] args) {
3         println("Welcome to Xtend class")
4         val x = "olleH".reverse();
5         "olleH" == x.reverse();
6     }
7
8     def static String reverse(String m) {
9         return new StringBuilder(m).reverse().toString();
10    }
11 }
12
```

PART II:

MAKING ECLIPSE YOUR OWN

Now this is where Eclipse gets the most interesting. Plugins are an essential piece of the Eclipse ecosystem, and they are valuable to the user experience because they provide a great number of features to the default installation.

Plugins and customizations

We must admit that it actually is a two-sided coin here; there's no easier way to make your Eclipse instance unbearable than to pollute it with incompatible or conflicting plugins. The consequences may vary from an unexpected color scheme on shell script editors to Eclipse refusing to start because your Mercurial plugin broke.

However, given some experience, it's relatively easy to enhance your Eclipse installation and build up the things it can do for you.

THE ECLIPSE MARKETPLACE

It used to be that installing plugins into Eclipse happened by either unzipping an archive of plugins and features into the Eclipse folder, or by giving an update site URL to the included software installation wizard via **Help -> Install New Software**. The unzipping method is pretty much gone, so now the installation is mostly done from update sites.

But what if you don't know any useful update sites? The Eclipse Foundation found that users also wanted a better way to discover plugins, so they created the [Eclipse Marketplace](#), which acts as a kind of a plugin registry. Your Eclipse IDE also includes a Marketplace Client (**Help -> Eclipse Marketplace**), which can be used to search and install plugins that are registered on that site.

Tip: You can also browse the marketplace website and drag "Install" links from the browser onto a menu or toolbar area in Eclipse to initiate installation using the Marketplace Client.

SOME BASIC PLUGINS ANYONE CAN ENJOY

EGit

There are a number of plugins that are really essential, everyone can benefit from having those installed. One of those is [EGit](#).

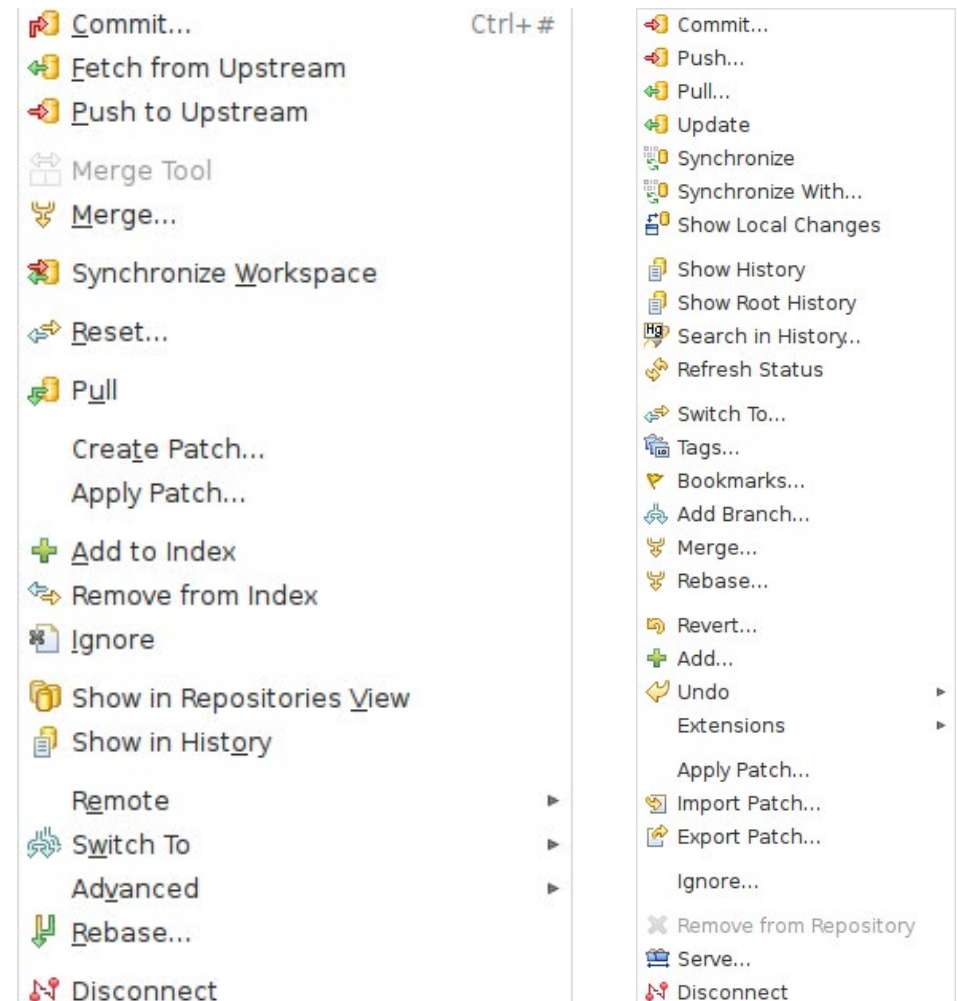
This is an Eclipse Team Provider that is based on a Java implementation of Git. EGit gives you ability to perform most Git commands using a Team menu of a project.

It's quite fast, useful, straightforward and works well with GitHub. There is some advice to follow, but it mostly is well aligned with common sense: don't create Git repositories in your eclipse workspace and don't create a Git repo with your project as a root.

If you're on the other side of the SCM camp and use Mercurial, don't worry, Eclipse has a plugin for you too...

MercurialEclipse

[MercurialEclipse](#) works quite the same as EGit, allowing you to synchronize, commit and push your changes to remote locations. It also gives you access to certain Mercurial extensions like mqueue, so you can do even more without leaving the IDE.



Eclipse Color Theme

Now, a completely different example of a generic plugin is the [Eclipse Color Theme](#). This makes the color scheme of the editors in Eclipse extremely customizable and provides several themes to pick from. If you fancy a dark color scheme there are several, plus you can install third-party themes from the internet (including solarized). Pick one that is the easiest on your eyes.

```
0 public class Demo {
1     private static final String CONSTANT = "String";
2     private Object o;
3
4     /**
5      * Creates a new demo.
6      * @param o The object to demonstrate.
7      */
8     public Demo(Object o) {
9         this.o = o;
10        String s = CONSTANT;
11        int i = 1;
12    }
13
14    public static void main(String[] args) {
15        Demo demo = new Demo();
16    }
17 }
18
```

```
0 public class Demo {
1     private static final String CONSTANT = "String";
2     private Object o;
3
4     /**
5      * Creates a new demo.
6      * @param o The object to demonstrate.
7      */
8     public Demo(Object o) {
9         this.o = o;
10        String s = CONSTANT;
11        int i = 1;
12    }
13
14    public static void main(String[] args) {
15        Demo demo = new Demo();
16    }
17 }
18
```

```
0 public class Demo {
1     private static final String CONSTANT = "String";
2     private Object o;
3
4     /**
5      * Creates a new demo.
6      * @param o The object to demonstrate.
7      */
8     public Demo(Object o) {
9         this.o = o;
10        String s = CONSTANT;
11        int i = 1;
12    }
13
14    public static void main(String[] args) {
15        Demo demo = new Demo();
16    }
17 }
18
```

There are other types of plugins that enable some functionality not related to external tools. For example, the MoreClipboard plugin allows you to cut/copy/paste using several clipboards like vim buffers. Obviously there are hotkeys to perform these operations.

JRebel for Eclipse



Disclaimer: We make JRebel, but that doesn't make it any less awesome.

Since productivity in Java development is extremely important, you can amplify your project by installing this plugin from the Eclipse Marketplace and eliminating the need to restart your application server each time you change code. With the 'Build automatically' checkbox enabled, you'll also enjoy instant builds along with no restarts—JRebel also maintains session state, so you never need to find your place in the code again after verifying that your code changes are correct.

CODE COMPLETION

If you're into Java, then you know that there's nothing more pleasant than enjoying smart content assist that literally creates the program on your behalf while you just ctrl+space (hotkey for calling content assist) like crazy.

[Eclipse Code Recommenders](#) takes assisting to the next level: it scans your codebase and learns from it to offer operations that are called on similar objects in similar conditions. You don't need to consider a hundred methods offered by a public API that is never used.

Code Recommenders is intelligent and provides subword completion, so you don't need to guess the prefix of the method name. Just start typing any part of it and Code Recommenders will figure it out.

There's also a chain completion mode, so if some chain of methods is used often, it will offer it to you as a whole in one go.

TESTING

Anyone adept at TDD would desire useful plugins to speed up your feedback cycle even more. We recommend [JUnitLoop](#) and [InfiniTest](#), which both monitor changes in your project and execute unit tests for areas that could be possibly affected. This means that after every change you'll know if something is broken. And we think that more “create test, red bar, code, green bar” cycles is always a good thing.



To help you utilize tests even more, check out additional code coverage plugins. For example, [EclEmma](#) will help you with visualizing code coverage data with thorough reports and syntax highlighting. So every time after running tests you will instantly see which code paths has or has not been tested.

WORKSPACE MECHANIC

A ‘workspace’ is a really nice project separation concept; it keeps all related projects together in one place, maybe along with some small or throwaway projects, but lets you separate it with different setups.

It’s great that you have total control of how your IDE treats your projects, but Eclipse keeps the configuration specific to the workspace, which is often a pain to recreate over and over again in each and every workspace you have.

[Workspace mechanic](#) to the rescue! It allows you to create consistently set workspaces or share your configuration with a team easily.

This plugin works by externalizing your settings and creating automated tasks to warn you if they’re not compliant. It makes configuring Eclipse much quicker and you won’t forget to set any settings again.

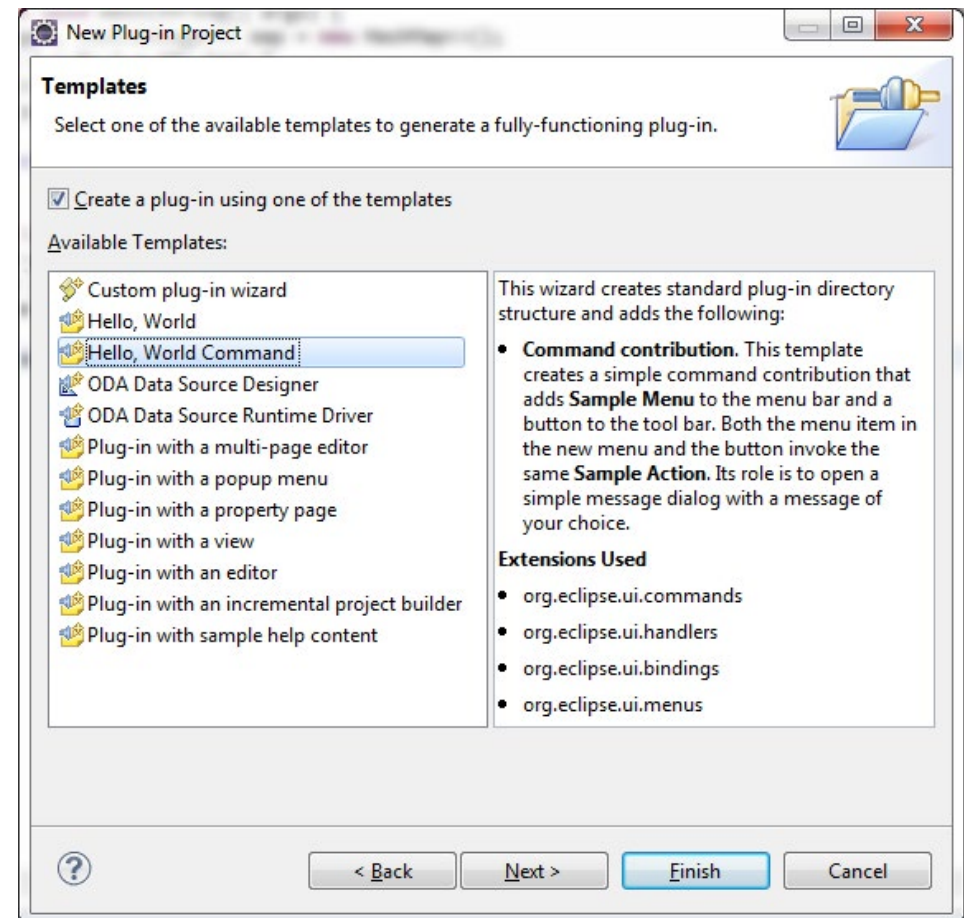
More advanced: How to write an Eclipse plugin

What if these existing plugins don't cut it? You could of course write your own. The Eclipse Standard bundle includes all you need for plugin development, including the Plugin Development Environment (PDE) and the Eclipse SDK, containing the source code for the platform.

If you start from something other than the Eclipse Standard package, you can get the SDKs from the update site named "The Eclipse Project Updates" (for Eclipse 4.3 this is at <http://download.eclipse.org/eclipse/updates/4.3>), by installing at least Eclipse SDK and Eclipse Platform SDK.

Eclipse is perhaps the best IDE to write plugins for because the API has been very stable over the years--plugins written for old versions can still work with the latest version of Eclipse. Eclipse itself is completely made of plugins (OSGi bundles) so adding new plugins is very natural. There are lots of "Extension Points" that let you hook into the IDE functionality, making it possible to create your own perspectives, views and editors.

To create a plugin, select **New -> Plug-in Project**. In the first wizard page, just enter a name, such as HelloPlugin, and click Next until you see a list of templates. Select "Hello, World Command" (or another template if you want).



Click **Finish** and accept to change to the PDE perspective or not. The PDE perspective doesn't add much--only the Plugins View, which shows all the plugins in your "Target Platform" (Target Platform is the running Eclipse instance by default, but you can also develop against a different version of Eclipse).

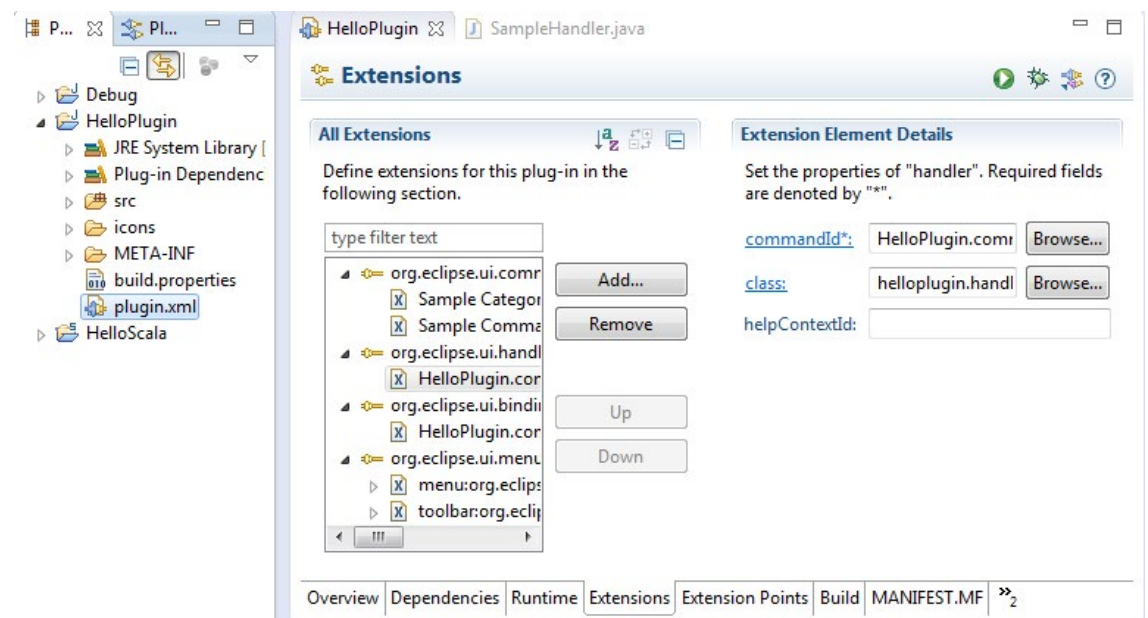
Observe what was created from the template:

META-INF/MANIFEST.MF is what defines your plugin's OSGi metadata, including dependencies on other plugins. By default it should depend on **org.eclipse.ui** and **org.eclipse.core.runtime**.

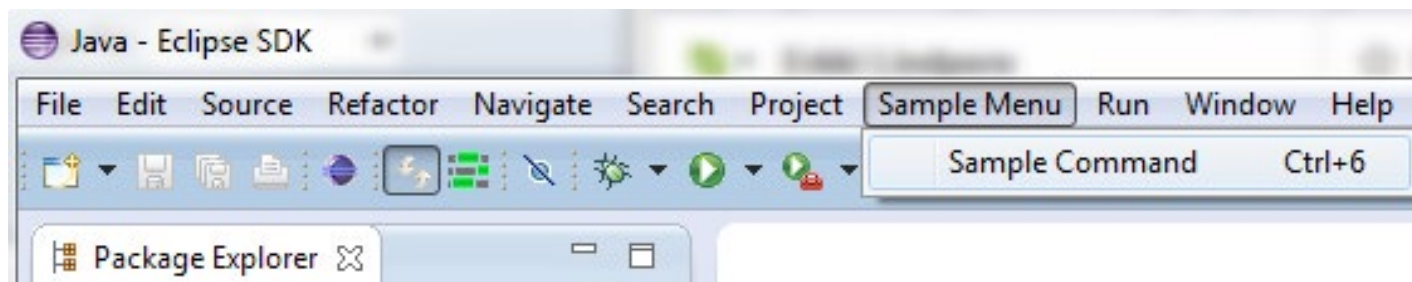
plugin.xml defines the extensions that your plugin provides.

Note that both of these files are opened with the same GUI editor by default, but their data is in different tabs. In the **Extensions** tab you can see that the plugin adds commands, handlers, bindings and menus. To see more information about extensions in this editor, you should have the Eclipse SDK installed.

Another note: the handler extension point has a "class" attribute that contains the qualified name of a handler class. This is the usual mechanism how code is associated with extensions--Eclipse will expect the class to have a default constructor and will instantiate it by calling that. This sample handler class just opens a message dialog.



To see how the plugin works, you can open the **Overview** page of the plugin editor. On the right side you can see a **Testing** section, with a link “Launch an Eclipse application”. This will launch a new instance of your current target platform with your plugin installed as well. The new Eclipse instance will launch with a new workspace, but you can later open the *Run Configurations* dialog and change the workspace or other properties of the Eclipse application launch.



Once launched, you can see the plugin has added a “Sample Menu” to the menu bar, a command to the menu, and a key-binding (Ctrl+6) to launch the command. This is perhaps the simplest plugin you could create, but to see what else is possible, check out some of the other plugin templates. You can also see separate Extension templates that you can add to the plugin by clicking **Add...** in the Extensions tab and selecting Extension Wizards.

Not all of the functionality you can add to Eclipse necessarily goes through extensions in plugin.xml. Sometimes you can also do things programmatically. If you know what you want to add, but don't know which extensions or APIs allow you to do it, the usual channels (Google, StackOverflow etc.) should provide help, and The Eclipse Foundation has collected some articles at <http://www.eclipse.org/resources/>.

A note about top Eclipse Marketplace plugins

If you look into the top plugins in the marketplace, you'll see a somewhat unrepresentative picture actually. The top spots are occupied by Subversion and Maven integration plugins. Essentially, we see nothing wrong with that, but lots of more interesting plugins are hiding out lower in the ranks, even in the Top 20. As of October 3, 2013, here is a quick snapshot of the top 20 all-time plugins downloaded from the Eclipse Marketplace.

For example, the **Mylyn** plugin can help you manage tasks in your task tracker and it has several connectors that integrate with [Github](#), [Jira](#), [FogBugz](#), [BugZilla](#), etc. We know that you can just as easily alt+Tab to the browser window to manage your tasks, but the plugin is really useful to track the time that went into development, mention tasks in commit messages and influences your life in other minor, but positive ways.

Rank	Solution	count
1	Maven Integration for Eclipse (Juno and newer)	545419
2	Subclipse	538594
3	Subversive - SVN Team Provider	530447
4	EGit - Git Team Provider	282199
5	Eclipse Color Theme	266740
6	Maven Integration for Eclipse WTP (Juno)	211583
7	PyDev - Python IDE for Eclipse	162523
8	FindBugs Eclipse Plugin	147256
9	Android Development Tools for Eclipse	140565
10	Spring IDE	110073
11	JBoss Tools (Juno)	109572
12	Spring Tool Suite (STS) for Eclipse Juno (3.8 + 4.2)	109268
13	EclEmma Java Code Coverage	106590
14	JBoss Tools (Indigo)	94801
15	Checkstyle Plug-in	89417
16	m2eclipse-wtp : Maven Integration for Eclipse WTP (from github)	76093
17	JRebel for Eclipse	75370
18	TestNG for Eclipse	62486
19	GlassFish Tools for Juno	55761
20	Spring Tool Suite (STS) for Eclipse Indigo (3.7)	52162

Eclipse integration with Application Servers and Build Tools

As Eclipse itself is an open platform for building basically anything you can imagine on top of it, it also contains a lot of integrations to application servers, build tools and other services or platforms. It makes sense to quickly review these as well.

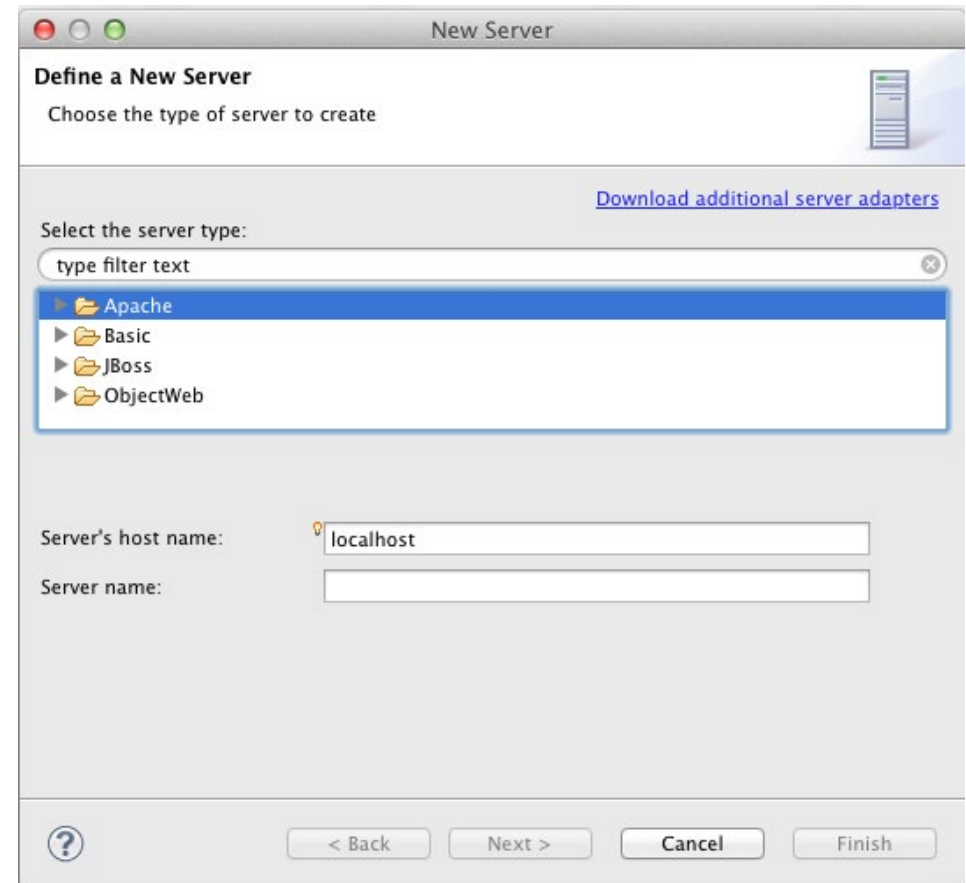
APPLICATION SERVERS

We did a list of common application servers and tried to find an adapter for them. To test if the integration works out of the box, we installed the latest Eclipse Kepler and imported a project called “[Petclinic](#)”, which is a Spring-based showcase application provided by SpringSource. We consider it as a typical, if simple, web application. There are two tasks that we expect to work for each app server adapter:

1. Automatic deployment & publishing to the App Server;
2. Ability to start and stop the server from Eclipse.

By default, Eclipse for Java EE developers comes with some bundled application server adapters for Apache Tomcat (mostly up to date), JBoss (outdated) and ObjectWeb.

Fortunately, it is quite easy to add additional app server adapters plus the links to the place where it should be:



Apache Tomcat

As we said, Eclipse for Java EE already comes with integration adapters for Apache Tomcat, which works really well too. The Tomcat plugin is always up to date and all latest versions are available in the plugin.

JBoss AS (aka WildFly)

Integration for JBoss is available and it is called "JBoss AS Tools". After installing JBoss AS 7.1 Petclinic worked almost out of the box. Although the context path was wrong in the initially opened browser window, but it wasn't a big deal. In general - integration with JBoss AS looks good.

Jetty

Jetty's integration with Eclipse seems to be outdated. Although there is already Jetty v9 out, the latest Jetty version from the Jetty adapter we were able to find info about was v8. And the old plugin was released by Mortbay Inc.

Oracle GlassFish Server

Oracle provides server tools for GlassFish, which are available in the list of available server adapters. Oracle Glassfish 4 works pretty well with Eclipse; we had no hassles when we installed it, and everything worked just fine. Well done, GlassFish team!

Oracle WebLogic Server

We had some problems getting Petclinic + Eclipse + WebLogic working out of the box. Publishing from Eclipse did not work properly, some libraries didn't get published and the application did not work properly.

IBM Liberty Profile

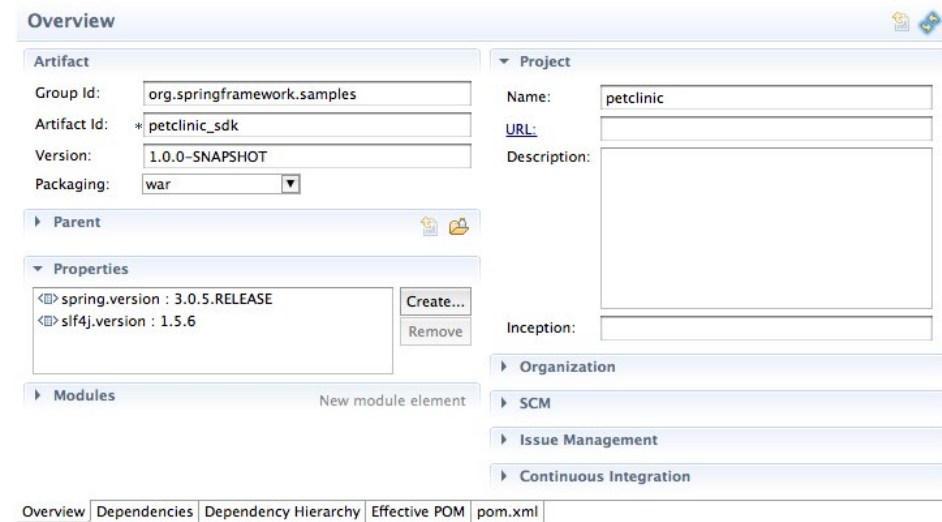
Even though there is no Liberty Profile plugin for the latest version of Eclipse (Kepler), the plugin meant for Eclipse Juno worked pretty well, and we did not encounter any problems. Everything worked, WebSphere was up and running, and Eclipse was able to deploy apps on it.

INTEGRATION WITH JAVA BUILD TOOLS

Development environments are not just plain IDEs that manage folders full of source codes. There are some other complicated tools around like build tools, dependency management tools and shell scripts. We are going to take a look at some most common tools here and see how Eclipse works with them.

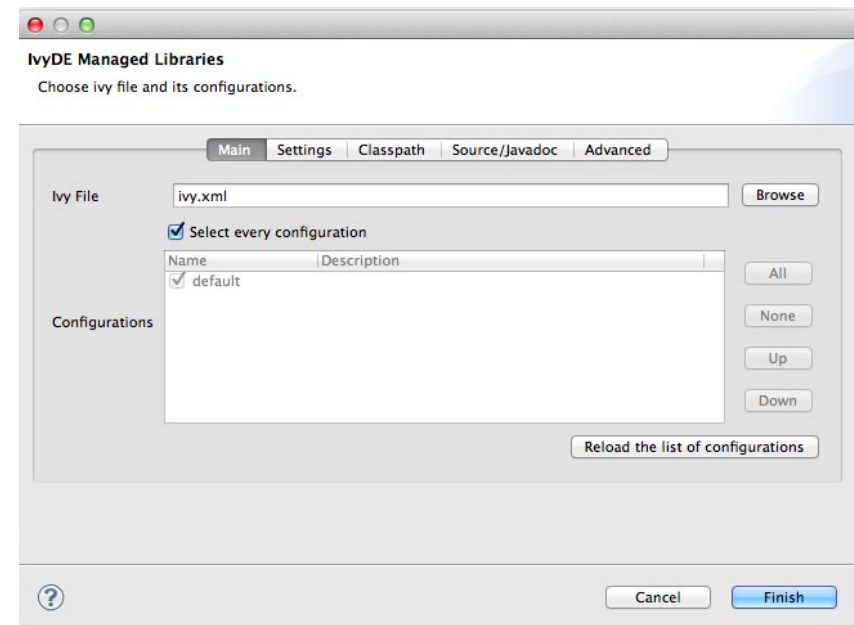
Maven

Maven integration for Eclipse is mostly based on the *m2e plugin* (<http://eclipse.org/m2e/>). The plugin is now under the umbrella of the Eclipse Foundation itself, which is probably good for the plugin in the long term.



Ant + Ivy

Ant support for Eclipse comes out of the box and it works pretty well. IvyDE enables you to integrate Ivy's dependency management into Eclipse. It seems to be well working and under active development / maintenance.



Gradle

Gradle support for Eclipse is excellent, with one of the best-documented integrations out there. You can migrate existing projects into Eclipse projects and you can also execute Gradle from within Eclipse. Just run "gradle eclipse" from the command line and all classpath & IDE specific configuration issues are solved. So, if you care a lot about having a well-documented tooling and project setup, then Eclipse & Gradle may be a good way to go here.

PART III:

TIPS AND TRICKS FOR USING ECLIPSE LIKE A SUPER-NINJA BAD@\$\$

Now that your Eclipse installation has all the best plugins installed, and you've configured integrations with the external software you use, it's time to be awesome. While some may consider your IDE more or less an advanced notepad, there are some tips we would like to share to make your experience more pleasant.

Navigating your Eclipse installation

QUICK ACCESS

Do you know why vim is still popular 22 years after its initial release? Because it enables excellent text navigation capabilities, which is the key to being productive with your IDE.

Eclipse offers you the same navigational supremacy, but with a much more intuitive interface. Navigation in Eclipse is based on shortcuts and key-bindings, with the most important being a quick access shortcut (**Ctrl+3**).

Start typing what you have in mind and see Eclipse search in available views, menus, possible action items, configuration, everything! Look what happens when we simply type “f”.

Don't go through menus to find that obscure setting that you need to toggle, jump right to it and get it done. For example, when you want to create new file you can use **Ctrl+N** combination, or instead of remembering all the shortcuts that you rarely use, try **Ctrl+3** > “new file”. And while it is just a tiny bit longer, you have quick access to amazing power right there.

Previous Choices	<ul style="list-style-type: none">File Associations - General/EditorsFavorites - Java/Editor/Content AssistColors and Fonts - General/AppearancePreferences (General > Appearance > Colors and Fonts) - Open the preferences dialog
Views	<ul style="list-style-type: none">Full Latex OutlineBytecode ReferenceGit ReflogJRebel Config Centre
Perspectives	<ul style="list-style-type: none">JRebel Config Centre
Commands	<ul style="list-style-type: none">Fetch From GerritFile Search - Open the Search dialog's file search pageFind Text in File - Searches the files in the file for specific text.Find Text in Project - Searches the files in the project for specific text.Find Text in Working Set - Searches the files in the working set for specific text.Find Text in Workspace - Searches the files in the workspace for specific text. (Ctrl+Alt+G)Forward search - Toggles a forward search in all open pdf documents (Ctrl+Alt+F)
Menus	<ul style="list-style-type: none">File - Search for Text in FileFile...Debug Configurations...Maximize Active View or Editor - Toggles Maximize/Restore State of Active View or EditorMigrate JAR File...
New	<ul style="list-style-type: none">Faceted Project -File - Create a new file resourceFilter - Create a new FilterFolder - Create a new folder resource&Source Folder -
Preferences	<ul style="list-style-type: none">Favorites - Java/Editor/Content AssistFile Associations - General/EditorsFile Content - TeamFile types - PyDev/Editor/Code Style

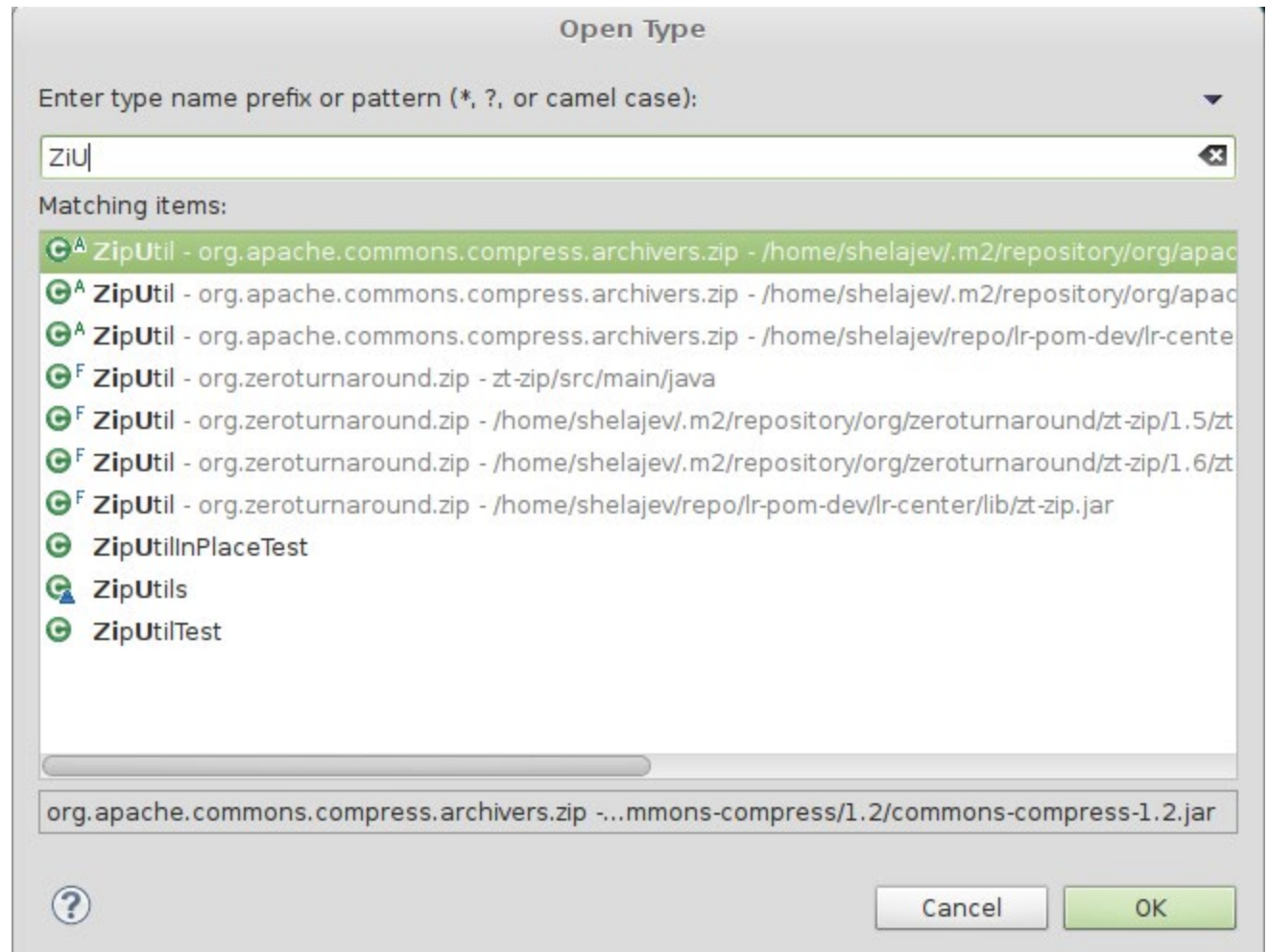
Press 'Ctrl+3' to show all matches

FILE NAVIGATION

Now you can navigate Eclipse like a ninja, but what about navigating files? **Open Type** and **Open Resource** are your friends.

Open Type (Ctrl+Shift+T) shows you a dialog where you can type the name of the class to open (aka CamelCase matching/search). Note that you can go with capital letters from class name only.

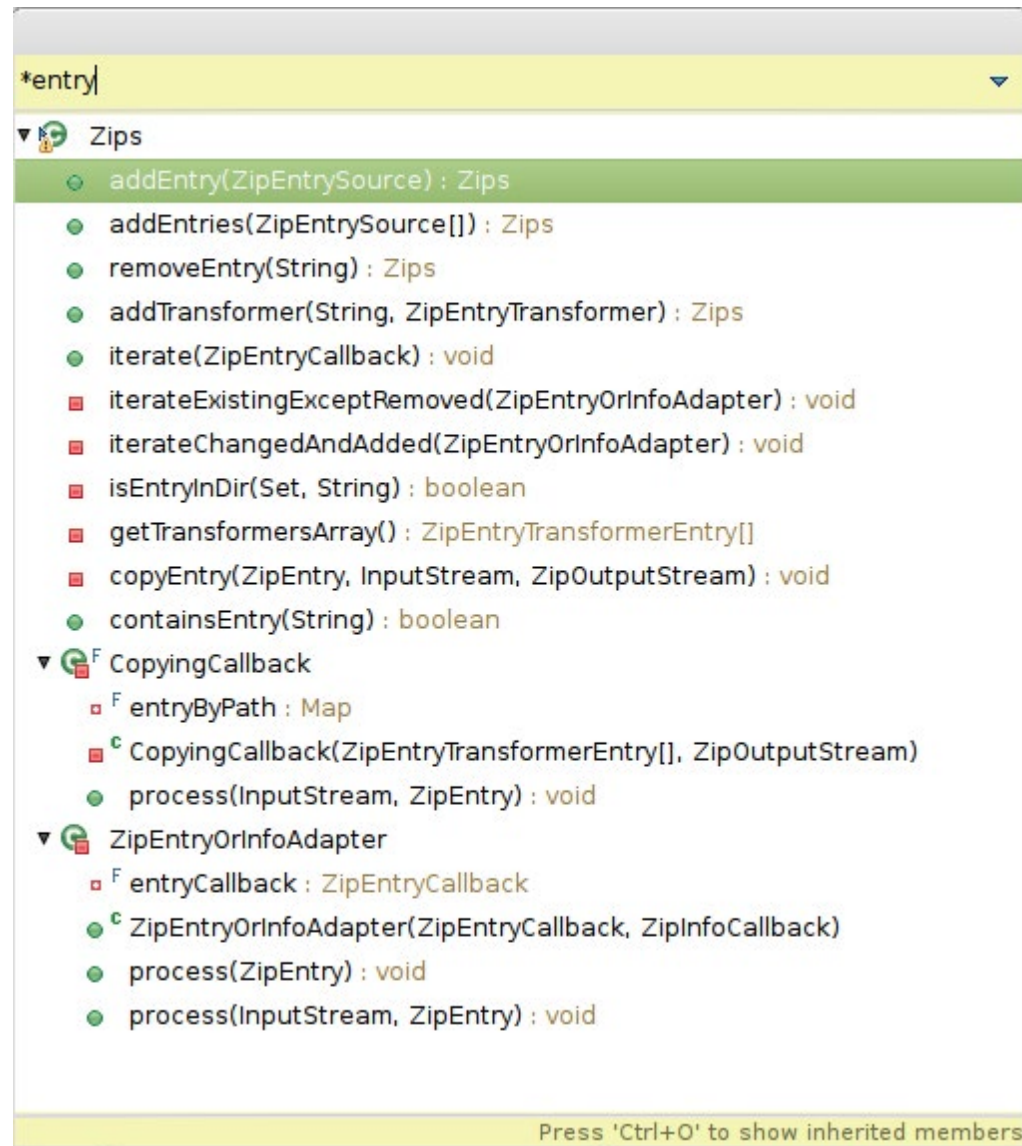
Open Resource (Ctrl+Shift+R) which operates in the same way but also includes non-class files. Looking for a certain *pom.xml* is done in a second.



CLASS OUTLINE

Finally, when in a class you can use the usual incremental search--**Ctrl+F**--like any other editor provides, or you can be more fancy and use **class outline (Ctrl+O)**, to quickly get to a definition of a class member, such as a method or field. Press **Ctrl+O** again and you can select and jump to inherited methods declarations.

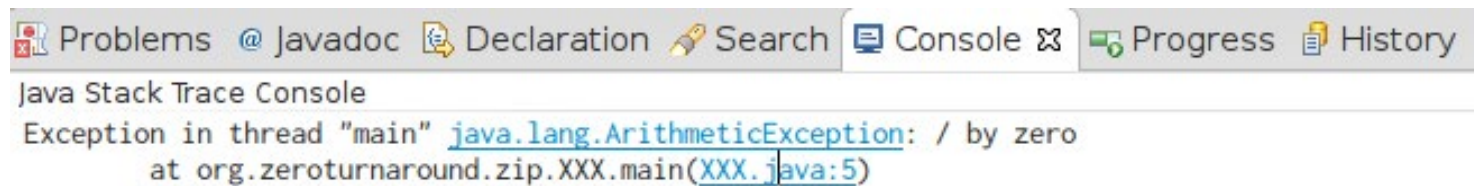
Finally, there is a **go to line** action (**Ctrl+L**), which can save you some scrolling.



STACK TRACE CONSOLE

The last tip to share about navigation is to make use of the **stack trace console**. Imagine you got an exception stack trace from somewhere like your support email, console logs or your CI server. Copy it to your system clipboard.

In Eclipse, use **Ctrl+3** to type “clipboard” and select “Open from clipboard”. Now just click on highlighted links to quickly go to the line from the stack trace.



This is a subtle productivity improvement, but sometimes it's very useful to traverse an exception trace.

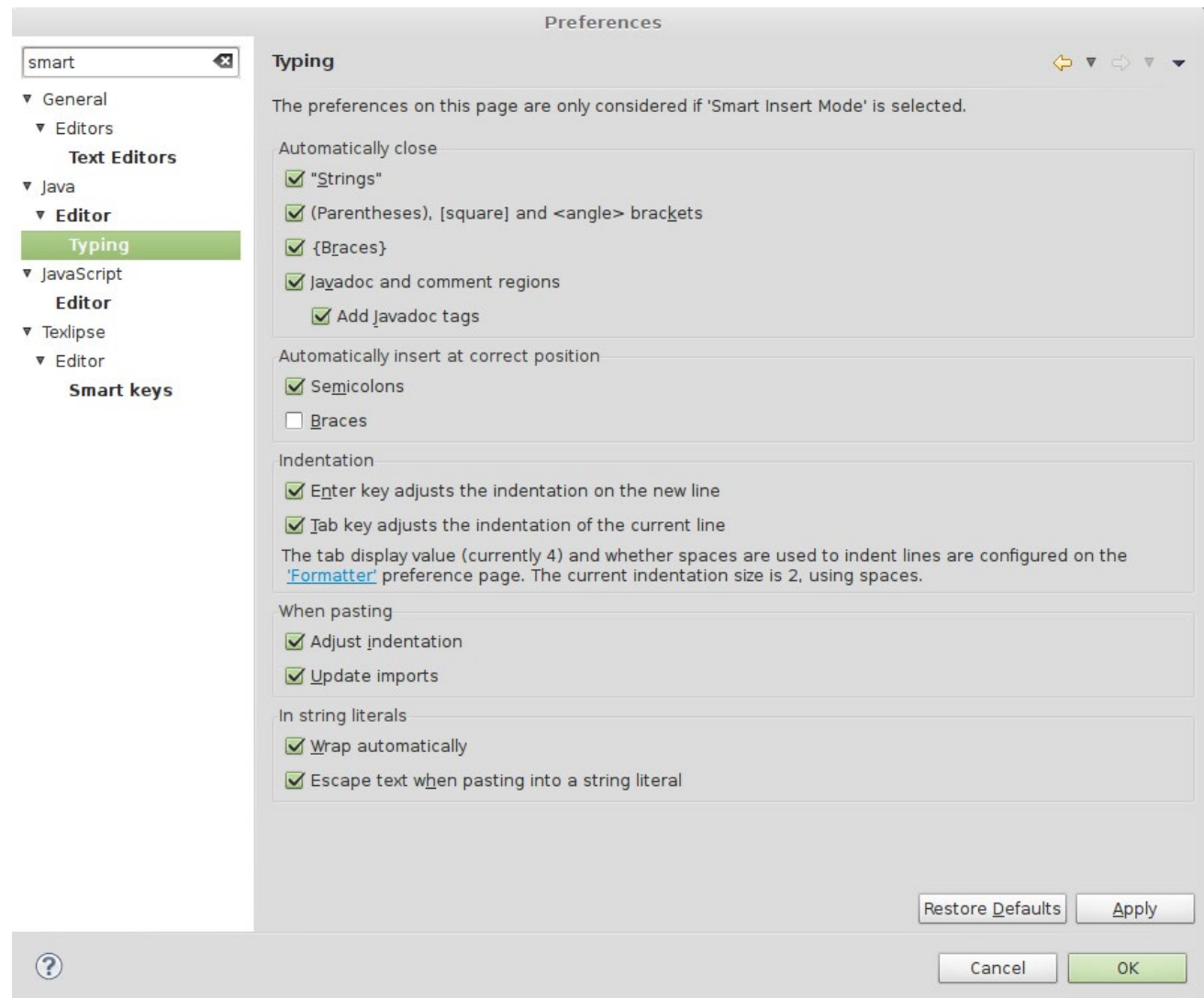
If you'd like to embed your navigation habits deep into motoric memory, you can use [IDE++ plugin](#). It will learn of your use of the Eclipse and sometimes show you a popup about how you can make certain things happen faster. In theory this is awesome, but the experience of some of our developers has shown us that this can get annoying, and after a while you'll be tempted to just pick a tip or two, like going to the place of your last edit operation with **Ctrl+Q**, and then uninstall it.

MAKING ECLIPSE A BIT MORE CLEVER

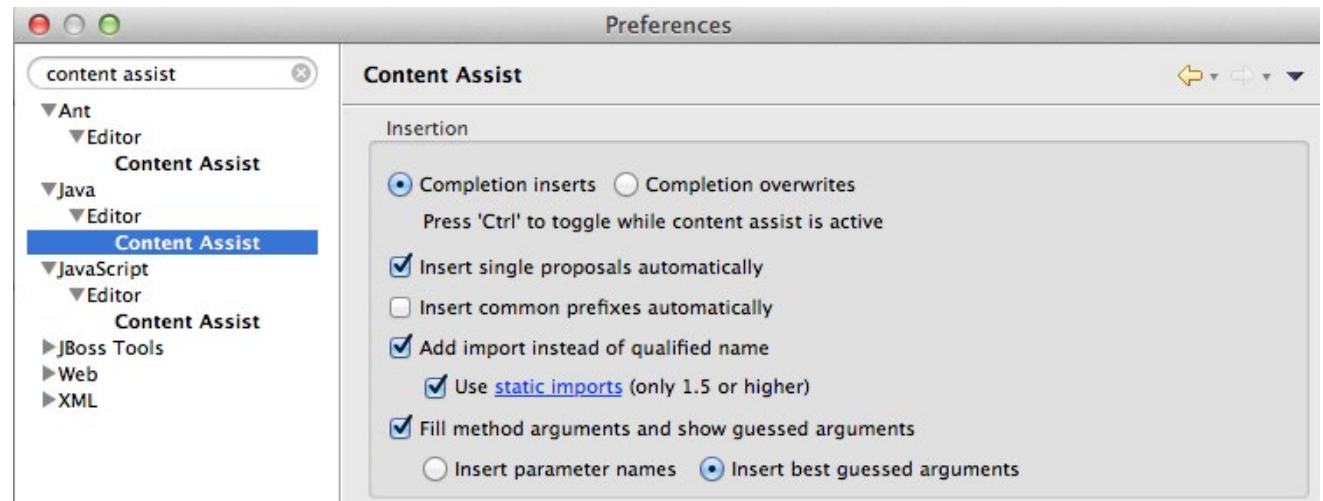
By this point, you should already be quite productive with Eclipse. However there are some settings to tweak to make Eclipse smarter and less irritating.

First of all, when we get a fresh copy of Eclipse to configure, it's smart to go over its preferences and make it work just like any other Eclipse installation you've used before. Muscle memory is super fast and convenient way to leave your brain's cycles for programming.

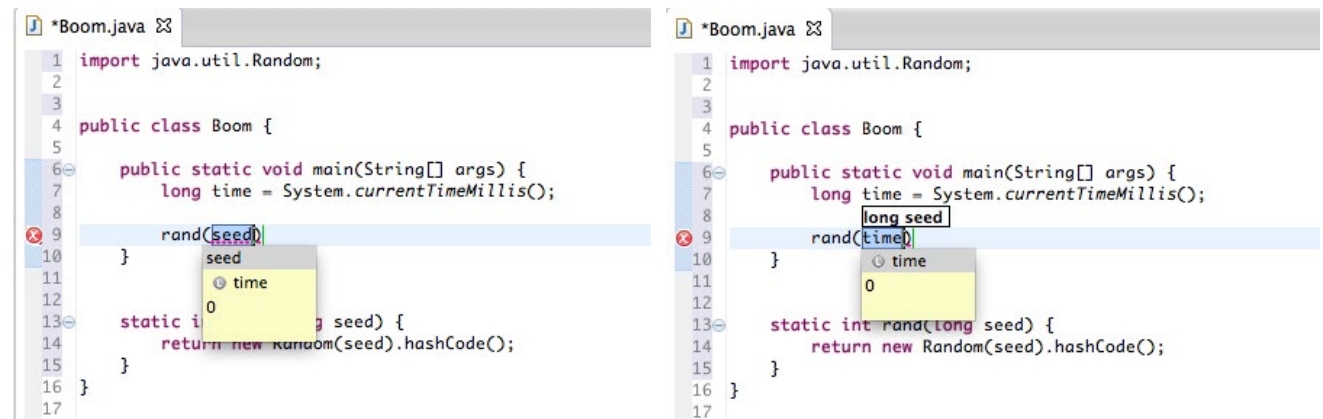
Typing is the one of the main activities, so be sure to make Eclipse automatically insert matching elements for all kinds of double elements: `"`, `{`, `[`, `(`, `<`. You can also make it possible to insert semicolons to the correct position magically. It really helps you type faster when you can finish any expression with *semicolon-enter* from any position inside of expression.



After you've configured the typing preferences according to your taste, another change we prefer to make in Eclipse is to fix up content assist. Sometimes, content assist finishes a method call for you, but instead of guessing which variables were meant to become arguments, it does something unintelligent. In fact, you can configure Eclipse to guess the parameters.



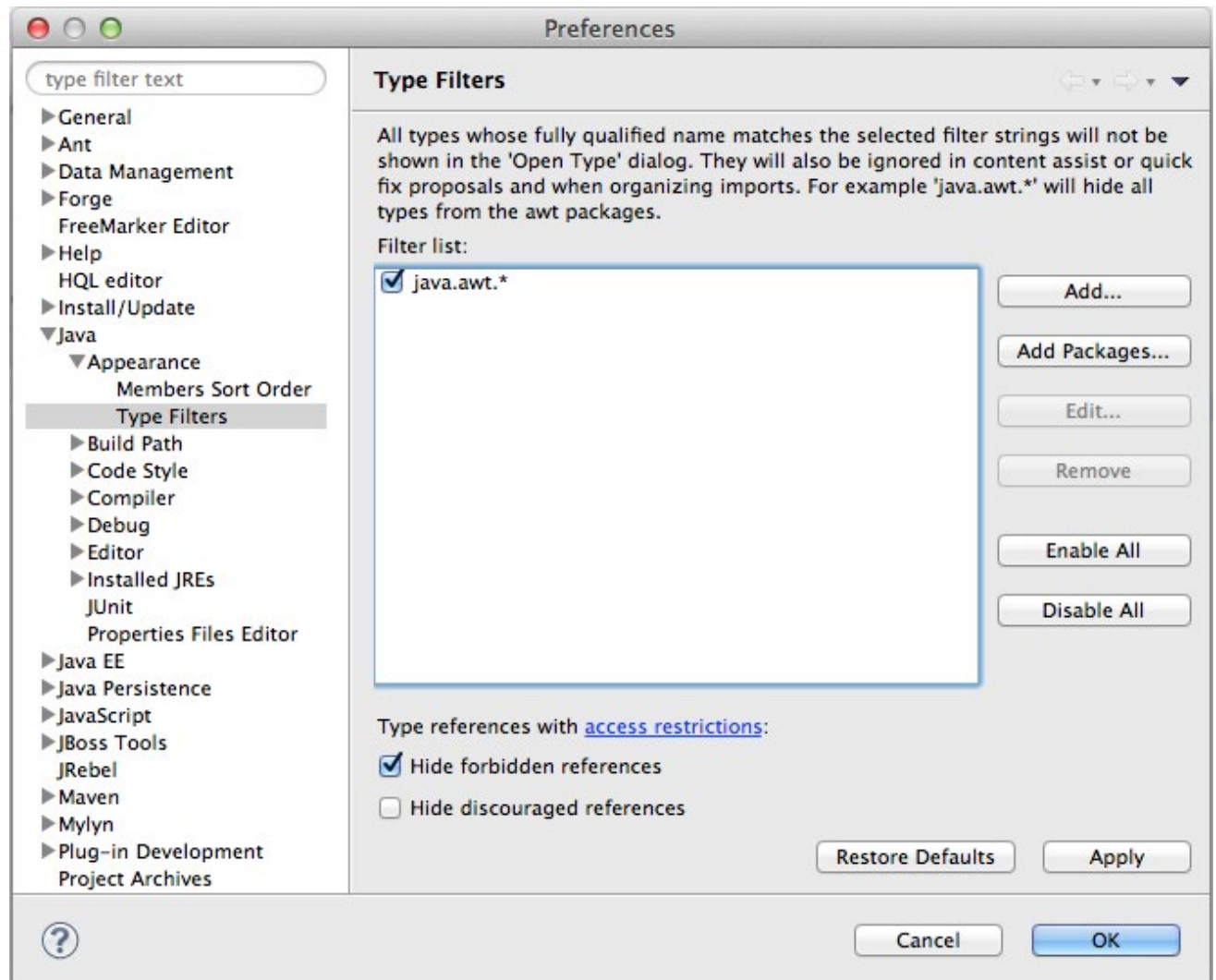
Check out the difference on the images below. On the left, you can see "Insert parameter names" and on the right are best guessed arguments. In our experience, the best-guessed arguments win in 99v%.



Another cool configuration tip is to make use of “Type Filters”. Basically, you can tell Eclipse that there are certain packages that you don’t want to deal with. Yeah, **java.awt.***, we’re looking at you!

You can find Type Filters configuration under: **Preferences > Java > Appearance > Type Filters** or **Ctrl+3 -> type filters**. Packages added there will be removed from content assist pop-ups and automatic import organization.

Now you can safely type **List list = new ArrayList();** and not be frustrated by irrelevant import options.



SAVE ACTIONS

Another useful feature in Eclipse are **Save Actions**. A save action is something that is automatically run on a file when it is saved, which can be quite useful and productive.

- **Format code** in a newer project go for formatting all lines in a saved file; when dealing with a larger, partially non-formatted codebase, format edited lines only.
- **Organize imports** - this allows you to skip importing classes, so you can just save your source file and enjoy the benefits of automation.
- **Don't remove unused code** - we won't suggest to remove unused code from source files, because it will annoy you from time to time.

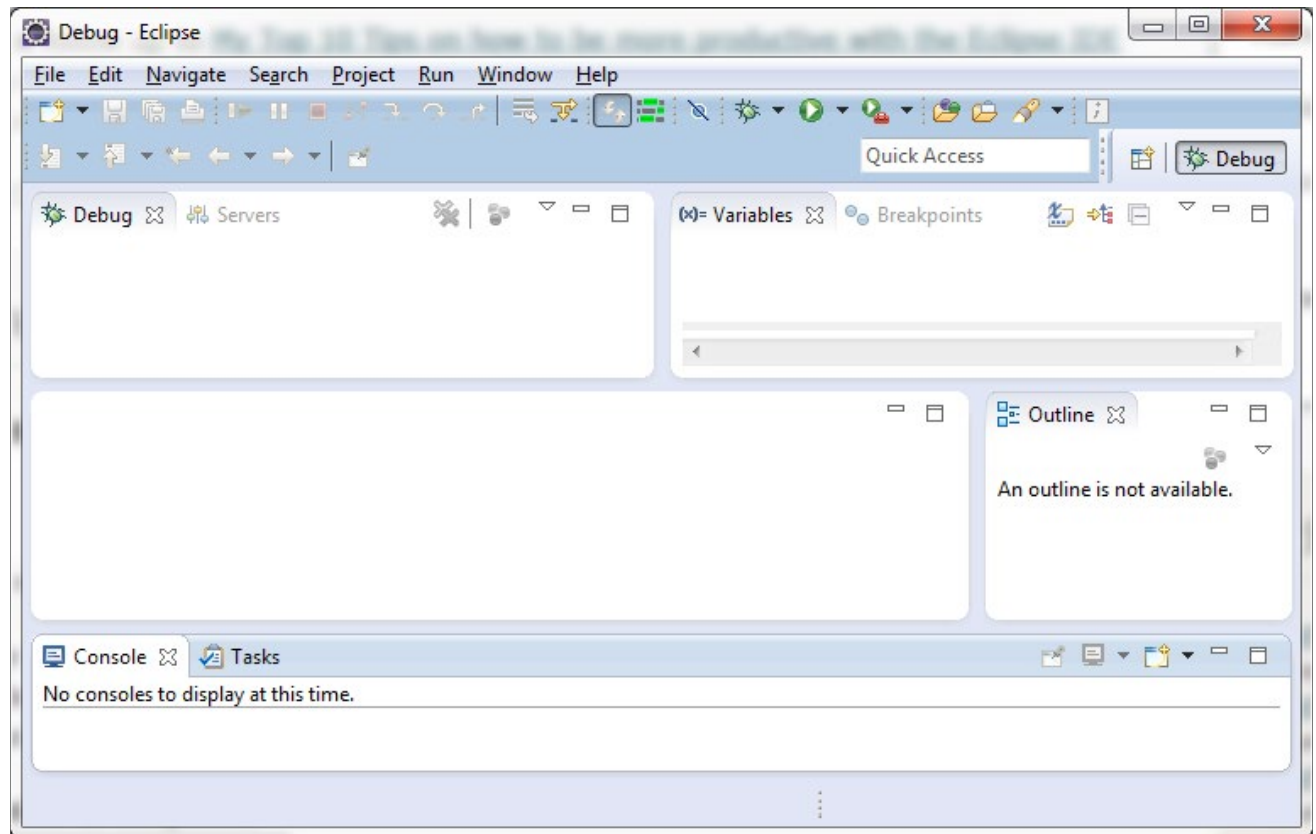
So far, we have a decent basic set of configuration options that you can employ to make your life in Eclipse more productive. We're sure that you can tell us about many more hidden options that make things better, but for now let's look at a few good shortcuts to know, like:

1. Ctrl + 1 - fix problem, open suggestion box
2. Alt + Shift + L - extract a local variable
3. Alt + Shift + I - inline a variable
4. Alt + Shift + M - extract a method
5. Alt + Shift + R - rename
6. Alt + left arrow - select an expression

Masterful use of those will improve the speed of development by a lot. Happy hacking!

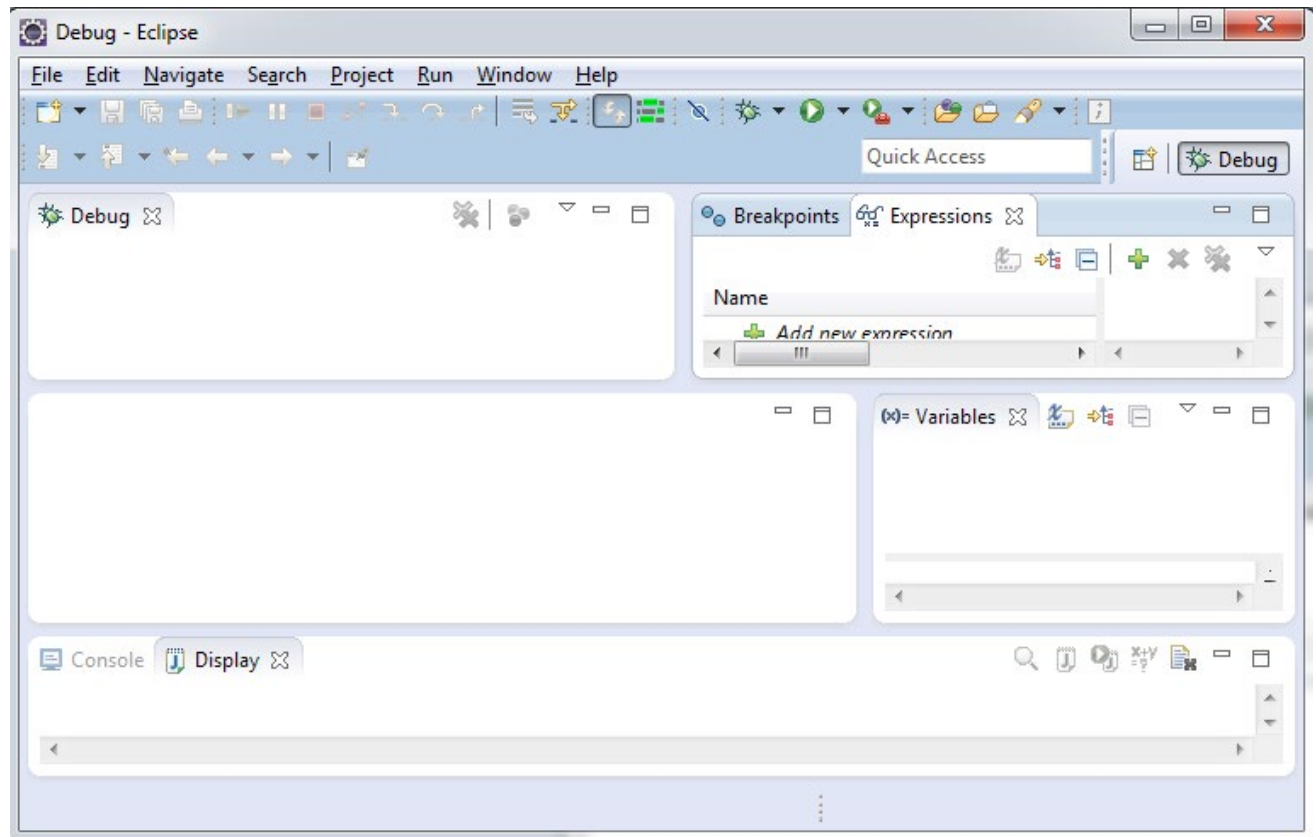
USE YOUR DEBUGGER LIKE A NINJA

Just like anything else, it's good to learn to use the Java debugger in Eclipse efficiently. Firstly, you should open the debug perspective and customize it--by default it has some irrelevant views open and instead hides some rather important ones! Here is what it might look like in the Eclipse 4.3 Kepler Java EE package:

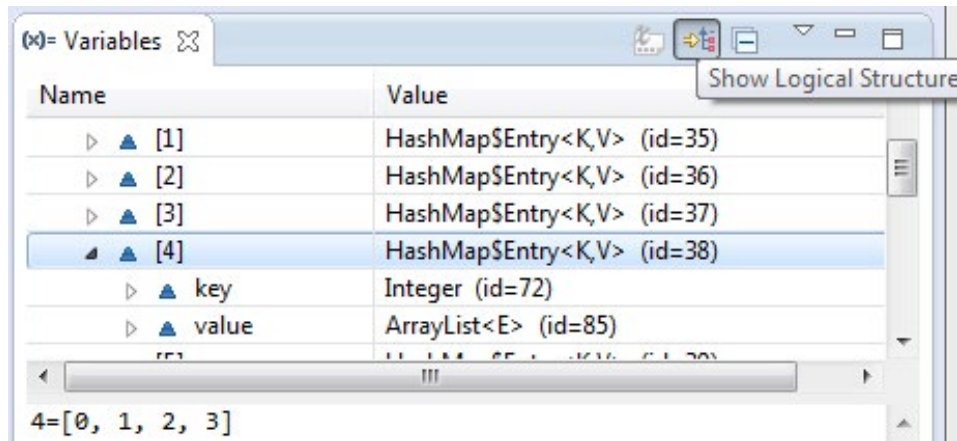


Remember, you always have Quick Outline available (Ctrl+O) if you need it, so why waste screen space for a permanent *Outline* view? We recommend you also close the *Servers* and *Tasks* views. Then you can drag *Variables* where *Outline* used to be, so you can view variables and have access to breakpoints at the same time, and open (Ctrl+3, <view name>) *Expressions* and *Display* views.

The *Display* view is perhaps one of the most important ones in Eclipse, since it lets you type code and evaluate it in the context of the current debug session. This is very useful when dealing with badly documented code with which you have to integrate, or just discovering APIs and seeing what they return at runtime.

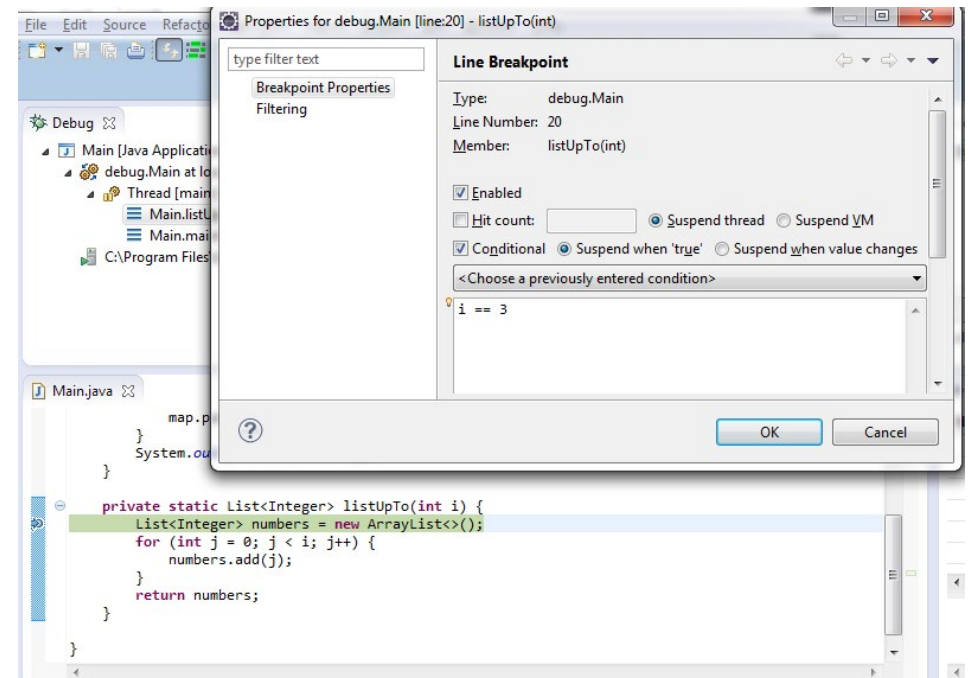


Next, you may want to switch the *Variables* view to show logical structure. This hides the implementation details of classes like `HashMap`, and instead shows you just the keys and values:



You may also want to check out the **Preference** page **Java > Debug** and play with settings for detail formatters, which can make it even easier to see the structure of collections.

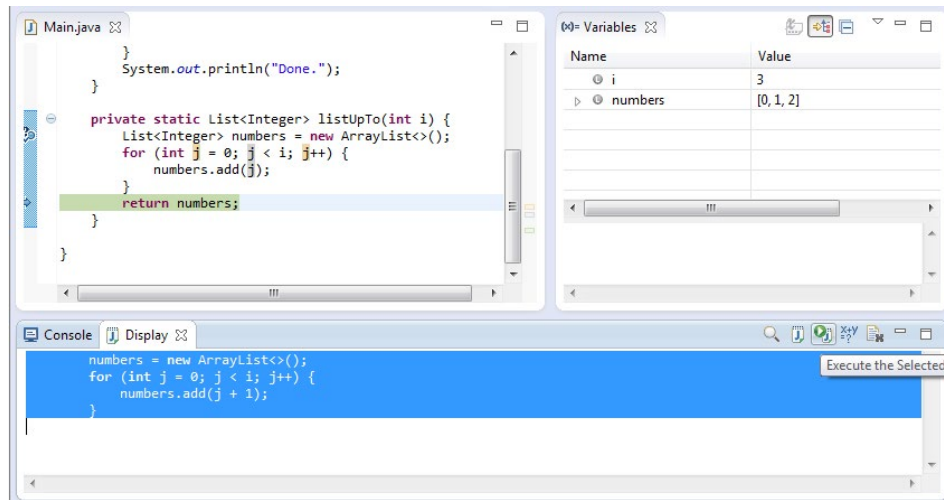
One of the coolest things that you can do with the debugger is change what has already happened and test out potential fixes to code at runtime. For example, this next example has a method that returns a list full of integers up to the number given in a parameter. This method is called with numbers from 0 to 9, but we want to debug a case where the number is somewhat bigger than 0, let's say 3. So first, we set a conditional breakpoint by right-clicking on the breakpoint and selecting *Properties*.



While the loop is already executing during debugging, you may discover that you wanted the numbers starting with 1 instead of 0, and the limit "i" should be inclusive, so you are expecting a list of [1, 2, 3].

The simplest way to do this in this case is to just fix the code (`numbers.add(j + 1)`) and save -- Eclipse will use HotSwap to update the method body and automatically "Drop Frame" to start execution of the method over again.

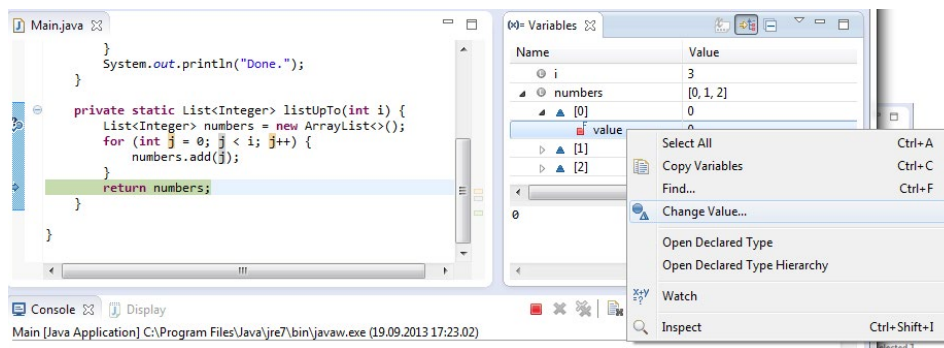
Another way, without saving the change yet, is to let the for loop run its course, then copy the code to the Display view, fix it there and execute it:



But in this simple case, we wouldn't bother and just use the first method with HotSwap and Drop to Frame, but in complex cases you might need a combination of these approaches: letting a fixed code run again, implementing modified code in *Display* view to change a variable, or changing values one by one in Variables view.

With the tools and methods above at your disposal, you should be able to start using Eclipse like a ninja fairly quickly. Next, we summarize what's been said in this report, go over some of the things that Eclipse does that other IDEs don't, and address of the common complaints by Eclipse users and some ways to fix them. And let's not forget, the famous XKCD comic.

Yet another way is to also let the loop finish, and then use the *Variables* view to change the value of each element in "numbers" manually:



TL;DR:

SUMMARY, CONCLUSION AND A GOODBYE COMIC ;-)

It's hard to see Eclipse IDE as a product alone--more than anything else, it's a long-term project and developing platform that grows with its community. It's a massive foundation from which many other tributaries to branch off, and it's quite likely that we could have added another 50 pages to this report as well. But all good things come to an end, and here is what we covered for you TL;DR lovers out there.

Summary of previous sections

PART I: GETTING STARTED WITH INSTALLATION AND MAINTENANCE

This section started off with a quick overview of the Eclipse bundles for Java EE, Java, C, C++ and Android, then went on to options for platform runtime binaries and a bit on OSGi. If you didn't know, the following IDEs here are all based on Eclipse: MyEclipse, IBM RAD and JBoss Developer Studio from Red Hat. It's possible to migrate to a new version of Eclipse without ruining all your previous settings and preferences by installing via an existing installation, and we have some tips here for easily increasing your performance early on and minimizing defaults that we soon found to be annoying. For Scala, Python and Xtend programmers, Eclipse has language packs for you too!

PART II: MAKING ECLIPSE YOUR OWN

Getting Eclipse set up how you want is going to need some plugins and customizations. In this section we introduce the Eclipse Marketplace and some plugins we think that any Java dev can enjoy, like EGit, MercurialEclipse, Eclipse Color Theme, JRebel, Eclipse Code Recommenders, JUnitLoop, InfiniTest and Workspace Mechanic. Then, we show you how to get started off on the road to Java stardom by writing your very own plugin for the Eclipse Marketplace. Then a quick overview of Eclipse's integrations with Application Servers like Apache Tomcat, JBoss AS (WildFly), Jetty, GlassFish, WebLogic and WebSphere (Liberty Profile) plus Build Tools like Maven, Ant+Ivy and Gradle.

PART III: TIPS AND TRICKS FOR USING ECLIPSE LIKE A SUPER-NINJA BAD@\$\$

For you seasoned Eclipse users, we offer some navigational tricks for improving file navigation, class outline, stack trace console and others. You can make your Eclipse installation more clever with configuring typing preferences, content assist, type filters and save actions. Finally, we finish off with some pretty cool ninja debugger actions that you might enjoy.

Concluding thoughts and a little humor

After reading this, you might think that Eclipse is the best thing since sliced bread. While we like to be able to express our opinions, especially pointing out the things in Eclipse that we don't always enjoy, it's not our intention to waste your time with complaints. This report intends to be a source for new and experienced users alike to use their selected tool better, that's all.

WHAT DEVS OFTEN SAY WHEN THEY COMPLAIN ABOUT ECLIPSE

As part of this conclusion, we should acknowledge some of the more common complaints we hear about Eclipse from users, and some brief suggestions for improving upon them.

"Eclipse is slow."

If you still use Juno, please upgrade asap. Other than that, configuring Eclipse to do less unnecessary things helps a bit. Playing with JVM arguments also gives it a boost, and uninstalling plugins that you don't use or building your IDE from Runtime Binaries is a way to get noticeably better performance.

"The UI is clumsy and I don't like it."

Utilize different perspectives and customise them as you would like. If you want to get the maximum working area, you can use **Ctrl+M** on your currently open editor. Check out [fullscreen plugin](#) to enhance that experience.

"Content assist and organise imports keep asking about awt and other irrelevant classes."

Use type filters to prevent Eclipse from suggesting classes that you never use. Use [Code Recommenders](#) plugin to improve what content assist does for you.

SOME THINGS THAT ECLIPSE DOES THAT OTHER IDES DON'T DO

Not every IDE is created equally, and it's worth more than half a groat to estimate the flamewar potential that this conversation brings along with it. But it turns out that Eclipse does a couple things that IntelliJ IDEA, NetBeans, and other IDEs do not.

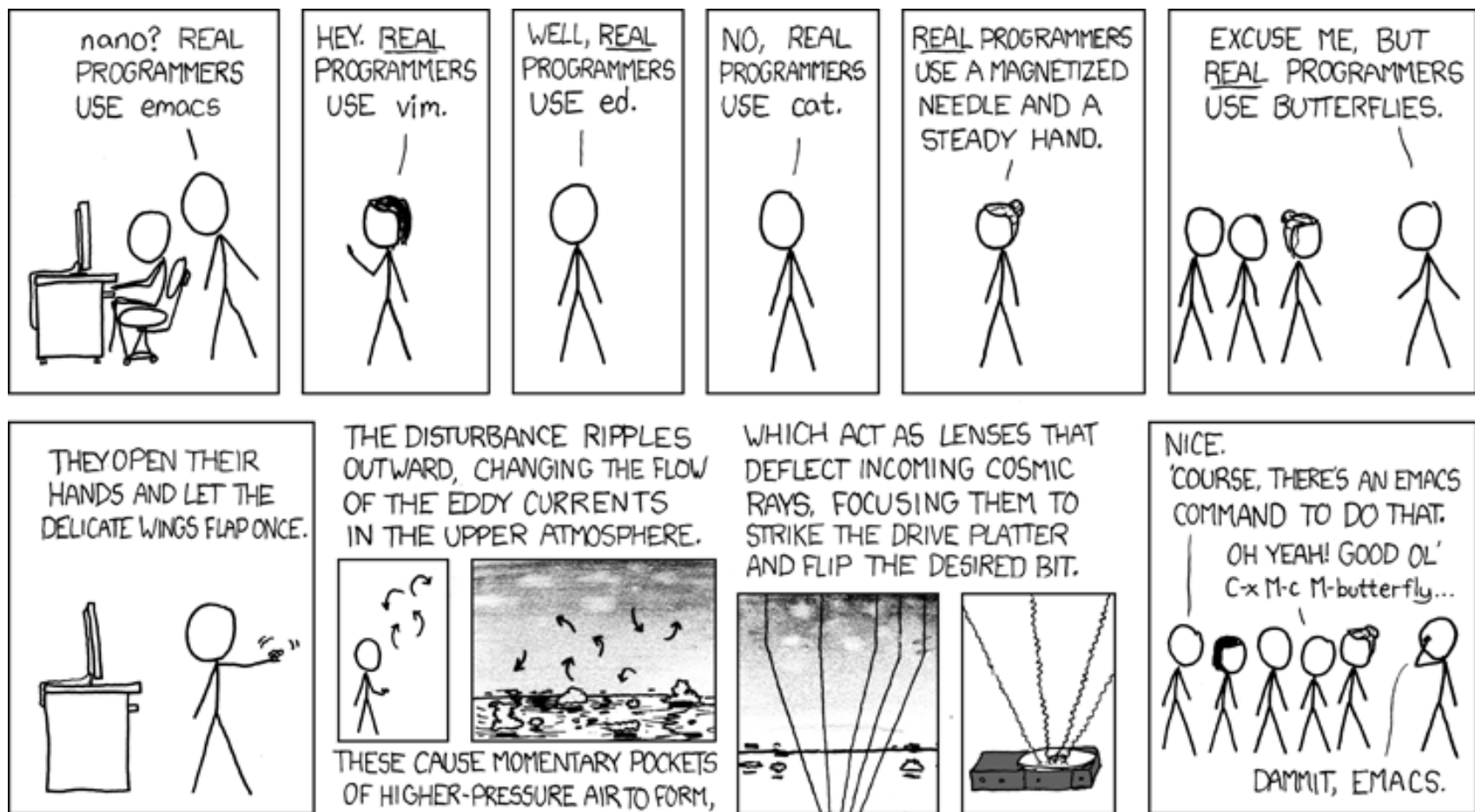
For example, **Eclipse is the only IDE that has its own compiler**, and it's pretty smart about code that contain errors. When your class file doesn't compile, Eclipse inserts an unresolved compilation error, like this one:

```
Exception in thread "main" java.lang.Error: Unresolved compilation
problems:
    Syntax error, insert "AssignmentOperator Expression" to
    complete Assignment
    Syntax error, insert ";" to complete BlockStatements
    a cannot be resolved to a variable
```

However, the class is compiled otherwise and is functional unless you're reaching for the method in question.

This second one is questionable, since we aren't 100% sure at the time of this writing if there isn't some similar functionality hiding somewhere in NetBeans or IntelliJ IDEA (we'll find out!), but we felt it important to mention Eclipse's Scrapbook functionality. Scrapbook is a regular file, not related to any projects which can be used to inspect and evaluate Java code expressions. The cool thing is that with Scrapbook, you can paste pieces of code there and quickly see what that code actually does--essentially, it lets you inspect, evaluate and debug, all in a separate window and away from the larger code base.

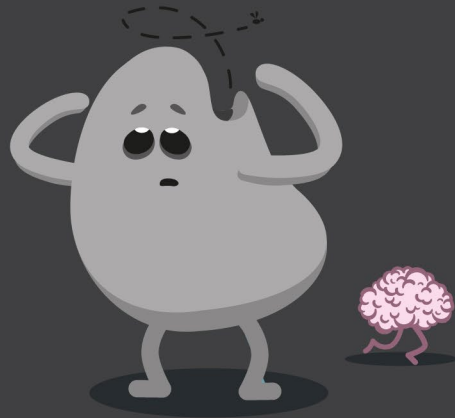
Can emacs do that? ;-)



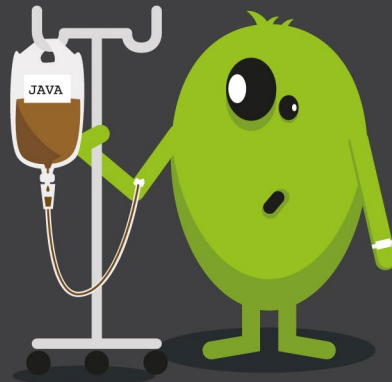
<http://xkcd.com/378/>

JAVA IS CURED

ARE YOU SUFFERING FROM THESE SYMPTOMS?



**OUTOFMEMORY
AMNESIA**



**REDEPLOY
MADNESS**



**CLASSLOADER
CRAMPS**

GET CURED

JRebel
FOR ECLIPSE



↙ Contact Us

Twitter: @RebelLabs

Web: <http://zeroturnaround.com/rebellabs>

Email: labs@zeroturnaround.com

Estonia

Ülikooli 2, 5th floor
Tartu, Estonia, 51003
Phone: +372 740 4533

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:

Oleg Šelajev, Erkki Lindpere, Sigmar Muuga,
Michael Rasmussen, Oliver White,
Ladislava Bohacova