10 time-saving techniques in Eclipse Europa

How to get the most out of the popular IDE

Skill Level: Intermediate

Nathan A. Good (mail@nathanagood.com)
Senior Information Engineer
Consultant

22 Apr 2008

Eclipse Europa offers many convenient features for navigating and editing Java™ files. Discover what they are and how to use them, and when you’re finished, you may be saying, "I didn’t know that!"

Section 1. Before you start

About this tutorial

This tutorial shows how to be more productive with Eclipse Europa, providing 10 ways to speed up your development in Eclipse. Many of the features shown for productivity are either new or improved features introduced in Eclipse Europa.

Objectives

You will learn about how to better use refactoring tools, search tools, workspaces, and code templates. After completing this tutorial, you should be an Eclipse power user — able to be more productive by doing less. Our goal is to show you how to leverage Eclipse features to perform mundane tasks.
Prerequisites

It helps if you have used Eclipse before and have done a bit of Java development. You don't have to be an expert, but you should know enough about the Java programming language to recognize the basic syntax and paradigms to get the most out of the refactoring parts of this tutorial.

System requirements

To follow the examples, you need Eclipse Europa and one of the operating systems that Eclipse supports — Mac OS X, Microsoft® Windows®, or Linux®. You also need a Java Runtime Environment (JRE) — at least JRE for Java V5 is recommended.

Section 2. A brief introduction to Europa and its shortcuts

Eclipse Europa

Prior to the V3.2 release of Eclipse, the releases of various plug-ins weren't necessarily organized. In fact, one of the complaints against Eclipse was that you would download the IDE, then spend a lot of time downloading and installing the plug-ins that made it useful for you. That changed with the first simultaneous release of many projects: Callisto. Eclipse Europa is the second simultaneous release of projects, which brings Eclipse up to V3.3 and is an improved set of standard projects organized in the Europa discovery site by the type of development you want to do.

To do dynamic Java Web development, like JavaServer Faces (JSF) or JavaServer Pages (JSP), all you have to do is download the latest version of Eclipse and install the Web development features. To do C and C++ development, there's a set of features for that, etc.

After you have installed Eclipse Europa, you can start to harness the power of having a really great full-featured IDE that has many productivity features you'll read about here.

Take the shortcut
Keyboard shortcuts can save you a lot of time over the course of the day, especially when it comes to executing a command or function when your hands are already on the keyboard. But one thing that is not easy is to remember is all the shortcuts. On one hand, it’s much faster to press Ctrl+Shift+O to organize your imports in Java class files automatically. And on the other hand, it’s not any faster if you have to comb through help or find your shortcut cheat sheet to find out which shortcut to use.

In Eclipse, you only have to remember one shortcut for your platform: Ctrl+Shift+L if you’re on a Windows or Linux computer, or Command+Shift+L on a Mac. That shortcut provides a "heads-up display" of the other shortcuts for you right in your IDE. When the shortcuts are displayed, you can scroll through them with the arrow keys and execute them by pressing Enter while the command is highlighted. The shortcut is displayed next to the command, as shown in Figure 1, so you can remember it for next time.

Figure 1. The built-in list of commands and their keyboard shortcuts
Table 1 provides some very useful keyboard shortcuts in Eclipse Europa.

**Table 1. Shortcut key combinations in the Java editor**

<table>
<thead>
<tr>
<th>Shortcut keys</th>
<th>What they do</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+R</td>
<td>Locates resources, such as files.</td>
</tr>
<tr>
<td>Ctrl+T</td>
<td>Locates types, such as classes, including those within and outside your source.</td>
</tr>
<tr>
<td>Ctrl+O</td>
<td>Lists the methods and properties in the Java</td>
</tr>
<tr>
<td>Hot Key</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Ctrl+1</td>
<td>When an error is highlighted, provides suggestions for fixing the error.</td>
</tr>
<tr>
<td>Ctrl+Space</td>
<td>Lists suggestions for templates, methods, or fields while you're typing.</td>
</tr>
</tbody>
</table>

### Using hot keys

**Customizing your shortcuts**

You can change which keys you press to execute commands. Either press `Ctrl+Shift+L` again after the list of shortcuts is displayed or change them in the Preferences window under Windows > Preferences > General/Keys.

The ability to find menus, commands, and preferences with the click of a hot key is one of the most exciting new features in Eclipse Europa. For instance, what if you've forgotten where the menu item is for cleaning up code or forgotten where the preferences page is for configuring the settings? All you have to do is press `Ctrl+3` (or `Command+3` on a Mac), and a list appears. Start typing what you're looking for, such as `clean`, and the results appear for commands, menus, and preferences that match.

**Figure 2. A list of menus, commands, and preferences**
Section 3. The Search menu: Tools for spelunking

Once I was talking with a colleague about having to remember how a large project's package structure was organized so I could quickly find the Java classes I needed. He said to me, "When I need to go spelunking, I generally just use search tools, instead." The term stuck with me, so whenever I refer to diving into the code to find something, I call it *spelunking*. It makes me feel adventurous and glorifies my search
for missing code.

Eclipse has powerful search tools, so much so that it can improve how your code is organized. You won't need to worry about using naming conventions or packages (directories) to make artifacts easier to find; rather, you use packages to group artifacts by domain or purpose.

The reason the Search menu is so powerful is that it allows you to do much more than search for text in files. You can use a Java search to look for method declarations, JUnit tests that refer to your class or method, or even for other code that references a method you have highlighted.

Find occurrences

Look at the Search menu in Figure 3. The powerful features discussed here are References, Declarations, Implementors, Read Access, and Write Access.

Figure 3. The Eclipse Search menu

Each of these searches has a submenu that allows you to quickly determine the scope of the search. The search scopes and what they include are listed in Table 2.

<table>
<thead>
<tr>
<th>Search scope</th>
<th>What it includes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workspace</td>
<td>Includes all the files in the current workspace.</td>
</tr>
</tbody>
</table>
Project

Searches only for occurrences inside the current project.

Hierarchy

Searches only in the current class's or interface's hierarchy, which limits it to parent classes, implementing classes, or subclasses.

Working Set

Searches only in the chosen working set (learn more about working sets in the section "Game, set, match").

Take a look at the simple interface Cow and the implementing class Hereford, shown in listings 1 and 2, respectively. The listings help you understand what type of results you can expect to see for the various types of searches.

Listing 1. The Cow interface

```java
public interface Cow {
    void moo();
}
```

The Hereford class implements the Cow interface and has a couple of methods for example purposes.

Listing 2. The Hereford class

```java
public class Hereford implements Cow {
    private String name;

    public void moo() {
        String message = "My name is " + name + ". Moo!";
        System.out.println(message);
    }

    public String getName() {
        return name;
    }

    public void setName(final String name) {
        this.name = name;
    }
}
```

References

Given the two listings, if you highlight name in Listing 2 and search for references, the results look like Figure 4, with method names that include some kind of reference to the name variable.

Figure 4. Search results for references
Declarations

If you highlight the `setName()` method and search for declarations within the project, the search results include all classes that have a method with the same name. Searching for declarations of `moo()` within the hierarchy displays the results shown in Figure 5, limiting the scope of the search to the current class and the interface that it implements (Cow).

Figure 5. Declaration search results

Implementors

By searching for implementors, you can find classes that implement an interface. This type of search works only when you highlight an interface name. If you highlight `Cow` and search for an interface, the `Hereford` class will be the only result listed in the Search window.

Read access
Performing a search for *read access* finds any places in which the variable you have selected is read. I've used it many times when debugging large classes in which I want to know what other code is using the value of the variable. Searching for read access of the `name` variable in Listing 2 gives you the results shown in Figure 6.

**Figure 6. Read-access search results**

Write access

Like read access, *write access* searches for occurrences of a variable or field, except that it limits the search to occurrences in which the variable is being set. Searching for `name` in Listing 2 gives you the results shown in Figure 7.

**Figure 7. Write-access search results**

Tweaking the search results

The Search tab in the results can be filtered and modified in pretty powerful ways to show you only the results you want to see. If you click the arrow on the Search tab, you get a menu that allows you to quickly filter the results by several criteria.
The Java search

The Search menu item displays a search window containing many tabs. The Java Search tab is one example of a language-specific search (a PHP search is available if you install the PHP Development Tools plug-in). It allows you to perform powerful searches by searching for the string in a language-specific context. In the Java programming language, this means that you can search for types, methods, fields, etc. Then, you can further limit this search to declarations, references, and more. Figure 9 shows the Java Search tab.

Figure 9. The Java Search tab
Section 4. Going back in time

I love food, which means I can't wait to go to lunch. Going out to eat is one of the high points of my day, except when I get back from lunch, sit down, and my unit tests won't run successfully. "I know I had this working before I left. What happened?" I say to myself. When I was using other IDEs, I would start using the Undo command, hoping it would get me back to where my code was working (before I got distracted by lunch), but at the same time not undo something important.

Comparing with local history

Fortunately, in Eclipse, there is a much better way to compare and restore your code to its previous glory. To use this feature, highlight the file you want to compare and
choose **Compare With > Local History**. The History tab appears, as shown in Figure 10 with the dates and times the file was saved. Double-click one of these times to access the compare editor. Using the compare editor, you can manually restore particular changes.

**Figure 10. The History tab showing file-save times**

![Figure 10](image)

**Replacing with local history**

To replace the entire file with a version from your local history, click **Replace With > Local History**, then select the version you want to use as a replacement. The compare editor gives you the chance to preview the changes, so you don't have to make blind replacements.

**Section 5. Time for a new perspective**

Perspectives give you ability to set up the IDE's windows and views, then save that setup.

**Use the right perspective**

By using different perspectives to perform different tasks, you can quickly view the types of windows, editors, or navigation trees that help you with your task. There's a Java perspective for writing Java code, a Debug perspective that changes the IDE to show breakpoints and stack traces, and a Team Synchronizing perspective that allows you to quickly compare code and repositories. To open any one of these perspectives, click **Window > Open Perspective**.

**Customize your work environment**
If you’re used to a different IDE and starting to use Eclipse Europa, you might want certain views in certain places. You might prefer the Package Explorer on the right side instead of the left, and you might prefer not to see the Outline or Task views at all. You could modify the Java perspective, but if you reset the perspective to its default settings, you’re sunk.

Pair programming?  
When it comes to perspectives, one thing I’ve noticed is that most people like theirs tweaked a bit from the default. If different people “drive” from time to time, the ability to create your own perspective is great: You can set up one just for you. It’s like having presets for a driver’s seat and mirror position in some cars.

Instead of modifying the default perspectives that come with Eclipse Europa, you can create a new perspective — just for you — and set that one up to match your preferences.

To set up your own perspective, first find one that makes a good starting point, like the Java perspective. Set it up the way you want by closing views, opening others, and moving tabs around. Now create your own perspective by choosing **Window > Save Perspective As**, then type a name. Now if you open other views, but want to return your custom perspective to the state at which you initially saved it, you can choose **Window > Reset Perspective**.

---

**Section 6. Hire a maid**

**Clean up**

From Callisto to Eclipse Europa, the code cleanup utilities in Eclipse have been much improved. By using the automated code-cleanup tools in Europa, you can automate many of the code tasks that make your source code better. Not only can you configure your own settings for code cleanup but you can save them in a format you can share with others. Being able to import the code-cleanup settings is invaluable in a team environment because it allows the entire team to have the code formatted and cleaned up in the same way.

To clean up your code, choose **Source > Clean Up**. If you already have a profile set up, you can select it from the list.

**Edit and save your own profile**
If you don’t have a profile set up for cleaning up code, you can easily create a new one. To create your own profile, click **Configure** in the Clean Up wizard, as shown below.

**Figure 11. The Clean Up wizard**

![Clean Up wizard](image)

After you click **Configure**, a window like the one shown in Figure 12 appears.

**Figure 12. Configuring the Clean Up wizard**

![Configuration window](image)
Select the **Enable project specific settings** checkbox, then click **New**. Name the profile (**MyCleanupProfile**, for example), then click **Next**. Eclipse Europa displays the Profile Editor. Using this editor, you can modify settings and see the changes in the Preview pane on the right, as shown in Figure 13. When you’re finished with the settings, click **OK** to save your new profile.

**Figure 13. The Preview pane**
When you're back in the **Clean Up** window, you can either click **Finish** to apply the clean-up profile to your code or you can click **Next** to preview the changes. Previewing the changes allows you to clear any that you don't want to make in the **Changes to be performed** list.

### Clean up on save

To make sure your code is always cleaned up consistently, you can configure Eclipse Europa to clean up your code whenever you save your work. To perform formatting and clean-up actions on save:

1. Open the project's properties.
2. Expand **Java Editor** and click **Save Actions**.

*Figure 14. Changing Java Editor Save Actions*
3. Select the **Enable project specific settings** checkbox.

4. Select the **Perform the selected actions on save** and **Additional actions** checkboxes.

5. Click **Configure** and select the same options for cleaning up your source.

---

**Section 7. Let the IDE do the work**

Put the Eclipse Europa IDE to work for you by using the commands in the **Source** menu, creating templates for frequently used code and modifying the build.xml file.

**The Source menu**

The Source menu provides many useful commands for generating code, from public accessors to constructors to overridden methods.
Override/Implement methods
This command helps you generate methods to override those in the base class.

Generate setters/getters
This command inserts public accessors for the private fields in the class in the current editor.

Generate delegate methods
This command inserts methods that allow you to create methods that expose internal functionality easily.

hashCode() and equals()
These generated hashCode and equals() methods allow you to select which fields in the class you want to use in the comparison for equality. This is perfect for business objects, in which equality might be based on a couple of attributes of the object. For instance, an object may be considered equal to another if the id field is the same in both.

Constructor using fields
Sometimes you want to initialize certain fields in an object's constructor. If you haven't created any constructors, use the IDE to generate them for you.

Constructor from superclass
This command inserts constructors that mirror those in the superclass. Like most of the other generate commands, this one allows you to select which constructors from the superclass you want to include in the current class.

Templates
Code templates allow you to insert sections of code into your editor, saving you time spent typing and allowing you to benefit from reproducible sections of code. To insert a snippet, just start typing one of the template names, then press Ctrl+Space to display the selection list of templates.

You can build your own templates by using the template Preferences window, which you access by expanding Java, expanding Editor, then clicking Templates, as shown in Figure 15. To add a new template, click New, then type a name and description for your template. Next, type the code you want to use as a template in the Pattern field.

Figure 15. Editing Java templates
Variables allow you to insert placeholders in the code pattern that automate some of the work. Consider the example of a static declaration of a logger shown below.

**Listing 3. Inline logger declaration**

```java
private static Log logger = LogManager().getLog(Hereford.class);
```

In this listing, the `Hereford.class` was automatically inserted as using the variable `${type}`.

**Build the project**

Setting up your own builder in Eclipse Europa gives you the ability to execute a particular process whenever the project is built. In the preferences pages for builders, you can specify the order in which the builders will run, so if you make a builder that generates source code, for instance, you can make sure to run that before the project's compiler runs.

**Building useful Ant files**

The Targets tab allows you to have pretty fine-grain control over which build targets are run during specific times. Use this to your
advantage and keep it in mind when you’re building your build files. For instance, my simple example should be changed to have a clean target with the delete task in it. That way, I can specify the clean target in the During a Clean in the properties for the new Ant builder.

Specifying an Ant file to build is one of the easiest ways to integrate a process into your build. The following section shows how to add a builder that automatically packages your compiled source into a Java Archive (JAR).

The build.xml file

The build.xml file has one target and task in it. Here's an easy way to get going with the build.xml file:

1. Create a new XML file. In the empty file, press Ctrl+Space to display a list of common tasks, then choose Buildfile template from the list. Listing 4 shows the build.xml file after removing the comments and extra target and adding the jar task. This task packages the compiled class files into a JAR placed in the dist directory.

Listing 4. A simple build.xml file

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project name="savingTime" default="dist">
  <description>
    A project showing all the features in Europa that save time.
  </description>
  <target name="dist">
    <delete file="dist/savingTime.jar" failonerror="false"/>
    <jar destfile="dist/savingTime.jar">
      <fileset dir="bin">
        <include name="**/*.class"/>
      </fileset>
    </jar>
  </target>
</project>
```

2. With the build file complete, choose Properties > Builders. For a standard Java project, the Java builder is the only one in the list by default. To add an Ant builder, choose Ant Builder from the list, click New, then click OK.

3. In the Edit launch configuration properties window, shown below, type the name of the new builder — My Jar Builder, for example.

**Figure 16. Launch configurations**
4. Click **Browse Workspace** to find your newly created build.xml file.

5. On the **Refresh** tab, select **Refresh resources upon completion**. Choosing to have the resources refreshed automatically means that when the build is complete, the IDE refreshes the workspace so any new or updated files that the build generated are available.

6. On the **Targets** tab, click **Set Targets** next to the **Auto Build** box and make sure that nothing is selected. If you're refreshing resources automatically and building automatically, you'll get stuck in a build loop because the build starts whenever the
workspace is refreshed.

7. After setting up the builder, save it and click **OK**.

---

**Section 8. Game, set, match**

*Working sets* are sets of projects in your workspace that allow you not only to organize your work into similar groupings but also to limit search scope and build scope. Also, you can set up the Project Explorer to show your projects organized by working sets. An example of using working sets to organize your workspace is to define two working sets called *UI Layer* and *Services Tier* in which you're working with several projects that make up either the UI or services.

Create a working set

To add a new working set:

1. Right-click in Project Explorer, then click **Assign Working Sets**.
2. In the **Working Set Assignments** window, shown below, click **New**. Type a name for the working set, then select the projects you want to include in the set.

*Figure 17. Working Set Assignments*
3. Click **Finish** when you're done, then click **OK** to close the **Working Set Assignments** window. Now that you have a working set assigned, you can organize the Package Explorer by working sets.

4. Click the menu in the upper-right corner of the view, then choose **Top Level Elements > Working Sets**.

5. From the same menu, choose **Configure Working Sets** to select the working sets you would like to see in your Package Explorer.

**Search by working set**

As you saw earlier, nearly all the search commands allow you to scope your
searches by working set, meaning that searches in large workspaces can take considerably less time. The common working-set selection available for searching allows you to select more than one at a time.

Build by working set

If you cleared the Build Automatically checkbox in the Project window, you can manually build your projects. You can also manually build working sets, which saves you time in large workspaces. Manually building a working set limits Eclipse to building only the projects defined in your working set, instead of the entire workspace.

Section 9. Painless refactoring

In most projects, refactoring code is necessary as features are added, defects are remediated, or iterations are completed. Eclipse already had significant refactoring tools, but Eclipse Europa has added even more to the Java tools.

Parameter-object refactoring

Parameter-object refactoring gives you the ability to easily introduce a parameter object, thus simplifying a method signature. Consider Listing 5.

Listing 5. Before parameter-object refactoring

```
public class Barn {
    private String color;
    private int stories;
    private int numberOfDoors;

    private Barn() {}  
    public static Barn createBarn(String color, int stories, int numberOfDoors) {
     Barn b = new Barn();
     b.color = color;
     b.stories = stories;
     b.numberOfDoors = numberOfDoors;
     return b;
    }
    /* Rest of the class ... */
}
```
When building the **Barn** object, I must specify several things. But maybe I might want to specify more or less in the future to make my factory method (`createBarn()`) smarter, and I don’t want to have to change the method signature all the time or have a bunch of overloaded `createBarn()` methods hanging around.

To introduce a parameter object, right-click the method — in this case, `createBarn()` — and choose **Refactor > Introduce Parameter Object**. Type the name of the parameter object (**BarnInfo**, for example) in the **Introduce Parameter Object** window, as shown in Figure 18. If you want public accessors created on the new class, select the **Create getters** and **Create setters** checkboxes. The destination determines whether the new class will be created as a top-level class or as an inner class created inside the one you’re modifying (in this case, **Barn**).

**Figure 18. Introducing a parameter object**
You can keep the original method around as a delegate and at the same time mark it as deprecated. This may be a good idea if you think other people might be using the old method outside your code base and you want to extend the professional courtesy of making your class backward-compatible. Click Preview to see all the modifications that will be made so you have a chance to back out if you don't like what's going to be done.
After refactoring, the new class and changes will look like those shown below.

Listing 6. After parameter-object refactoring

```java
public class Barn {
    private String color;
    private int stories;
    private int numberOfDoors;
    private Barn() {}
    public static Barn createBarn(BarnInfo barnInfo) {
        Barn b = new Barn();
        b.color = barnInfo.getColor();
        b.stories = barnInfo.getStories();
        b.numberOfDoors = barnInfo.getNumberOfDoors();
        return b;
    }
    /* Rest of the class... */
}
```

Inline refactoring

*Inline refactoring* is a smart refactoring tool that attempts to put local values assigned to local values directly inline, skipping the creation and usage of a variable altogether. The example code in Listing 7 demonstrates a local variable — `message` — being created and uses it in the `System.out.println()` call one line later.

Listing 7. Declaring a local variable

```java
public void moo() {
    String message = "My name is " + name + ". Moo!";
    System.out.println(message);
}
```

To remove this local variable, click `message` in the `System.out.println` method, then choose Refactor > Inline. Eclipse prompts you to replace the occurrence, and you can click either Preview or OK. Preview shows the changes about to be performed so you can cancel them if they’re not entirely what you had in mind. After refactoring, the code will look like that in Listing 8.

Listing 8. After inline refactoring

```java
public void moo() {
    System.out.println("My name is " + name + ". Moo!");
}
```
Section 10. Summary

Eclipse Europa has many features you can use to get your work done more quickly. From refactoring to searching to customizing the IDE itself, you have many tools available for increasing your productivity.

Many of the features listed in this tutorial are new to Eclipse Europa, and the others have seen wonderful improvements. For a list of more new features in Europa, see Resources or the help included with the IDE.
Resources

Learn

- **Europa Simultaneous Release project** is a great place to start learning more about Eclipse Europa.
- Read "A whirlwind tour of Eclipse Europa," which provides a quick tour of the features in Eclipse Europa.
- The [online Eclipse help](https://www.eclipse.org) has more information about what's new as well as additional tips and tricks.
- Check out the "Recommended Eclipse reading list."
- Browse all the [Eclipse content](https://www.ibm.com/developerworks) on developerWorks.
- New to Eclipse? Read the developerWorks article "Get started with Eclipse Platform" to learn its origin and architecture, and how to extend Eclipse with plug-ins.
- Expand your Eclipse skills by checking out IBM developerWorks' [Eclipse project resources](https://www.ibm.com/developerworks).
- To listen to interesting interviews and discussions for software developers, check out developerWorks [podcasts](https://www.ibm.com/developerworks).
- Stay current with developerWorks' [Technical events and webcasts](https://www.ibm.com/developerworks).
- Watch and learn about IBM and open source technologies and product functions with the no-cost [developerWorks On demand demos](https://www.ibm.com/developerworks).
- Check out upcoming conferences, trade shows, webcasts, and other [Events](https://www.ibm.com/developerworks) around the world that are of interest to IBM open source developers.
- Visit the developerWorks [Open source zone](https://www.ibm.com/developerworks) for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.

Get products and technologies

- Check out the latest [Eclipse technology downloads](https://www.ibm.com/developerworks) at IBM alphaWorks.
- Download [Eclipse Platform and other projects](https://www.eclipse.org) from the Eclipse Foundation.
- Download [IBM product evaluation versions](https://www.ibm.com/developerworks), and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- Innovate your next open source development project with [IBM trial software](https://www.ibm.com/developerworks), available for download or on DVD.
Discuss

- The Eclipse Platform newsgroups should be your first stop to discuss questions regarding Eclipse. (Selecting this will launch your default Usenet news reader application and open eclipse.platform.)
- The Eclipse newsgroups has many resources for people interested in using and extending Eclipse.
- Participate in developerWorks blogs and get involved in the developerWorks community.

About the author

Nathan A. Good
Nathan Good lives in the Twin Cities area of Minnesota. When he isn't writing software, he enjoys building PCs and servers, reading about and working with new technologies, and trying to get all his friends to make the move to open source software. When he's not at a computer (which he admits isn't often), he spends time with his family, at his church, and at the movies. Visit his Web site.