

# Using Eclipse Effectively

*Handout by Lekan Wang*

Eclipse is an immensely powerful IDE for Java development, but it can be daunting for novice users. If you took CS106A at Stanford and/or read Handout #3: Eclipse Starter, then you should already be familiar with its basic functionality—autocompletion after a dot, organizing your files, on-the-fly compilation with suggestions for warnings and errors, a great debugger, etc—all of which simplify your coding life. This handout assumes that you are familiar with these most basic features.

But, they say that watching a power-user of Eclipse navigate the IDE is like watching Neo from the matrix making code bend to his will from sheer mental will and badassitude.

This guide will not attempt to teach you how to dodge bullets in the Matrix, but will quickly introduce you to some techniques and tips that power-users of Eclipse are generally familiar with. All shortcuts will be **bolded**, and will generally include a short example of when it could come in handy if not immediately obvious.

Alls shortcuts described here will be PC shortcuts. However, on Macs, generally just replace Ctrl with the ⌘ (Cmd) key and Ctrl-Shift with Cmd-Option.

## Useful Views

Views can be selected by going to Window->Show View.

### Useful Views for the Java Perspective

The editor view will obviously be your most used view. What's useful sometimes is to drag an editor view to the lower half of your editor view so you have a split editor. This way, you can refer to some code while editing another piece of code. If you maximize the editor view by double-clicking the tab, or by using the shortcut **Ctrl-M**, all editor views will maximize, keeping the split view.

On the left sidebar generally is your Package Explorer. If you prefer a less Java-centric view, and instead want a view closer to a folder browser like Windows Explorer or the Mac Finder, then the Navigator view is for you. In both of these views is a “Link with Editor” button, signified by a dual left-right arrow. If this is checked, the Package Explorer or Navigator will automatically keep selected whatever file you are currently editing in the editor window. If you prefer to keep this unchecked, clicking this button is a quick way to “locate” the currently edited file in your folder/package hierarchy.

Often also on the left side is the Hierarchy view, which displays a selected class's type hierarchy. To use this view, position your cursor over any declaration or reference to an object. You can then press **F4** to show that class in the type hierarchy. Similarly, if your cursor is over a method name or instance/class variable, pressing **F4** will show the current class in the type hierarchy, and also

highlight the method or variable the cursor is over. This view is especially handy when you are working with an abstract class or an interface, and would like to see the concrete implementations of it.

At the bottom bar, I almost always have visible the Problems view. I suggest clicking the View menu (the down-facing triangle) on the far right of the tab bar, and selecting “Show->Errors/Warnings on Selection.” This way, this view will only show the problems contained in the currently selected file, and makes this view much more usable for larger projects.

The Task List is also useful if used properly. If you simply type the keyword “TODO” in any comment, Eclipse will bold it and interpret that as a “task” and include that comment line in the task list. Using these TODO tasks is an easy way to mark areas of code that need more work later.

The Declaration view shows how the highlighted variable, method, or class was declared.

On the right bar, the Outline view shows a very compact overview of all class and sub-class members.

Finally, remember that all views can be moved, so create an environment you are comfortable with by dragging around the tabs wherever you would like.

### **Useful Views for the Debug Perspective**

In the debug perspective, many of the views are familiar, but there are a few more in addition that are useful.

At the top-left, the Debug view shows all currently running processes, with the stack traces.

At top-right, the Variables view shows all variables in scope. Note, that, to access instance and class variables, you must look under the `this` variable. Also at top-right is the Breakpoints view, which should be pretty self explanatory. This view helpfully allows you to disable certain breakpoints without deleting them so you can re-enable them later. You can also right click on a breakpoint, choose “Breakpoint Properties...,” then specify a breakpoint condition. This is very handy when you have some loop that is throwing an exception after an unknown number of iterations, so with a breakpoint condition in place, you can selectively hit the breakpoint only when your code is about to be in a potentially bad state.

At bottom, you have the Console view. However, many prefer putting the Console on the right side, where the Outline view usually is, because they really like having the Display view always open at the bottom.

The Display view is perhaps my favorite Eclipse view, and allows you to execute arbitrary segments of code at the current program state. Just type some legal code—autocompletion and most of the normal code assistance works fine here—and highlight it. Then press **Ctrl-U** to execute it, or **Ctrl-Shift-D** to execute and display whatever is returned, which is similar in behavior to GDB’s `print` command. Code to be executed can be any number of lines, so this view can help you quickly drill down into bugs, print out the values in complex classes, print out only erroneous values in a data structure, modify some data structures on the fly, etc. The possibilities are endless.

## Navigation Shortcuts

To show a list of all shortcuts, use **Ctrl-Shift-L**.

### Basic Navigation

Often, you will find yourself browsing your code to check up on stuff, and then, want to return to where you were editing. **Ctrl-Q** will do exactly that—take you back to the point of your last edit. Together with **Alt-Left** and **Alt-Right**, which will go back and forward through the recently viewed editor windows, these three shortcuts are especially powerful and time-saving when combined with the following shortcuts.

When your cursor is currently over a variable, method, or class, hit **F3** to immediately go to the definition of that item. If you are prefer using your mouse, then a **Ctrl-click** will accomplish the same thing.

You can skip directly to a line number by hitting **Ctrl-L**. If line numbers aren't turned on, you can do so in Preferences->General->Editors->Text Editors->Show Line Numbers.

When you have many editors open, you can open a searchable list of all of them by pressing **Ctrl-E**, and typing.

### Search

To find text within your current editor, you can hit **Ctrl-F** for a normal search box, or, just **Ctrl-J** for incremental search. When you first press Ctrl-J, it doesn't look like anything's happening, but just start typing, and Eclipse will highlight what you search for. Use up and down arrows to go to the next or previous results.

Eclipse also has an enormously powerful search tool, accessible by **Ctrl-H**. File search will search through all files, and Java search will search through only Java source files, in a code-aware manner with many options—searching only for calls to methods, declarations to a variable, etc. What's really useful about using Search instead of one of the Ctrl-F or Ctrl-J shortcuts is that once you complete a search, your editor window will highlight all search results on the editor window's margin, which makes finding search results very easy.

### File Navigation

To open a Java file, instead of looking through your packages, which can be annoying if you have many files, use **Ctrl-Shift-T** (Open Type), and type ahead to open the file. A very similar shortcut, but will list all files, and not just Java files, is **Ctrl-Shift-R** (Open Resource).

### Java-aware Inferencing and Navigation

One great thing about Eclipse is continuous compilation, and many tools that can do code-aware operations.

For example, if you want a quick peek at a class's type hierarchy, but don't want to get rid of the current class in the Hierarchy view, you can hit **Ctrl-T** to get a quick type hierarchy at the cursor of whatever class the cursor is over.

If you have a large Java file, it can be a hassle to find where you declared a certain method or instance variable, even with the Outline view. Hit **Ctrl-O** instead to pull up a quick outline, and start typing in the type-ahead box to quickly get to the desired part of the code. If you hit **Ctrl-O** twice, the quick outline will also show inherited members of the current class.

Very often, you will encounter a method, and wonder where that method is being called. One way you can solve this eternal mystery is from the Search tool, but, more simply, you can just highlight that method and hit **Ctrl-Alt-H**, which will bring up the Call Hierarchy view, which will show where that method is being called, and where those calling segments of code are being called, and so on up the chain. When using the Call Hierarchy, beware that it is unfortunately not smart enough to discover if an overridden method is actually being called. For example, let's say you have a class `Car` extends `Vehicle`, and `Vehicle` is a class with a `public void drive()` method that `Car` then overrides with its own `drive()` method. If you pull up the Call Hierarchy for `Car.drive()`, you would, unfortunately, not discover the segments of code calling `Vehicle.drive()`.

Finally, to navigate to the next/previous error or warning on the current editor window, use **Ctrl-./Ctrl-,**.

## Editing Shortcuts

### Syntax Shortcuts

The most useful editing shortcut is perhaps **Ctrl-Space**, which autocompletes in a code-aware manner. Once you start using this, you'll start finding it tedious to ever fully type out a variable name ever again.

Similar to **Ctrl-Space** is **Ctrl-I**, which suggests "Quick Fixes," especially useful when the cursor is over an error or a warning. It will suggest fixes such as implementing unimplemented inherited methods, changing a declaration to match something else, etc. It's like having a code caddy with you suggesting your next move.

Eclipse does very smart tabbing. If you tab over a row, it will automatically tab it to the correct indentation. If you highlight a block of code and press **tab**, the entire section will indent one tab, and pressing **Shift-tab** will un-indent one tab. To automatically indent multiple lines to the right place, highlight the code to be indented, and press **Ctrl-I**.

To comment out a line, press **Ctrl-/**. This works on a selection of multiple lines. To uncomment, press **Ctrl-/** again, or **Ctrl-\**.

To delete the current line, simply press **Ctrl-D**.

## Refactoring Shortcuts

To rename anything it is almost always the better solution to have your cursor over what's to be renamed, and press **Alt-Shift-R**. This will allow you to automatically change all references to that variable at the same time, in a code-aware manner. However, your code must be compiling for you to use this refactoring shortcut.

Like Alt-Shift-R, you can also completely change a method signature and all calls to it, along with default values for extra parameters, with **Alt-Shift-C**. Use this *very* carefully, as it changes every call to that method everywhere.

Never type another `import` statement again with the **Ctrl-Shift-O** shortcut, which organizes your imports, adding all imports necessary, and deleting unused ones. If there are conflicts, such as `java.util.List` and `java.awt.List`, Eclipse will ask which one you mean to import.

In class, we mentioned that the “right” way to have public class and instance variables is to actually make those variables private, and then create public getters and setters. Creating those boilerplate getters and setters is quite the hassle, but fortunately, Eclipse has the shortcut **Alt-Shift-S + R** that automatically generates the getters and setters.

When coding, it is definitely best to code with reasonably sized methods from the beginning by design. However, if you suddenly discover that your method is starting to rival *War and Peace* in length, you can extract some of that code into a separate method with the extract method shortcut, **Alt-Shift-M**. Similarly, you can extract out a local variable with **Alt-Shift-L**.

## Other

### Hot Swapping

Eclipse supports hot swapping of code as you debug as long as there aren't changes at the class level—no changes in class-level variables or methods. This means that you can hit a breakpoint, realize that something's wrong in the current method, then edit that method while you're still in debug mode, then save, and Eclipse will back up the currently running code to an appropriate point, and continue to run the code without having to restart.

A useful trick that exploits hot swapping is to just type a random space somewhere and save. This forces Eclipse to hot swap in the new code, then back up and restart the run from the beginning of the current method, even though nothing of substance has changed. Just note that if there are class-level variables, they will not be “reset” to their original state at the beginning of the method.

### Save Actions

Save actions are code cleanup actions you can set to automatically run whenever you save a file. These include adding `@Override` annotations, cleaning up the imports, and other nice things. You can set them in Preferences->Java->Editor->Save Actions.