Eclipse Cheat Sheet

(version 1.1)
November, 2015

©Pr. Olivier Gruber
Université de Grenoble

The idea is to group here all the major tricks and tips I use everyday when coding in Java with Eclipse. For your information, NetBeans has usually very identical features, so if you are a NetBeans user, search for them and improve your coding experience.

Installing Eclipse, see the web for that.

Start reading the basics of Eclipse on the Web:

Concept of Workspace.
Concept of Project.
Concept of Perspective and views
Concept of Eclipse plugins (Eclipse software updates)

And you have a good start...

Know that you can import existing Eclipse projects in a workspace, see also the web.

Global Preferences and setup:

Windows → Preferences → and you can setup a lot of interesting things.

For Java, you can setup pretty printing preferences,
I usually use 2 spaces instead of tabs of 4 spaces, source code is more compact.
Why use this? Essentially to help reading your code but also to avoid false-positive changes when using source control (SVN or GIT).

Install a Java JDK, not a JRE... you will have the sources for the Java class libraries.
Also, with the JDK comes JVisualVM (see corresponding section below).

Make sure that Eclipse knows about your installed JDK:

Windows → Preferences → Java → Installed JRE

Also setup the Java compliance level for your Java project,
so that it matches your installed Java JDK:

Windows → Preferences → Java → Compiler

Updates and installing new plugins:
Help → Check for updates
Help → Install new software

Project preferences and setup:

Right-click on the project → properties (or Alt-Enter)

Pay special attention to Java Build Path, source folders, output folder, and libraries.
Source folders are the folders where Eclipse will look for your classes (sources).
The output folder (usually bin) is where you will find your compiled Java classes (.class).
Typicaly command execution:

```
$ java -cp ./bin my.package.MyClass
```

Note: notice that you can have one project depends on others, so you can structure your code in multiple projects when it makes sense.

Most useful shortcuts:

Ctrl-F: search (and replace)
Ctrl-space: content assist
Shift-Ctrl-F: pretty-print (reformat your source).
Shift-Ctrl-O: automatic management of your imported classes/packages.

Ctrl-O: outline as a popup, quick navigation to Java items in the current source)
Ctrl-Shift-T: search for a class (wildcards do work)
Ctrl-Shift-G: select something, you will see where it is used
for example, select a method and you will see all the places it is called from.
Ctrl-E: select a tab.
Ctrl-Shift-W: close all opened tabs... cool once you have so many opened that you are lost.
Ctrl-B: rebuild all.
F3: select a Java item (class, variable, or method) and navigate to it.
F4: select a class and see its hierarchy (super-classes) and sub-classes
this is especially useful when wanting to know which classes implement an interface.

Select a class that implements an interface or extends a super class, and for which the compiler complains that some methods are not implemented...
Crtl-1 → Add unimplemented methods
Voilà !

Refactoring: read on the web about it.
Alt-Shift-R: enables the renaming of the currently selected item (class, method, variable)

Degug:
F5: step in
F6: step over
F7: step out
F8: go
F11: relaunch last launch

Most useful functionality:

Windows → Project → Build / Clean / etc..
Windows → Editor → Toggle Split Editor
will give you two editors on the same file, great to see different parts of the same file at once.

Right-click → Compare
You can select multiple classes or files in the navigator view and compare them.

Navigator view:
Top button, like a dual yellow arrow → synchronize/de-synchronize the navigator with the currently selected source.

Global search → toolbar → the icon with a torch lamp.
Debug Configurations:
the bug icon in the toolbar, click its down arrow, select debug configuration...
you can basically tweak everything about the way a Java application is launched.

Breakpoints:
Breakpoints can be set on lines, rather traditional.
But breakpoints can also be setup on exceptions.
See the breakpoint view, in the debugger perspective, and click on the “!” button.

Refactoring... please read about Eclipse refactoring in Java on the web.

Eclipse Java Debugger:

One of the main reasons for using Eclipse... the Java world is extremely well supported and in particular Java debugging. One would need hours to cover all the great useful productive stuff one can do with a debugger. Here some starters, be inquisitive, and you will find new stuff regularly.

First and foremost, ******USE****** the debugger. If you are not doing performance measurements, **you must always run your programs under the debugger**.

Right-click on a class that defines a main method, and you can launch it under debugger. Once you have launched it, you can do so again from the "bug" icon (the one icon that looks like a round bug with legs).

Note that if you click on the right-hand side of the bug icon, on the arrow, you get a pull-down menu that allows you to choose which debug configuration you want, but also you can go and edit the configurations.

Debug configurations are very important. You can have many, with many different names, allowing you to start the same Java program with different settings, such as different arguments for your application and for the underlying JRE.

Note: you can debug multiple programs at once with Eclipse. This is especially useful when debugging distributed code across multiple JREs.

A debugger will help you getting your code off the ground quickly. Just execute step by step and catch all the minors bugs and mistakes one does when writing some new code. You know, null pointers, almost-correct string manipulations, forgotten field assignments, etc.

Notice that you can run incomplete code in Java, so you can start writing and start debugging, even though all your code is not there. Nice to try out stuff quickly.

After single-stepping, the most important functionality is setting breakpoints. A breakpoint is a place where execution will stop. Just left-click on the right-hand-side of any source line and you will set up a breakpoint on that line. Under the Debug perspective, you can look at the breakpoint (open the breakpoint view: Window → Show View → Other → Debug → Breakpoints).

IMPORTANT: You can setup breakpoints on exceptions. From the breakpoint view, select the ! icon, enter the name of an exception class, and choose caught/uncaught, and that's it. For instance, **you should always have a breakpoint set for NullPointerException, caught and uncaught**.

In the call stack, you have monitor information also that are extremely useful when debugging multi-threaded applications. In particular, this always to see deadlocks and understand them. Don't forget to "pause" your threads (the || icon on the debugger bar).

Dynamic Software Update (DSU) is also a powerful tool... while you debug, you can find and understand a bug, fix it and recompile, the JRE, while still under the debugger, will dynamically update your loaded classes in your running JRE. They are limitations, but in many cases, you can just keep going with your debugging session.

Eclipse as builtin support for JUnit:

When developing, you are writing your code, then you are debugging it, usually running some basic tests. Most of us approach testing through ad-hoc small programs... They tend to get lost... and more importantly, as your software evolves, you are not running the same tests systematically... so correcting new bugs might re-introduce bugs in parts of your code that you tested already.

JUnit is useful for two main reasons. First, it gives you a simple framework to structure your tests. Second, it gives you the ability to run all your tests easily and get a report of what is running and what is not.

http://junit.sourceforge.net/doc/cookbook/cookbook.htm
http://help.eclipse.org/mars/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2FgettingStarted%2Fqs-junit.htm

Eclipse also as a plugin for coverage:

Now, you wrote a number of tests, you are running them regurlarly, but how do you know that you tested everything? This is where covereage comes in. Run your tests, with coverage, and you will see which lines are code have been tested and which ones have not been tested.

→ Project right-click → Coverage As → Java application or JUnit test
→ Get hightlighted sources and a report coverage.
→ In EclEmma report view, can delete the current session (with the cross button).

http://www.eclemma.org/
http://www.eclemma.org/installation.html

Note that Cobertura is also an interesting coverage framework:

http://sourceforge.net/projects/cobertura/?source=typ_redirect
http://cobertura.github.io/cobertura/

Miscellaneous:

Eclipse supports printing Java sources, with multiple source pages per printed page, that is cool.

Eclipse CDT (the C/C++ plugin) is nice.

If your Eclipse crashes and refuses to restart, complaining that your workspace is already locked... then you need to remove the lock file in the .metadata directory in your workspace:

```
$ cd workspace
$ rm -f .metadata/.lock
```

Last but not least:

Oracle JvisualVM is a tool that comes with the Oracle Java JDK. **It is an absolutely fantastic tool.**
Launch it like:

```
$ $(JDK_HOME)/bin/jvisualvm
```

It will launch a window, in which all running Java processes will show up. You can select any one of them and connect to it, seeing everything you wanted to know about the runtime characteristics of your Java program.

Use it and you will love it. Guaranteed!