Osvaldo Jiménez

Eclipse Starter

Thanks to Nick Parlante for much of this handout

Eclipse

- Open source project out of IBM, www.eclipse.org.
- Many, many features -- somewhat bewildering. Can compile C, Python etc., but most famous for use with Java. Eclipse itself is written in Java (that's why it works on Linux, on Macs, etc.).
- See the compiler notes and links on the course page for full Eclipse information. This handout is just a quick introduction to get things started.

Import Existing Project (HW)

- The homework starter files will be distributed as Eclipse project directories, ready to import.
- An Eclipse project is equal to a directory -- it contains the source files and also hidden .project files that contain the Eclipse project info. Therefore, you want to copy/move your whole project directory when moving it from one place to another.
- Suppose you have a project directory in the filesystem somewhere
  - Contains all the .java files, and Eclipse will put the .class files there too
  - Also contains .project and .classpath files that contain the Eclipse configuration
- Select "*File > Import*"
- Select "*Existing project*"
- Browse to project directory and select. Eclipse will write its files, changes etc. to that directory (i.e. that's the directory to take with you, not lose, etc.)
- On import the "*copy to workspace*" option will copy the project directory into Eclipse's workspace directory, and make changes to the workspace copy and not the original. I recommend you leave this option false, so Eclipses uses your project directory in place.
- To create a new project, use *File > New > Java Project*

Running

- At the left, select the class containing `main()`
- Right-click (on the Mac ctrl-click), and select *Run > Java Application*
- Or, select *Run...* to manage the "configurations" to run such as setting some command line arguments
- The green triangle "run again" button is very handy -- run whatever you ran before again. It also has a menu of your most recent runs (different mains(), unit tests...).

Debugging

- Double click in the editor at the left of a line to set a breakpoint on that line
- Use "*Debug*" instead of "*Run*" to run in the debugger (Note: "*Run*" is different from "*Debug*" -- run ignores the breakpoints).
- This should prompt to switch to the "debugger" perspective
  - There are little buttons at the far upper right to switch perspectives
  - Also the "*Open Perspective*" menu
- In the debugger, look at the stack and variables

- Eclipse has the typical Step-over, Step-in, Step-out controls to advance the program in little steps. You have to have a breakpoint somewhere so the debugger gets control.
- Use the red-square to kill the program.
- Click the "*J-Java*" in the upper right to return to the editing perspective (after killing the program typically)
- In debug mode, an exception can automatically drop into the debugger, which is a very easy way to look at the variables etc. at the time of the exception (this happens when you "*Debug*" the program, not when you "*Run*" it). In the debug perspective, the *Run > Add Java Exception Breakpoint* command allows you to edit which exceptions drop automatically into the debugger.
- If you are about to add some print statements on a line ... consider using the debugger to break on that line instead. It will allow you to click around and look at all the variables. On the other hand, if the line is hit many times and only fails on the 67th time ... well `println` is handy for showing a time series like that.

Removing / Quitting

- To remove the project from Eclipse, right-click the project and select "*Delete*". This should prompt with a "*do not delete contents*", which is what we want ("contents" in this case refers to the .java and other files out in the directory). This removes the project from Eclipse, but leaves its directory, files, etc. intact out in the filesystem.
- This detaches the project from Eclipse, and you can quit.

Other Great Eclipse Features

- Eclipse has **many** features, but here are a few of my favorites
- Type `foo.` -- it pops up "auto completions". Hit return to select, escape to get out. Also, typing ctrl-space brings this up on command. For example. type `Collections.` and hit ctrl-space for it to pop-up `static` methods in the `Collections` class.
- Completion also works with generic syntax in some places -- a place where you can see from context that a `<String>` or whatever is called for -- ctrl-space and Eclipse just puts it in.
- Hover over a word in the source to pull up its javadoc.
- Select some code, use *Source > Correct Indentation* to fix indentation
- Use *Source > Shift Left/Right* (tab, shift-tab also work) to bulk change the indent of something
- Select a method name and use *Navigate > Open Declaration* (F3), and it goes to the definition of that method. Click the little left yellow arrow at the top of the window to go back. In this way, you can kind of surf through your sources.
- Select a method/variable/class name and use the *Refactor > Rename* command -- this is a superficial feature, but is sure is handy to fix up a name choice quickly.
- Select a block of code inside a method, and use *Refactor > Extract Method*. It's smart about what parameters are required.
- Use the *Source > Toggle Comment* (Ctrl+/) command to block comment/uncomment code.
- Use Ctrl+Shift+o to *Organize* Imports. Eclipse will automatically search through your java file and `import` any classes that you have used so far but haven't written the `import` line for.