Data Structures and Algorithms

Lecture 1 — Eclipse

Alan P. Sexton

University of Birmingham

Spring 2019

Eclipse

- Eclipse is one of a number of Integrated Development Environments (IDEs)
  - Intellij IDEA
  - Netbeans
  - Visual Studio
- Includes:
  - Source code editor with contextual code completion
  - build automation tools
  - debugger
  - class/package browser
  - code generation facilities
  - refactoring tools
  - extension support via plugins
- Everything that you can do in Eclipse, you can also do outside Eclipse using standard Java command line tools, though often at a cost of reduced convenience, efficiency and speed of development and debugging

GIT

GIT is a distributed version control system.

- All modern software development uses such systems
- GIT is by far the leading tool in this area
- Nearly all employers in the software development world will require you to know GIT.
- We will study GIT later in the module and we will require you to use your own git repository for your work. Until then, please do not use your own GIT repositories until some critical configuration information has been explained
- Until then, the only way your code is effected is that you should include a `.gitignore` file in the root of your project
- After we cover GIT, all your projects should be in GIT

.gitignore

- A `.gitignore` file in the root of a repository identifies files which GIT should never track or record
- There is a sample `.gitignore` file for Eclipse/Java projects in the `video-store` and `dsa-2019` repositories
- Files excluded:
  - Eclipse metadata files (`.project`, `.metadata` etc.)
  - Temporary files (`*~`, `*.tmp` etc.)
  - Generated files (`*.class`, `/bin`, etc.)
  - Operating system specific files (`.DS_Store`, `Desktop.ini`, ...)
- Note that this means that Eclipse project settings are **NOT** stored in GIT:
  - IDE settings are personal and should not be part of sources stored in GIT
  - Locations of repositories, libraries etc. may need to be different on different machines
  - DIfferent developers may use different IDEs
  - Slight inconvenience in that all instances (Laptop, Home desktop, School Lab machines) need to have the project settings set independently.

Eclipse Workspace

When starting Eclipse, a workspace directory must be chosen

- Always use the same workspace
- Never set the project folder to the default location (i.e. within the workspace), as doing so would:
  - pollute your GIT repository with Eclipse metadata: thus imposing your IDE version and settings on other developers or messing up your local GIT working directory
  - cause problems when updating Eclipse to new versions (as Eclipse controls the workspace and can modify and rearrange the files there on upgrades)
  - cause problems if Eclipse messes up (e.g. hitting your quota limit or running out of disk space) and your workspace needs to be deleted and re-created
  - couple your project sources to Eclipse, causing problems if you ever switch to a different IDE

General Settings

There are a number of setup/configuration issues to take care of when using Eclipse. These need to be setup only once, not separately for each project:

General Settings

Make sure you have added the sources and your **LOCAL** JavaDoc location to your JRE (Java Runtime Environment) setup. This ensures that you can browse and step through the Java library code and that you can view the JavaDoc locally and efficiently:

- `Window|Preferences|Java|Installed JREs`, double click on your JDK, select `.../rt.jar` (or `.../jrt-fs.jar` in newer JDKs)
  - On the lab machines, this file is in `/usr/lib/jvm/java-1.8.0-openjdk/jre/lib/rt.jar`
- In `Source attachment`: ensure that `.../src.zip` from the JDK directory is set in the External Location Path field
  - Lab machines: `/usr/lib/jvm/java-1.8.0-openjdk/jre/src.zip`
- In `Javadoc location`: ensure that the location is set to the home directory of your JDK's JavaDoc: e.g. `file:<XXX>/docs/api/`, where `<XXX>` is the directory that your JavaDoc is installed in
  - not installed on the lab machines: if you do not install these files, then looking up the API will be slower, but still possible

More General Settings

- Choose to use your normal browser instead of the internal one for help and JavaDoc:
  - `Window|Preferences|General|Web Browser` and set the tick on "`use external web browser`"
- Tell your debugger to step over library code by default:
  - `Window|Preferences|Java|Debug|Step Filtering`, ensure ticks are set on "`Use Step Filters`" and on all the "`Defined Step Filters`"
- Turn off initial folding of source code lines (which hides imports and header comments by default):
  - `Window|Preferences|Java|Editor|Folding`, ensure ticks are cleared on all the "`Initially fold these elements`" tick boxes

Setting your Coding Style

Projects tend to have their own agreed coding style. In Eclipse, you can set a default coding style for all projects and overide the default with a project specific coding style.

- The `video_store` and `java_skeleton` projects have a coding style exported from Eclipse in the root of the repository called `EclipseJavaFormat_APS_1.xml`
- Load the style into Eclipse with:
  - `Window|Preferences|Java|Code Style|Formatter|Import...` and select the above code style file
- If you want different coding styles for different projects then right click on the project in the `Package Explorer` window (left side), select `Properties|Java Code Style|Formatter`, tick the "`Enable project specific settings`" and set the active profile for this project to the style you want

Adding the logging and unit testing libraries to Eclipse

This is the normal process for adding any user library to Eclipse.
For this module you should add the libraries:

- log4j-1.2.17
- junit-4.12
- hamcrest-all-1.3 (or, alternatively, hamcrest-core-1.3: a smaller library missing some of the more advanced features of the "all" version) (hamcrest is needed by junit-4.12)

Do not download these libraries for use on the lab machines. They are all already there and available in
`/bham/pd/packages/java/1.8.0/lib/`
To obtaining the libraries for your own machines, either copy them from the lab machines or:

- log4j-1.2.17 can be downloaded from
  `http://logging.apache.org/log4j/1.2/download.html`
- junit4.12 and hamcrest-core-1.3 can be downloaded from
  `https://junit.org/junit4/`

Adding a user library to Eclipse

I will take the example of the log4j library. First add the log4j library to Eclipse:

- `Windows|Preferences|Java|Build Path|User Libraries|New...` and enter `log4j` as the name (NOT as a System Library)
- Select `Add External Jars...` and locate the `log4j-1.2.17.jar` file
- Double click on `Source Attachment`, and select `External location` and select either the `src` directory of the log4j distribution or the log4j `src` jar file
- Similarly either set the `Javadoc` location to `<log4j distribution directory>/site/apidocs` or to the `Javadoc` jar file
- Different libraries will also have source and API docs but possibly named or located differently

Adding the libraries to an Eclipse project

- Now that the library is available as a user library, you should never need to add it to Eclipse again. You can simplay add it to any of your projects as follows:
  - Open the project properties: right-click on the project in the Package Explorer Window and select `properties`
  - Select:

    `Java Build Path|Libraries|Classpath|Add Library|User Library`

    and select the user library that you want from the list provided

  Alternatively, when you create a new java project, you can select the already installed user libraries when you create it (see later)

Setting up a Project

- First set up your project directory: copy the `video-store` project directory from the "`video-store`" repository to some location outside any GIT repository under a new name (e.g. `my-video-store`) and remove the `.git` sub-directory from your new `my-video-store`
  - the `.git` sub-directory contains all the internal information for a GIT repository, but your new project is not in a GIT repository, and certainly not in the original `video-store` one

Creating an Eclipse Project from a Directory

Now that you have a project directory with sources, you can set it up as an Eclipse project directory:

- `File|New|Project...|Java Project` or `File|New|Java Project`
- Enter a project name
- Untick "`Use default location`" and choose the root of your project working directory
- Set the "`Project layout`" to use separate folders for sources and class files
- Click on "`Next>`"
- At this point it should have identified "`src/main/java`" and "`src/test/java`" as source directories (they should have a tiny cross in a box icon attached to the folder icon) and the "`src`" directory as NOT a source folder.
  - If not, right click on the directories and select "`Use as source folder`" or "`Remove from build path`" as appropriate.

Creating an Eclipse Project from a Directory cntd...

- Click on the "`Libraries`" tab
- Click on "`Add Library`"
- Select "`User Library` and then "`Next>`"
- Select all three of your user libraries (hamcrest, junit4 and log4j) and Click on "`Finish`"
  - Note: there is a bug in some versions of Eclipse and this last step may fail to add the libraries. If this happens in your version:
  - Right click on your project name in the Package Explorer
  - Select "`Build Path|Add Libraries`"
  - Select "`User Library`" and click on "`Next>`"
  - Select all three libraries and click on "`Finish`"
- If it offers to open the "`Java Perspective`", select the "`Remember this choice`" and agree

At this point, the project is set up. Now we have to create a run configuration (to tell Eclipse how to run the code and with what parameters) and to compile and run it.

Setting a run configuration

- `Run|Run Configurations...`, or click on the down arrow to the right of the round green run button in the toolbar

If this was a Java application with a `main` method, we would create a new Java Application configuration, but here we want a JUnit configuration:

- Double click on JUnit
- Give a name to this new configuration: usually the project name with "Test" for a JUnit configuration
- Select "`Run all tests in the selected project...`"
- Set the "`Test runner` to "`JUnit4`"
- Click on "`Apply`" then on "`Close`"

Running the Tests

- Click on the down arrow to the right of the round green run button in the toolbar and select the run configuration name you chose. The tests should run.
- After this run, everytime you click on the green run button, or Control-F11, it will rerun the tests.
- The test results appear in the JUnit tab to the left of the screen
- Any console log messages appear in the Console tab at the bottom of the screen.

Working on a Project in Eclipse

- Make sure that all other projects are closed
  - Eclipse assumes that all open projects need to be compiled and must be error free to compile and run the project you are working on.
- Perspectives are configurations of windows suitable for specific tasks
  - Java: for developing Java code
  - Debug: for debugging
- Drag and drop windows to modify a perspective for a task at hand
- `Window|Perspective|...` for operations on Perspectives or hover over the top-most, rightmost, buttons in the button bar to find the Perspective icons
- Right click and reset on the perspective icon to return to default setting
- If you close a perspective, you can get it back with `Window|Perspective|Open Perspective`

Javadoc for the Project's JavaDoc

- `Project|Properties|Javadoc Location` and set the `Javadoc location path`, i.e. the directory in which the rendered javadoc documentations should be created, usually "`javadoc`". This creates a directory of that name in the root of your GIT repository working directory when you generate the javadoc
- Note that the `.gitignore` file with the module projects will cause the `javadoc` directory to be ignored by GIT.
- Generate the JavaDoc with `Project|Generate Javadoc...`
- Thereafter you can view your Javadoc in the browser by putting your cursor on an identifier and pressing `Shift F2`
- Note that entering "`/**`" and then pressing `Enter` will generate a basic Javadoc skeleton appropriate to where you enter it (e.g. before a method declaration with parameters it fills in the parameter names etc.)

Editing in Eclipse

Keyboard shortcuts here are for Linux. Check them all in `Windows|Preferences|General|Keys`, Also shown in menus beside commands.

- If most are missing from menu commands in Ubuntu, then in Ubuntu's System Settings: `Personal|Appearance|Behaviour` Set "`Show the menus for a window`" to "`in the window's title bar`" and add to /etc/environment the line:

  ```
  UBUNTU_MENUPROXY=0
  ```

Navigating and Source Browsing

- `F3`: Go to declaration
- `Alt` $\leftarrow$ or `Alt` $\rightarrow$: Navigate through the history of cursor positions
- `Control Q`: Go to last position where you changed something
- `Control L`: Go to line number

More Editing

Viewing Javadoc

- Put cursor on type name or identifier and type `Shift F2`

Code Formatting

- `Shift Control F`

Code Generation

- `Source|Generate...`
- `Source|Overide/Implement Methods...`

Code Completion

- After "." in Java code or "#" or "@" in JavaDoc code, autocompletion is activated.
  - It can be turned off if you don't want it: `Windows|Preferences|Java|Editor|Content Assist|Auto Activation`. Either disable auto activation or change the triggers
- Otherwise, can be invoked at anytime with `Control Space`
- Completion is context sensitive
- Completion proposals are separated into types (e.g. Java Tasks, SWT, ...). You can cycle through different types of proposals by typing `Control Space` multiple times.
- Template proposals allow entering of (potentially large and complex) code snippets

Template Proposals

If you type the first few characters of a template name and then `Control Space`, the template will be proposed and, if chosen, intelligent options to set identifiers will be given

- Try typing, in a class where a method would go, "`main`" and complete
- Try typing, in a method where there is a number of collections declared, "`for`" and complete. Select to iterate over a collection, then use tab to switch to different parameters to change.
- The full list of templates can be viewed, changed or you can add your own new ones at

  `Window|Preferences|Java|Editor|Templates`

Debugging in Eclipse

- Add breakpoints by double-clicking on the far left of the line
- Control the debugger with Resume (F8), Step into (F5), Step over (F6), Step return (F7)
- Explore variable values in the Variables window or by hovering over the variables in the source window
- Explore variables and locations on the call stack by clicking on the stack frame entry in the Debug window
- Add a *Watch* to keep track of values of variables, or **expressions**
  - e.g. for an arraylist `myList`, you can add a watch on `myList.size()`
- Highlight an expression, evaluate it and view the result by right clicking and choosing "`Inspect`"

Breakpoints

There are a number of useful options in the `Breakpoint` window:

- Set a breakpoint as a trigger point: all other breakpoints are suspended until the trigger breakpoint is hit, upon which the trigger breakpoint is disabled and the remaining breakpoints become active
- Set a hit count so that the breakpoint only triggers after a specific number of hits upon which the breakpoint is disabled
- Set a condition upon which the breakpoint triggers
- Using the `J!` button, add a Java exception breakpoint
- Note that some exceptions, when hit, disable themselves