Workshop: Tools

A very brief introduction to Eclipse (and more), ...if you need it, Eclipse got it, search....

Eclipse can be run from Windows, Mac, and Linux. Description here is somewhat Linux oriented (shouldn't matter). Download at http://www.eclipse.org/. Eclipse Classic or Eclipse IDE for Java Developer will do.

1 Projects

Warnig Eclipse by default creates a directory "workspace" in your home directory. Never store anything in workspace. If Eclipse gets into serious problems remove the directory and you will get a fresh installation next time you start Eclipse (Eclipse asks for workspace on every start up, uncheck to get rid of).

When using Eclipse we'll always create a project (working with single files in Eclipse seems impossible?). A project consists of;

- A project directory. The project directory will hold all files for the project (not just source code).

  TIP: Use suffix .ep for project directory, for example myproj.ep, will avoid future confusion.

- In the directory there are two hidden files .project and .classpath. If lost Eclipse will not recognize the project. Usually no problem, it's always possible to recreate the project if you have the sources [1].

- In the project directory there will also be (at least) two directories /src and /bin. Src (source) is where the *.java-files will reside. Bin (binaries) is for the compiled Java-files i.e. the *.class-files.

  – No problem to remove the bin directory, Eclipse will recreate it when compiling the sources.

  – Eclipse compiles automatically and constantly in background, so you don't need to...

2 Eclipse Concepts

Editor: Window where you can modify the content of your resources (often files), e.g. the text editor for Java code.

[1] There's possibly a hidden directory .settings and a ~/.eclipse directory.

View: Single window showing supportive information, often related to the content of the active editor, e.g. the Outline view selects important content from the currently open file and displays it in a tree structure.

Perspective: An arrangement of editors and views.

3 Create a Project

Lets start with the traditional hello world-program. Eclipse can be found in the menu (or open a terminal and type; eclipse &). When started, if not "Package Explorer tab" is visible, see picture below, do Window > Show view > Package Explorer.

1. Create the project. File > New...> Project...> (possible Java) > Select Java Project > Next;

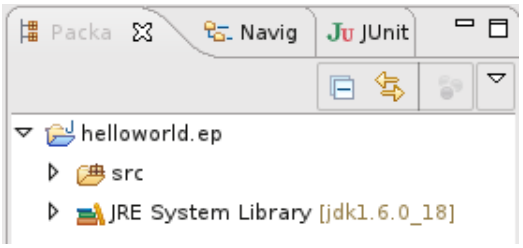
   a) Project name: helloworld.ep (this is the Eclipse internal project name)

   b) Uncheck "Use default location" and browse to project directory (i. e. ~/courses/tda550/workshop/helloworld.ep).

   NOTE: helloworld.ep will be the physical project directory on disk (possible to delete Eclipse project but keep directory, i.e. they are two distinct entities).

   Leave the rest. Next. Finish.

2. Should look like this in Package Explorer view (this year it should be jdk-1.7.0):



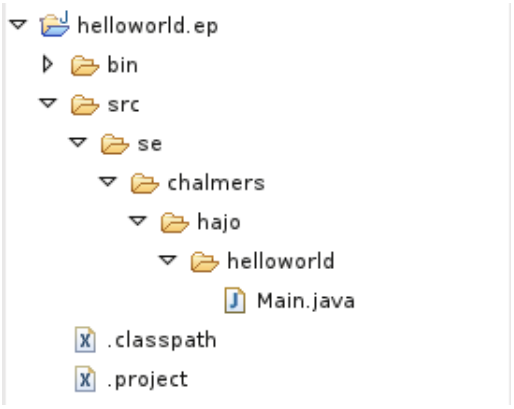
   NOTE: This is the normal view when coding (shows items relevant for coding). Use it (NOT Navigator view, see below).

3. Create the package hierarchy.

   a) Mark src in Packages tab, right click > New > Package, Name: se.chalmers.cid.helloworld (change cid to your login). Last part is the assumed name of the final application (helloworld).

   Customize view To customize the view of packages and files, select the View Menu (small upside down triangle, top right in picture above). Possible to show/or not empty packages etc.

4. Add a main-class. Mark package se.chalmers.cid.helloworld, right click > New > Class.
   a) Name: Main
   b) In "Which method stubs would you like to create:" Select public static void main, unselect others[2]. Finish.
5. Select Navigator tab (if not present Window > Show View > Navigator). The Navigator works as a file handler. Inspect directories and files! Important! Should look like (possible save to force a compilation to get the bin directory):



   If it looks different, mark project, right click > Delete, select "Also delete contents under ...." > Yes. Then start over again!

   NOTE: Very important that you really have this structure, else total confusion, ... can affect your mental health! Check again!!

6. Also ... check file system outside Eclipse using a terminal (use command "tree"), it should look like (if not, start over again);

```
helloworld.ep        // The project directory
    |-bin
    |-src
       |-se
          |- chalmers
              |- cid
                  |- helloworld
                        |- Main.java
```

7. Now do some coding, add System.out.println("Hello world") in main-method. Save!
8. Select Main class, right click > Run As > Java Application.

[2] This is a fairly standard convention, the main method always resides in a Main class.

9. Output should appear in Console tab (bottom) if not visible use Windows > ...
10. Inspect file system again, Navigator tab (look in bin dir).
11. On row before main-method add a "//TODO fix some things here", save. Watch left margin. Open Window > Show view > Tasks.

    TIP: Very good, make it a habit!!

TOP TIP: This is great for understanding! Mark standard Java class name (for example String) anywhere in code press F3, ... the source for the Java SDK class will show up! Note style!!

3.1 Run time configurations

To be able to pass arguments to the program we need a run time configuration. Continue on same project.

1. In Package Explorer mark project, right click > Run As > Run Configurations... (dialog shows up).
2. Fill in name: helloworld
3. Select Arguments tab. In "Program arguments " text field enter your name > Apply > Close.
4. Modify in Main class so that arg[0] in main-method is printed.
5. Look in the toolbar for the "green circle with triangle" (Ctrl + F11), click on drop down mark, "helloworld" should show up. Select and click.

3.2 Adding classes and interfaces

This has no real meaning it's just to shows some of Eclipses features (be Lazy, let Eclipse do the work...). Continue on same project.

1. Add interface named INet in se.chalmers.cid.helloworld package (Mark package > New > Interface >....
2. Add methods;

```
public void send( IMessage m );
public IMessage receive();
```

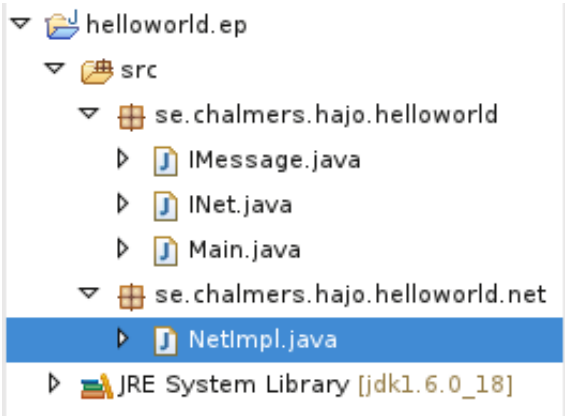
3. Two "light bulbs" will show up. Click on and accept suggestion to create an interface IMessage (in same package).
4. Add package se.chalmers.chl.cid.helloworld.net

5. Create a class NetImpl in net-package. Let class implement INet (do it directly in the code). Finish

   a) Import INet interface (new light bulb)

   b) Select Add unimplemented methods.

6. Save. Finally, should look like this;



4 Running outside Eclipse

Sometimes its convenient to run a program outside Eclipse.

1. Open a terminal and move to the project directory (and stay there).

2. Now try to run the project using the command line.

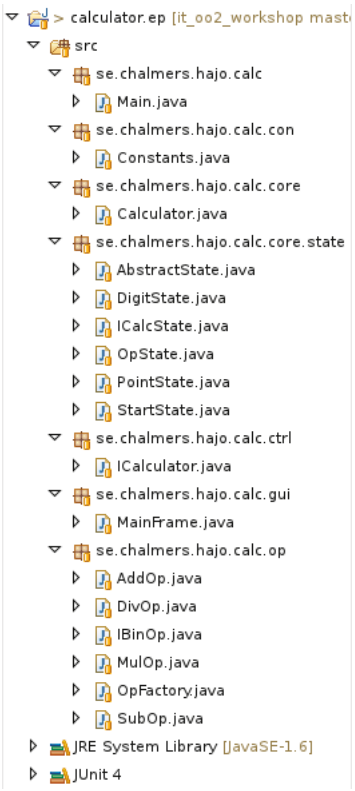
   TIP: You must tell the Java machine where to find you *.class-files, use the "-cp" (classpath) switch. You must supply the qualified name of class with Main method and finally the command line argument to the program.

5 Managing projects

This is just a file and package handling exercise. Download calculator.ep.zip from course page and import into Eclipse (File > Import > Existing Project ... > Next > Browse > ...).

5.1 Refactoring

All classes are in the same package (default package). Not good! Restructure to get the package structure below **but use your own cid instead of hajo** (create packages, move files to correct packages, etc.). This should be the final result (right click or use drag'n'drop).



6 Debugging

Eclipse reactions...

Syntax error Eclipse shows a red dot with a white cross. You'll get an immediate exception if try to run. Point on dot for tip what to do. Possible have to save to force a remove of the dot.

Language errors Light bulb with a very small red dot like above. Run behavior as above.

Warnings A light bulb with a exclamation mark. Program will compile and run but possible run time errors later. Point/click to get a list of suggestions what to do.

It's possible to customize the warnings. Window > Preferences > Java > Compiler > Errors/Warnings

Line numbers When working with errors it important to know the row number (it's in the error message). Eclipse shows row numbers in the bottom margin (like 22 : 6, row 22, column 6).

Problems view The Problems view lists all problems (incl. line number). Window > Show view > Problems

6.1 Get a program

Download debug.ep. zip from course page. Unzip and import into Eclipse.

6.2 Using Resume

Breakpoints To be able to stop the program and inspect values we need to set break-points in the code (at some statement). Set a break point by double clicking in the left margin before the selected statement. A blue dot shows up (double click again to remove).

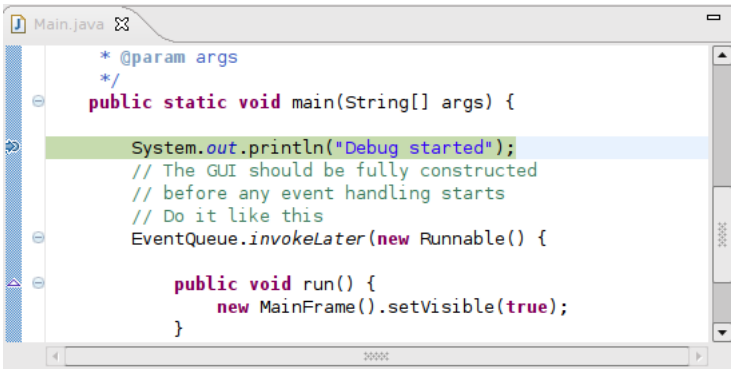
1. Do a short test run of the downloaded (meaningless) program.
2. Set a break point in the main method at;

   ```
   System.out.println("Debug started");
   ```

3. Select Main class > Debug As > Java Application

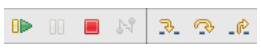
   Debug perspective Eclipse now switches to the debug perspective (= reorder and opens new windows). If a dialog shops up; Accept and click OK.

   Your code shows up in the editor window. Should look like;



   The program has halted at the break point. The statements to be executes is high lighted.

   The Debug toolbar Buttons from left to right (not counting disabled) , Resume, Stop, Step Into, Step Over, Step Return.



4. Have a quick look in the Debug tab. This shows all parallel activities (threads) that are running (you will learn about them later).

5. Click Resume. Program continues after the break point, the program shows up on screen. Inspect Debug tab again! How many activities are there?

6. Click Stop.

7. To go back to the Java perspective (the normal), click button Java far right/top in the main window margin.

6.3 Using Step over

1. We would like to verify that the buttons really works i.e. that a click ends up in some listener. Go to MainFrame and localize the listener for the Options > Silly button[3]. Put a break point before;

   ```
   jTxtFldInput.setText("");
   ```

2. Start debugging again.

3. We will hit the old break point. Click Resume (possible to inspect/remove all breakpoints in Debug perspective, Breakpoints tab).

4. Program should run. Input some text and click DoIt, click Other until you get a nice color.

5. Click Options > Silly to hit the second break point.

   Inspecting values Use the Variables tab to inspect variable values.

6. What's the value of the ActionEvents actionCommand?

7. Click "Step Over" to execute the statement. The program GUI want change (inspect Debug tab to see that the AWT-EventQueue-0 is suspended).

8. Click Step over again and them Resume (program runs again).

9. Now: In the same way, verify that all buttons are working.

6.4 Using Step into and Step Return

1. Remove all breakpoints (Breakpoints tab).

2. Put a breakpoint in MainFrame in ActionListener doit at;

   ```
   StringBuilder out = new StringBuilder();
   ```

3. Start debugging, add some input and click DoIt to hit the breakpoint.

4. Click Step over and inspect in.

[3] Code is ugly auto generated...

5. Now we would like to inspect the execute method of DoItCtrl class. Click Step into. You will now possible end up in Javas ClassLoader class. To get out click Step Return then click Step into again.

6. You now are in the constructor of DoItCtrl(). Step until you are back in MainFrame.

7. Click Step into. Now you should be in the execute method. Step through.

8. Do this over again but select Step into at

```
List<String> result = p.permutate(in);
```

9. For each round in for loop inspect String s and permStr (a list, inspect element-Data[n]). Any understanding??

10. If not.. you have at least done some debugging. Stop it.

6.5 The SortedBuffer class

This class has it's own main method , so possible to run (bad style, just for now). Select file > Right click > Run as ... There are bugs, or at leas one.... Fix them[4].

7 Try A Plugin

(Optional) Find the STAN Eclipse-plugin and try to install. Run STAN-analysis on some code... any news?

[4] Bugs are planted, not in original code.