

Getting Started with Java, Ant and Eclipse

1. Introduction

This handout will help you get started using Java and the Eclipse IDE. The tools we will use are cross-platform and free off the web.

Java - Object-oriented language and technology	http://java.sun.com/
Eclipse - an open extensible IDE	http://www.eclipse.org/
Ant - Java-based build tool	http://ant.apache.org/
JUnit - testing framework for Java unit tests	http://www.junit.org/index.htm

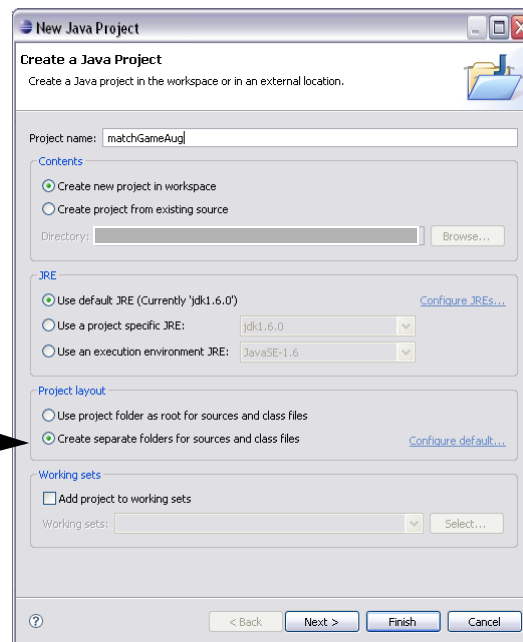
To illustrate these tools, consider the sample code, “MatchGame.java” that was used to check the installation of the Eclipse Java development environment on the TabletPCs. As software gets more complex and we need to link in additional libraries and perform other tasks, it is a standard practice to automate these tasks with a build script. We will use Ant to run such scripts. Ant also makes it much easier for team members to work together even when their machines and development environments are not set up identically.

2. Setting up a new Java project in Eclipse

1. Decide where you want to place your Java projects. Eclipse uses the idea of a workspace directory to store projects. You can make Eclipse start up using your preferred workspace.

2. Create a new Java project (File --> New --> Project --> Java Project). Let Eclipse make a project directory for you in your workspace (See Figure 1). Pick a name that will be easy for you to recognize (*match-GameAug*). Don't worry about the other options in this dialog since we will be using an Ant build script to control the compile and run procedures.

Figure 1: Eclipse 3.4, New Java Project dialog. Define project name and make sure that “Create separate folders for sources ...” is checked. Then hit “Finish”.

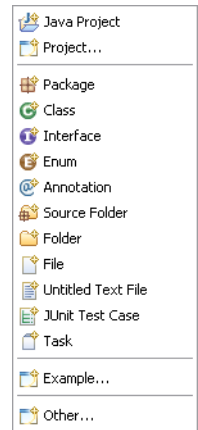


Within your project directory it is good practice to group different types of files into different directories. In particular it is generally a good idea to keep your source code separate from the compiled code. Java also uses a close correspondence between java packages and directories. It makes life much easier to set up these directories carefully early on so they work well with a build script.

A source folder called *src* (for source code files) should already be included in your project. Add a package (*cs435*) to the *src* folder (see Figure 2). Add a new java class to the package by adding a new *class* file. For instance, the class *MatchGame.java* was used to initially test the installation of Eclipse. IDEs like Eclipse provide options to run the code as an application but in practice, we discourage using this method. Instead an Ant build script will be used. Build scripts provide better machine independence and help to automate and document the build process.

Figure 2: Highlighting the *src* directory and popping up the right menu to get the New submenu is an easy way to create a new package within the *src* directory.

The same technique can be used to create a new class file within a package.



New submenu

3. Using Ant to compile and run a project in Eclipse

A build script to compile and run this code is given in Figure 3. This is not written in Java, it is XML specifying Ant instructions. Create it at the top level of your project directory (New --> File, create as a simple file) and name it *build.xml*. Copy and paste the xml text from Figure 3 into your *build.xml* file and save it.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- build.xml, for Ant to compile and run cs435F08 example, Wainer -->
<project basedir="." default="compile">
  <property name="build.dir" value="build" />
  <property name="src.dir" value="src" />
  <property name="dist.dir" value="dist" />

  <target description="Creates temporary directories" name="init">
    <mkdir dir="${build.dir}/classes" />
    <mkdir dir="${dist.dir}" />
  </target>

  <target depends="init" description="Removes temporary directories" name="clean">
    <delete dir="${build.dir}" />
    <delete dir="bin" failonerror="false" />
    <delete dir="${dist.dir}" failonerror="false" />
  </target>

  <target name="compile" depends="init">
    <javac debug="true" deprecation="true"
      destdir="${build.dir}/classes" srcdir="${src.dir}"
      verbose="true" />
  </target>
</project>
```


Figure 3: An Ant script, *build.xml*, which has targets to compile and run the *MatchGame* example.

```

<target name="run MatchGame" depends="compile" description="Run the Example">
  <java classname="cs435.MatchGame" failonerror="true" fork="true">
    <classpath>
      <pathelement location="\${build.dir}/classes" />
      <pathelement location="\${build.dir}/.." />
    </classpath>
  </java>
</target>
</project>

```

Figure 3 (continued): An Ant script, *build.xml*

After the file has been saved you should see it (with an ant icon  build.xml) in the package editor. Right menu it and open it with the Ant editor. If the code pasted over a little ugly, you might want to move the cursor to the editor, get the pop-up menu and select the “format” option to clean things up. The blocks tagged as targets represent tasks that the script can be called upon to do. Build scripts must be updated and maintained as a project evolves. Additional targets for archiving, testing and distribution may also be required.

Run the sample code by selecting the “run MatchGame” target in the Ant outline view (see Figure 4) and popping-up the right button menu. Select Run --> Ant Build (the first option without the ...). If you haven’t saved source files you’ll be given a chance to do so. The instructions in the build file will be executed by Ant and as the script executes, feedback messages may be sent to the Eclipse console. The build file contains dependency information so it knows, for instance, that before running, the code needs to be compiled. The MatchGame example will also open its application window on the screen.

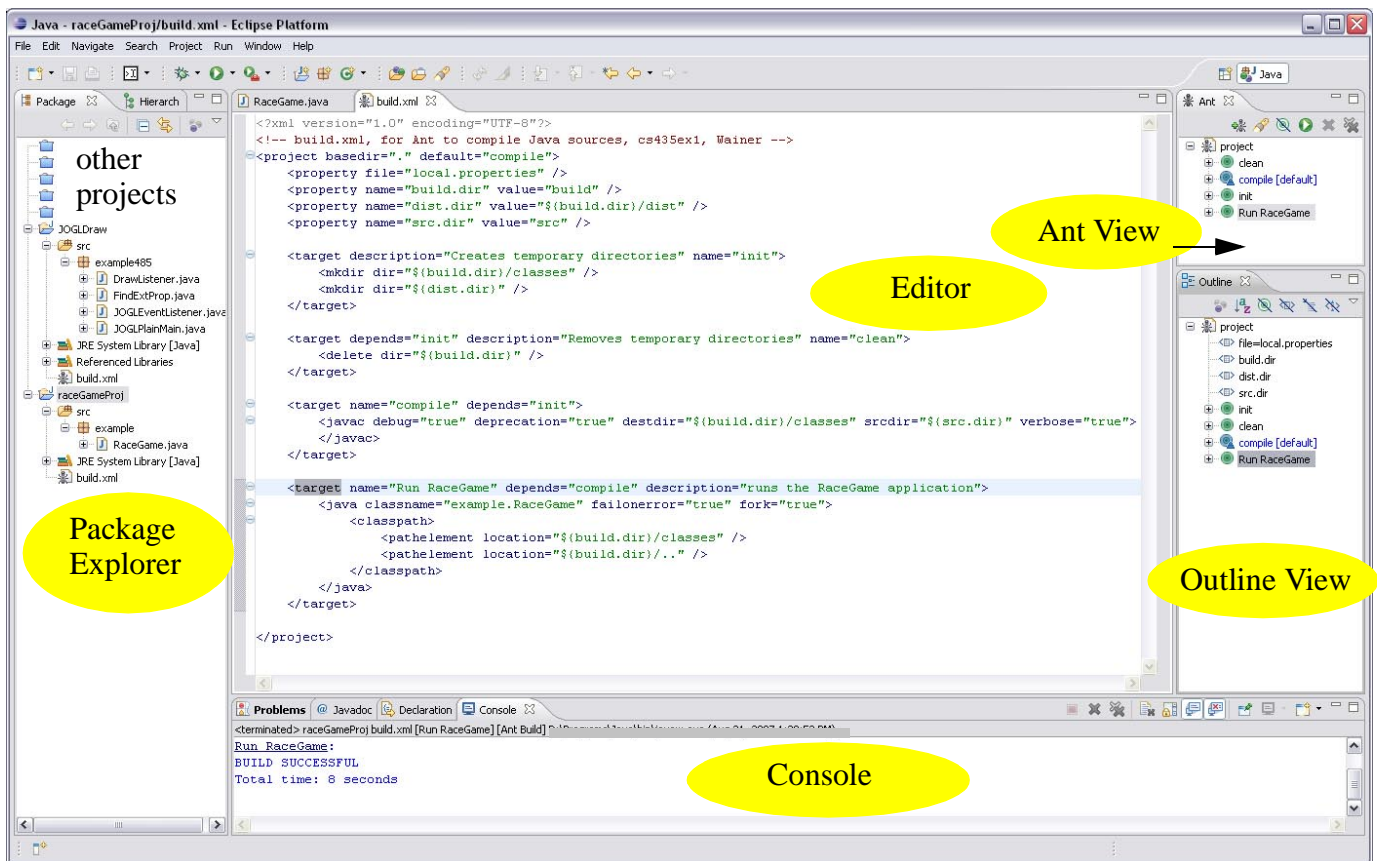


Figure 4: The Eclipse IDE showing interaction spaces useful when developing with Ant build scripts.

For more Ant goodness, try opening an “Ant view” under “Window --> Show View”. Under this view, double-clicking an Ant target will execute it.

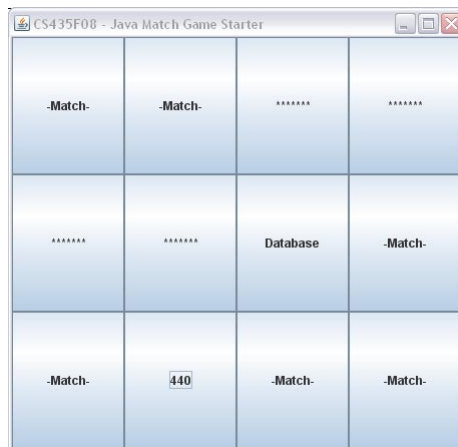
PROBLEMS/NOTES:

Ant Build can't find compiler (javac) - an error experienced when trying to compile code using an Ant script from within Eclipse sometimes occurs on Windows XP. Ant cannot find where the compiler is. A solution is to go to Eclipse's Window Menu. Select Preferences -> Ant -> Runtime. Pick the Classpath tab and access the Global Entries. From here, “Add External Jars”. You'll want to find and select the *tools.jar* underneath the Java jdk lib directory. That should solve the problem.

An error (Virtual Machine not found) may occur when trying to run Ant scripts. Go to the Run Menu, select External Tools and then the “External Tools Configurations ...” option. Highlight the build.xml file. Select the JRE tab. Mark the “JRE separate” option and select an installed jdk.

There may also be a mismatch between the compiler (tools.jar) and the compliance settings. Or a mismatch regarding the JRE system library for the project.

Screen dump of MatchGame after some user interaction (clicks on buttons)



Testing with JUnit will be considered shortly.