Yet Another Ant Tutorial

There are a lot of ant tutorials out there on the web. Some of my favorites are
http://www.exubero.com/ant/antintro-s5.html
http://ideoplex.com/focus/java#ant
http://en.wikibooks.org/wiki/Programming:Apache_Ant

You don't need to read all of them, just one is enough. After reading one, the only link you need to remember is this
http://ant.apache.org/manual/

So, what is this tutorial about if there are already good ones? This one is designed for this class. In this tutorial, I will work through the example of creating a fairly complete build.xml file for the scanner project. With that in mind, let's start.

0. Some notes

I recommend putting all source files into a directory called "src". You will also tar/gzip this directory and submit it. Please make sure all relevant source files and documentation are submitted.

1. Installing ant

Please see those links above for installing ant.

2. Why ant?

Makefile or a simple script can certainly do it, of course. But ant is so much more than that, especially for java applications. Since we're using Java in this class, it's a good opportunity to learn ant as well.

3. Running ant

Running ant is simple, you need a build file (similar to a makefile in the case of makefile). Because the language of ant is XML, naturally, ant will be looking for an XML file, namely build.xml.

Assuming the file build.xml exists. Running ant can be as simple as executing the command a*nt*

This will execute the default "target" defined in build.xml. If you want a more specific action:

ant build *(Execute the build section defined in build.xml)*
Or,
ant clean *(..)*
Or,
ant handin *(..)*

4. Ant build file

An ant build file is similar to a makefile, it has properties (variables), targets, and dependencies at the very least (and our tutorial shall cover those areas only)

4.1 A simple build file

Let's start with a simple build.xml which simply creates a directory called classes (where we will store our compiled .classs files)

```
<project name="A Simple build file" default="init">
      <description>
            A simple build file
      </description>

      <target name=”init”>
            <mkdir dir=”classes”/>
      </target>
</project>
```

The file is self-explanatory. The only important piece here is the property "default" of the project tag. It tells ant that if you don't specify any target in the ant command, the default "init" target will be assumed by ant.

4.2 Property

Now that we have the basic working file, let's make it a bit smarter. Say we have several places where we refer to the directory "classes", it would be wise to have a variable/property that can store this piece of information so that if we ever want to change it, we only need to do it at one place Changes to our original file are in blue.

```
<project name="A Simple build file" default="init">

            <property name=”destdir> value=”classes”/>

            <description>
                  A simple build file
            </description>

            <target name=”init”>
                  <mkdir dir=”${destdir}”/>
            </target>
</project>
```

4.3 Dependency

Dependency allows you to specify that before executing a target, ant must execute other target and its dependencies.

Let's introduce a target named build and it's dependent on our target init. For now, our build target doesn't really do anything other than output a message

```
<project name="A Simple build file" default="build">

            <property name=”destdir> value=”classes”/>

            <description>
                  A simple build file
            </description>

            <target name=”init”>
                  <mkdir dir=”${destdir}”/>
            </target>

            <target name=”build” depends=”init”>
                  <echo>Build target</echo>
            </target>

</project>
```

4.4 Ant Core Tasks

Tasks define what the target will do. You actually were introduced to a simple task in section 4.1: `<mkdir>`. Ant has a lot of pre-defined core tasks. In addition, it allows user defined task, but that is outside the scope of this tutorial.

Several core tasks are of great importance to our purposes: `<javac>, <java>, <mkdir>, <delete>, <exec>`…

Knowing the tasks alone isn't enough, you also need to know the associated properties of the tasks. Thankfully, the manual exists for a reason, and you only need to know several key properties.

<javac>

To build java files, you will need the `<javac>` task and at the very least, these properties

`srcdir:`	Specify where the java compiler will look for .java files
`destdir:`	Specify where the java compiler will place compiled .class files
`classpath:`	Self-explanatory

There are other properties, but for now, we can get by with these. For a more complete listing, see the ant manual

A simple build/compile target will look like this where the property `src` and `dest` specify where to look for source files and where to store the compiled class files, respectively.

```
<target name=”build” depends=”init”>
            <!-- Compiling source files -->
            <javac srcdir="${src}" destdir="${dest}"/>
</target>
```

<java>

To execute a java program, you will need the `<java>` task and at the very least, these properties

`classname:` Specify the name of the class where java finds the main method
`arg:` Specify arguments being supplied to the java app (optional)
`maxmemory:` Specify maximum heap memory (optional)
`jar:` Specify the jar file to run (if you want to run a jar file instead of a class)
`classpath:` Self-explanatory
`fork:` Specify whether the java app will be executed in a different JVM (if false, ant will use the jvm that was used to run ant to run the app)

A typical run target will look like this where the property main-class specifies the main class to execute

```
<target name="run" depends="build">
            <!-- Executing ${main-class} -->
            <java fork="true" classname="${main-class}">
                  <arg value=”arg1”/>
                  <arg value=”arg2”/>
            </java>
</target>
```

Other tasks
Now that we have covered our most important tasks (javac and java), let’s take a quick look at other tasks using some target samples

To tar/gzip files (for submission), use the tar and gzip tasks as follow (again, for a complete listing of properties associated with tasks, see the manual)

```
<target name="prepare" depends="build">
            <!-- Preparing for submission -->
            <tar destfile="${main-class}.tar" basedir="${src}"/>
            <gzip zipfile="${main-class}.tar.gz" src="${main-class}.tar"/>
</target>
```

To delete files (cleaning up), use the delete task as follow

```
<target name="clean">
      <!-- Cleaning up temporary files -->
      <delete dir="classes" quiet="true"/>
      <delete file="${main-class}.tar.gz" quiet="true"/>
      <delete file="${main-class}.tar" quiet="true"/>
</target>
```

To execute the handin utility, note that handin is not a ant core task, so we need to use exec to invoke the OS to do this for us

```
<target name="handin" depends="prepare">
      <!-- Submitting ${main-class}.tar.gz into ${submitdir} -->
      <exec executable="handin">
            <arg value="cs5470"/>
            <arg value="${submitdir}"/>
            <arg value="${main-class}.tar.gz"/>
      </exec>
</target>
```

5. Putting it all together

Now that we have gone through the key steps of constructing a build.xml file, let's put together a build.xml for the scanner project. `ant` or `ant build` will build the project. `ant handin` will tar/gzip the source files together and submit them. `ant clean` will clean up the temp files. ***The build file is still missing a target for generating java files from .jj file. This is left as an exercise for you.***

```
<project name="Scanner" default="build">
      <description>
            A build file for scanner
      </description>

      <property name="src" value="src"/>
      <property name="dest" value="classes"/>
      <property name="submitdir" value="scanner"/>
      <property name="main-class" value="scanner"/>

      <target name="init" depends="clean">
            <mkdir dir="${dest}"/>
      </target>

      <target name="build" depends="init">
            <!-- Compiling source files -->
            <javac srcdir="${src}" destdir="${dest}"/>
      </target>

      <target name="run" depends="build">
            <!-- Executing ${main-class} -->
            <java fork="true" classpath="${dest}" classname="${main-class}"/>
      </target>

      <target name="prepare" depends="build">
            <!-- Preparing for submission -->
            <tar destfile="${main-class}.tar" basedir="${src}"/>
            <gzip zipfile="${main-class}.tar.gz" src="${main-class}.tar"/>
      </target>

      <target name="handin" depends="prepare">
            <!-- Submitting ${main-class}.tar.gz into ${submitdir} -->
            <exec executable="handin">
                  <arg value="cs5470"/>
                  <arg value="${submitdir}"/>
                  <arg value="${main-class}.tar.gz"/>
            </exec>
      </target>

      <target name="clean">
            <!-- Cleaning up temporary files -->
            <delete dir="classes" quiet="true"/>
            <delete file="${main-class}.tar.gz" quiet="true"/>
            <delete file="${main-class}.tar" quiet="true"/>
      </target>
</project>
```