

**Massachusetts Institute of Technology**  
**Department of Electrical Engineering and Computer Science**

6.035, Fall 2005

Handout 5 — Athena, Tools

Wednesday, September 7

---

This document describes what you will need to know about Athena and Java tools for 6.035.

### **Athena clusters**

If you don't have an account on Athena, you should register for one immediately. Information can be found at <http://web.mit.edu/olh>

You can work in any of the public Athena clusters. Type `cvview` to see a list of clusters and available machines.

### **Communication**

We'll make course announcements via electronic mail. If you don't receive a message welcoming you to the class mailing list within a week, tell a TA immediately.

We'll answer questions via email. You can mail your TA directly or use `6.035-staff@mit.edu` to reach the entire staff.

### **Finding course files**

Handouts will be available on the course web site. Project directories, examples, and 6.035 programs are stored in `/mit/6.035`. The Java compiler, library, debugger, and associated programs and documentation are stored in the Java locker `/mit/java`.

You'll probably want to add these lines to your `.environment` file:

```
add -f java_v1.5.0
add 6.035
```

These commands attach the lockers and update your execution path to include the course software. You should also set your `CLASSPATH` environment variable to point to the jar files in:

```
/mit/6.035/provided/jars/
```

If you need to use different versions of java for other classes you can use the `-ver` switch.

```
java -ver 1.1.6 .....
javac -ver 1.1.6 .....
etc.
```

Note that the `-ver` must come before any other command line arguments.

## Working with Groups

Each group will be given a group locker that can be used to work on the project. There should be enough space in these lockers that you won't have any problems. Only group members and 6.035 staff will be able to access each group locker.

## Java Compiler

This year we'll be using Sun's JDK 1.5.0. It is available on the Sun and Linux Athena platforms. You can get a free version of the JDK from Sun's web site for other platforms (Windows, Mac). However, the only officially supported platforms for this class are Sun and Linux, so we may not be able to help you if you run into problems with other platforms. Since Java is platform independent, you can compile your final bytecodes on any platform.

Below we describe basic operation of JDK 1.5.0. For detailed information on JDK and Java API 1.5.0, consult <http://java.sun.com/>.

## Running JDK 1.5.0

Compile the source file(s) using the Java compiler. Use the `-g` flag to create debuggable bytecodes.

```
% javac dummy1.java dummy2.java
```

If compilation succeeds, the compiler creates a `.class` file (named `ClassName.class`) for each public class defined. If compilation fails, the compiler lists compilation errors.

You must have no more than one public class defined in each of the source files, and the filename must be *exactly* the same as the class name. You can define several private classes in one file, however the compiler will still generate a separate `.class` file for each class.

In 6.035 we'll be writing Java applications (not applets). Each Java application must have a public class that contains a `public static void main(String[] args)` method. To run the program simply type

```
% java MyProgram arg1 arg2 arg3
```

where `MyProgram` is the name of the class with the `main` method, and `arg1`, `arg2`, `arg3` are the command line arguments. These arguments are passed to the program in the `args` parameter to the `main` method, and can be accessed as `args[0]`, `args[1]`, etc.

## Java Debugger

If you use the `-g` flag when you compile your source code, you will be able to debug your program using the JDK debugger. Start the debugger using:

```
% java -debug MyClass
```

Once inside the debugger, `help` will list all available commands. Here is a very limited list of the most useful:

- `run <class> [args]` — Start execution of a loaded Java class
- `print <id> [id(s)]` — Print object or field
- `stop in <class id>.<method>` — Set a breakpoint in a `<method>`
- `cont` — Continue execution to the next break point.
- `locals` — Print all local variables in current stack frame
- `help` — Displays the list of recognized commands with descriptions

## Build Tools

You are required to use a build tool such as `make` or `ant` to build your projects, and to provide a `Makefile` (or the equivalent) the TA can use to create bytecodes from your source files. We assume general knowledge of using `Makefiles` from 6.170, but we've included some information here to refresh your memory.

The `make` utility automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them. Here's an example of a `makefile` you might use to build a standalone scanner.

```
JFLAGS = -g
CLASSPATH = /mit/6.035/provided/jars/JLex.jar

Main.class: Main.java Ylex.java
    javac $(JFLAGS) $^

Ylex.java: scanner.lex
    java -classpath $(CLASSPATH) JLex.Main $<
    mv $<.java $@

clean:
    $(RM) *.class Ylex.java
```

To use this `makefile`, we save it as `Makefile` in the directory with our sources. Running the `make` command recompiles any of the files that need it. This is confused a bit because the java compiler itself does this as well, but for 6.035 the above sort of `makefile` is entirely sufficient. You can do lots more with `make`. For more information, see the Info page (`C-h i` in `emacs`) or go to: <http://www.gnu.org/software/make/manual/make.html>

## CVS

You may want to use CVS (concurrent versions system) on your projects. CVS is a version control system designed to help you manage a source code base. You'll be writing a lot of code for 6.035, and for many of you this will be the first time you'll be working on a project of sufficient complexity to require source control.

There is an online manual available at <http://www.gnu.org/software/cvs/manual/>, as well as lots of information on the info page (again, `C-h i` in emacs). You'll need to understand the concepts of creating a repository (`cvs init`), checking out a working copy of your source tree (`cvs co`), checking in changes (`cvs commit`), getting diffs and logs (`cvs diff` and `cvs log`), and updating your working directory (`cvs update`). You may also find tagging and branching useful.

CVS is in the GNU locker on athena: `add gnu`