Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.035, Fall 2000 | Handout 5 — Athena, Tools | Wednesday, September 6

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

Athena clusters

If you do not have an account on Athena, you should register for one immediately. See the information in *Welcome to Athena* on registering for an Athena account.

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

Communication

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

We will also be answering questions via email. Email addresses for the staff are given on the general information handout. Mail to `6.035-staff` will reach all the TAs.

There is an informal zephyr instance, `6.035`, that can be used for instant communication. Generally most questions will be answered by students, but TAs may subscribe to the instance and answer questions if they are on-line. To be considerate, limit your use of this instance to substantive questions and answers. If you wish to flame the TAs, yell obnoxiously, demonstrate your limerick-writing talents, or come up with Java puns, please use `6.035.d`.

To subscribe to instance `6.035` type:

```
zctl add message 6.035 \*
```

To unsubscribe, type:

```
zctl del message 6.035 \*
```

To send a message that will be received by everyone subscribing to instance `6.035`, type `zwrite -i 6.035`.

Finding course files

Handouts, project directories, 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_v1.1.6`.

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

```
add -f java_v1.1.6
add 6.035
setenv CLASSPATH .:/mit/6.035/classes/
```

The first two `add` commands will both attach the lockers and update your execution path to include the course software. The last `setenv` line will update the `CLASSPATH` environment variable, which is needed for the Java compiler to be able to access the Java libraries.

If you already have a `CLASSPATH` setting, you'll want to adjust it to get the 6.035 libraries. In particular, you'll need to add `/mit/6.035/classes/` to your `CLASSPATH`, along with whatever directories your own class files are in. If you're unfamiliar with how class files work in Java, talk to your TA.

Working with Groups

Each group will be given a group locker which 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 the 6.035 staff will be able to access each group locker.

Java Compiler

This year we will be using Sun's JDK 1.1.6. It should be available on all Athena platforms (Sun, SGI, Linux, NetBSD). Sun workstations in Athena clusters have java installed locally. On other platforms you will need to add the java 1.1.6 locker (`java_v1.1.6`). In addition, you can get a free version of the JDK from Sun's web site for other platforms (Windows, Mac). However, the only officially supported platform for this class are the Suns, so the TAs may not be able to help you if you run into problems with any of the other platforms. Since Java is platform independent, you can compile your final bytecodes on any platfrom, and you can switch platforms during development.

Here, we describe basic operation of JDK 1.1.6. For detailed information on JDK and Java API 1.1.6, consult `http://java.sun.com/`.

Running JDK 1.1.6

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 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 will 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 into 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 all 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, the `help` command lists all the available commands. Here is a very limited list of the most useful ones:

- `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

Make

You are required to use `make` to build your projects, and provide a `Makefile` your TA can use to create bytecodes from your source files.

We assume general knowledge of using `Makefile`s 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 we use to compile the IR for our compiler.

```
JAVAC   = javac
JAVAI   = java
CLASSES = ../../../classes
JHOME   = $(JAVA_HOME)

CLASSPATH = /mit/6.035/classes:$(CLASSES)

all : code

code:
        $(JAVAC) -d $(CLASSES) -classpath $(CLASSPATH) *.java

clean:
        rm -rf *.class
        rm -rf ../../../classes/java6035/tools/IR/*.class
```

To use this makefile, we save it as `Makefile` in the directory with our sources. Running the `make` command will now recompile any of the java 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, see the Info page (`C-h i` in emacs) for more information.

CVS

You may want to use CVS (concurrect 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 that you'll be working on a project of sufficient complexity to require source control. Without it, you'll have a tough time.

CVS has an excellent tutorial available online at
`http://www.loria.fr/~molli/cvs/doc/cvs_toc.html`, as well as lots of information on its 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.