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

1 Athena

If you don't have an account on Athena, you should register for one immediately. Information can be found at http://ist.mit.edu/services/athena

You can work in any of the public Athena clusters. Type `cview` from an Athena machine to see a list of clusters and available machines. You may also log into Athena remotely over `ssh`; we recommend using the `athena.dialup.mit.edu` machines. We do not recommend using `linerva.mit.edu`.

2 Communication

Your primary source of information about 6.035 is the course website, accessible at http://6.035.scripts.mit.edu/. Announcements, the schedule, lecture slides, handouts, and assignments will all be posted to the website. You will submit your projects via the course website as well. We'll also make important 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 email the TAs at `6.035-tas@mit.edu`, or use `6.035-staff@mit.edu` to reach the entire staff.

3 Finding course files

Handouts will be available on the course web site. Project skeleton code, test cases, and individual and group repositories are hosted on the course Github page at http://github.com/6035. Git is stored in the athena locker `/mit/git`. The compiler and tools for the languages you may use are stored in their respective athena lockers: `/mit/java`, `/mit/scala`, and `/mit/ghc`, for Java, Scala, and Haskell, respectively. The course locker also, `/mit/6.035`, contains some files that will make it easier to use the Ant build system with Scala.

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

```
add -f (ghc, scala, or java)
add -f git
```

If you're using Scala, you'll also want to add the following lines to your `/.bashrc` file (or other initialization file for your shell):

```
export SCALA_HOME='/mit/scala/scala/scala-2.10.0'
```

These commands attach the lockers and update your execution path to include the course software.

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

```
java -ver 1.6.0_23  .....
javac -ver 1.6.0_23 .....
etc.
```

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

4   Working with Groups

The first project should be done individually. When the second project is assigned, the class will be partitioned into groups of 3 or 4 students.

For the first project, you will be given a private Github repository. This repository's name is based on your Github username and can be seen on http://github.com/6035; if you do not have a free Github account, you need to sign up for one. The TAs will collect Github usernames during the first week of classes and create these repositories for you; if you still do not have a Github repository by the project information session, let the TAs know and tell them your Github username.

After the first project, you must form groups. If you do not have a group, you may e-mail the TAs a message that they will post on the course website for other people looking for groups. Messages must be written in the style of dating personals ads. Once groups are formed, TAs will create Github repositories for each group.

5   Compilers and Tools

5.1   Java

We'll be using Sun's JDK 1.6.0 for Linux. 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 is 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 1.6.0. For detailed information on JDK and the Java API 1.6.0, consult http://java.sun.com/.

5.1.1   Running JDK 1.6.0

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.

5.1.2   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

5.2   Scala

Scala runs on the Java Virtual Machine and can use the Java API as well. We'll be using Scala version 2.10 on Linux. Scala 2.10 is available for any platform supported by the JVM, and may be downloaded from http://www.scala-lang.org/; however, we will only officially be supporting Scala on Linux, and may not be able to help you if you run into problems with other platforms. Scala is compiled in a manner similar to Java with the `scalac` command, taking `.scala` files. The resulting class files can be run with the `scala` command. For more information on Scala, including a comprehensive API reference and several texts, please refer to http://www.scala-lang.org/. The link "A Brief Scala Tutorial" under Documentation-¿Manuals will help you get started compiling Scala, and the book "Scala By Example" in the same location is a fantastic way to quickly get comfortable with the language.

5.3 Haskell

For Haskell, we will be using the Glasgow Haskell Compiler (ghc) and the Haskell Platform, version 7.4.0. In order to obtain the correct version of ghc on athena, **you must run `add -f ghc` and not just `add ghc`**. As with the other languages, the only officially supported platform for the course is Linux; we may not be able to help you on other platforms.

If you are using Haskell, you may not use any modules under and including `Data.Graph`, `Control.Lens.Plated`, or `Compiler`. Doing so may result in a reduced grade and angry TAs. You also may not use Parsec.

6 Build tools

6.1 Ant

If you are using Java or Scala, you are required to use `Apache Ant` to build your projects, and to provide a `build.xml` the TA can use to create bytecodes from your source files. Ant is a java-based build tool that strives to be platform independent. Ant is much like the familiar `Make` tools in that it resolves the dependencies necessary to perform a task. However, Ant is not shell-based. Project configuration files are written in hierarchical xml and Ant is extended by implementing java classes. Please visit http://ant.apache.org/ for more information on Ant. The handout describing the Scanner and Parser phase of the project will include a more detailed description of the build system, and the skeleton code includes an Ant build file to get you started.

6.2 Cabal

If you are using Haskell, you will use `Cabal` to build your projects. A starter file is included in the Haskell skeleton code. More information about Cabal is available on the internet.

7 Revision Control

You *must* use revision control on your projects to help manage changes to your 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. We require `git` for revision control, and we will be using Github to host our repositories.

More information on git can be found here:

http://git-scm.com/documentation

A guide to getting started on Github can be found here:

https://help.github.com/articles/set-up-git

Git can be found in its locker on athena: `add -f git`