

6.005 Software Construction

Problem Set 0: Getting Started

Before you start coding

- **Install the software you'll need on your laptop.** You need to install three things for this problem set:
 - **JDK 6** (for Windows or Linux; Mac users probably already have Java, if not, get it [here for 10.5](#). If you have a PowerPC or early Intel Mac or are running OS X 10.4, come talk to us). From this web page, download "JDK 6 Update 27"; you don't need NetBeans or Java EE.
 - **Eclipse 3.7.** Choose "Eclipse IDE for Java Developers", which will download a ZIP file. Unpack the ZIP file, go inside the resulting folder, and run Eclipse.
 - **Subclipse.** To install Subclipse, use Eclipse's Software Updates mechanism. The installation instructions on the Subclipse website are out of date, so follow [these](#) instructions instead.

Athena* already has this software installed, so if you'll be using an Athena, you don't need to install anything.

Eclipse

The Eclipse integrated development environment (IDE) is a powerful, flexible, complicated, and occasionally frustrating set of tools for writing, modifying, and debugging programs. It is especially useful for working in Java.

On Athena, Eclipse is in the `eclipse-sdk` locker; the command name is `eclipse`.

When you run Eclipse, you will be prompted for a "workspace," which is where Eclipse will store all the different projects you work on. On Athena, for example, the default location is a directory called `workspace` in your home directory. In addition to code, Eclipse stores its own "metadata" in hidden folders in the workspace. **You should not run more than one copy of Eclipse at the same time** with the same workspace, or the metadata will become corrupted.

The first time you run Eclipse, it will show you a welcome screen. Click the button to go

directly to the "workbench" and you're ready to begin.

Tab policy

At this time you should also go through and change tabs to spaces. Go to Window (Eclipse if you're running a Mac) → Preferences... →. In the toolbar go to Java → Code Style → Formatter. Click the "Edit..." button next to the active profile. In the new window you should change the Tab policy to " Spaces only " Keep the Indentation size and Tab size at 4. Enter a new name for the policy and press ok.

Subversion

To turn in this problem set, you will commit your code to Subversion. The last committed code before the problem set due date will be used for grading.

O'Reilly publishes [Version Control with Subversion](#), conveniently available online. This book describes what [Subversion](#) (or SVN) is and how to use it:

Subversion is a free/open-source version control system. That is, Subversion manages files and directories, and the changes made to them, over time. This allows you to recover older versions of your data, or examine the history of how your data changed. In this regard, many people think of a version control system as a sort of "time machine".

Subversion can operate across networks, which allows it to be used by people on different computers. At some level, the ability for various people to modify and manage the same set of data from their respective locations fosters collaboration. Progress can occur more quickly without a single conduit through which all modifications must occur. And because the work is versioned, you need not fear that quality is the trade-off for losing that conduit —if some incorrect change is made to the data, just undo that change. [[What is Subversion?](#)]

If you have used other version control software, such as CVS, many of the concepts and procedures of SVN will be familiar to you. If not, there are two important ideas to learn: *repositories* and *working copies*.

Repositories

Subversion is a centralized system for sharing information. At its core is a repository, which is a central store of data. The repository stores information in the form of a filesystem tree—a typical hierarchy of files and directories. Any number of clients connect to the repository, and then read or write to these files. By writing data, a client makes the information available to others; by reading data, the client receives information from others.

[...] What makes the Subversion repository special is that it remembers every change ever written to it: every change to every file, and even changes to the directory tree itself, such as the addition, deletion, and rearrangement of files and directories.

When a client reads data from the repository, it normally sees only the latest version of the filesystem tree. But the client also has the ability to view previous states of the filesystem. For example, a client can ask historical questions like, "What did this directory contain last Wednesday?" or "Who was the last person to change this file, and what changes did he make?" [[The Repository](#)]

Every SVN repository has a URL at which it can be accessed.

Working Copies

In order to make changes to files in the repository, you must obtain (or "check out") a copy of the current version of those files:

[... E]ach user's client contacts the project repository and creates a personal working copy—a local reflection of the repository's files and directories. Users then work simultaneously and independently, modifying their private copies. Finally, the private copies are merged together into a new, final version. [[Versioning Models](#)]

Any number of people can have any number of working copies (or "checkouts") of different parts of a single SVN repository. Those working copies might be on different machines and have different versions of files.

SVN in 6.005

In 6.005 you will be using a Subversion repository. The repository is located at <https://svn.csail.mit.edu/6.005/fa11>. You can also access it from any web browser and see the latest version of any file in the repository—great for quickly bringing up an example you're looking

for. This repository contains access-controlled folders for every assignment and example code presented in lecture.

- The "**published**" folder will contain example code presented in lecture. This folder can be accessed by anyone in the class under **/published/**.
- You will have your own **personal directory**, accessible only to you, for use in the problem sets. These are found in the **/users/your_username/** directory.
- You will have a **group directory** for each project you work on during the course, found under **/groups/project_name/student-student2-student3/**.

Setup

To setup your SVN folders and password, visit <https://courses.csail.mit.edu/6.005/svnadmin/>. You must have [certificates](#) installed. You only need certificates to access this administration page, however. Any other resource is accessed with your Athena username and your new SVN password. First, set your SVN password on the SVN admin page. Next, create your personal directory by clicking on the appropriate button and pull in the `ps0` assignment. That's it!

SVN in Eclipse

Eclipse has built-in support for working with version control systems, but does not include specific support for SVN. [Subclipse](#), a plug-in for Eclipse, provides this support, allowing you to check out files from a repository, check them in, and use all the other features SVN provides. You can read the [Subclipse documentation](#), also available inside the Eclipse help viewer, to learn more about the features of Subclipse.

On Windows and OS X, there is one extra step you need to take to configure Subclipse. Go to the SVN preferences page ("Window → Preferences... → Team → SVN") and under "SVN interface," choose "SVNKit (Pure Java)." The "JavaHL ([JNI](#))" option will only work if you have a command line Subversion client installed in your `$PATH`.

Connecting to Repositories

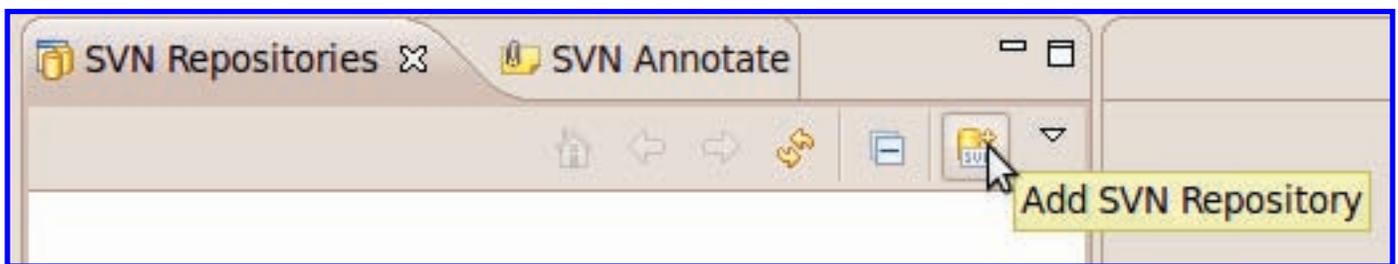
Working with Subclipse begins in the "[SVN Repository Exploring](#)" [perspective](#) (a perspective in Eclipse is a particular set and arrangement of interface panels). After you install Subclipse and restart Eclipse, switch to this perspective by following their instructions.

Follow the Subclipse documentation for [Creating a New Repository Location](#), with the URL:

The url for the repository is: <https://svn.csail.mit.edu/6.005/fa11>.

Note that to enter the "SVN Repository View" you should click the "Open Perspective" button in the upper right corner of the Eclipse window (it is a small window icon with a plus) and select the "Other..." option. You'll be presented with a dialog listing a number of perspectives, including "SVN Repository Exploring", the view you want.

In order to add the repository, you should click on the Add SVN Repository button shown below.



Courtesy of The Eclipse Foundation. Used with permission.

Checking Out Projects

Checking out a project from the repository once you've added it is straightforward, if that project was originally created in Eclipse. Right now, we'll check out the code for today's problem set. You may be asking: where did this code come from, if it's in my personal directory!? And even if you weren't asking, you should still know that the SVN admin site added the necessary code for you.

Navigate to the `ps0` folder in your directory, right click it, and select "Checkout..." (the documentation on this is under [Checking out a Project](#)). You should accept the name of the project unchanged, for this and all problem sets and all group projects.

If Eclipse does not prompt you, switch to the Java perspective with the menu in the top right of the workbench window.

✓ **Self-Checkpoint.** You should now be in the Java perspective with the `ps0` project checked out and ready to work on.

Automated Unit Testing with JUnit

JUnit is a widely-adopted Java unit testing library, and we will use it heavily in 6.005. A major component of the 6.005 design philosophy is to decompose problems into minimal, orthogonal units, which can be assembled into the larger modules that form the finished program. One benefit of this approach is that each unit can be tested thoroughly, independently of others, so that faults can be quickly isolated and corrected, as code is rewritten and modules are configured. Unit testing is the technique of writing tests for the smallest testable pieces of functionality, to allow for the flexible and organic evolution of complex, correct systems.

By writing thoughtful unit tests, it is possible to verify the correctness of one's code, and to be confident that the resulting programs behave as expected. In 6.005, we will use JUnit version 4.

The Anatomy of JUnit

JUnit unit tests are written method by method. There is nothing special a class has to do to be used by JUnit; it only need contain methods that JUnit knows to call, which will be referred to as test methods for the remainder of the problem set. Test methods are specified entirely through *annotations*, which may be thought of as keywords (more specifically, they are a type of [metadata](#)), that can be attached to individual methods and classes. Though they do not themselves change the meaning of a Java program, at run-time other Java code can detect the annotations of methods and classes, and make decisions accordingly. The Java annotation system, judiciously used, can create dynamic and powerful code. Though we will not deeply explore annotations in 6.005, you will see how other libraries, such as JUnit, make effective use of them.

Look closely at `RulesOf6005Test.java`, and note the `@Test` that precede method definitions. These are examples of annotations. The JUnit library uses these particular annotations to determine which methods to call when running unit tests. The `@Test` annotation denotes a test method; there can be any number in a single class. Even if one test method fails, the others will be run. The test methods contain calls to `assertEquals`, which is an assertion that compares two objects against each other and fails if they are not equal, `assertTrue` which checks if the condition is true, and `assertFalse` which checks if the condition is false. [Here is a list of the other assertions supported by JUnit](#). If an assertion in a test method fails, that test method returns immediately, and JUnit records a failure for that test.

Running Existing Tests

To run the tests in `RulesOf6005Test`, simply right click on the `RulesOf6005Test.java` file in either your Package Explorer, Project View, or Navigator View, and mouse-over the 'Run As' option. Click on the 'JUnit Test' option, and you should see the JUnit view appear, with a green bar indicating that all test methods ran successfully.

To see what a test failure looks like, try running `RulesOf6005Test.java` before making any changes to `RulesOf6005`. You should now see a red bar in the JUnit view, and if you click on `testHasFeature`, you will see a stack trace in the bottom box, which provides a brief explanation of what went wrong. In this case, `RulesOf6005` threw a Runtime Exception. Double clicking on lines in the Failure Trace will bring up the code for the test that failed. You should now implement the `hasFeature` method and then rerun the `RulesOf6005Test` tests. If you implemented the function correctly, you will see that a checkmark next to that test. Passing the public JUnit tests does not necessarily mean that your code is perfect. You should still look over the function specifications carefully and feel free to write your own JUnit tests to verify your code.

For a more thorough introduction, O'Reilly has a [JUnit and Eclipse tutorial](#), with screen-shots to help you get acquainted with using JUnit from within Eclipse. The guide was written for JUnit 3, so the code samples use the older (but still supported) JUnit API.

Checking In Changes to Subversion

You should always remember to commit your changes to the repository at the end of the problem set. In fact, it's not a bad idea to commit after every exercise, as soon as you're happy with the code you've written. When you're working on a team, you'll need to commit your code regularly—until you commit, no one else sees the changes you've made. It is, however, not a good idea to commit code that prevents the program from compiling or that makes it less stable. This is called "breaking the build" as it is a major developer *faux pas*.

You should also remember to commit your projects by the deadline. Where other courses ask you to "hand in" your assignments, in 6.005 you will submit all your assignments by checking them into Subversion.

Before checking in changes, it is always good practice to review what you've done. Right-click

on the `ps0` folder that holds your work for this problem set, and select "Team → Synchronize with Repository." Eclipse will prompt you to open yet a new perspective that shows how your local working copy compares with the repository (see [Team Synchronizing](#) for more explanation). Grey right-facing arrows indicate files you've modified that need checking in.

Right-click the file you changed and select "Open in Compare Editor" to see a detailed differences, or "diff," view that shows exactly what has been changed. Supposing you're satisfied that this is a change you want to check in, right click the project and select "Commit..." if you're in the Team Synchronizing perspective, or "Team → Commit..." from any other. Read the [documentation for the Commit dialog box](#), where you should:

- Write a message that describes your commit, and
- Make sure the set of files to be checked in is what you want—for example, don't commit Java class files, just source files.

Commit your `hasFeature` implementation right now. If all goes well, your changes should be committed to the repository. If you have any doubt about anything you've done up until now, or don't feel confident that your updated code was committed correctly, ask the LAs or staff. Otherwise—on with Java!

 **Self-Checkpoint.** You should now have `hasFeature` implemented and committed.

Implement `computeGrade`

Run the JUnit tests and commit your code.

Implement `extendDeadline`

You should read the api for [Calendar](#) and [GregorianCalendar](#). Run the JUnit tests and commit your code.

Run the `main()`

You can run the code in Eclipse by going to Run → Run As... → Java Application, and look for the output in the Console pane at the bottom pane of the screen. The Java Application is going to run the `main` method.

Final commit

This is the end of the problem set. Commit your solutions to your personal Subversion directory, as described above under Checking In Changes to Subversion.

MIT OpenCourseWare
<http://ocw.mit.edu>

6.005 Elements of Software Construction
Fall 2011

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.