

**11-791 Design and Engineering of Intelligent
Information System &
11-693 Software Methods for Biotechnology
Homework 0**

*Software installation, configuration, and your first Maven
project hosted on GitHub*

Important dates

- **Hand out: August 27.**
- **Turn in: September 8.** A bash script will automatically collect your submissions and execute your latest release (e.g., hw0-ID-0.0.1.jar) on our maven repository. The output from your jar will be compared with our expected output. Simply copying the code from this guideline should work!

2

Useful information

1. We suggest you to start Homework 0 as early as possible.
2. Even though we provide instructions for installing tools on Linux, Windows, and Mac, we strongly recommend to use Linux. We will try our best to help you solve whatever problems you might have, but we also encourage you, the experts in any of these platforms, to work with us to answer the questions from your classmates. We also strongly encourage you to use Google to find solutions for potential (installation) problems.
3. We expect that most of the general communication between the instructor team and students will take place on Piazza <https://piazza.com/class/hyvsubeilei6dd>. For private questions, e.g., regarding grades, you may contact instructors by e-mail. Your friendly TAs are: Avner Maiberg (amaiberg@andrew.cmu.edu), Parag Argawal (paraga@andrew.cmu.edu), Leonid (Leo) Boytsov (srchvrs@cmu.edu), or Xuezi (Manfred) Zhang (xueziz@andrew.cmu.edu),
4. Both source files and derived pdf file of this tutorial are publicly available (yes, *open source*!) on GitHub
<http://github.com/amaiberg/software-engineering-preliminary>
which means if you find any errors or easier ways to achieve the same goal, please feel free to fork the project and send a pull request back to me. Once your request is confirmed, a newer version will come out immediately to help other students! Or you can just report an issue at
<http://github.com/amaiberg/software-engineering-preliminary/issues>
5. We have compiled a list of some commonly occurring problems, which other students encountered. These are listed in Section 3.6. Please, study this list, especially if you ran into issues yourself. *We would also encourage you to use Google to find solutions to your problems.*

Task 1

Learning Git & Maven Basics and Creating Accounts

In this task, you are required to create your own GitHub (<http://www.github.com>) account if you haven't got one yet, and we will guide you through the process to create a Git repository on GitHub for this homework. We will also give you some basic concepts of Git and Maven, which are two important tools for the course and also for future software development.

Task 1.1 Knowing what Git is and what GitHub is

If you've already got familiar with Git, then you could skip this task.

If you have been working with CVS (Concurrent Versions System, not the CVS store), SVN (Subversion), or Mercurial, then Git is yet another revision control tool, which has some unique features. If you don't have any experience in working with a revision control system, then probably you might want to first learn what a *revision control* system is.

*Revision control, also known as version control and source control (and an aspect of software configuration management), is the management of changes to documents, computer programs, large web sites, and other collections of information.*¹

In general, revision control is super useful in software engineering. Intuitively, let us think about two use cases of revision control:

1. After days of development, you realized that you reached a dead-end and wanted to revert back to a very early version, but you couldn't remember what those changes you made during those days are, even though you might have your projects backed up and marked by dates. A revision control system could help you manage the changes you made each time you submit your change.

¹http://en.wikipedia.org/wiki/Revision_control

TASK 1. LEARNING GIT & MAVEN BASICS AND CREATING ACCOUNTS 2

2. Your friend admitted to work with you on your project, and you two are writing different parts of the project at the same time. Once you two decided to merge your parts, you found that you both had modified the same file, and you had to manually (or with some extra tool) check the difference between your modifications. A revision control system could automatically merge the changes you made.

From Wikipedia, Git² is described as *a distributed revision control and source code management (SCM) system with an emphasis on speed*. Besides the consideration of speed, you can find other features, e.g., *distributed development, non-linear development*, etc., at <http://git-scm.com/about>. Note that the entry page just introduces the first of many features, and you need to click on the other circles to view the other features. We recommend you to read at least the feature description page and the Wikipedia page to know those basic concepts of Git, and you might also want to further read the Pro Git book³ to know it better.

In addition, you should have an idea of the following concepts in Git, which are the most important ones for our course.

commit, index, branch

and you should also be familiar with the following commands:

`add, ignore, commit, fetch, pull, push, branch`

In this homework, we will show you how to run Git commands with EGit (the Git plug-in for Eclipse), and especially demonstrate the usages of `commit` and `push`. You are going to use more of them in your next few homeworks.

Finally, GitHub is a web-based hosting service for software development projects that use the Git revision control system.⁴ What you might want to add to your Bookmarks (or Favorates, or other social bookmark site) is the help page (<http://help.github.com/>).

Task 1.2 Creating your account on GitHub

In this task, you are required to register an account at GitHub and create a public repository.

Q1 Is there any difference between a public and a private repository?

A1 Every piece of code in a public repository can be seen by everyone from everywhere in the world even if he/she doesn't have a GitHub account. Codes in private repositories can be seen by the owner and the collaborators only.

²<http://git-scm.com>

³<http://git-scm.com/book>

⁴<https://en.wikipedia.org/wiki/GitHub>

TASK 1. LEARNING GIT & MAVEN BASICS AND CREATING ACCOUNTS 3

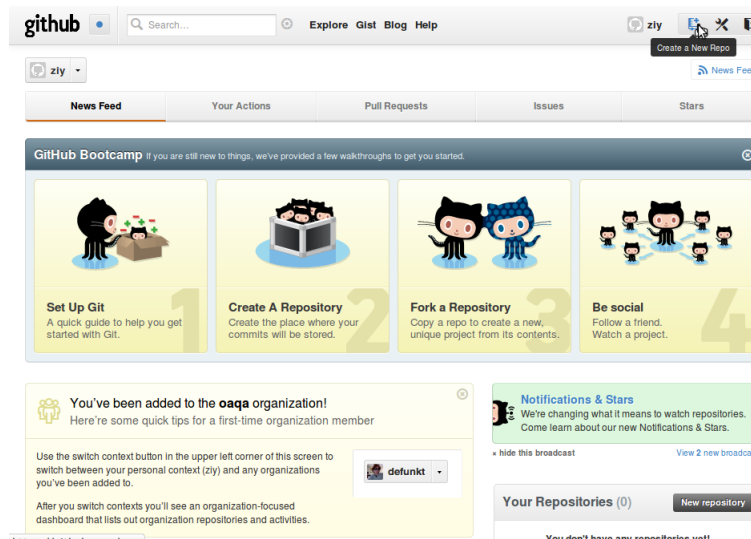


Figure 1.1: Registering a GitHub account

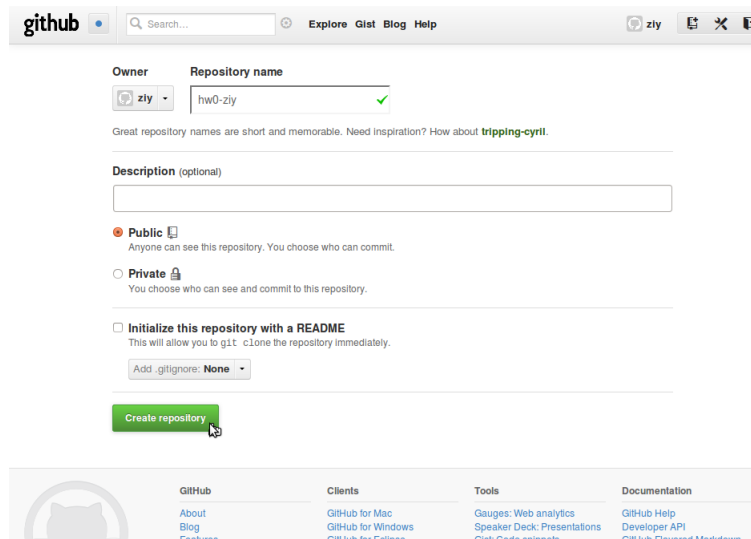
Q2 You asked us to create a public repository for homework codebase. But I feel it uncomfortable that other students might “plagiarize” my code and use for their homework submission.

A2 For Homework 0, you will soon realize all the codes will have been available in this guideline. If you prefer to fork other’s project to make your project, then you must already be an expert. For the rest of the homeworks, we believe if someone “plagiarizes” your codebase, know what is going on in your project, implement his/her code to work best with your code, and eventually achieve the best performance, then why not to let others look at your code?

1. Registration is easy, but you will need to generate an SSH key and add that to your GitHub account. This is a platform specific task, and we suggest you to refer to the help page at <http://help.github.com/articles/generating-ssh-keys>. Note that if you are using a Windows machine, you will need a Git Bash to run the commands, and it would be better not using a passphrase when generating the public key (instead directly pressing **Enter**), or the project will stuck at release because Eclipse is waiting for a passphrase but it doesn’t prompt us to do that.⁵ Another important step is to ensure you have verified your email address on GitHub, otherwise there might be risk that when Maven is executing “git push” command, it will also hang forever.

⁵Reported by Hector Liu.

TASK 1. LEARNING GIT & MAVEN BASICS AND CREATING ACCOUNTS 4



The screenshot shows the GitHub 'Create repository' page. At the top, there's a search bar and navigation links: 'Explore', 'Gist', 'Blog', and 'Help'. The main form has two sections: 'Owner' and 'Repository name'. The 'Owner' dropdown is set to 'zly'. The 'Repository name' text box contains 'hw0-zly' with a green checkmark. Below this, a hint says: 'Great repository names are short and memorable. Need inspiration? How about tripping-cyrl.'. The 'Description (optional)' section has a text area. The 'Public/Private' section has two radio buttons: 'Public' (selected) and 'Private'. The 'Initialize this repository with a README' checkbox is unchecked. Below it, a dropdown for 'Add .gitignore: None' is visible. A green 'Create repository' button is at the bottom. The footer contains the GitHub logo and links for 'About', 'Blog', 'Clients', 'Tools', and 'Documentation'.

Figure 1.2: Creating a public repository

Q1 What is SSH? What is a public key? Why do I need that?

A1 Informally, SSH allows you to securely connect to a remote server. Still informally, a public key allows you to make a connection without typing your password over and over again, but the server can still identify the person who tries to log in isn't anybody else but you! Because a Maven plug-in will execute Git commands without bothering you to type your passwords, GitHub needs to know your public key. If you are really interested in public-key cryptography, you need to find a textbook to read, which is far far beyond the scope of our course.

2. Creating a repository is also easy, on the top-right corner of the homepage, you will find an icon to create a repository. See Figure 1.1.
3. Type in a name for the repository. For example as shown in Figure 1.2, we suggest to name it as `hw0-ANDREWID`. Note, however, that the repository name can be different from your Java project name and Maven artifact name (which will become the name of your submission).
4. In the end, you will be notified about your repository URI. Copy it, which is useful in the next step.

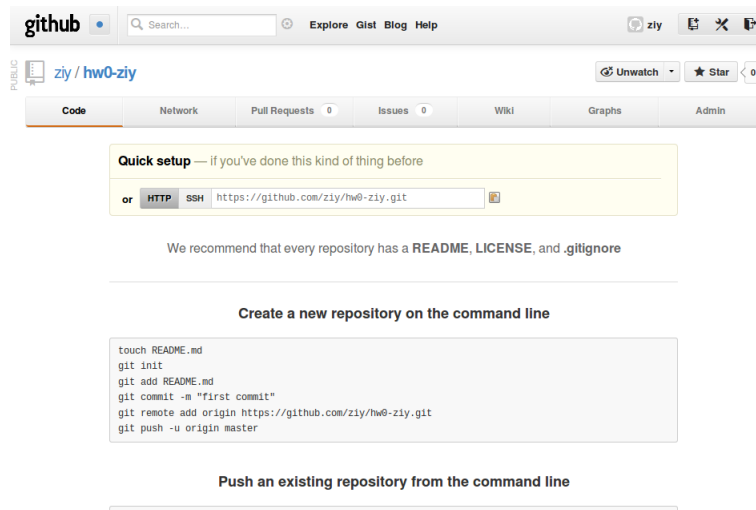


Figure 1.3: Copying your repository URI

Task 1.3 Knowing what Maven is

If you are a Maven expert, you could skip this task.

Maven⁶ is a build automation tool typically used for Java projects⁷, but can also work for many other programming languages. If you have experience in programming in C/C++, or have built C/C++ projects of someone else, then you must be familiar with Make⁸, or if you are a Java programmer, then Ant⁹ might be one of your everyday tools. They all belong to build automation tool.

If you are familiar with none of those, then you should learn what build automation tool is. From Wikipedia, build automation is defined as:

Build automation is the act of scripting or automating a wide variety of tasks that software developers do in their day-to-day activities including things like: compiling computer source code into binary code, packaging binary code, running tests, deployment to production systems, creating documentation and/or release notes.

For example, you want to clean your project before you rebuild it, and then you want to add all the dependencies into the build path, plus you also want to include other non-jar dependencies (e.g., configuration files) during building the jar by first copying them into the build directory. In addition, you want a tool to automatically run a set of tests to make sure it is still consistent with previous builds. What you need

⁶<http://maven.apache.org/>

⁷http://en.wikipedia.org/wiki/Apache_Maven

⁸[http://en.wikipedia.org/wiki/Make_\(software\)](http://en.wikipedia.org/wiki/Make_(software))

⁹<http://ant.apache.org/>

is a build automation tool, and a configuration file specifying a set of tasks for a build, like Maven, but Maven is more than that. (There are many articles to compare different build automation tools, especially between Maven and Ant, which you might be interested to read.) To better understand Maven, you should first read the Maven about page at <http://maven.apache.org/what-is-maven.html>, and then take a look at the feature page at <http://maven.apache.org/maven-features.html>.

Most important concepts in Maven for this course are:

pom (Project Object Model), plugin, goal, artifact, repository, central repository, project repository, local repository, archetype

Informally, *POM* can be considered as the configuration file for those tasks during building the jar, *plugin* is the tool to help you achieve each of your tasks, and *goals* are the set of predefined tasks you might need from many best practices in software engineering. *Artifacts* can be thought as the output (simplified as jars) when the build is done, and *repositories* are the place where those jars are storing, just like a warehouse of jars. *Central repository* is a global “supreme” repository (but of course distributed all over the world with many mirror sites). Formal definitions for these concepts can be found in the documentation of Maven at <http://maven.apache.org/guides/index.html>. We highly recommend you to go through all the topics in the documentation, especially the article entitled “Getting Started in 30 Minutes”.

Task 1.4 Being notified about your account on our Maven repository

We will collect your submissions for Homework 0 and all the following homeworks on our Maven repository.

Your user name will be your Andrew ID, and your initial password will be “11791” (without quotes). To change your password, you need to go to <http://mu.lti.cs.cmu.edu:8081/nexus/index.html>, click **Log In** at the top-right corner. After you’ve been logged in, you should click **Security** on the left console, then **Change Password** to reset your password. (The passwords are stored in encrypted texts.)

Now, you’ve done with your first task. You will need the URI of your project, and some guidances from the help page in Git and Maven to install these softwares in the next step.

Task 2

Installing and Configuring Software

In this task, you are required to install the tools you've already been familiar with (at least heard about).

Important notes: This might be the hardest part of your Homework 0, since installation of the same software on different platforms differs a lot, which means we could not show you detailed steps for each platform. Instead, we will provide you with links for the installation instructions for you to follow, and we have created three forums for you to discuss Windows/Linux/Mac platform related issues. We will try our best to help you solve whatever problems you might have, but we also encourage you, the experts in any of these platforms, to work with us to answer the questions from your classmates. You can find the discussion boards at <https://piazza.com/cmu/fall2014/11791/>.

Task 2.1 Installing JDK

If you have the latest JDK 6 installed¹, you could skip this task.

We assume you have experience in Java programming, but we still need to clarify the Java environment for the course. If you don't have any Java experience, probably you need to look for a Java textbook. It might also be fine if you think you have tons of experience in programming in C++/C# and you feel confident to learn Java by just reading others' codes and guessing their meanings. It's up to you!

1. Visit <http://www.oracle.com/technetwork/java/javase/downloads/index.html>, and choose the platform you are using to download JDK 6 SE 45².

Q1 Can I just install JRE instead of JDK?

A1 No.

¹You should check out the latest version at <http://www.oracle.com/technetwork/java/javase/downloads/index.html> as the version number grows really fast

²By the date of August 28, 2013

Q2 Can I install OpenJDK instead of SunJDK (or OracleJDK)?

A2 Sure, you can. But be aware that sometimes only binary files (aka JRE) are installed under a folder named `openjdk-version`, rather than `openjre-version`, which is a bit confusing.

Q3 Can I install JDK 7, 5 or older versions?

A3 You are not recommended to install JDK 7, since you have to modify the Maven pom file to compile your project, and the cluster that we will run and test your components does not have JDK 7 set up yet. But it would be fine (theoretically) if you have just JDK 5 installed, but it is still not recommended. Versions older than 5 should be completely avoided.

Q4 Can I use an earlier version of JDK 6 (e.g., 6u4)?

A4 It may not put you in a trouble most of time, but in some rare cases, we did find an exception was thrown due to a bug not in our code but in the runtime environment. Therefore, we recommend you to upgrade your JDK 6 to the latest version.

2. Install JDK from the executable file if available, and set PATH manually (if you are using a Windows machine). The Java installation page (at <http://www.oracle.com/technetwork/java/javase/index-137561.html>) might be useful to you.

Task 2.2 Installing Git

If you have Git installed, you can skip this task.

You will not need Git (specifically, execute Git goals from command line) in most cases, because we will have EGit (the Git plug-in for Eclipse) installed. But sadly, there is a case you have to install the Git, when the Maven plug-in for Eclipse (aka m2e) does need Git (not EGit) and the SCM URI you specified to execute Git commands. (Don't know what SCM is? Probably you need to go back to the previous task.)

1. Visit <http://git-scm.com/downloads> to download Git 1.8.4.
2. You can refer to the Pro Git book for how to install Git for different platforms (at <http://git-scm.com/book/en/Getting-Started-Installing-Git>), and how to set up Git (at <http://git-scm.com/book/en/Getting-Started-First-Time-Git-Setup>).

Task 2.3 Maven

If you have Maven installed, you can skip this task.

Similar to the Git installation, you will not need a standalone (as opposed to the m2e plug-in) Maven most of time, since m2e has an embedded Maven runtime by default. But we (and some other developers) found that in some environments, m2e could not find the correct installation path of the embedded runtime to execute certain goals (e.g., `deploy`, `release:prepare`, `release:perform`). If you find a feasible solution to get rid of this, please let us know.

1. Download Maven 3.1.0 (Binary, either tar.gz or zip) from <http://maven.apache.org/download.html>.
2. Follow the installation instructions (for difference platforms) at the bottom of the page to install Maven. The first note from the instruction is:

Maven is a Java tool, so you must have Java installed in order to proceed. More precisely, you need a Java Development Kit (JDK), the Java Runtime Environment (JRE) is not sufficient.

Task 2.4 Eclipse (Git, Maven plug-ins integrated)

If you have an Eclipse IDE for Java Developers with version ≥ 3.7 , you could probably skip this task. But if you are stuck in a situation where you were told Eclipse is missing a plug-in, you might want to return to this section. If you have other packages (Eclipse Classic or Eclipse for Java EE Developers), please do not skip this task.

1. Download Eclipse IDE for Java Developers 4.3 at <http://www.eclipse.org/downloads>.

Q1 Can I use an older version of Eclipse?

A1 You can try to use an older version, but some plug-ins (e.g., m2e) might complain about the Eclipse version if it is older than 3.7.

Q2 Can I use other Eclipse packages?

A2 Eclipse IDE for Java Developers includes almost all the Eclipse components (jdt, EGit, m2e, and so on) we need for this course. You could also work with other packages, e.g., Eclipse IDE for Java EE Developers or Eclipse Classics, and as they don't come with such plug-ins by default, you have to install these plug-ins all by yourself.

2. Install Eclipse by simply uncompressing the downloaded package.
3. Use the default workspace path or create your own workspace, as shown in Figure 2.1. And finally, we could see the Eclipse Welcome view at the end of workspace initialization. See Figure 2.2.

That's the start of your development. From now on, everything will become less platform specific, and we will show you how to configure the workspace, create your Maven project and release it in the rest of the homework.

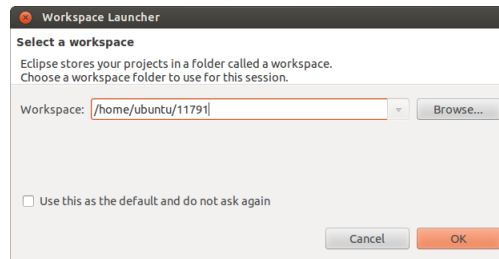


Figure 2.1: Eclipse choose workspace

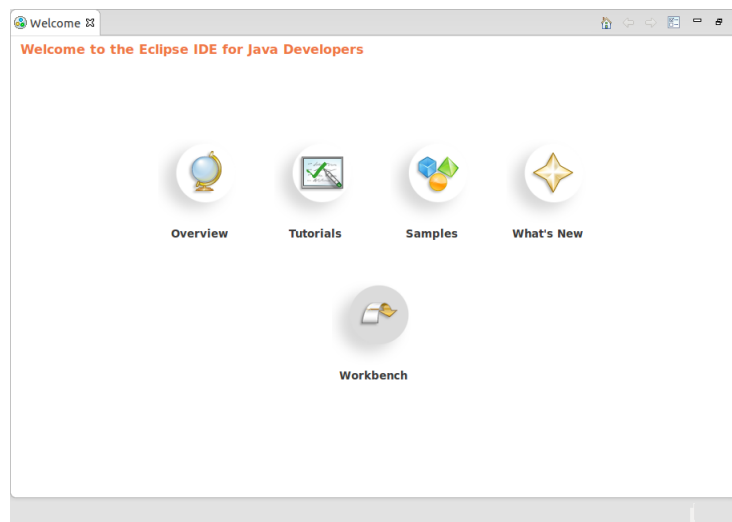


Figure 2.2: Eclipse Welcome view

Task 2.5 A Little Configuration

Letting m2e know your password

1. Click **Edit** (or **Window** depending on your OS) → **Preferences**, and choose **Maven** → **User Settings**, and find the default Maven setting path for your system. See Figure 2.3.
2. Create the `settings.xml` file at the given directory, and copy the text in Listing 2.1 into the file, which will store your ID and passwords. Remember to replace `ID` and `PASSWORD` with your personal Maven project repository account we provide you, and also don't upload this file to any remote repository or share it with others.
3. Go back to **Edit** (or **Window**) → **Preferences**, and choose **Maven** → **User Settings** again, you will see the plugin could find the setting file you specified

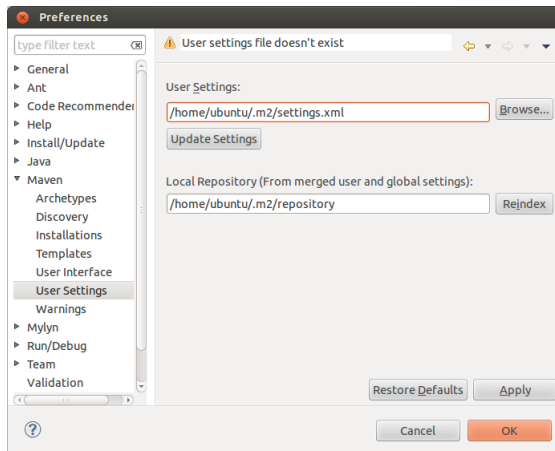


Figure 2.3: Eclipse maven user profile setting

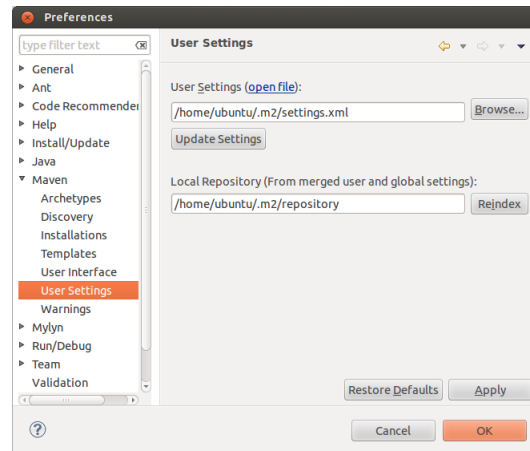


Figure 2.4: Eclipse maven user profile setting

```

1 <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                        http://maven.apache.org/xsd/settings-1.0.0.xsd"
4   >
5   <servers>
6     <server>
7       <id>deployment</id>
8       <username>ID</username>
9       <password>PASSWORD</password>
10    </server>
11    <server>
12      <id>snapshots</id>
13      <username>ID</username>
14      <password>PASSWORD</password>
15    </server>
16  </servers>
17 </settings>

```

Listing 2.1: Configuring settings.xml

(see Figure 2.4).

Importing Apache UIMA code style template

To development a software as a team, members should always adopt the same code conventions to improve the readability and maintainability of the project. We suggest you to view the *Code Conventions for the Java Programming Language* at <http://>

www.oracle.com/technetwork/java/codeconv-138413.html, which was published from Oracle. For our course homeworks, you are required to adopt a set of more specific coding conventions from Apache UIMA project. Details can be found at <http://uima.apache.org/codeConventions.html>. At the bottom of the page, you could find a link to download the Eclipse code style template³.

1. Download the template and save it in your local filesystem.
2. Click **Window** → **Preferences**, then go to **Java** → **Code Style** → **Formatter**, and click **Import...**

Remember before you finish editing a Java file, press **Ctrl+Shift+F** to perform an automatic code formation.

Another optional but useful tool for you to check your code style is the Eclipse Checkstyle plug-in. You can learn how to download and install the plug-in at <http://eclipse-cs.sourceforge.net/>.

³http://uima.apache.org/downloads/ApacheUima_EclipseCodeStylePrefs.xml

Task 3

Creating Your First Maven Project, Writing a Piece of Code and Performing Your First Maven Release

Task 3.1 Importing the Empty Project from GitHub

The following instruction to guide you create a Maven Git project is similar to a blog post¹. In fact, you could create a Maven Git project in various ways. For example, you can create a Maven project first, and share it to Git, or you can just create a Java project, and add Maven nature, then share it to Git, and so on. But, we will show you the way that we think the most comfortable to achieve the goal.

Checking out the empty GitHub project

1. Click **File** → **Import...** to get ready to import the empty project hosted on GitHub to local filesystem, as shown in Figure 3.1.
2. Then select **Git** → **Projects from Git** in the **Import** window. See Figure 3.2.
3. Select **URI** as the repository source, then click **Next**. See Figure 3.3.
4. Copy the URI from your GitHub repository page (c.f. Figure 1.3), and paste it here as the URI, and the Host and Repository path will be automatically generated for you. You also need to type in your Git Repository username and password, and probably you want to **Store in in Secure Store**. So just check the box at the bottom of the window, and then click **Next**. See Figure 3.4.

Important note: There are two GitHub main authentication approaches *HTTPS*, and *SSH*. For more information refer to the GitHub docs. If one uses *HTTPS*, operations `git push` and `git pull` will ask you for the

¹<http://springinpractice.com/2012/05/06/mavenizing-an-empty-github-project-in-eclipse/>

TASK 3. CREATING YOUR FIRST MAVEN PROJECT, WRITING A PIECE OF CODE AND PERFORMING YOUR FIRST MAVEN RELEASE

14

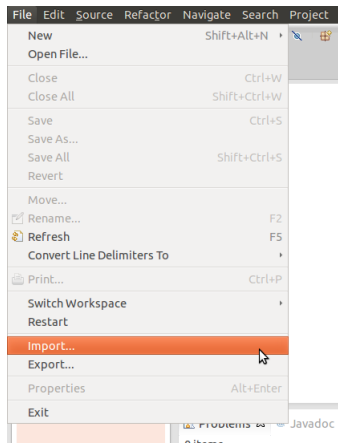


Figure 3.1: Importing project hosted on GitHub

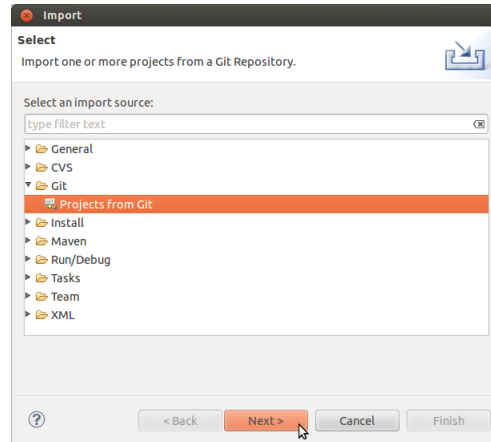


Figure 3.2: Choosing Git project

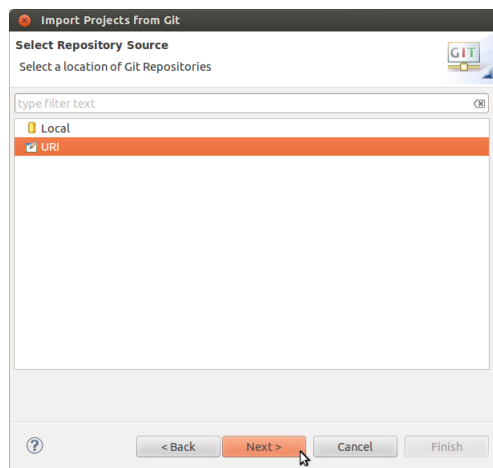


Figure 3.3: Choosing to import Git project specified by a URI

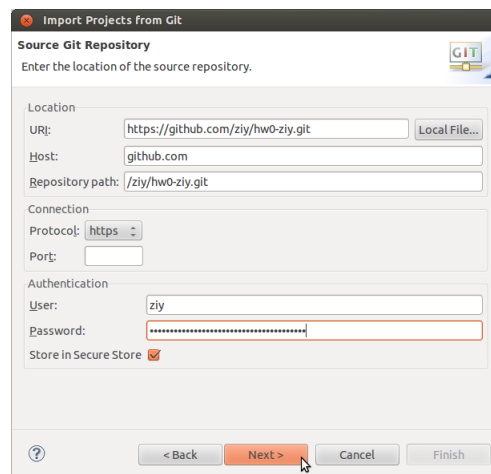


Figure 3.4: Typing in your GitHub repository URI

user and password. In Figure 1.3, we give an example of the *HTTPS* authorization. This type of authorization can be used in Eclipse, because when you import a GitHub project in Eclipse, Eclipse asks you the user id and the password. These id/password are subsequently memorized and used when necessary. However, Maven (see 3.3) will only be able to use the *SSH* authorization.

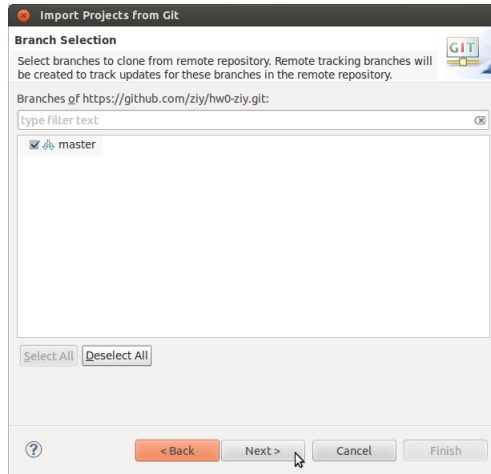


Figure 3.5: Choosing master branch

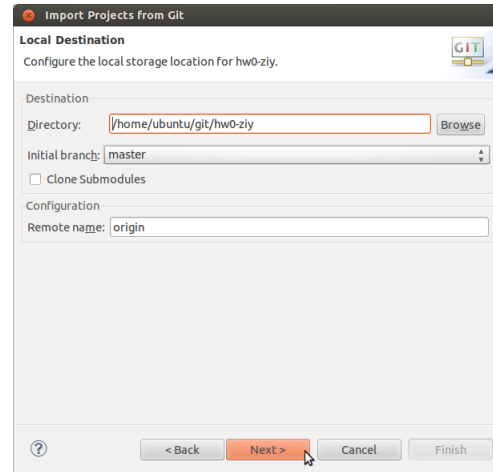


Figure 3.6: Choosing local Git project location

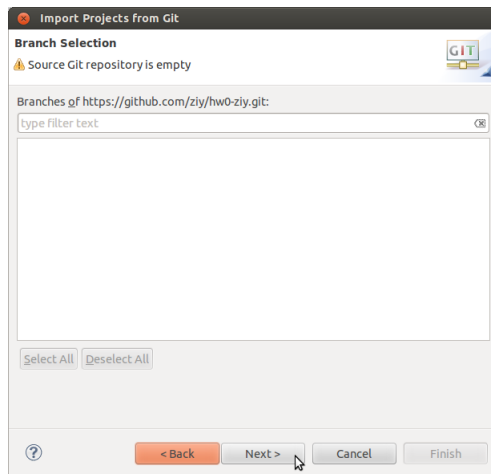


Figure 3.7: Failed to choose master branch

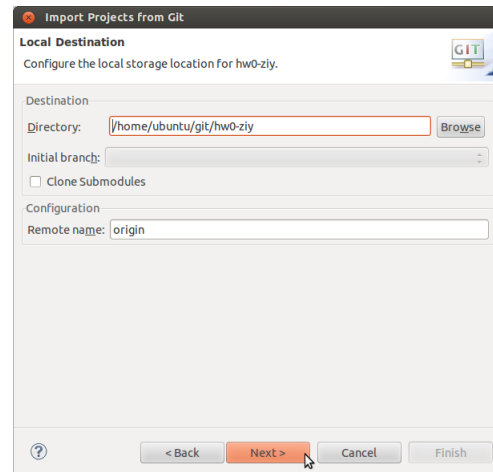


Figure 3.8: Choosing local Git storage location for the project

5. Most of the time, you will see EGit plug-in is able to detect the branch **master** does exist in the repository (as shown in Figure 3.5). Click **Next** to proceed to the next step where you can specify the local storage location for the project. If you are not sure what it means by a local storage, you should go back to Task 1 to learn one of the most important features of Git - distributed version control. Type in your local storage path in **Directory**, and **master** will be selected by default as the **Initial branch** as shown in Figure 3.6. Now you could proceed to next step.

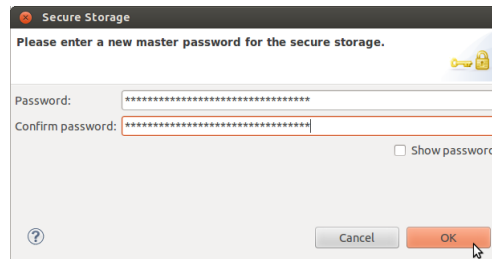


Figure 3.9: Typing in your password

Q1 If the master branch cannot be found automatically by EGit, i.e., Figure 3.7 is shown, where it says **Source Git repository is empty** and no branch can be selected, instead of Figure 3.5. What should I do?

A1 No problem. You could still click **Next** to proceed, and in the next window, you can still assign a local storage for the project, but you cannot specify the **Initial branch** (see Figure 3.8). You can specify the master branch at a later stage when we try to perform your first git-commit.

6. Finally, if you chose to store the password in a secure store, you need to type in (of course twice) a master password for the secure storage. Note that this is not your Maven or GitHub password, and it is only used within the scope of Eclipse. See Figure 3.9.

Creating a Maven project

So far, the empty project has been checked into the local filesystem. At this point, files that Eclipse should rely on, such as `.project`, have not been created, in other words, we haven't set up an Eclipse project yet.

1. Now, we select **Use the New Project wizard** in the "Project Import" window and click **Finish** to begin create an Eclipse Maven Project, as you can see in Figure 3.10.
2. Then select **Maven** → **Maven Project**, then click **Next** to continue. See Figure 3.11.
3. In the "New Maven Project" window, you select both **Create a simple project (skip archetype selection)** and **Use default Workspace location**. See Figure 3.12. If you have already forgotten what archetype and artifact are, you really need to go back to Task 1 to review the basic concepts of Maven. For Homework 0, you will not need an archetype to base on, but you will soon realize that Maven archetypes are crucial to software development. We will provide you

TASK 3. CREATING YOUR FIRST MAVEN PROJECT, WRITING A PIECE OF CODE AND PERFORMING YOUR FIRST MAVEN RELEASE

17

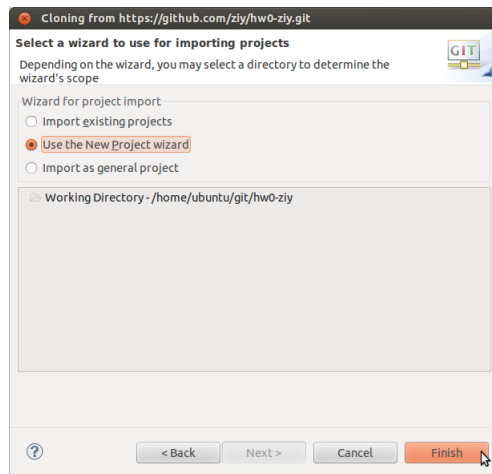


Figure 3.10: Creating new project

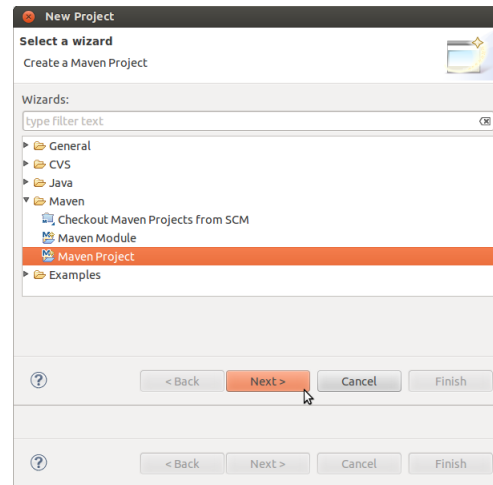


Figure 3.11: Choosing to create new Maven project

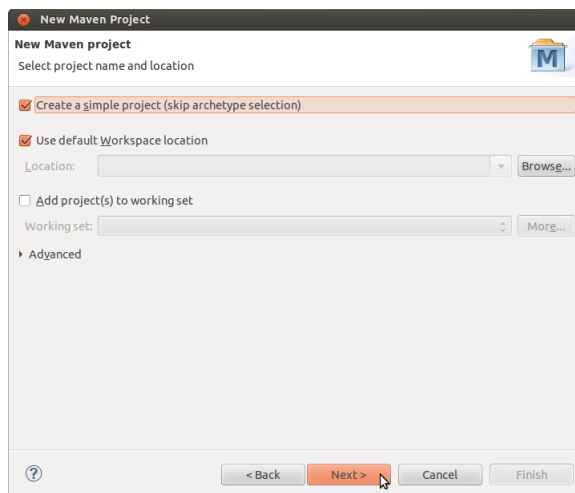


Figure 3.12: Specifying Maven Project location

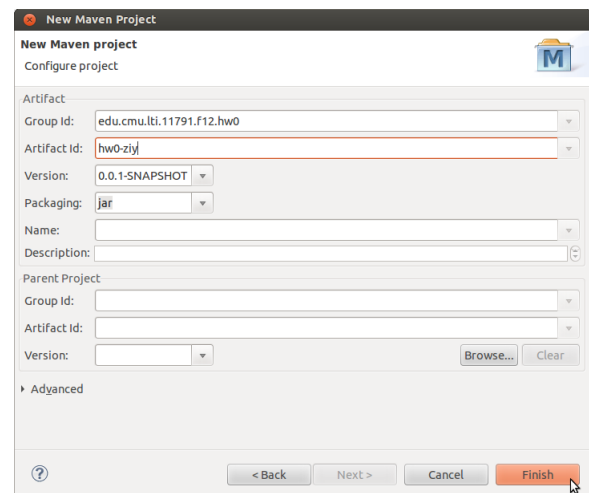


Figure 3.13: Typing in basic information for Maven artifact

the archetype you need for Homework 1 and 2 (of course, via our course Maven repository).

4. Then, in the next window, you need to type in the **Group Id** and **Artifact Id** for the artifact you are going to create. The Group Id for Homework 0 is

edu.cmu.lti.11791.f14.hw0

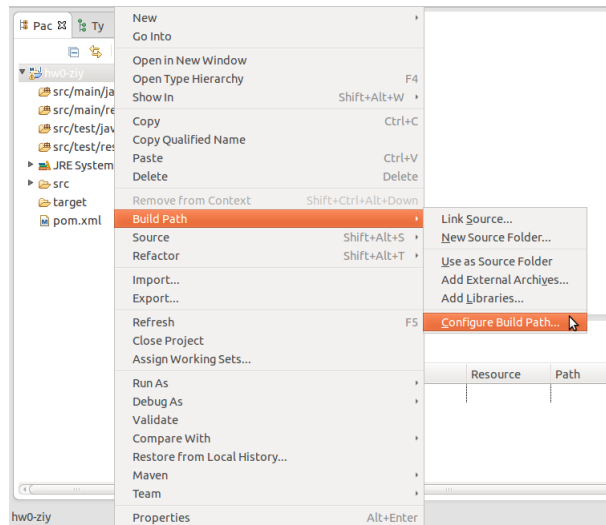


Figure 3.14: Configuring Build Path

and the Artifact Id should be

hw0-ANDREW_ID

Replace **ANDREW_ID** with your Andrew ID. You must type in the correct information to create your artifact, since both Group Id and Artifact Id will be used to generate the jar file and eventually submit the jar to the correct folder in our Maven repository. You do not need to change the rest of the configuration, and press **Finish** to complete creating the Maven project. On some platforms², you will need to fill in a Name and Description otherwise the wizard will generate an error.

5. If you find the project is built based on an earlier version of Java (e.g., JDK 1.5), then it is time to change it to JDK 6. If JDK environment that your project is using is 1.6, then you can skip this additional step. First, right-click on the project name (i.e. hw0-ziy) in the “Package Explorer” view, which is usually located on the left of your workspace. Then select **Build Path** → **Configure Build Path...**, as shown in Figure 3.14.
6. Then click the **Libraries** tab, select **JRE System Library**, and click **Edit**. See Figure 3.15.
7. In the popup window, select any Java 1.6 environment you have installed. Note that the options in your workspace might be totally different from what you see in Figure 3.16.

²For example, Windows XP/Helios as reported by Martin van Velsen

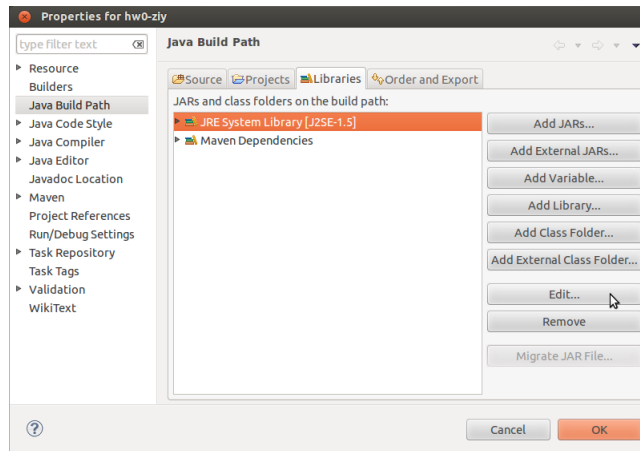


Figure 3.15: Viewing JRE version

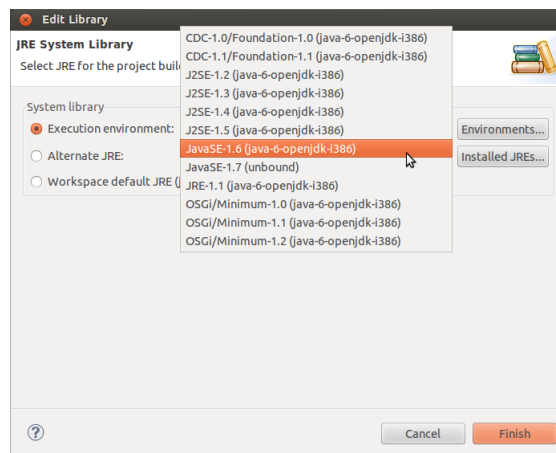


Figure 3.16: Choosing a different JRE version

Sharing your project via Git

If you encountered the problem we described in Figure 3.7 and 3.8, you need to re-establish the connection from the local Git repository to the repository you built on GitHub. If you do not have such issue, you can skip this part.

You need to right-click on the project name, and select **Team** → **Share Project...** in order to share your project on GitHub. See Figure 3.17. You should be able to proceed by selecting the repository type (Git). If you still find come across the NO-HEAD issue (see Figure 3.18) after you share your project, don't panic as you can specify the branch during your first git-commit.

TASK 3. CREATING YOUR FIRST MAVEN PROJECT, WRITING A PIECE OF CODE AND PERFORMING YOUR FIRST MAVEN RELEASE

20

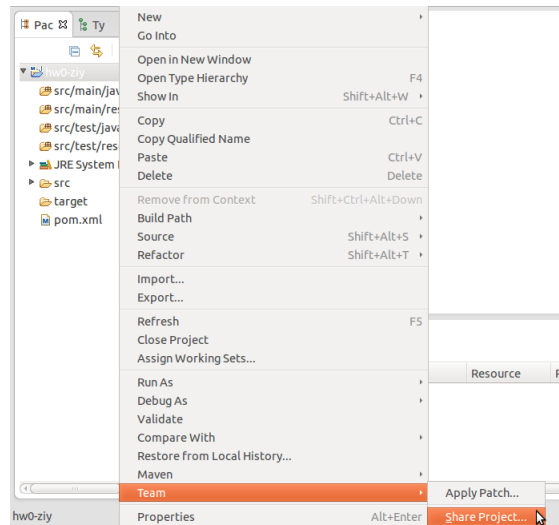


Figure 3.17: Sharing project again

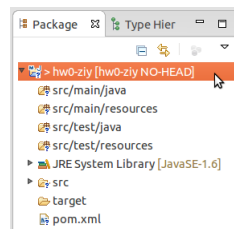


Figure 3.18: Nohead issue

Task 3.2 Committing and Pushing Your Maven Project Back to Git

You should see a “greater than” symbol between the project icon and your project name (see Figure 3.19, which means you have made some changes to the project so that there exist some differences between your current workspace version and the branch head. Question marks on the project element icon represent the corresponding elements are not indexed yet. You might be wondering what changes you’ve made because you thought you haven’t written any code. Actually, you have created a Maven project, which process will generate the `pom.xml` file, and modified the Eclipse configuration files. So let’s perform our first git-commit.

1. Right-click on the project name, and select **Team** → **Commit...** See Figure 3.19, and then type in your username and email address on GitHub in the popup “Identify Yourself” message window.

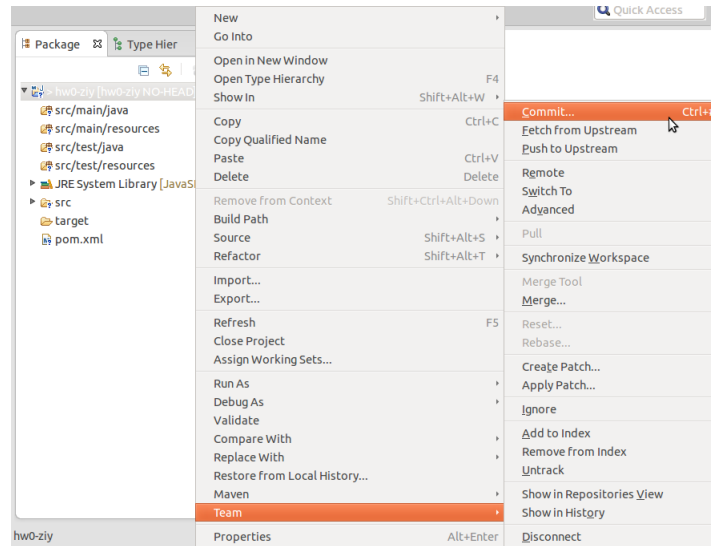


Figure 3.19: Performing a git-commit

2. In the “Commit Changes” window, you are allowed to choose the files you want to commit (and also automatically add to the index). As you can see in Figure 3.20, during creating the empty Maven project, several Eclipse and Maven related configuration files are generated. Moreover, type in a commit message to describe what changes you’ve made and leave your name as well as your email address (which is a convention) in the committer field. Finally, by clicking **Commit**, you’ve done with your first git-commit. You can see in Figure 3.21, the “greater than” symbol disappears, and the question marks on the committed files become a repository icon, which means the files are in the latest version. You could also find your git-commit helps the code merge into the new master branch from a NO-HEAD branch.

If you are using SVN, then you’ve done with synchronizing your local workspace with the remote repository once you execute a commit command, but remember the feature of Git? Your project repository is distributed, which means your previous git-commit conceptually affects all your project repositories, but in fact you should make it happen with an additional *push*. A nice picture at <http://gitready.com/beginner/2009/01/21/pushing-and-pulling.html> may help you better understand what is actually happening when you perform git commit, add, push, fetch, pull, etc.

3. Right-click on the project name, and select **Team** → **Push to Upstream**. See Figure 3.22.
4. Now you can see a progress indication window pops up (see Figure 3.23), which says **Pushing to remote repositories**.

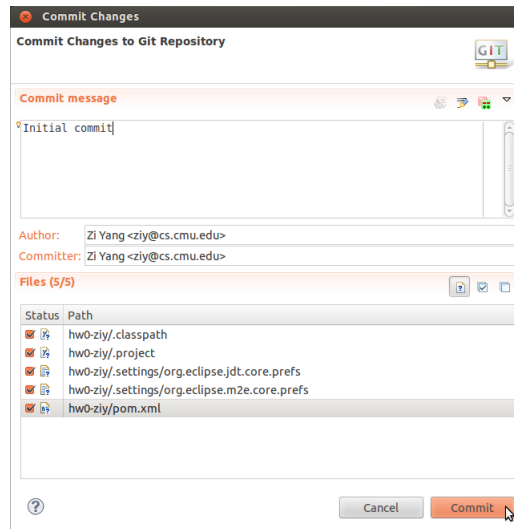


Figure 3.20: Viewing and confirming commit message

5. Finally, you could see the push results in the “Push Results” window. Click **OK** to close the window and get back to your workspace.

In your next homework, you will need to use other Git commands within Eclipse.

Task 3.3 Adding a dependency from Maven repository

Maven repository stores all the artifacts that repository users have deployed, and we can take the advantage of Maven repository to add project dependencies on your other projects or other developers’ projects. In this task, you are going to add a dependency of Stanford CoreNLP, which is hosted on the Central Repository. In the next homework, you will find several useful dependencies in our project repository.

1. Do you remember how the configuration of your Maven project is stored? Yes, the pom.xml file! Open it by double clicking the file name, and you will see the pom.xml is opened with “Maven POM Editor” as in Figure 3.25. If you find it is opened with a XML Editor or Text Editor, you should right click on the file name, you choose **Open With** → **Maven POM Editor**.
2. Before you add a dependency to your Maven project, you might want to look at all the tabs by clicking the tab name at the bottom of the Maven POM Editor. In the “Overview” tab, you need to type in some SCM information. Maven will use this information to connect to your remote GitHub repository. In general, the format of the scm is:

```
scm:<SCM_PROVIDER><DELIMITER><PROVIDER_SPECIFIC_PART>
```

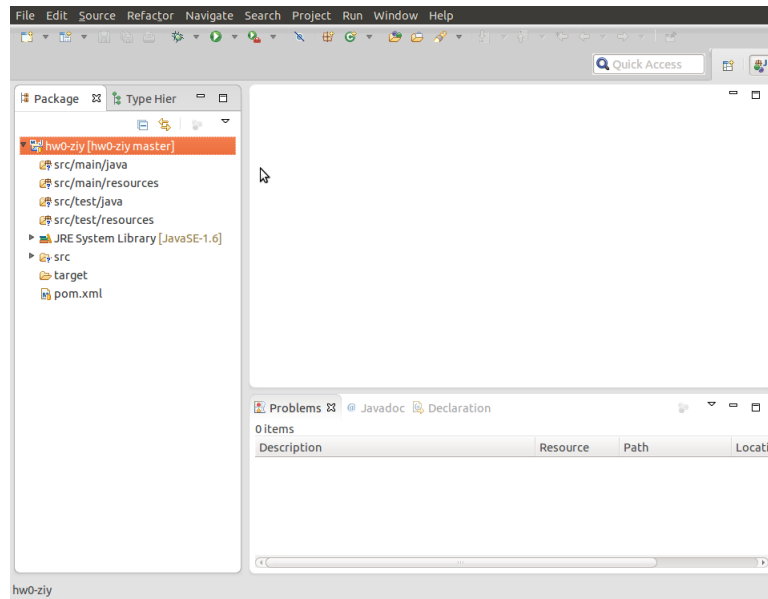



Figure 3.21: Committed project

For our purposes, it is best to use the following format:

```
scm:git:git@github.com:GITHUB_ID/REPOSITORY_ID.git
```

For this assignment specifically, you will need:

```
scm:git:git@github.com:GITHUB_ID/hw0-ANDREW_ID.git
```

Note that here we use *SSH* and not *HTTPS* authorization! To make it working, you need to create an SSH key as mention in Section 1.1. Please, refer to the documentation on the Git site.

You need to type in the SCM URL corresponding to your remote repository in **Connection** and **Developer** under **SCM** field. Details of SCM URL can be found at <http://maven.apache.org/scm/scm-url-format.html>.

3. Next, you should inform Maven what the URL of our course Maven repository is, which can be done by editing the POM file directly. Click the last tab “pom.xml”, you are able to edit directly from the Maven POM Editor. Compare your POM content with Listing 3.1, and add the missing lines to your POM.

As you’ve already specified the Group Id, Artifact Id, SCM information in early steps through the setup wizard or “Overview” tab of Maven POM Editor, you will find these contents (similar to Line 1 to 11 in Listing 3.1) are already there in you POM. You need to copy the content from Line 12 to Line 48 from

```

1 <project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
    apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
5  <groupId>edu.cmu.lti.11791.fl3.hw0</groupId>
  <artifactId>hw0-ANDREW_ID</artifactId>
7  <version>0.0.7-SNAPSHOT</version>
  <scm>
9    <connection>scm:git:git@github.com:GITHUB_ID/hw0-ANDREW_ID.git</
      connection>
    <developerConnection>scm:git:git@github.com:GITHUB_ID/hw0-
      ANDREW_ID.git</developerConnection>
11  </scm>
  <repositories><repository>
13    <id>deployment</id>
    <url>http://mu.lti.cs.cmu.edu:8081/nexus/content/groups/course</
      url>
15    </repository></repositories>
  <distributionManagement>
17    <repository>
    <id>deployment</id>
19    <url>http://mu.lti.cs.cmu.edu:8081/nexus/content/repositories/
      course-releases</url>
    </repository>
21    <snapshotRepository>
    <id>snapshots</id>
23    <url>http://mu.lti.cs.cmu.edu:8081/nexus/content/repositories/
      course-snapshots</url>
    </snapshotRepository>
25  </distributionManagement>
  <build><plugins>
27    <plugin>
    <groupId>org.apache.maven.plugins</groupId>
29    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
31      <source>1.6</source><target>1.6</target>
    </configuration>
33    </plugin>
    <plugin>
35      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-release-plugin</artifactId>
37      <version>2.2.1</version>
      <executions><execution>
39        <id>default</id>
        <goals><goal>perform</goal></goals>
41        <configuration>
          <pomFileName>hw0-ANDREW_ID/pom.xml</pomFileName>
43        </configuration>
        </execution></executions>
45      </plugin>
    </plugins></build>
47 </project>

```

Listing 3.1: Configuring pom.xml

TASK 3. CREATING YOUR FIRST MAVEN PROJECT, WRITING A PIECE OF CODE AND PERFORMING YOUR FIRST MAVEN RELEASE 25

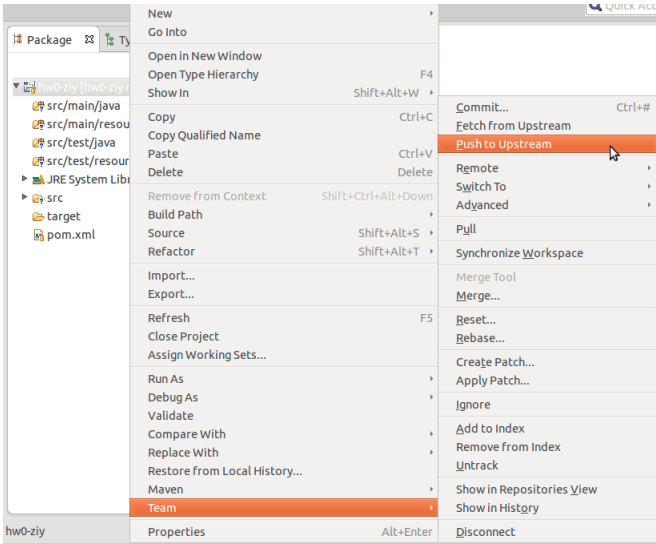


Figure 3.22: Performing a git push

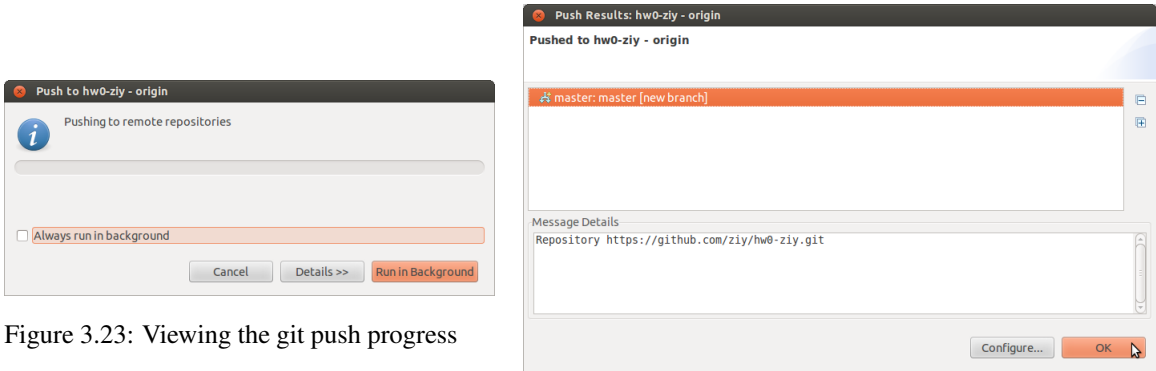


Figure 3.23: Viewing the git push progress

Figure 3.24: Viewing the git push result

Listing 3.1 into your POM. We will briefly describe what the meaning of each POM block is.

- repositories (Line 12 to 15) define a subset of your repositories (declared in your settings.xml file) related to the current Maven project. You could retrieve all the existing artifacts from these repositories to make your current project depend on. Note that once you add a new **repository** item in the **repositories** group, you are enabled to view the artifacts in the “Maven Repositories” view (in next task).
- distributionManagement (Line 16 to 25) will be used only if you plan to make a release for your artifact. For example, in Listing 3.1, repositories for releases and snapshots

TASK 3. CREATING YOUR FIRST MAVEN PROJECT, WRITING A PIECE OF CODE AND PERFORMING YOUR FIRST MAVEN RELEASE 26

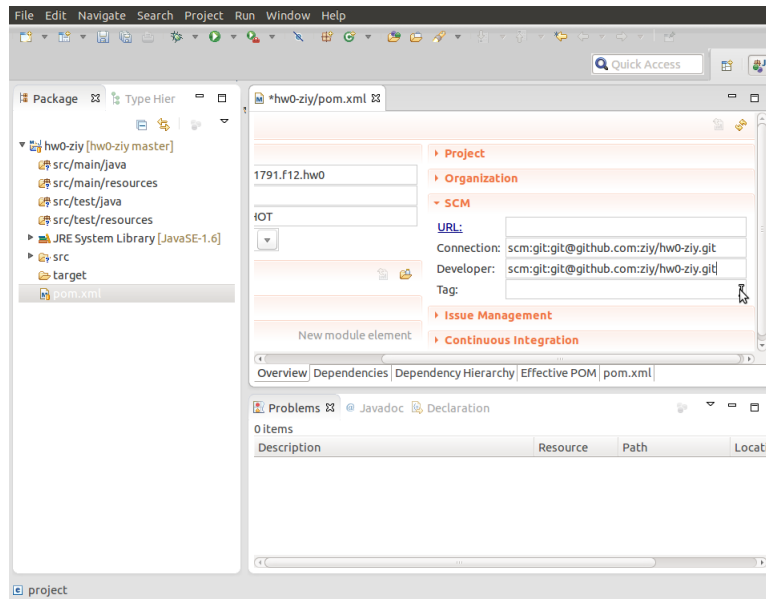


Figure 3.25: Opening a POM file

are separately defined.

plugins (Line 26 to 46) include specific plug-ins required by the project to perform Maven goals. For example, you should specify the location of your pom.xml file when you perform the `release:perform` goal.

Q1 Is m2e so stupid? Part of the POM can be edited with perfect GUI or wizard, but why do we need to manually type in additional XML texts?

A1 We are also able to add dependencies (in the next few steps) and add plugins into the pom.xml content with m2e GUI, but you have to add fields, such as **repositories**, **distributionManagement**, and so on, manually and directly to pom.xml, because the m2e plugin does not support add such fields in a more effect way from Maven POM Editor. Guess what it means? It means m2e does not recommend us to manually add these contents to pom.xml, although they are crucial to a Maven project.

Can you find the difference between the contents those can be added with GUI and those cannot? Contents added by GUI are usually project specific, while manually added contents are usually global contents shared across different projects in your workspace and organization, which means a “meta” pom should be generated for all the individual “child” Maven project. That’s how Maven and archetype were originally introduced. We

TASK 3. CREATING YOUR FIRST MAVEN PROJECT, WRITING A PIECE OF CODE AND PERFORMING YOUR FIRST MAVEN RELEASE 27

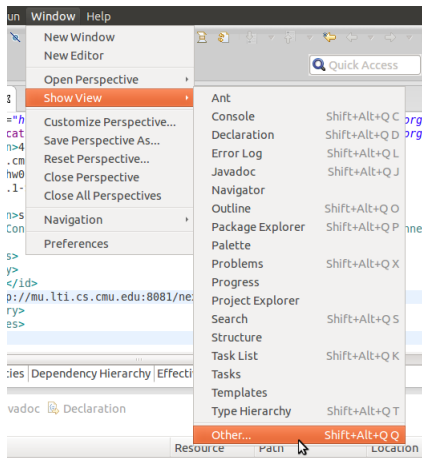


Figure 3.26: Opening a new view

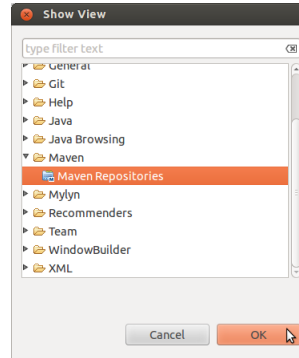


Figure 3.27: Choosing Maven Repository view

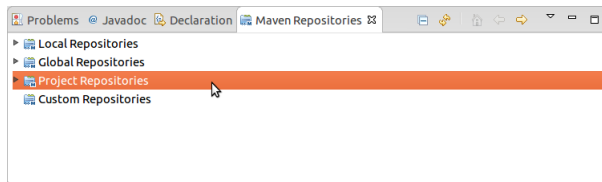


Figure 3.28: Viewing additional project repository from Maven Repository view

will learn to create a Maven project from an archetype in your next homework.

4. Finally, try to press **Ctrl+Shift+F** within the pom.xml editor scope!
5. Now click **Window** → **Show View** → **Other...** to select a new view. See Figure 3.28.
6. In the “Show View” window, select **Maven** → **Maven Repositories** to open the “Maven Repositories” view. See Figure 3.29.
7. You can see the “Maven Repositories” view at the bottom of your workspace, you are able to unfold “Project Repositories” since you have added items in the **repositories** field of your pom.xml file.

Q1 I could not unfold the **Project Repositories** or there is no artifact inside it.

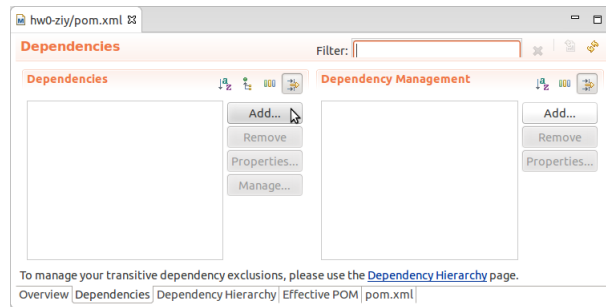


Figure 3.29: Adding a Jar dependency

A1 You should try the following three solutions:

- a) As you did before, go back to **Edit** (or **Window**) → **Preferences**, and choose **Maven** → **User Settings** again, you will see the plug-in could find the setting file you specified (see Figure 2.4), and click **Update Settings**.
- b) Click the “double arrows” icon (second icon in the top-right corner of “Maven Repositories” view) to reload settings.xml.
- c) Right-click on each individual repository in “Maven Repositories” view (e.g. **deployment** in our example), select **Minimum Index Enabled** or **Enable Full Index** instead of **Disable Index Details**.

8. Now you could add an artifact dependency for your first Maven Project. Go to the **Dependencies** tab in Maven POM Editor. See Figure 3.29. Click **Add...** in the “Dependencies” area.
9. In the popup window, directly type in “stanford-corenlp” in the search box under the message **Enter groupId, artifactId or sha1 prefix or pattern (*)**, and select **edu.stanford.nlp stanford-corenlp**. Now you will find the three fields at the top: **Group Id**, **Artifact Id**, and **Version** will be automatically filled with the correct information. Click **OK** to continue.
10. Finally, we could see the newly added dependency “stanford-corenlp: 1.3.3” is shown in the left column of “Dependencies” field. In addition, you will find the new dependency is also added to the Maven Repositories in the “Project Explorer” view. See Figure 3.31.

Now you have added the dependency on Stanford CoreNLP, and you will base on the dependency to write in your own code in the next task.

29

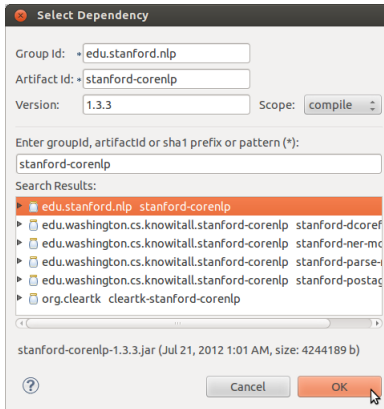


Figure 3.30: Searching for CoreNLP

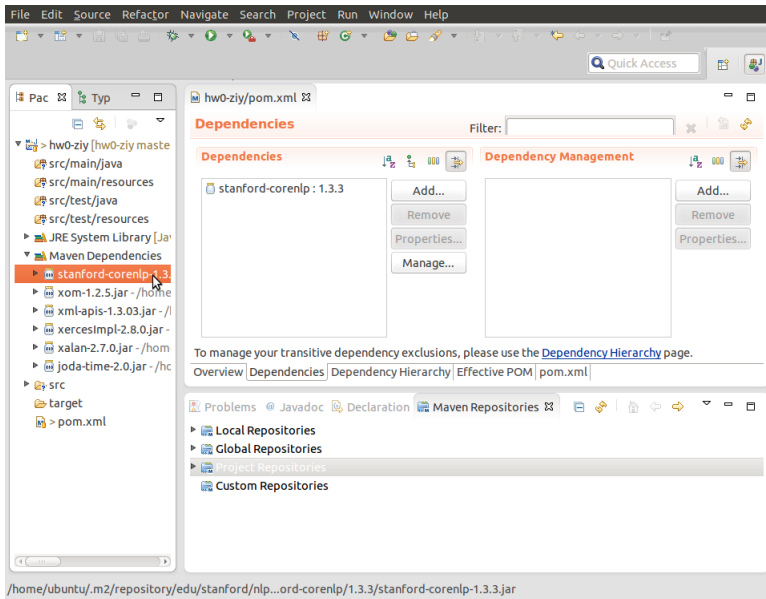


Figure 3.31: Stanford CoreNLP appearing in Maven Dependencies

Task 3.4 A Simple Code Based on Stanford NLP Tokenizer

If you have no idea of token, tokenizer, and tokenization, you should take a look at the Wikipedia page at http://en.wikipedia.org/wiki/Lexical_analysis. It says

A token is a string of characters, categorized according to the rules as a symbol (e.g., IDENTIFIER, NUMBER, COMMA). The process of forming

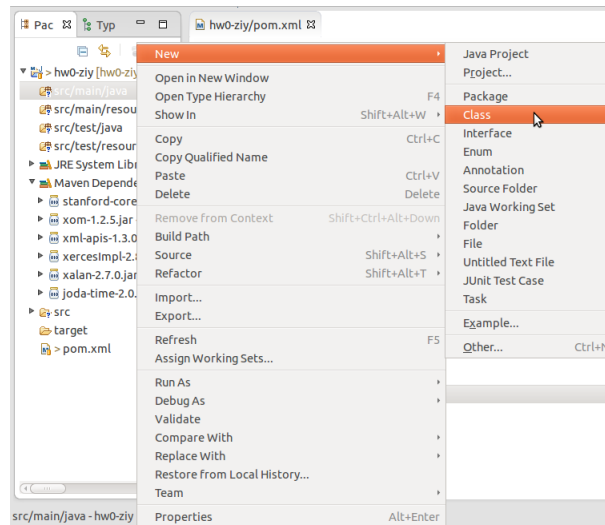


Figure 3.32: Creating a new Java class

tokens from an input stream of characters is called tokenization.

You will need to write a simple tokenizer that takes a sentence from console as the input, and prints out the tokens back to the console, based on Stanford CoreNLP. If you are curious why the directory layout looks different from an ordinary Java project, you need to review some basics you learned before³. In this task, you need to create the Java file under `src/main/java`, which is normally used as the base directory for Java source codes.

1. Now, you need to create a new Java class by right-clicking **src/main/java**, and select **New** → **Class**. See Figure 3.32.
2. In the popup window, keep the **Package** as blank (aka default package), and type in “DependencyExample” next to **Name**. Please do not try to specify a different name or package, because our evaluation script will try to evaluate your submission by looking for the main method within `src/main/java/DependencyExample`. Click **Finish** to proceed.
3. Now, the Java Editor will be automatically opened to allow you to edit your Java code. Please copy the content from Listing 3.2 to your Java code.
4. You could try to test your code with some simple sentence, e.g., “I’m feeling good!”. Your output should be the same as in Figure 3.34. That looks good, right? It can split “m” from “I” and can keep punctuations (“!”).

³<https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>

TASK 3. CREATING YOUR FIRST MAVEN PROJECT, WRITING A PIECE OF CODE AND PERFORMING YOUR FIRST MAVEN RELEASE 31

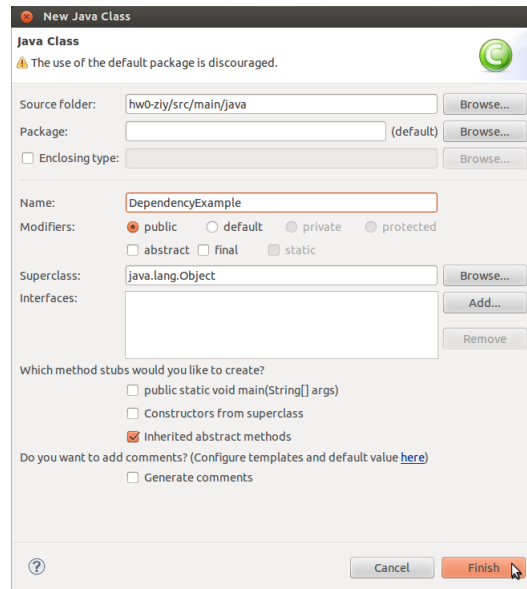


Figure 3.33: Typing in the classname

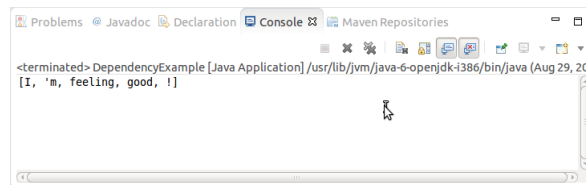


Figure 3.34: Showing tokenization result

Q1 How can I run the Java code within Eclipse? And how can I send command line arguments in Eclipse?

A1 These are Eclipse/Java related questions. You can refer to <http://www.javaprogrammingforums.com/java-jdk-ide-tutorials/362-how-send-command-line-arguments-eclipse.html> or search for other similar posts or articles.

Task 3.5 Performing a Maven Release

In the last task, you will need to submit your Java code by performing a release of your code. But remember that Maven release plug-in will check if all the changes you

```
1 import java.io.StringReader;

3 import edu.stanford.nlp.ling.Word;
import edu.stanford.nlp.objectbank.TokenizerFactory;
5 import edu.stanford.nlp.process.PTBTTokenizer.PTBTTokenizerFactory;
import edu.stanford.nlp.process.Tokenizer;

7
/**
9  * An example for Homework 0 of 11791 F13
10  *
11  * @author Zi Yang <ziy@cs.cmu.edu>
12  */
13 public class DependencyExample {

15     /**
16      * Tokenize a sentence in the argument, and print out
17      * the tokens to the console.
18      *
19      * @param args
20      *      Set the first argument as the sentence to
21      *      be tokenized.
22      */
23     public static void main(String[] args) {
24         TokenizerFactory<Word> factory =
25             PTBTTokenizerFactory.newTokenizerFactory();
26         Tokenizer<Word> tokenizer =
27             factory.getTokenizer(new StringReader(args[0]));
28         System.out.println(tokenizer.tokenize());
29     }
30 }
```

Listing 3.2: A simple tokenization program based on Stanford CoreNLP tool

have made have been checked into the remote repository (i.e. GitHub in our case). So, let's perform a git-commit and a git-push.

1. Similar to what you did earlier, you execute git-commit and git-push to the project, and you could see the “greater than” symbol disappears and a “master” label is attached to project path, which means you are successful with your git-commit and git-push.
2. Sometimes, the embedded Maven runtime from m2e cannot be successfully executed to perform a release goal. Therefore, we should add the externally installed Maven runtime into the Eclipse. Click **Window** (or **Edit**) → **Preferences** (see Figure 3.38).
3. Select **Maven** → **Installations**. You will see the “Embedded” runtime (as shown in Figure 3.37). Click **Add...** to locate the installation path of your

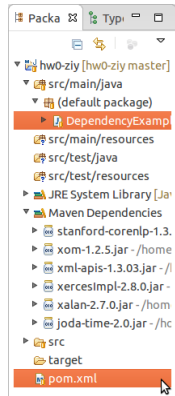


Figure 3.35: Performing a git-commit/push before preparing a release

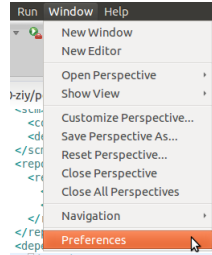


Figure 3.36: Starting to add an external Maven executable

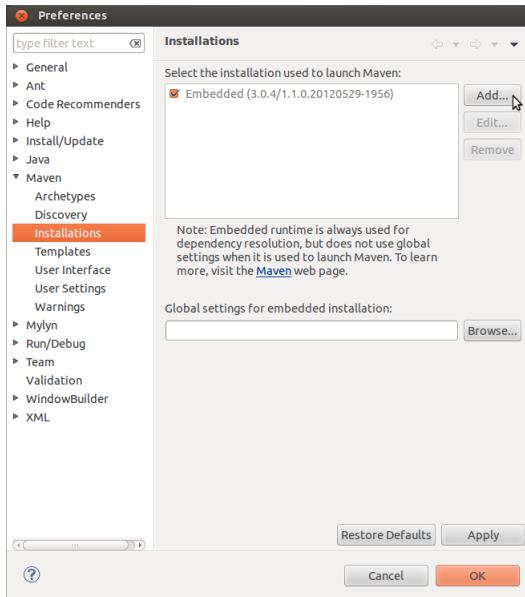


Figure 3.37: Adding another Maven executable

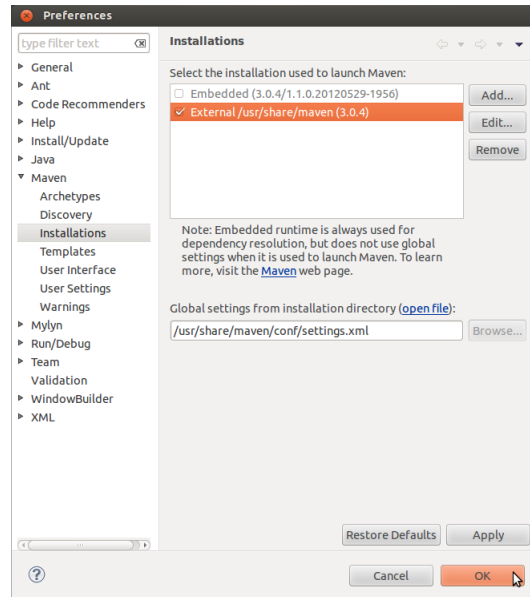


Figure 3.38: Viewing the added external Maven

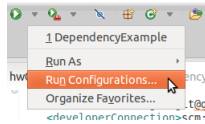


Figure 3.39: Getting ready for a release

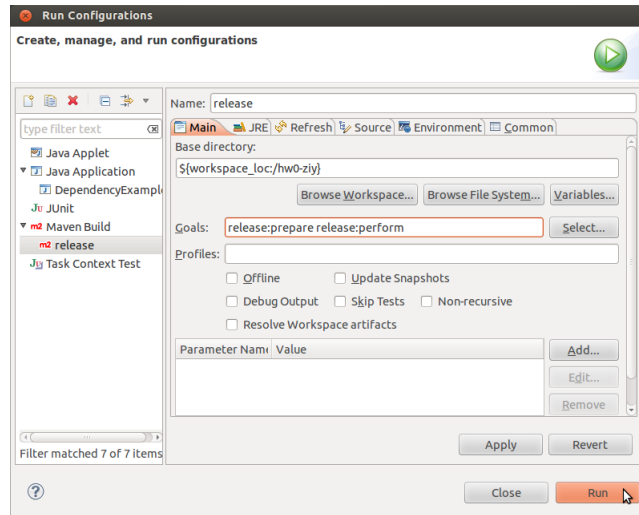


Figure 3.40: Configuring Maven goal

Maven runtime.

4. Then, you could see an “External” runtime in the installation lists. See Figure 3.38.
5. Now you can execute a Maven goal within Eclipse (of course, you can also do that outside Eclipse from command line). Click the down-arrow next to the **Run** button, and select **Run Configurations...** See Figure 3.39.
6. In the “Run Configurations” window, double-click **Maven Build** to create a new Maven goal. See Figure 3.40. Rename your run configuration name as “release” (optional), and click **Browse Workspace...** to select your project, and type in your goals as follows:

```
release:prepare release:perform
```

This actually defines two goals “release:prepare” and “release:perform”. As you will probably encounter tons of errors during this step, we should review some details of the Maven release.

TASK 3. CREATING YOUR FIRST MAVEN PROJECT, WRITING A PIECE OF CODE AND PERFORMING YOUR FIRST MAVEN RELEASE

35

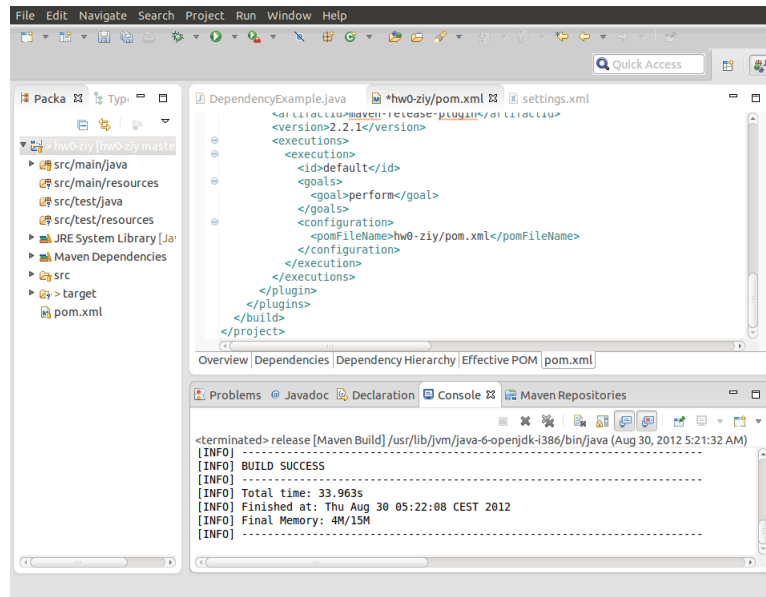


Figure 3.41: A successful release!

The Maven guide⁴ tells us what is happening behind these two goals:

The release:prepare goal will:

- Verify that there are no uncommitted changes in the workspace.
- Prompt the user for the desired tag, release and development version names.
- Modify and commit release information into the pom.xml file.
- Tag the entire project source tree with the new tag name.

The release:perform goal will:

- Extract file revisions versioned under the new tag name.
- Execute the maven build lifecycle on the extracted instance of the project.
- Deploy the versioned artifacts to appropriate local and remote repositories.

If the goals are not executed successfully, a relatively useful message will be printed out to console to help you discover where the problem is.

- Finally after you fixed all the problems (if any), you could see the very pleasant “BUILD SUCCESS” message in the console (as shown in Figure 3.41), which means you’ve done with your Homework 0! Congratulations!

⁴<https://maven.apache.org/guides/mini/guide-releasing.html>

- Q1 The “release:prepare” goal could be executed successfully, but there was an error during “release:perform”. How can I start all over again?⁵
- A1 A successful “prepare” with an unsuccessful “perform” always leads to a very subtle situation. Sometimes, you can try to execute a “release:rollback” Maven goal to revert to the previous status, or you can try to manually delete the `release.properties` file under your project directory, and make sure you are now working on a SNAPSHOT version. (You can check your version from Maven POM Editor, and you have an opened Maven POM Editor, be sure to close it and open it again since it will not be automatically refreshed after you execute a Maven goal.) If it is no longer a SNAPSHOT version, you should modify the **Version** field to something like “0.0.2-SNAPSHOT” (without quotes). You might want to refer to <http://maven.apache.org/guides/mini/guide-releasing.html> about the details of the release process.
- Q2 The process hangs forever during Maven “release:prepare” executing a “git push” command, in other words, it doesn’t terminate or produce any further useful information. What should I do?
- A2 We are sure there is some bug with the plugin related to this issue. But there are still some workarounds to consider.
- a) If you generate a public key with a passphrase (i.e., you typed in something before you pressed **Enter**), then try to regenerate another public key following the same instruction and paste it to GitHub.
 - b) Try to log back into GitHub to see if your email address has been verified.
 - c) Turn to command line to execute Maven goal by typing `mvn release:prepare release:perform` from the project directory to see if it terminates automatically or still hangs there forever.
 - d) Try to find the last command in the log message shown in the console (e.g., `git push ... master:master`), and directly type in the same command in the command line to see if additional verification (e.g., adding a host to `known_host` list) is required.
- Q3 What if I find some bugs after it has been released? How can I resubmit my code?
- A3 You can look at the “Overview” tab in the Maven POM Editor for your `pom.xml` file. The version should be “0.0.2-SNAPSHOT”, which indicates a previous version has been generated. Now, you could redo this task (git-commit, git-push, run Maven goals) to release the version 0.0.2. We will evaluate your code based on the latest release.
- Q4 How could I check if my submission is successful?

A4 First of all, don't worry about it, even if by the deadline, we haven't received your submission, we will notify you, and help you check what the problem may be before you start Homework 1.

If you want to check your submission, you can go to <http://mu.lti.cs.cmu.edu:8081/nexus/index.html> back again, and log in with your password, go to **View/Repositories** → **Repositories** on the left, then click **Course Releases** on the right, unfold directories along the groupId path, and if you see your artifact at the end, then you've done!

3.6 Some commonly encountered problems

1. The version of Maven and Maven plugins do matter

We highly recommend using Maven 3.1.1.

In addition, according to tests, not all versions of Maven and Maven release plugins are compatible. Some test results:

```
Maven 3.2.3 + maven-release-plugin 2.5 = SUCCESS
Maven 3.1.1 + maven-release-plugin 2.2.1 =SUCCESS
Maven 3.2.3 + maven-release-plugin 2.2.1 = FAIL
```

If Maven is not compatible with the release plugin, you may see a message like this:

```
(default-cli) API incompatibility while executing
org.apache.maven.plugins:maven-release-plugin:2.2.1:prepare:
java.lang.NoSuchMethodError.
```

Additionally, David Klapper proposed to use Maven 2.5 as well as to force some additional plugins to 1.9.1. In his opinion, the second part is probably not strictly necessary for us i.e., as long as your PoM is in the repository root but just to be sure.

Your PoM file may contain the following:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-release-plugin</artifactId>
  <version>2.5</version>
  <executions>
    <execution>
      <id>default</id>
```

```
<goals>
  <goal>perform</goal>
</goals>
<configuration>
  <pomFileName>hw0-ANDREW_ID/pom.xml</pomFileName>
</configuration>
</execution>
</executions>
<dependencies>
  <dependency>
    <groupId>org.apache.maven.scm</groupId>
    <artifactId>maven-scm-api</artifactId>
    <version>1.9.1</version>
  </dependency>
  <dependency>
    <groupId>org.apache.maven.scm</groupId>
    <artifactId>maven-scm-provider-gitexe</artifactId>
    <version>1.9.1</version>
  </dependency>
</dependencies>
</plugin>
```

2. Success on course-snapshots but not course-releases

Q I succeeded to push to course-snapshots, but not to course-releases. More specifically, it showed "release success" in the console and showed up on the <http://mu.lti.cs.cmu.edu:8081/nexus/content/repositories/course-snapshots/edu/cmu/lti/11791/f14/hw0/>, but I failed to see my artifact in the remote repositories of Course Releases. (<http://mu.lti.cs.cmu.edu:8081/nexus/content/repositories/course-releases/edu/cmu/lti/11791/f14/hw0/>) What probably caused this?

If you cannot figure out the right combination of Maven plugins, you can bump into this issue. One hacky solution is as follows:

1. First follow the instructions from the hw0 guide.
2. Remove -SNAPSHOT from the version in pom.xml, commit and push to git, but do not run maven release.
3. Add -SNAPSHOT back to the version. Then run maven release:prepare release:perform. You may see a "tag already existed" error. Just delete the

previous tag from git first. For tag-deletion instructions see e.g. <http://nathanhoad.net/how-to-delete-a-remote-git-tag>.

3. Maven executable not found

Several people failed to release, because Maven wasn't configured properly. In this case, people see an error message like this:

```
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-release-plugin:2.2.1:prepare (default-cli) on project hw0-ANDREWID: Failed to invoke Maven build. Error configuring command-line. Reason: Maven executable not found at: /Users/ANDREWID/Documents/workspace/hw0-ANDREWID/EMBEDDED/bin/mvn -> [Help 1]
```

First, go to Maven/Preferences/Windows/Installations and check if you replaced the embedded Maven with an external Maven installation (Please, refer to Figure 3.37).

Second, when you create a maven build to execute goals “release:prepare release:perform”, there is an option to specify a Maven runtime. In our version of Eclipse, it was a dropdown menu/list at the bottom of the window, which appears when you create a new run configuration (of the type Maven build). So, if you have such an option, make sure you specify an external Maven rather than an embedded one.

There were cases, when these recommendations did not help. As a last resort, one can invoke Maven directly. This can be done either from the command line (which is straightforward), or even from the Eclipse (as suggested by LIU Xi). To do this, go to Run/External tools/External tools configurations and create a new external run. A window similar to that in Figure 3.42 will appear.

4. Cannot tag, because the tag already exists.

If you repeat an operation multiple times (due to errors), you may see a message:

```
Error: unable to tag scm. fatal: tag already exists.
```

Tagging is a labeling mechanism of Git, which is used by Maven to label specific releases. The name of the tag is built based on the artifact version that is specified in the Pom file. Due to previous unsuccessful operations, the tag has already been created. Thus, it cannot be recreated again (with the same name).

Two solutions are possible. A good solution is to delete the previously created tag, see e.g. <http://nathanhoad.net/how-to-delete-a-remote-git-tag>. Unfortunately, the Maven release plugin cannot do it (though should) on its own. A hacky and dirty solution is to change the version of the artifact in the Pom file so that it does not collide with any of the previously created version. Bad approach for people

TASK 3. CREATING YOUR FIRST MAVEN PROJECT, WRITING A PIECE OF CODE AND PERFORMING YOUR FIRST MAVEN RELEASE

40

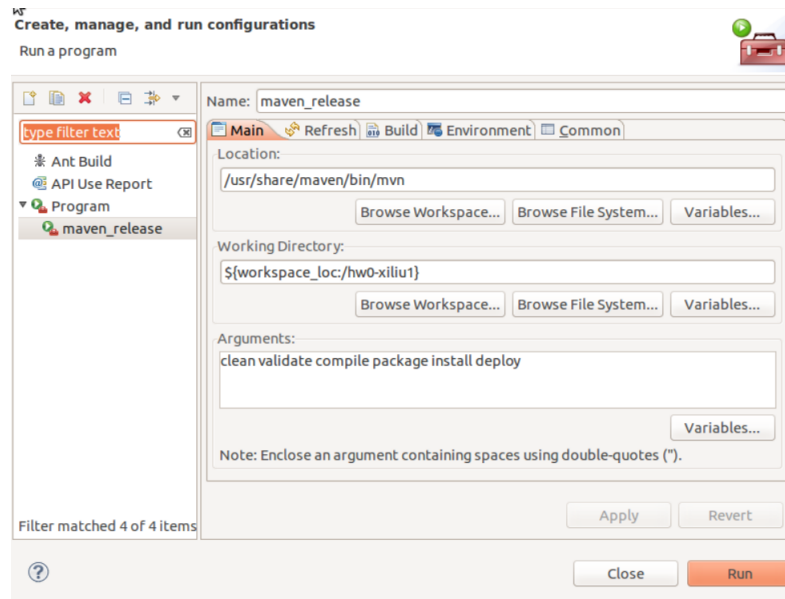


Figure 3.42: Invoking Maven as an external tool

working at Google, but, probably, good enough for you to finish the homework if you are running out of time.

5. Cannot tag scm/push to Git, because of the wrong authentication protocol

You may see an error:

```
fatal: cannot push gitxxxxxxxxxxxxxx, try use https://xxxxxxxxxx
```

As explained previously, one should not use *HTTPS* authentication for Maven. Please, check the scm-address in your Pom file. You should use the *SSH* authentication and the correct scm format should be:

```
scm:git:git@github.com:GITHUB_ID/hw0-ANDREW_ID.git
```

6. Cannot tag scm, because access is denied

If you see an error:

```
fatal: cannot push gitxxxxxxxxxxxxxx, access denied (publickey)
```

It is may caused by inappropriate SSH-key setting. Check or reset your SSH-key.

7. error: You don't have a SNAPSHOT project in the reactor projects list.

If you see the following error:

```
error: You don't have a SNAPSHOT project in the reactor projects list.
```

Check your pom.xml, <version> tag. Make sure you have -SNAPSHOT suffix link like this:

```
<version>0.0.1-SNAPSHOT</version>
```

8. Cannot push upstream

In some version of Eclipse (and OS), the option "PUSH TO UPSTREAM" is not available. One may be able to use **Remote**—>**Push** instead.

9. Error parsing Pom file

The XML can be malformed, even though visually it looks fine. One common error was encountered on Mac. Turns out, certain non-printable characters on Mac just never appear on screen: neither in Eclipse editor nor in Vi. These pesky characters can be detected using at least three approaches:

- Check how the file looks at GitHub ideally using your friend's computer (ideally it should have different locale settings);
- Clone your repository on one of the Linux machines (you may have access to ScS servers) and open the file in Vi.
- Use the command `hexdump -C pom.xml`.