CS108, Stanford
Winter 2011

Handout #35
Young

Subversion and Subclipse

Even relatively small software engineering projects involve thousands of files all being used and modified by dozens of team members. In order to control this chaos, modern software engineering teams use revision control systems to manage their files. A revision control system keeps track of changes to files, and allows management of different versions of the project.

For CS108 we recommend you use Subversion (SVN) in combination with Subclipse. Subversion is a popular revision control system, and Subclipse is a plug-in that you can add to your Eclipse software development environment to make using Subversion easy. Subversion can be run from the command line without Subclipse, but in this handout we will be assuming access via Eclipse and Subclipse wherever possible.

If you're more comfortable using another revision control system, feel free to use it, however, keep in mind that all members on your team must use the same system. In the past we've found that students who do not have a lot of UNIX experience or most comfortable using a system which can be accessed directly from Eclipse.

If you have trouble getting Subclipse installed and running, you can use an older revision control system called Concurrent Versions System (CVS) which already has an interface built into Eclipse. For more information on CVS see this handout from previous quarters of CS108:

http://www.stanford.edu/class/cs108/handouts092/S2SourceControlCVS.pdf

Getting Revision Control Working

You'll need to carry out two separate steps to get revision control up for your team. First, one and only one of your team members needs to create a shared storage location called a repository on the Stanford computers. Next each of the team members needs to get Subclipse installed into their copy of Eclipse.

Creating a Repository

Have one of your team members create a repository. You may create a repository in any location in your Leland directory structure. Go to the location in which you wish to create the repository and run the command:

```
> svnadmin create directory
```

where *directory* is the name of the repository directory you want to create. This will generate a new UNIX directory, along with some new files and subdirectories. If desired, a single SVN repository can be used for multiple projects. Alternatively you can create as many SVN repositories as you want on Leland.

Once you've created your repository. You'll need to change the file permissions on the repository directory so that your other team members can access it. You can do this with the fsr command.

```
> fsr setacl -dir directory -acl user all
```

where *directory* is the name of the repository directory and *user* is the SUNetID of your team member. Do this for each member of your team.

Repositories are accessed using a URL just like webpages and other Internet resources. For Stanford Subversion access you'll need to use the svn+ssh URL schema. Get the full name of the directory, add on svn+ssh://vine.stanford.edu/ and you'll have the URL for your repository. For example, this is the URL for a repository in my Leland directory space:

```
svn+ssh://vine.stanford.edu/afs/ir.stanford.edu/users/p/s/psyoung/repos
```

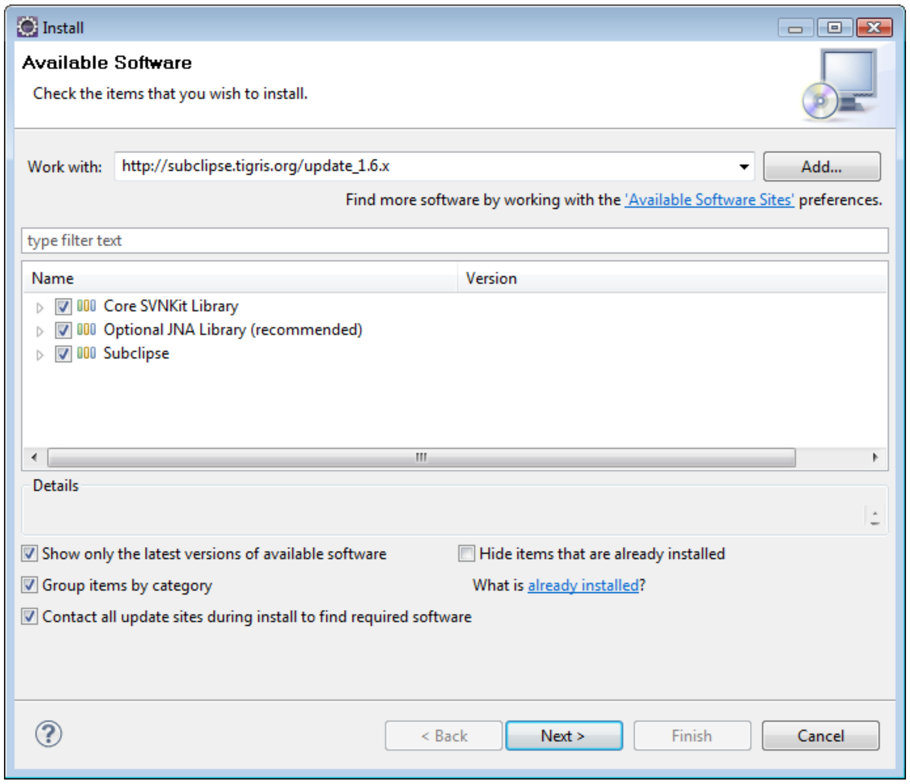
Mail out this URL to all your team members.

Installing Subclipse into Eclipse

All team members should install Subclipse into their Eclipse Enterprise Edition software development environments. To do this, go ahead and startup your copy of Eclipse EE. Go to the "Help" menu and select "Install New Software …" In the dialog box which comes up add the site:

```
http://subclipse.tigris.org/update_1.6.x
```

Click on "Add" and you should see three libraries show up:


Install
Available Software
Check the items that you wish to install.
Add...
Find more software by working with the 'Available Software Sites' preferences.
type filter text
Name
Version
Details
< Back
Next >
Finish
Cancel

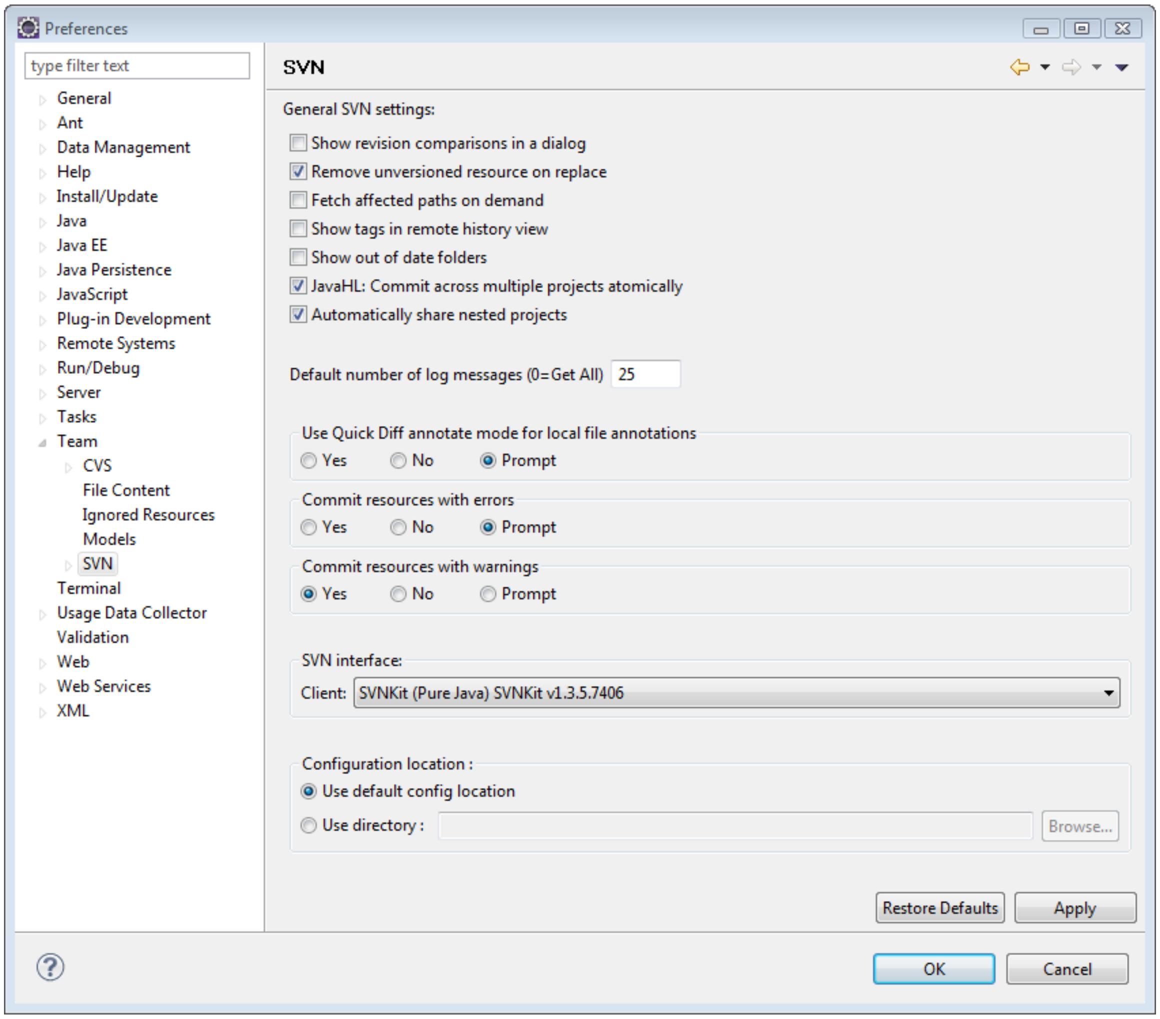
Work with: http://subclipse.tigris.org/update_1.6.x

- [✓] Core SVNKit Library
- [✓] Optional JNA Library (recommended)
- [✓] Subclipse

[✓] Show only the latest versions of available software
[ ] Hide items that are already installed
[✓] Group items by category
What is already installed?
[✓] Contact all update sites during install to find required software

Go ahead and select all three and install them.

Go to "Window > Preferences" and switch to the "Team > SVN" tab when the preferences window comes up. Scroll down to SVN Interface: Client and switch it to the "SVNKit" which you just installed along with Subclipse:



SVN

General SVN settings:

- [ ] Show revision comparisons in a dialog
- [✓] Remove unversioned resource on replace
- [ ] Fetch affected paths on demand
- [ ] Show tags in remote history view
- [ ] Show out of date folders
- [✓] JavaHL: Commit across multiple projects atomically
- [✓] Automatically share nested projects

Default number of log messages (0=Get All) 25

Use Quick Diff annotate mode for local file annotations: Yes No [✓ Prompt]

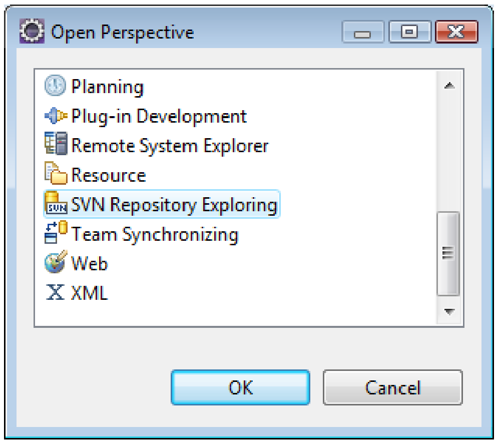
Commit resources with errors: Yes No [✓ Prompt]

Commit resources with warnings: [✓ Yes] No Prompt

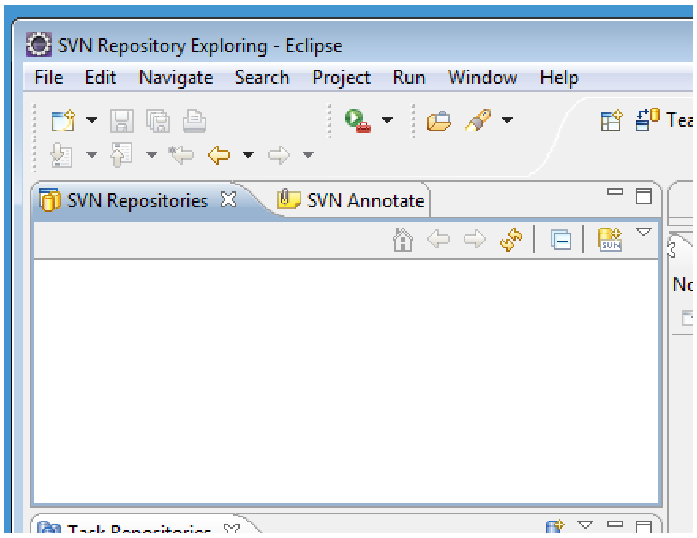
SVN interface:
Client: SVNKit (Pure Java) SVNKit v1.3.5.7406

Configuration location :
[✓ Use default config location]
Use directory : Browse...

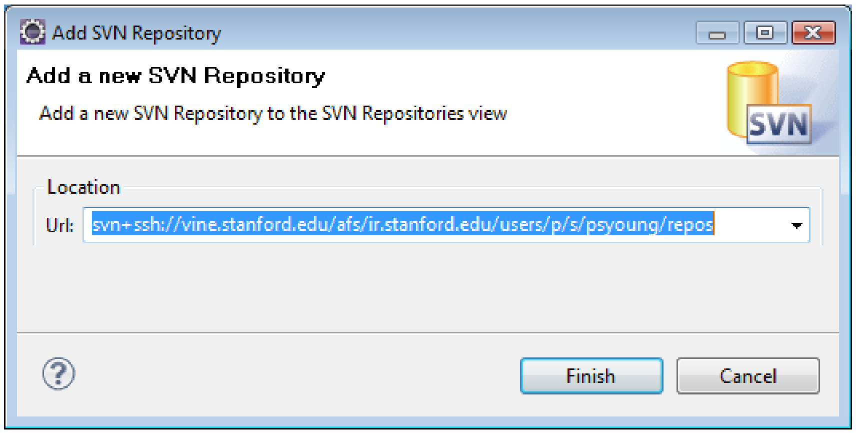
We're now ready to use Subclipse. Go to "Window > Open Perspective > Other …" choose "SVN Repository Exploring"



We will now connect to the repository our team mate created on the Stanford Leland systems. In the top-left corner of the SVN Repository Exploring view you should see the "SVN Repositories" tab.


SVN Repository Exploring - Eclipse
File Edit Navigate Search Project Run Window Help
SVN Repositories
SVN Annotate

To connect to the repository either click on the "Add SVN Repository" button or right-mouse click in the window and choose "New > Repository Location …" Enter the svn+ssh: URL that the team mate who created the repository has sent you:



Add a new SVN Repository

Add a new SVN Repository to the SVN Repositories view

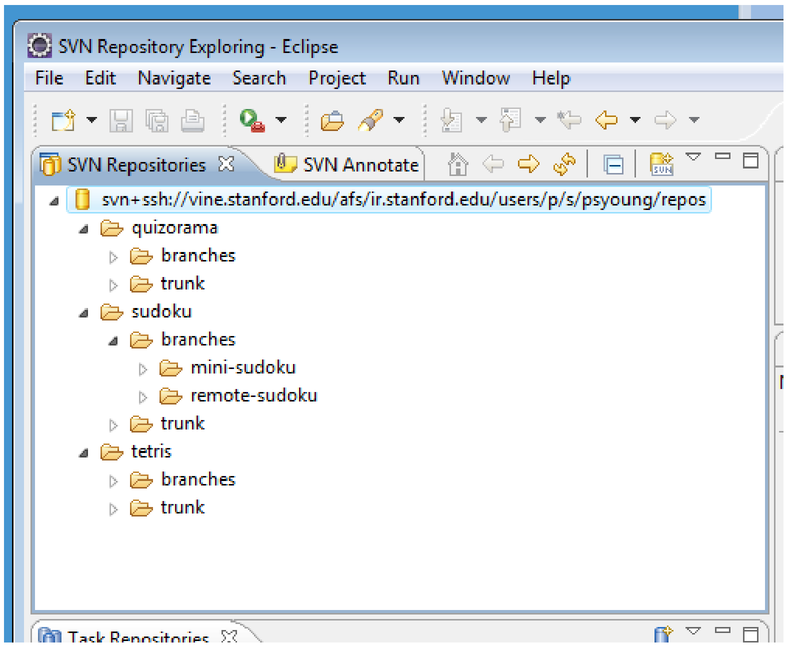
Url: svn+ssh://vine.stanford.edu/afs/ir.stanford.edu/users/p/s/psyoung/repos

Your repository should now show up in the "SVN Repository" tab.

Structuring Your Repository (Optional)

If you want, you can go ahead and start adding your Quiz-O-Rama files directly into your repository. However, adding some folders to your repository before getting your quiz website files checked in may prove useful.

Have one of your team members add folders to the SVN repository. To do this right click on the repository and choose “New > New remote folder”. An SVN repository might have folders for several projects at the top level. In addition, traditionally each project folder will have a trunk folder and a branches folder. Here is an example of the structure of an SVN repository:

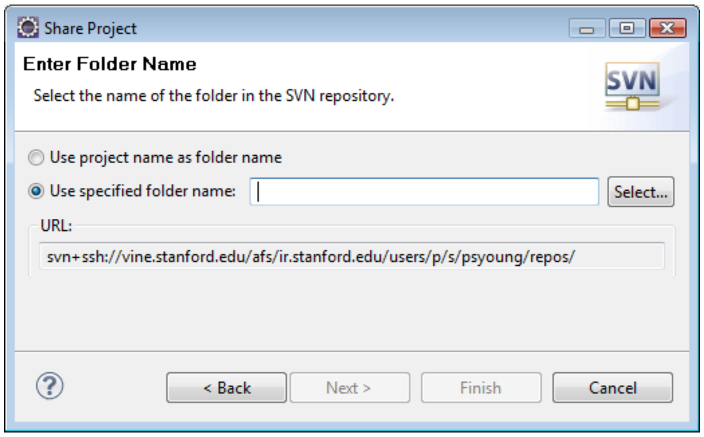

SVN Repository Exploring - Eclipse
File Edit Navigate Search Project Run Window Help
SVN Repositories
SVN Annotate
svn+ssh://vine.stanford.edu/afs/ir.stanford.edu/users/p/s/psyoung/repos
quizorama
branches
trunk
sudoku
branches
mini-sudoku
remote-sudoku
trunk
tetris
branches
trunk
Task Repositories

The trunk folder is used for your main line of development. Files associated with your project go directly into it. If you decide for some reason to create a new development branch, you would create a folder for that branch in the branches folder. A branch starts off as a copy of the original development stored in the trunk folder, but it evolves independently after it branches off. In the screenshot above, you can see that I have created two separate versions of Sudoku, both based on the original code, but with additional specialized code added in. Once created these branches would contain their own separate copies of the code.

Be very cautious creating branches, trying to keep multiple development branches can get messy. My general assumption is that you will not create multiple branches for this project. I mention them here so you are used to seeing a traditional SVN directory structure with both trunk and branches directories.

Adding a Project to SVN Repository

There are several ways to add a project to our SVN repository. The easiest thing to do is to have one of your team members add their copy of the project to the repository. To do this switch to the traditional Java perspective in Eclipse. Right click on the project folder in Eclipse’s Project Explorer and Choose “Team > Share Project …” When the dialog shows up choose SVN for the repository type. Tell it to use your existing repository. When it asks which folder to use, if you performed the steps in the “Structure Your Repository” section above, click on the "Select" button and manually choose the trunk directory that you created for the project. Otherwise, let it "Use project name as folder name".

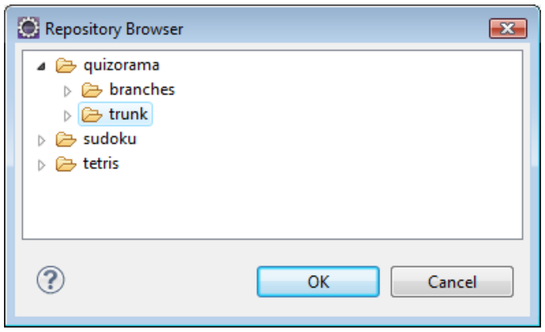

Share Project

Enter Folder Name

Select the name of the folder in the SVN repository.

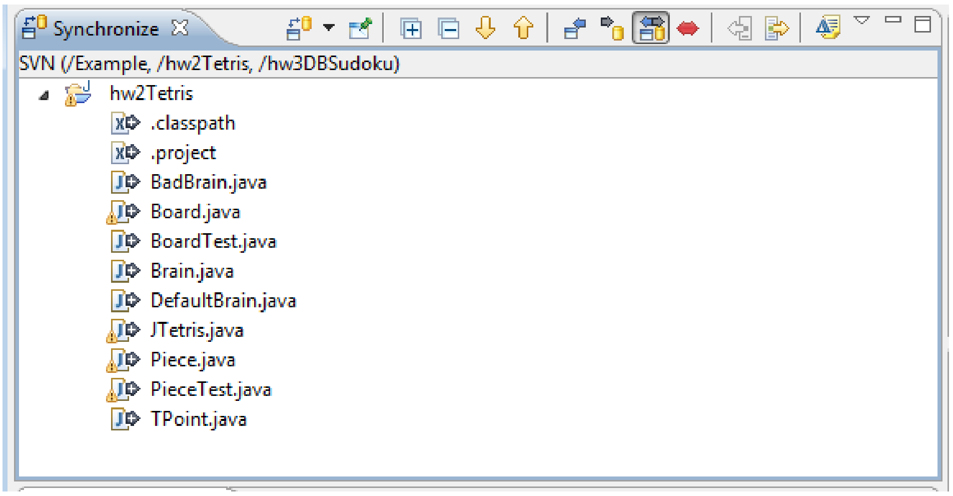
Use project name as folder name [✓ Use specified folder name:]

URL: svn+ssh://vine.stanford.edu/afs/ir.stanford.edu/users/p/s/psyoung/repos/


Repository Browser

- quizorama
  - branches
  - trunk
- sudoku
- tetris

You will then be moved to the synchronize view where you will see something like this (files shown for my checking in Tetris):



SVN (/Example, /hw2Tetris, /hw3DBSudoku)

- hw2Tetris
  - .classpath
  - .project
  - BadBrain.java
  - Board.java
  - BoardTest.java
  - Brain.java
  - DefaultBrain.java
  - JTetris.java
  - Piece.java
  - PieceTest.java
  - TPoint.java

This window is telling you what the differences are between the copy you are checking in and what's already in the repository. As all your files are new (i.e., there are not copies already in the directory) each of your files has a "+" and an arrow icon. The icon would be different if there was an existing copy in the database. We want all our files to get copied over, so just check on the "Commit all outgoing changes" button: . You can provide a brief comment in the commit window if you want. "Initial import." is a traditional message for the first time a project is added to the repository. We'll take a closer look to exactly how this window works in a moment.

You files are now in the repository. You can continue to work with your own copy in Eclipse, but your team mates can now check out copies for themselves. <u>Do make sure your</u>

teammates check out copies rather than using copies that they may have had before you placed the project under revision control.

Checking Out an Existing Project

If your team mate has checked in the initial version project, you'll want to get a copy of the project from the repository. To do this switch to the "SVN Repository Exploring" view. Find the directory corresponding to the project. This is either the folder with the project name or the "trunk" folder if you are using the fancier "trunk/branch" folder organization. Choose "Checkout".

Congratulations your project is now under revision control.

Subversion Revision Control

Now that you've got Subversion and Subclipse setup, let's learn the basics of Subversion revision control.

Copy-Modify-Merge

Subversion uses a revision control paradigm which is sometimes referred to as copy-modify-merge. The basic idea is that you receive a copy of the project from the repository. You can make whatever modifications you want to the project. When you're done you *commit* your changes to the database. Sometimes when you try to commit changes to the database you'll discover that someone else has already made changes to some of the files that you are committing. If this is the case, you'll be notified that there is a conflict. You will then have to *merge* your changes with the changes found in the database.

Working Copy

One of the most important concepts is the idea of a *working copy*. When you checkout a copy of the project from Subversion you get a working copy. You can do whatever you want with this working copy. It is entirely separated from what other team members are doing. You can modify it, you can even toss it out and get another fresh working copy.

Subversion Operations

There are a variety of different Subversion operations you can perform. The most important are:

Checkout – In order to get a copy of an existing project you perform a checkout. This creates a working copy on your computer that you can modify to your heart's content. To checkout a new working copy, switch to the "SVN Repository Exploring View", right-click on the folder that you want and select "checkout".

Update – Sometimes someone will make some modifications to the project and you'll want to have those changes propagated to your own files. You can do this by performing an update operation. You can update either an individual file or your entire project. To update, right-click the folder or file that you want to get an updated copy of and either select “Team > Update to HEAD” or “Team > Update to Version”. Update to HEAD gets the most recent changes to the file or folder. If there is a specific version you want, other than the latest version, you can use “Update to Version.”

Updating the entire project will make changes to all your files including ones you are currently editing. If Subversion does not believe that changes in the version in the repository do not conflict with changes that you’ve made, it will go ahead and integrate those changes into your file.

Commit – When you are ready to share your changes with the rest of your team, you check your changes in by performing a commit. To commit, right-mouse click on the folder or file that you want to commit and select “Team > Commit …” There are several important things to know about performing a commit.

Your commit has absolutely no impact on anyone else’s working copies. They won’t see your changes unless they specifically perform an update (perhaps in response to an e-mail from you, suggesting they get the new copy that you’ve just committed).

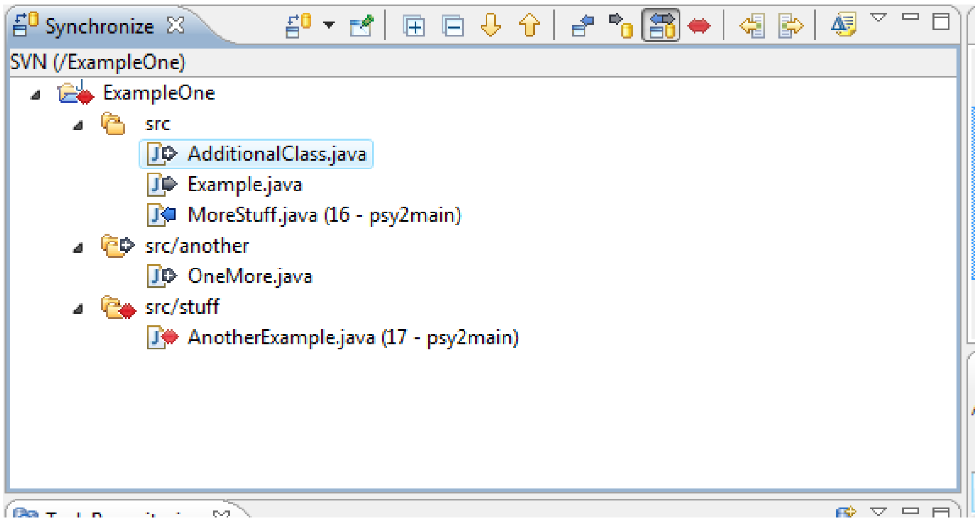
Your commit may cause a conflict if you are committing a file that someone has changed on the repository since either your original checkout or since your last update. When this happens you’ll have to resolve the conflict. We’ll take a look at how to do this momentarily.

Subversion Revision Numbers

While Subversion does keep track of changes to individual files, Subversion revision numbers refer to changes to the entire project. Suppose our project consists of three files. The current revision number for the project is 5. I make changes to one of those files and check in my changes. The new revision number for the entire project is 6. While only one of those files has changed, they all now correspond to revision 6.

Synchronization

When you’re ready to check in changes, you should check and see how your working copy of the project compares to what’s on the repository. You can do this by selecting “Team > Synchronize with Repository”. This will take us to the Synchronize view.



```
SVN (/ExampleOne)
  ExampleOne
    src
      AdditionalClass.java
      Example.java
      MoreStuff.java (16 - psy2main)
    src/another
      OneMore.java
    src/stuff
      AnotherExample.java (17 - psy2main)
```

This view shows us which files are the same, which are different, where we have created new files not on the repository, and where the database contains files we do not yet have copies of.

In the screenshot above, you can see that we have added a new package (src/another) and two new Java files – AddtionalClass.java and OneMore.java. These are indicated by the green arrow with a + sign on it. The Example.java class contains code that the copy on the repository does not have (indicated by the icon of a green arrow pointed toward the right). The file MoreStuff.java has extra material from revision 16 modified by the user psy2main which our copy does not contain.

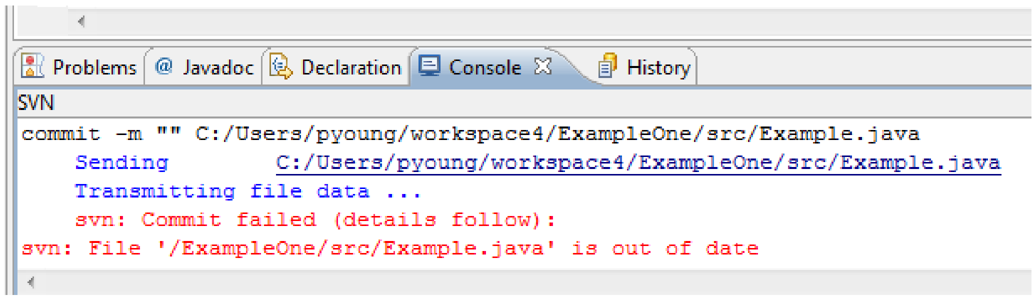
All of the changes that we've seen so far can automatically be resolved by doing an Update with Subversion. However, there is one more issue which is more problematic. The red diamond on AnotherExample.java indicates that there is a conflict between the version on the repository and our version. If we right-click on a conflict file and choose "Team > Compare with", this takes us to a conflict resolution screen. We'll look at this in the next section.

One very important thing to note is that running Synchronize by itself does not actually change any of the file in your working copy or on the repository. You can take advantage of the Synchronize screen to update or commit, but unless you do so, no changes will occur.

Conflict Resolution

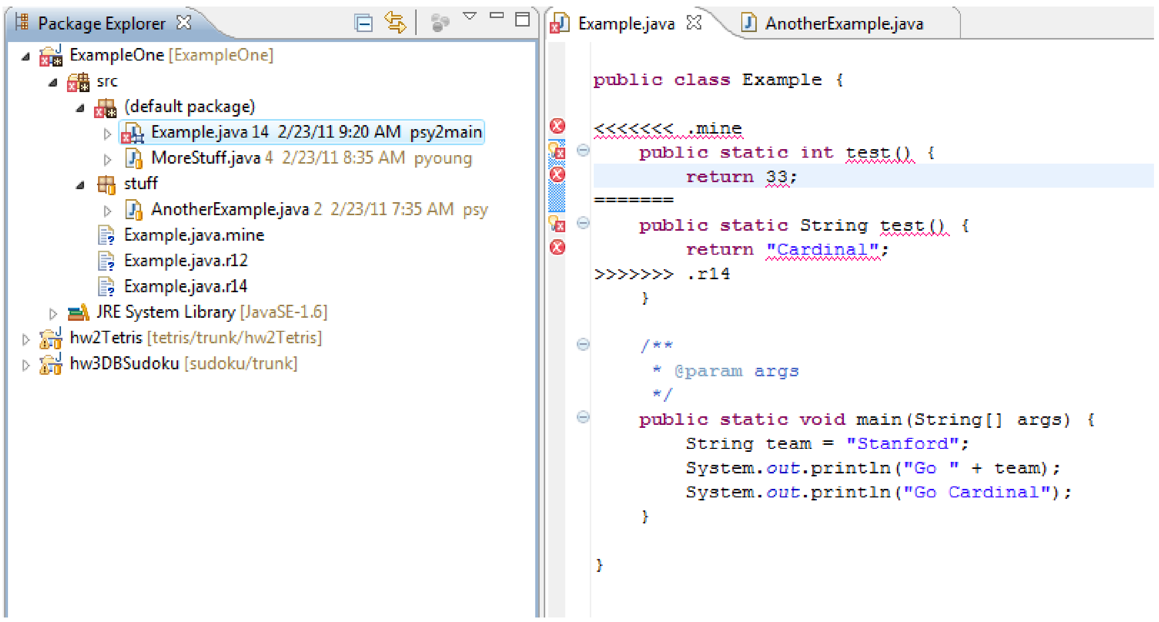
Conflict can occur from several sources. As we saw in the previous example, when we run Synchronize, we may see that there are conflicts between our file and one on the repository. Alternatively, if I try to commit a project, that someone else has made modifications a conflict may occur.

When I try to commit my changes if there is a conflict, I'll get an error warning in the Console pane (bottom panel in Eclipse). This error message will tell me that the commit has failed. <u>Pay attention to the panel, otherwise you may think your commit has been successful, the error message is not very intrusive and can easily be missed.</u>



```
commit -m "" C:/Users/pyoung/workspace4/ExampleOne/src/Example.java
    Sending        C:/Users/pyoung/workspace4/ExampleOne/src/Example.java
    Transmitting file data ...
    svn: Commit failed (details follow):
svn: File '/ExampleOne/src/Example.java' is out of date
```

This means that I need to do an update, before I can commit. If I do an update, and Subversion decides that there's a conflict between the two versions, it will notify me in the Console Window, it will display conflict icons in the package explorer, will add special conflict files, and will mark sections of the file itself to show the conflict. Here we see that Example.java is in conflict.

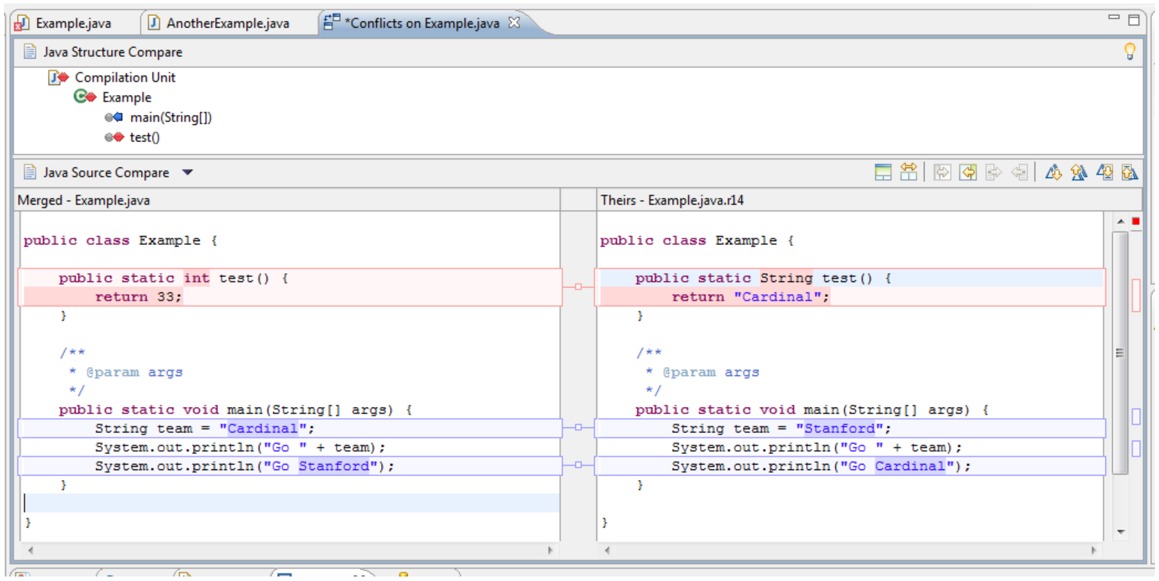


```
ExampleOne [ExampleOne]
  src
    (default package)
      Example.java 14  2/23/11 9:20 AM  psy2main
      MoreStuff.java 4  2/23/11 8:35 AM  pyoung
    stuff
      AnotherExample.java 2  2/23/11 7:35 AM  psy
    Example.java.mine
    Example.java.r12
    Example.java.r14
  JRE System Library [JavaSE-1.6]
hw2Tetris [tetris/trunk/hw2Tetris]
hw3DBSudoku [sudoku/trunk]
```

```
public class Example {

<<<<<<< .mine
    public static int test() {
        return 33;
=======
    public static String test() {
        return "Cardinal";
>>>>>>> .r14
    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        String team = "Stanford";
        System.out.println("Go " + team);
        System.out.println("Go Cardinal");
    }

}
```

The Example.java file has <<<<, ====, and >>>> markers added to indicate the sections which are in conflict. The Example.java.mine, Example.java.r12, and Example.java.r14 files are temporary files created to track the differences between the versions.

At this point, I have several possible methods I can use to resolve the conflict. I can edit the Example.java file directly, which might be simplest if there are only a few differences, or if there are many differences, I may want to use the special Edit Conflict view. You can do that by right-clicking on the file and choosing "Team > Edit Conflicts". Here's what that view looks like:



Merged - Example.java

```
public class Example {

    public static int test() {
        return 33;
    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        String team = "Cardinal";
        System.out.println("Go " + team);
        System.out.println("Go Stanford");
    }

}
```

Theirs - Example.java.r14

```
public class Example {

    public static String test() {
        return "Cardinal";
    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        String team = "Stanford";
        System.out.println("Go " + team);
        System.out.println("Go Cardinal");
    }

}
```

Once I’ve resolved the conflict either directly in the regular text editor by removing the <<<<, ====, and >>>> conflict markers or in the Conflict Editor, right-mouse click on the file and select “Team > Mark Resolved.” This will remove the .mine and .r revision files and notify Subclipse that the file can now be committed.