

## 1 Objectives

- Learn to use Subversion
- Implement an ADT while using Subversion

In this lab, you learn about the version control tool called Subversion and you will implement a Java class given an interface. *You will be using Subversion on all of your labs and projects this semester.*

## 2 What is Subversion?

Subversion is a popular version control system used in the software industry. Using it, you can record the history of your source files.

For example, you might modify your software and introduce a bug, but you might not detect the bug until a long time after you make the modification. With Subversion, you can easily retrieve old versions to see exactly which change caused the bug. This can sometimes be a big help.

You could of course save every version of every file you have ever created. This would, however, waste an enormous amount of disk space and keeping track of all those versions would present a huge management problem. Subversion stores all the versions of a file in a single file in a clever way that only stores the *differences* between versions.

Subversion also helps you if you are part of a group of people working on the same project. It is all too easy to overwrite each others' changes unless you are extremely careful. Some editors, like emacs, try to make sure that the same file is never modified by two people at the same time. Unfortunately, if someone is using another editor, that safeguard might not work. Subversion solves this problem by isolating the different developers from each other. Developers work in their own directory, and Subversion merges the work when each developer is done.

Subversion allows the developer to keep old versions of files (usually source code); keep a log of who, when, and why changes occurred; compare the differences between versions; and reconstruct old versions. Subversion also helps to manage releases and to control the concurrent editing of source files among multiple developers.

Subversion does not just operate on one file at a time or one directory at a time, but operates on hierarchical collections of directories consisting of version controlled files.

Subversion is great as part of a disaster plan to allow a development team to recover from a major blunder or error.

### 3 Getting Started With Subversion

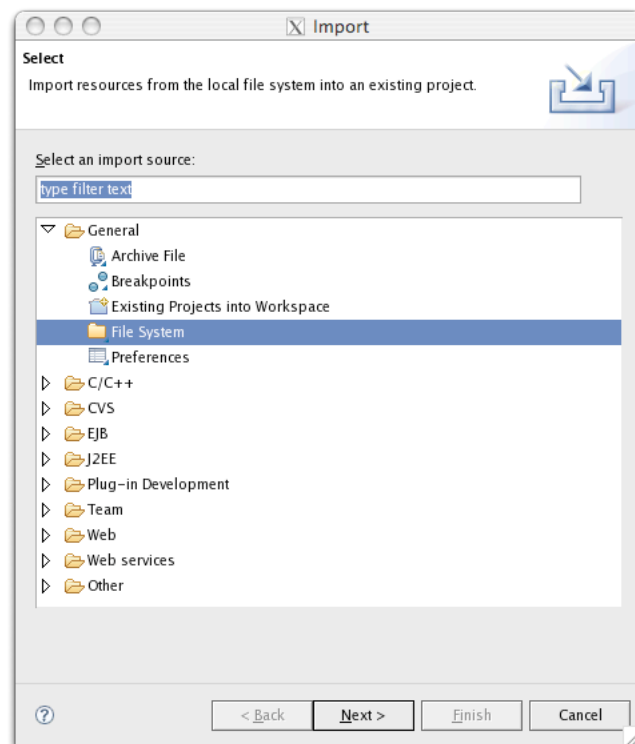
This lab will walk you through the steps of creating a lab *project* with Subversion. Read each step and follow the instructions.

#### 3.1 Create an Eclipse Project

Start Eclipse and select **File** → **New** → **Java Project** (or select the **New Java Project** icon from the toolbar. Use `lab02` as the project name, then click on the **Finish** button.

#### 3.2 Import Files for Today's Lab

Click on the disclosure triangle next to `lab02` and you will see a folder called `src`. This is a *source* folder where your Java source files (the `.java` files) will go. Click *once* on `src` so it is highlighted, then select **File** → **Import...** In the dialog that appears, select **General** → **File System**.



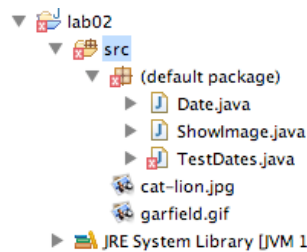
Click on the **Next** button.

To the right of *From directory*, use the **Browse** button to navigate to the files for today's lab. **Hint:** Once in the file browser, click on the **accounts** button and then double click on COURSES to get to the csci204 directory. Continue to double click on directory names until you get to the directory for today's lab:

```
~csci204/2009-fall/student/labs/lab02
```

When lab02 is highlighted (don't open it!), click **OK**. In the next dialog, check the box next to lab02. Click the **Finish** button.

The files for today's lab are now in Eclipse. You should be able to expand the view of your lab02 project to see the following structure.

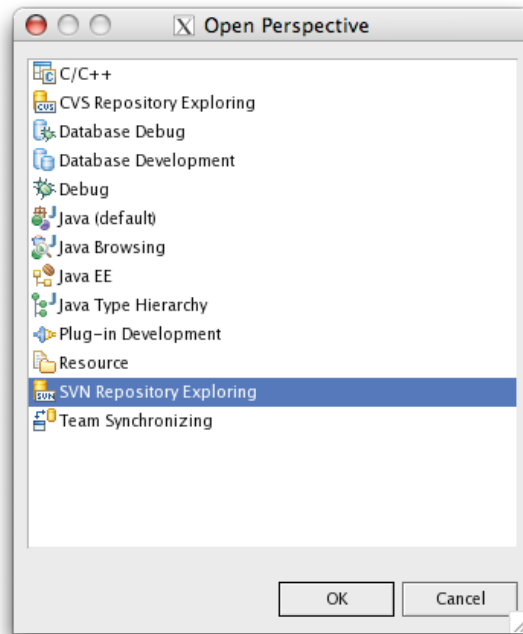


*If this is not what you see, ask for help now!* The red marker on **TestDates.java** indicates that there is an error in the file. This is OK. Double click on a Java file to open it. Don't make any changes yet!

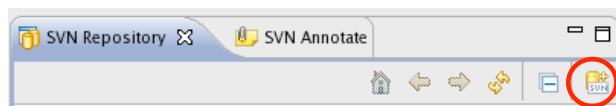
### 3.3 Tell Subversion About Your Lab Repository

Bucknell has a Subversion repository that you will be using for your labs and projects. You will now add today's lab project to your Subversion repository. You will begin by informing Subversion where your lab repository resides. You will need to do this just once this semester.

Open the *Subversion perspective* in Eclipse. This is a special mode of Eclipse in which it knows more about Subversion. Do this by selecting **Window** → **Open Perspective** → **Other...** and select **SVN Repository Exploring**. Click **OK**.



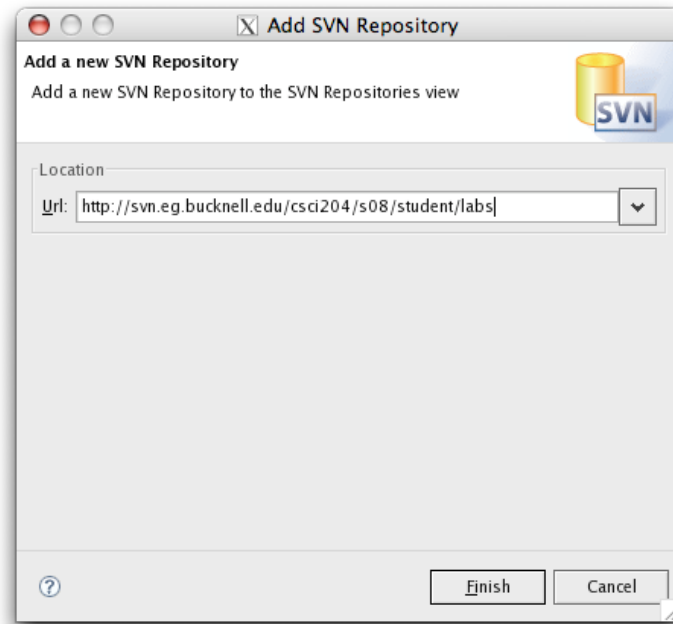
Click on the **Add SVN Repository** button



Enter the location of your repository (note the “s” in https)

`https://svn.eg.bucknell.edu/csci204/f09/student/labs`

replacing student with your login ID. Press the **Finish** button.



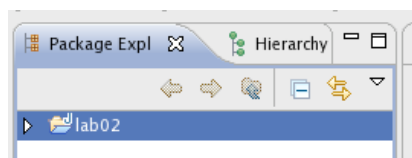
You will be prompted to enter your Username and Password for the repository. Enter them, check the **Save Password:** checkbox and press the **OK** button. (Note that if you do not want to save your password, that is fine: you will simply need to enter it every time you access the repository.) When prompted about accepting certificates, choose **Accept Certificate Permanently**.

If everything worked properly, your Subversion repository should have appeared in your Eclipse window on the left.

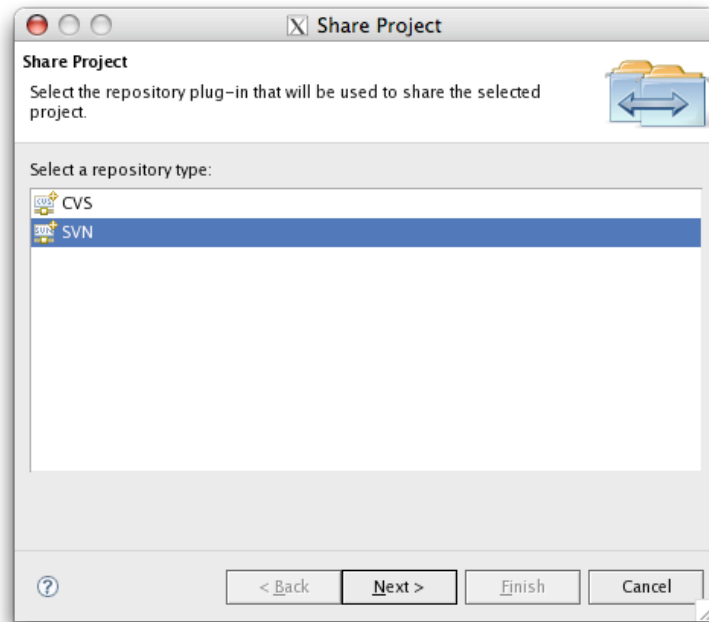
Switch back to the Java (default) perspective by selecting **Window** → **Open Perspective** → **Other...**

### 3.4 Adding Your Lab Project Folders to the Repository

Now you are ready to add your lab02 project to the repository. Click once on the lab name so it is highlighted.



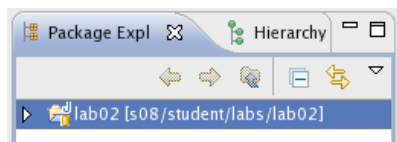
Right click on the lab and select **Team** → **Share Project...** Click on SVN and press **Next**.



In the next dialog, your SVN repository will be highlighted. Click **Next**.

The next dialog asks you for a folder name. The default is to use the project name which you should accept. Press the **Next** button.

The final dialog asks for a comment. Whenever you add something to the repository, you should add a comment that explains why you are adding it. In this case, Subversion recognizes this as your *initial import* and the comment already says that. Click the **Finish** button. In the next dialog, click the **OK** button to complete the import. Note that Eclipse has added the path to your repository after the lab name.



Open a webpage to <https://svn.eg.bucknell.edu/csci204/f09> and click on your username and labs. You should see a folder for this lab. Click to go into the folder. Notice that there are **no** files in it. You have told SVN that this lab will eventually go in that folder but you have not done the last step yet.

### 3.5 Committing Your Project

When you have made a significant change in one or more files, you will want to tell Subversion about your changes so other team members can see your changes.

Edit one of the files. For this exercise, just add a comment to `Date.java` and save your change. You should now see that the icon for `Data.java` has been marked with a black and white star to indicate that there is a difference with the repository version of the file.

Normally you will commit changes to the repository when you have just completed a portion of your project and you want to take a snapshot. You will commit your project to the repository (take a snapshot) now, even though you haven't done anything significant. The idea is to show you how to commit.

Tell Subversion about your change by right clicking on the project and selecting **Team** → **Commit...** Eclipse will present you with a dialog showing you which files have changed since the last commit. Enter a comment in the box provided and press the **OK** button.

Subversion will update the revision number next to the file name.

Again, open a webpage to <https://svn.eg.bucknell.edu/csci204/f09> and click on your username and labs. This time when you go into the folder for today's lab, **you should see files and folders**. Your actual lab files are in the src folder. Double-check that you see them there.

### 3.6 Updating Project Files

Subversion will keep track of all the changes committed by your team. If you want to update your files to include any changes that have been made, right click on the project and select **Team** → **Update**. Try this now.

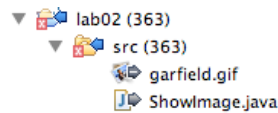
You have no partner who might have changed the files so no changes should have been recorded. Subversion will give you a message telling you the current revision number.

### 3.7 Synchronizing Project Files

Edit the `ShowImage.java` file, by adding a comment, and remove the file `garfield.gif`. (Right click and select **Delete**.)

`ShowImage.java` is now newer than any committed files. Its icon has changed to indicate that the latest version is not in the repository. You don't want to lose all your valuable work! One of the image files is missing. Oh no!

Right click on the lab02 directory and select **Team** → **Synchronize with Repository**. Confirm that you want to switch to the Team Synchronizing perspective by clicking **Yes**.



Files marked with a right arrow are *outgoing* files. That is, if you were to commit to the repository, new versions of these files would go out to the repository. Notice that `garfield.gif` is also marked with a minus sign indicating it would be deleted.

Right click on `garfield.gif` and select **Revert...** You will get a confirmation dialog. Click **OK** and the file will be restored. `garfield.gif` has disappeared from the list of changes since you have restored it.

If you double click on `ShowImage.java`, this will open up the *Source Comparison* view. In particular, you should notice that Eclipse has highlighted your comment as a change between the `ShowImage.java` in your workspace and the one currently held by the Subversion repository. If you have multiple changes, you can use the up and down arrows in the view to visit each difference. You can even choose to import and export only select changes if you wish.

You can experiment more with the *Team Synchronizing* perspective at another time. For now, return to the Java perspective by clicking on the double right arrow in the upper right of your screen and selecting **Java**.



### 3.8 Add Files to a Project

For this lab, you need to create a new class `Dates` and add it to the project. Your `Dates` class will implement the `Date` interface. Create the new class by selecting **File** → **New** → **Class** (or select the **New Java Class** icon in the toolbar). Enter `Dates` as the class name.

Press the **Add...** button to add an interface. The interface you are implementing is `Date`. Check the **Generate comments** box and press **Finish**. Eclipse will open your new file. Note that it has created stubs and comments for all your methods!

Eclipse has indicated that `Dates.java` is not part of version control by placing a small question mark on the file icon. To add this new file to version control, right click on `Dates.java` and select **Team** → **Add to Version Control**. Now a plus sign appears on the icon but there's no version number yet because it hasn't been committed.

### 3.9 Tracking Your Changes

Subversion keeps track of all the changes made and the date on which a file is changed or added in your project. You can get all this information by right clicking on the project and selecting **Team** → **Show History**. The revision history appears at the problem.

### 3.10 Discovering What Has Changed

One of the advantages of Subversion is that programmers may retrieve earlier versions of a particular file or group of files. The basic idea is that Subversion keeps track of changes you made over time as long as you commit all the changes to the program. At a later time, you may retrieve an earlier version of the program. Let's practice how to do this.

If you haven't done so, commit all your changes now by right clicking on the project and selecting **Team** → **Commit**. Add another comment to `ShowImages.java` and commit to the repository again.

Suppose that you have made some changes to a file and your project no longer works. Unfortunately, you can't remember all of the changes that you made. This is a good job for Subversion! It will tell you what has changed. To find out what has changed in `ShowImages.java` since the last version, right click on the file name and select **Compare With** → **Revision...** Eclipse will present you with the versions with which you can compare. Double click on one of them to open the *Source Comparison* view like the one you saw in the *Team Synchronizing* perspective.

### 3.11 Further Subversion Information

Using SVN and Eclipse, you can access your files anywhere in the world from Windows, Linux, or Mac as long as you have an internet connection.

*Version Control with Subversion* <<http://svnbook.red-bean.com/>> is a *free* book about Subversion that is available on the Internet. It is well written and it explains all of the features of Subversion.

The Subversion FAQ <<http://subversion.tigris.org/faq.html>> is another good source of information. Visit the Subversion home page <<http://subversion.tigris.org/>> if you want to find out about installing Subversion on your personal computer. You can also ask your professor for help.

## 4 Completing Your Project Using Subversion

You have already learned the basic ideas of classes. A *class* typically includes a set of instance variables and a set of methods. The instance variables and the methods together formulate a *data type*. If the user of the data type doesn't have to know the details of the class implementation in order to use the data type, the data type is called an *abstract data type* or ADT. An ADT has the advantage of hiding unnecessary details from the user so the user can concentrate on *what* the ADT does, not *how* the ADT does things. Examples of ADTs are everywhere, in our daily life or in programming. For example, we don't have to know the details of how a TV set works in order to use it; we don't have to know how an engine works in order to drive a car. In programming we don't have to know how exactly `System.out.println()` works in order to use it.

In Java we can use the `interface` construct to declare the interface to an ADT. The implementation can be done in a class that *implements* the interface. The methods of a class can be public or private. The public methods are typically the *interface* to the outside world. The private instance variables hold the information that the designer of the class would like to hide from the public. They usually involve details that are not important for using the data type.

Your task is create a `Dates` (note the "s" on the end) class to implement all the member functions of the `Date` interface so that the driver in `TestDate.java` will work. Read the interface class `Date.java` for some hints. Don't spend too much time worrying about how to check inputs for valid data. The focus of this lab is on making the methods work as specified. Do only as much error checking as you need to.

Commit your project to the repository each time you have made significant changes. That is, if you just made a significant step forward in your work, and you want to take a snapshot of the current state of your project, it's time to commit. Remember to fill out the comments.

### 4.1 Some Details

- The `Dates` class has three private instance variables, `day`, `month`, and `year`.
- The default constructor sets the date to January 1, 1970.
- The alternate constructor sets the date as specified. The parameters are assumed in the order of month, day, and year. You do not need to detect errors in this lab. You won't see input, for example, trying to set the date to September 33.

- The method `setDate()` job similar to the alternate constructor.
- The method `displayShort()` displays the date on the screen in the form `mm/dd/yy`, e.g., `02/24/05` for February 24, 2005.
- The method `displayVerbose()` displays the date on the screen in a long form, e.g., `February 24, 2005`.
- The method `nextDay()` increments the day by one. Note that you have to pay special attention when the day reaches the end of a month. Make sure your program knows how many days there are in the current month when updating (e.g., there are 30 days in September, there are 28 days in February in a non-leap year and 29 days in a leap year).
- The method `isLeapYear()` determines if the current year is a leap year. A leap year is a year that can be divided by 4 evenly but not by 100, or a year that can be divided by 400 evenly.

## 5 What is Due

When you are finished with the lab, make sure you commit the final version to the repository. The TA will check the lab by retrieving it from the repository. We won't be checking the file from the Subversion portion of the lab, just the `Dates` class. Make sure you have Javadoc comments for each method.