Eclipse and Subclipse Lab

CS 245

Oct. 6, 2010

Basics

In the following, replace `[Name]` with your name. For example, `[Name]Project` becomes `TomProject`.

1. Login to a Windows workstation.
2. Start Eclipse (Once the full window is displayed, Eclipse will still need a bit of time to initialize before it becomes fully responsive.).
3. From the Java Perspective, create a new Java project, named `[Name]Project`.
4. Within the project, create the class `Colors` within the package `myPackage`.
5. From the SVN Repository Exploring Perspective, add the `https://phoenix.goucher.edu/svn/sample` repository. There will be a `Practice` directory within the repository. Open `Practice` and double-click `Colors.java` within `Practice` to open it. Copy the source code, pasting it into the file containing *your* Colors.java class and replacing everything already in the file.

   (You don't ordinarily use Subversion this way. What you're doing here is creating a project which you'll soon share/checkin to the repository.)
6. Just to make sure everything's working as it's supposed to, back in the Java Perspective, run the applet.
7. Using the `Team` menu (right-click the project folder in the Package Explorer), share your project out to the repository, into the directory `[Name]Project`. (Under Windows 7 in the labs, follow this with an immediate commit.) This is how you initially share new work with your project team.

   Confirm that the project is now in the repository (you may have to click Refresh View for the changes to appear).
8. Using the `Team` menu, disconnect your project from the repository, deleting the meta-data. (This isn't something you would ordinarily do, but is necessary for the next step, in which you'll practice the initial checkout of an existing project from Subversion.)

   Completely delete the *local* copy of your project. (Again, this isn't something you would ordinarily do.)
9. From the SVN Repository Exploring Perspective, right-click the directory of your project repository and checkout the project. This is how you get a local, working copy of something in the repository that you don't currently have locally.

Resolving Editing Conflicts

What happens if two team members make a change to the same line of a source file, and then try to commit their separate changes? This is known as a *conflict* and must be resolved. The team members would need to discuss the conflicting changes, resolve them, and commit the final change. Work in teams of two or three for this part of the lab.

1. One of you should check-out the other's project from the repository.
2. Both of you should edit line 119 of `Colors.java`, one changing `blue` to `red`, the other changing `blue` to `green`.
3. One of you should now commit your working copy of the project to the repository.
4. The second of you should now try to commit your working copy. The commit will fail. From the `Team` menu, select `Update`, getting the current version of resources from the repository. Choose one of the conflicted files and use the `Edit conflicts Team` command to see your version and the current version side-by-side. You would use the `Team` command `Show in Resource History` to determine who committed the last version, so you could discuss changes.
5. Once you've resolved the differences and saved the final version, use `Mark resolved` from the `Team` menu to mark the conflict resolved and then commit the update.

If you think about it, while you are working so are others, and they are committing changes to the repository. How do you pick-up these changes? This is what the `Update` command on the `Team` menu is for. When your own changes are stable, but before you commit, use `update` to get the latest, most complete set of project files and re-test your changes. At the time of this update, you may be alerted to editing conflicts, which must be resolved. You should also always perform an update just before committing changes to the repository.