Tutorial III. Eclipse

Outline

- Basics
- Eclipse Plug-in feature, MVC
- How to build Plug-ins
- Exploring Eclipse source code for Editor
- Using CVS inside Eclipse
- Eclipse JDK Tips

Basics

- Eclipse projects:
  - Eclipse platform
    - Plugin architecture
    - Platform, JDT, PDT
    - A number of integrated plugins: JUNIT, CVS, etc.
  - Eclipse tools project
    - CDT, VE, AspectJ, Hipikat
- The official website of Eclipse: http://www.eclipse.org
- Eclipse forums
- Articles
- Eclipse plugins repository
- Eclipse bugzilla
- Using Eclipse in CDF:

  >setenv LD_LIBRARY_PATH /local/lib/eclipse (“.cshrc”)

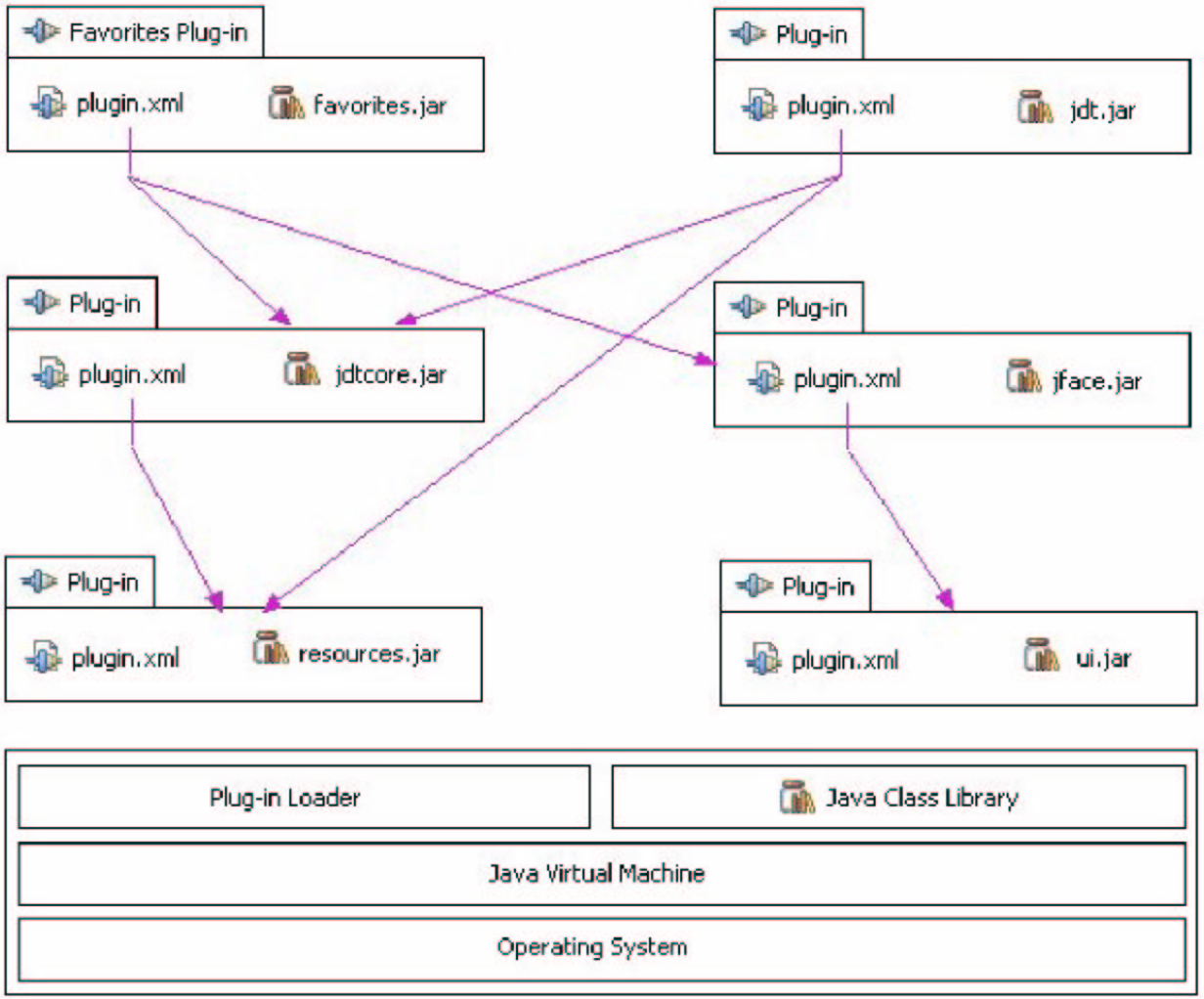
  >eclipse

Eclipse Plug-in Feature

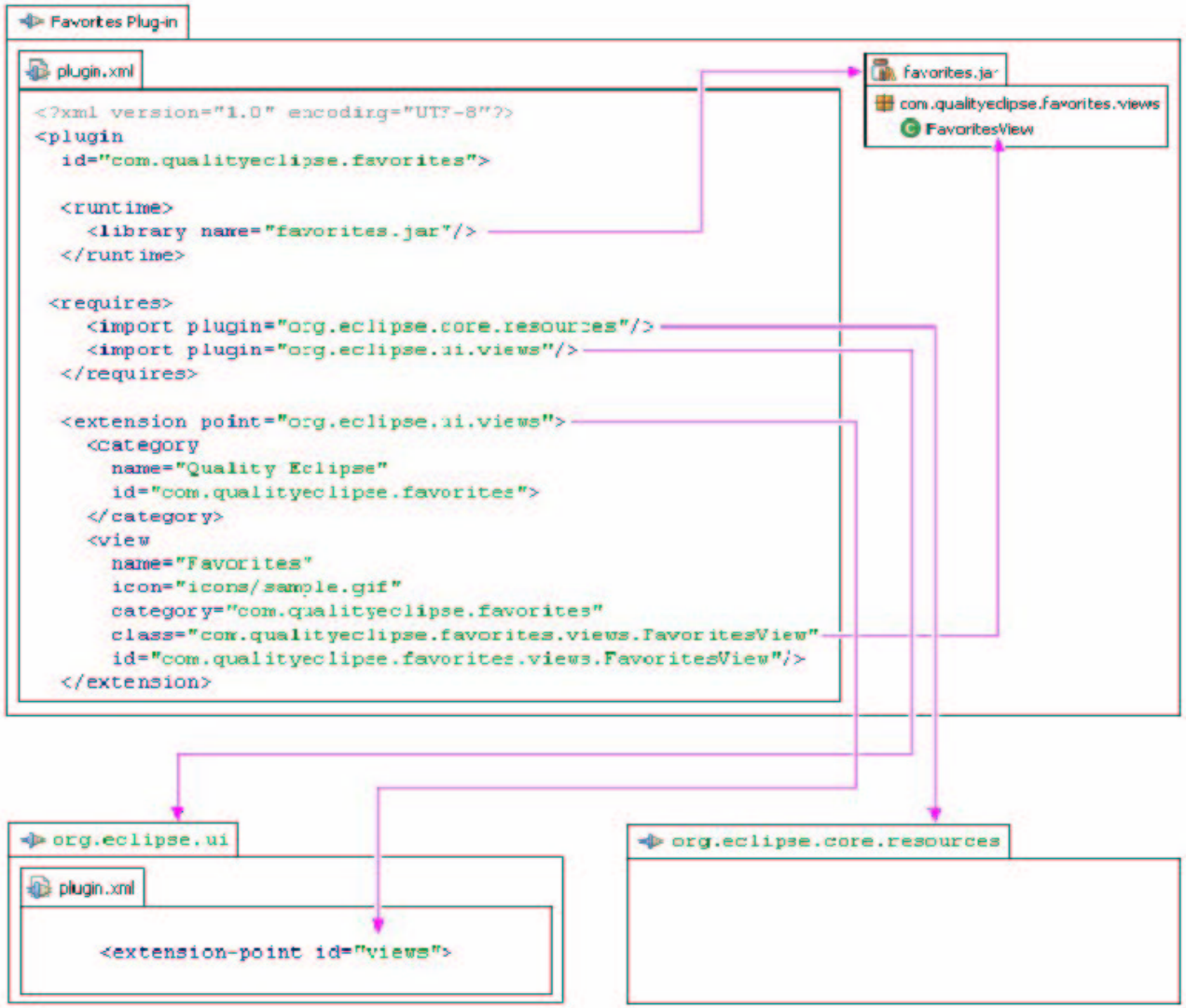
- Eclipse =

  a core **runtime engine** + a set of **plug-ins**
- Plug-in: the smallest extensible unit to contribute additional functions to the system.
- Extension point: boundaries between plug-ins

Eclipse Plug-in Structure



Plug-in Manifest file (plugin.xml)



Plug-in Lifecycle

- Plug-in registry
- Lazy loading
- Unfortunately, never unloaded
- Equinox project (www.eclipse.org/equinox)

How to build Plug-Ins

- Plug-ins contribute functionality to the platform by contributing to pre-defined **extension points**.
- The platform has a well-defined set of extension points - places where you can hook into the platform and contribute system behavior.

How to build Plug-Ins (cond’)

1. Decide how your plug-in will be integrated with the platform.
2. Identify the extension points that you need to contribute in order to integrate your plug-in.
3. Implement these extensions according to the specification for the extension points.
4. Provide a manifest file (plugin.xml) that describes the extensions you are providing and the packaging of your code.

Customized Editor

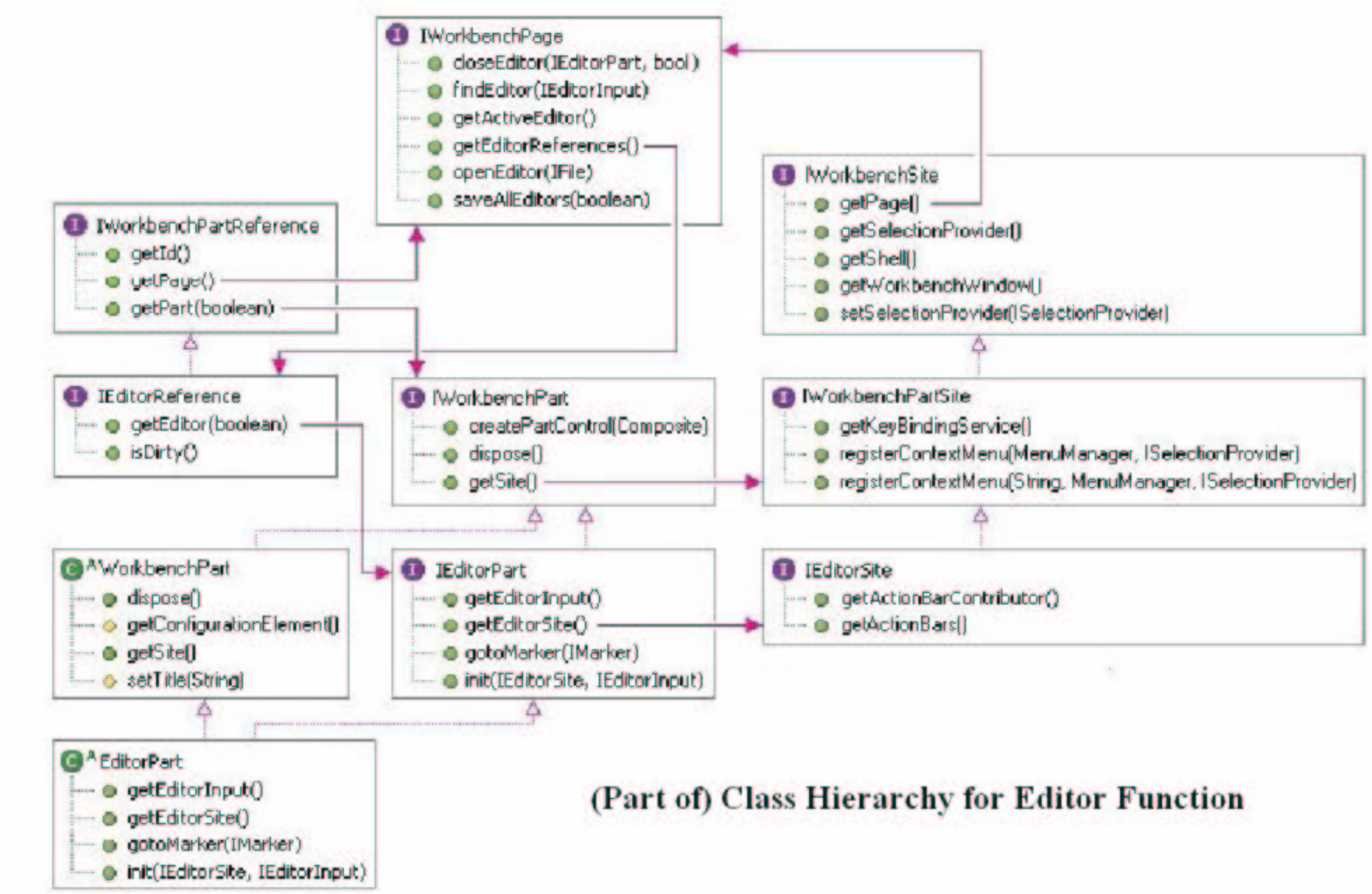
- An editor is a workbench part that allows a user to edit an object (often a file). An editor is always associated with an input object (**IEditorInput**).

- The interface for editors is defined in **IEditorPart**, but plug-ins can choose to extend the **EditorPart** class rather than implement an **IEditorPart** from scratch.

Using plug-in to enhance existing editors

- The workbench defines extension points that allow plug-ins to contribute behaviors to existing editors or to provide implementations for new editors.
- The workbench extension point **org.eclipse.ui.editors** is used by plug-ins to add editors to the workbench.
- Plug-ins that contribute an editor must register the editor extension in their **plugin.xml** file, along with configuration information for the editor.
- Editors can also define a **contributorClass**, which is a class that adds actions to workbench menus and tool bars when the editor is active

Exploring Eclipse Source Code



(Part of) Class Hierarchy for Editor Function

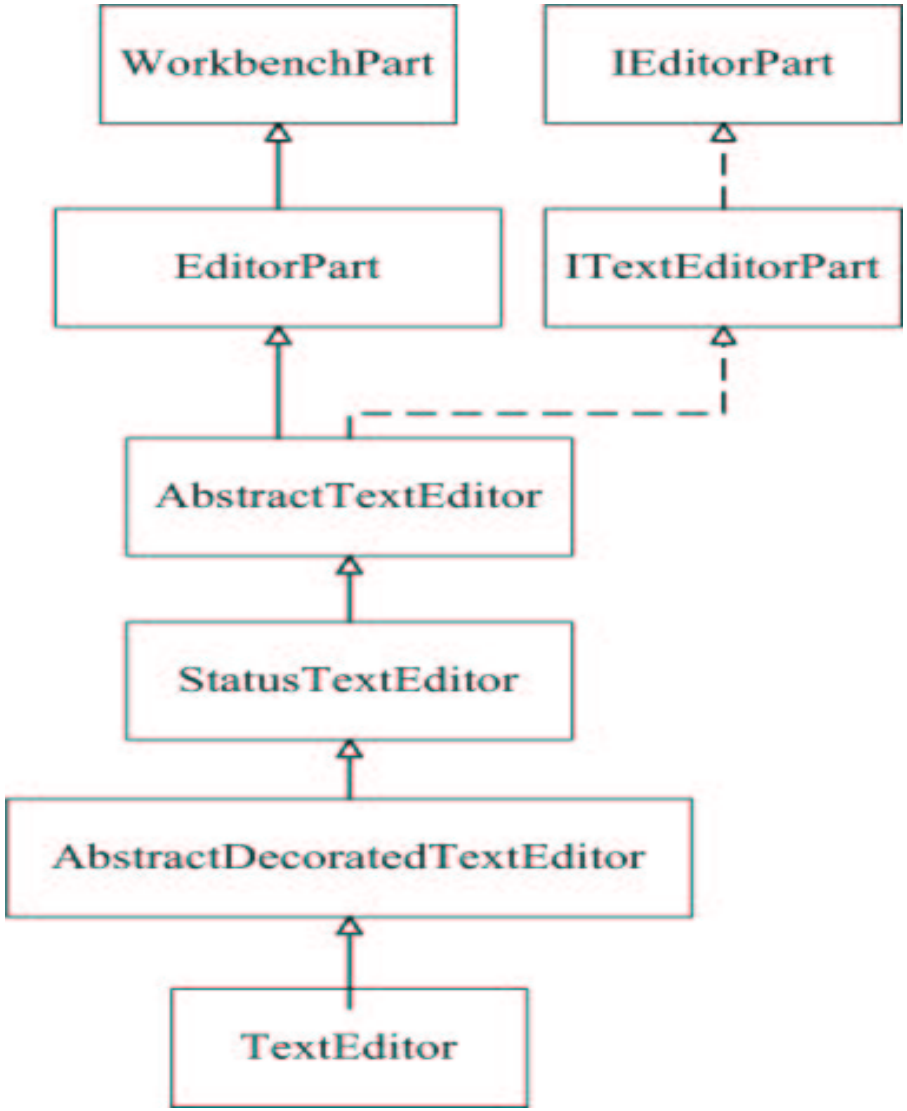
Class hierarchy for Text Editor

ITextEditor is defined as a text specific extension of **IEditorPart**.

The implementation of **ITextEditor** in the platform is structured in layers:

AbstractTextEditor defines the framework for extending the editor to support source code style editing of text. This framework is defined in **org.eclipse.ui.texteditor**.

The concrete implementation class **TextEditor** defines the behavior for the standard platform text editor. It is defined in the package **org.eclipse.ui.editors.text**.



A good example

- The text editor framework provides a model-independent editor that supports the following features:
  - presentation and user modification of text
  - standard text editing operations such as cut/copy/paste, find/replace
  - support for context and pulldown menus
  - syntax highlighting
  - content assist
  - key binding contexts

  …….

- Exploring how these features can be implemented in an editor by studying the **org.eclipse.ui.examples.javaeditor** example.

CVS (Concurrent Versions System)

- Help support and enhance the process of managing source code in two major ways:
  - by **controlling access** to the source code, using a locking system to serialize access
  - by **keeping a history** of the changes made to every file.

Using CVS inside Eclipse

CVS repository parameters for CDF:

CVS Server: werewolf or seawolf

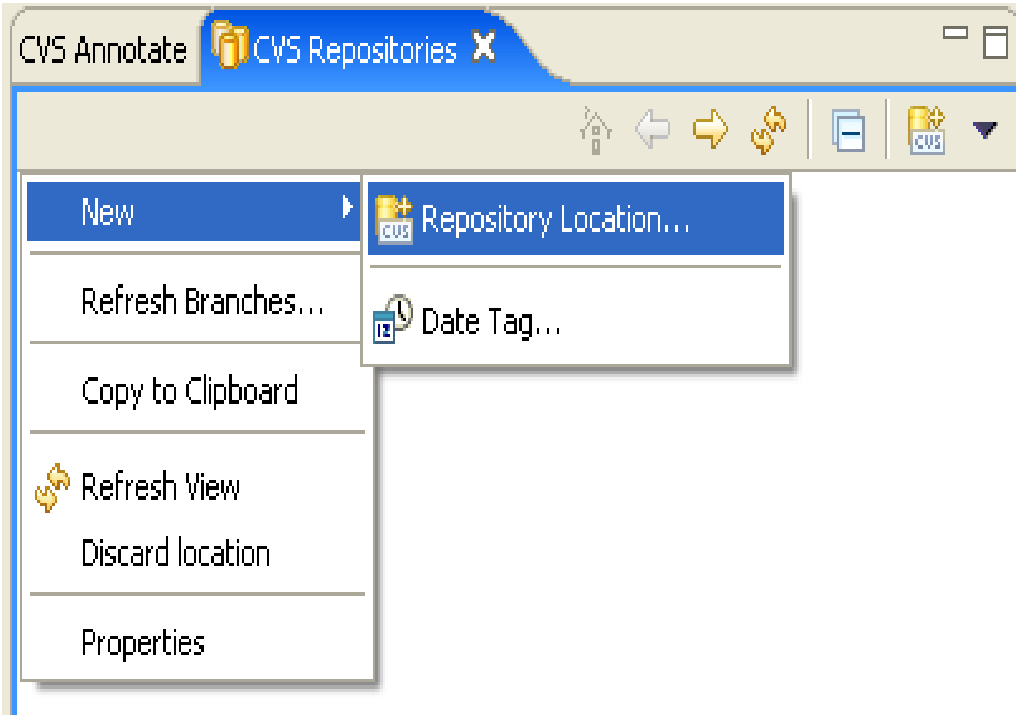
Repository Path: /u/yijun/cvsroot/c408h001

Connection Type: extssh, NOT pserver

Step 1: Creating a repository location

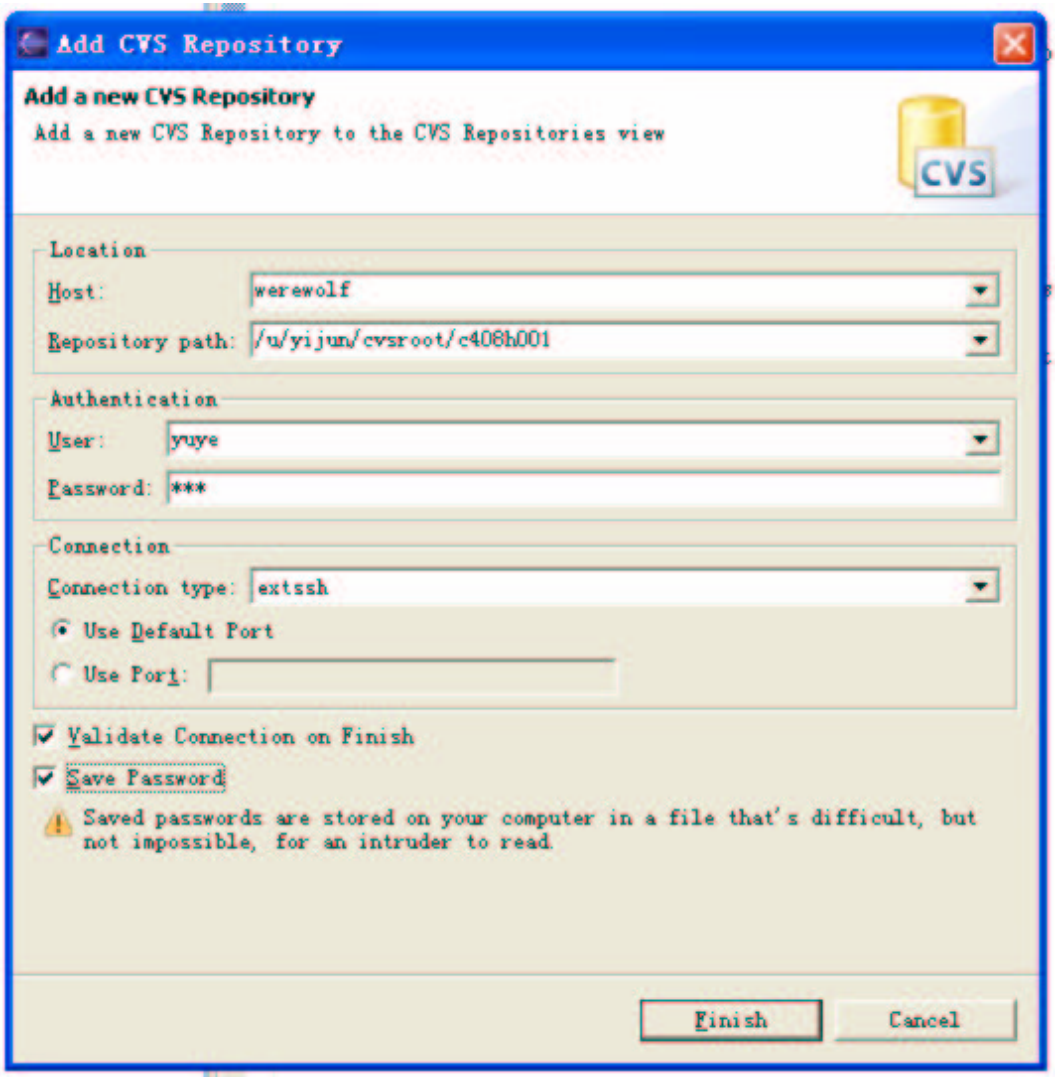
1. Using the **Window > Open Perspective > Other** command to open **CVS Repository Exploring Perspective.**

2. Right-click within the **CVS Repositories** view and select the **New > Repository Location** command from the context menu.



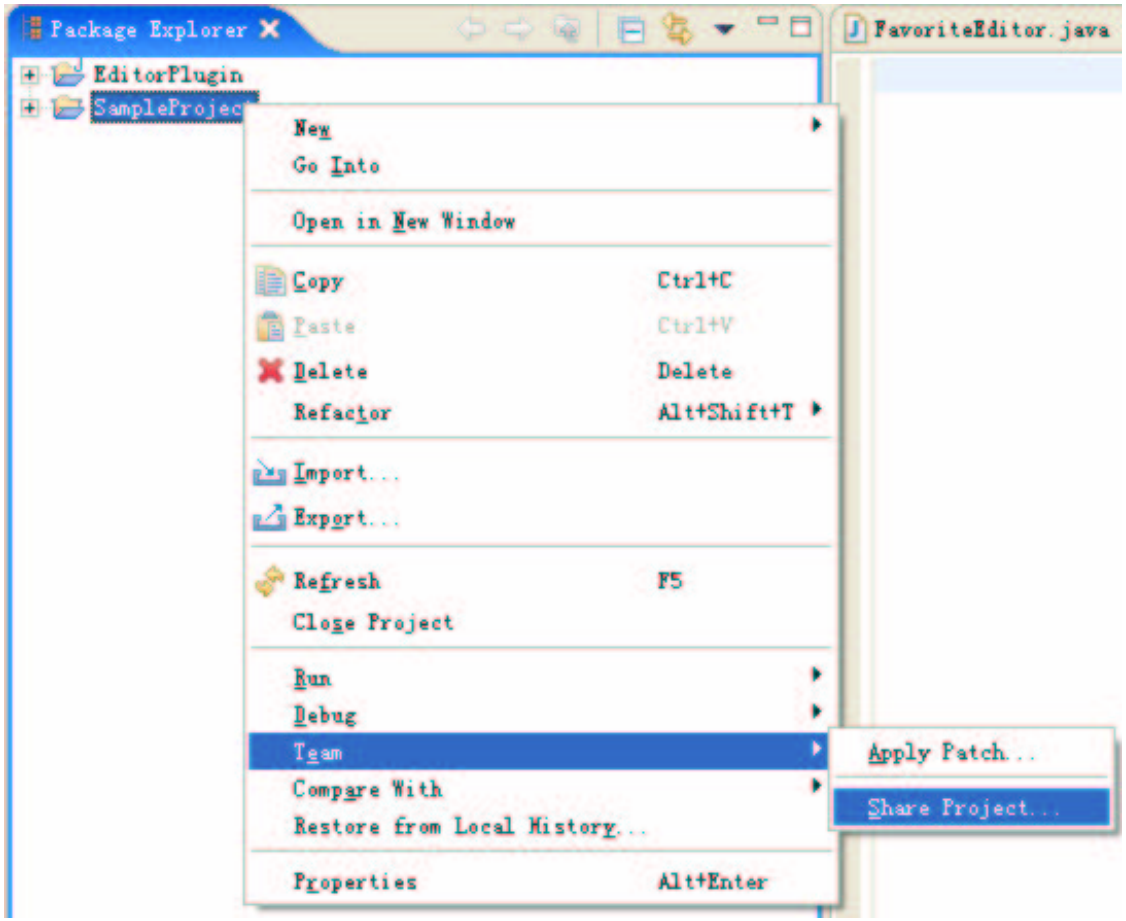
Step 1: Creating a repository location (cond')

1. Specify the address of **CVS host**;
2. Specify the **location** of your repository;
3. Enter your **login information**;
4. Select **Connection Type**;
5. Check '**Save Password**' (Optional);
6. Click '**Finish**' button.



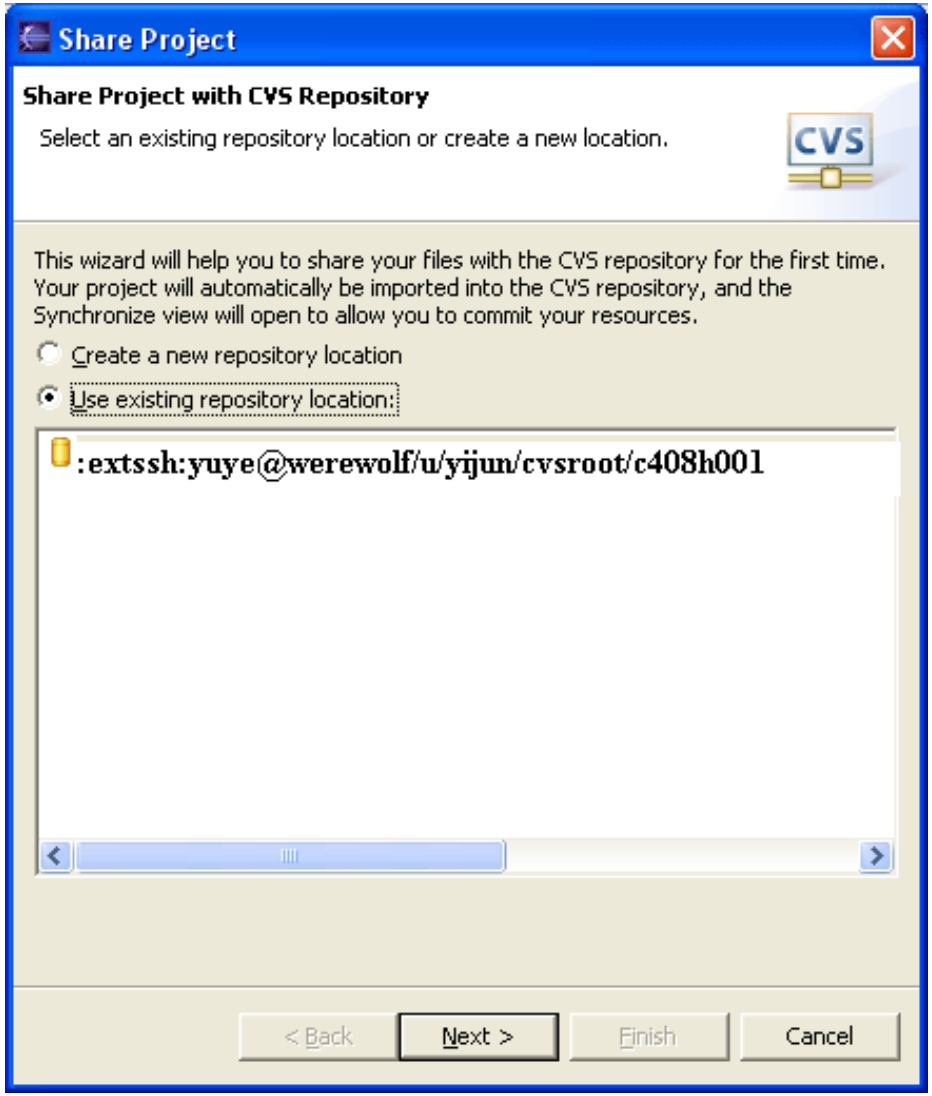
Step 2: Share a project

1. In the Navigator view select the project **SampleProject**.
2. From the project's context menu choose **Team > Share Project.**



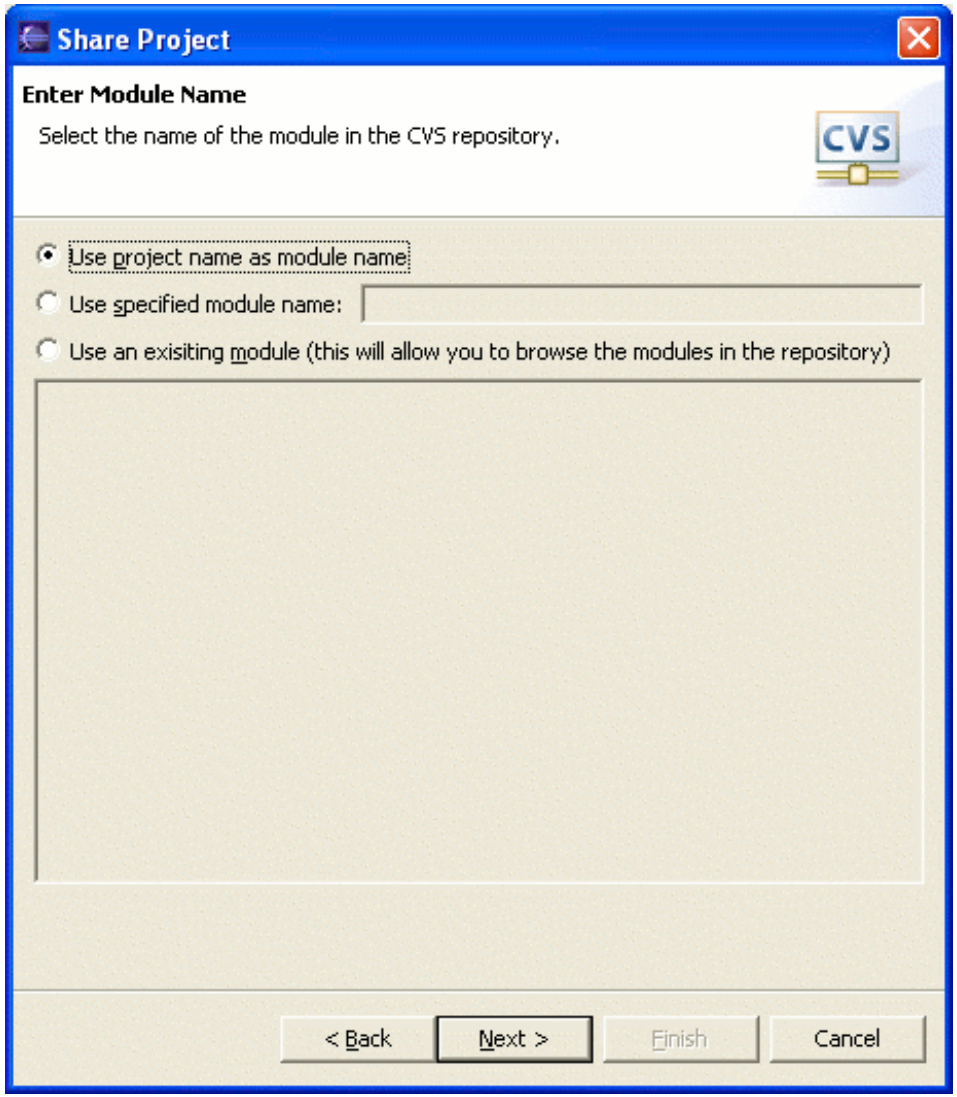
Step 2: Share a project (cond')

- In the sharing wizard page, select the location that was previously created.



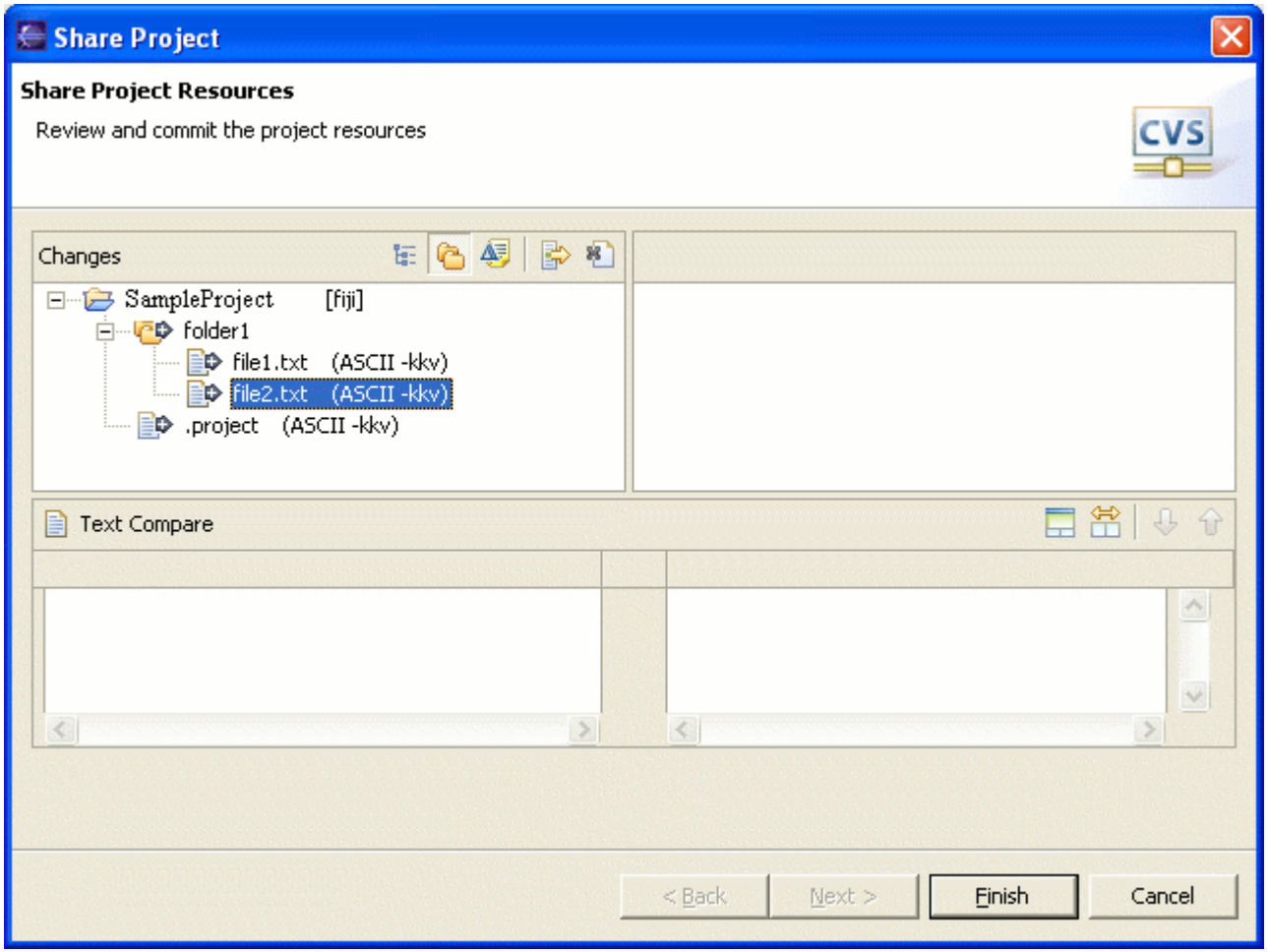
Step 2: Share a project (cond’)

- Specify the module name to create on the server. Simply use the default value and use the name of the project you are sharing. Click **Next**.

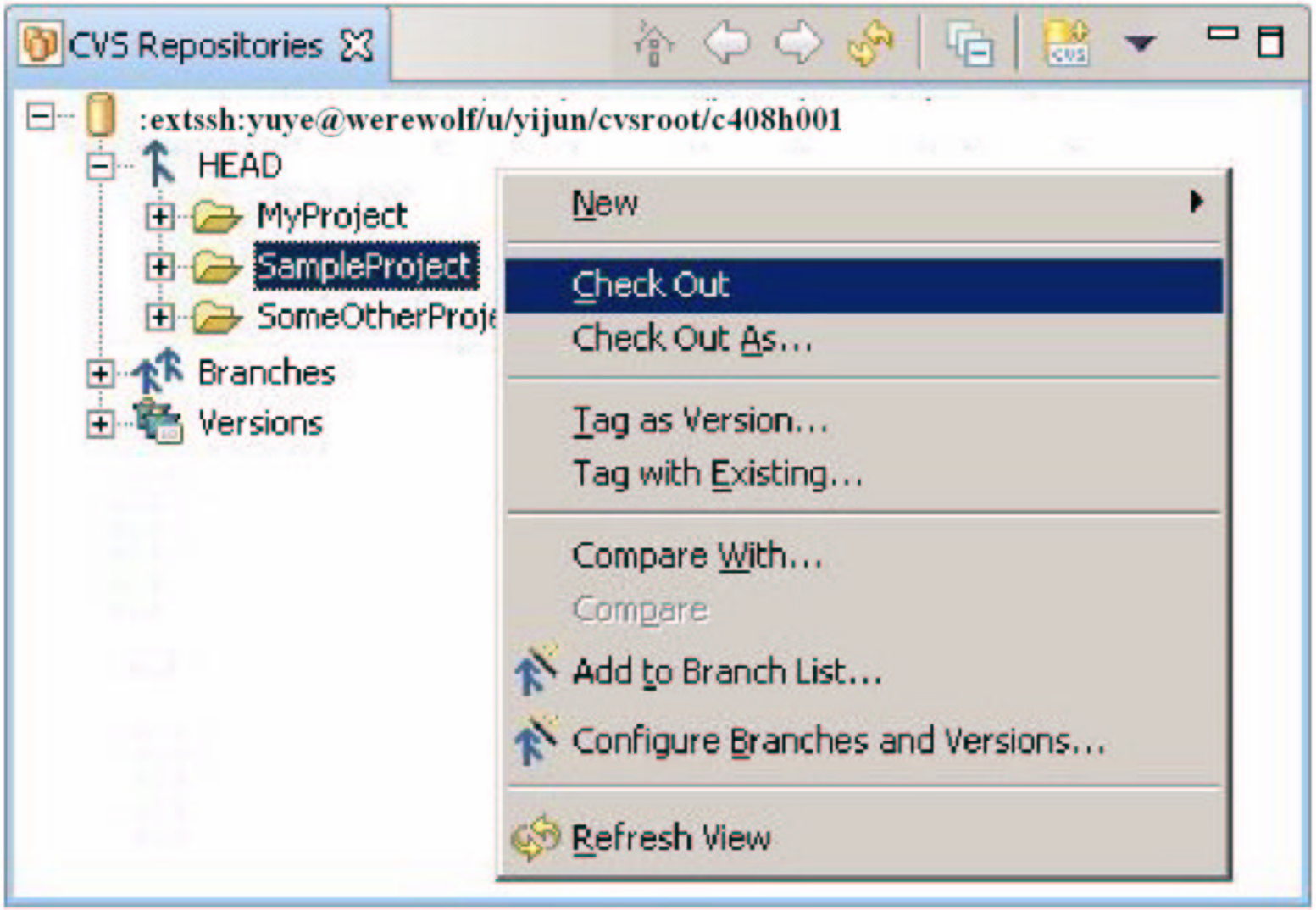


Step 2: Share a project (cond’)

- This page will allow you to see the files that are going to be shared with your team. The arrows with the plus sign show that the files are new outgoing additions.
- Click ‘**Finish**’ button

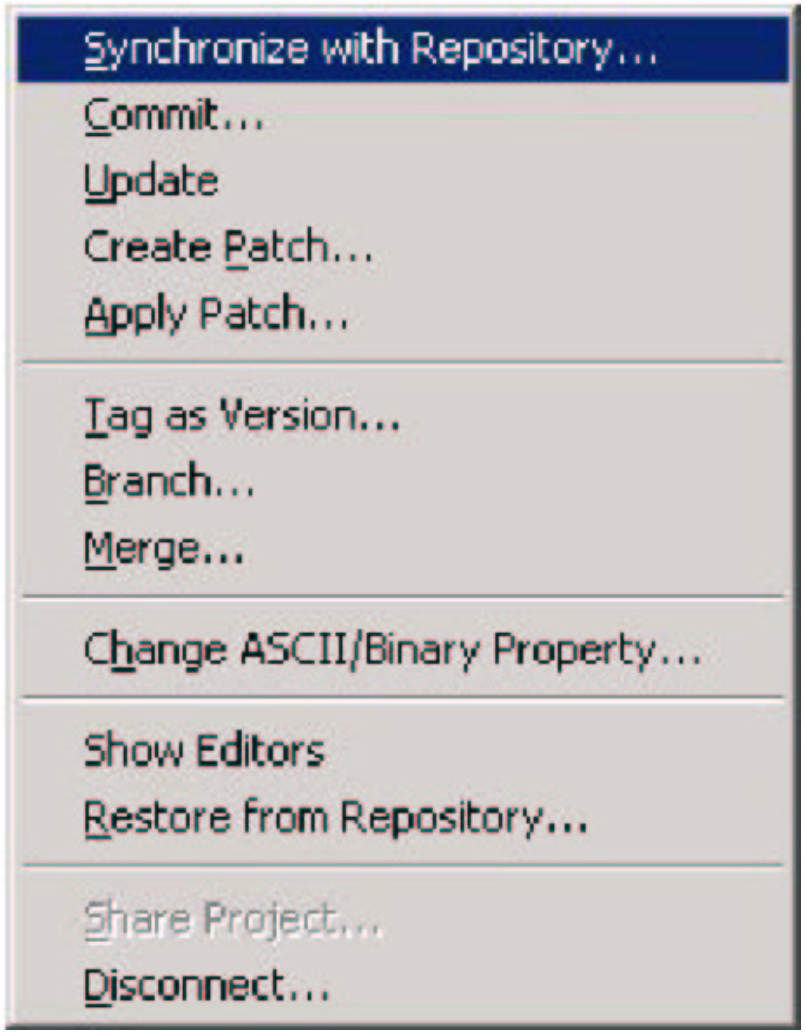


Check out a project from CVS



Step 3: Synchronize with the repository

Right-click on the resource (or the project containing the resource) and select the **Team > Synchronize with Repository** command.

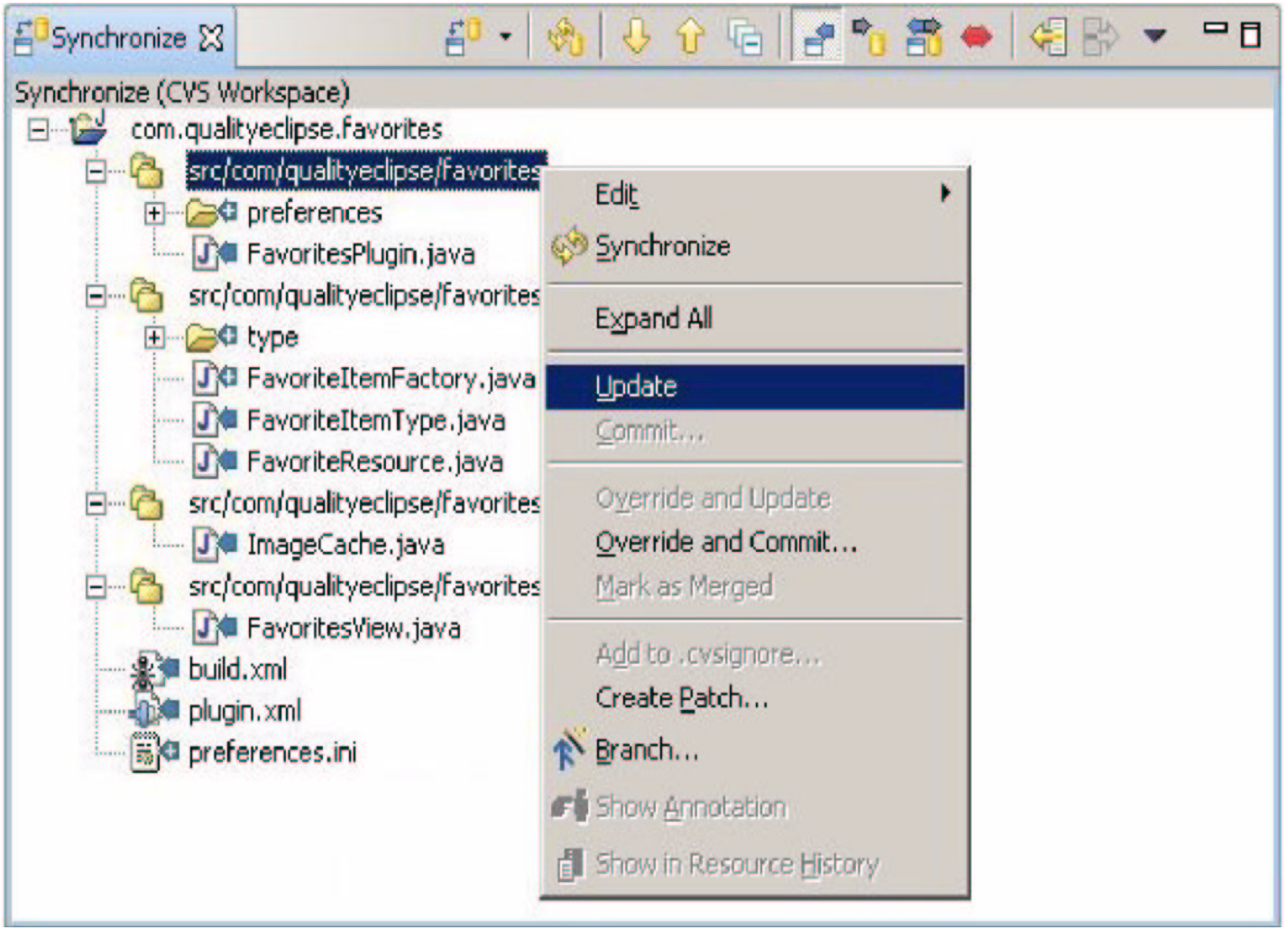


Step 4: update/commit the changes

The **Incoming Mode** causes the view to only show incoming changes.

The **Outgoing Mode** causes the view to only show outgoing changes

The Incoming/ Outgoing Mode will show changes in both sides.

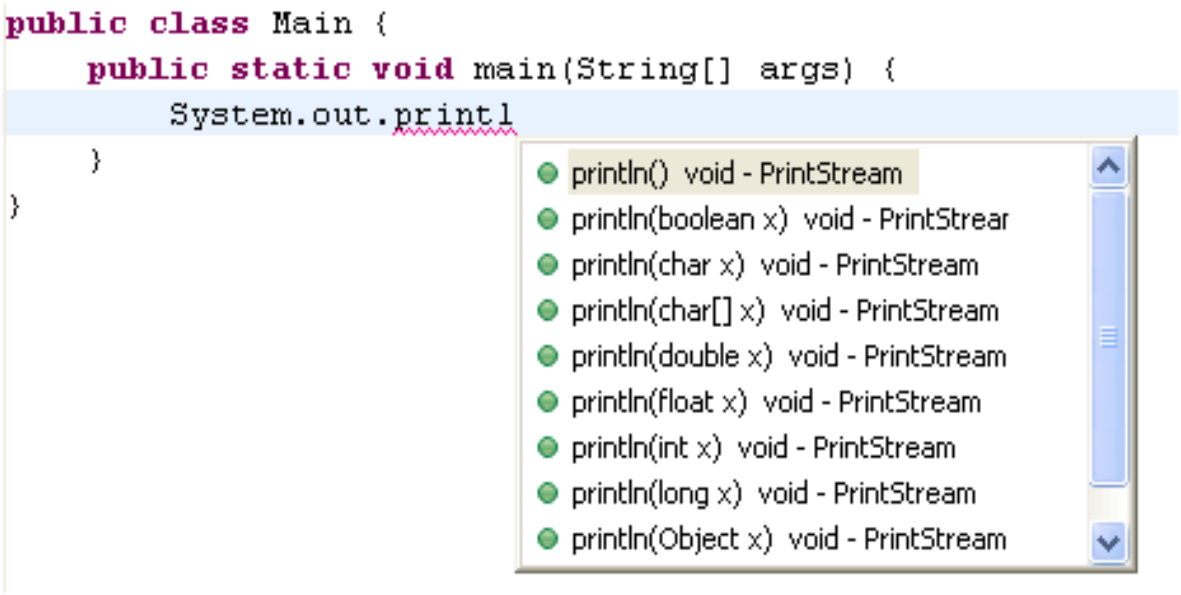


.cvsignore

Eclipse JDK Tips - **Content Assist**

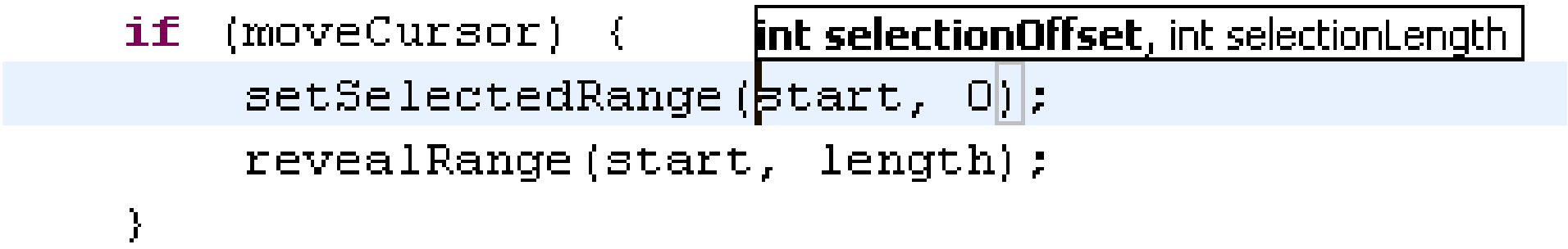
- **Content assist** provides you with a list of suggested completions for partially entered strings.

  Ctrl+Space or ***Edit > Content Assist***



Eclipse JDK Tips - **Parameter Hints**

- With the cursor in a method argument, you can see a list of parameter hints.
- ***Ctrl+Shift+Space*** or ***Edit > Parameter Hints.***



Eclipse JDK Tips - **Quick Fix**

- Start with the method invocation and use **Quick Fix** to create the method.
- ***Ctrl+1***

Eclipse JDK Tips – **Code Navigation**

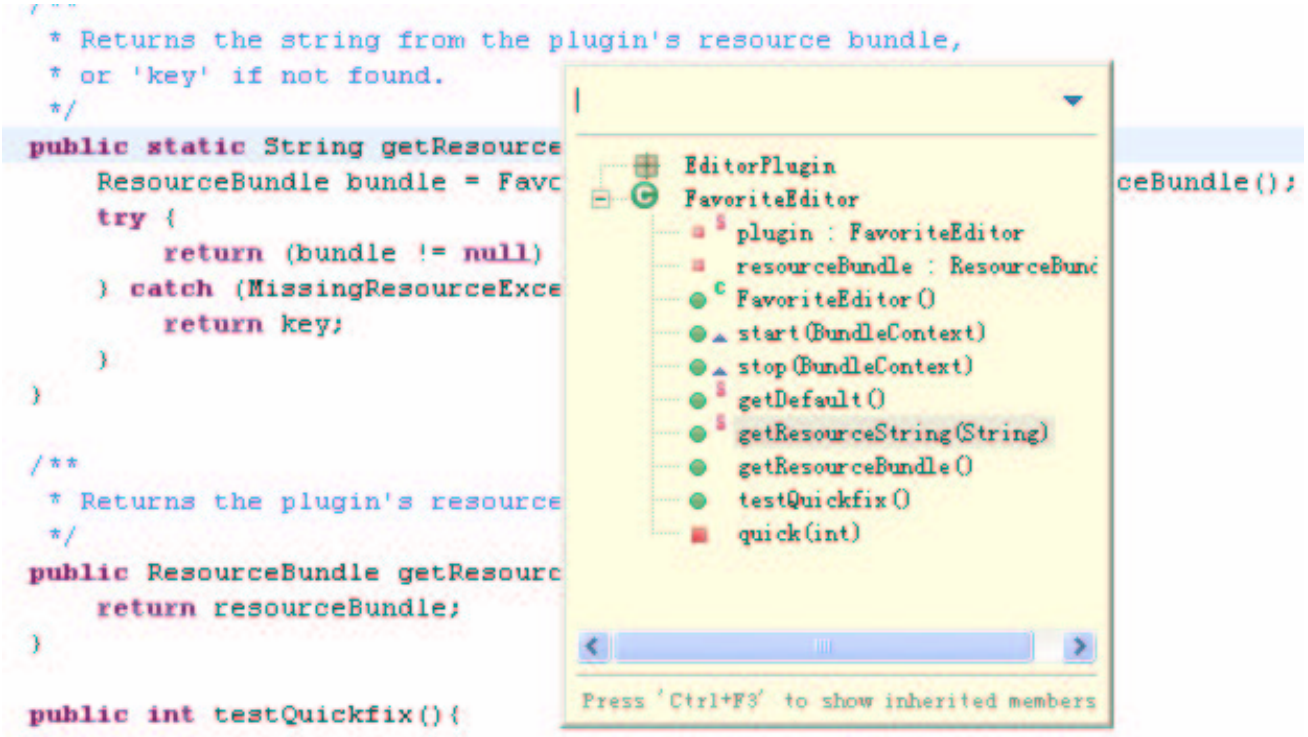
- There are two ways that you can open an element from its reference in the Java editor.

1. Select the reference in the code and press ***F3*** **(*Navigate > Open Declaration*)**
2. Hold ***Ctrl*** and move the mouse pointer over the reference.

```
MessageDialog.openError(fShell, title, message);
return;
```

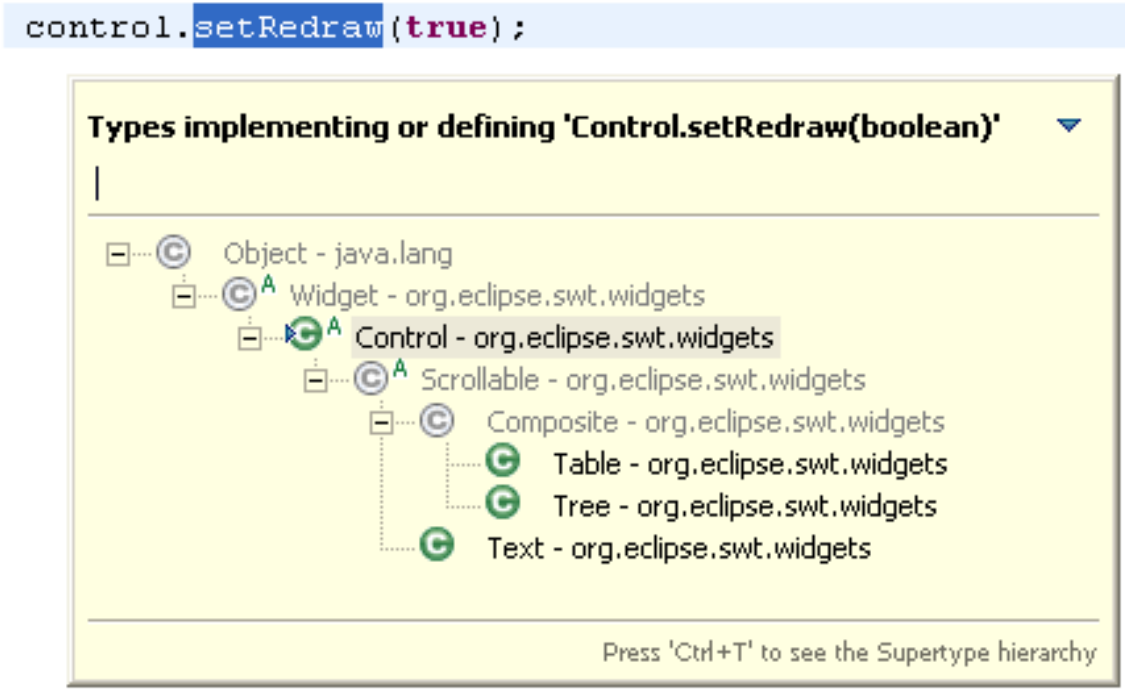
Eclipse JDK Tips – **In-place outlines**

- Press ***Ctrl+F3*** in the Java editor to pop up an in-place outline of the element at the current cursor position.



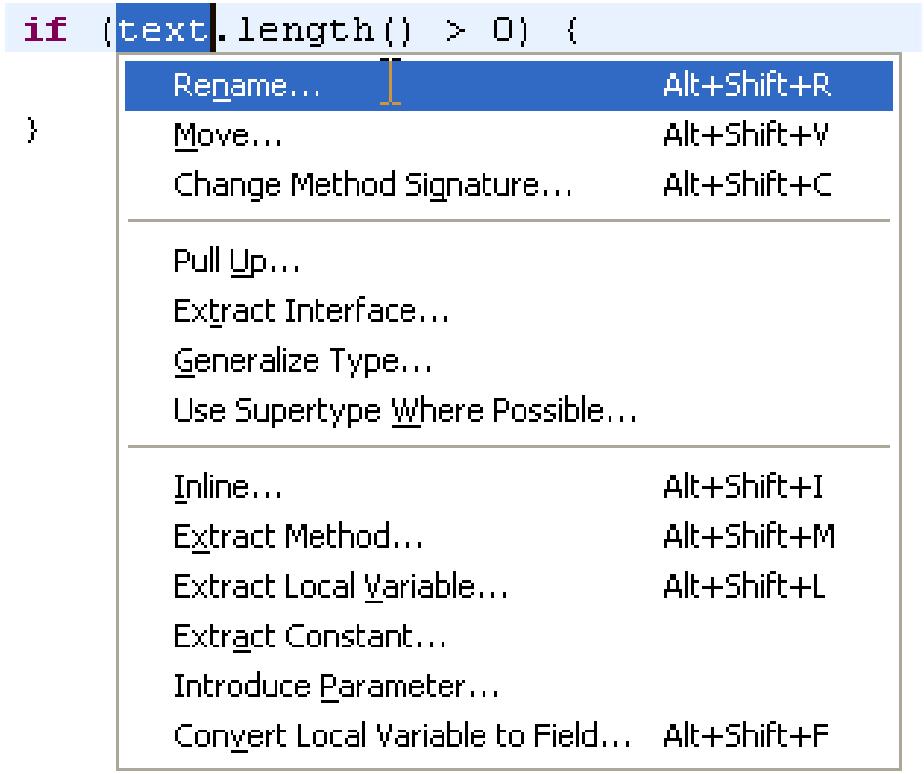
Eclipse JDK Tips – **In-place hierarchy**

- Place the cursor inside the method call and press ***Ctrl+T*****.**The view shows all types that implement the method with a full icon.



Eclipse JDK Tips - Refactoring

- Select the element to be manipulated in the Java editor or in a Java view and press **Alt+Shift+T** for the quick refactor menu.



Eclipse JDK Tips

- More Tips and Tricks can be found in ***Eclipse Help > Tips and Tricks…***

Checkout the right version of the Editor part

- cvs repository: :pserver:anonymous@dev.eclipse.org:/home/eclipse
- Versions
- org.eclipse.ui.editors
- org.eclipse.ui.editors R3_0
- Checkout

Reference

- *Eclipse 3.0 Help*
- *<< Building Commercial Quality Eclipse Plug-ins >>*
- *<< Eclipse In Action >>*