

HOW TO TRAIN THE JDT DRAGON



TUTORIAL 1. FLAP THY WINGS, LEARN TO FLY


Objective: Learn the usage of Java model through a simple plugin

To do: Create a plugin that contributes a command such that when a given java element is selected in the package explorer and the command is clicked, a dialog pops up to display the name of the selected element and its children.

Steps: (see also the git how-to)

1. Go to git repositories view -> expand the org.eclipsecon2012.jdt.tutorial repository. Expand "Remote tracking" and create a new local branch called 'learn' off of the 'origin/learn' branch.
2. Work with the org.eclipsecon2012.jdt.tutorial1 plugin project.
3. The org.eclipsecon2012.jdt.tutorial.handlers.SampleHandler class has some code written for you.

Exercise 1. collectNamesForProject(..)

- a. Check if the current project is a java project using the `JavaCore.NATURE_ID`.
- b. Append the project name to the message.
- c. Make a call to `collectNamesForPackage(..)` to handle the next java element i.e. package fragment.
- d. Test your plugin. Create a new Run/Debug Configuration (Debug will enable hot swapping so that any further changes in the code do not require a re-launch.). Double click on "Eclipse Application" -> name the config. -> go to the "plug-ins" tab and click on "Validate plugins" to see if everything is ok, and then click "Run"/"Debug". You will now see a  icon in the toolbar (coolbar) corresponding to our new command (You will also see a "sample command" in the new "Sample menu"). Create a new Java Project. You can now use the provided populator plugin to generate some sample classes. To do so, right click on the "src" folder and select JDT Tutorial -> Populate with sample classes. Now, select the project in the package explorer and click on the new command. You should get a dialog printing the project name.

Exercise 2. collectNamesForPackage(..)

- a. Check if this is a source package using `IPackageFragmentRoot.K_SOURCE`. If yes, proceed.
- b. Obtain its name and append it to the message.
- c. Make a call to `collectTypeName(..)` to handle the next java element i.e. a java class.
- d. Test the plugin as before.

Exercise 3. Implement default functionality – what happens if no valid element is selected and the command is used?



- Start from the IWorkspace workspace = ResourcesPlugin.getWorkspace(); line in the execute(..) method and obtain a list of all projects in the workspace.
- Use the functionality developed in earlier exercises to print the names of the projects and their packages and types.
- How will you test the plugin now? Go figure. ☺

Exercise 4. (slightly advanced) A java element is missing in the hierarchy we print. Which one is it? Write some more code!

(Hint: What happens if you select on the “src” folder and use the command? What is expected?)

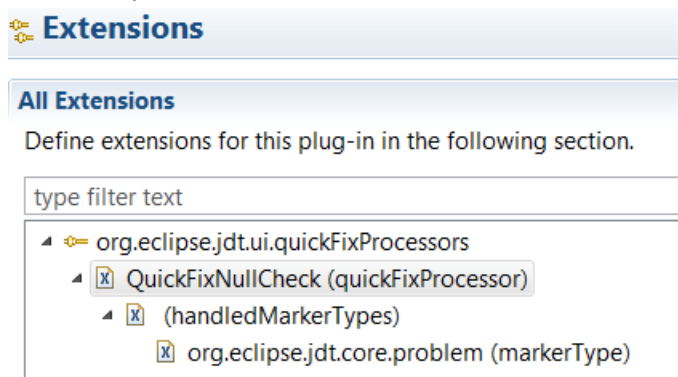
TUTORIAL 2. LEARN TO FIGHT YOUR ENEMIES

Objective: Learn modifying source code using AST Rewrite, interpreting java problems and creating a simple quick fix.

To do: Create a plugin that contributes a quick fix which adds a null check for a variable on which “potential null pointer access” problem is reported (See C1.java generated by the populator plugin during tutorial 1.).

Steps: (see also the git how-to)

- Make sure you have the org.eclipsecon2012.jdt.quickFix1 project from the org.eclipsecon2012.jdt.tutorial repository.
- Create a fresh branch from the commit “tutorial2 project” from the history.
- Look at the manifest file of the plugin. We extend the org.eclipse.jdt.ui.quickFixProcessors extension point.






4. We also want to deal only with JDT/Core's java problem markers, so we add the `handledMarkerTypes` attribute.
5. Also look at the "Dependencies" tab of the manifest file to see what all plugins are needed for extending JDT/UI in quick fixes, quick assists, clean-ups, etc.
6. Now look at the `org.eclipse.jdt.quickfix1.processor.QuickFixNullCheck` class. A quick fix has to implement the `org.eclipse.jdt.ui.text.java.IQuickFixProcessor`.

Exercise 1. `process(..)`

The problem id for the "potential null pointer access" problem is `IProblem.PotentialNullLocalVariableReference`. The `process(..)` method needs to act on this problem.

Exercise 2. `addNullCheckProposal(..)`

Go to the runtime workbench and open the `ASTView` view. Select the variable on which the problem is reported and click on  command in the view. Inspect the AST elements you're dealing with. What is the AST node corresponding to the variable? Code has been provided to you in `addNullCheckProposal(..)` to obtain the selected node, its parent AST, an `ASTRewrite` and using these, to create a quick fix proposal, which is basically a no-op right now. Add sanity checks in the code.

- a. Inspect the `ASTView` once again to figure out which `ASTNode` has to be **replaced** by an `if` statement.
- b. Create the building blocks of the new AST node – an if statement, a condition expression and a then block.
- c. Set the operands and the operator for the if's condition expression.
- d. Add the dereferencing statement on which the error was reported into the **then** block of the **if**. Since the statement already exists, we simply prepare to move it, rather than creating a new statement mimicking it. That's why, `ASTRewrite` comes into play.
- e. Use `ASTRewrite` to **replace** the problematic dereferencing statement with the new `if` statement.
- f. Test the quick fix using classes `P1`, `C1` and `TestAnnot`.

Slightly advanced:

- g. Why does the code fail for `TestAnnot`? Can you implement a valid quick fix for that case? (*Hint: `ASTView` is your friend.*)
- h. Can you change the default image which comes in the popup preceding the quick fix label?