Refactoring with Eclipse
Using the new open source IDE for improving working code

Daniel Steinberg

Object oriented developers recognize the value of refactoring working code. Until recently good tools have not been available. At OOPSLA 2001 in Tampa, Florida, OTI showed their new open source IDE that features Refactoring support.

The Eclipse Platform

Eclipse is an open source IDE created by the team at Object Technology International (OTI) that was responsible for VisualAge for Java and the ENVY Developer and Smalltalk editions. Eclipse is freely available as an open source tool and will also be the foundation for the WebSphere Workbench. You can find out more by joining the Eclipse community (see Related topics later in this article).

Eclipse is a platform designed to get most of its functionality from plugins written in Java. At the center of Eclipse is the Platform that provides the look of the application and the core functionality. When you start up Eclipse, this Platform discovers and registers the available plugins. The plugin isn't loaded into memory until it is needed. This architecture allows fast switching between plugins based on your context. If you are working with a JSP then you'll need to use an HTML-aware tool some of the time and a Java development tool at other times. Eclipse switches between them as needed.

Given that much of Eclipse is written in Java, you may expect the tool to run on any platform with a Java VM. The presentation layer, however, is not built on Swing but instead on what they call the Standard Widget Toolkit (SWT). The SWT, like Java's AWT, has to be customized to each operating system on which it runs. The SWT uses native widgets when they exist and finds ways of emulating them where they don't. The advantage is that Eclipse is not limited as AWT was to support only those widgets that are common to all targeted operating systems. The disadvantage is that, unlike Swing, an application doesn't look exactly the same on all platforms since it is using the native widgets. Native widgets can behave slightly different on each platform. However, this can be considered a benefit since it gives the application the same look as other native applications.
Refactoring with Eclipse

Erich Gamma is the team lead for Java tools for Eclipse. Gamma was one of the Gang of Four known for creating the book *Design Patterns: Elements of Reusable Object Oriented Software*. He also created JUnit with Kent Beck (see Related topics). Refactoring is recognized as another valuable practice in object oriented programming but, until recently, only few tools had support for it. At OOPSLA 200, Eclipse developers demonstrated the Refactoring support in Eclipse. They stressed that refactoring should not alter a program's behavior.

The Extract Method is considered a key refactoring in refactoring circles. For refactoring to be a regular part of the development process, it has to be easy for the developer to do and to undo refactorings. The user interface of your tool should easily support various refactorings. Currently, Eclipse supports renaming refactorings that allow you to rename a compilation unit, type, method, field, or parameter. Even such a straightforward refactoring as a renaming has to locate all of the references to the renamed item. Other supported refactorings allow you to move code, extract methods, and self encapsulate fields.

There is more refactoring support coming but getting it all to work the way a developer might expect is a non-trivial task. Eclipse developers pointed out that their implementation of the Extract method required over 200 unit tests. He explained that when you provide support for a refactoring, you begin by detecting the affected compilation units, you then perform a detailed analysis of the program's structure, and finally you perform the code changes while allowing the user to easily undo the current refactoring. The UI allows the user to choose the refactoring and to pass in the information needed for the selected refactoring. The user then sees the "before" and "after" version of the code and has a chance to respond to any problems flagged by the tool.

**Step 1. Detecting affected compilation units**

The first step in implementing a refactoring is to identify all of the code it affects. Eclipse has a program database with information about program elements and the references to them. So if you are going to change the name of method `doThis()` to `doThat()` then every place that *this particular* `doThis()` is called the reference must be changed to `doThat()`. The database is kept in sync with the code and not with the compiled version of the code. The Eclipse search infrastructure associates the declared or referenced name with a file's path and also rates the quality of the match. The matched files are then parsed with a Java parser that adds information such as the location of a reference within the compilation unit.

**Step 2. Analyzing the code structure**

Once you know what code is affected by a refactoring, the next step is to analyze the structure of the code within the program. This structural analysis is performed in two ways using Eclipse. First the Java Model makes it possible to look at declared types, methods, and fields as well as the results of import statements. This gives you a view of what the various classes are and the fields and methods that they contain. This isn't enough because some of the refactorings are more fine grained. The abstract syntax tree provides access to the statement level program analysis that your refactoring might need. Consider what would be required to take a block of code from...
within a particularly long method and use Extract Method to create a separate method with this functionality as well as a call to that method in place of the code you extracted. This requires a view of the program not provided by the Java Model.

**Step 3. Creating code changes**

The final step of the process is to actually make the change and to allow for the change to be undone. In Eclipse this is done with objects called Change objects. As you might guess, because you want to support undo, this process is implemented using the Command pattern. The two basic types of changes are structural changes and textual changes. The structural changes involve manipulation of the file system such as deleting a file. Textual changes involve changing the code written in one or more compilation unit.
Related topics

- You can join the Eclipse community and download this new open source IDE being pioneered by OTI at the Eclipse.org site. There you'll also find a description of Eclipse projects, along with articles and newsgroups.
- Martin Fowler's site on refactorings contains useful links and references.
- Learn more about Erich Gamma's and Kent Beck's open source unit testing tool, JUnit.
- Find the latest news on the WebSphere tools at IBM's WebSphere Developer Domain.
- Download the C/C++ IDE Plugin for the Eclipse Project workbench (from IBM alphaWorks), which provides a fully functional C and C++ Integrated Development Environment (IDE).
- Browse more Open source resources on developerWorks.