Refactoring in Eclipse [1, 2]

by Raphael Enns

Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, Canada

Last revised: February 25, 2004

Overview:

In this tutorial, we will discuss refactoring in Eclipse. We will describe what refactoring is and why it is used. We will take a look at each refactoring that Eclipse allows you to do and see when and where to use the refactorings.

What is Refactoring?

Refactoring is the process of changing the structure of a program while maintaining all of its functionality. There are many types of refactorings that you can do such as renaming a class, changing a method signature, or extracting some code into a method. With each refactoring, you carry out a number of steps that keep your code consistent with the original code.

Why Refactoring is Important:

When refactoring by hand, it is easy to introduce errors into your code such as spelling mistakes or missing a step in the refactoring. To prevent and quickly fix these errors, thorough testing should be performed before and after each refactor. You may wonder if refactoring is worth going through all this.

There are several reasons why refactoring should be used. You may want to update a program that is poorly coded. Perhaps none of the original design team is present and no one on the current design team understands the code. In order to update it, you will have to redesign and restructure the program to fit what you want it to do. Another reason is that you may want to add a feature that the original design cannot accommodate. In order to add it, you will have to restructure the code. The third reason is that an automatic refactoring tool, such as the refactorings in Eclipse, can generate code for you.

---

[1] This work was funded by an IBM Eclipse Innovation Grant.

By using refactorings, you can easily change the structure of a program to what makes logical sense while rewriting code as little as possible and still keeping its functionality.

If refactoring is used on a regular basis to constantly keep a good structure, less time will be needed to fix any bugs and it will be easy to add new code to the design.

Testing:

Testing your code is very important when refactoring. Because refactoring changes the structure of your code, you must make sure that the resulting code functions the same way as the code did before the refactoring.

When doing the refactorings by hand, a good suite of tests is a must. When using an automated tool to refactor, you should still test, but it need not be quite as often. This is because the tool will not make some of the errors you might make when you refactor by hand, such as spelling mistakes.

Use JUnit to easily create tests for your application in Eclipse. For more information on creating tests using JUnit, see the tutorial *Using JUnit in Eclipse*.

Refactoring in Eclipse:

The Java part of Eclipse, JDT, is able to perform several types of automatic refactorings on Java projects, classes, and their members. There are several ways to quickly select a refactoring for an element in a Java project.

To perform a refactoring for one or more elements, you must have the element(s) selected. You can select elements in several views including the Outline view and Package Explorer view. To select multiple elements in a view, hold down Ctrl or Shift and then select the elements. Another way to select an element is to highlight it in the editor or place your caret in it in the source file (a caret is the blinking cursor displaying where the text will go when you type). After you have selected the element(s) you want to refactor, either choose a refactoring from the dropdown Refactor menu, or right-click and choose a refactoring from the Refactor submenu of the popup menu. There are also shortcut keys that can be used for several of the refactorings.

Some refactorings can be applied to any type of element. Other refactorings only apply to certain types of elements such as a method or a class. There is a table at the end of this tutorial listing the refactorings, the elements each refactoring can be applied to, as well as any shortcut keys for the refactorings.

All of the refactorings in Eclipse are able to be previewed before performing the refactor. Simply click on the Preview button in the refactoring dialog and view all the changes that will occur. The only refactoring that does not have a preview button is Pull Up. The preview pane will always appear on the last screen in the Pull Up refactoring wizard. Individual changes that will result from the refactoring can be deselected in the preview pane so that they will not occur.

Undo and Redo:

There are Undo and Redo items in the Refactor menu. These are not the same as the Undo and Redo items in the Edit menu. The Undo and Redo items in the Edit menu only modify the current file, even if a refactoring modified several files. The Undo and Redo items in the Refactor menu will undo or redo any changes made in all the files modified during a refactoring. The Undo and Redo items in the Refactor menu also have some restrictions on their use.

You cannot undo or redo a refactoring if any files have been modified and saved after a refactoring, regardless if they were modified in the refactoring. If a file was modified in a refactoring then modified outside of a refactoring but not saved, an error message will appear saying that you must revert unsaved files when you try to undo or redo a refactoring.

As long as you take note of the restrictions above, you can undo and redo refactorings as much as you want. You can even compile and run your program as well as run tests and still be able to undo the refactorings as long as you do not modify and save any files.

Types of Refactoring in Eclipse:

If you look in the Refactor dropdown menu in Eclipse, you will notice four sections. The first section has Undo and Redo in it and the other three sections contain the three different types of refactorings available in Eclipse.

The first type contains refactorings that change the physical structure of the code and classes such as Rename and Move. The second type contains refactorings that change the code structure on a class level such as Pull Up and Push Down. The third type contains refactorings that change the code within a class such as Extract Method and Encapsulate Field. The sections and their refactorings are shown below.

Type 1 – Physical Structure

- Rename
- Move
- Change Method Signature
- Convert Anonymous Class to Nested
- Convert Nested Type to Top Level (Eclipse 2 only)
- Move Member Type to New File (Eclipse 3 only)

Type 2 – Class Level Structure

- Push Down
- Pull Up
- Extract Interface
- Generalize Type (Eclipse 3 only)
- User Supertype Where Possible

Type 3 – Structure inside a Class

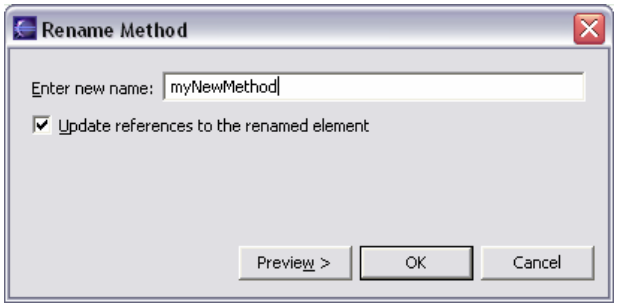
- Inline
- Extract Method
- Extract Local Variable
- Extract Constant
- Introduce Parameter (Eclipse 3 only)
- Introduce Factory (Eclipse 3 only)
- Encapsulate Field

Rename:

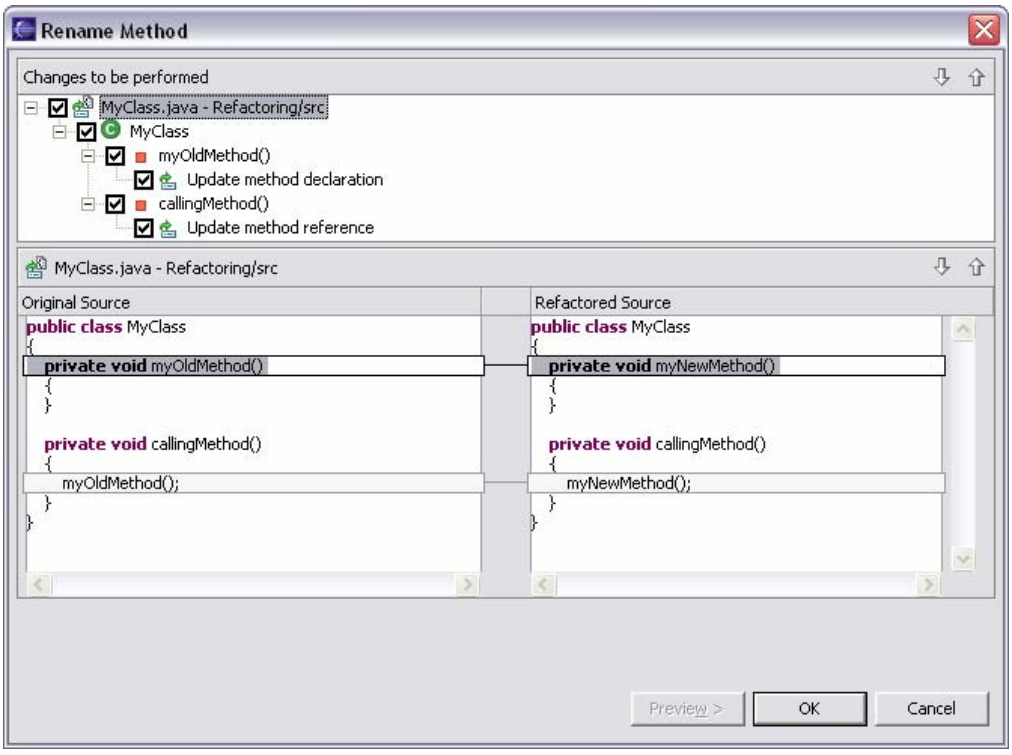
The Rename refactoring does just that, it renames a Java element. Although you can rename Java files and Java elements by hand, this will not update any references to those files and elements. You would have to search through the files in your project and replace the appropriate references by hand. There is always the chance that you might miss a reference or replace a wrong reference. The Rename refactoring will intelligently update all of the appropriate references in a project for you.

Sometimes the name of a Java element may not be clear or its function may have changed. In order to maintain the readability of the code, the name of the element should be updated. Using the Rename refactoring, it is quick and easy to rename the element and update all references to it.

To rename a Java element, select the element in either the Package Explorer view, the Outline view, or the Java source file and either select Rename from the Refactor menu or use the shortcut key, Alt + Shift + R. The Rename dialog will appear. You can then enter in the new name and select whether to update references to the element.

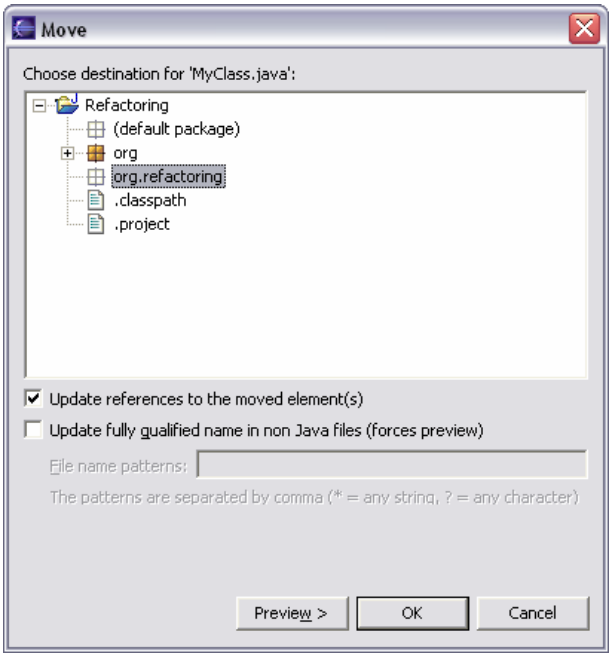


Clicking on the Preview button will bring up the preview screen where you can see and modify what changes will be made.



Move:

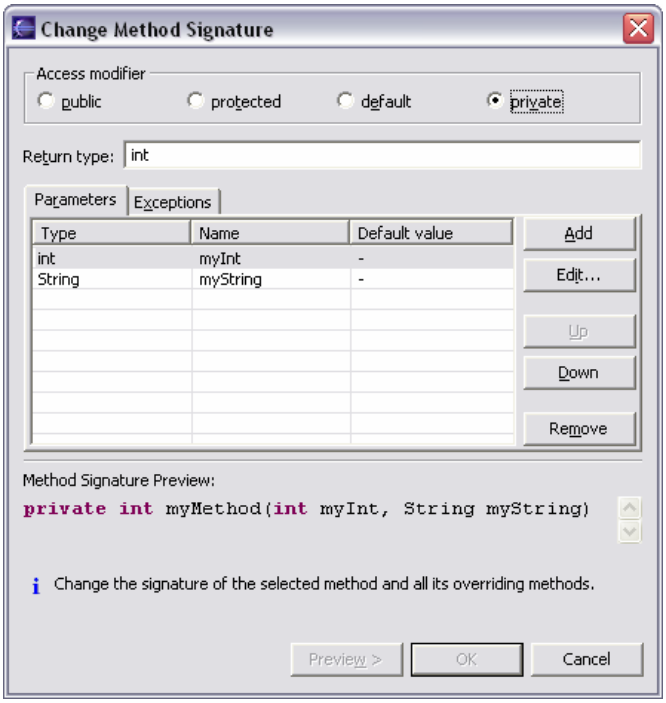
Move is very similar to Rename. It is used to move elements from one place to another. It is primarily used to move classes to different packages. To move an element, select the element, select Move from the Refactor menu or use the shortcut key, Alt + Shift + V, and select the destination in the window that appears. Again, you can click on Preview to examine the changes, or simply press OK to carry out the refactoring without previewing.



Change Method Signature:

The Change Method Signature refactoring is able to change parameter names, parameter types, the parameter order, the return type, and the method's visibility. Parameters can also be added or removed.

To change the signature of a method, select the method and select Change Method Signature from the Refactor menu. The Change Method Signature dialog will appear.

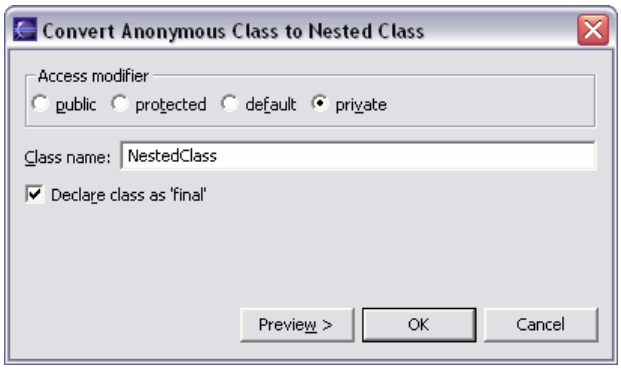


From the dialog you can set the visibility, return type, and parameters of the method. Parameters can be added, edited, moved, and removed by clicking on the buttons on the right. The default value is used when a new parameter is added. This value is then placed in the location of the new parameter in any calls to the method being changed.

Changing the signature of a method can cause problems in the method. If any problems are found, they will be flagged when you click Preview or OK.

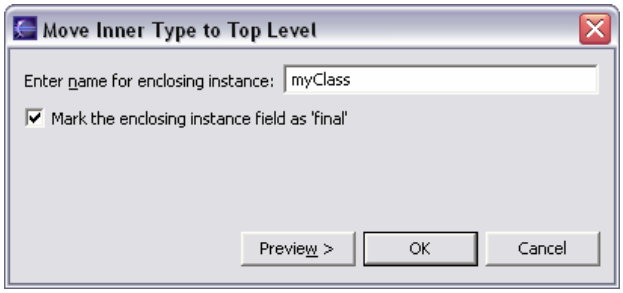
Convert Anonymous Class to Nested:

An anonymous class allows you to instantiate a class implementing an abstract class or interface without having to give it a name. This is often used in creating listeners for a user interface. When an anonymous class gets too large, it should be made into its own class. To do this, place the caret inside of an anonymous class and select Convert Anonymous Class to Nested from the Refactor menu. This opens the Convert Anonymous Class to Nested Class dialog where you can set the new name of the class and its visibility.



Convert Nested Type to Top Level (Eclipse 2 Only):

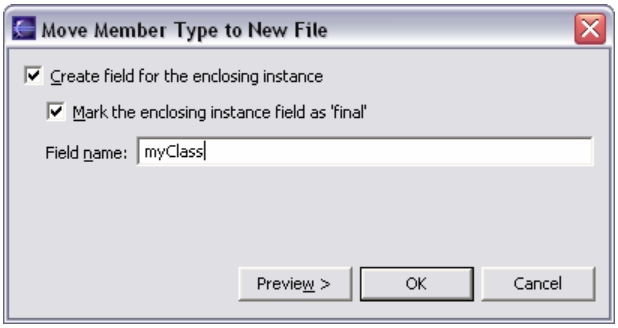
This refactoring converts a nested class into a top level class. A new Java file is created with the previously nested class in it. To convert a nested class to a top level class, select the nested class and select Convert Nested Type to Top Level from the Refactor menu. A dialog will appear asking for the name of the enclosing instance.



Because the nested class previously had access to the outer class' members, a new instance of the outer class is made to give the new class access to those members.

Move Member Type to New File (Eclipse 3 Only):

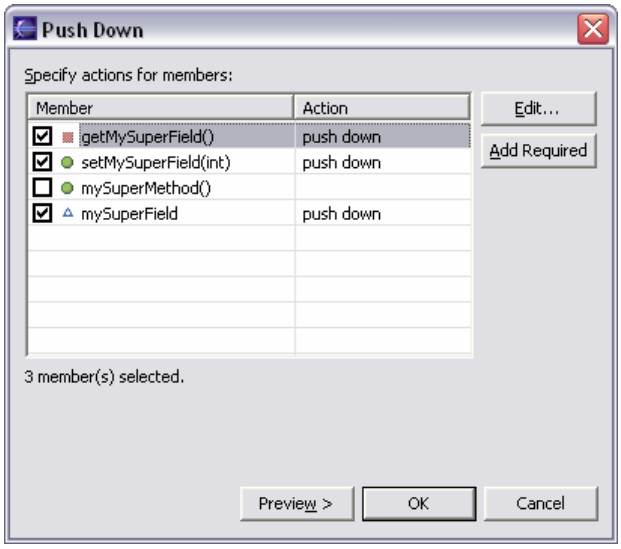
This refactoring is exactly the same as the Convert Nested Type to Top Level refactoring from Eclipse 2, except it allows you to specify whether or not you want the enclosing instance to be created.



Push Down:

The Push Down refactoring moves selected methods and fields from a class to all the classes that directly extend it. Any methods that are pushed down can optionally be left as an abstract declaration. A Push Down refactoring can be useful for restructuring the design of a project.

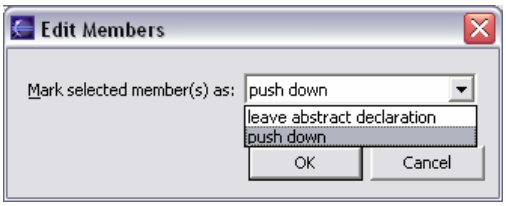
Selecting one or more methods or fields and then selecting Push Down from the Refactor menu will bring up the Push Down dialog.



Individual methods and fields can be selected from the table. All selected elements are then pushed down to all the subclasses of the current class.

When the Add Required button is clicked, any methods and fields that are found to be required by the elements already checked in the table are also checked. This will not always add every element that is required, so you should double-check to make sure that everything that is needed is checked in the table.

The Edit button is enabled when one or more methods are selected in the table in the Push Down dialog. When the Edit button is clicked, the Edit Members dialog appears. In this dialog you can choose to leave an abstract declaration in the current class for the selected method(s) or to simply remove the method declaration(s) in the current class.



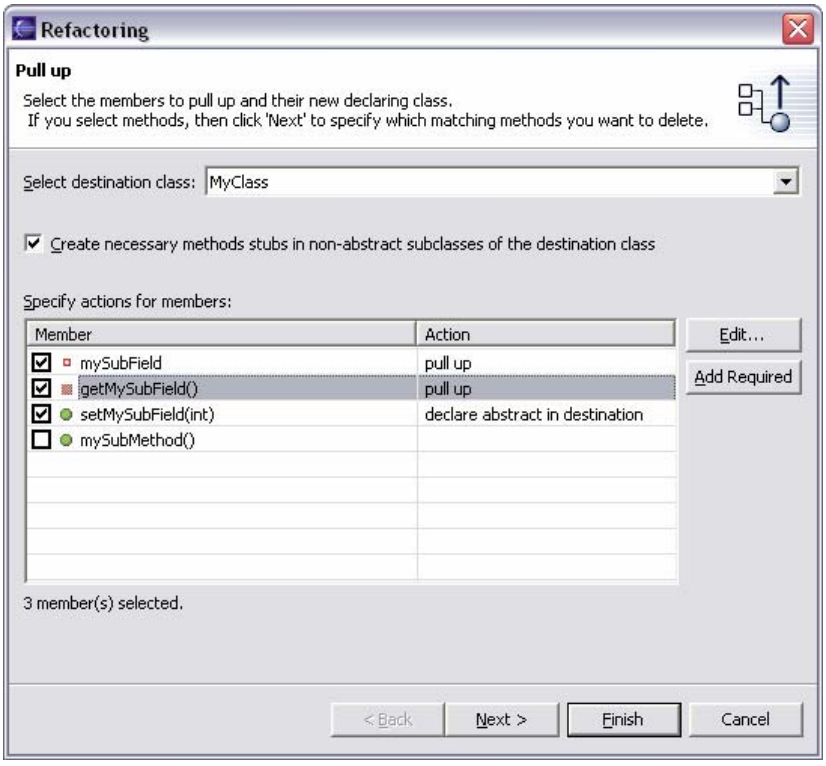
Double-clicking on a checked method in the table in the Push Down dialog will also bring up the Edit Members dialog. Clicking on a method under the Action column in the table will cause a dropdown list to appear where you can also select between “leave abstract declaration” and “push down”. Press Enter after selecting a value in the list to apply the value.

Pull Up:

The Pull Up refactoring is similar to the Push Down refactoring in that it involves moving selected methods and fields between classes. The Pull Up refactoring, however, moves methods and fields from a class to one of its superclasses. Selecting one or more methods or fields and then selecting Pull Up from the Refactor menu will bring up the Pull Up wizard.

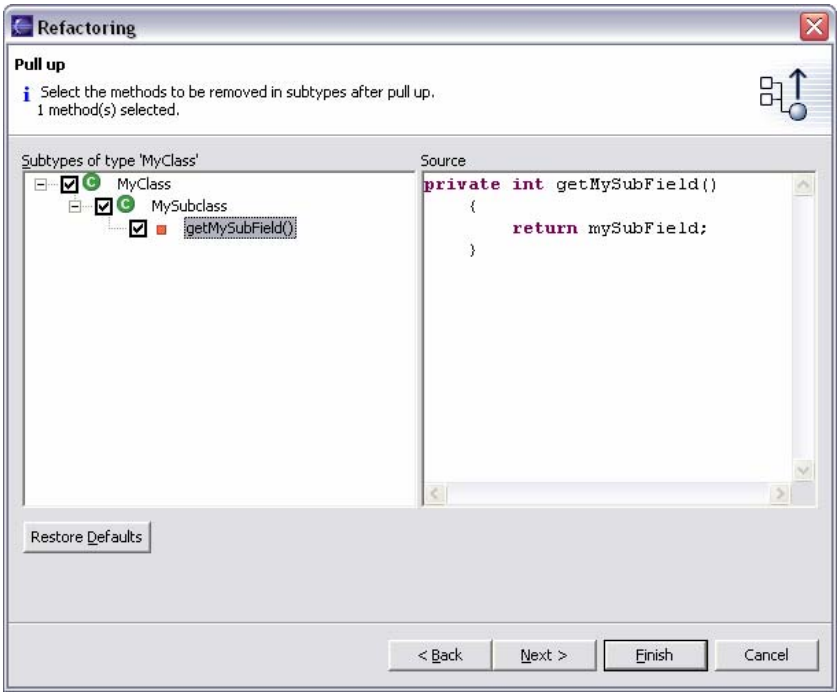
The “Select destination class” combo box contains a list of all the superclasses that the current class extends. The superclasses may be several levels removed from the current class. You are only able to pull up the selected fields and methods to one of the superclasses.

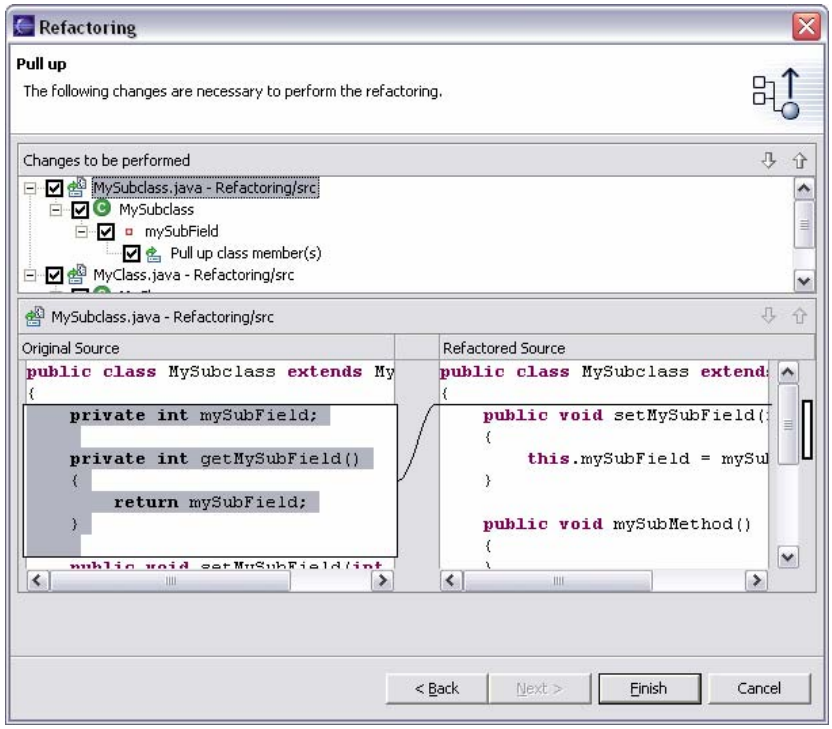
If any selected methods have their action set to “declare abstract in destination”, the “Create necessary methods in non-abstract subclasses of the destination class” check button is enabled. When this button is checked, all subclasses of the destination class who do not have or inherit the selected method that is to be declared as abstract will have method stubs for that abstract method added to them.



As in the Push Down refactoring, selecting one or more methods and clicking on the Edit button or double-clicking on a method will open the Edit Members dialog. The two options are “pull up” and “declare abstract in destination”. The “pull up” action will simply copy the members to the superclass and give the option to remove the members from the current class. The “declare abstract in destination” action will create an abstract method in the superclass, make the superclass abstract if it is not already, and leave the method in the current class. As in the Push Down refactoring, clicking in the Action column of a selected method will cause a dropdown list to appear where you can select one of the two actions. Press Enter after selecting an action in the list to apply the action.

If any selected method has its action set to “pull up”, clicking on Next will display the wizard page allowing you to select which methods you wish to remove from the current class. Only methods with their action set to “pull up” are displayed. If a method in this page is checked, it will be completely removed from the current class. Clicking on elements in the tree on the left side of the page shows the source code for that element on the right side of the page.



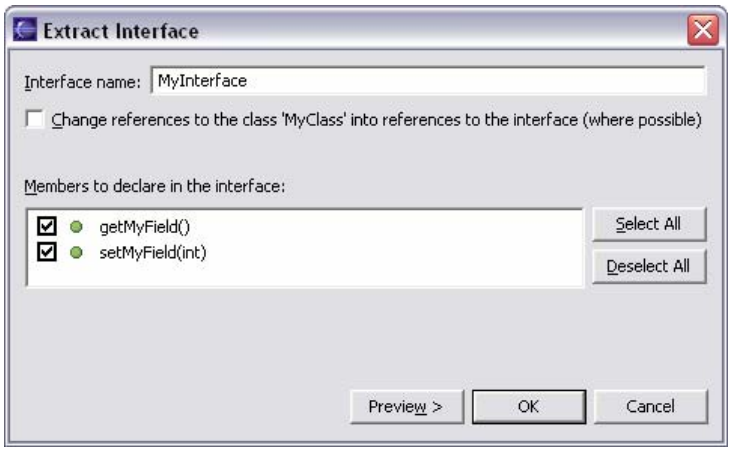


Clicking Next on the first wizard page without any methods set to “pull up”, or clicking Next on the wizard page where you can remove selected methods will bring you to the preview wizard page. This page is similar to the other preview pages and works the same way.

Clicking on the Finish button at any time during the wizard will do the refactoring with the defaults for any pages that would be displayed by clicking on the Next button.

Extract Interface:

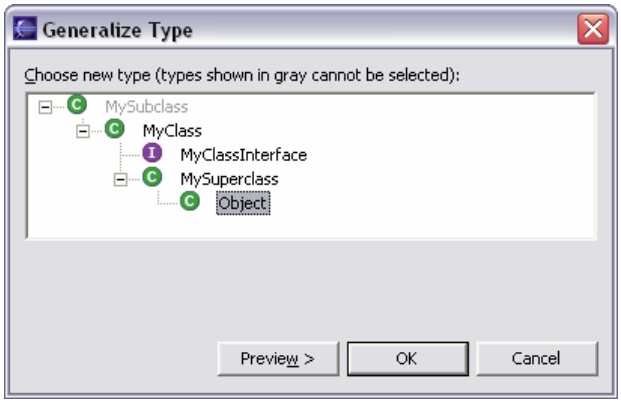
The Extract Interface refactoring allows you to create an interface from an existing class. You are able to select which methods from the current class will be included in the interface. To open the Extract Interface dialog, select a class and then select Extract Interface from the Refactor menu.



In the Extract Interface dialog, type the desired name for the interface and select the methods that you want to be in the interface. Only public methods are listed in the dialog. Checking the “Change references to the class ‘[current class name]’ into references to the interface (where possible) check button will change any references to the current class to the interface if it is able to. Click on OK to apply the refactoring.

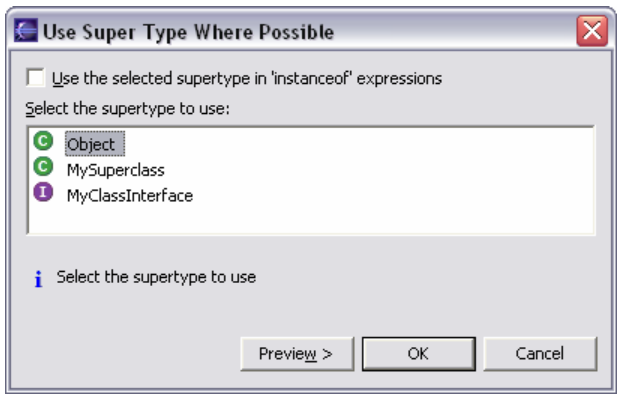
Generalize Type (Eclipse 3 Only):

The Generalize Type refactoring changes the type of an object in its declaration to one of its supertypes. Select a declaration of a variable, parameter, field, or method return type and Generalize Type from the Refactor menu. In the Generalize Type dialog, select the new type of the object and click OK to apply the refactor.



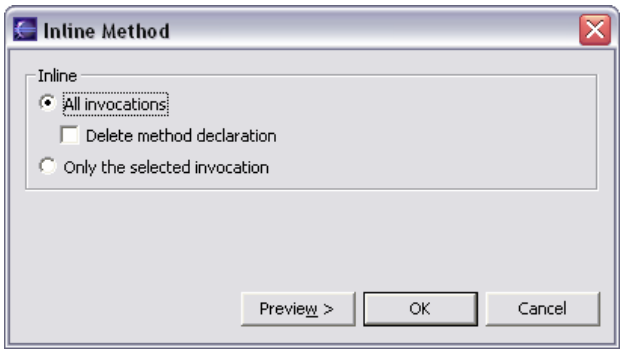
Use Supertype Where Possible:

The Use Supertype Where Possible refactoring will replace references to a certain type of object by one of its supertypes. To open the Use Super Type Where Possible dialog, select a class and then select User Supertype Where Possible from the Refactor menu. In the dialog, select the supertype to use. If the "Use the selected supertype in 'instanceof' expressions" check button is checked, instanceof expressions will also be modified during the refactoring. Eclipse 3 does not have the check button and will always modify the instanceof expressions during the refactoring.



Inline:

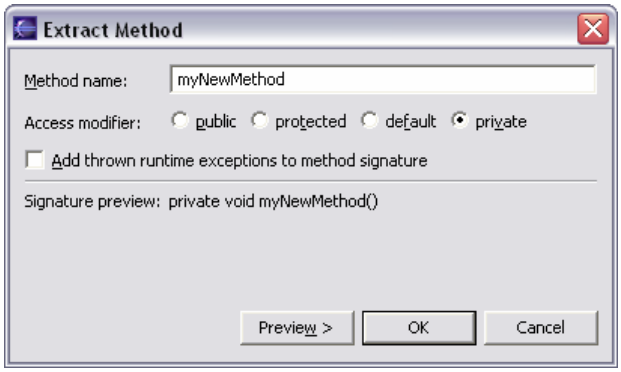
The Inline refactoring takes a reference to a method, static final field, or a local variable and replaces it with its code or value. For example, if you inline a method call, the call will be replaced by the body of code in the called method. To inline a method, static final field, or a local variable, select the element and select Inline from the Refactor menu. You can also use the shortcut key Alt + Shift + I. In the Inline dialog, you can select whether to inline "All invocations" or "Only the selected invocation". If you inline all invocations, you also have the option to delete the declaration itself. Below is shown the dialog to inline a method. Similar dialogs are available for static final fields and local variables.



Extract Method:

If a method does more than one distinct operation, is too long, or if some code in a method is used several times, it may be beneficial to extract a section of the method into its own method. It is easy to do this using the Extract Method refactoring.

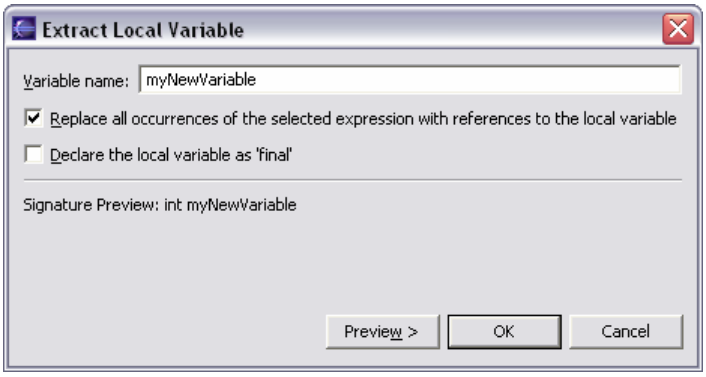
To extract a method, select the lines of code you want to extract and select Extract Method from the Refactor menu or use the shortcut key Alt + Shift + M.



In the Extract Method dialog, type in the name of the new method, select the appropriate access modifier and choose whether you want thrown runtime exceptions to be added to your method signature. The "Signature preview" line on the bottom of the dialog displays what your new method's signature will look like.

Extract Local Variable:

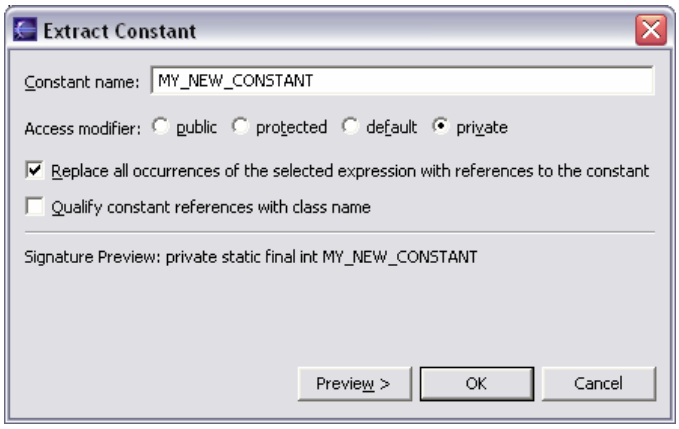
Using a variable instead of an expression can have several benefits. It may increase performance if the same expression is used in several places and it can also make the code easier to read and understand. To extract an expression to a local variable, select the expression and then select Extract Local Variable from the Refactor menu. You can also use the shortcut key Alt + Shift + L.



Type in the new variable's name and select whether you want to replace all expressions by the variable and whether you want the variable to be final. As with Extract Method, a preview of the variable's signature is displayed at the bottom of the dialog.

Extract Constant:

The Extract Constant refactoring is very similar to the Extract Local Variable refactoring. The only differences are that the Extract Constant refactoring allows you to choose what access modifier you want the constant to have and that it places the constant as a field in the class.



Introduce Parameter (Eclipse 3 Only):

Introduce Parameter creates a new parameter in a method and then replaces an instance of a field or local variable with the new parameter. To use the Introduce Parameter refactoring, select a reference to a field or a local variable in a method and then select Introduce Parameter from the Refactor menu.

Introduce Factory (Eclipse 3 Only):

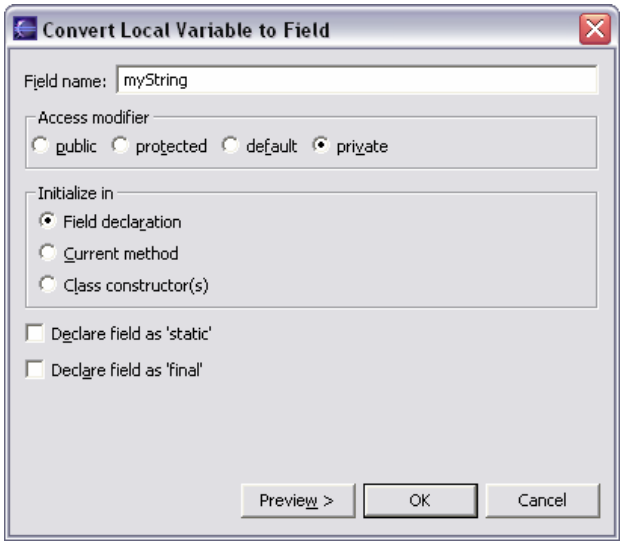
A factory is a method which creates a new object and then returns that object. You can create a factory by selecting the constructor you want to create a factory for and then selecting Introduce Factory in the Refactor menu.

In the Introduce Factory dialog, type the desired factory method name and which class you want the factory to appear in. Then choose whether you want the selected constructor to be made private.

After you click OK, a new method with the name you specified will appear in the specified class. It will create a new instance of the current class using the selected constructor. The method's return value will be the new instance it created.

Convert Local Variable to Field:

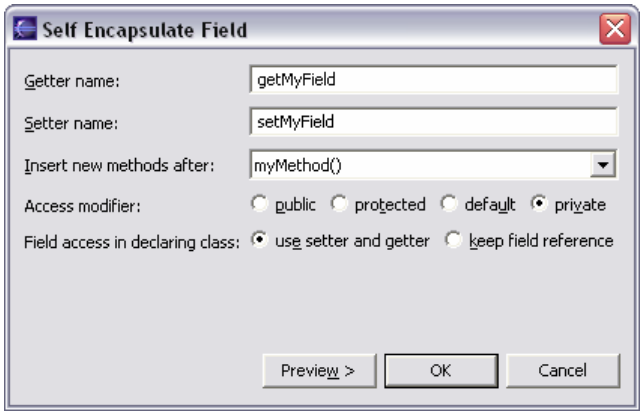
The Convert Local Variable to Field refactoring moves a declaration of a variable from inside a method to the top of the class where it is then visible to the entire class. Select a local variable and then select Convert Local Variable to Field in the Refactor menu to bring up the Convert Local Variable to Field dialog.



In the Convert Local Variable to Field dialog, type in the new name for the field, select its access modifier and specify where the field is to be initialized. The "Initialize in" radio buttons are only available if the variable is initialized on creation. The "Declare field as 'static'" and "Declare field as 'final'" check buttons are enabled depending on where you choose to initialize the field.

Encapsulate Field:

To use proper object orientated coding practices, you should make your fields private and then create accessors and mutators to access the fields. It can take a bit of time to do this mundane task. It is easy to create the accessors and mutators of a field using the Encapsulate Field refactoring. Simply select a field or a reference to a field and then select Encapsulate Field from the Refactor menu.



In the Self Encapsulate Field dialog, type in the desired getter and setter method names and then select the method where you want the new methods to appear after. Different access modifiers may be available depending on what the access modifier of the field is. Applying the Encapsulate Field refactoring will create two new methods, set the access modifier of the field to private, and will change references to the field to references to the new methods.

List of Refactorings:

Below is a table of all the refactorings, the different types of Java elements each refactoring is available on, and any shortcut keys for the refactorings. Information from the table was taken from the Eclipse Java help.

Name	Element**s Refactoring is Available On**	**Keyboard Shortcut**
Undo	Can be performed after a refactoring.	Alt + Shift + Z
Redo	Can be performed after an undo of a refactoring.	Alt + Shift + Y
Rename	Is available on methods, fields, local variables, method parameters, objects, classes, packages, source folders, and projects.	Alt + Shift + R
Move	Is available on one instance method (which can be moved to a component), one or more static methods, static fields, objects, classes, packages, source folders and projects.	Alt + Shift + V
Change Method Signature	Is available on methods.	
Convert Anonymous Class to Nested	Is available on anonymous inner classes.	
Convert Nested Type to Top Level (Eclipse 2)	Is available on nested classes.	
Move Member Type to New File (Eclipse 3)	Is available on nested classes.	
Push Down	Is available on one or more methods and fields declared in the same class.	
Pull Up	Is available on one or more methods, fields and nested classes declared in the same class.	
Extract Interface	Is available on classes.	
Generalize Type (Eclipse 3)	Is available on declarations of an object.	

Use Supertype Where Possible	Is available on classes.	
Inline	Is available on methods, static final fields and local variables.	Alt + Shift + I
Extract Method	Is available on selections of code.	Alt + Shift + M
Extract Local Variable	Is available on selections of code that can be resolved to local variables.	Alt + Shift + L
Extract Constant	Is available on static final fields and selections of code that can be resolved to static final fields.	
Introduce Parameter (Eclipse 3)	Is available on references to fields and local variables in a method.	
Introduce Factory (Eclipse 3)	Is available on constructors.	
Convert Local Variable to Field	Is available on local variables.	
Encapsulate Field	Is available on fields.	

Summary:

This tutorial described the refactorings currently available in Eclipse. These refactorings are easy to use and can make restructuring your code easier and safer. They can also help you generate code in order to increase your productivity. By using the automatic refactoring tools included with Eclipse, you will have a more enjoyable coding experience.

Note: Applying some refactorings that modify the structure of classes such as Push Down and Pull Up may not modify the other classes in a project. In this case you will have to make sure that all references to the modified elements are updated. This is why a good test suite is needed, although you may have to update references in the test classes as well. An example of this is using the Push Down refactoring on a method. If another class has an instance of the class that originally had the method and makes a call to the method, after the Push Down refactoring there will be an error in the calling class because the method is no longer found in the original class.