



DevelopIntelligence
Developing Developers

Refactoring with IDEA IntelliJ

*A DevelopIntelligence Publication
Written By: Kelby B. Zorgdrager*

Refactoring with IntelliJ IDEA 7.X

Version 1.0.1

Publication date: Feb 29, 2008

This material is licensed by DevelopIntelligence LLC and shall not be reproduced, edited, or distributed, in hard or soft copy format, without the written consent of DevelopIntelligence.

Information in this document is subject to change without notice. Companies, names, and data used in the examples herein are fictitious unless otherwise noted.

Refactoring with IntelliJ IDEA 7.X is a publication of DevelopIntelligence, LLC. For more information regarding this publication or others, please contact us via email info@DevelopIntelligence.com, or by phone (303) 395-5340.

Java, Java Standard Edition, Java SE, Java Enterprise Edition, Java EE, Enterprise JavaBeans, and all other Java-based trademarks and logo trademarks are registered trademarks of Sun Microsystems, Inc., in the United States and other countries. All other products referenced herein are trademarks of their respective holders.

Copyright 2008 DevelopIntelligence LLC. All rights reserved.

Refactoring

Section Overview

This section focuses on refactoring capabilities found within IntelliJ, including the advanced refactoring support provided by RefactorX.

Section Assumption

- Familiarity with refactoring
- Familiarity navigating IntelliJ

Learning Objectives

- Understand how refactoring supports code maturity
- Apply some common refactoring concepts to a project

Test Driven Development

Test-driven development (TDD) is a software development paradigm which focuses on software quality through testing. TDD relies on two primary concepts: automated testing and unit testing.

In TDD, test cases are written before the actual program source code. Only when an automated unit test fails, either during compilation or execution, is new program code added.

History of Refactoring

Over the years, refactoring has grown in popularity. Partially due its significance in the build lifecycle to create maintainable code. But more recently, it has grown in popularity as a result of test driven development.

The test-driven development prescribes two workflows: the first workflow is aimed at creating new code and returning the software to a stable state, namely with all tests passing. The second workflow, *the refactoring workflow* is aimed at eliminating duplication and improving the implementation.

Overview of Refactoring

In general, a refactoring is any change to an implementation that maintains the semantic of the computation, but that changes the internal structures or organization. In practice, however, refactoring is an activity that impacts implementations, possibly across multiple modules.

There are a few different ways to perform a refactoring. The most historical way, was to perform a global search and replace across your source tree. As effective as this may be, it is also error prone. Today, the preferred method of refactoring is to leverage a tool that manages the changes, ensuring the source code maintains its stable state after the refactoring has been completed.

Even though tools, like IntelliJ, have robust refactoring capabilities, it is still a good practice to adopt the TDD approach to refactoring. The TDD approach to refactoring requires that you have enough test coverage to guarantee your code is in the same stable state before and after the refactoring. If after refactoring, a test fails, you know some portion of the refactoring did not succeed as expected. Of course, the local history within IntelliJ will help you roll back to a former stable state if it is required. Keep this in mind while making changes in multiple modules.

TDD and Refactoring

TDD only helps with refactoring if there are a comprehensive list of tests. If features are not tested, then invalid refactorings to those features will not be validated. For this reason, it is very important that you take note of any tests that you think are missing and stay disciplined about adding them to the test suite.

Refactoring Catalogue

The current thought leader on refactoring is Martin Fowler. As part of his work on refactoring, Fowler has created a refactoring catalogue, much like the catalogues associated with design patterns. In fact, many of the refactorings found in his catalogue are included within IntelliJ.

Refactor Preview

The refactor preview option will allow you to examine the impact of the refactor before performing the action. In essence, the preview allows you to determine the dependent pieces effected by the refactor.

Refactoring within IntelliJ

IntelliJ supports over 45 different refactoring techniques across both general Java source code and XML source code. Like most things within IntelliJ, the refactoring capabilities are context sensitive. In general, IntelliJ supports refactoring across:

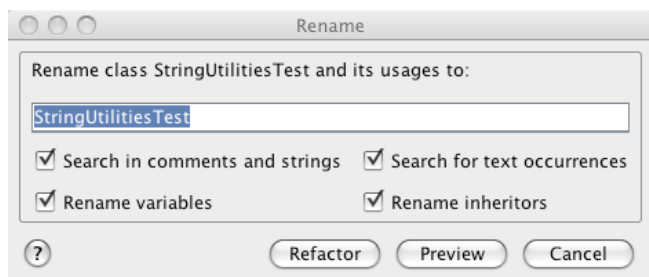
- Packages - moving classes to new packages, renaming packages, etc.
- Classes - modifying the structure of classes through renaming, creating inheritance, etc.
- Member - adjusting definitions by introducing variables, in-lining variables, etc.
- Methods - modifying both declarations and definitions by changing signatures, extracting methods, replacing methods with design patterns, etc.

Though each one of the refactorings are interesting, the basic behavior and outcome of a refactoring effort within IntelliJ is relatively consistent. Simply select (either select a block or move the cursor within a keyword), right-click, and select the refactoring from the Refactor menu.

Once the refactoring has been selected, in most cases a refactor dialog will appear allowing you to define the parameters governing the refactoring effort.

Renaming a Class

It's not uncommon during a development effort to rename a class. Since most people adopt the Java convention where the file name mirrors the classname, renaming a class is not necessarily as easy as invoking a global search and replace. Within IntelliJ, renaming a class is a relatively straightforward process: select the class to rename, select the rename option from the Refactor menu, and complete the refactor dialog.



Locating Duplicates

You can locate duplicates within your source code through the Analyze menu.

Select the Locate Duplicates option from the Analyze menu. If duplicates are found, they will be displayed in a duplicates pane.

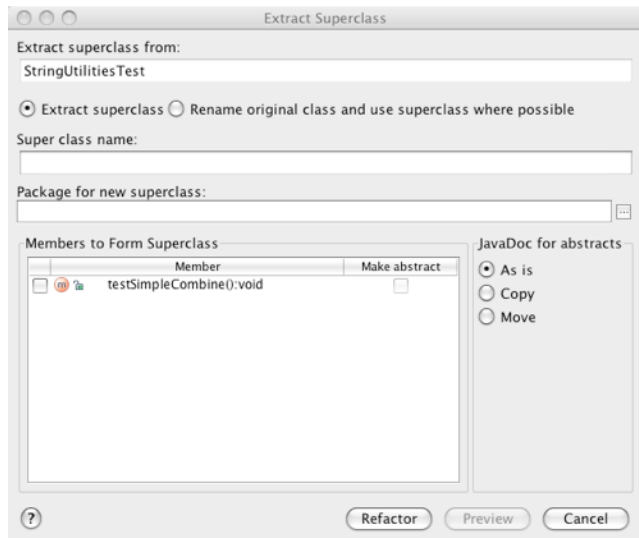
Refactor Options

- Change Class Signature
- Change Method Signature
- Convert Anonymous to Inner
- Convert to Instance Method
- Copy/Clone Class
- Encapsulate Fields
- Extract Includes
- Extract Interface
- Extract Method
- Extract Superclass
- Generify
- Inline
- Introduce Constant
- Introduce Field
- Introduce Parameter
- Introduce Variable
- Invert Boolean
- Make Class Static
- Make Method Static
- Migrate
- Move
- Pull Members Up
- Push Members Down
- Rename
- Replace Constructor with Factory Method
- Replace Inheritance with Delegation
- Replace Method Code Duplicates
- Replace Temp with Query
- Safe Delete
- Use Interface where Possible

In some cases, simply renaming the class is not sufficient. You may actually want to rename the class and move it to a new package location. The move refactor allows you to relocate the class, and its related source code file in one action.

Creating Inheritance

As the complexity of an application grows, it is worthwhile to analyze the duplicate source code within your project. If and when you find duplication source code, it may make sense to refactor out the duplication through inheritance. If the classes are already part of joined inheritance chain, you could extract a method, and then move the method to a parent class. If the classes aren't part of an inheritance chain, you could extract either an interface or a superclass and select the method(s) you wish to appear in the new parent.



Safe Delete

Deleting something can be one of the most problematic refactoring exercises. To ease the delete refactoring, IntelliJ provides Safe Delete. Safe Delete will determine the dependencies and allow you to address them before deleting anything, whether its a variable or a file.