

The way to good code

Refactoring from handcraft to machines

Leo Büttiker und Mirko Stocker

August 14, 2007

And not about silly star wars toys!



Agenda

- 1 Introduction
- 2 Bad Code
- 3 Refactoring
- 4 Thanks and Questions

Who we are

Mirko Stocker

Leo Büttiker

Studied Computer Science at the HSR from 2003-2007.
Term Projects and Diploma Theses supervised by Peter
Sommerlad:

Ruby Refactoring Plug-In for
RDT

C++ Refactoring Plug-In for
CDT



INSTITUTE
FOR
SOFTWARE



Bad Code – What's that?

- Not running code
- Nice Weather Code – runs only when the sun is shining
- Scary Code that probably doesn't even exist anymore or code that runs and nobody knows why
- Minefield Code: Not tested, hard to understand, hell to work with, connected everywhere, complicated
- Chinese Code: (probably) not so bad code per se, but mysterious naming, too many comments
- And sometimes all together

What the heck does that mean?

- Make code more readable, because readable code is:
 - easier to maintain
 - easier to test
 - easier to spot bugs
- but it doesn't change the behaviour:
 - Refactoring doesn't fix bugs
 - Refactoring doesn't make your code faster

How to get rid of bugs

- Not done by refactorings
- Read about:
 - Unit Testing / Code-Coverage
 - Continuous Integration
 - Source Metrics
 - Static Code Analysis
 - Bug Tracking
 - IDE
 - Performance-Analysis
 - Design Patterns



What you need before you start refactoring

- Version Controlling
 - Developing code without VC is like bungee jumping without a rope!
- Automated Unit Tests
 - Necessary for all bigger refactorings
- Cool IDE
 - Not mandatory, but makes your work easier

Examples of Refactorings

- Rename Variable (or other stuff)
- Move Field
- Remove Parameter

R <u>e</u> name...	Shift+Alt+R
M <u>o</u> ve...	Shift+Alt+V
<u>C</u> hange Method Signature...	Shift+Alt+C
E <u>x</u> tract Method...	Shift+Alt+M
E <u>x</u> tract L <u>o</u> cal Variable...	Shift+Alt+L
E <u>x</u> tract C <u>o</u> nstant...	
I <u>n</u> line...	Shift+Alt+I
<u>C</u> onvert Anonymous Class to Nested...	
<u>C</u> onvert Member Type to Top Level	
<u>C</u> onvert Local Variable to Field...	
E <u>x</u> tract Superclass...	
E <u>x</u> tract Interface...	
Use Supertype Where Possible...	
P <u>u</u> sh D <u>o</u> wn...	
P <u>u</u> ll <u>U</u> p...	
I <u>n</u> troduce I <u>n</u> direction...	
I <u>n</u> troduce F <u>a</u> ctory...	
I <u>n</u> troduce P <u>a</u> rameter...	
E <u>n</u> capsulate F <u>e</u> ld...	
<u>G</u> eneralize Declared Type...	
I <u>n</u> fer <u>G</u> eneric Type Arguments...	
M <u>i</u> grate JAR File...	
<u>C</u> reate S <u>c</u> ript...	
A <u>p</u> ply S <u>c</u> ript...	
<u>H</u> istory...	

Refactorings - Concrete Example: Rename Local Variable

RENAME LOCAL VARIABLE

The following changes are necessary to perform the refactoring.

Changes to be performed

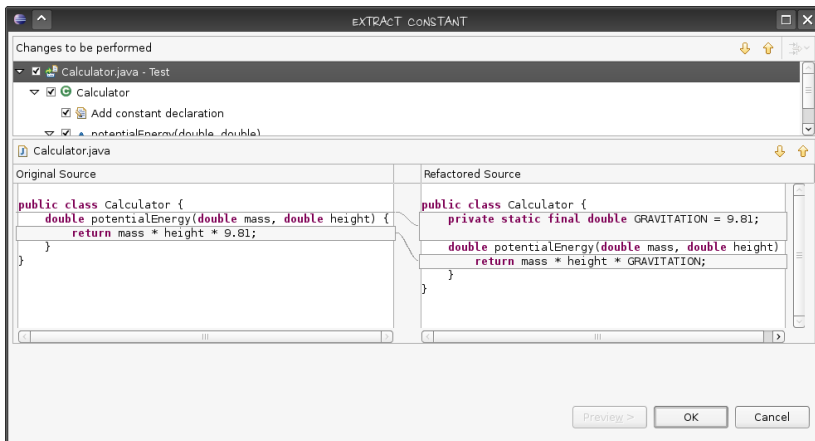
/Mongrel/lib/mongrel.rb

mongrel.rb

Original Source	Refactored Source
<pre>def read_body(remain, total) begin # write the odd sized chunk first @params.http_body = read_socket(remain % Const::CHU </pre>	<pre>def read_body(rest, total) begin # write the odd sized chunk first @params.http_body = read_socket(rest % Const::CHU </pre>
<pre> remain -= @body.write(@params.http_body) </pre>	<pre> rest = rest - @body.write(@params.http_body) </pre>
<pre> update_request_progress(remain, total) </pre>	<pre> update_request_progress(rest, total) </pre>
<pre> # then stream out nothing but perfectly sized chunks until remain <= 0 or @socket.closed? # ASSUME: we are writing to a disk and these writes alwa @params.http_body = read_socket(Const::CHUNK_SIZE) remain -= @body.write(@params.http_body) </pre>	<pre> # then stream out nothing but perfectly sized chunks until rest <= 0 or @socket.closed? # ASSUME: we are writing to a disk and these writes @params.http_body = read_socket(Const::CHUNK_S rest = rest - @body.write(@params.http_body) </pre>
<pre> update_request_progress(remain, total) end rescue Object STDERR.puts "ERROR reading http body: #{\$!}" \$!.backtrace.join("\n") end</pre>	<pre> update_request_progress(rest, total) end rescue Object STDERR.puts "ERROR reading http body: #{\$!}" \$!.backtrace.join("\n") end</pre>

< Back Next > Finish Cancel

Refactorings -Concrete Example: Extract Constant



Refactorings -Concrete Examples: Extract Method

EXTRACT METHOD

Changes to be performed

- Printer
 - printOwing()
 - Substitute statement(s) with call to printDetails
 - Create new method 'printDetails' from selected statement(s)

Printer.java

Original Source	Refactored Source
<pre>private String getOutstanding() { return "5"; } void printOwing() { printBanner(); //print details System.out.println("name :" + name); System.out.println("amount:" + getOutstanding()); } }</pre>	<pre>void printOwing() { printBanner(); //print details printDetails(); } private void printDetails() { System.out.println("name :" + name); System.out.println("amount:" + getOutstanding()); } }</pre>

Preview > OK Cancel

Refactoring by M. Fowler

154

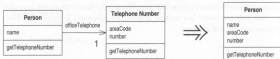
MOVING FEATURES BETWEEN OBJECTS

Inline Class

A class isn't doing very much.

Move all its features into another class and delete it.

Inline Class



Motivation

Inline Class is the reverse of *Extract Class* (149). I use *Inline Class* if a class is no longer pulling its weight and shouldn't be around any more. Often this is the result of refactoring that moves other responsibilities out of the class so there is little left. Then I want to fold this class into another class, picking one that seems to use the runt class the most.

Mechanics

- Declare the public protocol of the source class onto the absorbing class. Delegate all these methods to the source class.
 - ➔ *If a separate interface makes sense for the source class methods, use Extract Interface (341) before inlining.*
- Change all references from the source class to the absorbing class.
 - ➔ *Declare the source class private to remove out-of-package references. Also change the name of the source class so the compiler catches any dangling references to the source class.*
- Compile and test.
- Use *Move Method* and *Move Field* to move features from the source class to the absorbing class until there is nothing left.
- Hold a short, simple funeral service.

INLINE CLASS

155

Example

Because I made a class out of telephone number, I now inline it back into person. I start with separate classes:

```
class Person...
public String getName() {
    return _name;
}
public String getTelephoneNumber() {
    return _officeTelephone.getTelephoneNumber();
}
TelephoneNumber getOfficeTelephone() {
    return _officeTelephone;
}

private String _name;
private TelephoneNumber _officeTelephone = new TelephoneNumber();
```

```
class TelephoneNumber...
public String getTelephoneNumber() {
    return ("(" + _areaCode + ") " + _number);
}
String getAreaCode() {
    return _areaCode;
}
void setAreaCode(String arg) {
    _areaCode = arg;
}
String getNumber() {
    return _number;
}
void setNumber(String arg) {
    _number = arg;
}
private String _number;
private String _areaCode;
```

I begin by declaring all the visible methods on telephone number on person:

```
class Person...
String getAreaCode() {
    return _officeTelephone.getAreaCode();
}
void setAreaCode(String arg) {
    _officeTelephone.setAreaCode(arg);
}
String getNumber() {
    return _officeTelephone.getNumber();
}
```

Inline Class

Refactoring made easy

- It's not that complicated
- Mostly straight forward
- Use an IDE
 - Less oversights and mistakes
 - Much faster than manual search-and-replace

Tools

- Eclipse:
 - Java
 - Ruby
 - C++
 - Python
 - JavaScript (coming soon)
 - Groovy (coming soon)
- Visual Studio
 - Some C# Refactorings
 - Lots of third party tools
- NetBeans
 - Some Java refactorings, fewer than Eclipse
 - Rename Local Variable for Ruby
- PHP
 - nothing really worth to mention, yet

Live Example in Java

Why doesn't my IDE support refactoring?

- Cause your IDE sucks!
- Problems:
 - Understanding the code (depends on the language)
 - Comments (to which code does a comment belong?)
 - Language dependent
 - Dynamic typing (Ruby, PHP, Python, ...)
 - Complexity (C++ and templates)
 - Language “features”:
 - Direct memory access (C/C++)
 - Embedding other languages (Macros, Inline ASM, JNI)
 - Reflection
 - C++ Friends

Solution

- Refactorings don't need to be 100% bullet proof
 - You (hopefully) have unit tests to catch problems
- IDE-Integration
 - Writing your own IDE → bad idea
 - Write as plug-in ...
 - ... but most times not enough (e.g. changes to AST are needed).
 - Eclipse is open source (although its not that easy to get your code into it)!

Our Contributions

- Ruby Refactoring Plug-ins for Eclipse RDT
 - About 15 refactorings for Ruby, Rails support in work
 - Integrated into Aptana's Web-IDE (former RadRails)
- C++ Refactoring Plug-ins for Eclipse CDT
 - 6 refactorings
 - Currently under development at the IFS

Thanks and Questions

