

Refactoring

Lecture 2

DAT159/H18
Volker Stolz

**Supported by the bilateral SIU/CAPES
project “Modern Refactoring” 2017/18**

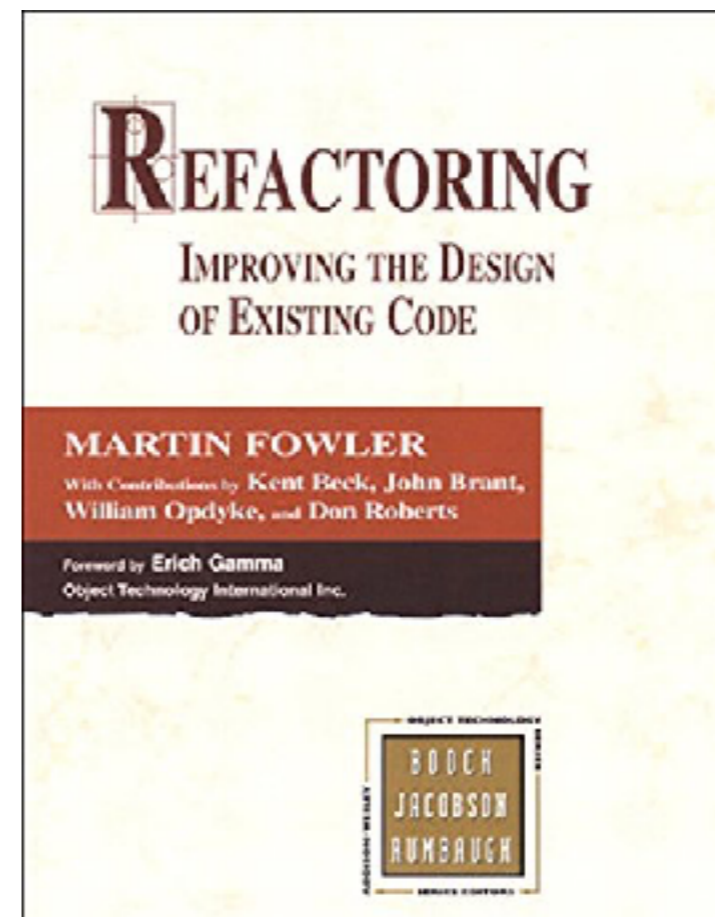


**Høgskulen
på Vestlandet**

OO Refactorings

Fowler has a *pretty loooong list* of refactorings in his book:

- ▶ Add Parameter
- ▶ Change Bidirectional Association to Unidirectional
- ▶ Change Reference to Value
- ▶ Change Unidirectional Association to Bidirectional
- ▶ Change Value to Reference
- ▶ Collapse Hierarchy
- ▶ Consolidate Conditional Expressions
- ▶ Consolidate Duplicate Conditional Fragments
- ▶ Decompose Conditional
- ▶ Distill Observed Data
- ▶ Dynamic Method Definition
- ▶ Eagerly Initialized Attribute
- ▶ Encapsulate Collection
- ▶ Encapsulate Downcast
- ▶ Encapsulate Field
- ▶ Extract Class
- ▶ Extract Interface
- ▶ Extract Method
- ▶ Extract Module
- ▶ Extract Subclass
- ▶ Extract Superclass
- ▶ Extract Surrounding Method
- ▶ Extract Variable
- ▶ Form Template Method
- ▶ Hide Delegate
- ▶ Hide Method
- ▶ Inline Class
- ▶ Inline Method
- ▶ Inline Module
- ▶ Inline Temp
- ▶ Introduce Assertion
- ▶ Introduce Class Association
- ▶ Introduce Expression Builder
- ▶ Introduce Foreign Method
- ▶ Introduce Gateway
- ▶ Introduce Local Extension
- ▶ Introduce Named Parameter
- ▶ Introduce Null Object
- ▶ Introduce Parameter Object
- ▶ Isolate Dynamic Receiver
- ▶ Locally Initialized Attribute
- ▶ Move Deal from Runtime to Parse Time
- ▶ Move Field
- ▶ Pull Up Constructor Body
- ▶ Pull Up Field
- ▶ Pull Up Method
- ▶ Push Down Field
- ▶ Push Down Method
- ▶ Recombine Conditional
- ▶ Remove Assignments to Parameters
- ▶ Remove Control Flag
- ▶ Remove Middle Man
- ▶ Remove Named Parameter
- ▶ Remove Parameter
- ▶ Remove Setting Method
- ▶ Remove Unused Default Parameter
- ▶ Remove Method
- ▶ Replace Abstract Dependence with Module
- ▶ Replace Array with Object
- ▶ Replace Conditional with Polymorphism
- ▶ Replace Constructor with Factory Method
- ▶ Replace Data Value with Object
- ▶ Replace Delegate with Hierarchy
- ▶ Replace Delegation with Inheritance
- ▶ Replace Dynamic Encapsator with Dynamic Method Definition
- ▶ Replace Error Code with Exception
- ▶ Replace Exception with Test
- ▶ Replace Hash with Object
- ▶ Replace Inheritance with Delegation
- ▶ Replace Loop with Collection Closure Method
- ▶ Replace Magic Number with Symbolic Constant
- ▶ Replace Method with Method Object
- ▶ Replace Nested Conditional with Guard Clauses
- ▶ Replace Parameter with Explicit Methods
- ▶ Replace Parameter with Method
- ▶ Replace Return with Data Class
- ▶ Replace Subclass with Fields
- ▶ Replace Temp with Chain



<https://refactoring.com/catalog/>

Some Refactorings In Detail

From Fowler's catalog (<https://refactoring.com/catalog/>):

- Simple refactorings:
 - Extract Method
 - Inline Method
 - Extract Variable
- OO refactorings:
 - Move Method
 - Pull Up Field/Method
 - Push Down Field/Method
 - Extract Class
 - Replace Type Code With Polymorphism

Example Input

- Fowler's Movie Store:
<https://github.com/selabhvl/refactoring-fowler>
- Individual examples:
<https://github.com/selabhvl/dat159-refactoring>

Extract Method

- Most frequently used refactoring
- Give name to fragment of code
- Reduces size of method
- Example:

```
void printOwing() {
    printBanner();

    //print details
    System.out.println ("name: " + _name);
    System.out.println ("amount " + getOutstanding());
}
```

```
void printOwing() {
    printBanner();
    printDetails(getOutstanding());
}

void printDetails (double outstanding) {
    System.out.println ("name: " + _name);
    System.out.println ("amount " + outstanding);
}
```

- Question: Which lines?
Parameters?

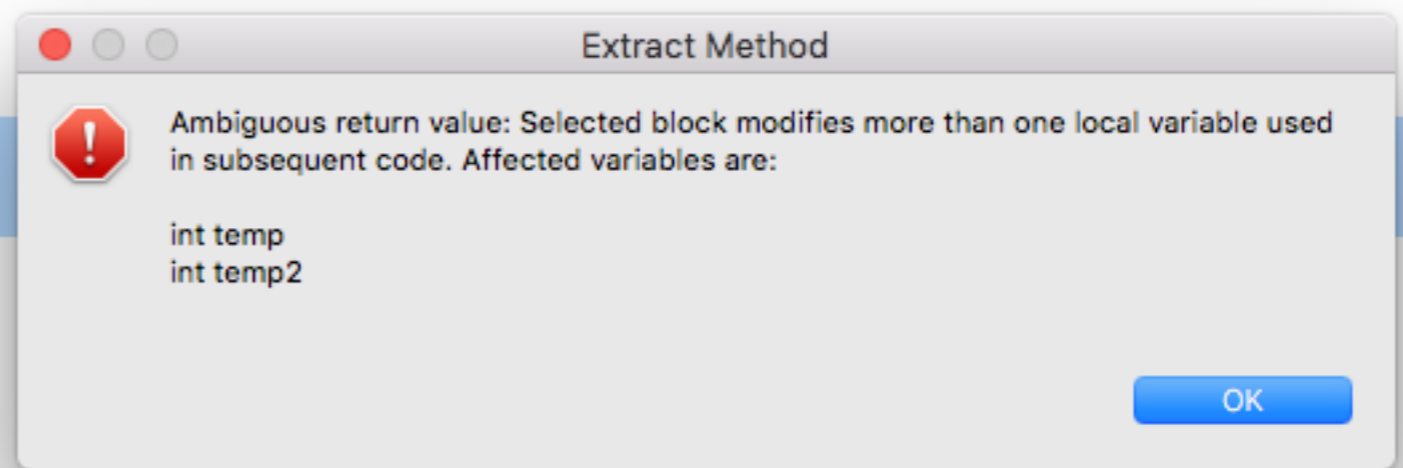
Extract Method:

Detailed Steps [Fowler, 110]

1. Create a new method with proper name.
2. Copy extracted code from source into new method.
3. Identify variables that are local in scope to the source method: add as parameters.
4. Identify and declare temporary variables in extracted code.
5. Does the extracted code modify variables in the local scope? If just one, implement function with return value; but if multiple -> extraction not possible!
6. *Compile!*
7. Replace extracted extracted code with call to new method and right parameters.
8. Compile; eliminate dead code in source method.

Extract Method: Examples

```
void nope() {  
    System.out.println("The output is:");  
    int temp = fields_are_easy+1;  
    int temp2 = fields_are_easy+1;  
    System.out.println(getF());  
  
    // Avoid dead code above:  
    System.out.println(temp);  
    System.out.println(temp2);  
}
```



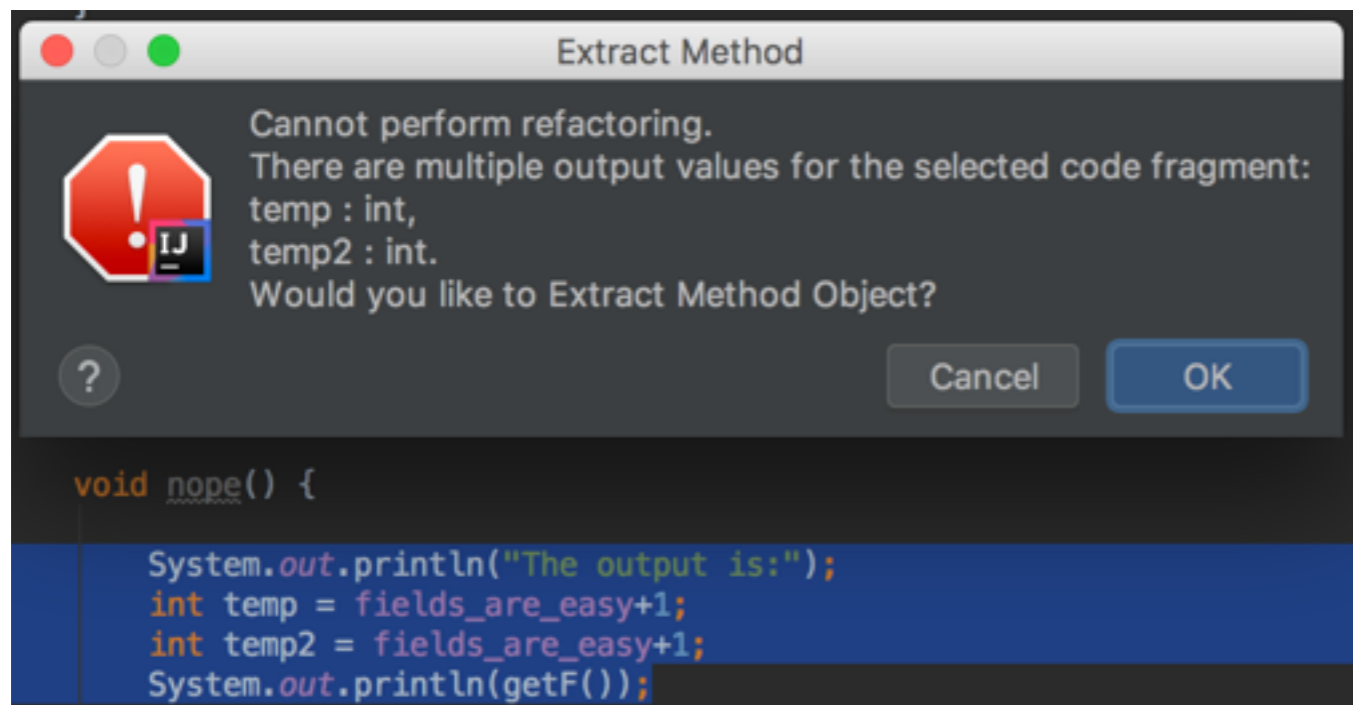
```
void localVariable() {
```

```
s:");
```

```
void nope() {
```

```
    System.out.println("The output is:");  
    int temp = fields_are_easy+1;  
    int temp2 = fields_are_easy+1;  
    System.out.println(getF());
```

```
    // Avoid dead code above:  
    System.out.println(temp);  
    System.out.println(temp2);  
}
```



Inline Method

- Reverts effect of Extract Method
- Useful to get rid of very simple code fragments only
- Exercise: Maybe enable other refactorings (combine with upcoming Extract Variable)
- Exercise: “Breaking Things” (see below)
- Need to consider class hierarchy

Extract Variable

Break apart complicated expressions:

```
if ( (platform.toUpperCase().indexOf("MAC") > -1) &&
      (browser.toUpperCase().indexOf("IE") > -1) &&
      wasInitialized() && resize > 0 )
{
    // do something
}
```

```
final boolean isMacOs    = platform.toUpperCase().indexOf("MAC") > -1;
final boolean isIEBrowser = browser.toUpperCase().indexOf("IE") > -1;
final boolean wasResized = resize > 0;

if (isMacOs && isIEBrowser && wasInitialized() && wasResized)
{
    // do something
}
```

Refactoring: Extract Local Variable

| | Before | After |
|---|----------------------------------|----------------------------------|
| 1 | <code>public void f() {</code> | <code>public void f() {</code> |
| 2 | <code> a.b.c.d.m();</code> | <code> D temp = a.b.c.d;</code> |
| 3 | <code> a.b.c.d.n();</code> | <code> temp.m();</code> |
| 4 | <code> a.b.foo(a.b.c.d);</code> | <code> temp.n();</code> |
| 5 | <code> a.b.bar();</code> | <code> a.b.foo(temp);</code> |
| 6 | <code> a.b.c.d.m();</code> | <code> a.b.bar();</code> |
| 7 | <code>}</code> | <code> temp.m();</code> |
| | | <code>}</code> |

Compute complex (expensive) expression only once.

Extract Local Variable: Formally

input : e – an expression of non-void type E
 : S – a selection, as a list of consecutive statements
 : $context$ – the outermost, non-type scope containing S
output: $context$ with e extracted to a local variable in S

- 1 $v \leftarrow$ fresh variable name;
- 2 **for** $s \in S$ **do**
- 3 | in s replace all occurrences of e with v ;
- 4 **end**
- 5 add a new variable declaration $E v = e context$ just before S ;

Master thesis A. Eilertsen, 2016

Extract Variable (2)

Not as easy as it seems:

```
int plusOne() {  
    resize = resize+1;  
    return resize;  
}  
  
int dangerWillRobinson() {  
    // Extract subexpressions:  
    int temp = plusOne()+plusOne();  
    return temp;  
}
```

**Is your IDE too clever?
Watch out for side effects!**