Refactoring
Lecture 2

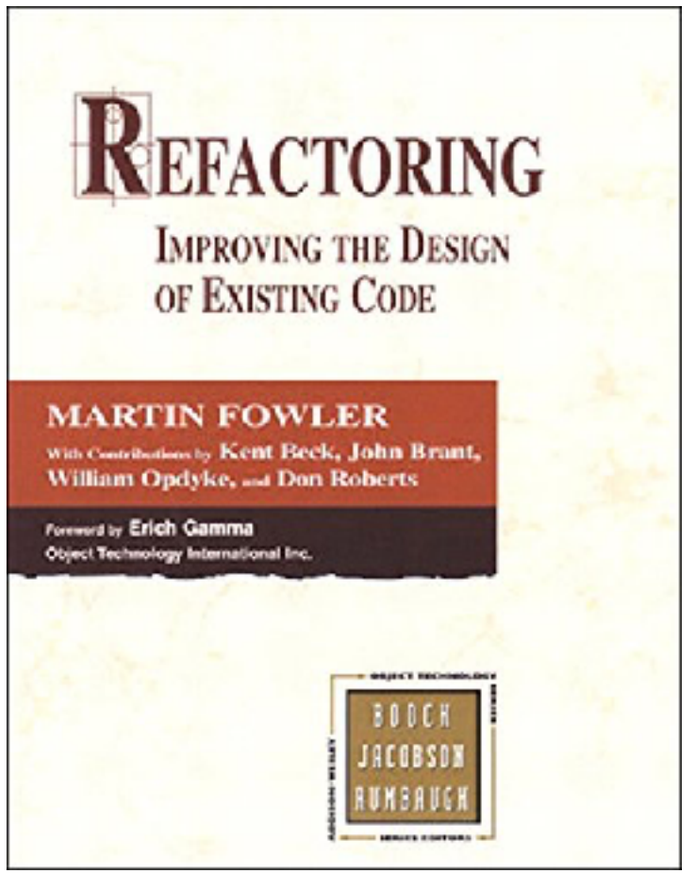
DAT159/H18
Volker Stolz

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:



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);
}
```

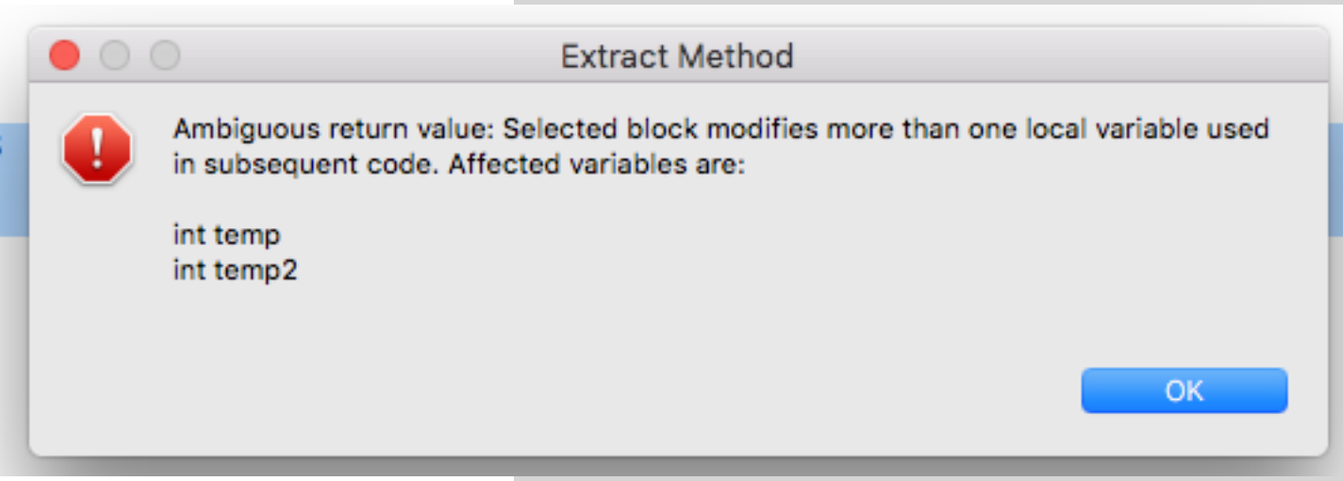
- Question: Which lines? Parameters?

Extract Method: Detailed Steps [Fowler, 110]

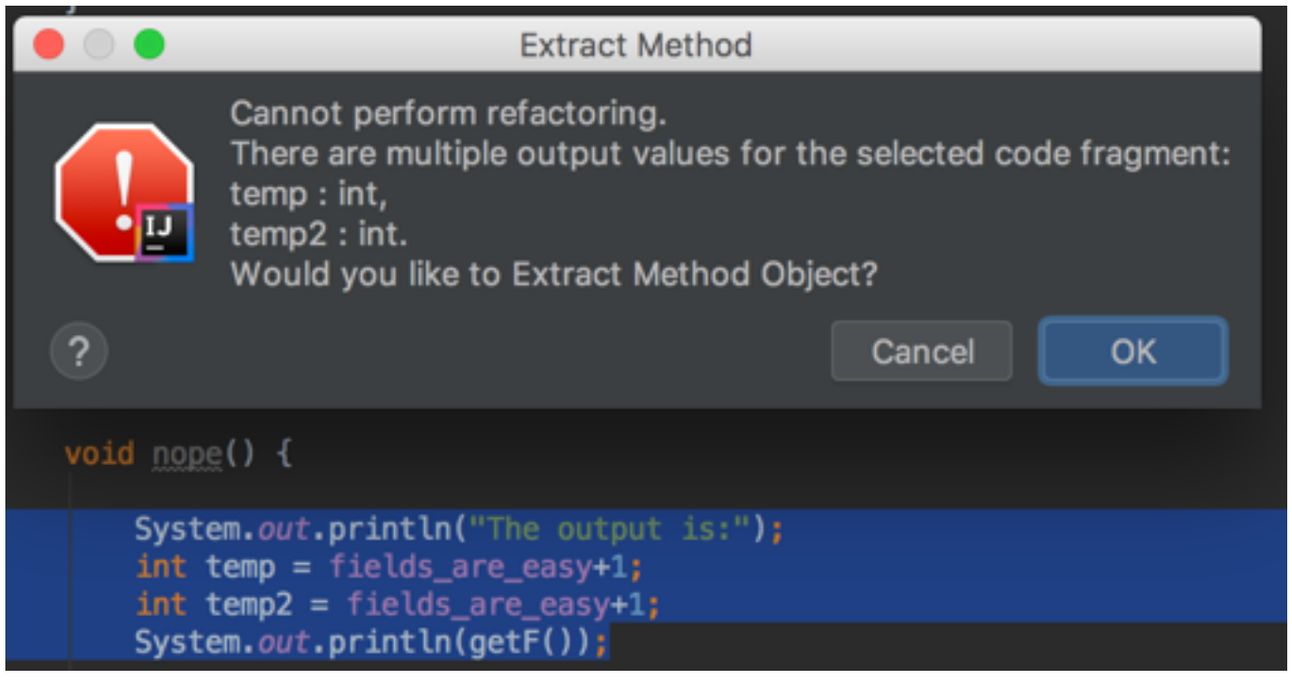
1. Create a new method with proper name.
2. Copy extracted code from source into new method.
3. Identify variables that are local in scope to the source method: add as parameters.
4. Identify and declare temporary variables in extracted code.
5. Does the extracted code modify variables in the local scope? 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

```
1  public void f() {
2    a.b.c.d.m();
3    a.b.c.d.n();
4    a.b.foo(a.b.c.d);
5    a.b.bar();
6    a.b.c.d.m();
7    }
```

After

```
public void f() {
  D temp = a.b.c.d;
  temp.m();
  temp.n();
  a.b.foo(temp);
  a.b.bar();
  temp.m();
}
```

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 ← fresh variable name;
2 for s ∈ 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!