Refactoring

Cpt S 321

Washington State University

Software aging

Refactoring

What is refactoring?

- noun: *"a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior"*.
- verb: *"to restructure software by applying a series of refactorings without changing its observable behavior"*.
- *"It is a disciplined way to clean up code that minimizes the chances of introducing bugs."*
- *"In essence when you refactor you are improving the design of the code after it has been written."*

and tested!

Martin Fowler [1]

Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional, 1st edition, 1999.

Types of refactoring

- Floss refactoring: frequent refactoring interleaved with other kinds of program changes.

- Root canal refactoring: infrequent, protracted periods of refactoring, during which programmers perform few if any other kinds of program changes.

- You perform floss refactoring to maintain healthy code, and you perform root canal refactoring to correct unhealthy code.

E. Murphy-Hill, C. Parnin, and A. P. Black, "How we refactor, and how we know it," In Proceedings of the International Conference on Software Engineering (ICSE), 2009.

Refactoring steps

1. **Identify places** where software should be refactored (e.g., **code smells**).
2. **Determine** which **refactoring**(s) should be applied (**Fowler's catalog**).
3. **Guarantee** that the applied refactoring(s) **preserve behavior**.
4. **Apply** the **refactoring**(s).
5. **Assess** the effect of the refactoring on **design quality**.
6. **Maintain consistency** between the refactored code and other **software artifacts** (such as documentation, design documents, tests, etc.)

T. Mens, and T. Tourwé, "A Survey of Software Refactoring," *IEEE Transactions on Software Engineering*, vol. 30, no. 2, pp. 126-139, February 2004.

Identifying where to apply which refactorings

- Code smells: structures in the code that suggest (**sometimes scream for!**) the possibility of refactoring.
  - Example: Smell: **duplicated code**; refactoring that can fix it: **extract method** (fix: e.g., extracting duplicated code from within a method to factor out the common code)

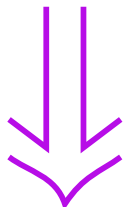
- Fowler informally links bad smells to refactorings: https://refactoring.com/catalog/

Extract Method

You have a code fragment that can be grouped together.

Turn the fragment into a method whose name explains the purpose of the method.

```
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);
}
```

inverse of *Inline Method*

ThoughtWorks®

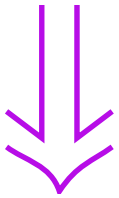
https://refactoring.com/catalog/extractFunction.html

Extract Variable

You have a complicated expression.

Put the result of the expression, or parts of the expression, in a temporary variable with a name that explains the purpose.

```
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
}
```

inverse of *Inline Temp*

Naming: In the books, this refactoring is called "Introduce Explaining Variable", but most tools and people now use the (better) name "extract variable"

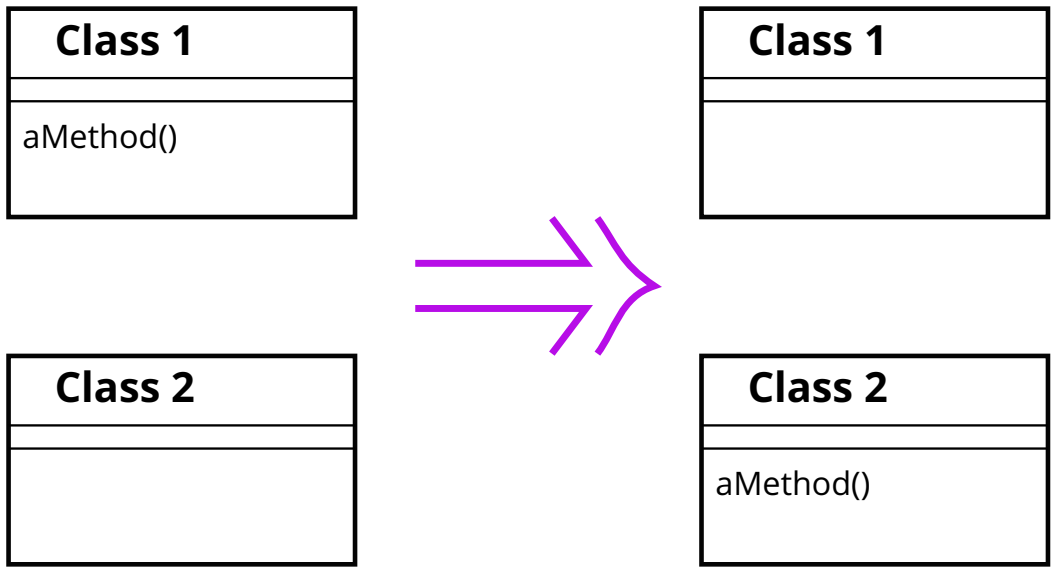
ThoughtWorks®

https://refactoring.com/catalog/extractVariable.html

Move Method

A method is, or will be, using or used by more features of another class than the class on which it is defined.

Create a new method with a similar body in the class it uses most. Either turn the old method into a simple delegation, or remove it altogether.



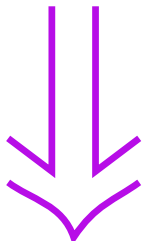
https://refactoring.com/catalog/moveFunction.html

Introduce Assertion

A section of code assumes something about the state of the program.

Make the assumption explicit with an assertion.

```
double getExpenseLimit() {
  // should have either expense limit or a primary project
  return (_expenseLimit != NULL_EXPENSE) ?
    _expenseLimit:
    _primaryProject.getMemberExpenseLimit();
}
```



```
double getExpenseLimit() {
  Assert.isTrue (_expenseLimit != NULL_EXPENSE || _primaryProject != null);
  return (_expenseLimit != NULL_EXPENSE) ?
    _expenseLimit:
    _primaryProject.getMemberExpenseLimit();
}
```

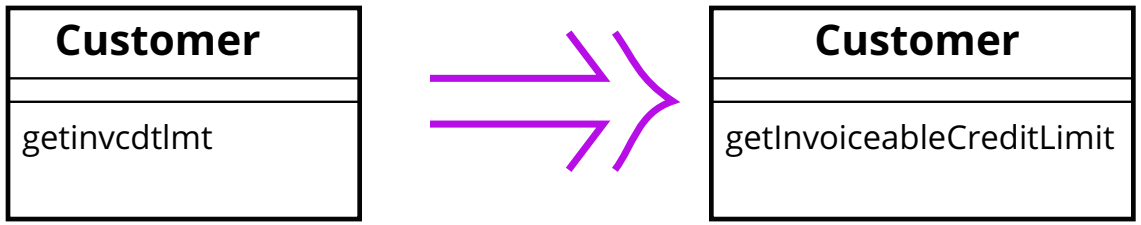
ThoughtWorks®

https://refactoring.com/catalog/introduceAssertion.html

Rename Method

The name of a method does not reveal its purpose.

Change the name of the method.



https://refactoring.com/catalog/changeFunctionDeclaration.html

Refactoring tool support

- IDEs: Many IDEs have automatic support incorporated for refactorings
  - Visual Studio: Supported refactorings
  - JetBrains Ryder: Suported refactorings
- ReSharper: Supported refactorings
- And others…

Tool Comparison

	Visual Studio Microsoft Visual Studio 2022 17.1	ReSharper Microsoft Visual Studio 2022 17.1 + ReSharper 2022.1	Rider JetBrains Rider 2022.1
Refactoring	✓ 15 solution-wide refactorings	✓✓ 58 solution-wide refactorings	✓✓ 58 solution-wide refactorings
	✓ 120+ local refactorings (automated local code transformations)	✓✓ 470+ context actions (automated local code transformations)	✓✓ 520+ context actions (automated code transformations)
	✓ Project-level cleanup refactoring: Sync Namespaces	✓✓ Project-level cleanup refactorings: Adjust Namespaces, Move Types into Matching Files, and Remove Unused References.	✓✓ Project-level cleanup refactorings: Adjust Namespaces and Move Types into Matching Files.
	✗ No matching functionality	✓ Structural replace for same or similar code	✗ No matching functionality

VS – Rename refactoring

```
using System;

namespace ConsoleApplication1
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            double r = 1.23;

            double area = Math.PI * r * r;
        }
    }
}
```

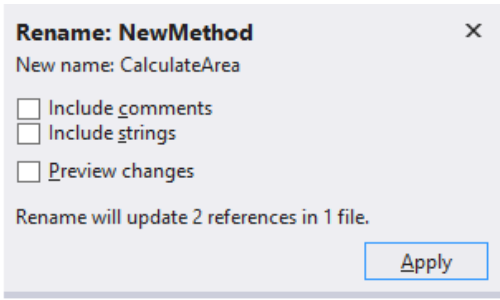
1. Highlight or place the text cursor inside the item to be renamed
2. Next, use your keyboard or mouse as follows:
    - **Keyboard**
        - Press **Ctrl+R**, then **Ctrl+R**. (Note that your keyboard shortcut may be different based on which profile you've selected.)
    - **Mouse**
        - Select **Edit > Refactor > Rename**.
        - Right-click the code and select **Rename**.
3. Rename the item simply by typing the new name.

VS – Extract method refactoring

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        double radius = 1.23;

        double area = Math.PI * radius * radius;
    }
}
```

Step 1

```
using System;

namespace ConsoleApplication1
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            double radius = 1.23;
            CalculateArea(radius);
        }

        1 reference
        private static void CalculateArea(Double radius)
        {
            double area = Math.PI * radius * radius;
        }
    }
}
```



Step 2

1. Highlight the code to be extracted
2. Next, do one of the following:
   - **Keyboard**
     1. Press **Ctrl+R**, then **Ctrl+M**. (Note that your keyboard shortcut may be different based on which profile you've selected.)
     2. Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu and select **Extract Method** from the Preview window popup.
   - **Mouse**
     1. Select **Edit > Refactor > Extract Method**.
     2. Right-click the code and select **Refactor > Extract > Extract Method**.
     3. Right-click the code, select the **Quick Actions and Refactorings** menu and select **Extract Method** from the Preview window popup.

   The method will be immediately created. From here, you can now rename the method simply by typing the new name.
3. When you're happy with the change, choose the **Apply** button or press **Enter** and the changes will be committed.

ReSharper – Introduce Variable refactoring

```
static void LogError(Exception ex)
{
    Console.WriteLine("Something has failed...");
    File.WriteAllText(@"c:\Error.txt",
        "Something has failed..." + ex);
}
```

1. Select an arbitrary expression inside member code
2. Right-click the code and select Introduce Variable
3. Give the variable a meaningful name

A new implicitly or explicitly typed local variable will be declared and initialized with the selected expression. The original expression will be replaced with the name of the variable. Should there be multiple occurrences of the original expression in your code, you will be given an option to replace all of them with the newly created variable.

Coding Demo

1. Browse through
   - The refactoring defined by Fowler
   - The refactoring supported by your IDE
2. Look at the Expression file from the ExpressionTree coding demo and identify applicable refactorings that improve the quality of the code. Those might be refactorings that you already applied when working on the coding demo or new refactorings that you did not think at that time.
3. Identify additional refactorings in the refactored version of your ExpressionTree coding demo. List refactoring in today's survey in the form:

   <Refactoring name>(<defined by>);<Old/New>: Details.

   Example:
   Substitute Algorithm (Fowler); old: Changed the implementation of the Compile method of Expression to use the Shunting Yard Algorithm and build the expression tree using a stack.

References

- M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, Refactoring: Improving the Design of Existing Code, Addison Wesley, 1999.
- W. J. Brown, R. C. Malveau, H.W. McCormick III, and T. J. Mowbray, AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis, John Wiley & Sons, 1998.
- Material from Nikolaos Tsantalis, Associate Professor at the department of Computer Science & Software Engineering at Concordia University, Montreal, Canada.