Refactoring

- Process of changing a software system in such a way that it does not alter the external behavior of the code, yet improves its internal structure.
- Disciplined way to clean up code that minimizes the chances of introducing bugs.
Refactoring – Why?

- Improves the design of software
- Makes software easier to understand
- Helps you find bugs
- Helps you program faster.

Refactoring, Martin Fowler, 1999

Refactoring – When?

- When you add a function
- When you need to fix a bug
- When you do a code review
Code Smells

- Duplicated Code
- Long method
- Long Parameter List
- Large Class
- Divergent Change
- Shotgun Surgery
- Feature Envy
- Data Clumps
- Primitive Obsession
- Switch Statements
- Parallel inheritance hierarchies
- Lazy class

- Speculative Generality
- Temporary Field
- Message Chains
- Middle Man
- Inappropriate Intimacy
- Alternative Classes with Different Interfaces
- Incomplete Library Class
- Data Class
- Refused Bequest
- Comments

Common Refactorings

<table>
<thead>
<tr>
<th>Smell</th>
<th>Common Refactorings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alternative Classes with Different Interfaces</td>
<td>Rename Method, Move Method</td>
</tr>
<tr>
<td>Comments</td>
<td>Extract Method, Introduce Assertion</td>
</tr>
<tr>
<td>Data Class</td>
<td>Move Method, Encapsulate Field, Encapsulate Collection</td>
</tr>
<tr>
<td>Data Clumps</td>
<td>Extract Class, Introduce Parameter Object, Preserve Whole Object</td>
</tr>
<tr>
<td>Divergent Change</td>
<td>Extract Class</td>
</tr>
<tr>
<td><strong>Duplicated Code</strong></td>
<td><strong>Extract Method</strong>, Extract Class, Pull Up Method, Form Template Method</td>
</tr>
<tr>
<td>Feature Envy</td>
<td>Move Method, Move Field, Extract Method</td>
</tr>
<tr>
<td>Inappropriate Intimacy</td>
<td>Move Method, Move Field, Change Bidirectional Association to Unidirectional, Replace Inheritance with Delegation, Hide Delegate</td>
</tr>
<tr>
<td>Incomplete Library Class</td>
<td>Introduce Foreign Method, Introduce Local Extension</td>
</tr>
<tr>
<td>Large Class</td>
<td>Extract Class, Extract Subclass, Extract Interface, Replace Data Value with Object</td>
</tr>
<tr>
<td>Lazy Class</td>
<td>Incline Class, Collapse Hierarchy</td>
</tr>
<tr>
<td><strong>Long Method</strong></td>
<td><strong>Extract Method</strong>, Replace Temp with Query, Replace Method with Method Object, Decompose Conditional</td>
</tr>
</tbody>
</table>
Common Refactorings

<table>
<thead>
<tr>
<th>Smell</th>
<th>Common Refactorings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Long Parameter List</strong></td>
<td><strong>Introduce Parameter Object</strong>, Replace Parameter with Method, Preserve Whole Object</td>
</tr>
<tr>
<td>Message Chains</td>
<td>Hide Delegate</td>
</tr>
<tr>
<td>Middle Man</td>
<td>Remove Middle Man, Inline Method, Replace Delegation with Inheritance</td>
</tr>
<tr>
<td>Parallel Inheritance Hierarchies</td>
<td>Move Method, Move Field</td>
</tr>
<tr>
<td>Primitive Obsession</td>
<td>Replace Data Value with Object, Extract Class, Introduce Parameter Object, Replace Array with Object, Replace Type Code with Class, Replace Type Code with Subclasses, Replace Type Code with State/Strategy</td>
</tr>
<tr>
<td>Refused Bequest</td>
<td>Replace Inheritance with Delegation</td>
</tr>
<tr>
<td>Shotgun Surgery</td>
<td>Move Method, Move Field, Inline Class</td>
</tr>
<tr>
<td>Speculative Generality</td>
<td>Collapse Hierarchy, Inline Class, Remove Parameter, Rename Method</td>
</tr>
<tr>
<td>Switch Statements</td>
<td>Replace Conditional with Polymorphism, Replace Type Code with Subclasses, Replace Type Code with State/Strategy, Replace Parameter with Explicit Methods, Introduce Null Object</td>
</tr>
<tr>
<td>Temporary Field</td>
<td>Extract Class, Introduce Null Object</td>
</tr>
</tbody>
</table>

Smell – Duplicated Code

- Same code structure in more than one place – unify them
- Example:
  - same expression in methods of the same class
  - Use Extract Method refactoring and call the new method from both places
- Example:
  - same expression in sibling classes
  - Use Extract Method refactoring in both classes, and Pull Up Method
Smell – Long Method

- Shortest methods live the best and longest
- Most of the time all you need to do is Extract Method refactoring to long methods.
- Long lists of parameters can be slimmed down with Introduce Parameter Object and Preserve Whole Object Refactoring.

Refactor - Extract Method

- Create a new method, name it after intention
- Copy extracted code from source into new method
- Scan code for references to variables that are local in scope to source method - local variables and parameters for new method.
- See whether any temporary variables are only used within the extracted code and declare in new method.
- Look to see whether any of these local variables are modified by the extracted code. May need to use a different approach.
- Pass into the target method as parameters the local-scope variables that are read from the extracted code.
- Compile when you have dealt with locally scoped variables.
- Replace the extracted code in the source methods with a call to the new methods.
- Remove temporary variables defined and used in the new method from the source methods.
- Compile and test.
Smell – Long Parameter List

• Shortest methods live the best and longest
• Most of the time all you need to do is Extract Method refactoring to long methods.
• Long lists of parameters can be slimmed down with Introduce Parameter Object and Preserve Whole Object Refactoring.

Refactor - Introduce Parameter Object

• Create a new class to represent the group of parameters you are replacing. Make the class immutable.
• Compile.
• Retain the old method signature with a new body that maps the parameters to the new objects.
• Create a new method signature for the body of the original method and modify the method to use the new object parameters.
• Compile and test.
• Replace calls to the old method signature with calls to the new method signature using the new class for parameters. Consider doing the incrementally if there are many callers, compiling and testing after you convert each caller.
• Remove the old method signature.
• Compile and test.