DCC / ICEx / UFMG

Refactoring

Eduardo Figueiredo

http://www.dcc.ufmg.br/~figueiredo

Refactoring

- Refactoring is a change made to the internal structure of software without changing its observable behavior
  - Its goal is to make the software easier to understand and cheaper to modify
- When is refactoring needed?
  - "If it stinks, change it."

Grandma Beck

Why Should You Refactor?

- To improve design of code
  - Without refactoring, code quality will decay
- To make software easy to understand
  - Refactoring makes your code more readable
- To help you find bugs
  - Easy understand the code, easy to find bugs
- To help you program faster
  - Essential for agile software development

Refactoring Classification

- Composing Methods
- Moving Features Between Objects
- Organizing Data
- Simplifying Conditional Expressions
- Making Method Calls Simpler
- Dealing with Generalization
- Big Refactorings

Composing Methods

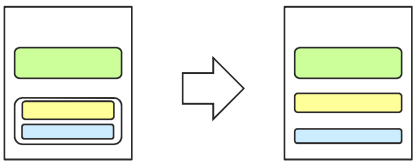
Chapter 6

Composing Methods

Extract Method	Inline Method	Inline Temp
Replace Temp with Query		Introduce Explaining Variable
Split Temporary Variable		Remove Assignments to Parameters
Replace Method with Method Object		Substitute Algorithm

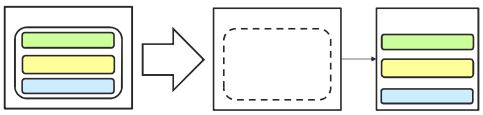
Extract Method

- You have a code fragment (part of a method) that can be grouped together
  - Turn the fragment into a method whose name explains its purpose



Replace Method with Method Object

- You have a long method that uses local variables in such a way that you cannot apply Extract Method
  - Turn the method into its own object
  - All local variables become fields
  - Decompose the method into other methods



Moving Features Between Objects

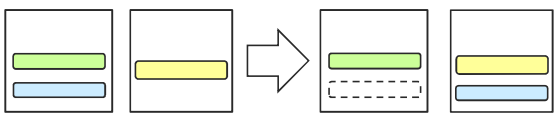
Chapter 7

Moving Features Between Objects

Move Method	Move Field
Extract Class	**Inline Class**
Hide Delegate	Remove Middle Man
Introduce Foreign Method	Introduce Local Extension

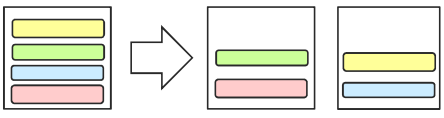
Move Method

- A method is using more features of another class than its own class
  - Create a new method with a similar body in the class it uses most
  - Either remove or turn the old method into a delegation



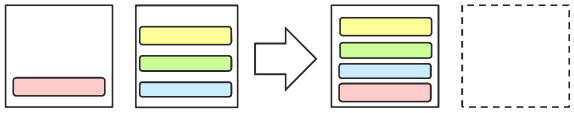
Extract Class

- You have one class doing work that should be done by two
  - Create a new class
  - Move the relevant fields and methods from the old class into the new class



Inline Class

- A class isn't doing very much
  - Move all its features into another class
  - Delete the empty class



Organizing Data

Chapter 8

Organizing Data

Self Encapsulate Field	Replace Data Value with Object
Change Value to Reference	Change Reference to Value
Replace Array with Object	Duplicate Observed Data
Replace Magic Number with Symbolic Constant	**Encapsulate Field**
Encapsulate Collection	Replace Record with Data Class
Replace Type Code with Class	Replace Subclass with Fields

Self Encapsulate Field

- You are accessing a field directly, but the coupling to the field is awkward
  - Create getting and setting methods
  - Use only those methods to access the field

```
private int low, high;
boolean verify(int arg) {
  return arg >= low &&
    arg <= high;
}
```

```
private int low, high;
boolean verify(int arg) {
  return arg >= getLow() &&
    arg <= getHigh();
}
int getLow() {return low;}
int getHigh() {return high;}
```

Encapsulate Field

- There is a public field
  - Make it private or protected
  - Provide accessors

```
public String name;
```

```
private String name;
public String getName() {
  return name;
}
public void setName(String arg) {
  name = arg;
}
```

Simplifying Conditional Expressions

Chapter 9

Simplifying Conditional Expressions

Decompose Conditional	Consolidate Conditional Expression
Consolidate Duplicate Conditional Fragments	Remove Control Flag
Replace Nested Conditional with Guard Clauses	**Replace Conditional with Polymorphism**
Introduce Null Object	Introduce Assertion

Decompose Conditional

- You have a complicated conditional statement
  - Extract methods from the *condition*, *then* part, and *else* part

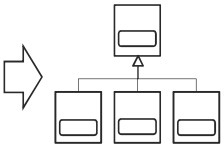
```
if (date.before(SUMMER_START)) || (date.after(SUMMER_END))
  charge = quantity * winterRate + winterServiceCharge;
else
  charge = quantity * summerRate;
```

```
if (notSummer(date))
 charge = winterCharge(quantity);
else
 charge = summerCharge(quantity);
```

Replace Conditional with Polymorphism

- You have a conditional that chooses different behavior depending on the type
  - Move each leg of the conditional to a overriding method in a subclass
  - Make the original method abstract

```
double speed() {
  switch(type) {
    case(EUROPEAN)
      return getBaseSpeed();
    case(AFRICAN)
      return getBaseSpeed()*nunCoconuts;
    case(NORWEGIAN)
      return getBaseSpeed()-loadFactor();
}}
```



Making Method Calls Simpler

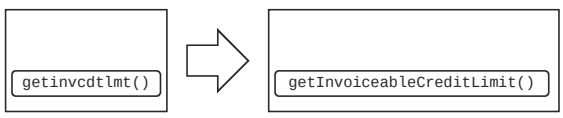
Chapter 10

Making Method Calls Simpler

Rename Method	Add Parameter	Remove Parameter
Separate Query from Modifier	Parameterize Method	
Replace Parameter with Explicit Methods	Preserve Whole Object	
Replace Parameter with Method	**Introduce Parameter Object**	
Remove Setting Method	Hide Method	
Replace Constructor with Factory Method	Encapsulate Downcast	

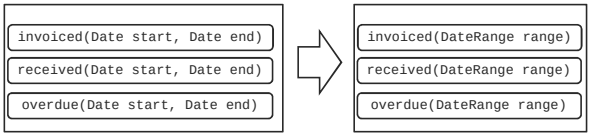
Rename Method

- The name of a method does not reveal its purpose
  - Change the name of the method



Introduce Parameter Object

- You have a group of parameters that naturally go together
  - Replace them with an object



Dealing with Generalization

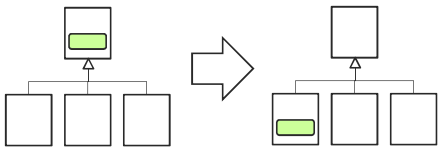
Chapter 11

Dealing with Generalization

Pull Up Field	Pull Up Method
Push Down Method	Push Down Field
Pull Up Constructor Body	**Extract Subclass**
Extract Superclass	Extract Interface
Collapse Hierarchy	Form Template Method
Replace Inheritance with Delegation	Replace Delegation with Inheritance

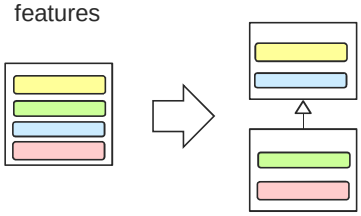
Push Down Method

- Behavior on a superclass is relevant only for some of its subclasses
  - Move it to those subclasses



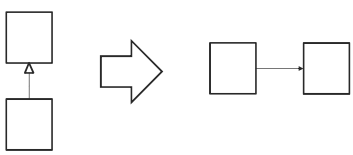
Extract Subclass

- A class has feature that are used only in some instances
  - Create a subclass for that subset of features



Replace Inheritance with Delegation

- A subclass uses only part of a superclass interface
  - Create a field for the superclass
  - Adjust methods to delegate to superclass
  - Remove the inheritance relationship



Big Refactorings

Chapter 12

Big Refactorings

- Tease Apart Inheritance
- Convert Procedural Design to Objects
- Separate Domain from Presentation
- Extract Hierarchy

Bibliography

- Martin Fowler. **Refactoring: Improving the Design of Existing Code**. Addison-Wesley Professional, 1st edition, 1999.
  - Chapters 2, 6, 7, 8, 9, 10, 11, and 12