Advanced Object-Oriented Design

Lecture 8

Software Refactoring

Part IV: In-class, class hierarchies and other refactorings

Bartosz Walter

<Bartek.Walter@man.poznan.pl>

Agenda

1. Transforming inter-class associations
2. Managing complex conditional expressions

Catalog of software refactorings

Modifying the inheritance hierarchy

- moving members up and down the hierarchy
- extracting new entities

Pull Up Field/ Method

by M. Fowler

Name

Pull Up Field/ Method

Summary

Two subclasses have the same field/ method

Goal

Move the field/ method to the superclass

Mechanics

- inspect the member to ensure they are identical
- create a new method in superclass, copy the body of one of methods, adjust and compile
- delete one subclass method, compile & test
- proceed with deleting subclass methods

Pull Up Constructor Body

by M. Fowler

Name

Pull Up Constructor Body

Summary

Constructors on subclasses are almost identical

Goal

Create a superclass constructor, call it from subclasses

Mechanics

- define a superclass constructor
- move the common code from the subclass to the superclass
- call the superclass constructor as first step in the subclass constructor
- compile & test

Push Down Field/ Method

by M. Fowler

Name

Push Down Field/ Method

Summary

Behavior on superclass is relevant only for some subclasses

Goal

Move it to those subclasses

Mechanics

- declare a method in all subclasses and copy the body into each subclass (beware of access level!)
- remove method from superclass (or declare abstract)
- compile & test
- remove method from subclasses that do not need it

Extract Interface

by M. Fowler

Name

Extract Interface

Summary

Some clients use the same subset of class's interface

Goal

Extract the subset into an interface

Mechanics

- create an empty interface
- declare the common operations in the interface
- declare the relevant classes as implementing the interface
- adjust the client type declarations to use the interface

Extract Superclass

by M. Fowler

Name

Extract Superclass

Summary

There are two classes with similar features

Goal

Move the common features to a newly created superclass

Mechanics

- create a blank abstract superclass
- pull up fields, whole methods and constructor body
- compile & test at every change
- if necessary, split remaining methods and pull them up
- change references in clients to superclass (if possible)

Extract Subclass

by M. Fowler

Name

Extract Subclass

Summary

Some features of a class are used only in some its instances

Goal

Extract a subclass for these features

Mechanics

- define a new subclass
- define appropriate constructors for the subclass (use Factory Method if needed)
- replace calls to superclass constructor with a subclass one appropriately
- push down selected fields/ methods to the subclass
- eliminate remaining fields that controls behavior of original class now indicated by the subclass

Catalog of software refactorings

In-class refactorings

- changing inter-method relationships

Remove Setter

by M. Fowler

Name

Remove Setter

Summary

A field should be set at creation time and never altered

Goal

Remove any updating method for that field

Mechanics

- check if setter for the field is called only in the constructor (directly or by other method called by constructor)
- make the constructor to access the field directly
- compile & test
- remove the setter and make the field final

Form Template Method

by M. Fowler

Name

Form Template Method

Summary

Two methods in subcl. perform similar stops in same order

Goal

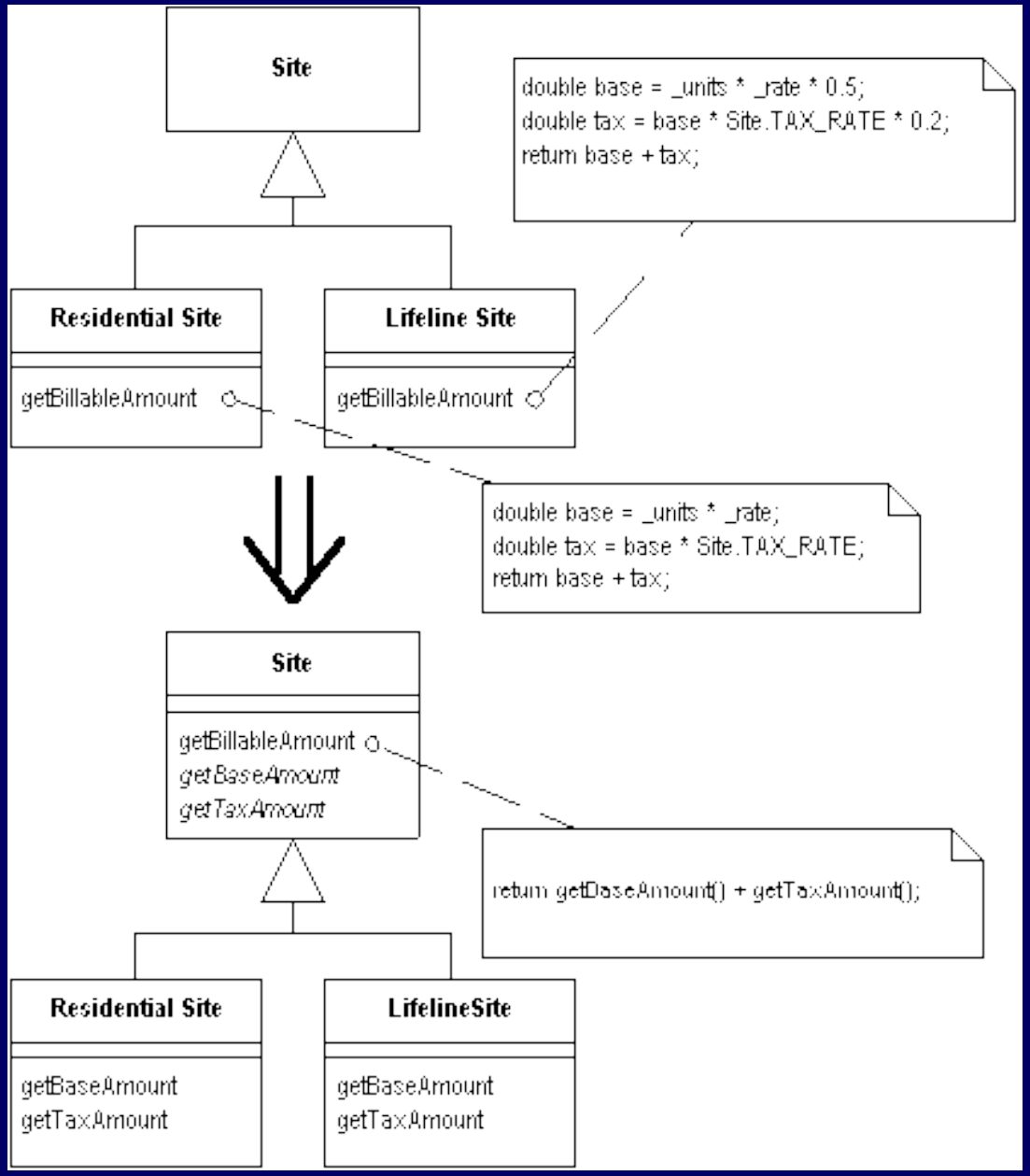
Give them same signature and then pull them up

Mechanics

- decompose methods so that extracted methods are either identical or completely different
- pull up the identical methods into the superclass
- rename different methods so the signatures of all methods at each step are the same, compile & test
- pull up one of original methods; make signatures of different methods abstract at superclass
- compile & test
- remove the other methods, compile & test

Example: Form Template Method

by M. Fowler



Inline Class

by M. Fowler

Name

Inline Class

Summary

A class does not earn for itself

Goal

Move its features to another class and delete it

Mechanics

- move the public protocol from the source class to the absorbing one
- delegate the public methods to the source class
- change all references from the source class to the absorbing one
- compile & test
- move fields & methods to the absorbing class

Replace Method with Method Object

by M. Fowler & K. Beck

Name

Replace Method with Method Object

Summary

A long method uses local variables so that it cannot be split

Goal

Turn the method into its own object

Mechanics

- create a new class and name it appropriately
- give it a final field for the object that hosted original method and fields for temporary variables and parameters
- create a constructor that takes a source object and each parameter
- move the original method into the new class
- compile
- replace the call to the method with creation of an instance of the new class and call its method

Replace Static Variable with Parameter

by M. Vittek

Name

Replace Static Variable with Parameter

Summary

A function depending on a static variable needs to be reused in more general context.

Goal

Pass the variable as parameter

Mechanics

- if the function calls other functions using the static variable in question, then use this refactoring on all those invoked functions first.
- add a new argument to the function
- add the static variable as actual argument to all callers of this function in.
- replace all references to the static variable within the function by the new argument

Example: Replace Static Variable with Parameter

by M. Fowler

```
void printValues() {
   for (int i = 0; i < people.length; i++) {
      System.out.println(people[i].name+" has salary "+people[i].salary);
   }
}

public static void main(String args[]) {
   ... printValues();
}
```

```
void printValues(PrintStream outfile) {
   for (int i = 0; i < people.length; i++) {
      outfile.println(people[i].name+" has salary "+people[i].salary);
   }
}

public static void main(String args[]) {
   ... printValues(System.out);
}
```

Catalog of software refactorings

Other refactorings

- changing algorithms
- dealing with non-existing methods
- dynamic & static initialization

Split Loop

by M. Fowler

Name

Split Loop

Summary

A loop is doing two things

Goal

Split the loop

Mechanics

- copy the loop and remove the differing pieces from each loop
- compile and test
- reorganize the lines to group the loop with related code from outside the loop
- compile and test.
- consider applying *Extract Method* or *Replace Temp with Query* on each loop

Example: Split loop

by M. Fowler

```
private Person [] people;

void printValues()  {
   double averageAge = 0;
   double totalSalary = 0;
   for (int i = 0; i < people.length; i++) {
      averageAge += people[i].age;
      totalSalary += people[i].salary;
   }
   averageAge = averageAge / people.length;
   System.out.println(averageAge);
   System.out.println(totalSalary);
}
```

Example: Split loop

by M. Fowler

```
private Person [] people;

void printValues()  {
   double averageAge = 0;
   double totalSalary = 0;
   for (int i = 0; i < people.length; i++) {
      totalSalary += people[i].salary;
   }
   for (int i = 0; i < people.length; i++) {
      averageAge += people[i].age;
   }
   averageAge = averageAge / people.length;
   System.out.println(averageAge);
   System.out.println(totalSalary);
}
```

Example: Split loop

by M. Fowler

```
private Person [] people;

void printValues()  {
   double averageAge = 0;
   for (int i = 0; i < people.length; i++) {
      totalSalary += people[i].salary;
   }

   double totalSalary = 0;
   for (int i = 0; i < people.length; i++) {
      averageAge += people[i].age;
   }

   averageAge = averageAge / people.length;

   System.out.println(averageAge);
   System.out.println(totalSalary);
}
```

Example: Split loop

by M. Fowler

```
void printValues() {
   System.out.println(averageAge());
   System.out.println(totalSalary());
}

private double averageAge() {
   double result = 0;
   for (int i = 0; i < people.length; i++) {
      result += people[i].age;
   }
   return result / people.length;
}

private double totalSalary() {
   double result = 0;
   for (int i = 0; i < people.length; i++) {
      result += people[i].salary;
   }
   return result;
}
```

Substitute Algorithm

Name

Substitute Algorithm

Summary

You want to replace an algorithm with one that is clearer

Goal

Replace the body of the method with the new algorithm

Mechanics

- prepare your alternative algorithm and get it compiling
- run the new algorithm against the tests
- if tests fail, use the old algorithm for comparison in testing and debugging

Example: Substitute Algorithm

by M. Fowler

```
String foundPerson(String[] people)  {

   for (int i = 0; i < people.length; i++)   {
      if (people[i].equals ("Don"))  {
         return "Don";
      }
      if (people[i].equals ("John"))  {
         return "John";
      }
      if (people[i].equals ("Kent"))  {
         return "Kent";
      }
   }

   return "";
}
```

Example: Substitute Algorithm

by M. Fowler

```
String foundPerson(String[] people)  {

   List candidates = Arrays.asList(new String[] {"Don", "John", "Kent"});
   for (int i=0; i<people.length; i++)
      if (candidates.contains(people[i]))
         return people[i];

   return "";
}
```

Introduce Assertion

by M. Fowler

Name

Introduce Assertion

Summary

A section of code assumes sth about the state of program

Goal

Make the assumption explicit with an assertion

Mechanics

- assertions by default should not change the behavior
- do they?

Introduce Foreign Method

by M. Fowler

Name

Introduce Foreign Method

Summary

A server class needs a new method, but cannot be modified

Goal

Create class in the client class and pass a server class instance to it as the first argument

Mechanics

- create the needed method in the client class
- make an instance of the server class the first parameter
- comment appropriately to avoid accidental execution

Introduce Local Extension

by M. Fowler

Name

Introduce Local Extension

Summary

A server class needs a new method, but cannot be modified

Goal

Create a new class with extra method. Make it a wrapper or subclass of the original

Mechanics

- create an extension class as either wrapper or subclass of the original
- add converting constructors to the extension
- add new features to the extension
- replace the original with the extension where needed
- move any foreign methods defined for this class up to now onto the extension

Replace Recursion with Iteration

by I. Mitrovic

Name

Replace Recursion with Iteration

Summary

Code that uses recursion is hard to understand

Goal

Replace recursion with iteration

Mechanics

- determine the base case of the recursion
- implement a loop that will iterate until the base case is reached
- make a progress towards the base case; send the new arguments to the top of the loop instead to the recursive method

Example: Replace Recursion with Iteration

by I. Mitrovic

```
public class CountDown {
   public void countDown(int n) {
      if (n == 0) return;
      System.out.println(n + "...");
      waitASecond();
      countDown(n-1);
   }

   public void waitASecond() {
      try {
         Thread.sleep(1000);
      } catch (InterruptedException ignore) { }
   }

   public static void main(String[] args) {
      CountDown c = new CountDown();
      c.countDown(10);
   }
}
```

Replace Iteration with Recursion

by I. Mitrovic

Name

Replace Iteration with Recursion

Summary

It is not obvious what each iteration in loop is doing

Goal

Replace iteration with recursion

Mechanics

- identify the candidate loop that modifies one or more scoped locals and then returns a result based on their final values
- move the loop into a new function
- compile & test
- replace the loop with a function that accepts the local variables, and which returns the final result

Replace Iteration with Recursion

by I. Mitrovic

Name

Replace Iteration with Recursion

Mechanics

- the implementation of the function should be an 'if' statement, which tests the looping condition (the condition expression in "while (condition) ...;"); the "then" clause should calculate/return the final result; the "else" clause should make the recursive call, with appropriately modified parameters
- compile & test

Example: Replace Iteration with Recursion

by I. Mitrovic

```
unsigned greatest_common_divisor (unsigned a, unsigned b)  {
   while (a != b) {
      if (a > b) {
         a -= b;
      } else if (b > a) {
         b -= a;
      }
   }
}
```

Example: Replace Iteration with Recursion

by I. Mitrovic

```
unsigned greatest_common_divisor (unsigned a, unsigned b)  {
   if (a > b)  {
      return greatest_common_divisor ( a-b, b );
   } else if (b > a)  {
      return greatest_common_divisor ( a, b-a );
   } else // (a == b)  {
      return a; }
}
```

Q&A