

CS 635 Advanced Object-Oriented Design & Programming  
Spring Semester, 2009  
Doc 9 Some Refactoring  
Feb 24, 2009

Copyright ©, All rights reserved. 2009 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (<http://www.opencontent.org/opl.shtml>) license defines the copyright on this document.

## Reading

Refactoring, Fowler, 1999, Chapters 6 & 9

## References

Refactoring, Fowler, 1999, Chapters 6 & 9

Refactoring to Patterns, Kerievsky, 2005, pp 129-143, 191-201

# Extract Method

## Example

```
void printOwning(double amount) {  
    printBanner();  
  
    //print details  
    System.out.println("name: " + name);  
    System.out.println("amount: " + amount);  
}
```



```
void printOwning(double amount) {  
    printBanner();  
    printDetails(amount);  
}
```

```
void printDetails( double amount) {  
    System.out.println("name: " + name);  
    System.out.println("amount: " + amount);  
}
```

# 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

Apply when

A method is too long

Code needs a comment to understand

# Inline Method

A method's body is just as clear as its name

Put the method's body into the body of the callers and remove the method

```
int getRating() {  
    return (moreThanFiveLateDeliveries()) ? 2 : 1;  
}  
boolean moreThanFiveLateDeliveries() {  
    return numberOfLateDeliveries > 5;  
}
```



```
int getRating() {  
    return (numberOfLateDeliveries > 5) ? 2 : 1;  
}
```

# Simplifying Conditionals

Don't you get tired of unreadable if and case statements?

# Decompose Conditional

You have complicated conditional conditional statement

Extract methods from the condition, then part, and else parts

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



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

# Consolidate Duplicate Conditional Fragments

```
if (hasVolumeDiscount()) {  
    total = price * amount * 0.95;  
    send();  
}  
else {  
    total = price * amount * 1.075  
    send();  
}
```



```
if (hasVolumeDiscount())  
    total = price * amount * 0.95;  
else  
    total = price * amount * 1.075  
send();
```

# Remove Control Flag

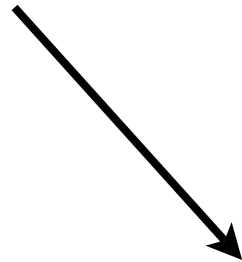
You have a variable that is acting as a control flag for a series of boolean expressions

Use a break or return instead

```
done = false
while (! done )
    if ( some test)
        do something
        done = true
```

# Example

```
void findBadApple(Fruit[] basket) {  
    boolean found = false;  
    for (int k= 0;k < basket.length; k++) {  
        if (! found ) {  
            if (basket[k].isApple() & basket[k].isBad()) {  
                report(k);  
                found = true;  
            }  
        }  
    }  
}
```



```
void findBadApple(Fruit[] basket) {  
    for (int k= 0;k < basket.length; k++) {  
        if (basket[k].isApple() & basket[k].isBad()) {  
            report(k);  
            return;  
        }  
    }  
}
```

# Replace Nested Conditional with Guard Clauses

Nested conditionals obscure normal path of execution

Use guard clauses for all the special cases

```
double sillyExample() {  
    double result;  
    if (foo < 0 )  
        result = negativeFoo();  
    else {  
        if (bar > 0)  
            result = positiveBar();  
        else result = -12* bar/foo;  
    }  
    return result;  
}
```



```
double sillyExample() {  
    if (foo < 0 ) return negativeFoo();  
    if (bar > 0) positiveBar();  
    return -12* bar/foo;  
}
```

# Introduce Null Object

You have repeated checks for a null value

Replace the null value with a null object

# Replace Conditional Logic with Strategy

Conditional logic in a method controls which of several variants of a calculation are executed

Create a Strategy for each variant and make the method delegate the calculation to a Strategy instance

```
capital() {  
    if (expiry == null && maturity != null)  
        return commitment * duration() * riskFactor();  
    if (expiry != null && maturity == null)  
        if (getUnusedPercentage() != 1.0)  
            return aLongCalculation  
        else  
            return someOtherLongCalculation  
    }  
    return 0.0  
}
```

```
capital() {  
    return capitalStrategy.capital(this);  
}
```

# Replace Conditional Dispatcher with Command

Conditional logic is used to dispatch requests and execute actions.

Create a Command for each action

Store commands in a collection

Replace the conditional logic with code to fetch and execute Commands

# Replace Conditional Dispatcher with Command

```
if (actionName.equals(NEW_USER)) {  
    //code to add new users  
} else if (actionName.equals(LOGIN)) {  
    //code to login in  
} else more conditions  
etc
```



```
handler = lookupHandlerBy(actionName);  
handler execute();
```

# When

You need runtime flexibility

Have bloated code that performing conditional dispatch

# Mechanics

- 1 Find code that handles a request  
Apply Extract Method until have execution method  
That is a method that invokes codes behavior

Compile and test

- 2 Repeat step 1 to extract all request-handling code into separate execution methods

Compile and test

# Mechanics

3 Apply Extract Class on each execution method to produce a concrete command

Compile and test

4 Define the command interface or abstract class

Compile

5 Make each concrete command implement or extend your command

Compile and test

# Mechanics

- 6 Add and populate command map on class that contains the conditional dispatcher

Compile

- 7 On class that contains the conditional dispatcher replace conditional code with code to fetch correct command and execute it

Compile and test