Refactoring away from code smells

Kai-Hui Liang

Code smells

Bloaters

- Long Method
- Large Class
- Primitive Obsession
- Long Parameter List
- Data Clumps

Object-Orientation Abusers

- Alternative Classes with Different Interfaces
- Refused Bequest
- Switch Statements
- Temporary Field

Change Preventers

- Divergent Change
- Parallel Inheritance Hierarchies
- Shotgun Surgery

Dispensables

- Comments
- Duplicate Code
- Data Class
- Dead Code
- Lazy Class
- Speculative Generality

Couplers

- Feature Envy
- Inappropriate Intimacy
- Incomplete Library Class
- Message Chains
- Middle Man

Refactoring operations (or recipes)

Composing Methods

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

Moving Features between Objects

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

Dealing with Generalization

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

…

How to leverage **refactoring operations** to refactor code away from **code smells**?

Demo cases:

Code smells:

Long method

Refactoring recipes:

1. Extract methods
2. Reduce Local Variables and Parameters Before Extracting a Method
   - Replace Temp with Query,
   - Introduce Parameter Object or
   - Preserve Whole Object.
3. Replace Method with Method Object
4. Conditionals and Loops
   - Decompose Conditional
   - Extract methods in loop

Code smell: Long method

A method contains too many lines of code. Generally, any method longer than ten lines should make you start asking questions.

Code smell: Long method

A method contains too many lines of code. Generally, any method longer than ten lines should make you start asking questions.

Reasons for the Problem

- Something is always being added to a method but nothing is ever taken out. Since it's easier to write code than to read it, this "smell" remains unnoticed until the method turns into an ugly, oversized beast.
- Mentally, it's often harder to create a new method than to add to an existing one: "But it's just two lines, there's no use in creating a whole method just for that..." Which means that another line is added and then yet another, giving birth to a tangle of spaghetti code.

Code smell: Long method

A method contains too many lines of code. Generally, any method longer than ten lines should make you start asking questions.

Treatment

- As a rule of thumb, if you feel the need to comment on something inside a method, you should take this code and put it in a new method.
- Even a single line can and should be split off into a separate method, if it requires explanations. And if the method has a descriptive name, nobody will need to look at the code to see what it does.

Long method - recipe #1: extract method

To reduce the length of a method body, use Extract Method.

Long method - recipe #1: extract method

```
void printOwing() {
  Enumeration elements = orders.elements();
  double outstanding = 0.0;

  // print banner
  System.out.println ("**********************************");
  System.out.println ("****** Customer totals ******");
  System.out.println ("**********************************");

  // print owings
  while (elements.hasMoreElements()) {
    Order each = (Order) elements.nextElement();
    outstanding += each.getAmount();
  }

  // print details
  System.out.println("name: " + name);
  System.out.println("amount: " + outstanding);
}
```

Long method - recipe #1: extract method

```java
void printOwing() {
  Enumeration elements = orders.elements();
  double outstanding = 0.0;

  // print banner
  System.out.println ("*****************************");
  System.out.println ("****** Customer totals ******");
  System.out.println ("*****************************");

  // print owings
  while (elements.hasMoreElements()) {
    Order each = (Order) elements.nextElement();
    outstanding += each.getAmount();
  }

  // print details
  System.out.println("name: " + name);
  System.out.println("amount: " + outstanding);
}
```

```java
void printOwing() {
  Enumeration elements = orders.elements();
  double outstanding = 0.0;

  printBanner();

  // print owings
  while (elements.hasMoreElements()) {
    Order each = (Order) elements.nextElement();
    outstanding += each.getAmount();
  }

  // print details
  System.out.println("name: " + name);
  System.out.println("amount: " + outstanding);
}

void printBanner() {
  System.out.println("*****************************");
  System.out.println("****** Customer totals ******");
  System.out.println("*****************************");
}
```

Long method - recipe #1: extract method

```
1  void printOwing() {
2    Enumeration elements = orders.elements();
3    double outstanding = 0.0;
4
5    // print banner
6    System.out.println ("*****************************");
7    System.out.println ("****** Customer totals ******");
8    System.out.println ("*****************************");
9
10   // print owings
11   while (elements.hasMoreElements()) {
12     Order each = (Order) elements.nextElement();
13     outstanding += each.getAmount();
14   }
15
16   // print details
17   System.out.println("name: " + name);
18   System.out.println("amount: " + outstanding);
19 }
```

```
1  void printOwing() {
2    Enumeration elements = orders.elements();
3    double outstanding = 0.0;
4
5    printBanner();
6
7    // print owings
8    while (elements.hasMoreElements()) {
9      Order each = (Order) elements.nextElement();
10     outstanding += each.getAmount();
11   }
12
13   // print details
14   System.out.println("name: " + name);
15   System.out.println("amount: " + outstanding);
16 }
17
18 void printBanner() {
19   System.out.println("*****************************");
20   System.out.println("****** Customer totals ******");
21   System.out.println("*****************************");
22 }
```

Test before moving on!

Long method - recipe #1: extract method

```
1  void printOwing() {
2    Enumeration elements = orders.elements();
3    double outstanding = 0.0;
4
5    printBanner();
6
7    // print owings
8    while (elements.hasMoreElements()) {
9      Order each = (Order) elements.nextElement();
10     outstanding += each.getAmount();
11   }
12
13   // print details
14   System.out.println("name: " + name);
15   System.out.println("amount: " + outstanding);
16 }
17
18 void printBanner() {
19   System.out.println("**********************************");
20   System.out.println("****** Customer totals ******");
21   System.out.println("**********************************");
22 }
```

Long method - recipe #1: extract method

```
1  void printOwing() {
2    Enumeration elements = orders.elements();
3    double outstanding = 0.0;
4
5    printBanner();
6
7    // print owings
8    while (elements.hasMoreElements()) {
9      Order each = (Order) elements.nextElement();
10     outstanding += each.getAmount();
11   }
12
13   printDetails(outstanding);
14 }
15
16 void printBanner() {
17   System.out.println("*****************************");
18   System.out.println("****** Customer totals ******");
19   System.out.println("*****************************");
20 }
21
22 void printDetails(double outstanding) {
23   System.out.println("name: " + name);
24   System.out.println("amount: " + outstanding);
25 }
```

```
1  void printOwing() {
2    Enumeration elements = orders.elements();
3    double outstanding = 0.0;
4
5    printBanner();
6
7    // print owings
8    while (elements.hasMoreElements()) {
9      Order each = (Order) elements.nextElement();
10     outstanding += each.getAmount();
11   }
12
13   // print details
14   System.out.println("name: " + name);
15   System.out.println("amount: " + outstanding);
16 }
17
18 void printBanner() {
19   System.out.println("*****************************");
20   System.out.println("****** Customer totals ******");
21   System.out.println("*****************************");
22 }
```

Long method - recipe #1: extract method

```
1  void printOwing() {
2    Enumeration elements = orders.elements();
3    double outstanding = 0.0;
4
5    printBanner();
6
7    // print owings
8    while (elements.hasMoreElements()) {
9      Order each = (Order) elements.nextElement();
10     outstanding += each.getAmount();
11   }
12
13   printDetails(outstanding);
14 }
15
16 void printBanner() {
17   System.out.println("*****************************");
18   System.out.println("****** Customer totals ******");
19   System.out.println("*****************************");
20 }
21
22 void printDetails(double outstanding) {
23   System.out.println("name: " + name);
24   System.out.println("amount: " + outstanding);
25 }
```

```
1  void printOwing() {
2    Enumeration elements = orders.elements();
3    double outstanding = 0.0;
4
5    printBanner();
6
7    // print owings
8    while (elements.hasMoreElements()) {
9      Order each = (Order) elements.nextElement();
10     outstanding += each.getAmount();
11   }
12
13   // print details
14   System.out.println("name: " + name);
15   System.out.println("amount: " + outstanding);
16 }
17
18 void printBanner() {
19   System.out.println("*****************************");
20   System.out.println("****** Customer totals ******");
21   System.out.println("*****************************");
22 }
```

Test before moving on!

Long method - recipe #1: extract method

```
1  void printOwing() {
2    Enumeration elements = orders.elements();
3    double outstanding = 0.0;
4
5    printBanner();
6
7    // print owings
8    while (elements.hasMoreElements()) {
9      Order each = (Order) elements.nextElement();
10     outstanding += each.getAmount();
11   }
12
13   printDetails(outstanding);
14 }
15
16 void printBanner() {
17   System.out.println("*****************************");
18   System.out.println("****** Customer totals ******");
19   System.out.println("*****************************");
20 }
21
22 void printDetails(double outstanding) {
23   System.out.println("name: " + name);
24   System.out.println("amount: " + outstanding);
25 }
```

Long method - recipe #1: extract method

```
1  void printOwing() {
2    Enumeration elements = orders.elements();
3    double outstanding = 0.0;
4
5    printBanner();
6
7    // print owings
8    while (elements.hasMoreElements()) {
9      Order each = (Order) elements.nextElement();
10     outstanding += each.getAmount();
11   }
12
13   printDetails(outstanding);
14 }
15
16 void printBanner() {
17   System.out.println("*****************************");
18   System.out.println("****** Customer totals ******");
19   System.out.println("*****************************");
20 }
21
22 void printDetails(double outstanding) {
23   System.out.println("name: " + name);
24   System.out.println("amount: " + outstanding);
25 }
```

```
1  void printOwing() {
2    printBanner();
3    double outstanding = getOutstanding();
4    printDetails(outstanding);
5  }
6
7  void printBanner() {
8    System.out.println("*****************************");
9    System.out.println("****** Customer totals ******");
10   System.out.println("*****************************");
11 }
12
13 void printDetails(double outstanding) {
14   System.out.println("name: " + name);
15   System.out.println("amount: " + outstanding);
16 }
17
18 double getOutstanding() {
19   Enumeration elements = orders.elements();
20   double outstanding = 0.0;
21   while (elements.hasMoreElements()) {
22     Order each = (Order) elements.nextElement();
23     outstanding += each.getAmount();
24   }
25   return outstanding;
26 }
```

Long method - recipe #2: Reduce Local Variables and Parameters Before Extracting a Method

If local variables and parameters interfere with extracting a method, use

- Replace Temp with Query,
- Introduce Parameter Object or
- Preserve Whole Object.

Long method - recipe #2 -1: **Replace Temp with Query**

Problem

You place the result of an expression in a local variable for later use in your code.

```
double calculateTotal() {
  double basePrice = quantity * itemPrice;
  if (basePrice > 1000) {
    return basePrice * 0.95;
  }
  else {
    return basePrice * 0.98;
  }
}
```

Solution

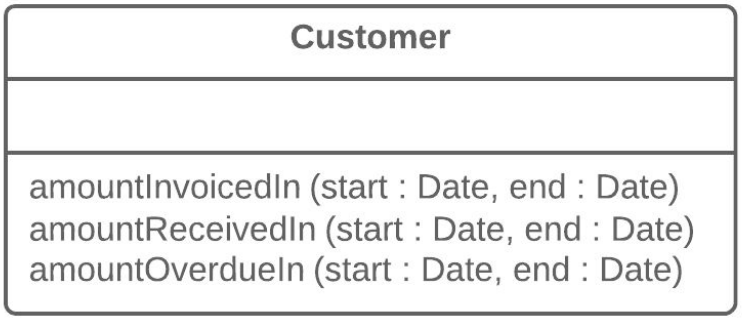
Move the entire expression to a separate method and return the result from it. Query the method instead of using a variable. Incorporate the new method in other methods, if necessary.

```
double calculateTotal() {
  if (basePrice() > 1000) {
    return basePrice() * 0.95;
  }
  else {
    return basePrice() * 0.98;
  }
}
double basePrice() {
  return quantity * itemPrice;
}
```

Long method - recipe #2 -2: **Introduce Parameter Object**

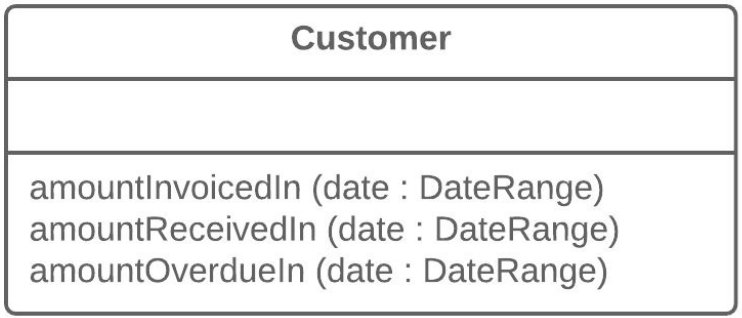
Problem

Your methods contain a repeating group of parameters.



Solution

Replace these parameters with an object.



Long method - recipe #2 -3: **Preserve Whole Object**

Problem

You get several values from an object and then pass them as parameters to a method.

```
int low = daysTempRange.getLow();
int high = daysTempRange.getHigh();
boolean withinPlan = plan.withinRange(low, high);
```

Solution

Instead, try passing the whole object.

```
boolean withinPlan = plan.withinRange(daysTempRange);
```

Long method - recipe #3: **Replace Method with Method Object**

If none of the previous recipes help, try moving the entire method to a separate object via **Replace Method with Method Object.**

Long method - recipe #3: **Replace Method with Method Object**

☹ Problem

You have a long method in which the local variables are so intertwined that you can't apply *Extract Method*.

```
class Order {
  // ...
  public double price() {
    double primaryBasePrice;
    double secondaryBasePrice;
    double tertiaryBasePrice;
    // Perform long computation.
  }
}
```

😊 Solution

Transform the method into a separate class so that the local variables become fields of the class. Then you can split the method into several methods within the same class.

```
class Order {
  // ...
  public double price() {
    return new PriceCalculator(this).compute();
  }
}

class PriceCalculator {
  private double primaryBasePrice;
  private double secondaryBasePrice;
  private double tertiaryBasePrice;

  public PriceCalculator(Order order) {
    // Copy relevant information from the
    // order object.
  }

  public double compute() {
    // Perform long computation.
  }
}
```

Long method - recipe #4: **Conditionals and Loops**

Conditional operators and loops are a good clue that code can be moved to a separate method. For conditionals, use **Decompose Conditional**.
If loops are in the way, try **Extract Method**.

Long method - recipe #4-1: Decompose Conditional

☹ Problem

You have a complex conditional (`if-then` / `else` or `switch`).

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

☺ Solution

Decompose the complicated parts of the conditional into separate methods: the condition, `then` and `else` .

```
if (isSummer(date)) {
  charge = summerCharge(quantity);
}
else {
  charge = winterCharge(quantity);
}
```

Long method - recipe #4-2: Extract method (in loop)

☹ Problem

You have a code fragment that can be grouped together.

```
void printProperties(List users) {
  for (int i = 0; i < users.size(); i++) {
    String result = "";
    result += users.get(i).getName();
    result += " ";
    result += users.get(i).getAge();
    System.out.println(result);

    // ...
  }
}
```

☺ Solution

Move this code to a separate new method (or function) and replace the old code with a call to the method.

```
void printProperties(List users) {
  for (User user : users) {
    System.out.println(getProperties(user));

    // ...
  }
}

String getProperties(User user)  {
  return user.getName() + " " + user.getAge();
}
```

Long method - Payoff

Among all types of object-oriented code, **classes with short methods live longest.**

The longer a method or function is, the harder it becomes to understand and maintain it.

In addition, long methods offer the perfect hiding place for unwanted duplicate code.

References

Code and examples are from: https://refactoring.guru/