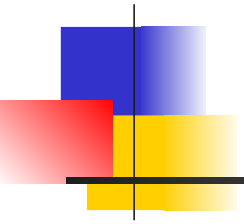

Refactoring Signs & Patterns

Rename Symbol

Assign a more descriptive, meaningful name to a variable, method, class, or package.

Motivation: you find a better, more descriptive name for a variable, method, class, or package.

Code evolves over time, The purpose of some piece of code may change so the original name isn't quite right. Or, you think of a better, more descriptive name.

Mechanics: use an IDE's Refactor -> Rename feature to consistently change the name. Don't use search and replace, which may change unintended matches.

Rename Example

BankAccount has a property for the available balance.

```
@property
def available(self):
    """Get the value of the available balance"""
    holds = sum([check.value
                 for check in self.pending_checks])
    return (self.balance
            - max(self.min_balance, holds))
```

"**available**" could have many interpretations. Is the BankAccount *available*? *Available* for what?

Rename the method to **available_balance.**

Extract Method

Extract a block of code as a separate method.

Motivation: a) method is long and difficult to understand, b) a code block can be reused by several methods.

Mechanics: see references. Selecting which variables should be parameters, return value, or surrounding scope are key steps.

Example: extract logic for computing movie rental price from long "statement( )" method.

Inline Temp

You have a local variable that is assigned to and then used only once. The expression is not complicated.

Solution: Improve readability by putting the expression right where it is used (without assigning to a temp var).

Motivation: a) excess assignment to temps makes code harder to read, b) the assignment to temp is getting in way of other refactorings.

See Also: Introduce Explanatory Variable which is the opposite of this!

Move Method

A method uses more members of another class than members of it's own class.

Solution: *Move it to the other class.*

Motivation: reduces coupling and often makes the code simpler and classes more coherent.

Mechanics: see references.

Example: computing price of a movie rental depends on rental data, not customer info. So move it to the rental class.

Introduce Explaining Variable

You have a complicated expression, making it hard to understand the intent of the code.

Solution: Assign result of part of the expression to a local variable whose name describes the meaning.

Motivation: reduces coupling and often makes the code simpler and classes more coherent.

Mechanics: let the IDE do it! Just select the part of statement to extract and choose Refactor -> assign to local variable or Refactor -> extract local variable.

Example

```
if (
  Calendar.getInstance().get(Calendar.HOUR_OF_DAY)
  > 22)
    System.out.println("You should sleep now.");
```

```
int currentHour =
 Calendar.getInstance().get(Calendar.HOUR_OF_DAY);
if (currentHour > 22)
    System.out.println("You should sleep now.");
```

Replace Constructor with Creation Method

Some classes have multiple constructors and their purpose is not clear.

Solution: Replace constructor with static method that create objects, use a name that describe intention of the method.

Motivation: makes creating objects easier to understand.

Mechanics: Define a static method (class method) that creates and returns a new object.

You may have several such methods for different cases.

Symptoms for Refactoring

Sign or signal that you should consider refactoring.

Often called "code smells".

The purpose of refactoring:

- *Make this code easier to read or maintain.*

Symptoms, not Smells

These are "symptoms" or "signs" that code could be hard to verify or maintain.

There are objective criteria for identifying them.

I don't like the phrase "*code smells*"

- code doesn't have a smell
- *smell* is subjective, whereas symptoms are observable characteristics. They are reasonably objective.

Name some "symptoms" or "signs"

Name some signs that code may need refactoring.

1. Duplicate logic or duplicate code.
2.
3.
4.
5.
6.

List of Symptoms

A good online list is:

`https://blog.codinghorror.com/code-smells/`

Chapter 3 of *Refactoring* book has longer explanation.

Chapter 24 of *Code Complete* also has a good list.

Duplicate Code or Duplicate Logic

The #1 symptom.

Solutions:

Extract Method

Pull up Method

Define a strategy that performs the duplicate code.

Other symptoms we already covered

Long method

Large class - class with many methods and attributes

Incohesive class - class with many weakly related or unrelated responsibilities

Long parameter list

Temporary field - a class has an attribute that is used only rarely, and can easily be recreated as needed.

Data Class

A class that is just a holder for data (like a 'struct' in C).

Doesn't have any responsibilities, just get/set methods.

Solution:

Look at how other classes are using the data class.

You may simplify the code by moving behavior to the data class. Use the Move Method or Extract Method.

Eclipse Show References: Right click on class name and choose References -> Project. Shows all places where this class is used.

Python dataclass

Python 3.7 `dataclass` provides automatic constructor and methods for classes that are intended to be data "containers".

A data class is used as a container for related data, or data + data specific methods.

```
from dataclasses import dataclass

@dataclass
class Coordinate:
    x: float
    y: float
```

Lazy Class

A step above Data Class.

Motivation: A lazy class doesn't do enough to justify its existence.

Solution:

Either give the class something to do (Move Method) or eliminate it.

Speculative Generality

"*I think we might need this in the future*".

Design for change is good.

But if it involves a lot of extra code or classes, be critical.

Symptoms: Abstract classes that don't do anything.
Interfaces with only 1 implementation.

Solution:

Collapse class hierarchy by moving behavior.

Exercise

Find the *refactoring symptoms* in this code.

Suggest refactorings.

https://vivekagarwal.wordpress.com/2008/06/21/code-smelling-exercise/

Resources

Refactoring by Martin Fowler (1999).

http://refactoring.com - patterns from *Refactoring* book.

Refactoring to Patterns by Kerievsky (2004).

Chapter 24 "Refactoring" in Code Complete, 2E by McConnell

List of "code smells" (many lists like this):
`https://blog.codinghorror.com/code-smells/`

Refactoring Symptoms & Solutions

List of "code smells"

https://blog.codinghorror.com/code-smells/

Code Smells Cheat Sheet

http://www.industriallogic.com/wp-content/uploads/2005/09/smellstorefactorings.pdf

and blog post "*Smells to Refactorings*"

https://www.industriallogic.com/blog/smells-to-refactorings-cheatsheet/