Code Smells

Agenda

- What is Code Smell
  - Code Smells Defined
- Identify Code Smells in your code
  - Code Smells patterns and Identifiers
- Recipes to remove smells
  - Refactor your code (refactoring Recipes)
- An Example with SonarQube
- QA

Code Smells Defined

- A **code smell** is any characteristic in the source code of a program that possibly **indicates a deeper problem**.
- Determining what is and is not a code smell is subjective, and varies by language, developer, and development methodology.

- The term was popularized by **Kent Beck in the late 1990s**
- Usage of the term increased after it was featured in the book **Refactoring: Improving the Design of Existing Code by Martin Fowler**.

Source: Wikipedia

Code smells Identifiers

Bloaters

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

Preventers

- Divergent Change
- Shotgun Surgery
- Parallel Inheritance Hierarchies

Couplers

- Message Chains
- Middle Man
- Inappropriate Intimacy
- Feature Envy

Abusers

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

Dispensables

- Lazy Class
- Duplicated Code
- Speculative Generality
- Data Class

Ok, My Code has smells, what should I do to get rid of smells?

refactor

What is Refactoring?

“Refactoring is a process of improving the internal structure of our code without impacting its external behavior”

How do we refactor code?

To refactor code , we need to apply refactoring techniques a.k.a recipes

Refactoring Recipes (some of them ☺)

- Extract Class
- Extract Method
- Replace Temp with Query
- Replace Method with Method Object
- Move Method
- Move Field
- Remove Middle Man
- Replace Array With Object
- Extract Subclass
- Decompose Conditional
- And many more ……

Code Smell: Long Method

- *A method contains too many lines of code.*
- *Generally, any method longer than 10 lines should make you start asking questions.*
- 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.

Code Smell: Long Method

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.

Code Smell: Long Method: Treatment

Extract Method

Problem
You have a code fragment that can be grouped together.

```
void printOwing() {
 printBanner();
// Print details.
System.out.println("name: " + name);
System.out.println("amount: " + getOutstanding());
}
```

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

```
void printOwing() {
printBanner();
printDetails(getOutstanding());
}

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

}
```

Code Smell: Long Method: Treatment

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.

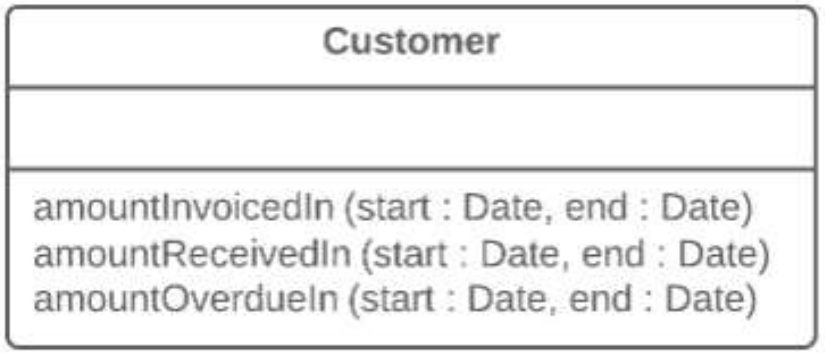
```
double calculateTotal() {
  if (basePrice() > 1000) {
    return basePrice() * 0.95;
  }
  else {
    return basePrice() * 0.98;
  }
}
double basePrice() {
  return quantity * itemPrice;
}
```

Code Smell: Long Method: Treatment

Introduce Parameter Object

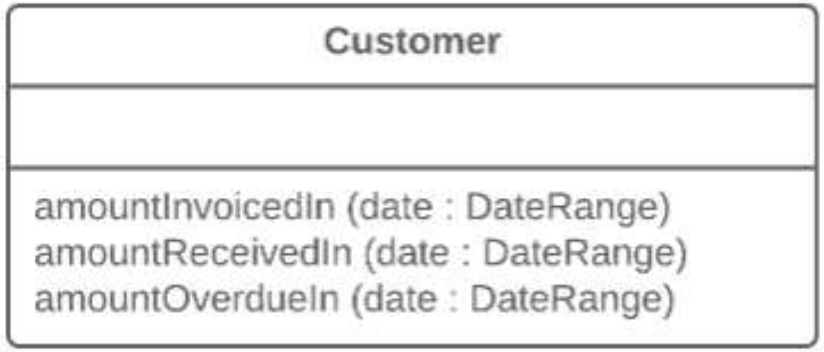
Problem

Your methods contain a repeating group of parameters.



Solution

Replace these parameters with an object.



Code Smell: Long Method: Treatment

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);
```

Code Smell: Large Class

- *A class contains many fields/methods/lines of code.*
- Classes usually start small. But over time, they get bloated as the program grows.
- As is the case with long methods as well, programmers usually find it mentally less taxing to place a new feature in an existing class than to create a new class for the feature.

Code Smell: **Large Class: Treatment**

When a class is wearing too many (functional) hats, think about splitting it up.

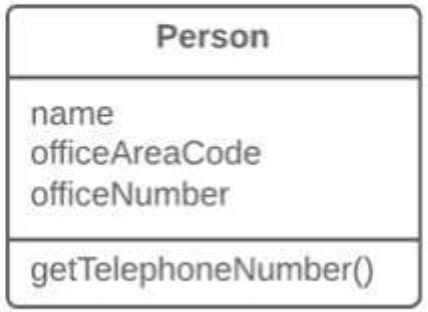
Code Smell: Large Class:Treatment

Extract Class

Extract Class helps if part of the behavior of the large class can be spun off into a separate component.

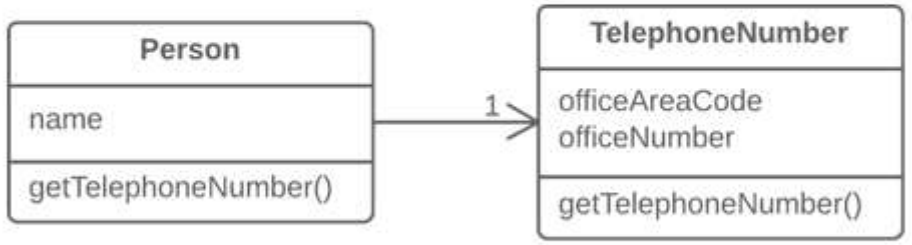
Problem

When one class does the work of two, awkwardness results.



Solution

Instead, create a new class and place the fields and methods responsible for the relevant functionality in it.

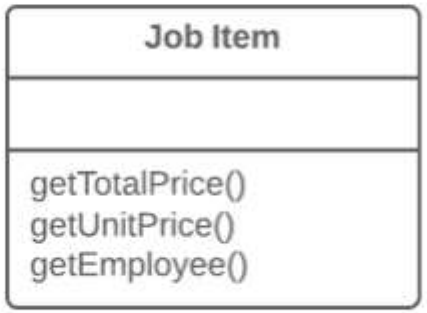


Code Smell: Large Class:Treatment

Extract Subclass helps if part of the behavior of the large class can be implemented in different ways or is used in rare cases.

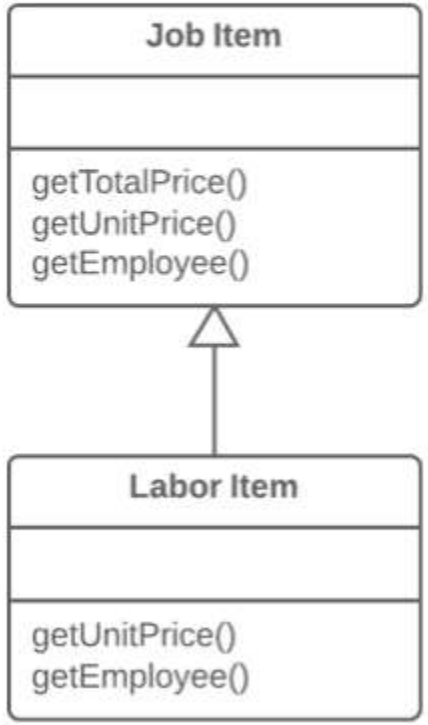
Problem

A class has features that are used only in certain cases.



Solution

Create a subclass and use it in these cases.



The Pay off

- Refactoring of these classes spares developers from needing to remember a large number of attributes for a class.

- In many cases, splitting large classes into parts avoids duplication of code and functionality.

Code Smell: Primitive Obsession

- Use of primitives instead of small objects for simple tasks (such as currency, ranges, special strings for phone numbers, etc.)
- Use of constants for coding information (such as aconstant USER_ADMIN_ROLE = 1 for referring to users with administrator rights.)
- Use of string constants as field names for use in data arrays.

Code Smell: Primitive Obsession

- **Reasons for the Problem**
- Like most other smells, primitive obsessions are born in moments of weakness. "Just a field for storing some data!" the programmer said. Creating a primitive field is so much easier than making a whole new class, right? And so it was done. Then another field was needed and added in the same way. Lo and behold, the class became huge and unwieldy.

Code Smell : Primitive Obsession: Treatment

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);
```

Code Smell:Primitive Obsession:Treatment

Some more examples

```
//Method call Relying on primitives only

addHoliday(7, 4);  //no readability
```

```
drawLine(2, 2, 34, 55, 0, 0, 255);

transferFunds(23423, 33424, 44353)
```

```
//Refactored using Object
Date holiday = new Date(7, 4);
addHoilday(holiday);  // more readable
```

Object-Orientation Abusers

Object-Orientation Abusers

All these smells are incomplete or incorrect application of object-oriented programming principles.

Code Smell: Switch Statement

You have a complex switch operator or sequence of if statements.

Code Smell: Switch Statement:Treatment

To isolate switch and put it in the right class, you may need **Extract Method** and then **Move Method.**

Problem

You have a code fragment that can be grouped together.

```
class Order {
  // ...

  public double calculateTotal() {
    double total = 0;
    for (Product product : getProducts()) {
      total += product.quantity * product.price;
    }

    // Apply regional discounts.
    switch (user.getCountry()) {
      case "US": total *= 0.85; break;
      case "RU": total *= 0.75; break;
      case "CN": total *= 0.9; break;
      // ...
    }

    return total;
  }
```

Solution

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

```
class Order {
  // ...

  public double calculateTotal() {
    double total = 0;
    for (Product product : getProducts()) {
      total += product.quantity * product.price;
    }
    total = applyRegionalDiscounts(total);
    return total;
  }

  public double applyRegionalDiscounts(double total) {
    double result = total;
    switch (user.getCountry()) {
      case "US": result *= 0.85; break;
      case "RU": result *= 0.75; break;
      case "CN": result *= 0.9; break;
      // ...
    }
    return result;
  }
```

Code Smell: Temporary Field

- *Temporary fields get their values (and thus are needed by objects) only under certain circumstances.*
- *Outside of these circumstances, they're empty.*

Code Smell: Temporary Field:Treatment

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.
  }
}
```

Change Preventers

Change Preventers

These smells mean that if you need to change something in one place in your code, you have to make many changes in other places too.

Program development becomes much more complicated and expensive as a result.

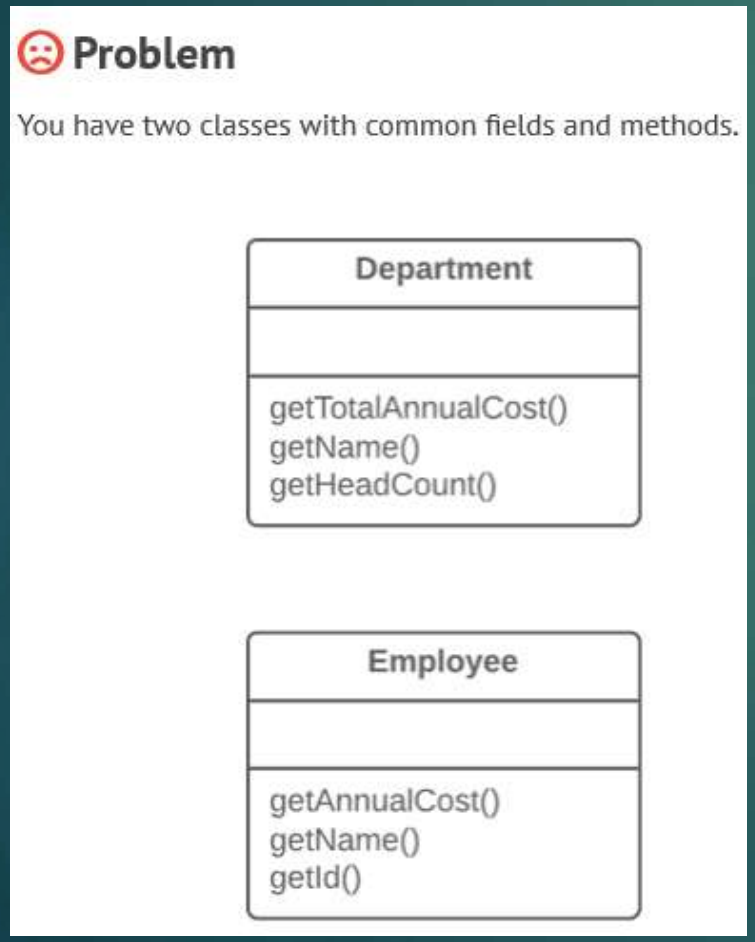
Code Smell: Divergent Change

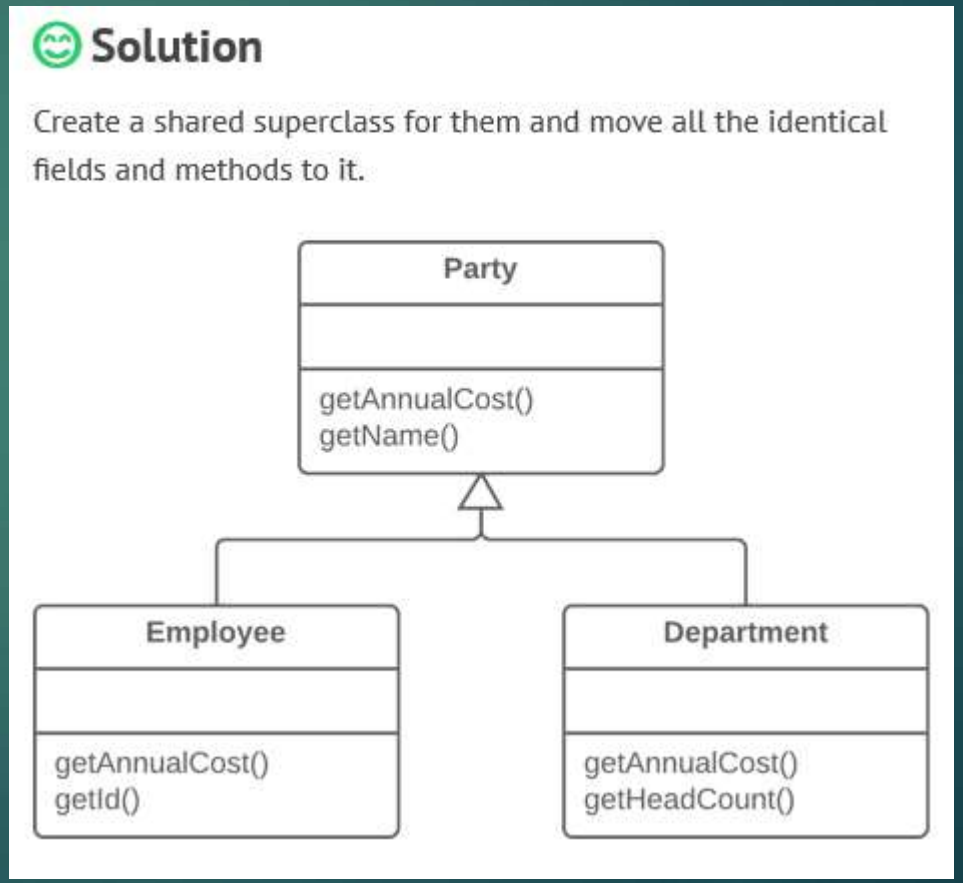
You find yourself having to change many unrelated methods when you make changes to a class.

For example, when adding a new product type you have to change the methods for finding, displaying, and ordering products.

Code Smell: Divergent Change: Treatment

If different classes have the same behavior, you may want to combine the classes through inheritance (**Extract Superclass** and **Extract Subclass**).





Questions?