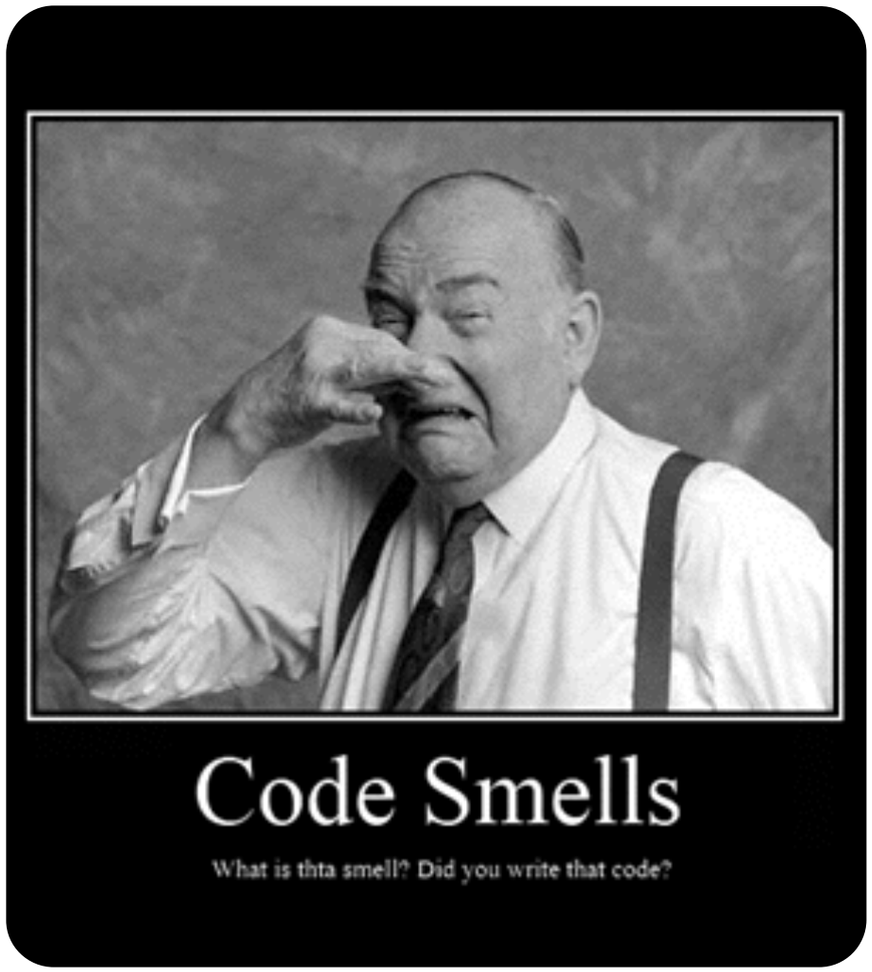
CODE SMELLS

VON AVELINA OTT

„Any fool can write code that computer can understand.
Good programmers write code that humans can understand.“

Martin Fowler

Inhaltsverzeichnis



01

Code Smells

Code Smells

- Anti-Pattern im Programmcode
- Schlechte Programmierform
- Nicht falsch, nur unschön

Code Smells - Problem?

- Schlecht lesbarer Code
- Schlecht wartbarer Code
- Fehleranfällig

Wie kommt es zu Code Smells?

- Schlecht geschriebener Code
  - „von Anfang an“

Rettung von Code Smells?

- „Refactoring“
- „What is Refactoring“ von Martin Fowler

Refactoring

„A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour oft he software.“

von Martin Fowler

02

Refactoring

Verbreitete Code Smells

- Bloaters
- Object-Orientation Abusers
- Change Preventers
- Dispensables
- Couplers

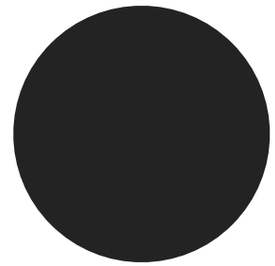
03

Verbreitete Code Smells

Bloaters

Long Method

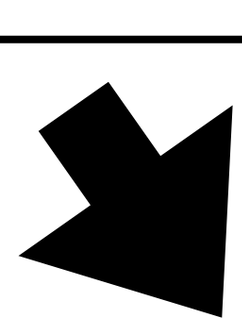
- Problem: Methode ist zu lang/ macht zu viel
  - unübersichtlich —> fehleranfällig
- Lösung:
  - Blöcke in extra Methoden extrahieren
  - komplexe Bedingungen vereinfachen
- Rat: Eine Methode macht nur eine Sache

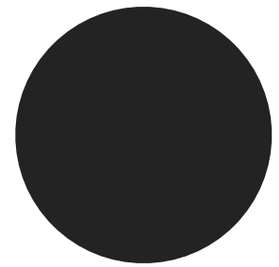


Long Method

Blöcke in extra Methoden extrahieren

```
ausgabe(){
  sammeln_von_Information;
  ausgeben_der_Information;
}
```

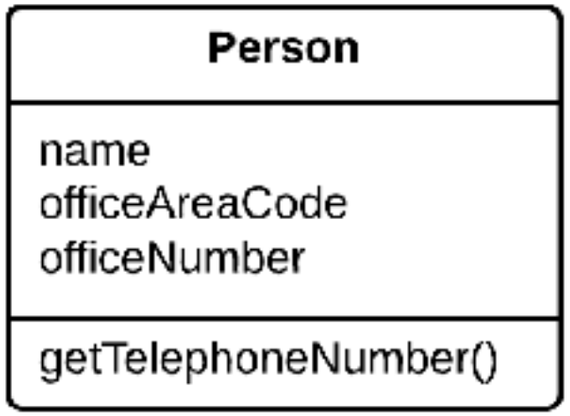




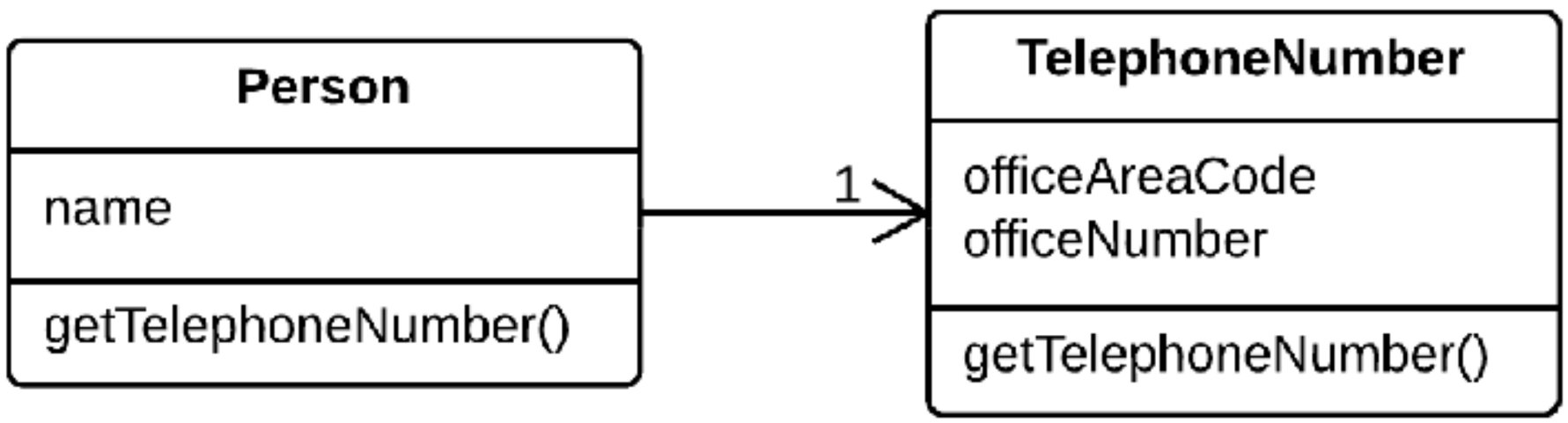

```
function ausgabe(){
  getInformation();
  printInformation();
}
```

Large Class

- Problem: Klasse ist zu lang

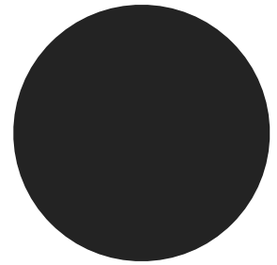


- Lösung:
  - Aufteilung in mehrere Klassen



Long Parameter List

- Problem: Methode hat viele Übergabeparameter
  - schwer zu lesen/testen
- Lösung:
  - Objekte übergeben
    - Aufrufe mit „.getXY()“



Long Parameter List

Objekte übergeben

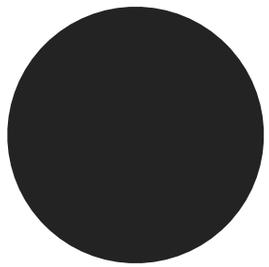
```
checkFahrzeug(int reifen,
bool hupe, int tueren){
  …
}
```

```
checkFahrzeug(vehicle aCar){
  x = aCar.getReifen();
}
```

Object-Orientation Abusers

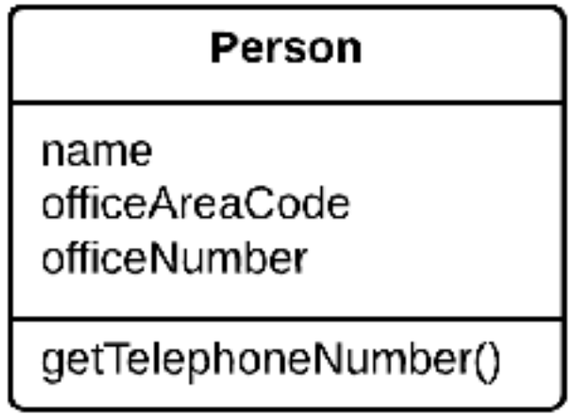
Temporary Field

- Problem: Ein Objekt verwendet eine Variable nur unter bestimmten Umständen
    - Code ist schwer zu verstehen & zu Debugger
- Lösung:
    - Klasse extrahieren

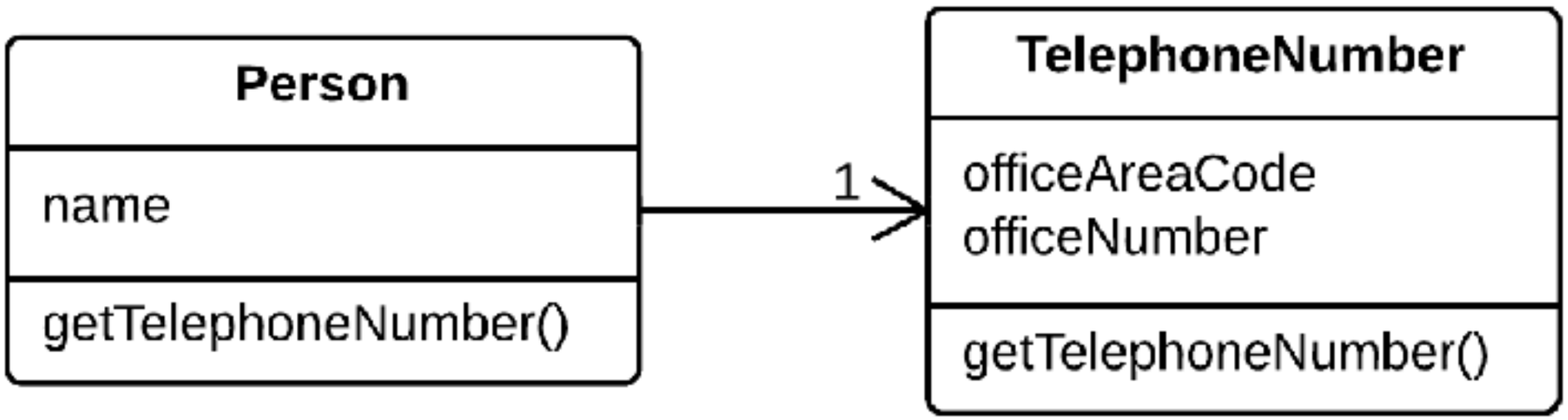


Temporary Field

Problem:



Lösung:



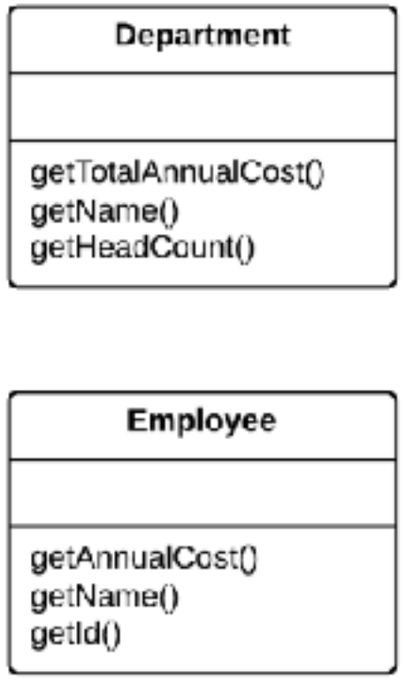
Extract Superclass

Refused Bequest

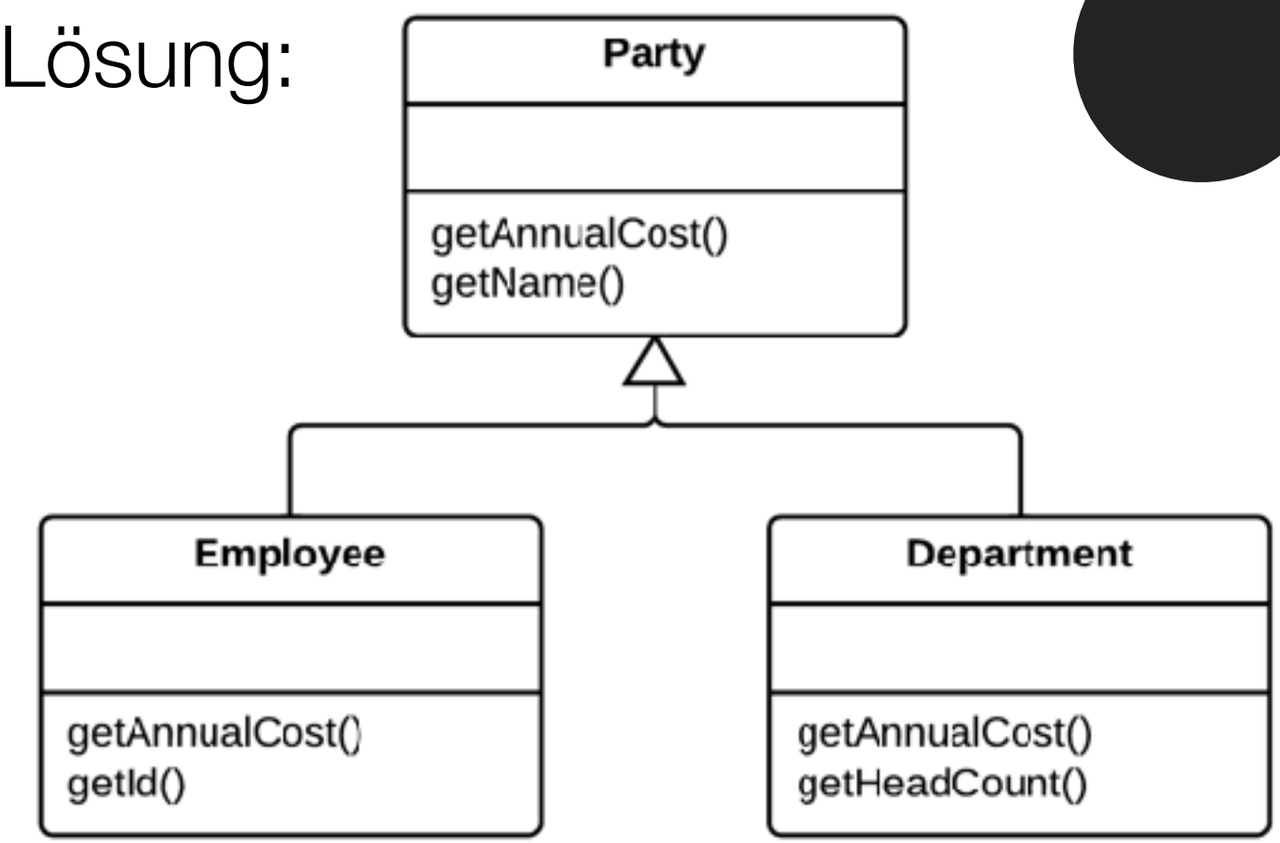
- Problem: Unterklassen brauchen Methoden gar nicht
  - erben von der Oberklasse
- Lösung:
  - Oberklasse extrahieren
  - Ersetzung der Vererbung mit Delegation

Refused Bequest

Problem:



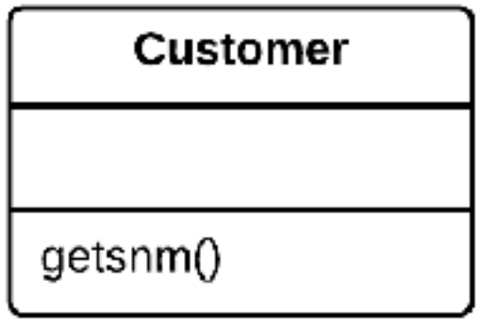
Lösung:



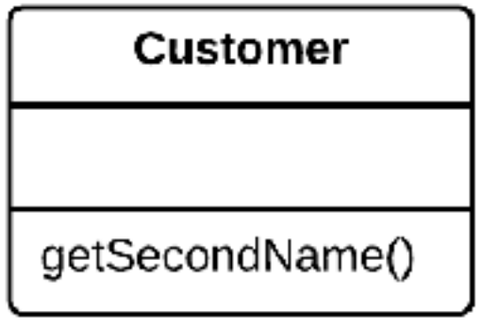
Extract Superclass

Alternativ Classes with Different Interfacer

- Problem: Zwei Klassen haben dieselbe Funktion, aber andere Methodennamen



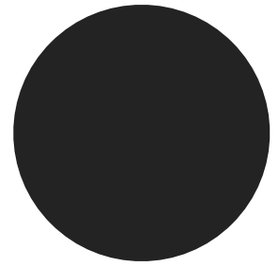
- Lösung:
  - Umbenennen der Methoden



Change Preventers

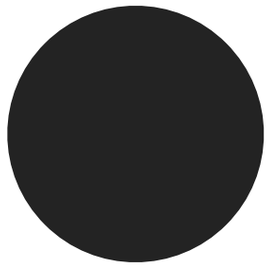
Divergent Change

- Problem: für eine Änderung muss eine Klasse an mehreren Stellen angepasst werden
- Lösung:
  - Klasse extrahieren



Shotgun Surgery

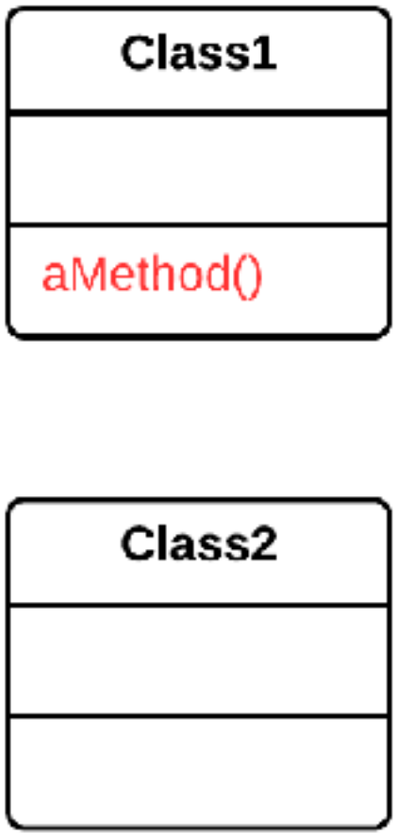
- Problem: für eine Änderung müssen weitere Änderung an vielen Klassen durchgeführt werden
- Lösung:
  - Methode bewegen
  - Feld bewegen



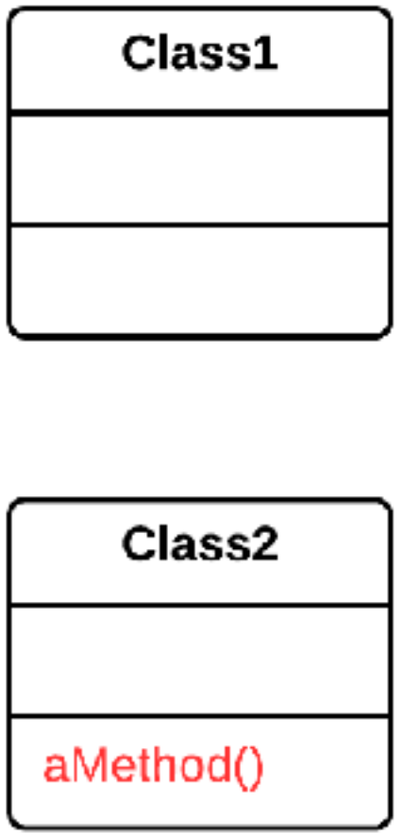
Shotgun Surgery

Methode bewegen

Problem:

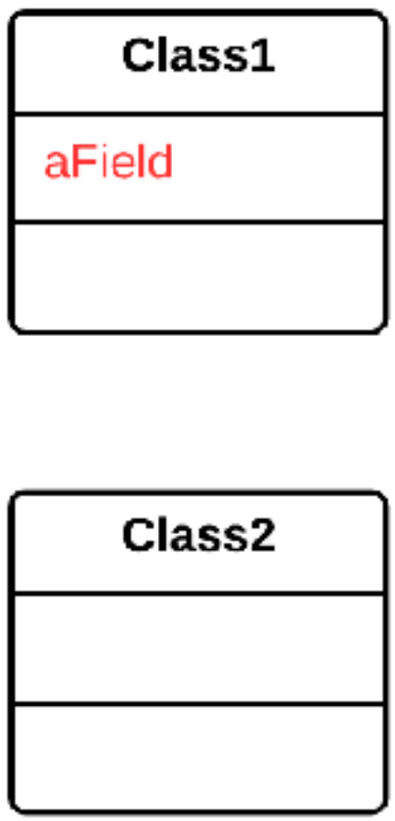


Lösung:

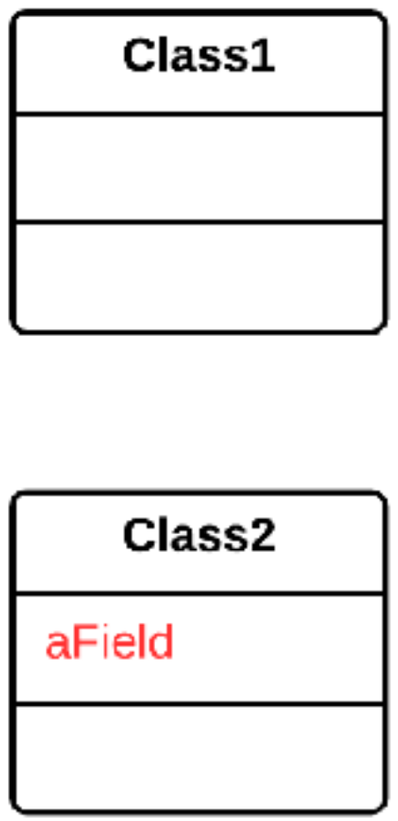


Feld übergeben

Problem:

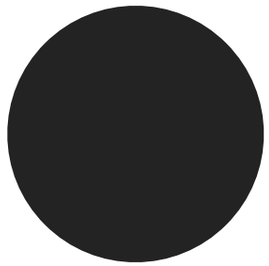


Lösung:



Parallel Inheritance Hierarchies

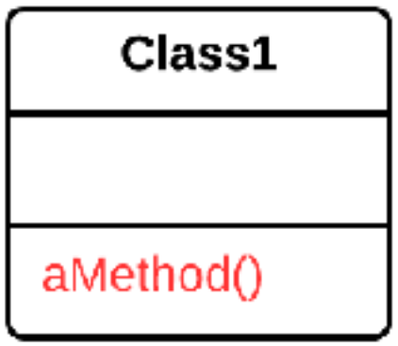
- Problem: jede Unterklasse in der einen Hierarchie hat immer eine Unterklasse in einer anderen Hierarchie
- Lösung:
  - Methode bewegen
  - Feld bewegen



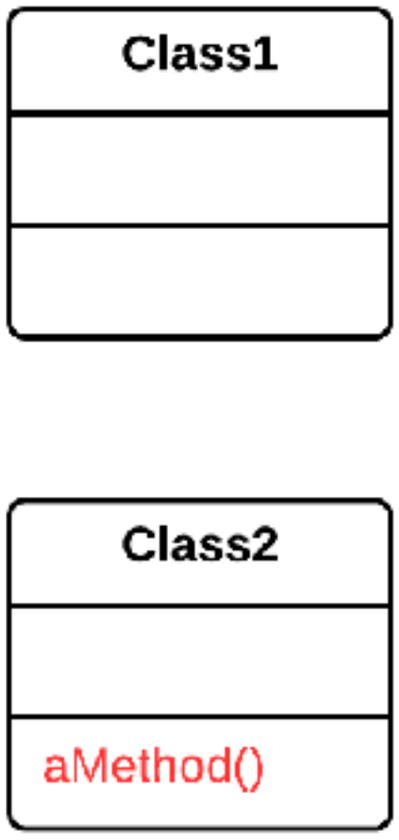
Parallel Inheritance Hierarchies

Methode bewegen

Problem:

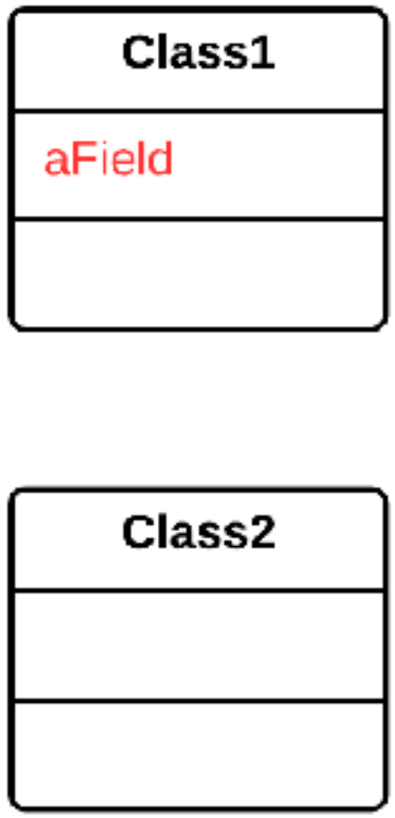


Lösung:

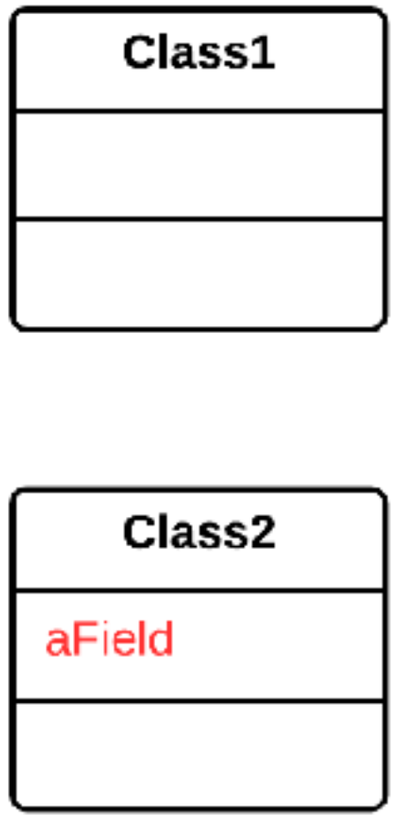


Feld übergeben

Problem:



Lösung:



Dispensables

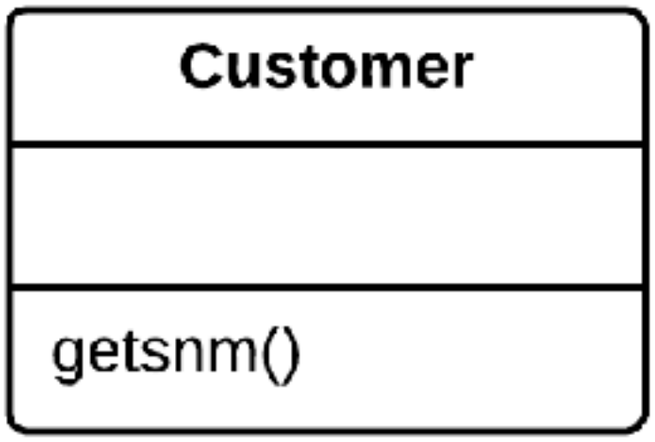
Comments

- Problem: Code ist überfüllt mit Kommentaren
- Lösung:
  - Variable umbenennen mit komplexen Ausdruck
  - Methode umbenennen mit einer Beschreibung des Codes
- Rat:
  - „The best comment is a good name for a method or class.“
  - Kommentare sind Signale für Code Smells

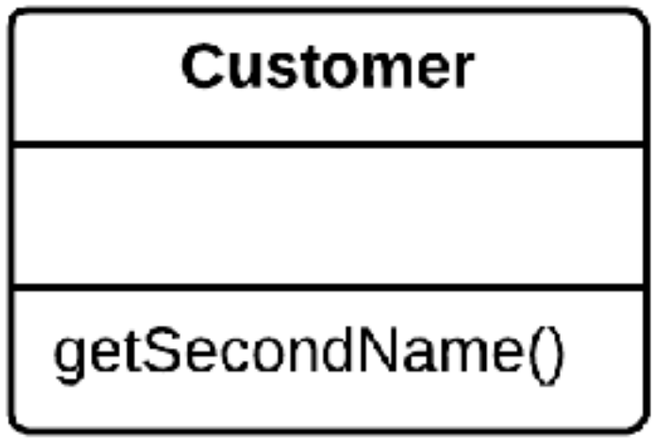
Comments

Methode umbenennen

Problem:

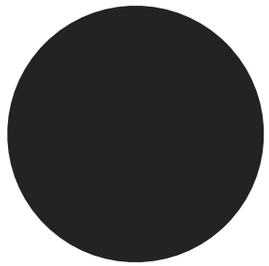


Lösung:



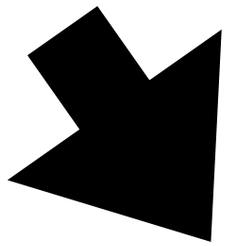
Duplicate Code

- Problem: gleicher Code kommt an verschiedenen Stellen vor
- Lösung:
  - Methode extrahieren
  - Klasse extrahieren



Duplicate Code

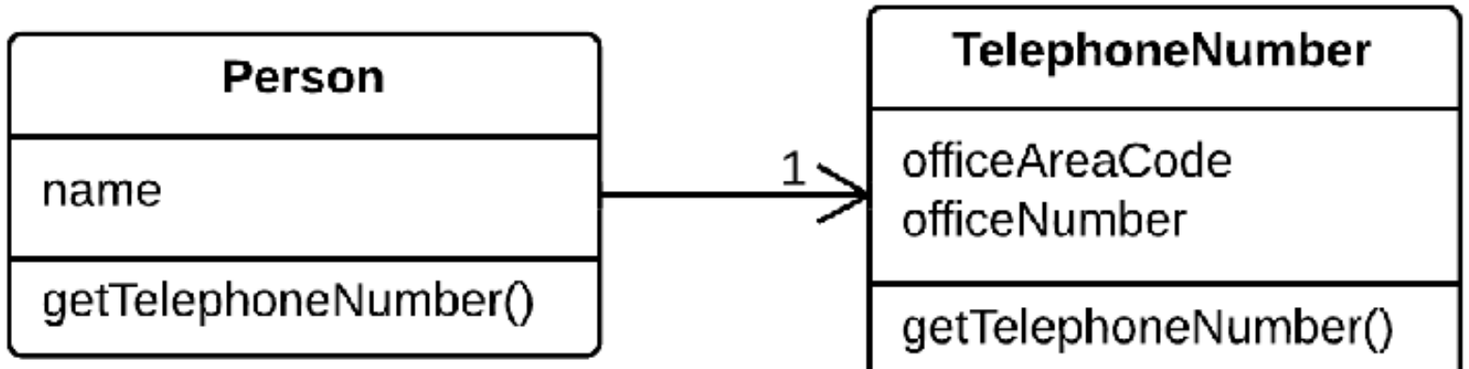
```
if x < 10 && x > 5 then
  setPostion(x,0);
else
  setPostion(0,0);
end

if y < 10 && y > 5 then
  setPostion(y,0);
else
  setPostion(0,0);
end
```



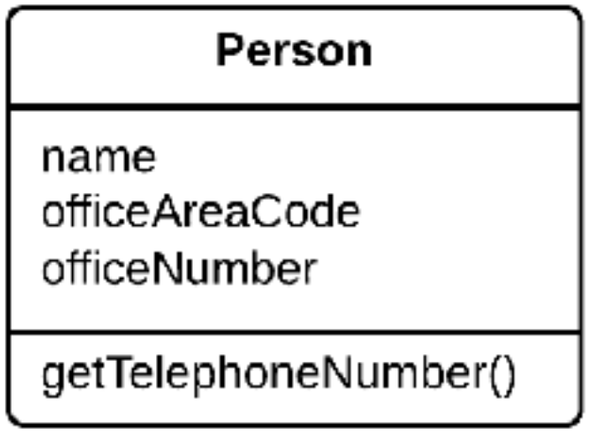

```
function isPosition(){
  if (x < 10 && x > 5){
    setPostion(x,0);
  }else{
    setPostion(0,0);
  }
}
```

Lazy Class

- Problem: Klasse macht zu wenig für ihre Existenz



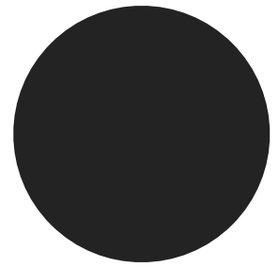
- Lösung:
  - mit einer anderen Klasse verbinden



Data Class

- Problem: Klasse mit Feldern ohne Funktionalität
- Lösung:
  - Feld abkapseln
  - Methode bewegen oder extrahieren

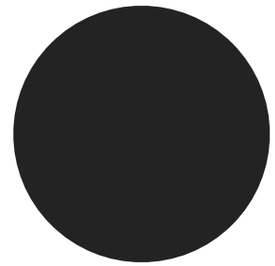
```
class Person {
  public String name;
}
```




```
class Person {
  private String name;

  public String getName() {
    return name;
  }
  public void setName(String arg) {
    name = arg;
  }
}
```

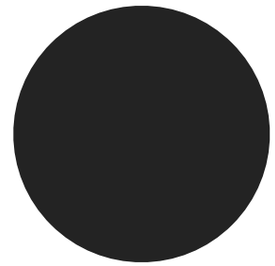
Dead Code

- Problem:
  - Code der nicht (mehr) verwendet wird
- Lösung:
  - Löschen des Codes



Speculative Generality

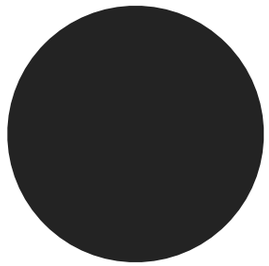
- Problem: Methode ist zu lang/ macht zu viel
  - unübersichtlich —> fehleranfällig
- Lösung:
  - Blöcke in extra Methoden extrahieren
  - komplexe Bedingungen vereinfachen
- Rat: Eine Methode macht nur eine Sache



Couplers

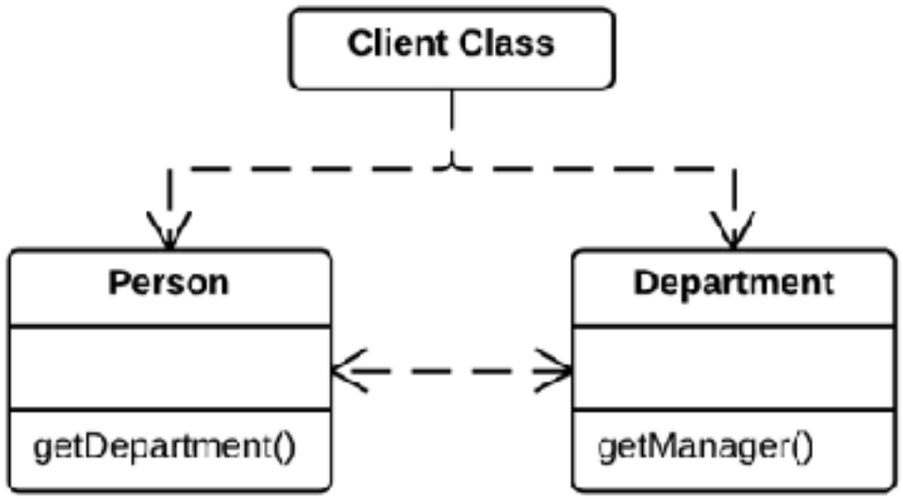
Feature Envy

- Problem: eine Methode interessiert sich mehr für die Eigenschaften & Daten einer anderen Klasse als für ihre eigene
- Lösung:
  - Methode bewegen in eine andere Klasse

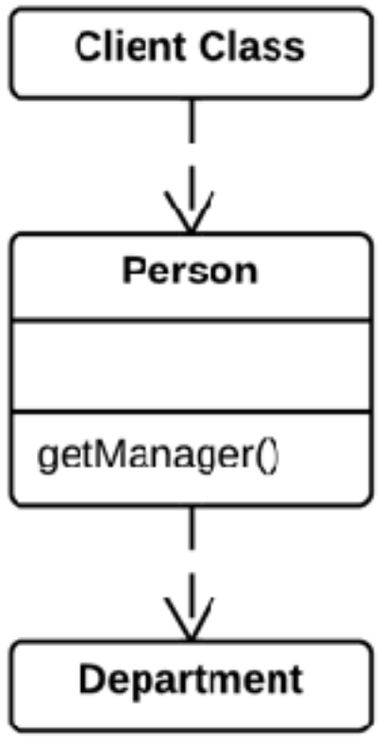


Message Chains

- Problem: viele Funktionen rufen sich gegenseitig auf

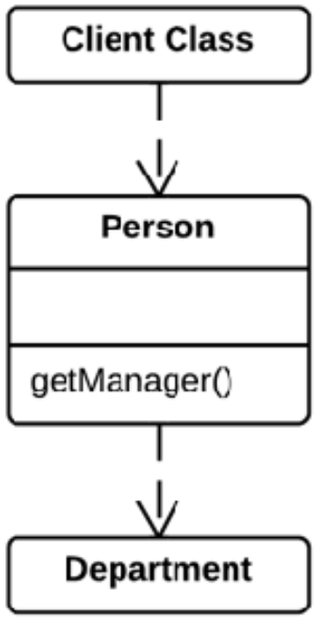


- Lösung:
  - Methode bewegen
  - Delegation verstecken

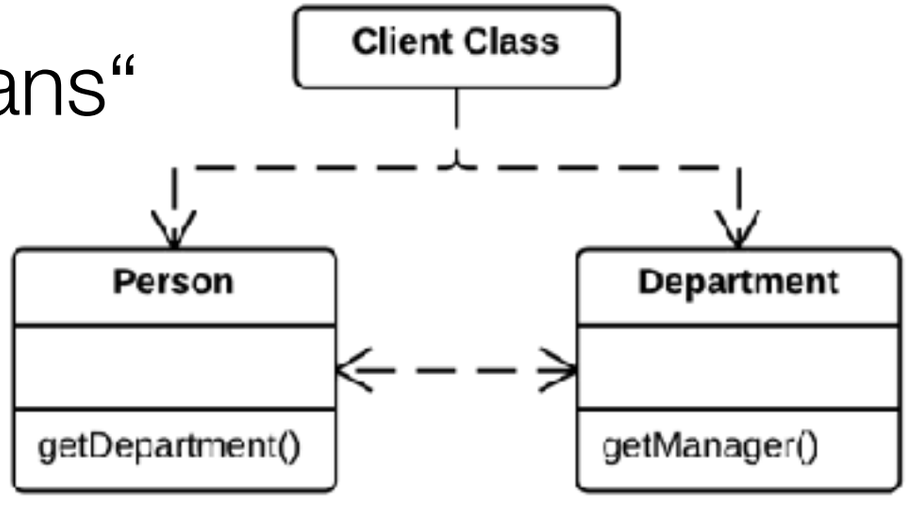


Middle Man

- Problem: eine Klasse delegiert alle Methodenaufrufe an eine andere Klasse



- Lösung:
  - Entfernen des „Middle Mans“



04

Fazit

Fazit

- einfache Verbesserungen mit großen Ergebnis

Literaturvorschläge

- https://sourcemaking.com/refactoring/smells
- https://refactoring.com/catalog/
- http://www.industriallogic.com/wp-content/uploads/2005/09/smellstorefactorings.pdf
- Refactoring: Improving the Design of Existing Code (Object Technology Series) by Martin Fowler & Kent Beck

05

Literaturvorschläge

Quellen

https://martinfowler.com/bliki/CodeSmell.html
https://sourcemaking.com/refactoring/smells
https://martinfowler.com/books/refactoring.html

Bilder:
http://www.rogoit.de/webdesign-typo3-blog-duisburg/wp-content/uploads/2014/05/clean-code-smells-heuristiken-3.png

06

Quellen

Danke für die Aufmerksamkeit!

Habt ihr noch Fragen?