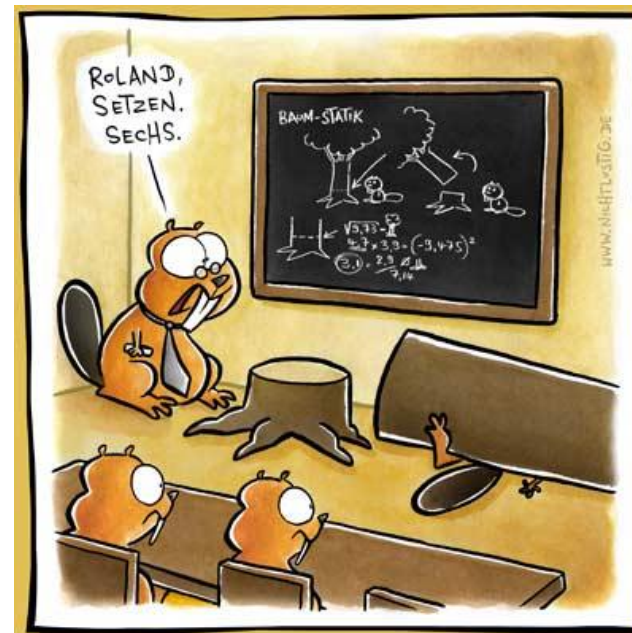


8. Refactoring

Dr.-Ing. Clemens Reichmann

Clemens.Reichmann@vector.com,
Tel. 0721-91430-200

Institut für Technik der Informationsverarbeitung
Fakultät für Elektrotechnik & Informationstechnik
Universität Karlsruhe (TH)



8.1 Quellcodequalität

8.2 „Bad Smells“ in Code

8.3 Refactoring



Literatur:

Martin Fowler: Refactoring. Wie Sie das Design vorhandener Software verbessern, Addison-Wesley Verlag, ISBN 3-8273-1630-8

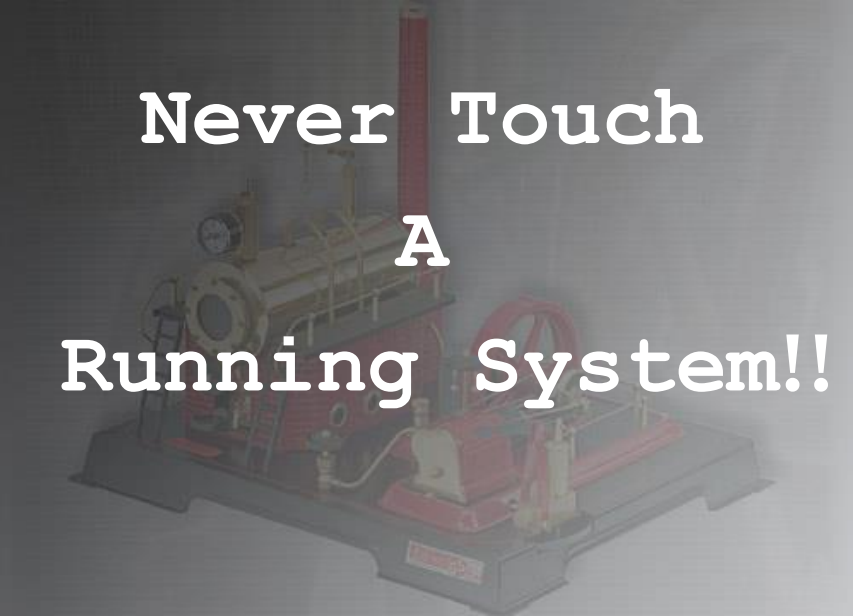


William C. Wake: Refactoring Workbook, Addison-Wesley, ISBN 0-321-10929-5

Joshua Kerievsky: Refactoring To Patterns, Addison-Wesley, ISBN 0321213351

- Wartbarkeit (Änderungsfreundlichkeit)
 - Wartung heißt meistens erweitern und anpassen; alles, was sich in einem Programm ändern könnte, in abstrakten Klassen kapseln, um flexibel zu sein. Die Merkregel hier lautet: Programmieren mit Schnittstellen, nicht mit Implementierungen
- Übersichtlichkeit
 - Codierungsrichtlinien
 - An bekannten Konzepten orientieren (Entwurfsmuster)
- Wiederverwendbarkeit
 - Wiederverwendung heißt den gleichen Code in einer anderen Umgebung für die selbe Funktion einzusetzen.
 - Beim Entwurf andere Einsatzgebiete erkennen und bedenken
- Portabilität
 - Portieren von Software bedeutet das Übertragen eines Programmes in eine andere Betriebssystemumgebung
 - Kapseln aller betriebssystemkritischen API's und Vorgehensweisen
 - Möglichst keine Abweichungen von Standards

8.2 Bad Smells in Code



Never Touch
A
Running System!!

Woran erkennt man „schlechten“ Code?

- Wir würden gerne lernen, wie man Code verbessert.
- Doch woran erkennt man schlechten Code?
- Martin Fowler und Kent Beck haben eine Liste von typischen „Bad Smells“ angelegt.
- Dennoch ist es schwer zu sagen, wann genau es „riecht.“

- A.) Zuviel des Guten
- B.) Viel wird geändert, wenig verbessert.
- C.) Viel Arbeit, wenig Lohn
- D.) Arbeiten für andere
- E.) Schlechte Angewohnheiten

- a) Duplizierter Code (Duplicated Code)
- b) Lange Parameterliste (Long Parameter List)
- c) Lange Methode (Long Method)
- d) Große Klasse (Large Class)
- e) Switch Statements (Switch Statements)
- f) Alternative Klassen mit unterschiedlichen Interfaces
(Alternative Classes with Different Interfaces)
- g) Kommentare (Comments)
- h) Toter Code (dead code)

- Gleiche Codestruktur an mehr als einer Stelle
- Problem:
 - Redundanz
 - Bei jeder Änderung muß dieser Code an mehreren Stellen geändert werden.
- Refactorings
 - Extract Method
 - Pull Up Field
 - Form Template Method
 - Substitute Algorithm

- Methode hat sehr viele Parameter
- Problem:
 - schwer zu verstehen
 - schwer zu handhaben
 - häufige Änderungen nötig, wenn man mehr Daten braucht
 - Methode sollte sich ihre Daten selbst besorgen.
- Refactorings
 - Replace Parameter with Method
 - Preserve Whole Object
 - Introduce Parameter Object

- Methode ist sehr groß
- Problem:
 - schwer zu verstehen:
 - Was genau macht die Methode?
 - Wie genau macht sie es?
 - viele Kommentare nötig
- Refactorings
 - Extract Method
 - Replace Temp with Query
 - Introduce Parameter Object
 - Preserve Whole Object
 - Replace Method with Method Object

- Klasse hat sehr viele Instanzvariablen / zu viel Code
- Problem:
 - unübersichtlich
 - Duplizierter Code ist gut möglich.
 - Möglicherweise werden nicht alle Variablen von jedem Objekt genutzt.
- Refactorings
 - Extract Class
 - Extract Subclass

- Code enthält viele switch (oder case) statements
- Problem:
 - Duplizierter Code ist sehr wahrscheinlich.
 - Oft kommt das gleiche Statement an verschiedenen Stellen des Programms vor.
 - schlechte Erweiterbarkeit
 - Bei jeder Änderung muß man alle switch statements suchen und ändern.

- Klassen/Methoden tun das Gleiche mit unterschiedlichen Signaturen
- Problem:
 - Unterschied schwer zu erkennen, da sie den gleichen Namen haben

- Kommentar erklärt, was die Methode tut
- Problem:
 - Code offenbar nicht verständlich genug
 - Methodename nicht aussagekräftig
 - Methodename ist redundant z.B.
`schedule.add(course)` nicht:
`schedule.addCourse(course)`
- Refactorings
 - Extract Method
 - Rename Method
 - Introduce Assertion

- Eine Methode, Variable, Codefragment, Klasse, etc. wird nirgends verwendet
- Refactorings
 - Delete the code

B.) Viel wird geändert, wenig verbessert.

- a) Verteilte Änderungen (Shotgun Surgery)
- b) Divergente Änderungen (Divergent Change)
- c) Parallele Erbschafts-Hierarchien (Parallel Inheritance Hierarchies)

- Eine Änderung betrifft viele Klassen
- Problem:
 - Es ist schwer zu sagen, wo man überall ändern muß.
 - Möglicherweise übersieht man eine wichtige Änderung.
- Refactorings
 - Use Move Method and Move Field to put all the changes into a single class.
 - Often you can use Inline Class to bring a whole bunch of behavior together.

- Verschiedene Änderungsarten betreffen gleiche Klasse
- Problem:
 - Objekte unterscheiden sich nicht genug voneinander.
 - Änderungen sind schwer zu lokalisieren.
- Refactorings
 - Identify everything that changes for a particular cause and use Extract Class to put them all together.

- Spezialfall von Verteilten Änderungen
- Problem:
 - Jedes Mal, wenn man eine Unterklasse einer Klasse bildet, muß man auch eine Unterklasse einer anderen Klasse bilden.
- Refactorings
 - Use Move Method and Move Field to combine the hierarchies into one.

- a) Faule Klasse (Lazy Class)
- b) Temporäres Feld (Temporary Field)
- c) Unvollständige Library-Klasse (Incomplete Library Class)
- d) Spekulative Allgemeinheit (Speculative Generality)
- e) Daten-Klasse (Data Class)

- Klasse tut fast nichts mehr
- Problem:
 - Verstehen und entwickeln einer Klasse kostet Geld
 - Klasse wird dank Refactoring nicht mehr gebraucht
- Refactorings
 - Collapse Hierarchy
 - Inline Class

- Instanzvariablen werden manchmal nicht benutzt
- Problem:
 - Code schwer zu verstehen
 - Man erwartet, daß ein Objekt alle Variablen nutzt.
- Refactorings
 - Replace Field with Temp

- In einer Library-Klasse fehlen wichtige Funktionen
- Problem:
 - Niemand kann 100%ig sagen, welche Funktionen man später benötigt.
 - Es ist schwer, eine Library-Klasse im Nachhinein zu ändern.

- Features werden auf Verdacht implementiert
- Problem:
 - Niemand weiß, ob sie wirklich einmal benötigt werden.
 - möglicherweise unnötiger Zeitaufwand
 - schwerer zu verstehen
 - komplexeres Design
- Refactorings
 - If you have abstract classes that aren't doing much use Collapse Hierarchy
 - Remove unnecessary delegation with Inline Class
 - Methods with unused parameters - Remove Parameter
 - Methods named with odd abstract names should be brought down to earth with Rename Method

- Klasse enthält nur Instanzvariablen
- Problem:
 - Klasse hat nur get- und set-Methoden.
 - Gefahr einer Faulen Klasse
 - Manche Variablen werden möglicherweise nicht gebraucht.
- Refactorings
 - Move in behavior with Move Method

- a) Neid (Feature Envy)
- b) Mittlerer Mann (Middle Man)
- c) Nachrichten-Ketten (Message Chains)

- Methode kümmert sich hauptsächlich um andere Klasse
- Problem:
 - Sehr viele get-Methoden beziehen sich auf anderes Objekt.
 - Methode will offenbar woanders sein.
- Refactorings
 - Move Method May need to use Extract Method first, then Move Method

- Klasse delegiert meiste Arbeit an andere Klasse
- Problem:
 - Die meisten Methoden lösen Aufgaben nicht selbst.
 - Verkapselung und Delegation gehören zusammen; dies sollte doch nicht zu weit gehen.
- Refactorings
 - Remove Middle Man
 - If only a few methods aren't doing much, use Inline Method
 - You could also consider turning the middle man into a subclass with Replace Delegation with Inheritance

- Client fragt Objekt nach anderem Objekt, das wiederum nach einem anderen fragt etc.
- Problem:
 - Viele get-Methoden werden hintereinander ausgeführt.
 - Bei jeder Änderung der Beziehungen zueinander muß der Client sich auch ändern.
- Refactorings
 - Hide Delegate
 - Or try using Extract Method and then Move Method to move it down the chain.

- a) Daten-Klumpen (Data Clumps)
- b) Verweigertes Vermächtnis (Refused Bequest)
- c) Unangemessene Intimität (Inappropriate Intimacy)
- d) Fixierung auf primitive Datentypen (Primitive Obsession)

- Bestimmte Daten kommen immer gemeinsam vor
- Problem:
 - erhöhte Komplexität
 - lange Parameterlisten
 - Klumpen sollten **eigenes Objekt** bilden.
- Refactorings
 - Turn the clumps into an object with Extract Class
 - Then continue the refactoring with Introduce Parameter Object
 - or Preserve Whole Object

- Unterklasse nutzt geerbte Variablen/Methoden nicht
- Problem:
 - Hierarchie ist vermutlich falsch.
 - Oberklasse enthält **nicht nur allgemeine Daten**.
 - Ein Objekt der Unterklasse kann nicht einfach als Objekt der Oberklasse eingesetzt werden.
- Refactorings
 - Most of the time that's fine, just don't use what you don't need. Occasionally you'll need to create a new sibling class and use Push Down Method and Push Down Field
 - The smell is worst if a subclass is reusing behavior but does not want to support the interface of the superclass. In this case use Replace Inheritance with Delegation

- Zwei Klassen stöbern oft in den jeweils anderen Daten
- Problem:
 - Gemeinsame Interessen sollten eigene Klasse bilden.
- Refactorings
 - Move Method Move Field
 - Change Bidirectional Association to Unidirectional Association
 - Extract Class
 - Hide Delegate
 - Replace Inheritance with Delegation

- viele Variablen von primitiven Datentypen
- Problem:
 - schlechte Erweiterbarkeit
 - primitive Datentypen leisten weniger als Objekte
- Refactorings
 - Replace Data Value with Object
 - Replace Type Code with Class
 - Replace Type Code with Subclasses
 - Replace Type Code with State/Strategy
 - If you have a group of fields that should go together, use Extract Class
 - If the primitives occur in a param lists use Introduce Parameter Object
 - When working with an array consider Replace Array With Object

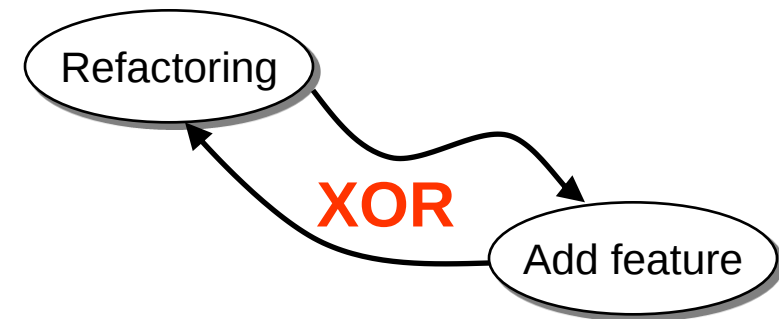
- Jeder muß selbst entscheiden, wann es „riecht“.
- Die Liste der Smells von Fowler und Beck ist nur eine Heuristik, um solche Makel zu entlarven.
- Man kann die Smells (auch Antipattern genannt) durch **Refactoring** entfernen.



- **Refactoring**: deutsch meist unzutreffend mit Refaktorisierung und manchmal mit Refaktoriierung bezeichnet
 - Refactoring ist ein wichtiger Bestandteil des Extreme Programming (XP) Entwicklungsprozess
 - Quelltext umgestaltet mit dem Ziel:
 - Lesbarkeit
 - Übersichtlichkeit
 - Verständlichkeit
 - Erweiterbarkeit
 - Vermeidung von Redundanz
 - Testbarkeit
- Ein Programm ändern ohne dessen Funktionalität zu ändern

- Ein tiefes Verständnis des eingesetzten Programmierparadigmas, z.B. OOP
- **Unit-Tests**, die als **Regressionstests** belegen können, dass das Programm sich immer noch gleich verhält und durch das Refactoring nicht versehentlich Fehler eingeführt wurden
- Werkzeuge, insbesondere integrierte **Entwicklungsumgebungen**, die eine Unterstützung bei der Durchführung von Refaktorisierungen anbieten.

- Ziel des Refactorings bestimmen
 - Beseitigen von Codesmells
 - Vorbereiten des bestehenden Codes, um neues Feature zu implementieren



- Test: Funktionsüberprüfung des bestehenden Codes
 - Achtung: Check des Tests durch temporäres Verändern des Codes
- Stufen der Refactorings bestimmen
- Durchführung
- Test zeigt funktionale Äquivalenz (soweit Test ausgearbeitet ist)

- A) Methoden zusammenstellen (Composing Methods)
- B) Eigenschaften zwischen Objekten verschieben (Moving Features Between Objects)
- C) Daten organisieren (Organizing Data)
- D) Bedingte Ausdrücke vereinfachen (Simplifying Conditional Expressions)
- E) Methodenaufrufe vereinfachen (Making Method Calls Simpler)
- F) Umgang mit Generalisierung (Dealing with Generalization)
- G) Weiteres

A) Methoden zusammenstellen (Composing Methods)

- Ein Codefragment kann zusammengefasst werden
- Setze die Fragmente in eine Methode, deren Namen den Zweck kennzeichnet.

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



```
void printOwing() {  
    printBanner();  
    printDetails(getOutstanding());  
}  
  
void printDetails (double outstanding) {  
    System.out.println ("name:          " + _name);  
    System.out.println ("amount        " + outstanding);  
}
```

- Eine temporäre Variable wird nur mit einem einfachen Ausdruck initialisiert und die Variable stört andere Refactorings
- Ersetzte Verweise auf die Variable mit dem Ausdruck

```
double basePrice = anOrder.basePrice();  
return (basePrice > 1000)
```



```
return (anOrder.basePrice() > 1000)
```

- Eine temporäre Variable wird nur mit einem einfachen Ausdruck initialisiert und die Variable stört andere Refactorings.
- Ersetzte Verweise auf die Variable mit dem Ausdruck

```
double basePrice = a + b + 100;  
return (basePrice > 1000)
```



```
return ((a+b+100) > 1000)
```

- Ersetze temporäre Variable durch Anfragemethode (Replace Temp with Query)

```
double basePrice = _quantity * _itemPrice;  
if (basePrice > 1000)  
    return basePrice * 0.95;  
else  
    return basePrice * 0.98;
```



```
if (basePrice() > 1000)  
    return basePrice() * 0.95;  
else  
    return basePrice() * 0.98;  
...  
double basePrice() {  
    return _quantity * _itemPrice;  
}
```

- Es gibt einen komplizierten Ausdruck
- Setze das Ergebnis des Ausdrucks (oder Teile) in eine Zwischenvariable. Der Name der temporären Variable erklärt den Zweck

```
if ( (platform.toUpperCase().indexOf("MAC") > -1) &&  
    (browser.toUpperCase().indexOf("IE") > -1) &&  
    wasInitialized() && resize > 0 ) {  
    // do something
```

```
}
```



```
final boolean isMacOs = platform.toUpperCase().indexOf("MAC") > -1;  
final boolean isIEBrowser = browser.toUpperCase().indexOf("IE") > -1;  
final boolean wasResized = resize > 0;  
if (isMacOs && isIEBrowser && wasInitialized() && wasResized) {  
    // do something  
}
```

- Der Wert einer temporäre Variable ändert sich **mehr als einmal** und die Variable ist kein Schleifenzähler oder temporärer Iterator.
- Erstelle eine neue temporäre Variable für jede Zuweisung.

```
double temp = 2 * (_height + _width);  
System.out.println (temp);  
temp = _height * _width;  
System.out.println (temp);
```



```
final double perimeter = 2 * (_height + _width);  
System.out.println (perimeter);  
final double area = _height * _width;  
System.out.println (area);
```

- Entferne Zuweisung an Parametervariable (Remove Assignments to Parameters)
- Programmcode belegt Parametervariablen
- Nutze eine temporäre Variable

```
int discount (int inputVal, int quantity, int yearToDate) {  
    if (inputVal > 50)  
        inputVal -= 2;  
}
```

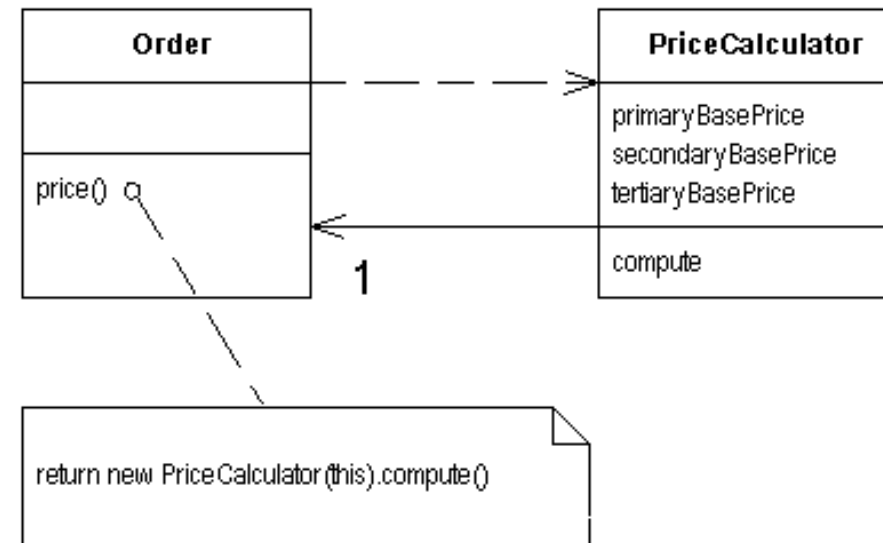
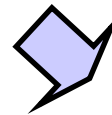


```
int discount (int inputVal, int quantity, int yearToDate) {  
    int result = inputVal;  
    if (inputVal > 50)  
        result -= 2;  
}
```

- Ersetze eine Methode durch ein Methoden-Objekt (Replace Method with Method Object)

class Order...

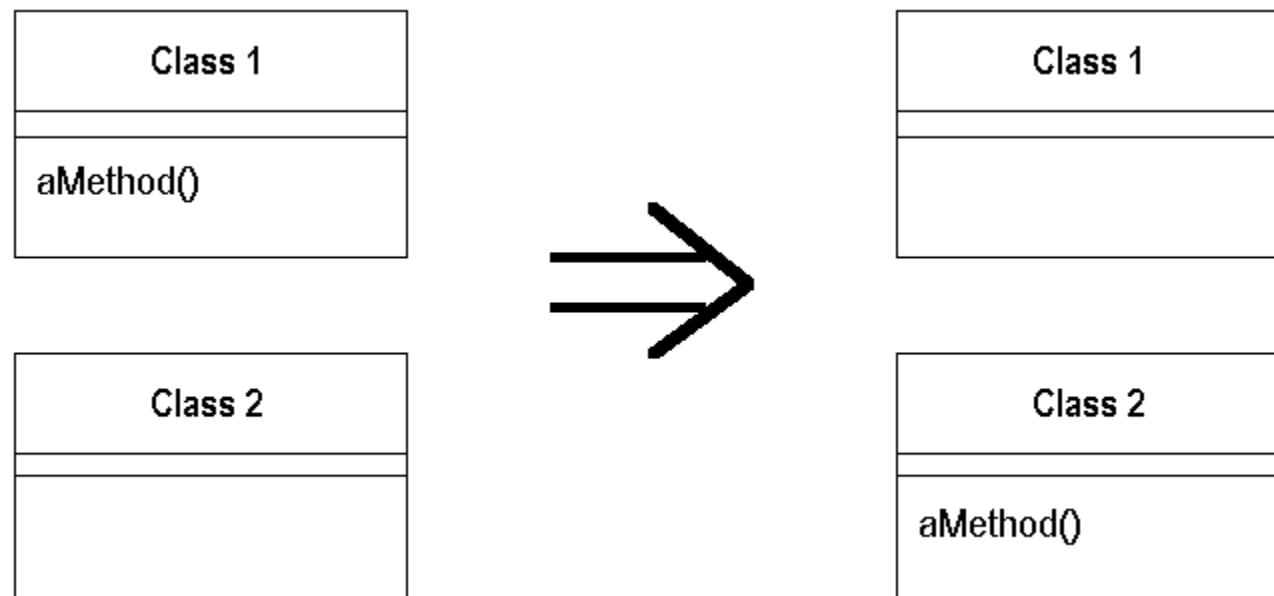
```
double price() {  
    double primaryBasePrice;  
    double secondaryBasePrice;  
    double tertiaryBasePrice;  
    // long computation; ...  
}
```



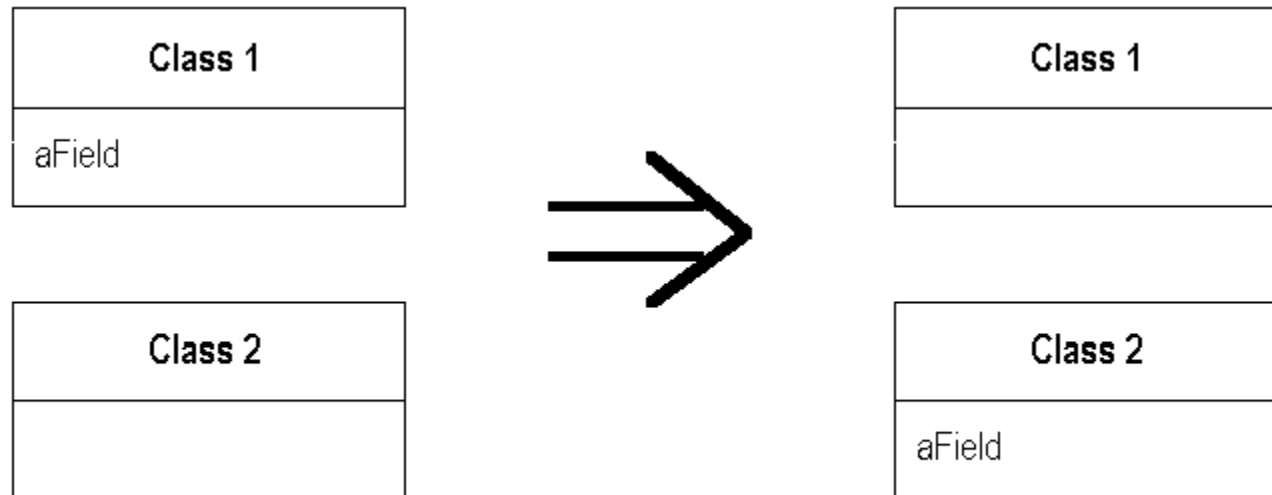
- Der Algorithmus soll gegen einen verständlicheren ausgetauscht werden
- Ersetze den Körper einer Methode mit dem neuen Algorithmus

B) Eigenschaften zwischen Objekten verschieben

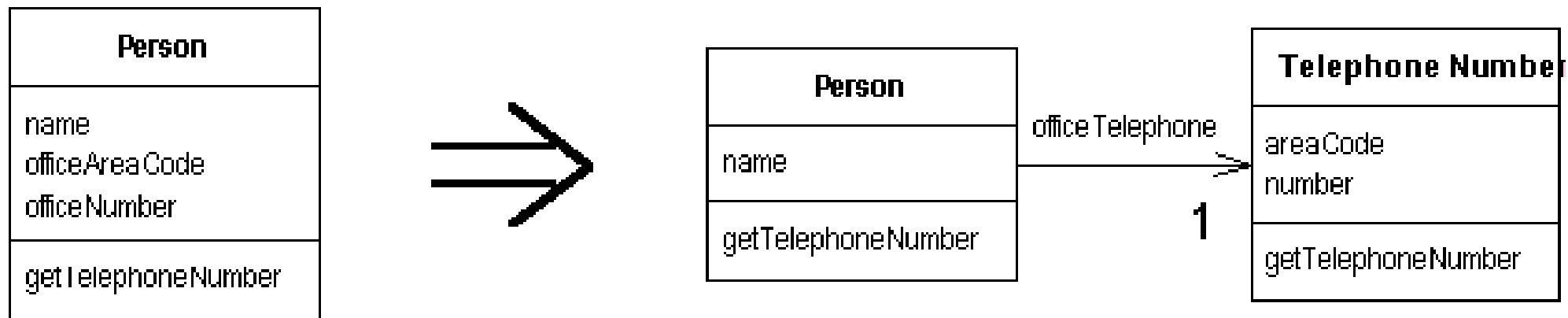
- Eine Methode wird von einer anderen Klasse mehr verwendet als von der definierenden Klasse
- Erzeuge eine Methode mit ähnlichem Rumpf in der Klasse, die die Eigenschaft am meisten nutzt. Delegiere in der alten Methode an die neue Implementierung, oder lösche sie



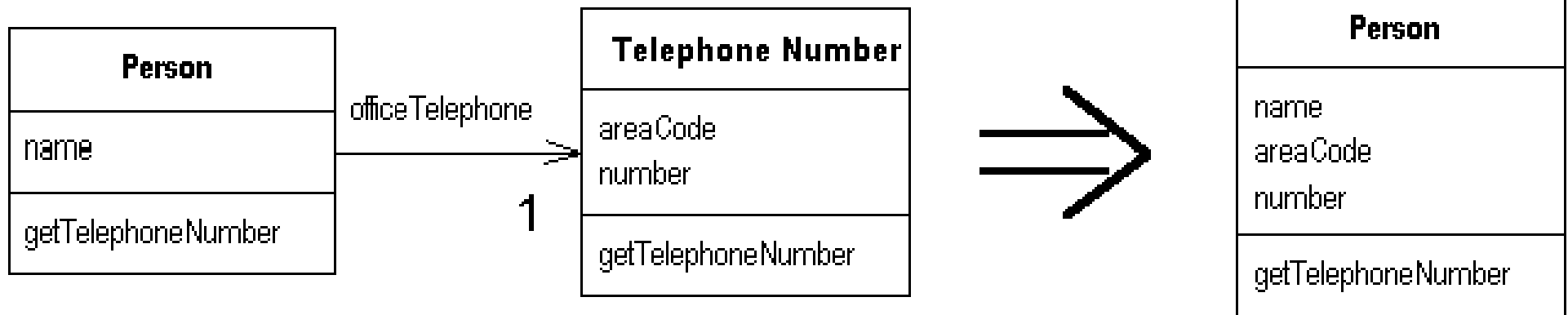
- Eine andere Klasse nutzt ein Attribut mehr als die eigene Klasse, die das Attribut definiert.
- Erzeuge ein neues Attribut in der Ziel-Klasse und ändere die Nutzer.



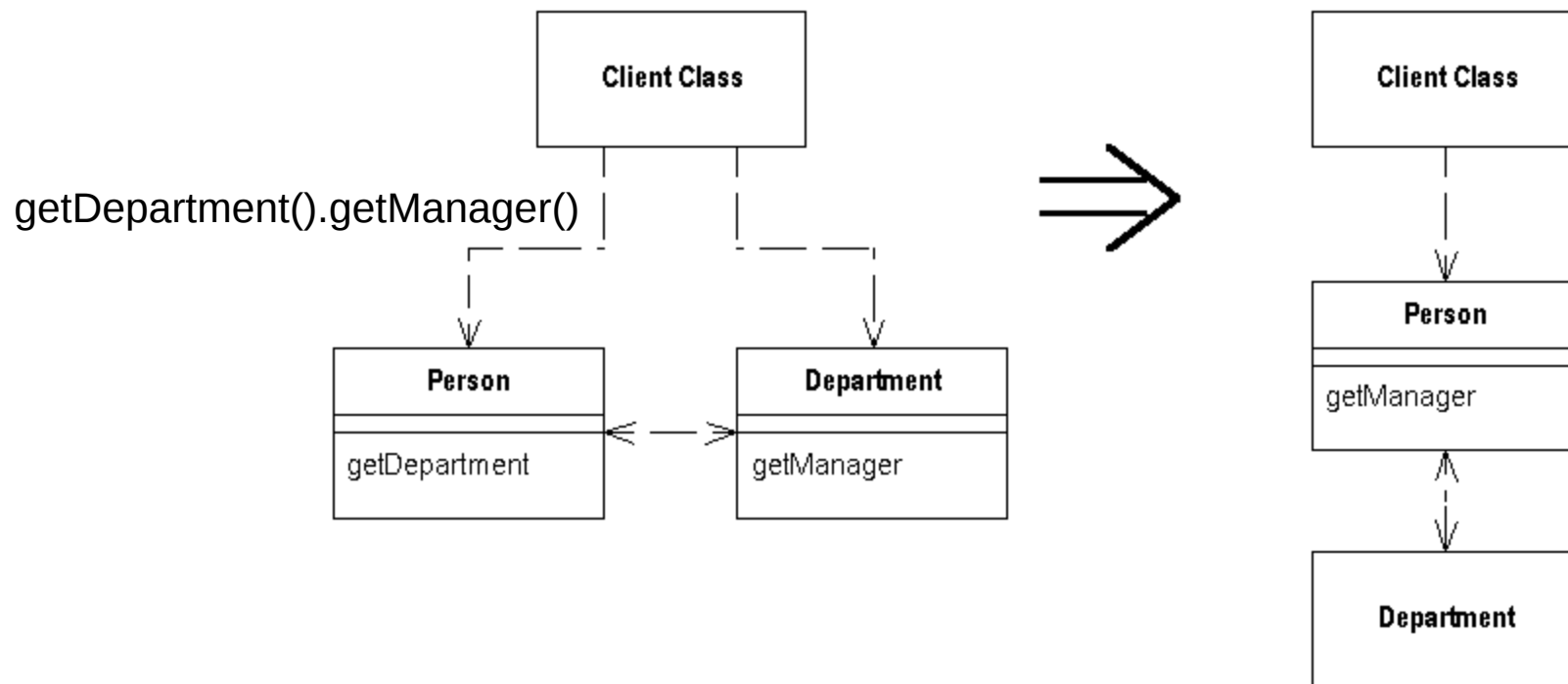
- Eine Klasse macht Arbeit, die zwei Klassen tun sollten
- Erzeuge eine neue Klasse. Bewege die relevanten Attribute und Methoden von der alten Klasse in die neue



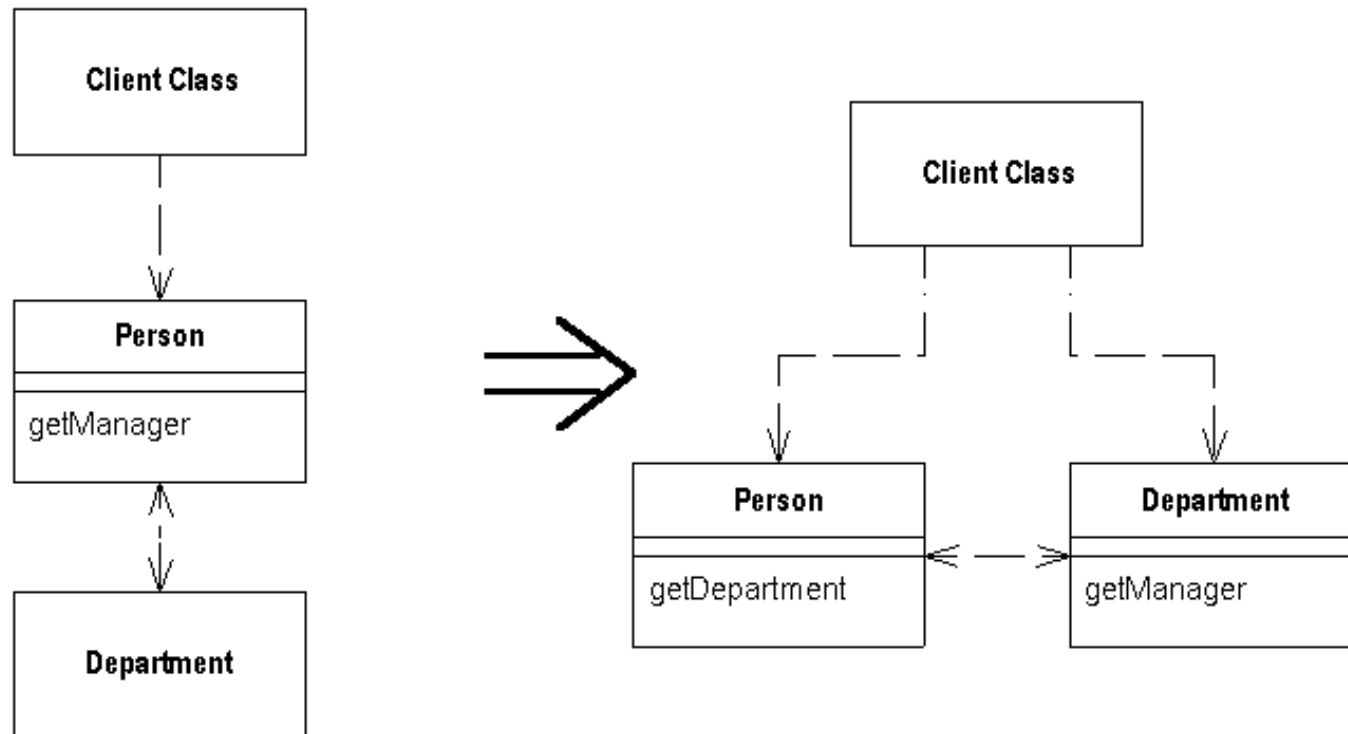
- Eine Klasse macht nicht viel.
- Verschiebe alle Eigenschaften in eine andere Klasse und lösche sie



- Ein Client ruft über eine Delegation eine Methode auf
- Erzeuge auf dem Server die Methode



- Eine Klasse macht zu viele einfache Delegationen
- Der Client soll selbst die Methoden des Delegates aufrufen.



- Eine nicht änderbare Klasse benötigt eine zusätzliche Methode.
- Erzeuge die Methode beim Client mit einem Verweis auf die Klasse als erstes Argument.

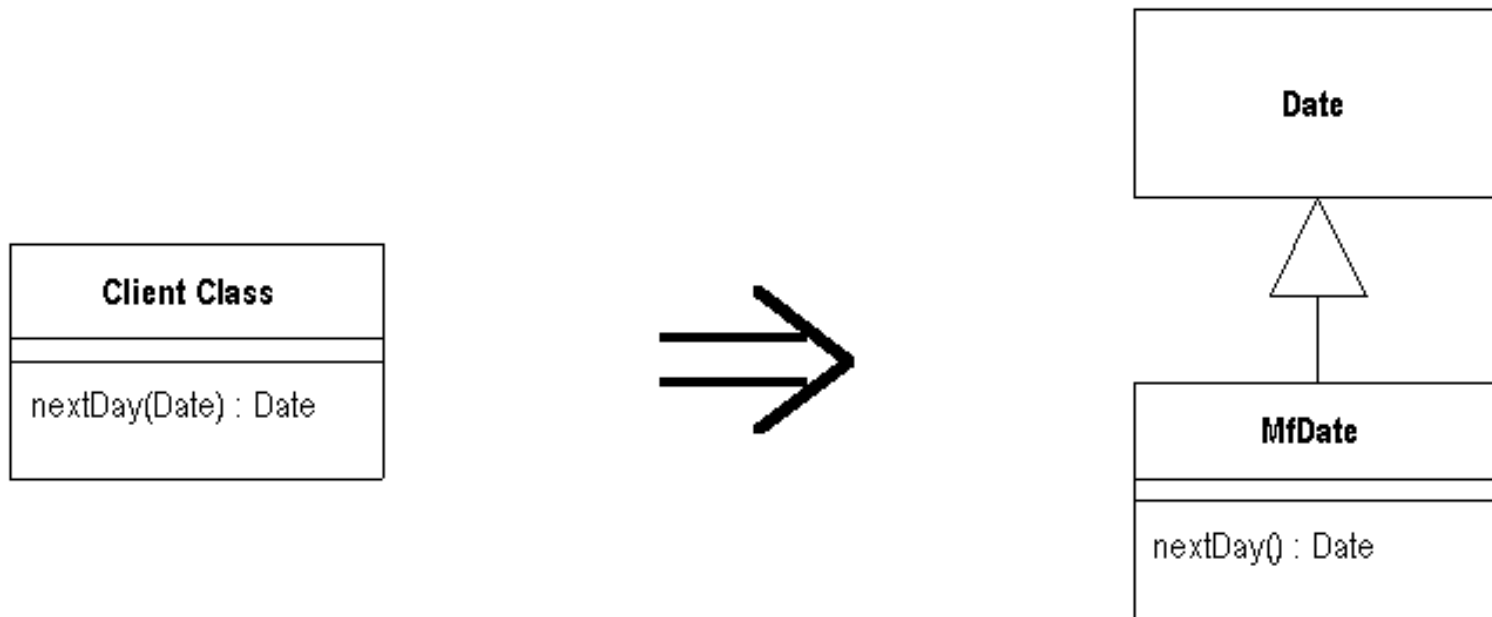
```
Date newStart = new Date (    previousEnd.getYear(),  
                             previousEnd.getMonth(),  
                             previousEnd.getDate() + 1);
```



```
Date newStart = nextDay(previousEnd);
```

```
private static Date nextDay(Date arg) {  
    return new Date (arg.getYear(),arg.getMonth(), arg.getDate() + 1);  
}
```

- Eine Serverklasse braucht mehrere weitere Methoden aber man kann die Serverklasse nicht verändern
- Die neuen Methoden werden in der neuen Klasse erstellt und die Serverklasse erbt von der neuen Klasse



C) Daten organisieren

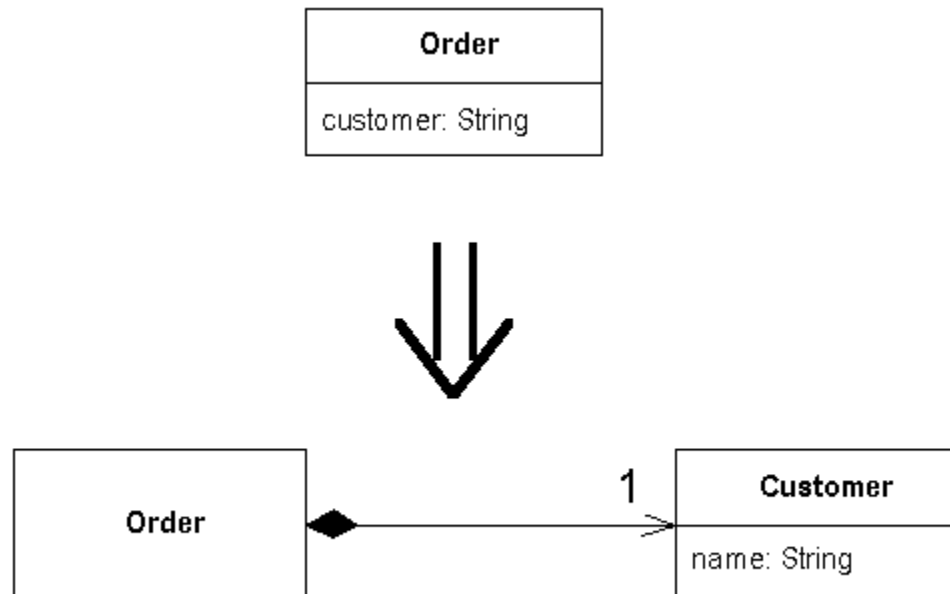
- Die Klasse greift direkt auf ihre Attribute zu, doch die Verbindung an dieses Attribut wird unbequem
- Erzeuge Getter und Setter für das Feld und nutze nur diese zum Attributzugriff

```
private int _low, _high;  
boolean includes (int arg) {  
    return arg >= _low && arg <= _high;  
}
```

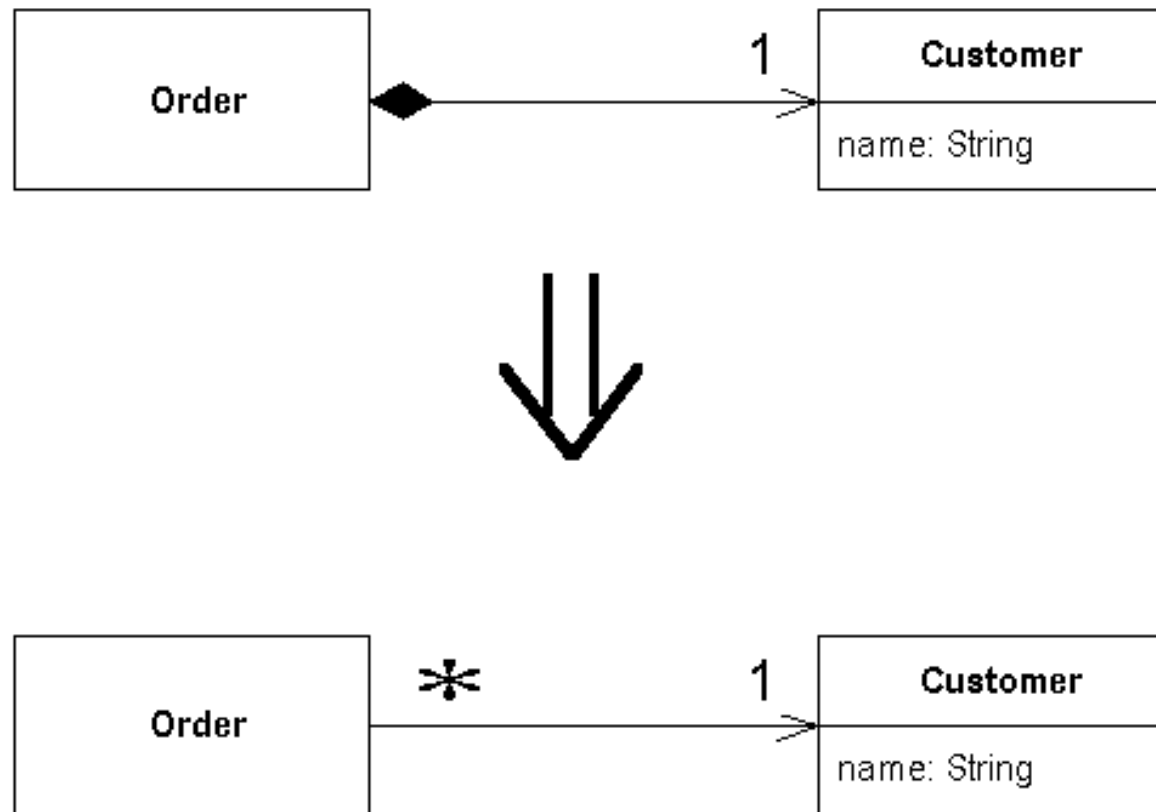


```
private int _low, _high;  
boolean includes (int arg) {  
    return arg >= getLow() && arg <= getHigh();  
}  
int getLow() {return _low;}  
int getHigh() {return _high;}
```

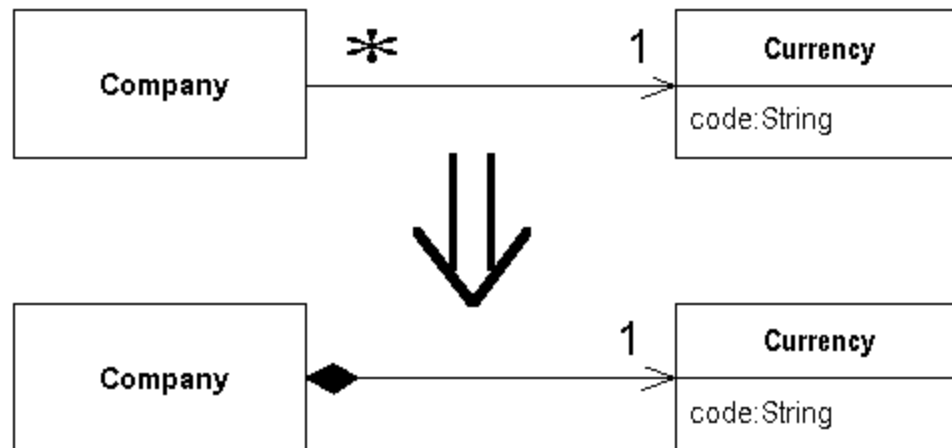
- Ersetze eigenes Attribut durch Objektverweis (Replace Data Value with Object)
- Die Klasse definiert ein Element, was zusätzliche Daten oder Verhalten benötigt
- Mache aus dem Element ein eigenständiges Objekt



- Ersetzte Wert durch Referenz (Change Value to Reference)
- Eine Klasse mit vielen gleichen Instanzen durch ein Objekt ersetzen



- Ersetzte Verweis durch Wert (Change Reference to Value)



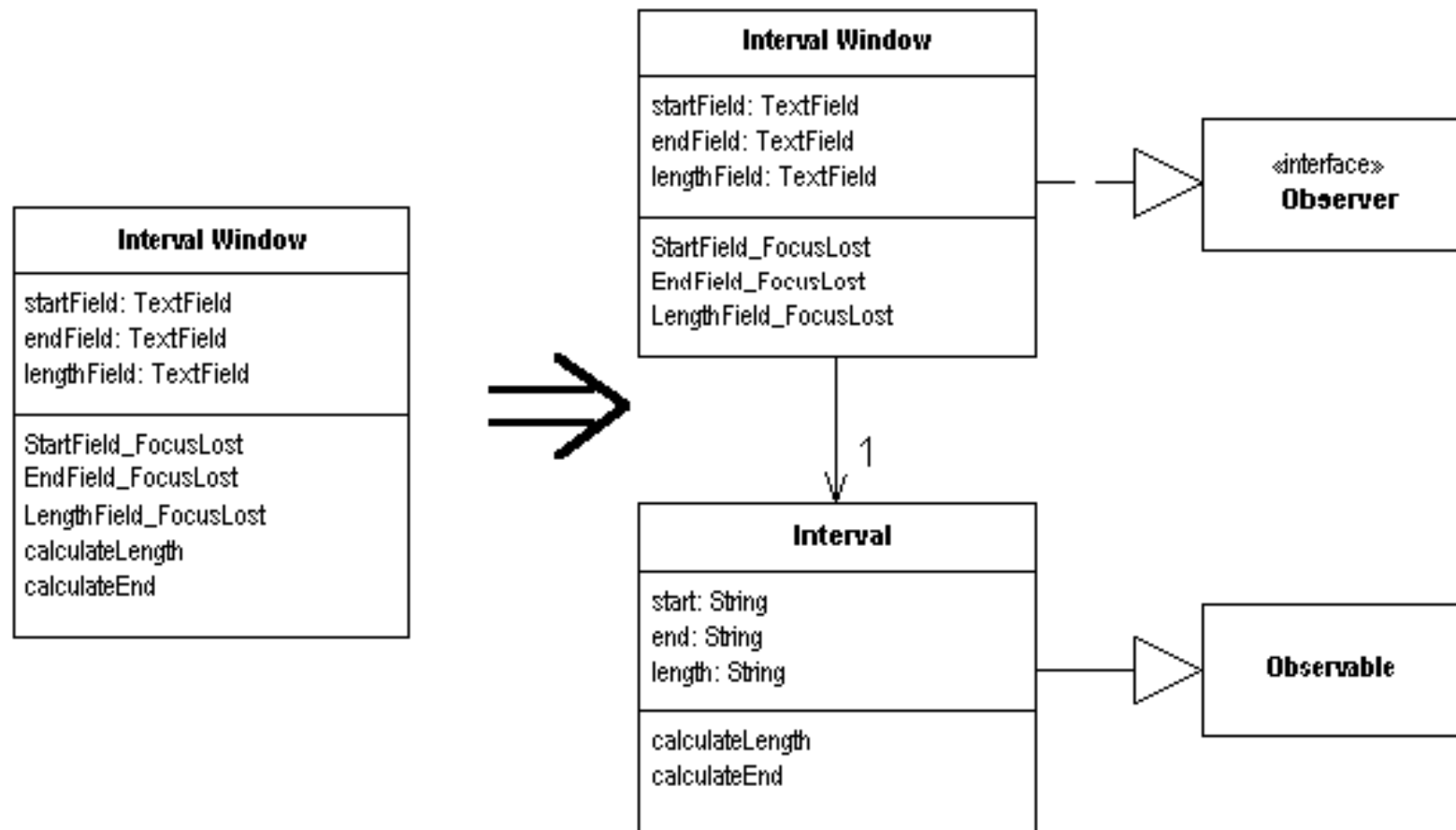
- Ein Feld nimmt Dinge unterschiedlicher Bedeutung an.
- Ersetze das Feld durch ein Objekt, welches Eigenschaften für die Elemente anbietet

```
String[] row = new String[3];  
row [0] = "Liverpool";  
row [1] = "15";
```

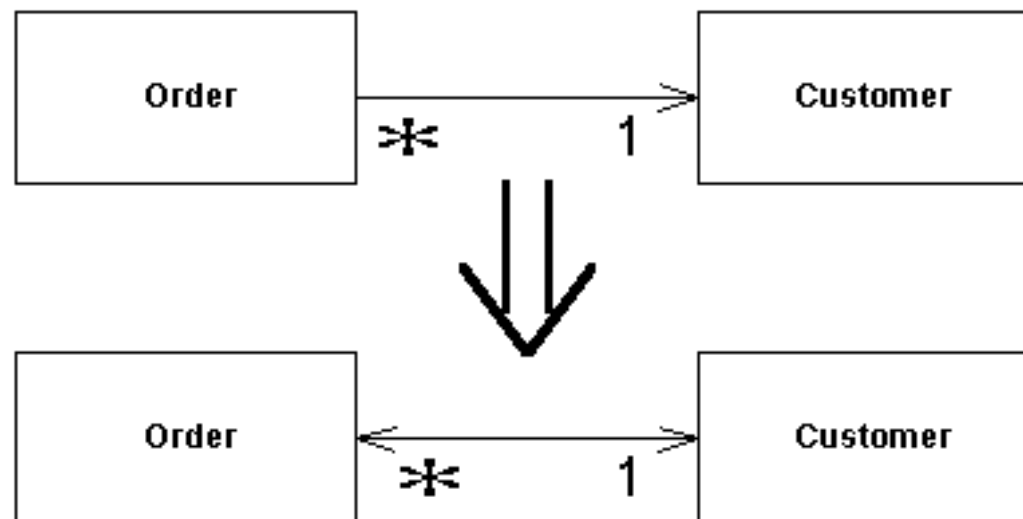


```
Performance row = new Performance();  
row.setName("Liverpool");  
row.setWins("15");
```

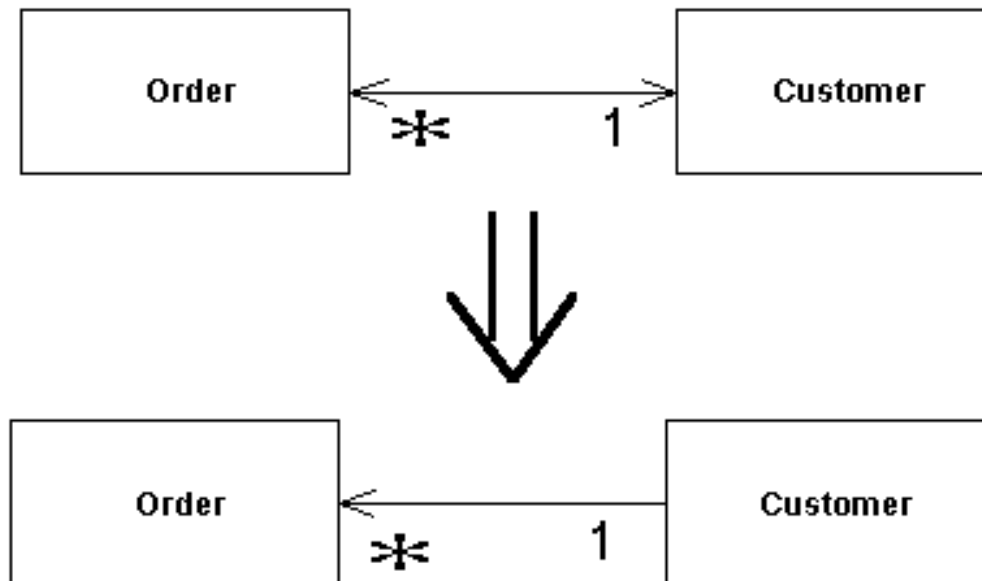
- Daten der Domäne stehen nur in der GUI zur Verfügung und Domänen Methoden brauchen Zugriff.



- Unidirektionale Beziehung in bidirektionale Beziehung ändern (Change Unidirectional Association to Bidirectional)
- Zwei Klassen müssen gegenseitig Eigenschaften nutzen, doch es gibt nur eine Navigation in eine Richtung.
- Setzte Verweise und ändere Zugriffsfunktionen für Aktualisierungen auf beiden Seiten.



- Bidirektionale Beziehung in unidirektionale Beziehung ändern (Change Bidirectional Association to Unidirectional)
- Es gibt eine beidseitige Beziehung, doch eine Klasse benötigt Eigenschaften der anderen Klasse nicht mehr.
- Entferne die unnötige Seite der Assoziation



- Ersetze magische Zahl durch symbolische Konstante (Replace Magic Number with Symbolic Constant)
- Ein Zahl-Literal steht für eine bestimmte Bedeutung
- Lege eine Konstante an. Nenne sie nach den Grund und ersetze die Zahl mit dieser Konstanten

```
double potentialEnergy(double mass, double height) {  
    return mass * height * 9.81;  
}
```



```
double potentialEnergy(double mass, double height) {  
    return mass * GRAVITATIONAL_CONSTANT * height;  
}  
static final double GRAVITATIONAL_CONSTANT = 9.81;
```

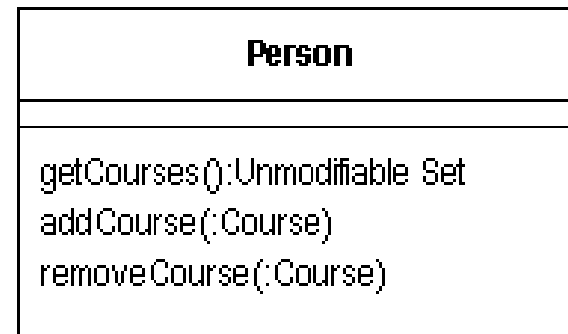
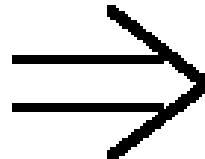
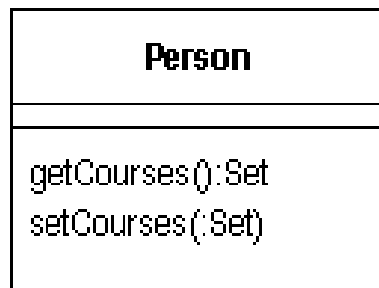
- Es gibt ein öffentliches Attribut
- Mache es privat und biete Zugriffsmethoden.

```
public String _name
```



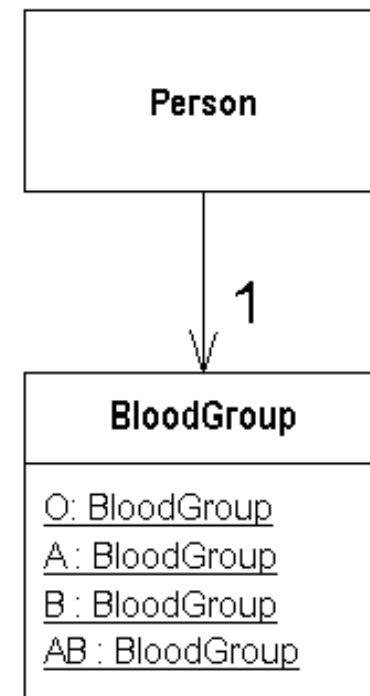
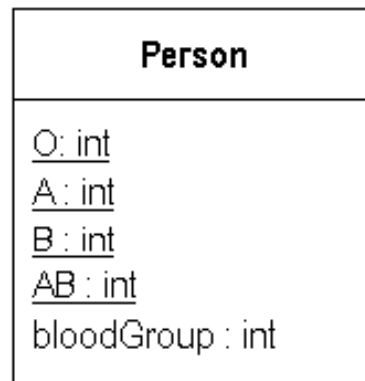
```
private String _name;  
public String getName() {  
    return _name;  
}  
public void setName(String arg) {  
    _name = arg;  
}
```

- Eine Methode liefert als Rückgabe eine Collection.
- Die Rückgabe wird eine nur-lese Sicht und Hinzufüge- sowie Lösche-Methoden werden angeboten.

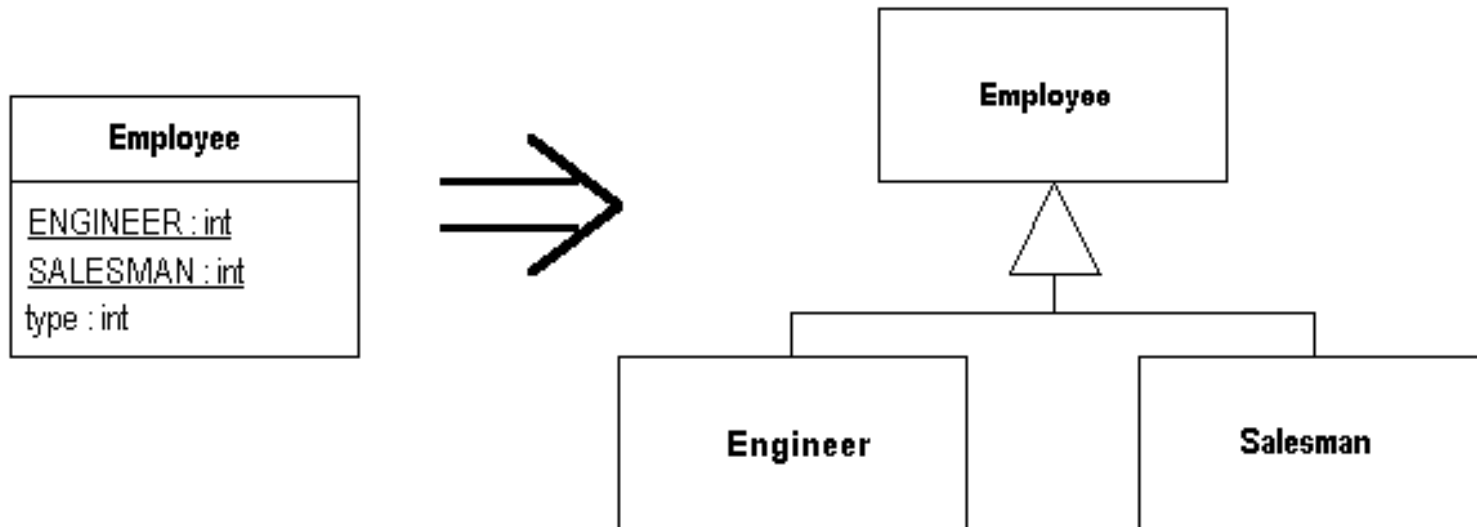


- Ersetze einen Datensatz durch eine Datenklasse (Replace Record with Data Class)
- Eine Verbindung zu einem Datensatz einer traditionellen Programmierumgebung ist nötig.
- Mache für den Datensatz ein einfaches Datenobjekt.

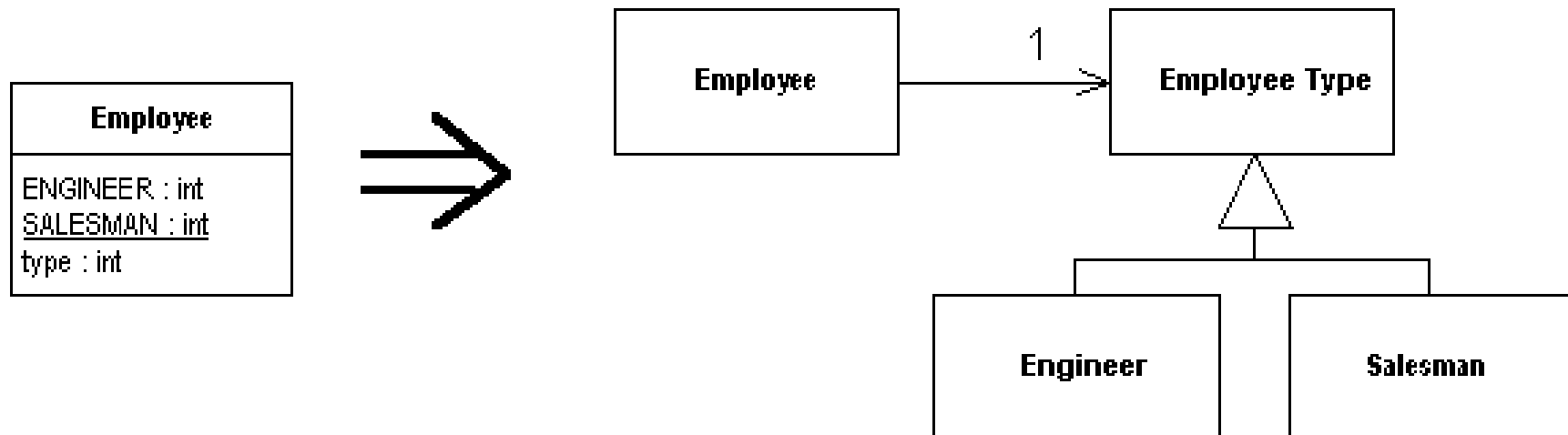
- Ersetze Typenschlüssel durch eine Klasse (Replace Type Code with Class)
- Eine Klasse hat einen numerischen Code, der das Verhalten nicht beeinflusst
- Ersetze die Zahl mit einer Klasse



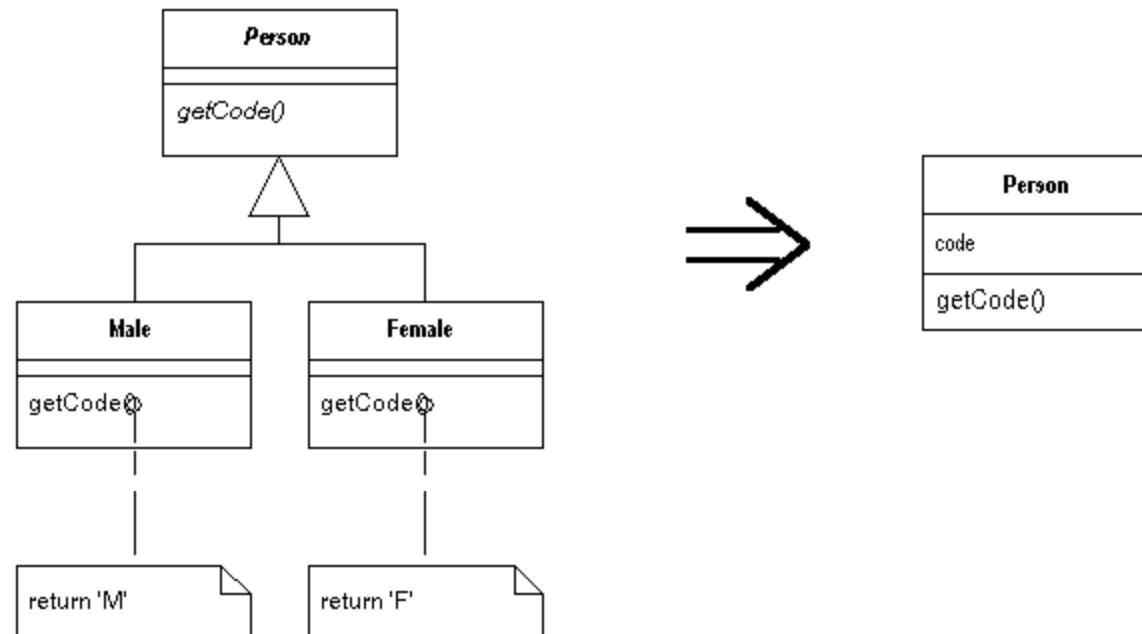
- Ersetze Typ-spezifischen Code durch Unterklassen (Replace Type Code with Subclasses)
- Ein unveränderlichen Programmcode bestimmt das Verhalten einer Klasse
- Ersetze den Typ-spezifischen Code durch eine Unterklasse.



- Ersetze Typ-spezifischen Code durch Zustand/Strategie-Muster (Replace Type Code with State/Strategy)
- Eine "Was-bin-ich"-Variable bestimmt das Verhalten, aber Unterklassen für das spezifische Verhalten können nicht gebildet werden
- Ersetze den Typ-spezifischen Code durch ein Zustands-Objekt.



- Ersetze Unterklassen durch Attribute (Replace Subclass with Fields)
- Unterklassen unterscheiden sich nur durch Methoden, die einen konstanten Wert liefern.
- Ändere die Methode mit einem Zugriff auf ein Attribut der Oberklasse und entferne die Unterklassen.



C) Bedingte Ausdrücke vereinfachen

- Es gibt einen komplizierten Ausdruck in einer Fallunterscheidung
- Fasse die Bedingung in einer Methode zusammen

```
if (date.before (SUMMER_START) || date.after(SUMMER_END))  
    charge = quantity * _winterRate + _winterServiceCharge;  
else charge = quantity * _summerRate;
```



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

- Bedingungen von Ausdrücken zusammenführen (Consolidate Conditional Expression)
- Es gibt eine Reihe von Tests, die zum gleichen Ergebnis führen.
- Fasse sie zu einer einzigen Bedingung zusammen und extrahiere das zu einer Methode

```
double disabilityAmount() {  
    if (_seniority < 2) return 0;  
    if (_monthsDisabled > 12) return 0;  
    if (_isPartTime) return 0;  
    // compute the disability amount
```



```
double disabilityAmount() {  
    if (isNotEligibleForDisability()) return 0;  
    // compute the disability amount
```

- Wiederholte Anweisungen aus Bedingungen zusammenführen (Consolidate Duplicate Conditional Fragments)
- Das gleiche Codefragment taucht in allen Zweigen einer Fallunterscheidung auf.
- Lege den gemeinsamen Teil nach außen.

```
if (isSpecialDeal()) {  
    total = price * 0.95;  
    send();  
} else {  
    total = price * 0.98;  
    send();  
}
```



```
if (isSpecialDeal())  
    total = price * 0.95;  
else  
    total = price * 0.98;  
send();
```

- Eine Variable steht für ein Kontroll-Flag, was den Ausstieg aus einer Schleife steuert.
- Nutze ein `break` oder `return` stattdessen.

- Geschachtelte Bedingungen durch Wächter ersetzen
(Replace Nested Conditional with Guard Clauses)

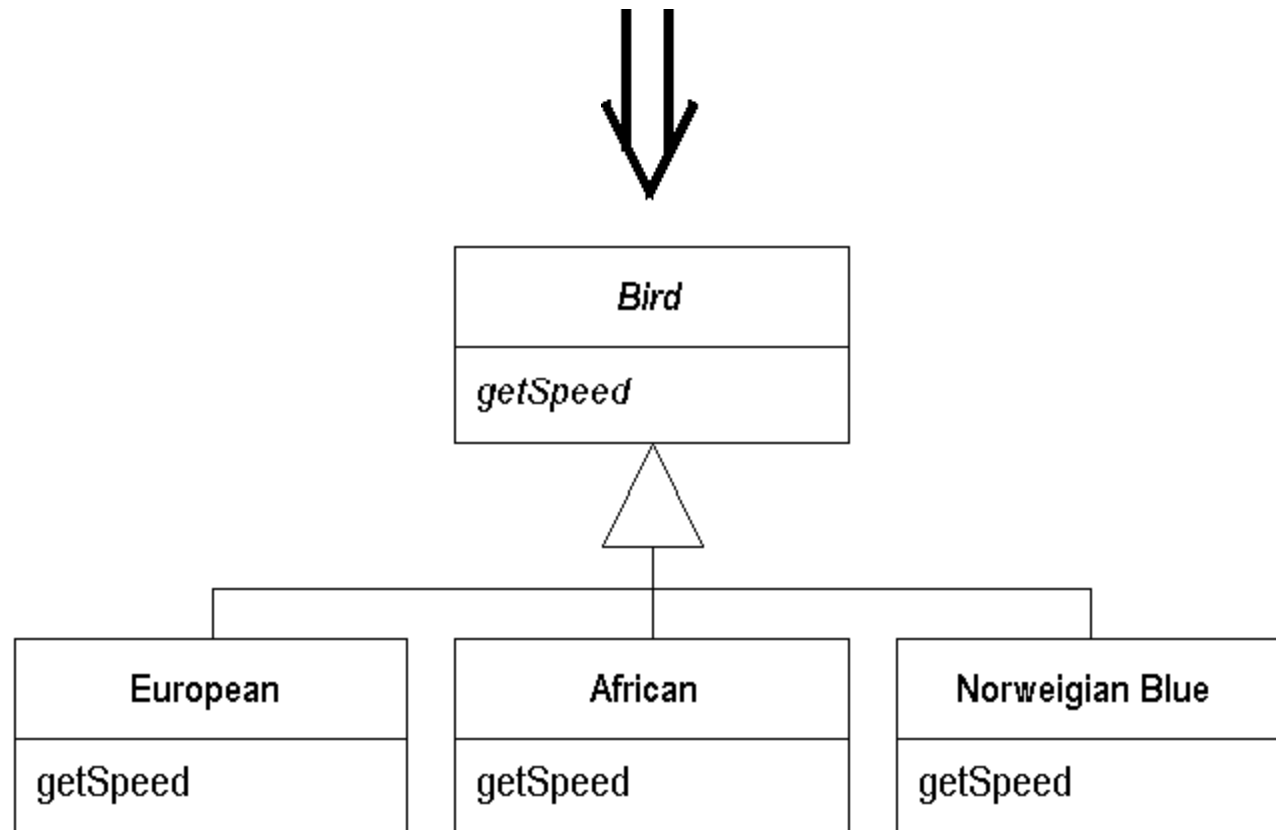
```
double getPayAmount() {  
    double result;  
    if (_isDead) result = deadAmount();  
    else {  
        if (_isSeparated) result = separatedAmount();  
        else {  
            if (_isRetired) result = retiredAmount();  
            else result = normalPayAmount();  
        }  
    }  
    return result;  
};
```



```
double getPayAmount() {  
    if (_isDead) return deadAmount();  
    if (_isSeparated) return separatedAmount();  
    if (_isRetired) return retiredAmount();  
    return normalPayAmount();  
};
```

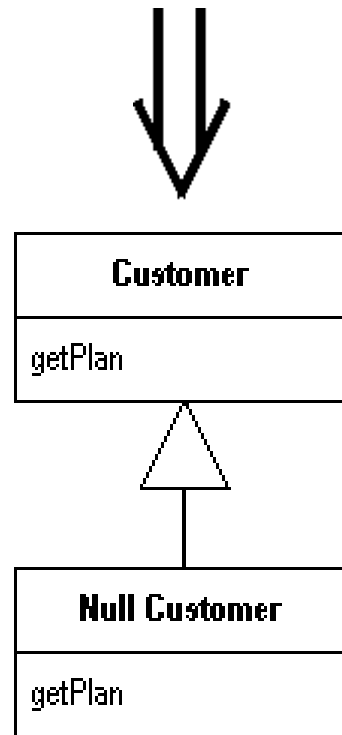
- Ersetze Fallunterscheidungen durch Polymorphie (Replace Conditional with Polymorphism)
- Eine Fallunterscheidung führt abhängig vom Typ eines Objektes unterschiedliches Verhalten aus.
- Lege jeden Zweig der Fallunterscheidung in eine Methode einer Unterklasse. Die Methode der Oberklasse wird abstrakt und die neuen Methoden überschreiben die abstrakte Methode.

```
double getSpeed() {  
    switch (_type) {  
        case EUROPEAN: return getBaseSpeed();  
        case AFRICAN: return getBaseSpeed() - getLoadFactor() * _numberOfCoconuts;  
        case NORWEIGIAN_BLUE: return (_isNailed) ? 0 : getBaseSpeed(_voltage);  
    } throw new RuntimeException ("Should be unreachable");  
}
```



- Es gibt wiederholte Abfragen auf null.
- Ersetze den null-Wert mit einem null-Objekt.

```
if (customer == null) plan = BillingPlan.basic();  
else plan = customer.getPlan();
```



- Ein Programmteil macht Annahmen über den Status des Programms
- Die Annahmen werden ausdrücklich über eine Zusicherung (Assertion) realisiert.

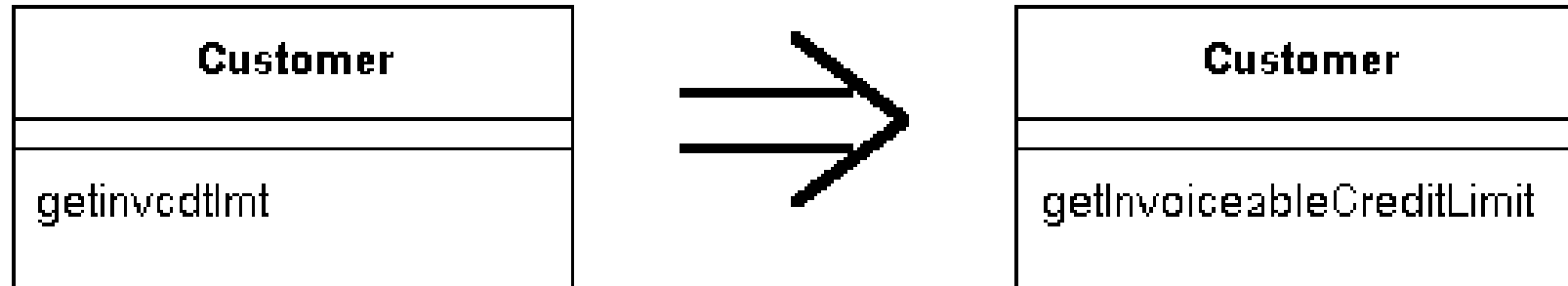
```
double getExpenseLimit() {  
    // should have either expense limit or a primary project  
    return (_expenseLimit != NULL_EXPENSE) ?  
        _expenseLimit: _primaryProject.getMemberExpenseLimit(); }
```



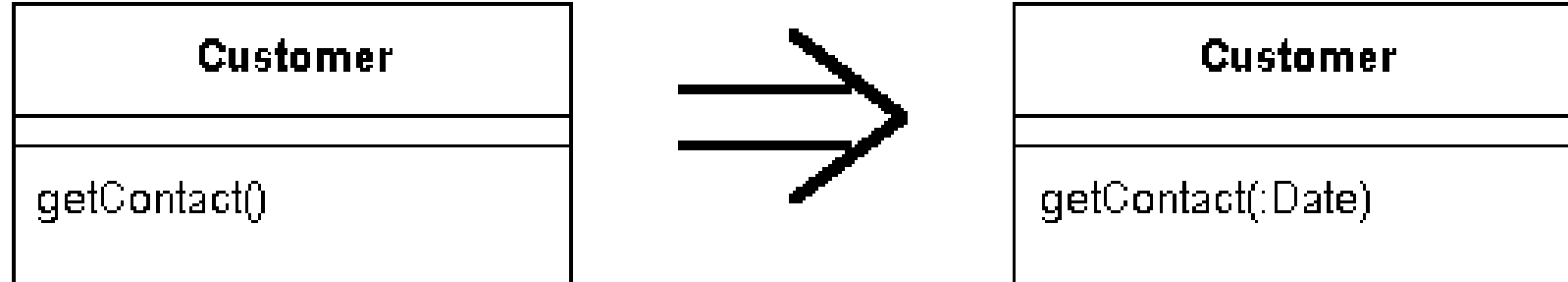
```
double getExpenseLimit() {  
    Assert.isTrue (_expenseLimit != NULL_EXPENSE || _primaryProject != null);  
    return (_expenseLimit != NULL_EXPENSE) ?  
        _expenseLimit: _primaryProject.getMemberExpenseLimit(); }
```

D) Methodenaufrufe vereinfachen

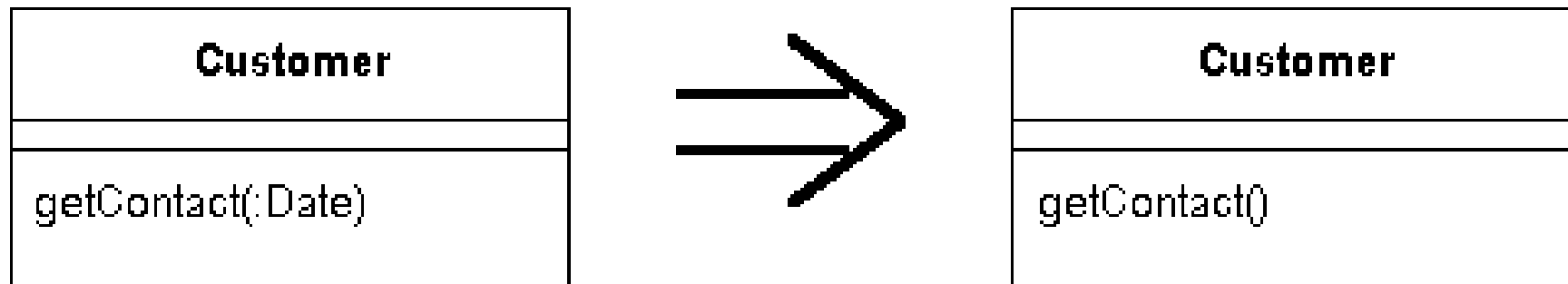
- Der Name einer Methode macht ihre Absicht nicht deutlich
- Ändere den Namen der Methode.



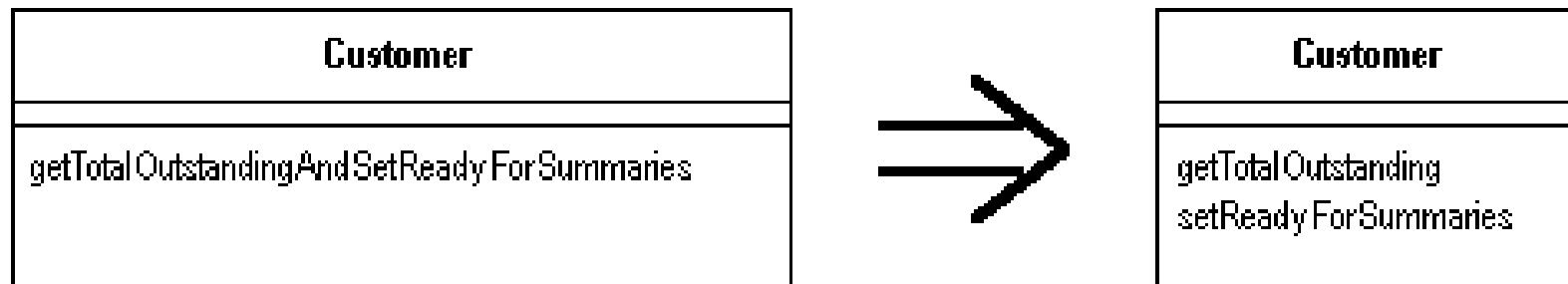
- Eine Methode benötigt mehr Informationen vom Aufrufer.
- Nimm einen Parameter für ein Objekt hinzu um die zusätzliche Informationen mitzugeben



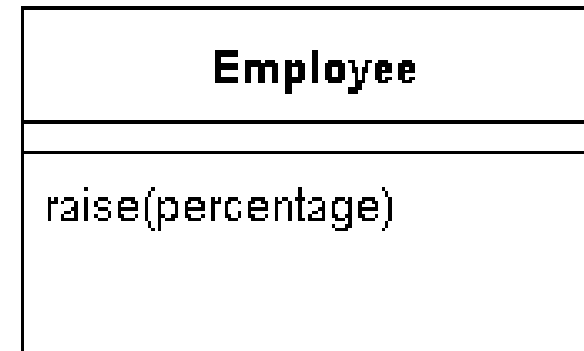
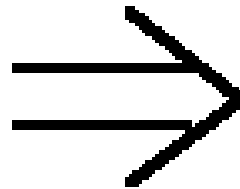
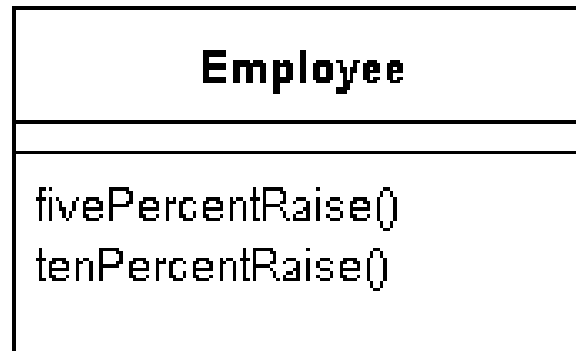
- Ein Parameter wird vom Methodenrumpf nicht mehr benötigt
- Entferne es.



- Trenne Anfragen von Veränderungen (Separate Query from Modifier)
- Eine Methode liefert einen Wert zurück und ändert gleichzeitig den Zustand des Objekts
- Schreibe zwei Methoden. Eine für die Anfrage und eine zur Modifikation



- Verschiedene Methoden führen ähnliche Dinge mit verschiedenen Werten im Rumpf aus.
- Schreibe eine Methode, die einen Parameter für die verschiedenen Werte nutzt.



- Ersetze Parameter durch explizite Methoden (Replace Parameter with Explicit Methods)
- Eine Methode führt abhängig von bestimmten übergebenen Argumenten Programmcode aus
- Lege für jedes der möglichen Argumente eine Methode an.

```
void setValue (String name, int value) {  
    if (name.equals("height")) {  
        _height = value; return;  
    }  
    if (name.equals("width")) {  
        _width = value; return;  
    }  
    Assert.shouldNeverReachHere();  
}
```



```
void setHeight(int arg) { _height = arg; }  
void setWidth (int arg) { _width = arg; }
```

- Vorhandenes Objekt übergeben (Preserve Whole Object)
- Von einem Objekt ausgelesene Werte sind Argumente einer anderen Methode.
- Übergebe gleich das ganze Objekt.

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



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

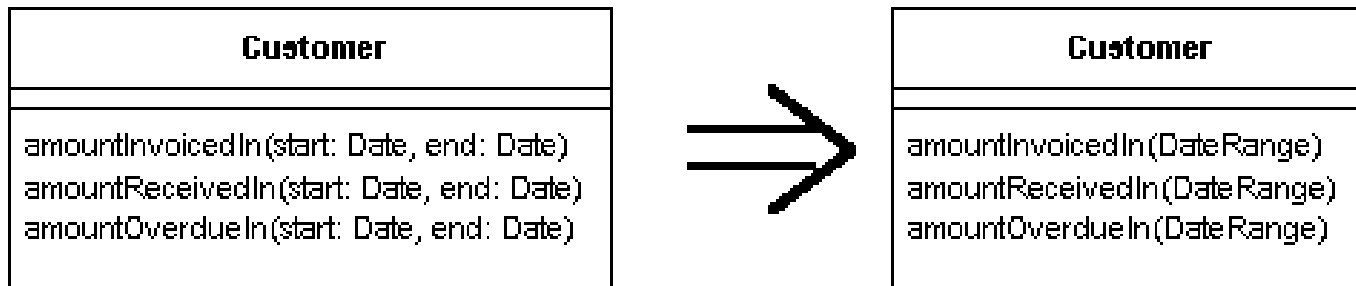
- Ersetze Parameter durch Methode (Replace Parameter with Method)
- Ein Objekt ruft eine Methode und übergibt das Ergebnis als Argument an eine andere Methode. Der Empfänger könnte ebenso diese Methode aufrufen
- Entferne den Parameter und lasse dem Empfänger die Methode aufrufen

```
int basePrice = _quantity * _itemPrice;  
discountLevel = getDiscountLevel();  
double finalPrice = discountedPrice (basePrice, discountLevel);
```

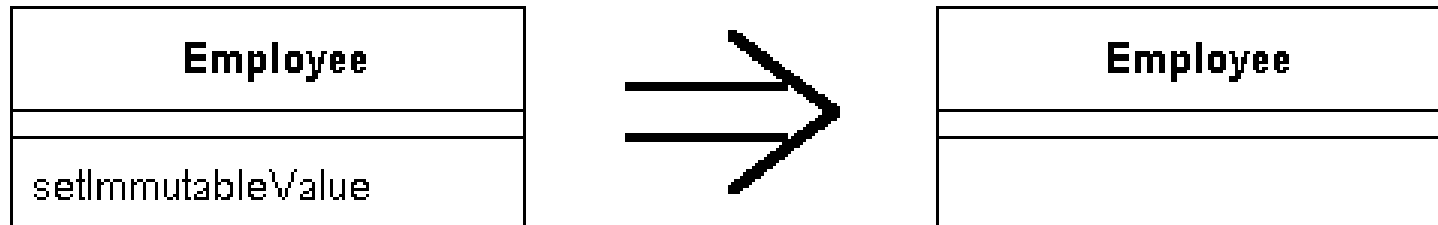


```
int basePrice = _quantity * _itemPrice;  
double finalPrice = discountedPrice (basePrice);
```

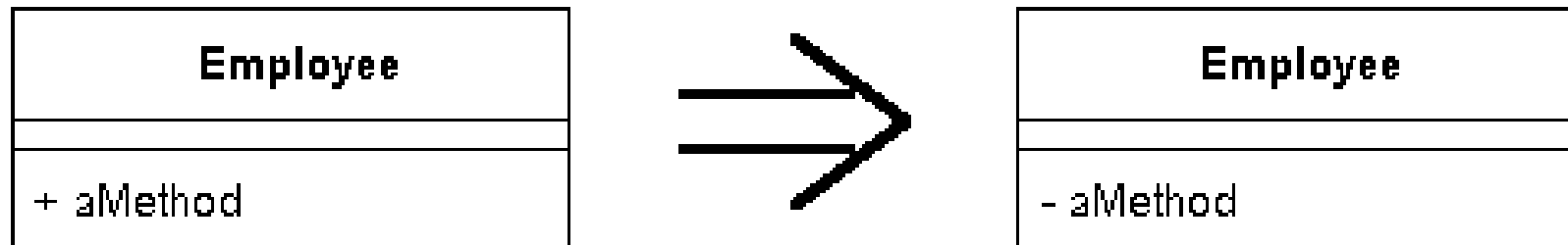
- Parameter-Objekt einführen (Introduce Parameter Object)
- Einige Parameter gehören zusammen
- Ersetzte diese mit einem Objekt



- Ein Feld soll bei der Erzeugung einen Wert zugewiesen bekommen und diesen während der Laufzeit nicht mehr ändern
- Entferne alle „setter“ Methoden für das Feld



- Eine Methode wird von keiner anderen Klasse genutzt
- Mache die Methode private



- Ersetze Konstruktor mit Fabrik-Methode (Replace Constructor with Factory Method)
- Wenn ein Objekt erzeugt wird, soll noch mehr gemacht werden
- Ersetze den Konstruktor durch eine Fabrik-Methode.

```
Employee (int type) {  
    _type = type;  
}
```



```
static Employee create(int type) {  
    return new Employee(type);  
}
```

- Eine Methode liefert ein Objekt, welches vom Aufrufer mit einer expliziten Typanpassung angepasst werden muss.
- Die explizite Typanpassung wird in die Methode gesetzt.

```
Object lastReading() {  
    return readings.lastElement();  
}
```



```
Reading lastReading() {  
    return (Reading) readings.lastElement();  
}
```

- Ersetze Fehlercodes durch Ausnahmen (Replace Error Code with Exception)
- Eine Methode liefert im Fehlerfall einen speziellen Rückgabewert
- Löse im Fehlerfall eine Exception aus

```
int withdraw(int amount) {  
    if (amount > _balance) return -1;  
    else {  
        _balance -= amount; return 0;  
    }  
}
```



```
void withdraw(int amount) throws BalanceException {  
    if (amount > _balance)  
        throw new BalanceException();  
    _balance -= amount;  
}
```

- Vermeide Ausnahmen durch Tests (Replace Exception with Test)
- Auf Grund eines Fehlers vom aufrufenden Client wird eine Exception ausgelöst.
- Die aktuellen Parameter werden vorher getestet

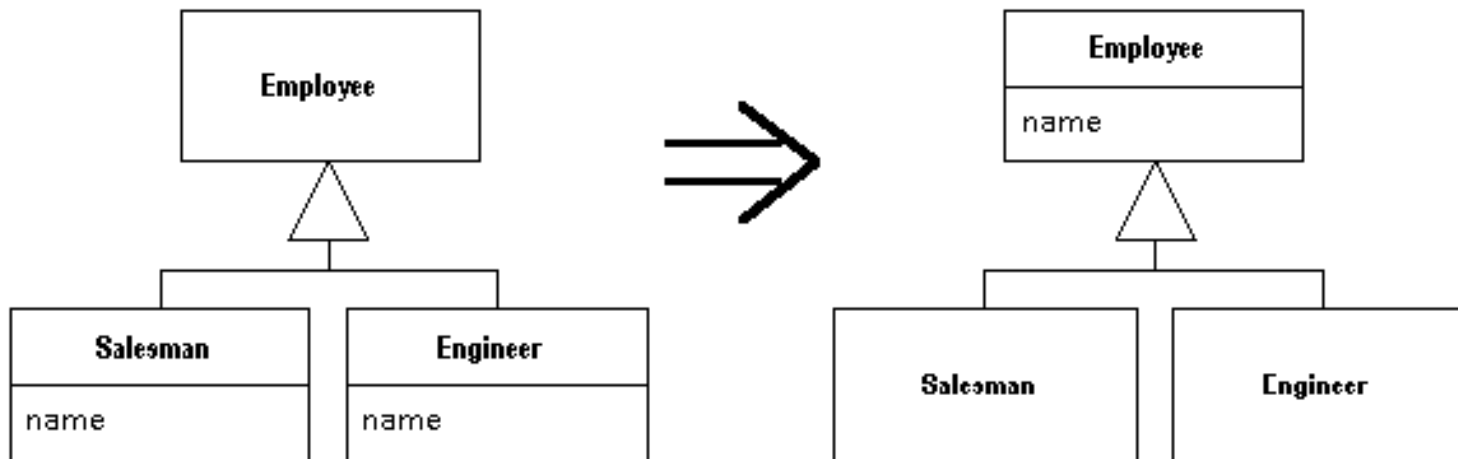
```
double getValueForPeriod (int periodNumber) {  
    try {  
        return _values[periodNumber];  
    } catch (ArrayIndexOutOfBoundsException e) {  
        return 0;  
    }  
}
```



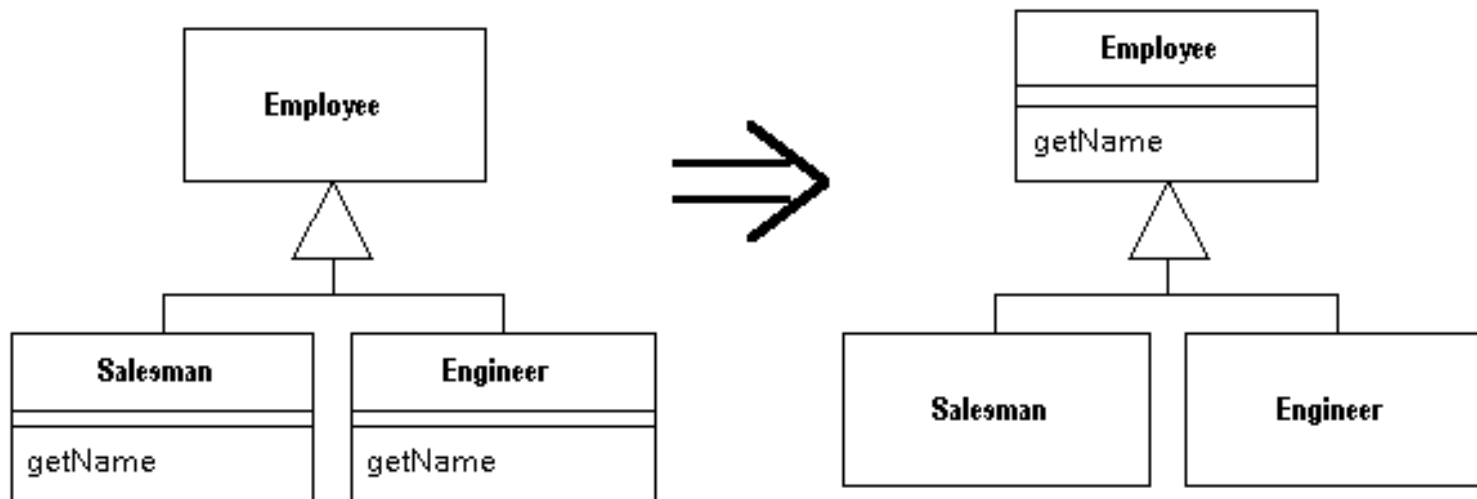
```
double getValueForPeriod (int periodNumber) {  
    if (periodNumber >= _values.length)  
        return 0;  
    return _values[periodNumber];  
}
```

E) Umgang mit Generalisierung

- Zwei Unterklassen besitzen das gleiche Attribut
- Verschiebe das Attribut in die Oberklasse.



- Es gibt Methoden mit identischen Ergebnissen in den Unterklassen
- Verschiebe sie in die Oberklassen.



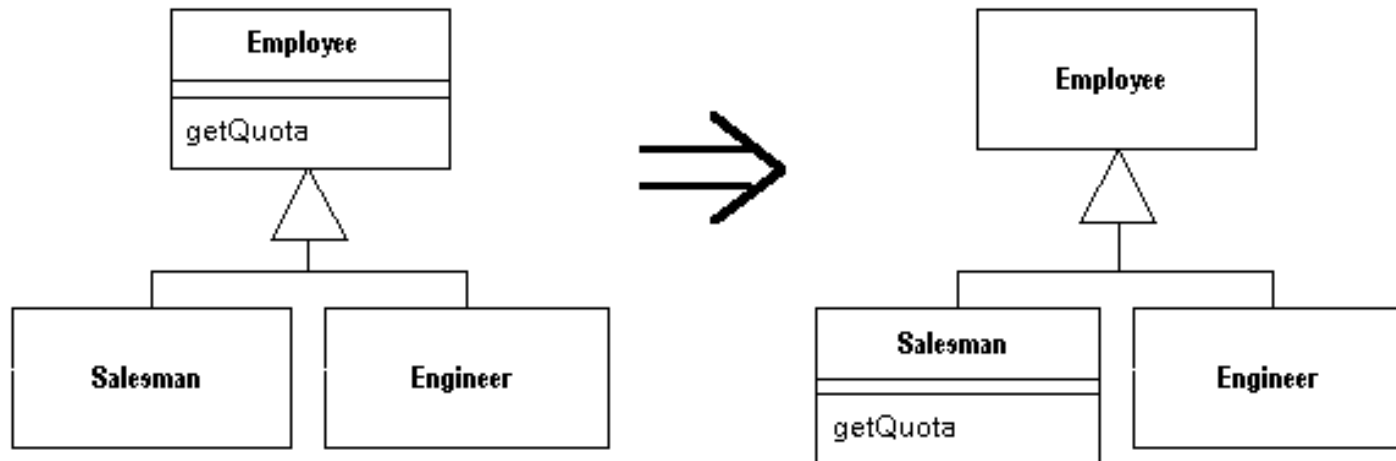
- Konstruktor-Körper hochziehen (Pull Up Constructor Body)
- Methodenrümpfe in den Konstruktoren der Kindklassen sind fast identisch
- Erzeuge einen Vaterklassen Konstruktor und rufe diesen aus den Kindklassenkonstruktoren auf

```
class Manager extends Employee...  
public Manager (String name, String id, int grade) {  
    _name = name;  
    _id = id;  
    _grade = grade;  
}
```

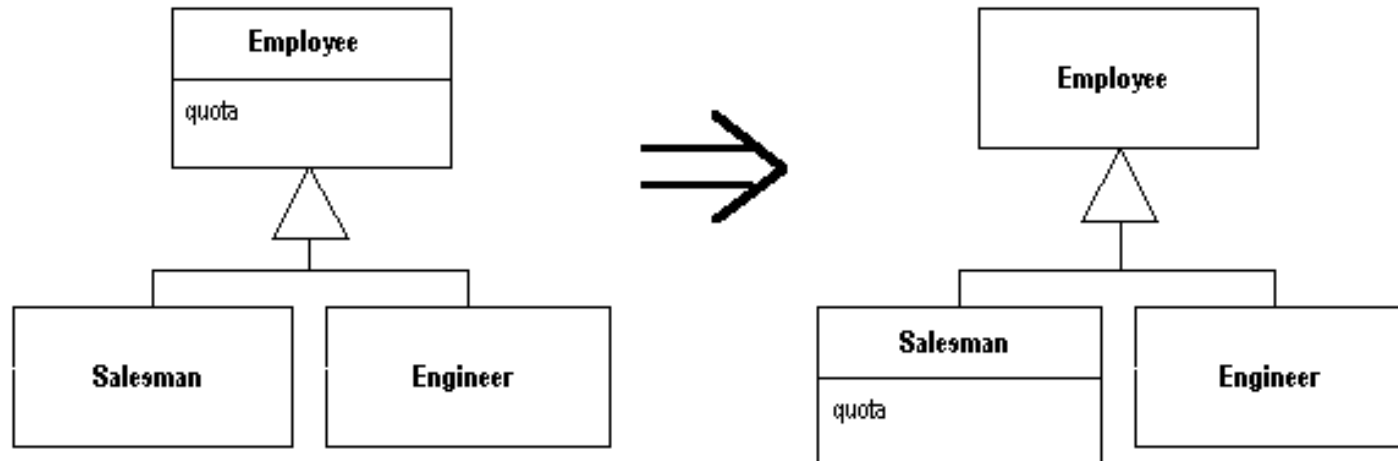


```
public Manager (String name, String id, int grade) {  
    super (name, id);  
    _grade = grade;  
}
```

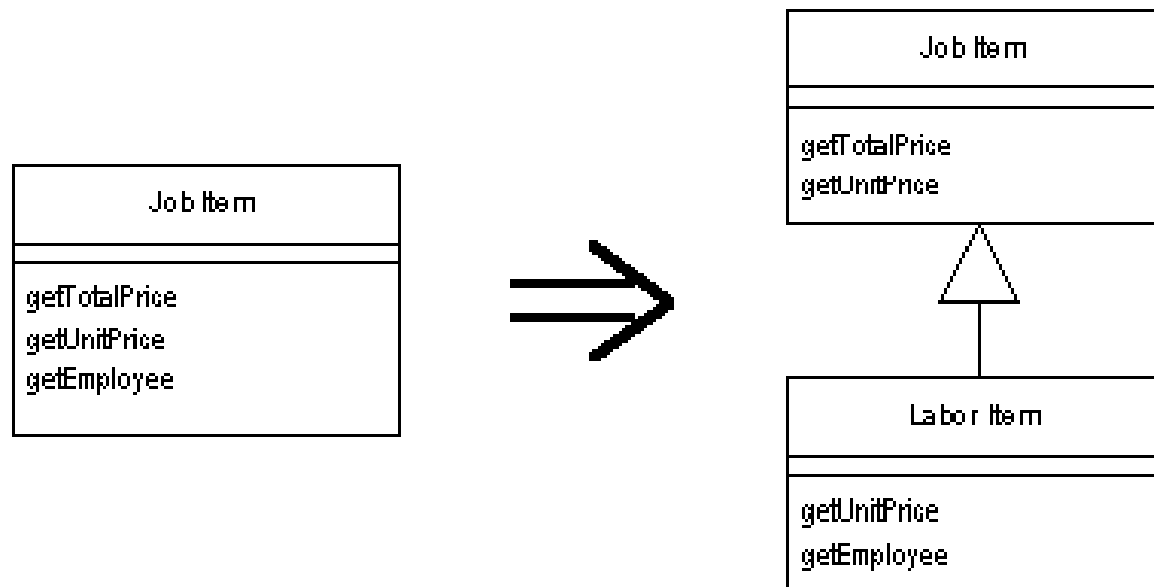
- Verhalten einer Oberklasse ist nur relevant für einige Unterklassen.
- Verschiebe die Methode in diese Unterklassen



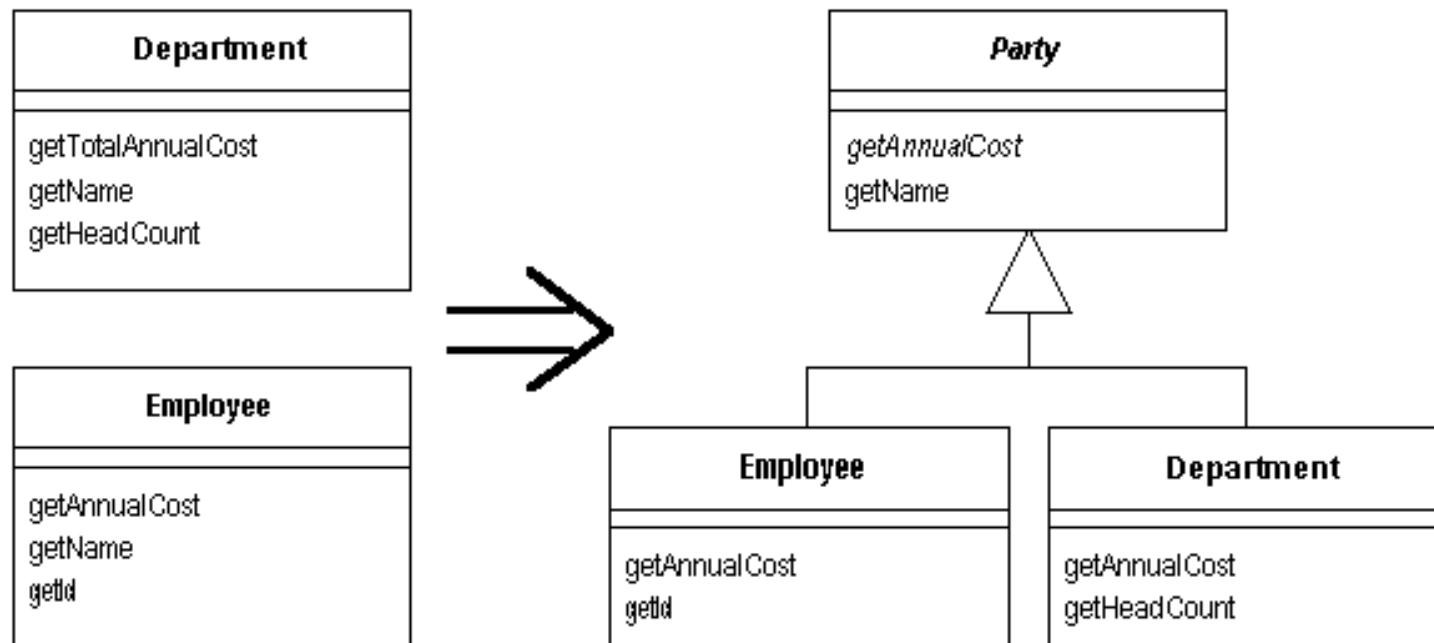
- Ein Attribut wird noch von einigen Unterklassen verwendet
- Verschiebe das Attribut in die Unterklassen.



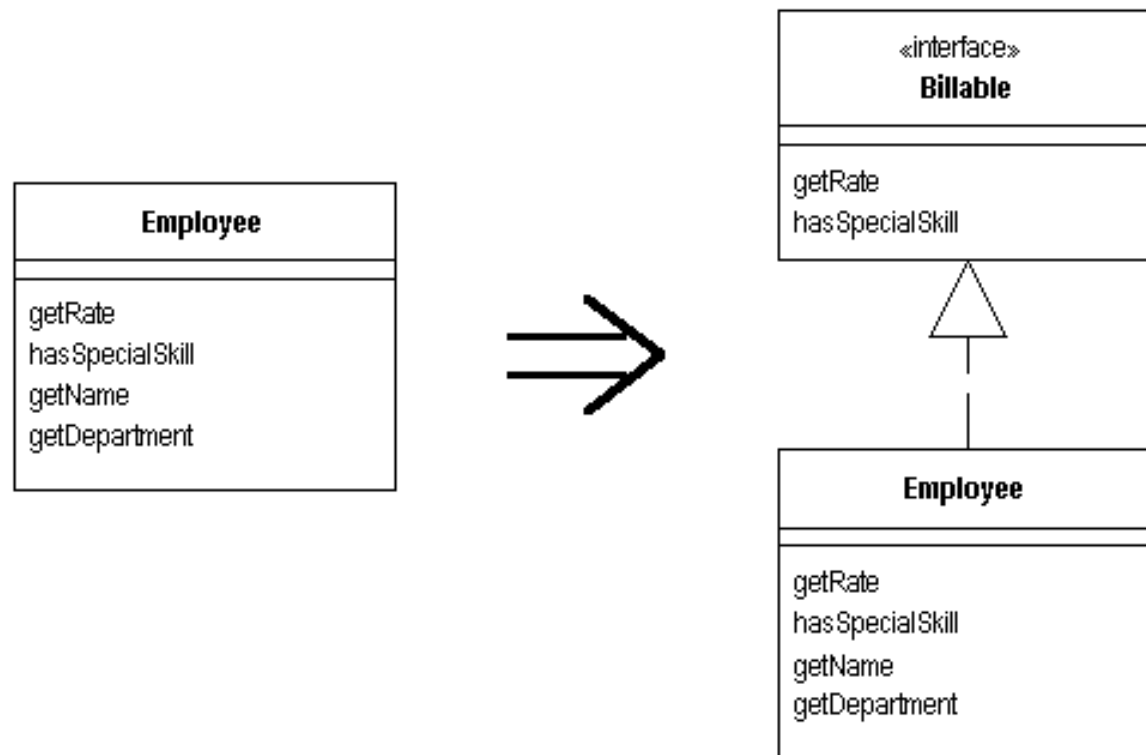
- Eine Klasse hat Eigenschaften, die nur von einige Exemplare verwendet wird.
- Erzeuge für die Untermenge an Eigenschaften eine Unterklasse



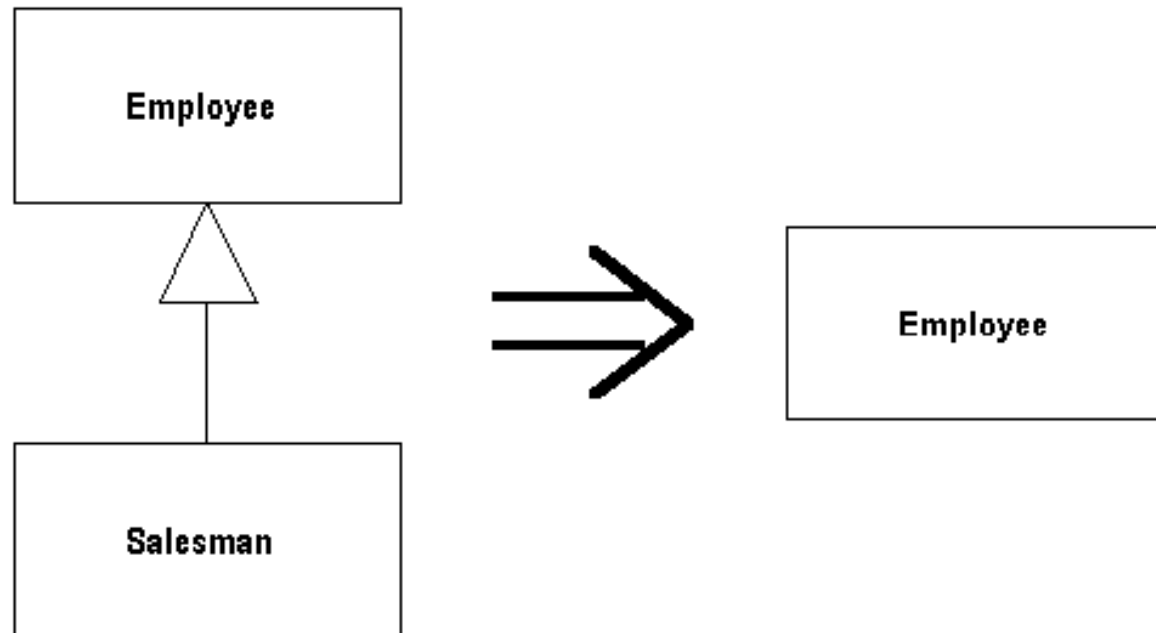
- Es gibt zwei Klassen mit gemeinsamen Eigenschaften
- Erzeuge eine neue Oberklasse und lege die gemeinsamen Eigenschaften in diese neue Klasse.



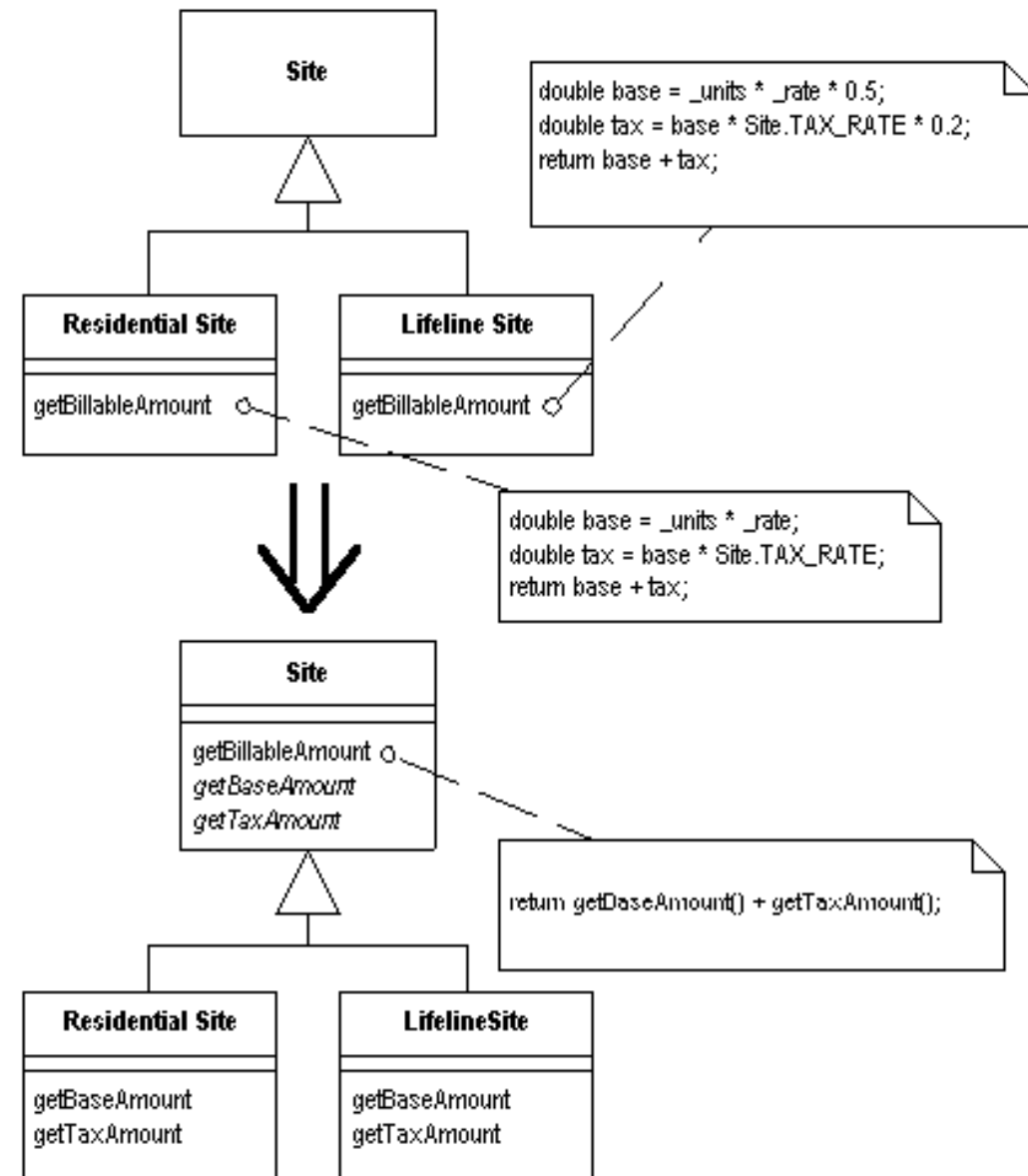
- Unterschiedliche Klienten nutzen die gleiche Untermenge einer Schnittstelle der Klasse. Zwei Klassen haben zum Teil gleiche Schnittstellen.
- Mache aus der Teilmenge eine eigene Schnittstelle.



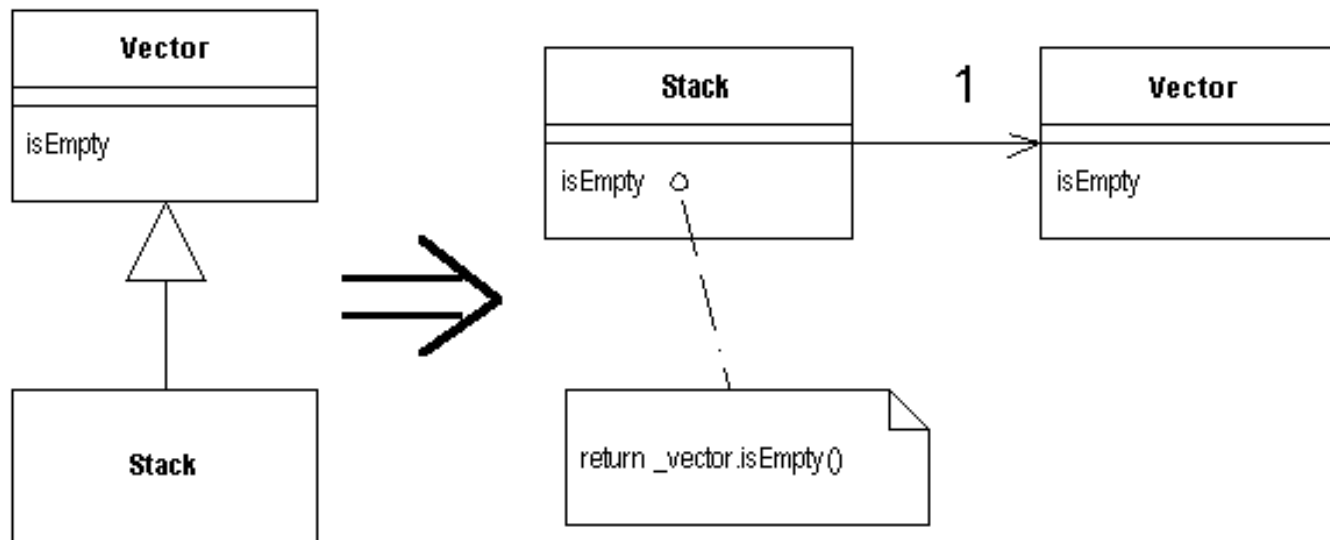
- Eine Unterklasse und Oberklasse sind nicht sehr verschieden
- Bringe sie zusammen.



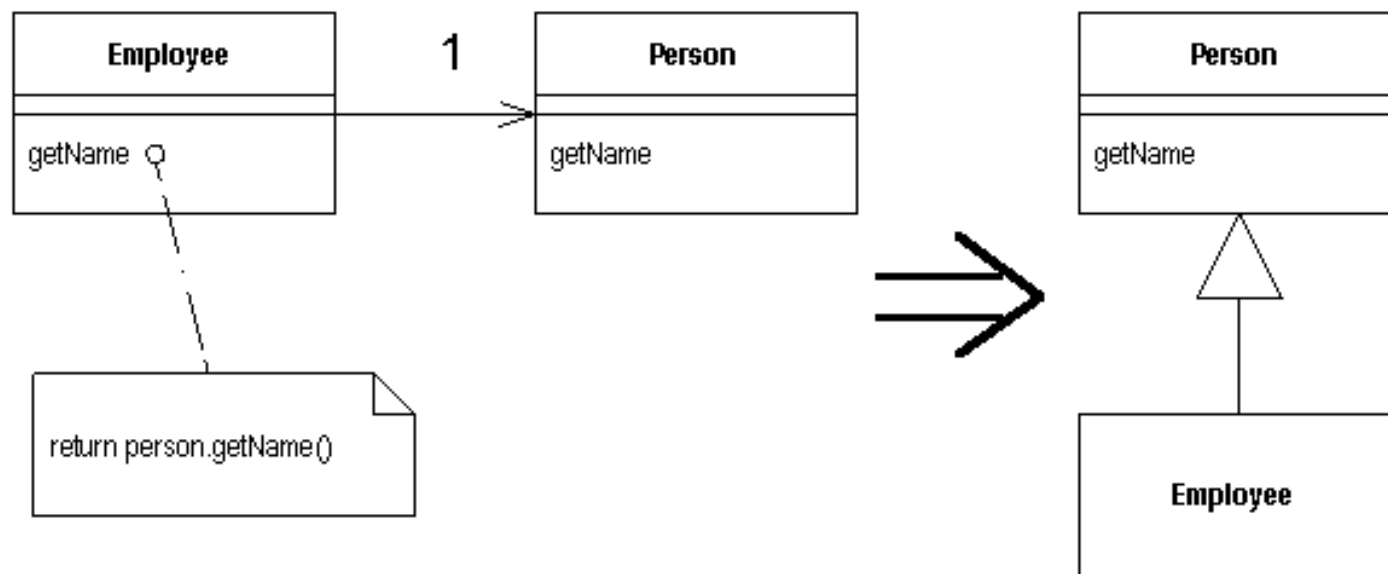
- Zwei Methoden in Unterklassen führen ähnliche Schritte in gleicher Reihenfolge aus, doch die Schritte sind im Detail anders.
- Fasse die jeweiligen Schritte zu Methoden mit gleicher Signatur zusammen.

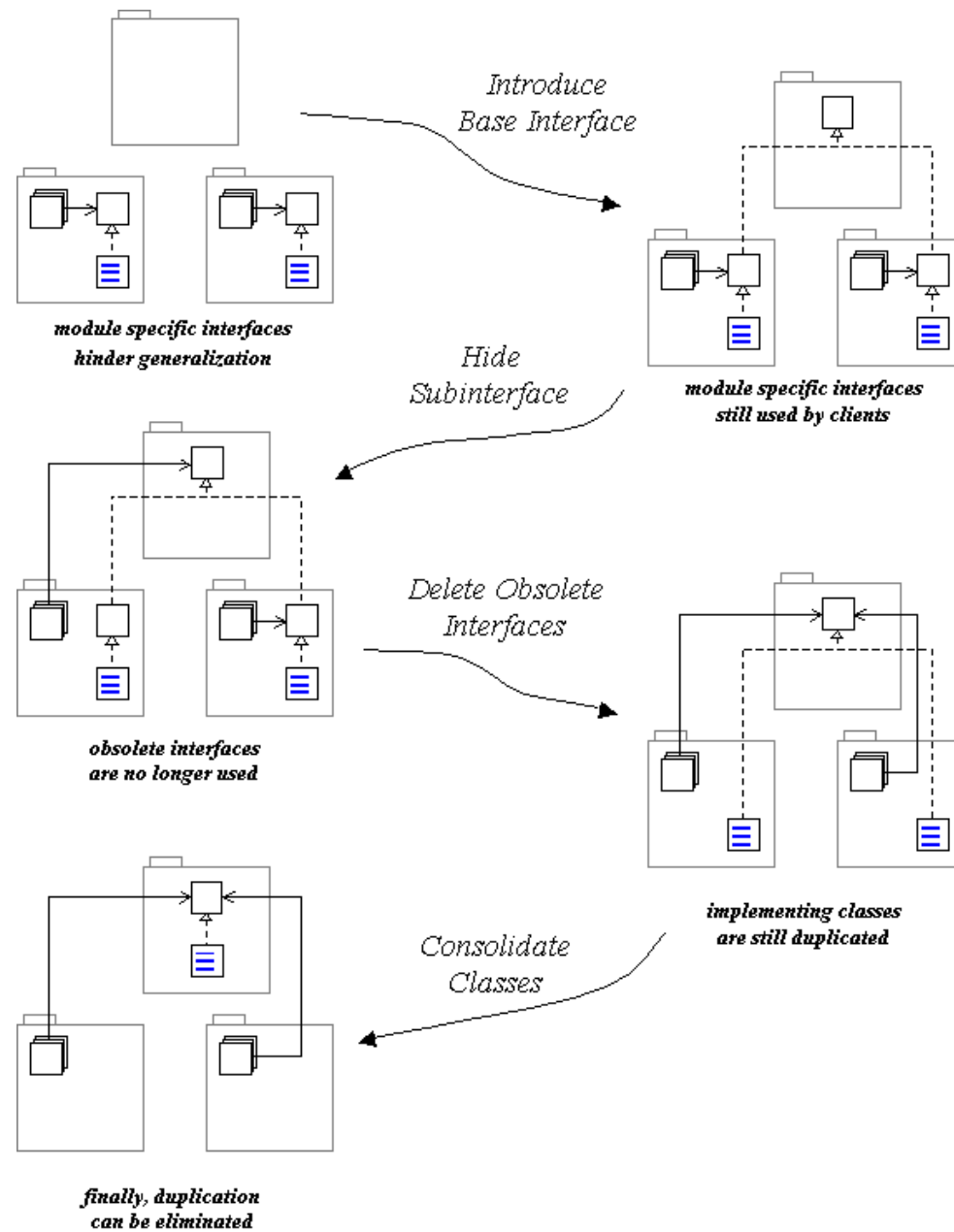


- Ersetzte Vererbung durch Delegation (Replace Inheritance with Delegation)
- Eine Unterklasse nutzt nur Teile der Schnittstelle der Oberklasse oder will keine Eigenschaften erben
- Erzeuge ein Attribut für die Oberklasse und passe die Methoden an, do dass sie auf die Methoden der Oberklasse delegieren. Entferne die Unterklasse.



- Ersetzte Delegation durch Vererbung (Replace Delegation with Inheritance)
- Methoden einer Klasse delegieren fast immer an eine andere Klasse.
- Der Delegate wird zur Oberklasse





- Smalltalk
 - Smalltalk Refactoring Browser
 - The original refactoring tool, and still one of the most full-featured. To all those who think that PhD projects are doomed to irrelevance consider that these efforts led to a whole new capability in programming tools.
- Java
 - IntelliJ Idea
 - This is a fully fledged IDE whose abilities go far beyond refactoring. I think they have succeeded in really moving forward the state of the art for IDEs. Suffice to say that when I use IntelliJ I almost don't miss Smalltalk any more :-)
 - Eclipse
 - An open source platform for IDE development - in many ways Eclipse is the Emacs for the 21st century. It ships with a built in Java IDE that includes a strong refactoring capability. The IBM commercial WSAD tool is built on Eclipse.
 - JFactor
 - A plug-in tool - works with JBuilder and Visual Age. Instantiations is a well respected outfit with a long history in Smalltalk and Java VM and compiler technology.
 - XRefactory
 - An emacs plug-in that offers code completion and other facilities as well as refactoring.
 - Together-J
 - Mostly known as a CASE-like tool with UML capabilities, but also does refactoring and many other things. Recently bought by Borland (or whatever they call themselves these days :-)
 - JBuilder
 - Borland's primary tool offers some refactoring support, but isn't over the rubicon yet. Various plug in tools offer deeper support.
 - RefactorIt
 - "A plug in for NetBeans, Sun Java Studio (fka Forte), Eclipse, JDeveloper and JBuilder and also usable as a stand-alone tool. Besides refactorings includes smart code searches, metrics and audits (i.e. smell detectors) with corrective actions."
 - JRefactory
 - A plug-in for JBuilder, NetBeans, and Elixir IDEs. Also does UML diagrams.
 - Transmogrify
 - JafaRefactor
 - A plug in for jEdit
 - CodeGuide
 - "CodeGuide offers advanced refactoring capabilities. It can rename methods, fields, variables, labels, classes or packages and automatically update all references throughout the project. It can move classes and packages and update all references. CodeGuide will check for potential problems prior to refactoring to make the refactorings safe. There are a couple of intelligent coding tools that will help you perform common tasks such as Javadoc stub insertion, setter & getter creation and import organization."

- .NET
 - ReSharper
 - As well as refactoring it also brings many other features from the popular Java IntelliJ IDE to the the C# world, so much so that many of my buddies already can't manage without it.
 - C# Refactory
 - Fully integrated with Visual Studio - adds a very comforting "Refactoring" item to the edit menu.
 - Refactor! Pro
 - A general .NET refactoring tool that supports both C# and Visual Basic. A VB only version (below) is available as a free download from Microsoft. Works with VS 2005 only.
 - C# Refactoring Tool
 - I haven't tried this one, but it boasts an interesting set of refactorings
- C/C++
 - SlickEdit
 - SlickEdit is a long time programmer editor that many people like a lot. It's latest version adds pretty-reasonable sounding refactoring support.
 - Ref++
 - Ref++ is a visual studio add-in that adds refactoring support for C++.
 - Xrefactory
 - A plug-in for emacs. Supports renaming, parameter manipulations and extract method.
- Visual Basic
 - Refactor! for Visual Basic
 - "Refactor! for Visual Basic 2005 Beta 2 is a free plug-in from Developer Express Inc., in partnership with Microsoft, that enables Visual Basic developers to simplify and re-structure source code inside of Visual Studio 2005, making it easier to read and less costly to maintain. Refactor! supports more than 15 individual refactoring features, including operations like Reorder Parameters, Extract Method, Encapsulate Field and Create Overload."
 - Refactor! Pro
 - This is the 'Pro' version of the free download above. It has some extra features and also supports C#.
 - Aivosto Project Analyzer
 - "a Visual Basic code optimization utility. While not strictly a refactoring tool, it does include a lot of related functionality. For example, it can automatically do Encapsulate Field, Remove Parameter, and Hide Method."
- Python
 - Bicycle Repair Man
 - Given the context, this at least gets a prize for the best name!
- Haskell
 - HaRe
 - An ongoing project to develop a refactoring tool for Haskell at the Universit of Kent. It's part of a general research effort to bring refactoring to functional programming languages.
 - Self
 - Guru
 - "Given a collection of Self objects, Guru produces an equivalent set of objects in which there is no duplication of methods or certain types of expressions. To achieve this, Guru creates a replacement inheritance hierarchy and methods for factoring out expressions."
- Delphi
 - Model Maker
 - This apparantly does a few refactorings, not doing extract method, but does seem to do some beyond simple renames, at least as far as I can tell from the web site. There's also a code explorer that puts this support directly into the IDE.
- General
 - X-develop
 - "X-develop is our new multi-language cross-platform IDE for professional coders. It offers code-centric semantic-driven productivity features for C#, Java, JSP, J# and VB.NET and supports .NET, Mono and the Java platform on Windows, Linux and Mac OS X. X-develop includes many common refactorings: rename variable/member/class, change method signature, extract method, inline method, move classes, introduce variable, inline variable and more."

