Refactoring

Vortrag im Rahmen des Softwareprojekts: Übersetzerbau

Referenten: Vivienne Severa
Alpin Mete Sahin
Florian Mercks

Datum: 20.06.2013

Überblick

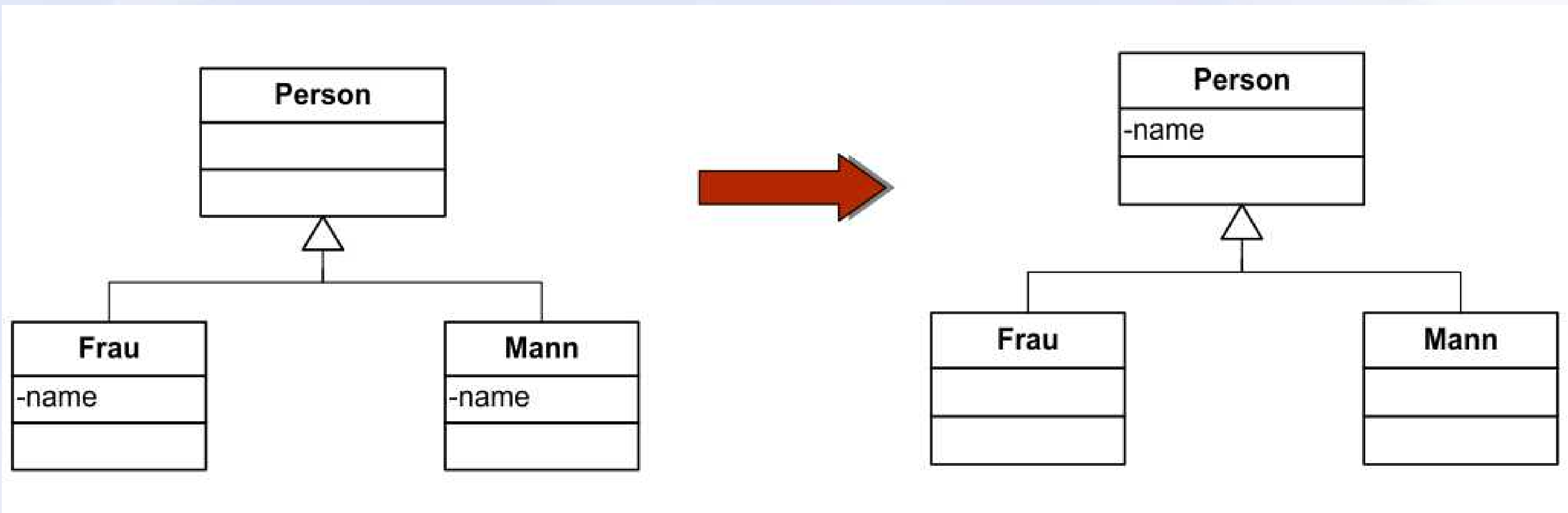
- Einführung
- Refactoring im Einsatz
- Werkzeugunterstützung
- Fazit

Was ist Refactoring?

- Refactoring als einzelne **Operation**:
  - Quellcodetransformation, sichtbares Verhalten des Software-Systems wird nicht verändert
- Refactoring als **Prozess**:
  - interne Code-Struktur-Veränderung eines Software-Systems durch den Einsatz von Refactoring-Operationen

Beispiel 1

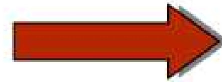
- Pull Up Field:



Beispiel 2

Extract Method:

```
void foo()
{
    // berechne Kosten
    kosten = a * b + c;
    kosten -= discount;
}
```



```
void foo()
{
    berechneKosten();
}
```

```
void berechneKosten()
{
    kosten = a * b + c;
    kosten -= discount;
}
```

Entstehung

- seit jeher im Sinne eines „guten“ Programmierstils
- Seit den 80ger Jahren üblich (Smalltalk)
- 1999: Martin Fowler veröffentlicht DEN Refactoring-Katalog, anerkannt als Standard
- Refactoring ist ein wichtiger Bestandteil von extreme Programming

Wozu Refactoring?

Durch **Refactoring**:

- wird Code verständlicher
- wird die Fehlersuche beschleunigt
- lässt sich Code leichter modifizieren
- wird das Design eines Software-Systems verbessert
- schnelleres Vorankommen beim Programmieren

Was soll refaktorisiert werden?

„Schlechter“ Code !!!

Dieser entsteht durch:

- ständiges Ändern
- Copy und Paste
- ständiges Hinzufügen von Funktionalität
- fehlende oder unklare Requirements
- Termindruck

Bad Smells

- **„Bad Smells“**: typische Strukturen im Code, die ein Refactoring nahe legen

- Hier gibt es keine objektiven Kriterien
- lediglich Tendenzen in Richtung Refactoring erkennbar

„If it stinks, change it!“

[Grandma Beck]

Bad Smell 1

Duplicated Code:

- Hier existiert die gleiche Codestruktur an mehr als einer Stelle

Bad Smell 2

Long Parameter List:

- Lange Parameterlisten:
  - kompliziert zu begreifen
  - unhandlich im Gebrauch

Bad Smell 3

Large Class:

- Klasse erledigt die Arbeit von mehreren anderen Klassen
- Zeichen:
  - große Anzahl von Instanzvariablen
  - viele und lange Methoden / zu viel Code

Bad Smell 4

Feature Envy:

- Kapselung als zentrales Prinzip der objekt-orientierten Programmierung
- Hier beziehen sich die Methoden mehr für Objekte anderer Klassen, als für ihre eigenen

Bad Smell 5

Shotgun Surgery:

- Viele kleinere Änderungen folgen zwangsweise auf eine simple Änderung an einer Stelle

Wann wird refaktorisiert?

- in kleinen Schritten während des **gesamten** Projektverlaufs
- „Rule of Three“ als Orientierung

Aber **besonders:**

- zum besseren Verständnis von Code
  - bei der Fehlerfeststellung
  - vor Codereviews
- vor dem Hinzufügen neuer Funktionalität

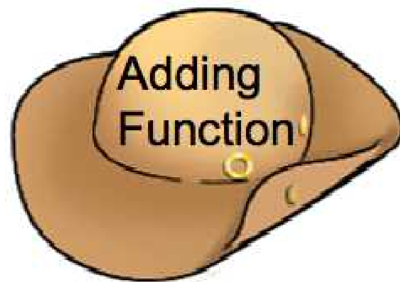
Wann sollte man NICHT refaktorisieren?

- **Code ist zu schlecht:**
  - lässt sich nicht kompilieren
  - besitzt viele Fehler
  - Redesign, da Refactoring zu viel Zeit in Anspruch nehmen würde
- **Deadline drängt:**
  - Termin könnte nicht eingehalten werden
  - Produktivität vorrangig, Refactoring nachträglich

Diszipliniertes Refactoring

„The Two Hats"-Prinzip:

Entweder Funktionalität hinzufügen oder refaktorisieren!



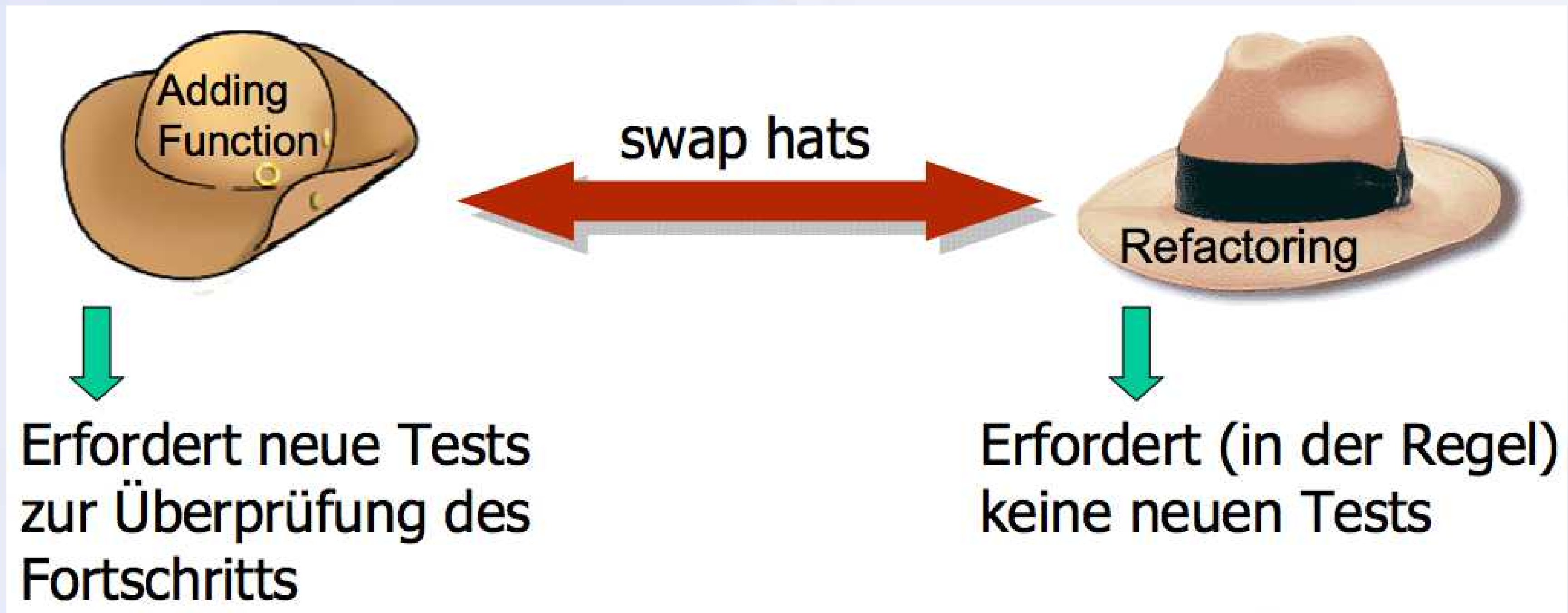
swap hats

WARUM ?

Diszipliniertes Refactoring

„The Two Hats"-Prinzip:

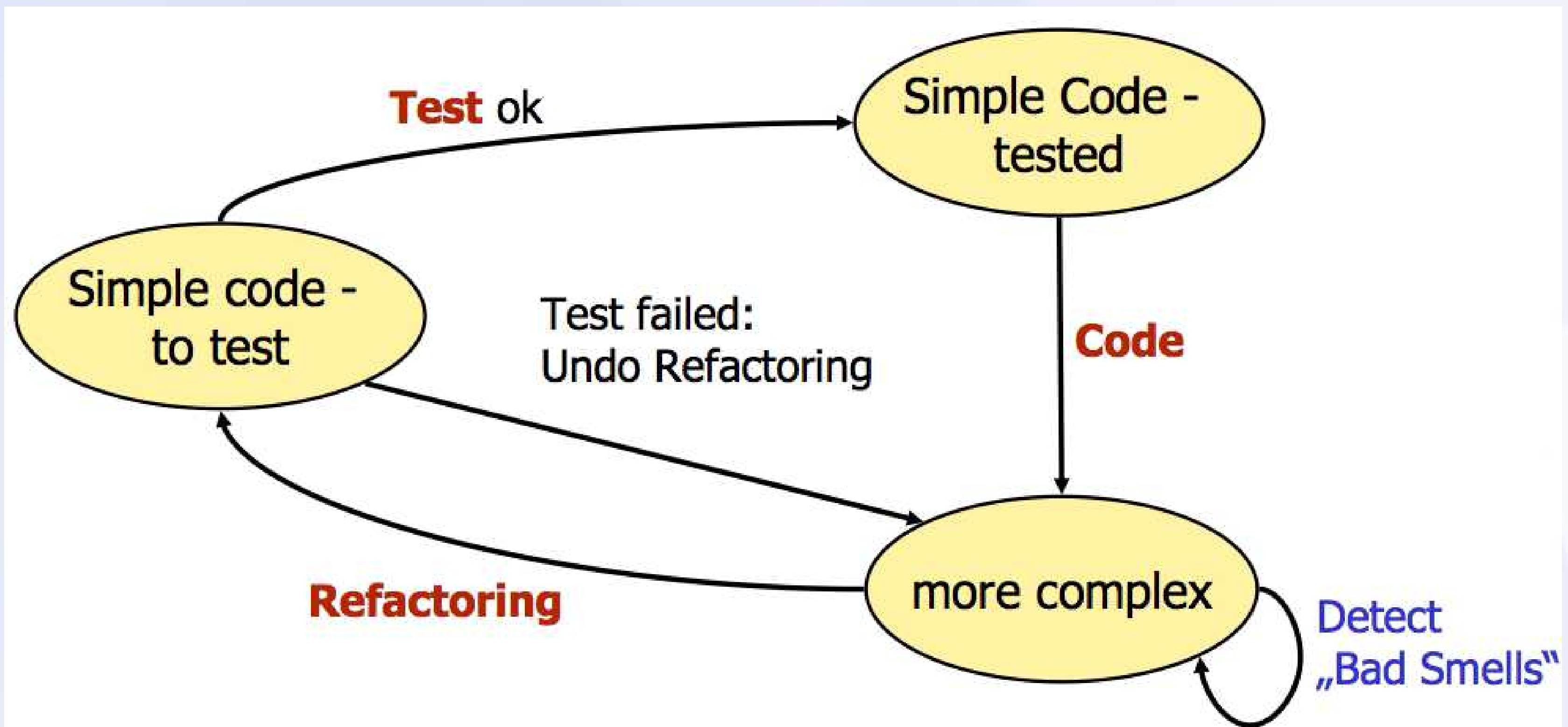
Entweder Funktionalität hinzufügen oder refaktorisieren!



Was ist zu beachten?

- **Zyklus einhalten:**

  Test → Code → Refactor → Test → Code → ...



Katalogisierung

- Zur Behebung von Bad Smells:
  - Anwenden einer oder mehrerer Refactoring-Operation(en) nötig
- Im **Katalog:**
  - „Bad Smells“ und wie man sie findet
  - gesammelte Refactoring-Muster, die sich bewährt haben
    - ähnlicher Ansatz wie beim „Design Patterns“-Katalog von der „Gang of Four“

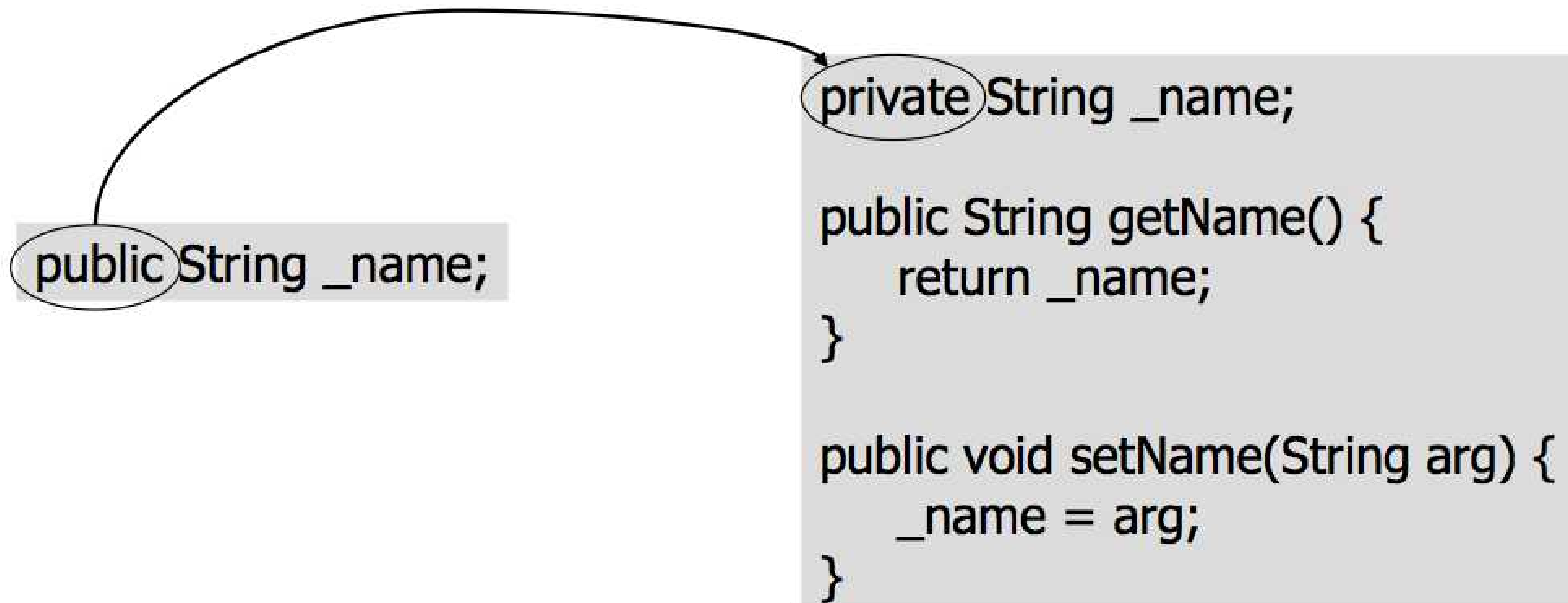
Katalog-Aufbau

Klassifizierung von Refactoring-Operationen:

- Umstrukturieren von Methoden
- Eigenschaften zwischen Objekten bewegen
- Daten organisieren
- Vereinfachen logischer Ausdrücke
- Vereinfachen von Methodenaufrufen
- Veränderungen der Vererbungshierarchie

Refactoring-Operation 1

Encapsulate Field

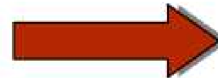


Refactoring-Operation 2

Introduce Parameter Object

You have a group of parameters that naturally go together.
- *Replace them with an object.*

Student
+setGeburtstag(in tag : int, in monat : int, in jahr : int) +setStudiumBeginn(in tag : int, in monat : int, in jahr : int) +setStudiumEnde(in tag : int, in monat : int, in jahr : int)



Student
+setGeburtstag(in datum : Date) +setStudiumBeginn(in datum : Date) +setStudiumEnde(in datum : Date)

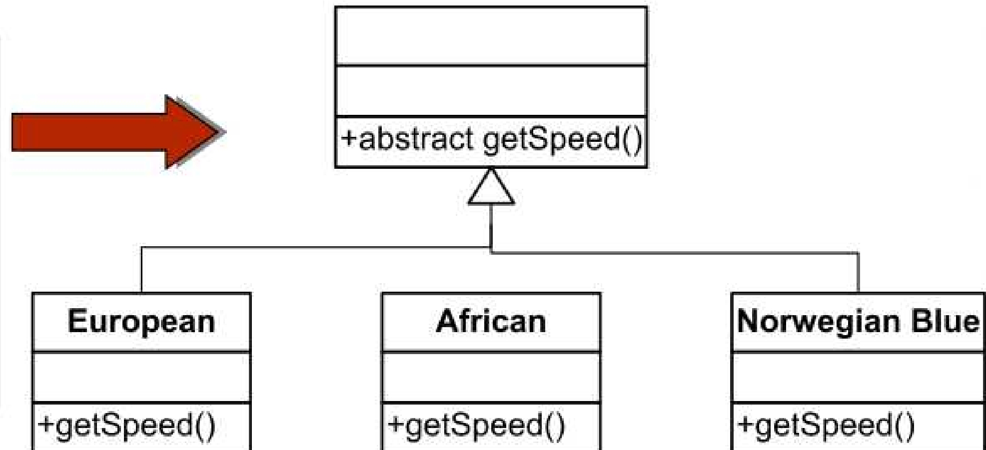
Refactoring-Operation 3

Replace Conditional with Polymorphism

You have a conditional that chooses different bahvior depending on the type of an object.

- *Move each leg of the conditional to an overriding method in a subclass.*
- *Make the original method abstract.*

```
double getSpeed() {
    switch(_type) {
            case EUROPEAN: ... ;
            case AFRICAN: ... ;
            case NORWEGIAN_BLUE: ... ;
    }
}
```

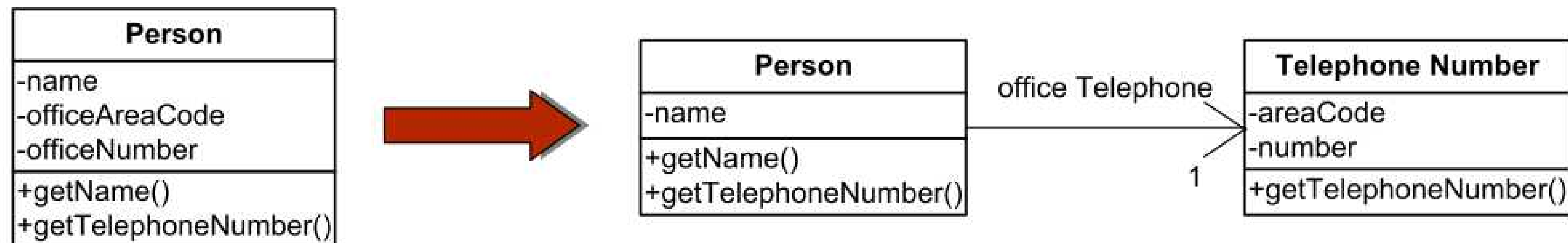


Extract Class - ausführlich

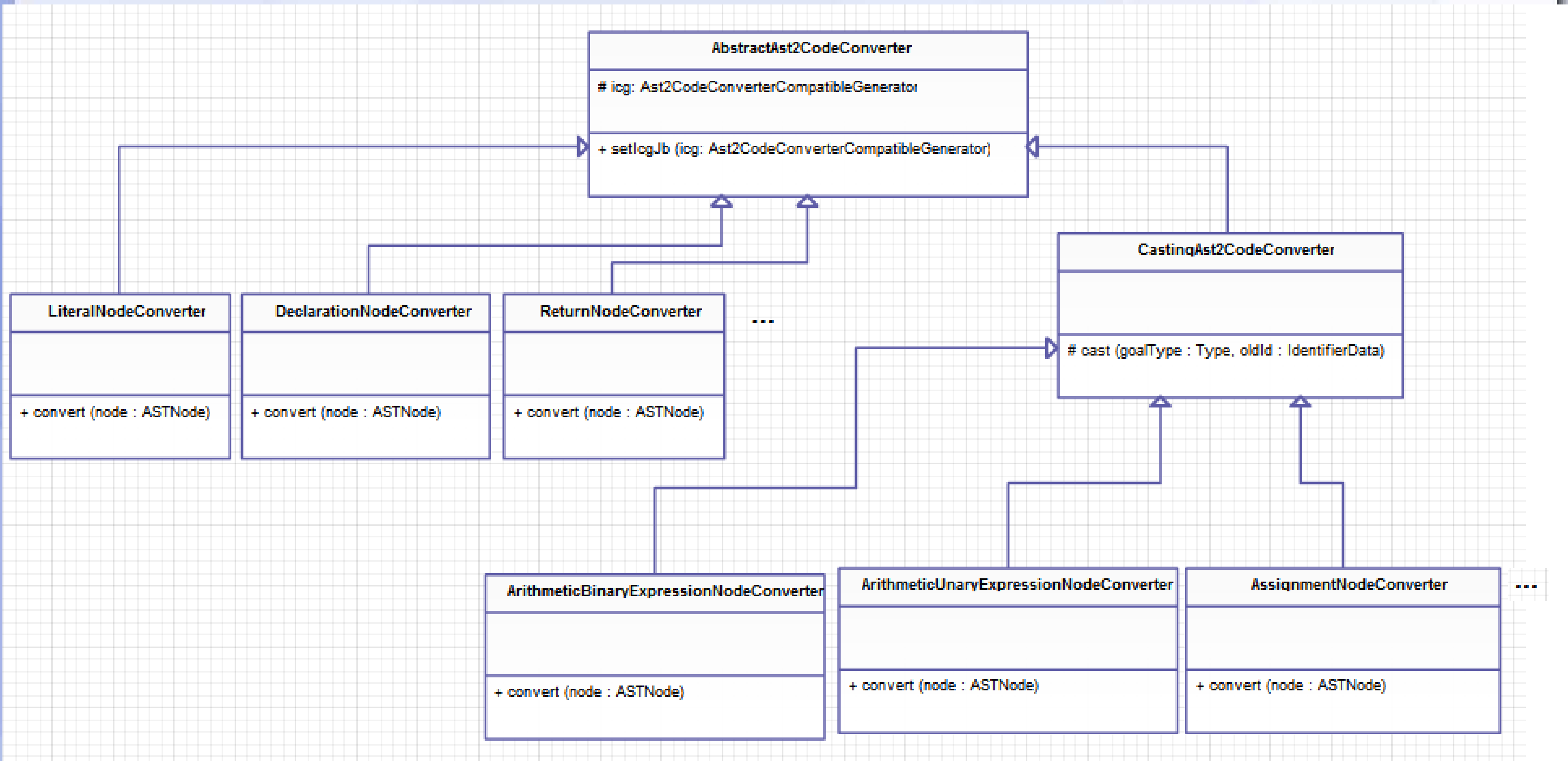
Extract Class

You have one class doing work that should be done by two.

- *Create a new class*
- *move the relevant fields and methods from the old class into the new.*



Refactoring-Beispiel bzgl. Javabite Codegen



Nachteile

Refactoring:

- kann umständlich sein
- sowie auch zeitaufwändig
- Fehleranfälligkeit macht häufiges Testen notwendig
- erfordert eine disziplinierte Arbeitshaltung
- führt „scheinbar“ zuerst zu einer geringen Performance

Refactoring-Werkzeuge

- automatisieren das Refactoring
- reduzieren den Zeitaufwand
- vermeiden manuelle Refactoring-Fehler
- Auswirkungen einer Refactoring-Operation einschätzen + Hinweise auf mögliche Probleme

Analysewerkzeuge

Werkzeugtypen für die Analyse (Smelling)

- Metrics Engines
  - Strukturschwächen-Analyse mit Hilfe von Metriken (**Quantitative Analyse**)
- Problem Detection
  - Spezifische Strukturschwächen-Analyse mit Hilfe von vordefinierten Regeln (**Qualitative Analyse**)
  - Regeln spielen Erfahrungen wieder

Restructing-Werkzeuge

- Existierende Restructing-Werkzeuge

Name	Typ
RefactorIt	Standalone
Xrefectory	Plugin für Editor (Emacs/XEmacs)
jFactor	Plugin für IDE (JBuilder/VisualAge)
Eclipse	IDE

- Problem: Es gibt bisher kein Werkzeug, dass Analyse und Refactoring zusammen anbietet
  - Erster Schritt in die Richtung ist JArt

Fazit

- evolutionäre Softwareentwicklung
- Refactoring hat auch seine Grenzen
- Unterlassen von Refactoring bringt Nachteile
- Refactoring bringt langfristige Vorteile
- Testen als fester Bestandteil von Refactoring
- Handhabung mit Tools

Quellen

- **Refactoring, Ausarbeitung im Rahmen des Softwareprojekts im 2003/2004 von Danuta Ploch:** https://swt.cs.tu-berlin.de/lehre/sepr/ws0304/Ausarbeitungen/Refactoring.pdf
- [Fowler 1999] FOWLER, Martin: “Refactoring: Improving the Design of Existing Codeü. Addison Wesley, 1999.
- [Dudziak, Wloka 2002] DUDZIAK, Thomas; WLOKA, Jan: “ Tool-Supported Discovery and Refactoring of Structural Weakness in Codeü. Diploma Thesis of the Faculty of Computer Science Technical University of Berlin, 2002.
- [Wloka 2001] WLOKA, Jan: “ Refactoring@OOSEü . TU-Berlin, 2001.

  http://swt.cs.tuberlin.de/lehre/oose/ss01/folien/Refactoring.pdf [20.11.2003]
- [Sokenou 2003] SOKENOU, Dehla: “ Refactoring@OOSEü . TU-Berlin, 2003.

  http://swt.cs.tu-berlin.de/lehre/oose/ss03/folien/x3.ps [02.07.2003]
- [Hunger, Vulpius 2000] HUNGER, Michael; VULPIUS, Steffen: “ Refactoringü . TU-Dresden, 2000.

  http://pdai.inf.tu- dresden.de/de/Sonstiges/Downloads/Vortrag_Michael_Hunger_Refactoring.pdf [19.11.2003]

Danke für die Aufmerksamkeit!