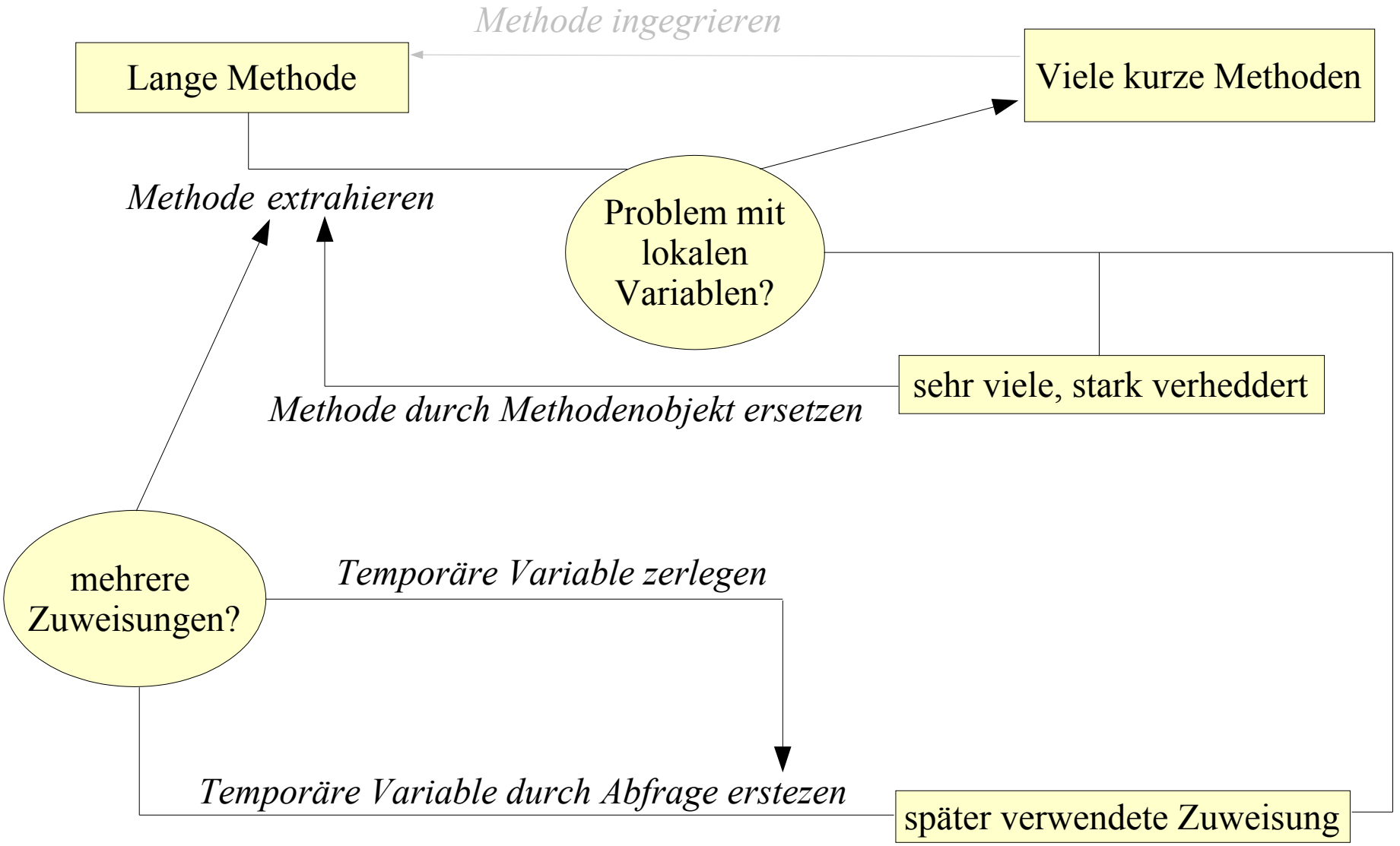
Refactoring I

Nach Martin Fowler - “Refactoring”

Kapitel 6 : Methoden zusammenstellen
Kapitel 9 : Bedingte Ausdrücke vereinfachen
Kapitel 10: Methodenaufrufe vereinfachen

- Mario Boley -

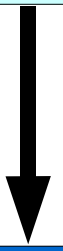
Methoden zusammenstellen



Methode extrahieren

Ausgangsituation:

Codefragment, das zusammengefasst werden kann



Nach Refactoring:

- Fragment ist neue Methode
- Name der Methode erklärt Aufgabe

Methode extrahieren - Warum?

- Nachteile langer Methoden
  - verbergen Informationen
  - zugrundeliegende Logik schwerer nachzuvollziehen

- Vorteile kurzer Methoden
  - bessere Wiederverwendbarkeit
  - Methoden höherer Ebenen besser lesbar
  - einfacheres Überschreiben

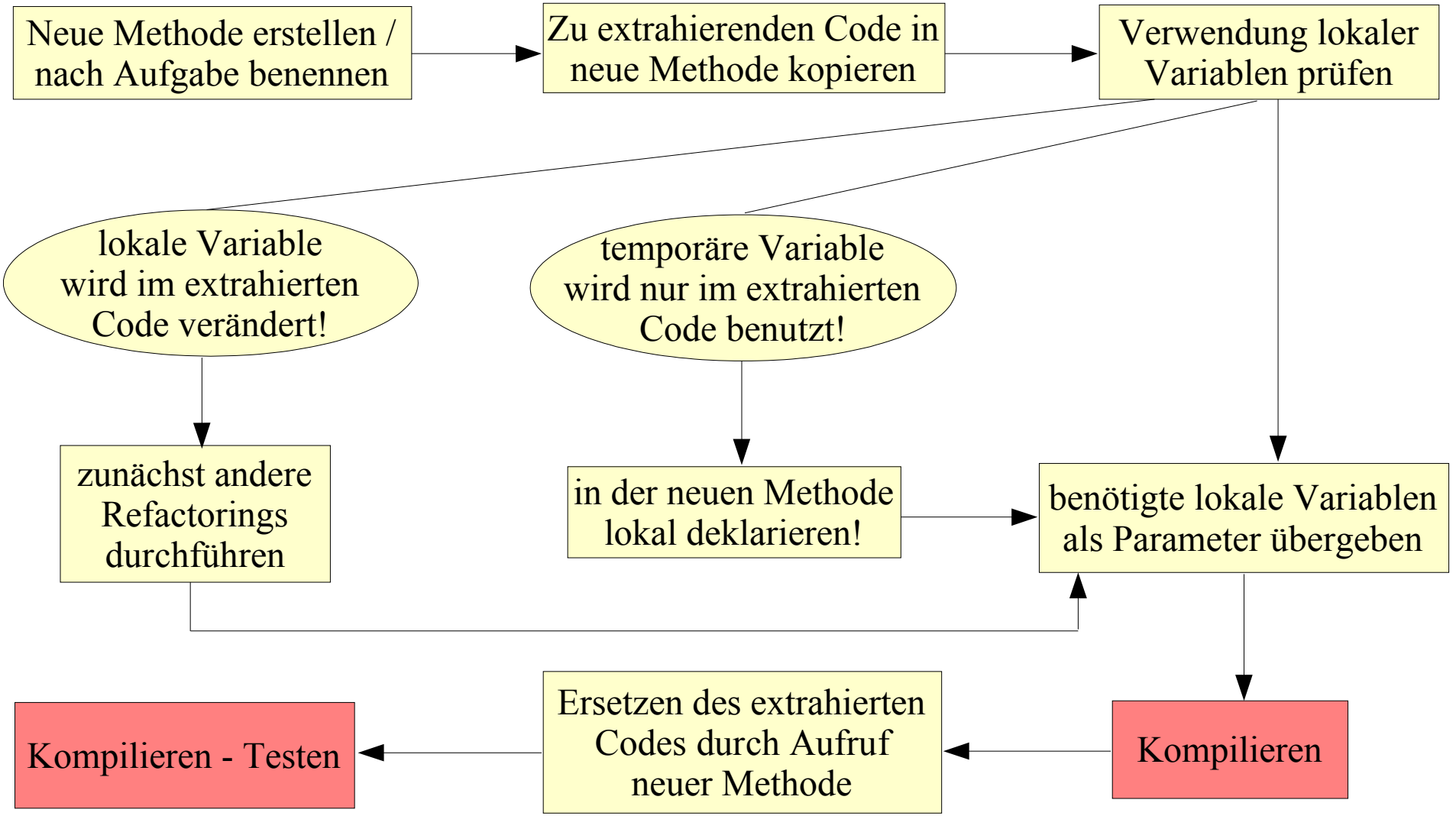
Methode extrahieren - Warum?

- Nachteile langer Metho
  - verb
    Infor
  - zugrundeliegende Logik schwerer nachzuvollziehen

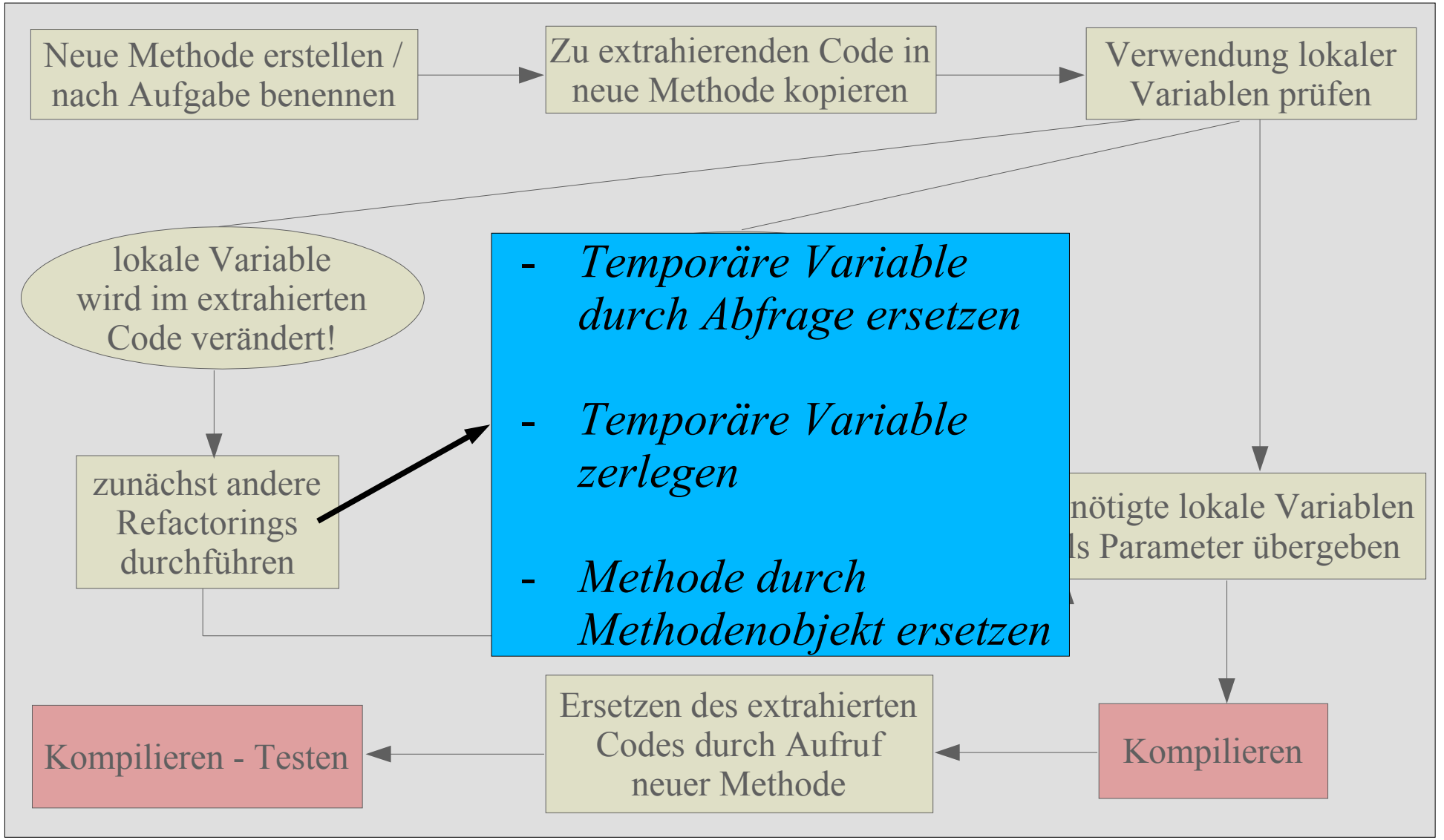
Wichtig dazu: aussagekräftige Methodennamen

- Vorteile kurzer Methoden
  - bessere Wiederverwendbarkeit
  - Methoden höherer Ebenen besser lesbar
  - einfacheres Überschreiben

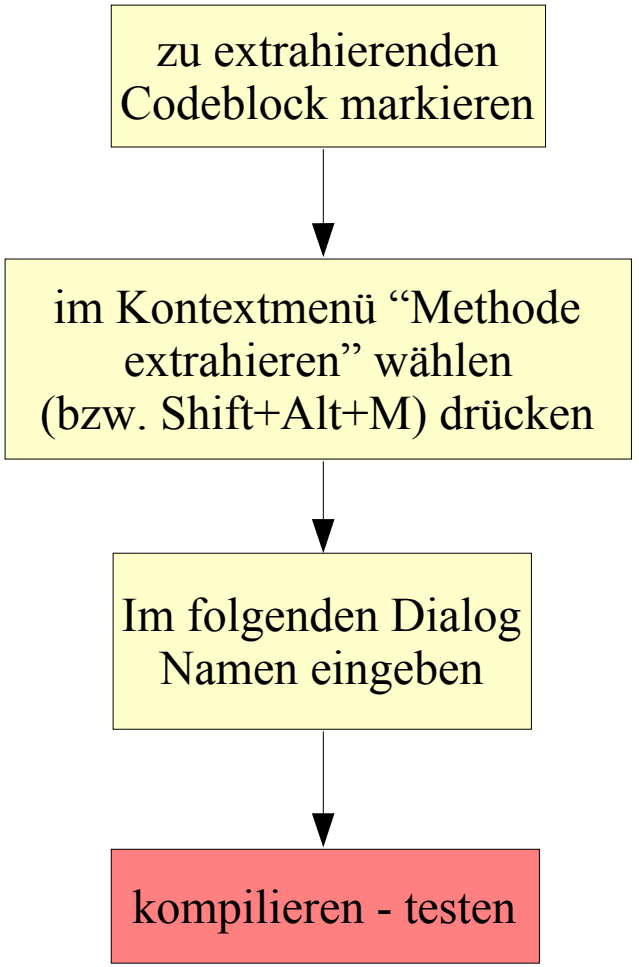
Methode extrahieren – Vorgehen ohne Eclipse



Methode extrahieren - Vorgehen



Methode extrahieren – Vorgehen mit Eclipse



Temporäre Variable durch Abfrage ersetzen

Ausgangsituation:

temporäre Variable speichert Ergebnis eines Ausdrucks

Nach Refactoring:

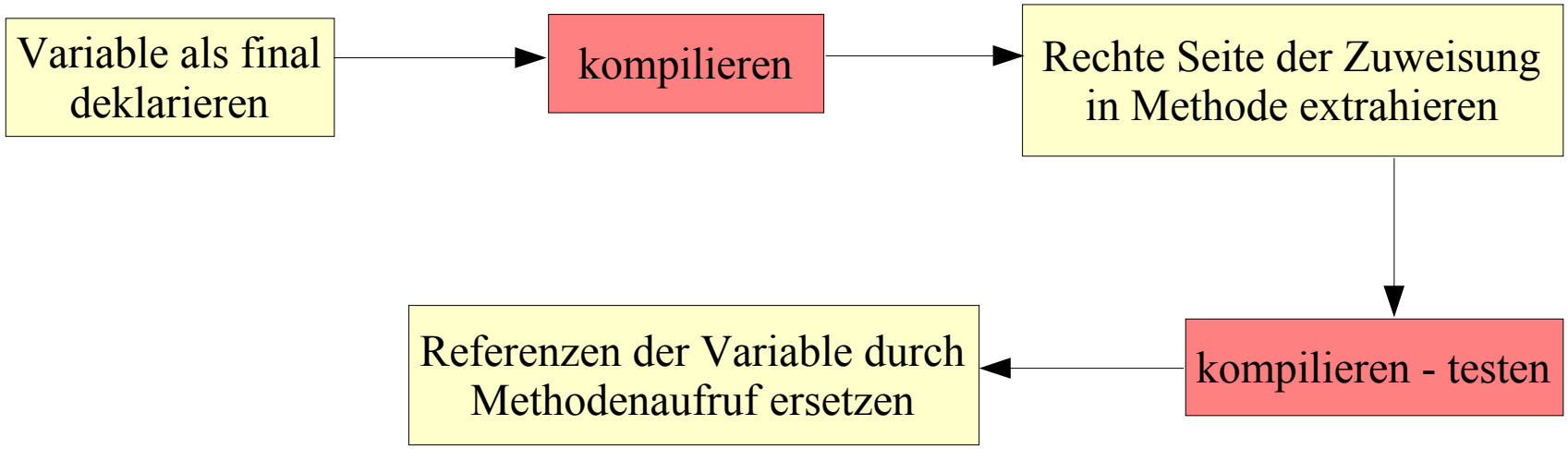
- Audruck durch Methode ersetzt
- Referenzen der Variable durch Aufruf dieser Methode ersetzt

Temporäre Variable durch Abfrage ersetzen - Warum?

- temporäre Variablen
  - nur lokal benutzbar
  - fördern das Schreiben langer Methoden
  - müssen oft entfernt werden, damit *Methode extrahieren* angewendet werden kann

- Abfragemethode
  - kann von gesamter Klasse benutzt werden
  - fördert Enstehung von klarem Code

Temporäre Variable durch Abfrage ersetzen - Vorgehen



Temporäre Variable zerlegen

Ausgangsituation:

temporärer Variable wird mehrfach etwas zugewiesen
ist aber weder Schleifenvariable
noch Ergebnis sammelnd

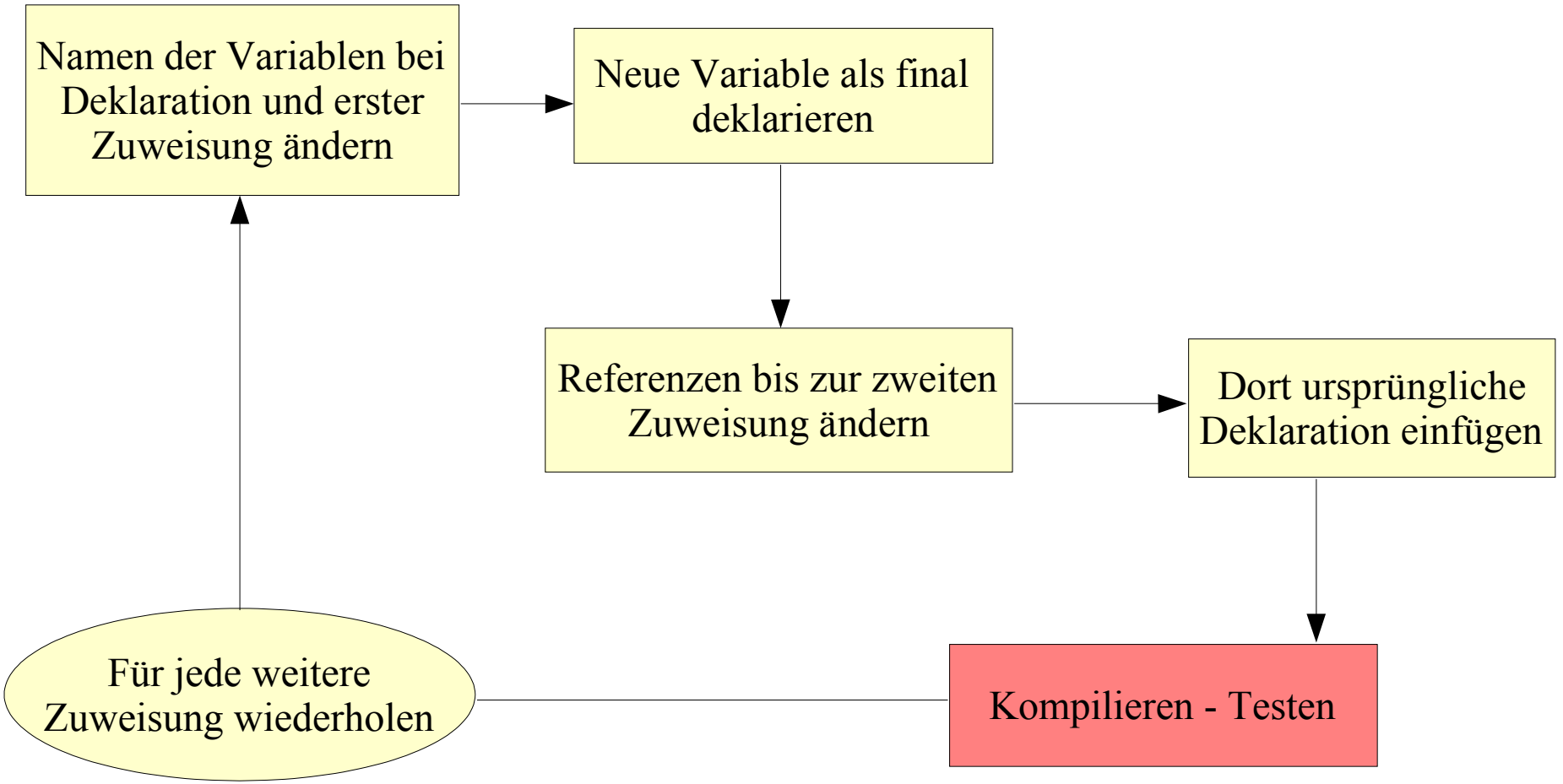
Nach Refactoring:

für jede Zuweisung eigene temporäre Variable

Temporäre Variable zerlegen - Warum?

- Klare Zuordnung Variable / Verantwortlichkeit
- Erhöht Lesbarkeit
- Vereinfacht späteres Zerlegen

Temporäre Variable zerlegen - Vorgehen



Methode durch Methodenobjekt ersetzen

Ausgangsituation:

lange Methode mit mehreren lokalen Variablen, die verhindern, daß *Methode extrahieren* angewandt werden kann

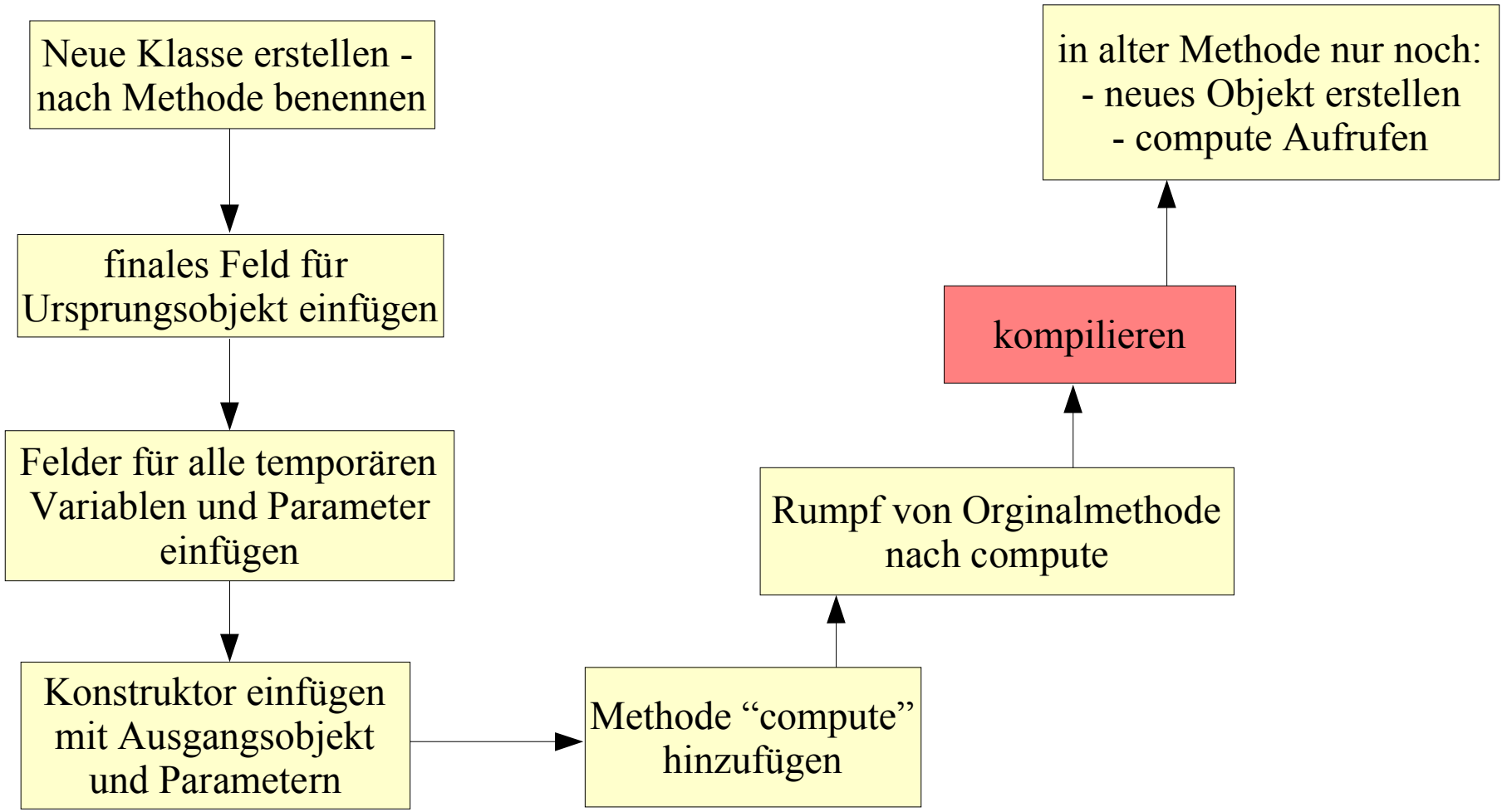
Nach Refactoring:

- Methode wird eigene Klasse
- lokale Variablen Felder dieser Klasse
- Neue Klasse hat Verweis auf Ausgangsklasse

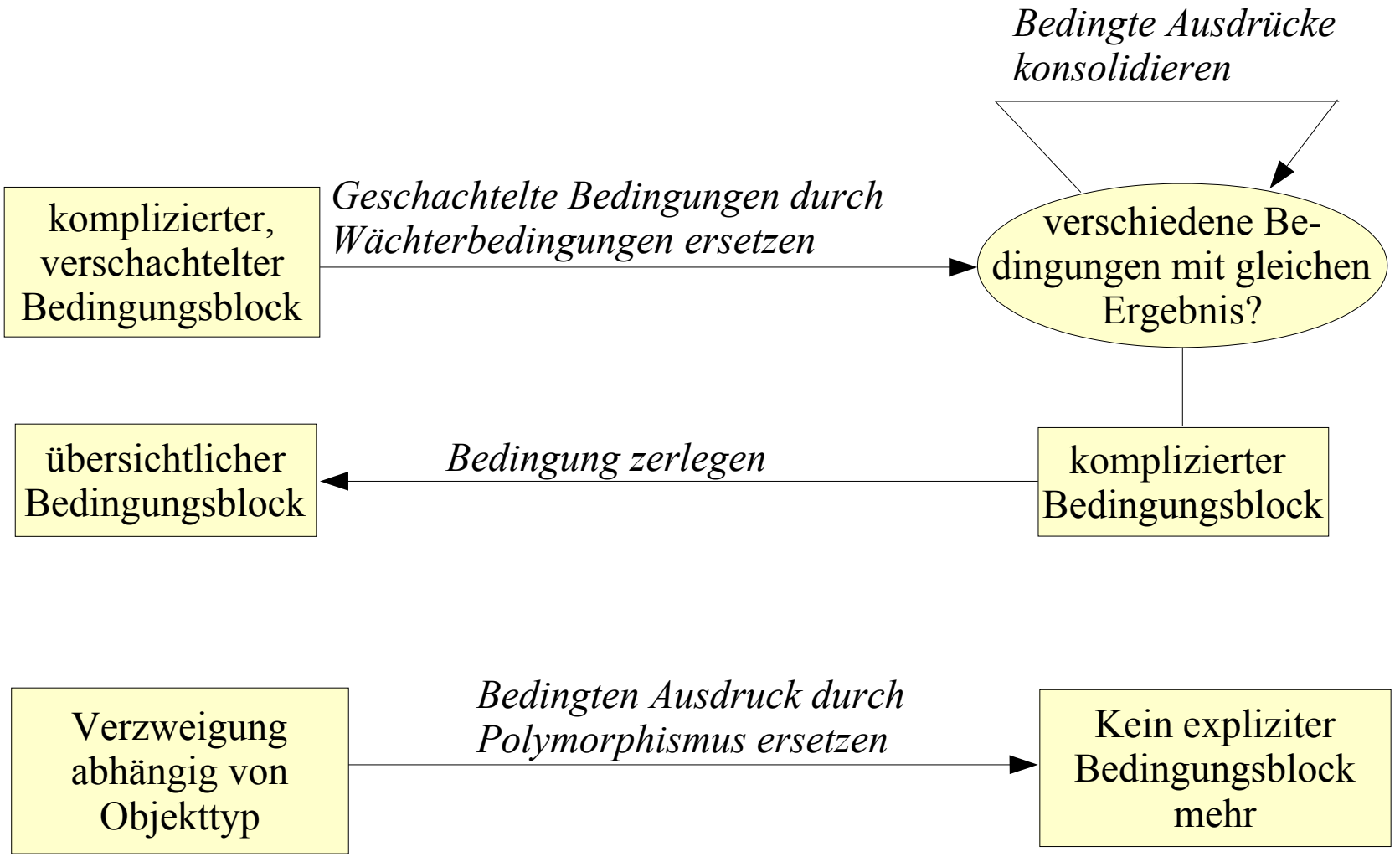
Methode durch Methodenobjekt ersetzen - Warum?

- Zerlegen von Methoden wird durch lokale Variablen erschwert
- Manchmal reicht *temporäre Variable durch Abfrage ersetzen* nicht aus
- Auf der neuen Klasse kann einfach *Methode extrahieren* angewandt werden

Methode durch Methodenobjekt ersetzen - Vorgehen



Bedingte Ausdrücke vereinfachen



Geschachtelte Bedingungen durch Wächterbedingungen ersetzen

Ausgangsituation:

verschachtelter Bedingungsblock,
der normalen Ablauf nicht leicht erkennen läßt

Nach Refactoring:

Sonderfälle vorgelagert durch “Wächterbedingungen”

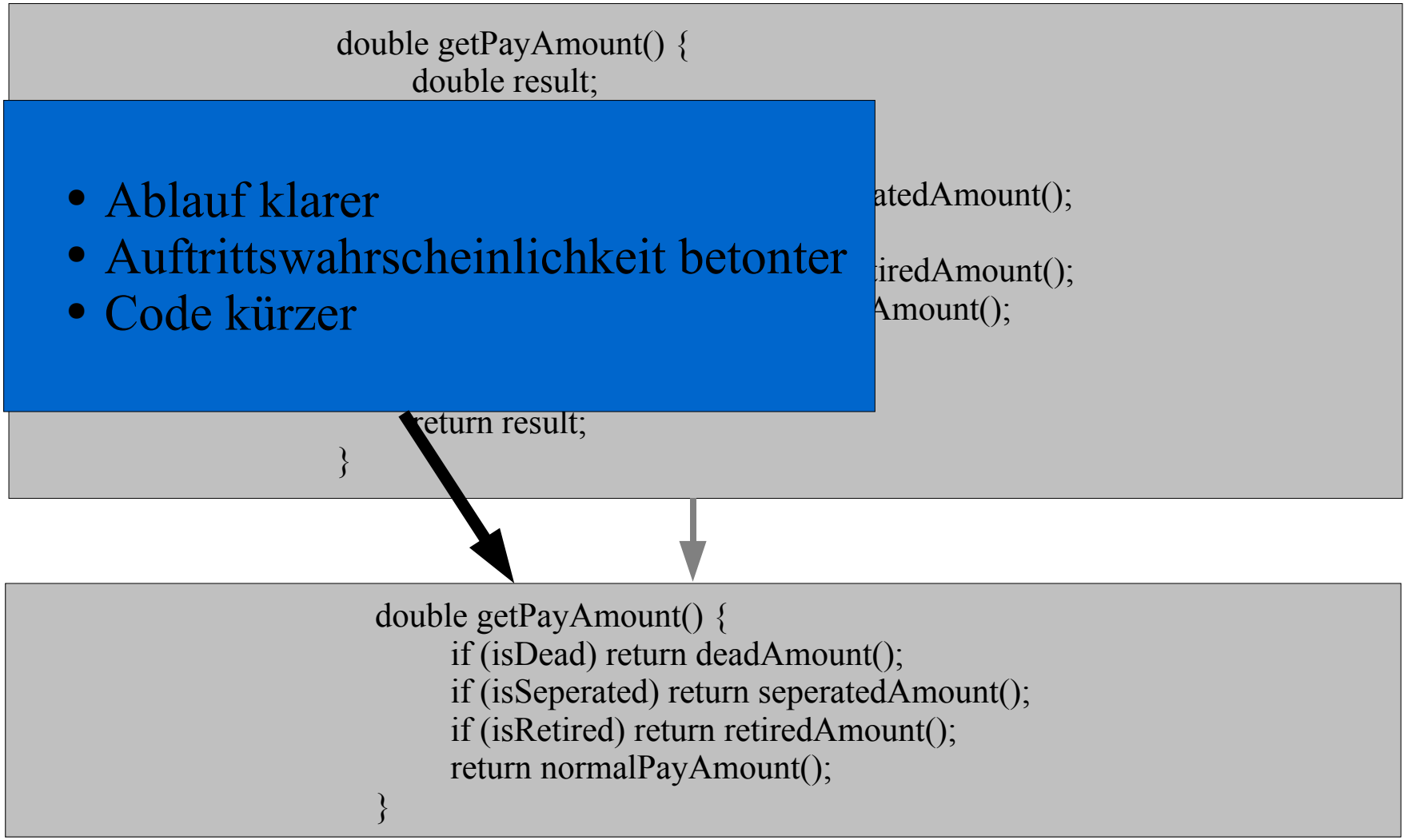
“Bewachen” den eigentlichen Block vor Ausnahmen

Geschachtelte Bedingung durch Wächterbedingung ersetzen - Beispiel

```
double getPayAmount() {
        double result;
        if (isDead) result=deadAmount();
        else {
                if (isSeparated) result=seperatedAmount();
                else {
                        if (isRetired) result=retiredAmount();
                        else result=normalPayAmount();
                }
        }
        return result;
}
```

```
double getPayAmount() {
        if (isDead) return deadAmount();
        if (isSeperated) return seperatedAmount();
        if (isRetired) return retiredAmount();
        return normalPayAmount();
}
```

Geschachtelte Bedingung durch Wächterbedingung ersetzen - Beispiel



Geschachtelte Bedingung durch Wächterbedingung ersetzen - Vorgehen

- Für jede Prüfung Wächterbedinung einrichten
- Wächterbedingung führt entweder zu “return” oder wirft Exception
- Nach jeder Ersetzung kompilieren und testen

Bedingte Ausdrücke konsolidieren

Ausgangsituation:

Folge von Bedingungen mit gleichen Ergebnis

Nach Refactoring:

Bedingungen kombiniert und in eigene Methode extrahiert

Bedingte Ausdrücke konsolidieren - Warum?

- Prüfung wird übersichtlicher
- Zusammengehörigkeit von verschiedenen Prüfungen wird deutlich
- Oft wird *Methode extrahieren* möglich

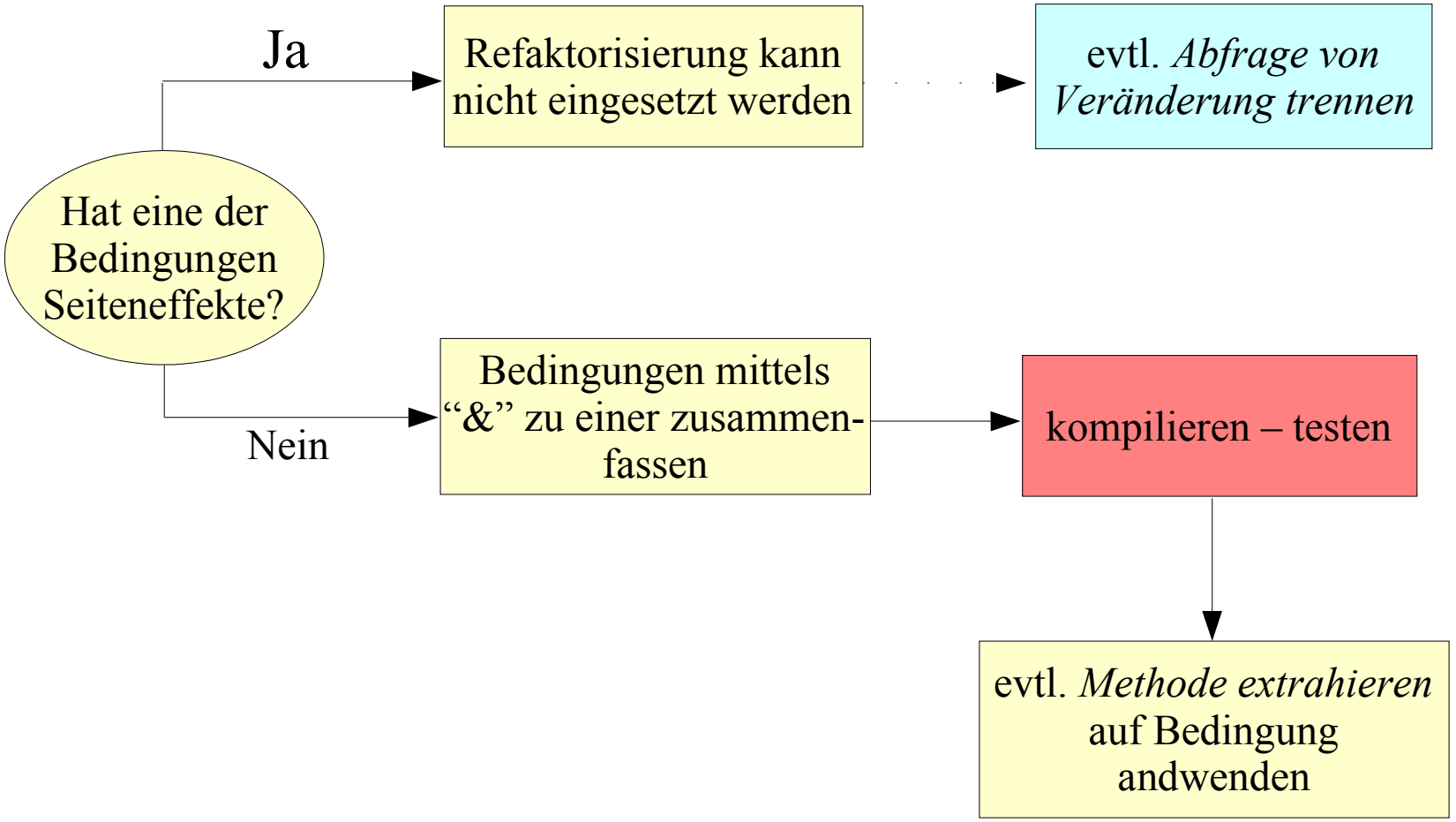
Bedingte Ausdrücke konsolidieren - Warum?

- Prüfung wird übersichtlicher
- Zusammengehörigkeit von verschiedenen Prüfungen wird deutlich
- Oft wird *Methode extrahie*

Diese Refaktorisierung nicht durchführen, wenn:

- Bedingungen stehen nur zufällig nebeneinander
- logisch nicht verknüpft

Bedingte Ausdrücke konsolidieren - Vorgehen



Bedingung zerlegen

Ausgangsituation:

komplizierte if-then-else-Bedingung

Nach Refactoring:

extrahierte Methode für:

- die Bedingung
- den then-Zweig
- den else-Zweig

Bedingung zerlegen

- Gleiche Motivation wie bei *Methode extrahieren*
- Genauso das Vorgehen. Je einmal *Methode extrahieren* für:
  - Bedigung
  - Then-Block
  - Else-Block

Bedingten Ausdruck durch Polymorphismus ersetzen

Ausgangsituation:

Bedingung wählt unterschiedliches Verhalten aus - abhängig vom Typ eines Objekts

Nach Refactoring:

- Jeder Bedingungszweig in überschreibender Methode einer Unterklasse
- ursprüngliche Methode abstrakt

Bedingten Ausdruck durch Polymorphismus ersetzen

Entsprechende Vererbungsstruktur muß vorhanden sein oder erstellt werden:

- *Typenschlüssel durch Unterklassen ersetzen*

- *Typenschlüssel durch Zustand/Strategie ersetzen*

liches Verhalten aus -
ekts

Nach Refactoring:

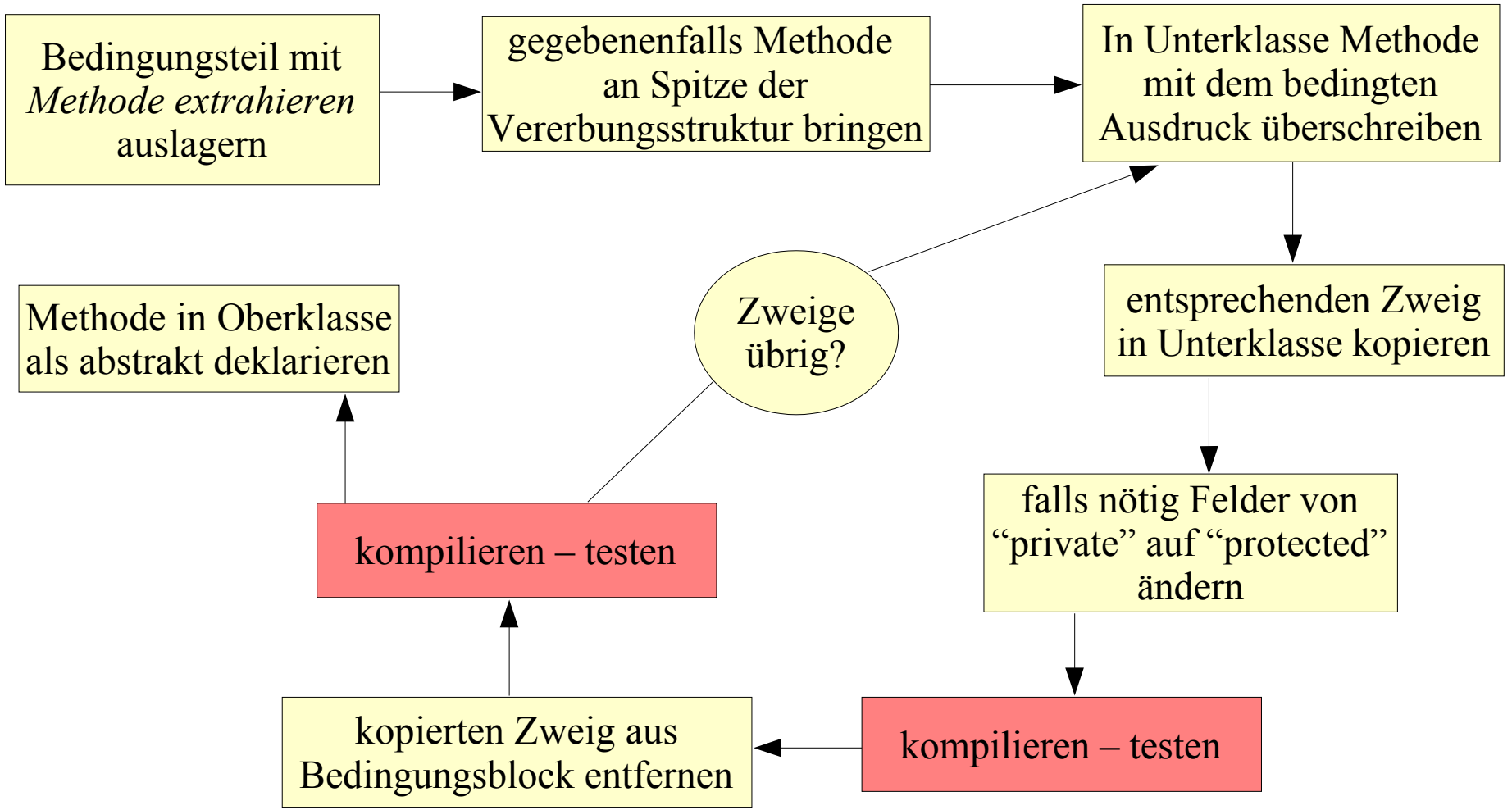
- Jeder Bedingungszweig in überschreibender Methode einer Unterklasse

- ursprüngliche Methode abstrakt

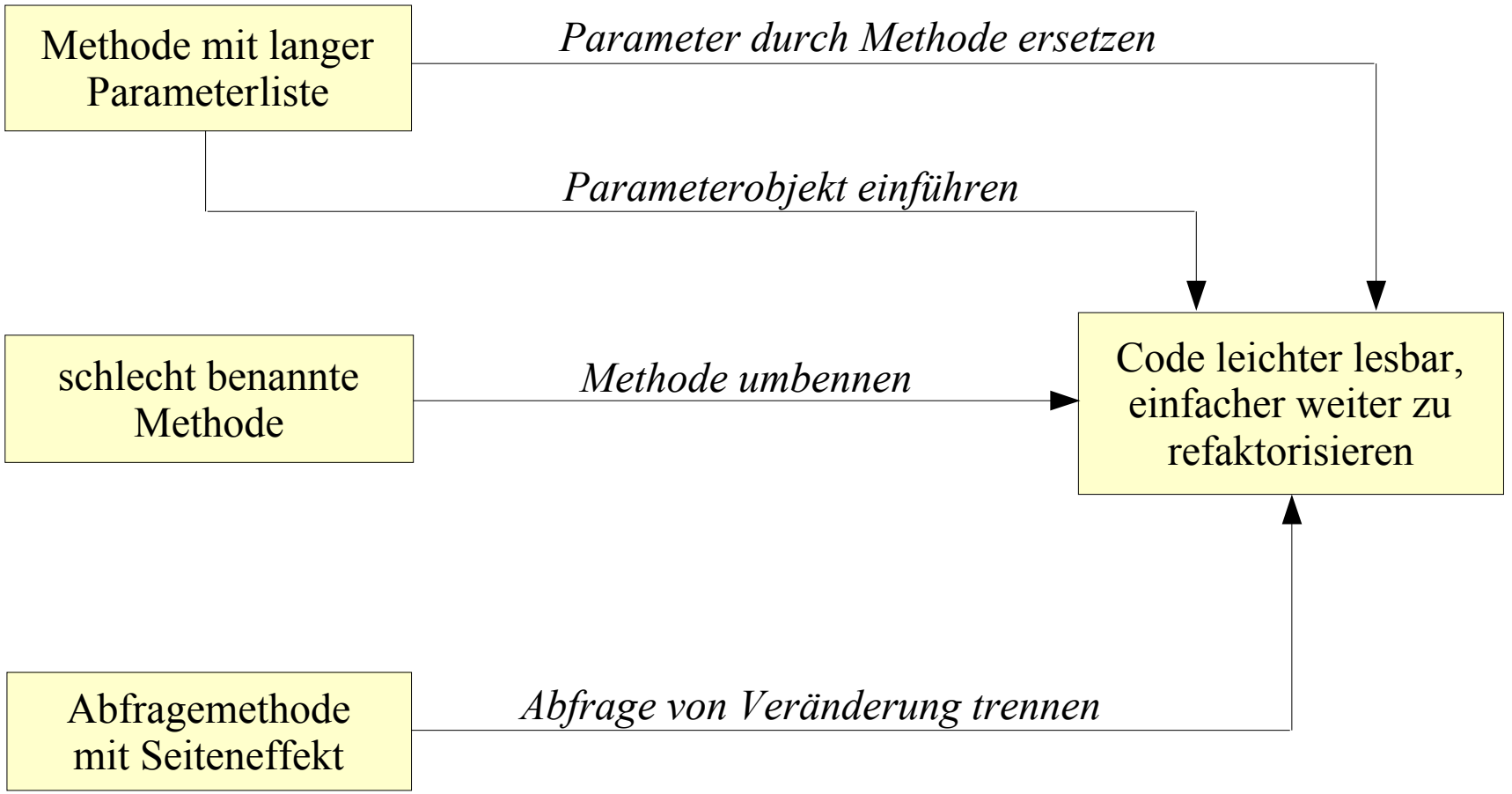
Bedingten Ausdruck durch Polymorphismus ersetzen - Warum?

- Explizite Bedingungen werden überflüssig
- Hinzufügen eines neuen Typs erfordert nur noch Erstellung neuer Unterklasse
- Abhängigkeiten werden verringert

Bedingten Ausdruck durch Polymorphismus ersetzen - Vorgehen



Methodenaufrufe vereinfachen



Methode umbenennen

Ausgangsituation:

Methode hat Namen, der ihre Aufgabe nicht klar macht

Nach Refactoring:

Methode hat einen guten Namen :-)

Methode umbennen – Warum ?

- Viele kleine Methoden (siehe *Methode extrahieren*) sind gewünscht
- Bringen vollen Nutzen aber nur bei guter Bezeichnung
- sind für schnelles Verständnis des Codes unerlässlich

Methode umbenennen – Vorgehen in Eclipse

- Cursor auf beliebiges Vorkommen des Methodennamen
- Entweder:
  - “Shift+Alt+R” drücken oder
  - Im Kontextmenü “Refactor/Rename” auswählen
- Im folgenden Dialog neuen Namen eingeben

Abfrage von Veränderung trennen

Ausgangsituation:

eine Methode gibt einen Wert zurück,
ändert aber auch den Zustand des Objekts

Nach Refactoring:

- eine Methode für die Abfrage
- eine Methode für die Änderung

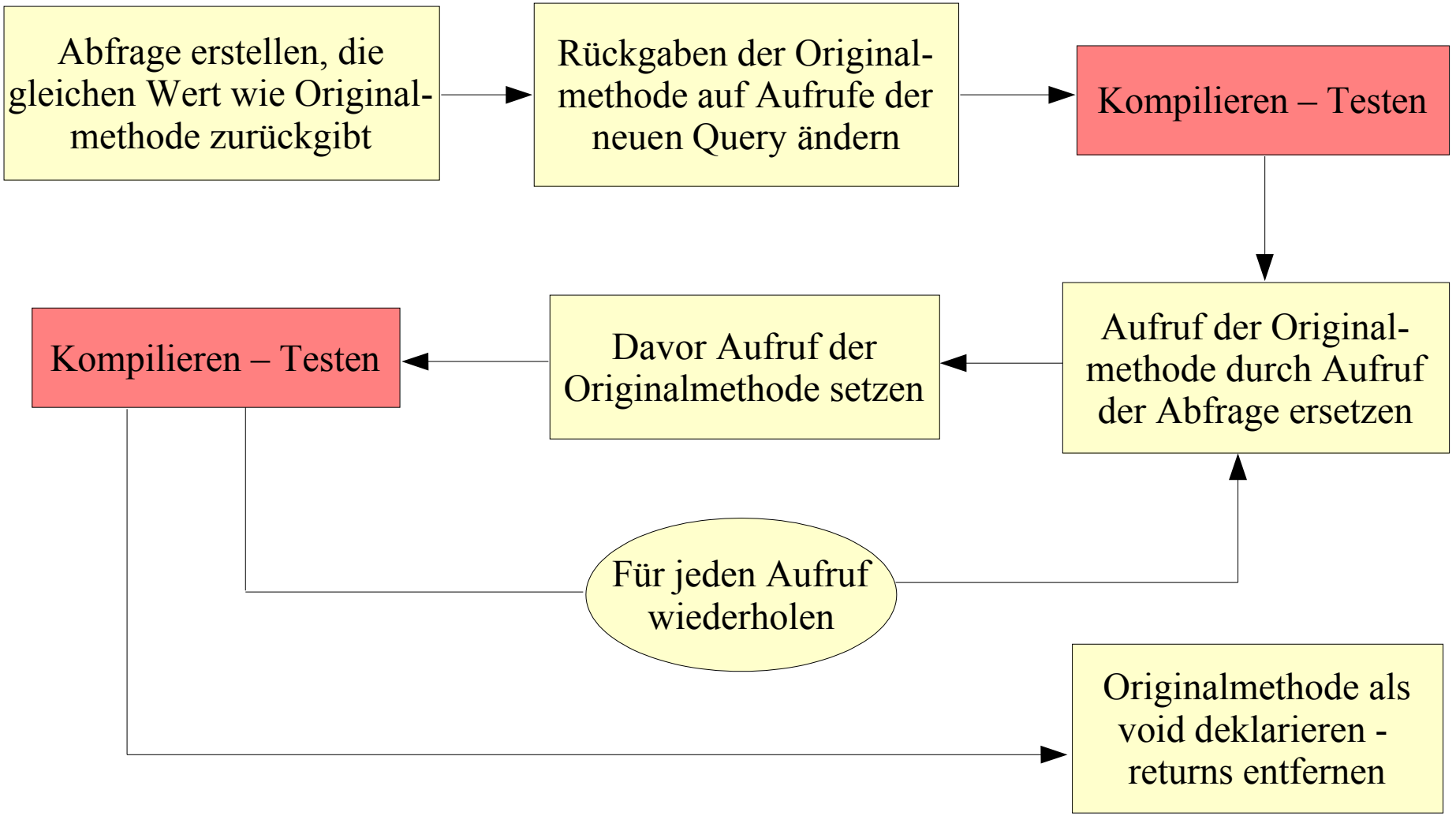
Abfrage von Veränderung trennen – Warum ?

- Methode ohne Seiteneffekte kann beliebig oft aufgerufen werden
- Aufrufe können einfach innerhalb der Methode verschoben werden

Abfrage von Veränderung trennen – Wann ?

- nur bei “erkennbaren” Seiteneffekten
- z.B. Verwendung von Pufferfeldern in Abfrage nicht erkennbar
- Da jede Abfragefolge ergibt gleiche Ergebnisse

Abfrage von Veränderung trennen - Vorgehen



Parameter durch Methode ersetzen

Ausgangsituation:

Methode A wird aufgerufen und mit Ergebnis als Parameter wird Methode B aufgerufen.

B könnte A aber auch selber aufrufen.

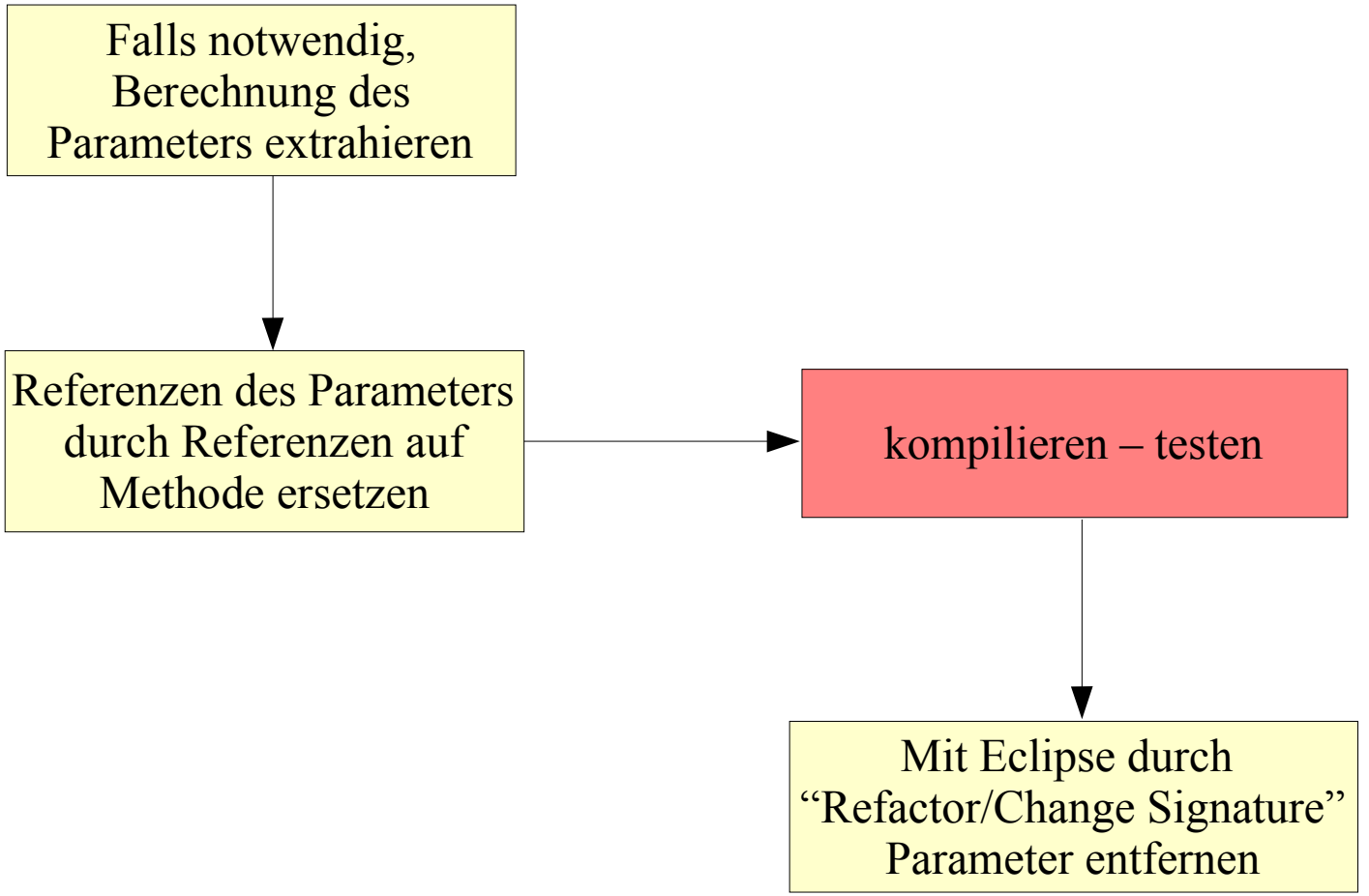
Nach Refactoring:

- gebenenfalls neue Abfragemethode für die Berechnung des Parameters
- Parameter entfernt

Parameter durch Methode ersetzen

- Motivation : kürzere Parameterlisten sind leichter zu verstehen
- Kann nicht durchgeführt werden, wenn Berechnung einen Parameter der aufrufenden Methode verwendet

Parameter durch Methode ersetzen – Vorgehen



Parameterobjekt einführen

Ausgangsituation:

Gruppe von Parametern, die auf natürliche Weise zusammengehören

Nach Refactoring:

- Klasse fasst Gruppe zusammen
- ein Objekt dieser Klasse wird als Parameter übergeben

Parameterobjekt einführen – Warum?

- Parameterliste wird kürzer
- Sinneinheit deutlicher
- Verhalten kann eventuell in die neue Klasse verschoben werden

Parameterobjekt einführen – Vorgehen