Effizientes Programmieren

Heutiges Thema:

Refactorings

Michaela Rindt
(mrindt@informatik.uni-siege.de)

Refactoring

- Entwicklungsmethode -

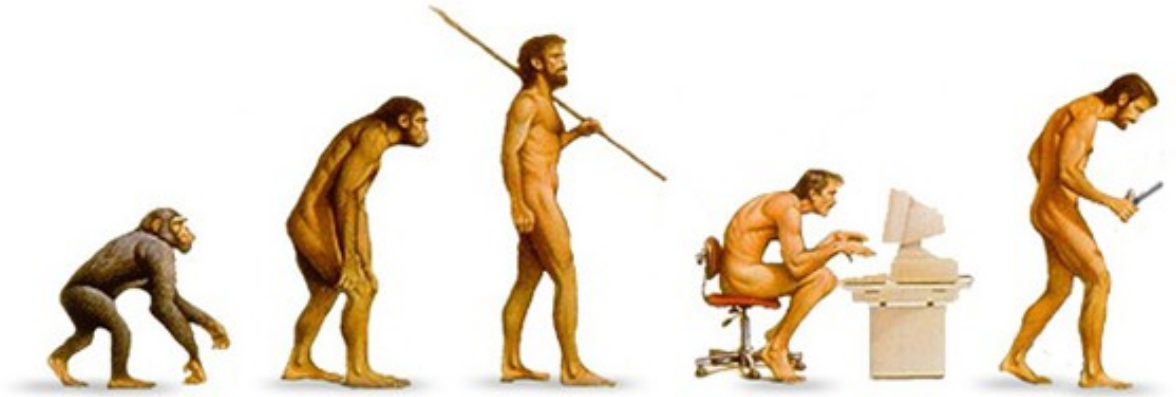


Bild-Quelle: http://www.goldbachinteractive.com/aktuell/kurzbeitraege/next-step-of-evolution

Refactoring

- … ist wesentlicher Bestandteil von Agiler Entwicklung (aber auch unabhängig davon)
- ... ist das **Verbessern** von vorhandenem Quellcode
- ... verändert nicht das äußere Verhalten eines Programmes, sondern nur seine interne Struktur.

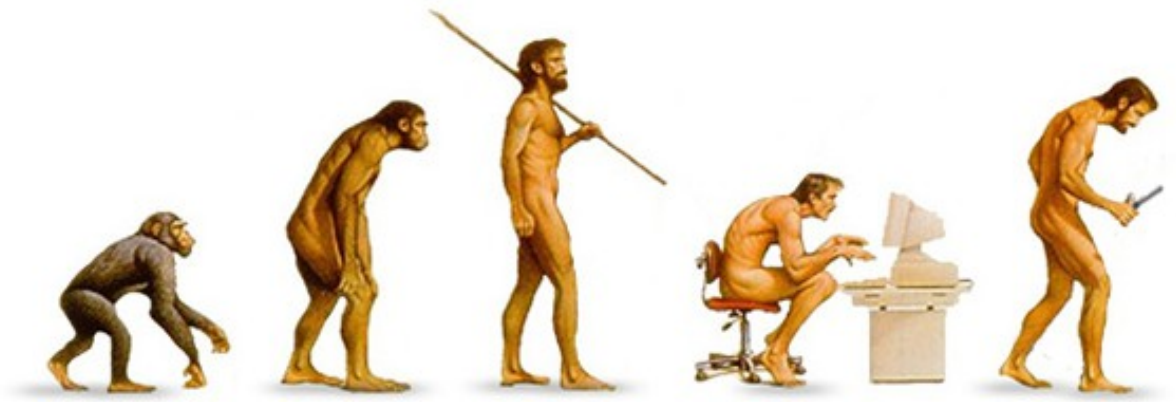


Bild-Quelle: http://www.goldbachinteractive.com/aktuell/kurzbeitraege/next-step-of-evolution

Refactoring

Positive Effekte:

- Lesbarkeit und Nachvollziebarkeit des Codes wird verbessert
- Leicht erweiterbarer Code, d.h. zukünftige Änderungen müssen nicht umständliche 'gehackt' werden
- Beseitigt **bad smells** und minimiert damit Wahrscheinlichkeiten für die Entstehung neuer Bugs
- Integrität und Qualität des Programmes wird langzeitlich gesichert
- Aufwand für Ergänzungen und die Einarbeit neuer Mitarbeiter wird minimiert

Refactoring

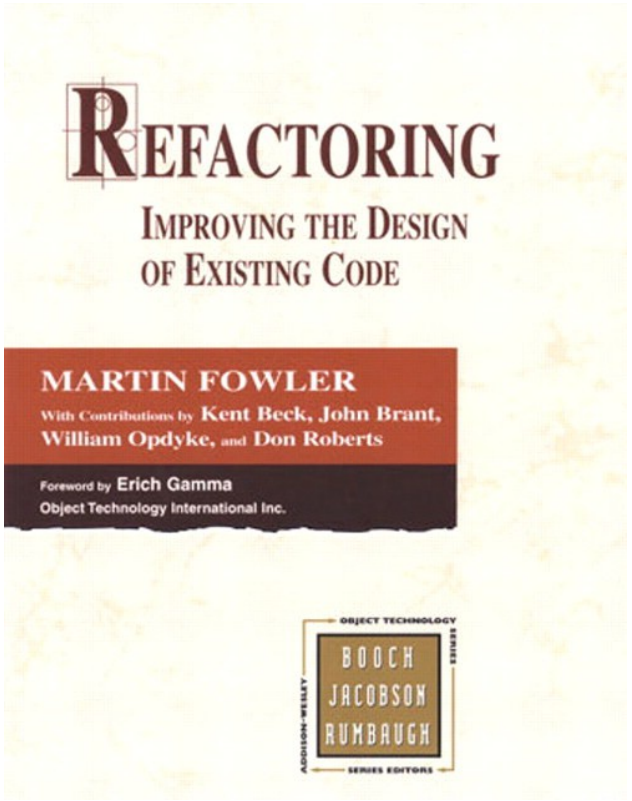
Typische Änderungen im Refactoring-Prozess:

- Reduktion oder auch Ergänzung von Kommentaren / Dokumentation
- Variablen- und Methodennamen Anpassung entsprechend ihrer Funktion
- Zusammenlegung oder auch Aufteilung oder Auslagern von Code-Fragmenten
- Entfernen von Redundanzen (z.B. überflüssige Imports / unnötige Methoden oder Variablen)
- Strukturelle Verschiebungen von Code-Fragmenten
- Testbarkeit von Funktionen ermöglichen

Beispiele für Refactorings

Alle folgenden Beispiele (u.v.m.) stammen von Martin Fowler. Sie sind in diesem Buch nachzuschlagen.

oder unter: http://refactoring.com/catalog/

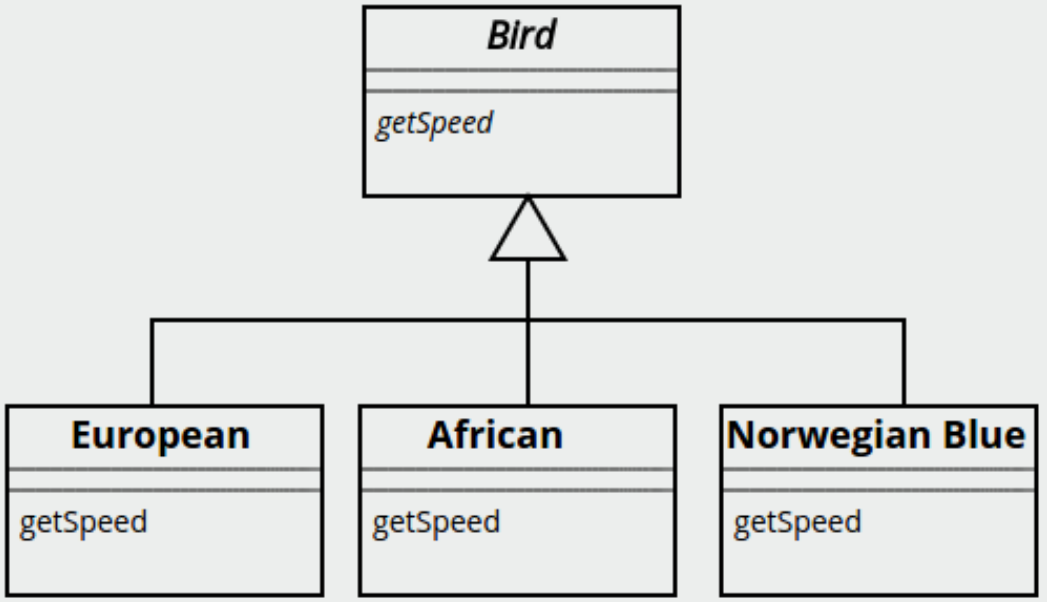


Bedingungen mit Polymorphismus ersetzen

Ähnlich verlaufende Bedingungen..

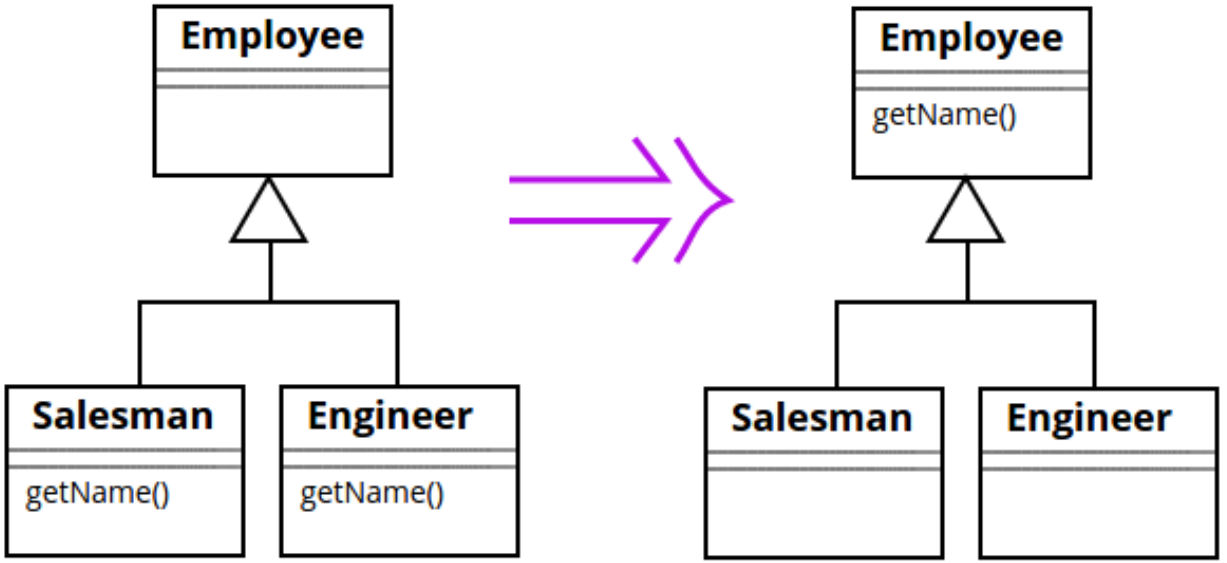
```
double getSpeed() {
  switch (_type) {
    case EUROPEAN:
      return getBaseSpeed();
    case AFRICAN:
      return getBaseSpeed() - getLoadFactor() * _numberOfCoconuts;
    case NORWEGIAN_BLUE:
      return (_isNailed) ? 0 : getBaseSpeed(_voltage);
  }
  throw new RuntimeException ("Should be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

durch abstrakte, überschreibbare Methode den Unterklassen bereitstellen



Quelle: http://refactoring.com/catalog/replaceConditionalWithPolymorphism.html

Methoden nach oben ziehen



Quelle: http://refactoring.com/catalog/pullUpMethod.html

Geschachtelte Bedingungen mit geschützten Klauseln ersetzen

Unübersichtlich, verschachtelte if-else-if.. Bedingungen

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

…ersetzen mit klarer, eindeutiger Auswertung pro Bedingung.

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

Quelle: http://refactoring.com/catalog/replaceNestedConditionalWithGuardClauses.html

Mehrere Bedingungen konsolidieren

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

Alle Bedingungen liefern den gleichen Wert wenn true

Methoden vereinen

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

Quelle: http://refactoring.com/catalog/consolidateConditionalExpression.html

Code-Duplikate in einen Ausdruck packen

Beide Fälle
Wenden gleiche
Methode an

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

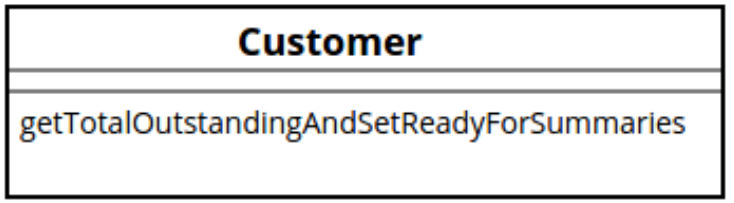
Methode nach
'außen'
verfrachten

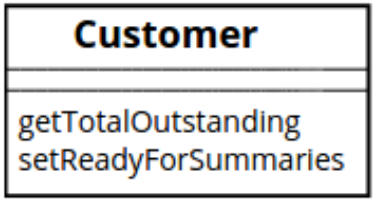
```
if (isSpecialDeal())
  total = price * 0.95;
else
  total = price * 0.98;
send();
```

Quelle: http://refactoring.com/catalog/consolidateDuplicateConditionalFragments.html

Abfrage und Modifizierung trennen

Methoden, die mehreren Berechnungen dienen



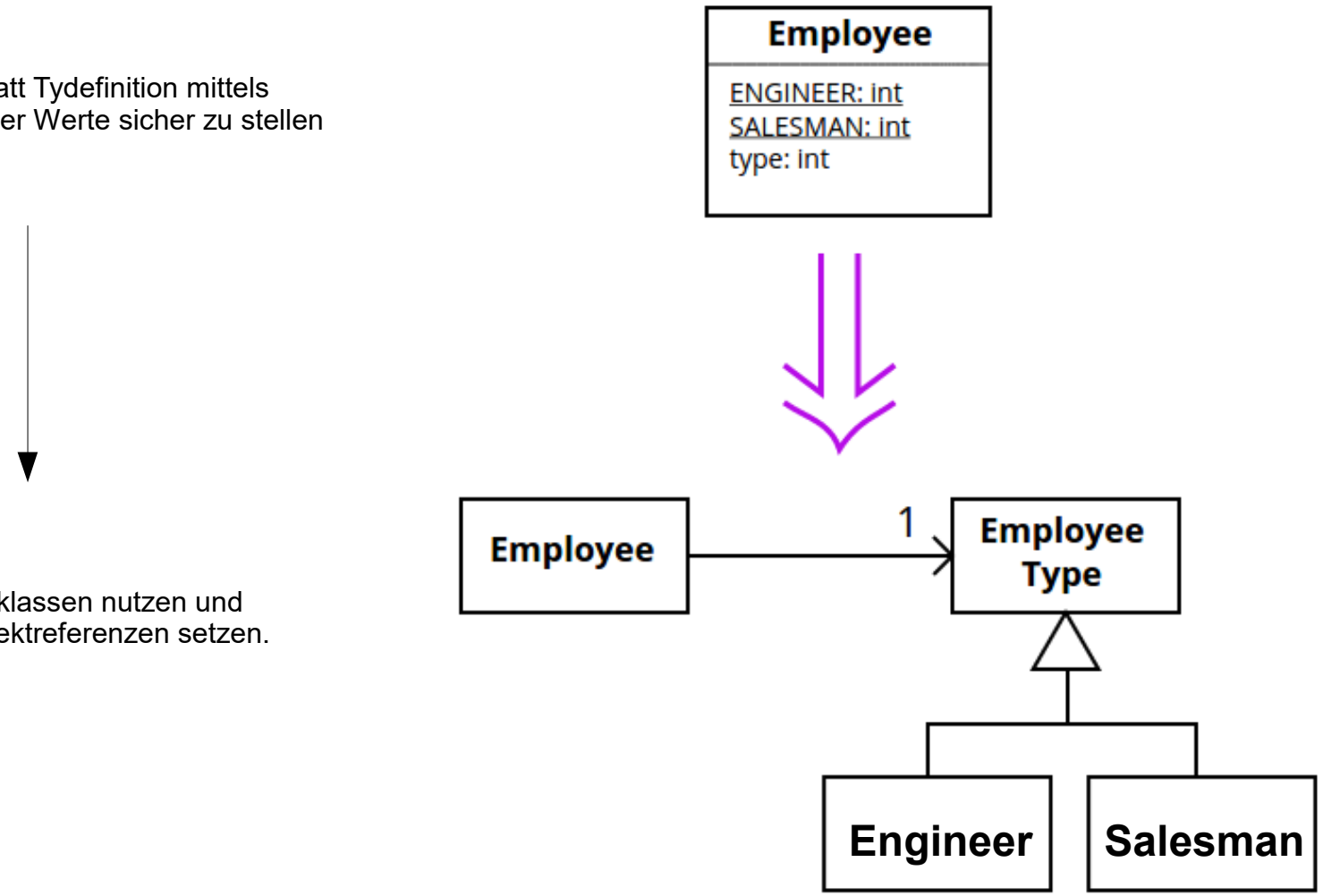


...aufspalten in separate Methoden
(u.a. auch mit kürzeren Bezeichnungen)

Quelle: http://refactoring.com/catalog/separateQueryFromModifier.html

Typ-Codierung mit Strategy ersetzen

Anstatt Tydefinition mittels
Integer Werte sicher zu stellen



Typklassen nutzen und
Objektreferenzen setzen.

Hinweis:
Fehler im original Bild:

Quelle: http://refactoring.com/catalog/replaceTypeCodeWithStateStrategy.html

Array mit Objekten ersetzen

Anstatt Informationen an festen Positionen in statischem Array abzulegen

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

… besser Klassenobjekte mit get/setter für feste Informationstypen verwenden

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

Quelle: http://refactoring.com/catalog/replaceArrayWithObject.html

Magische Nummern mit symbolischen Konstanten ersetzen

Wiederkehrende Zahlen

Verwendung von 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;
```

Quelle: http://refactoring.com/catalog/replaceMagicNumberWithSymbolicConstant.html

Konstructor-Zuweisungen nach oben ziehen

Initialisieren von geerbten Attributen

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

… kann an Constructor der Oberklasse weitergereicht werden.

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

Quelle: http://refactoring.com/catalog/pullUpConstructorBody.html

Bedingungen zerlegen

Aus komplizierten Bedingungen

… Methoden extrahieren und verwenden

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

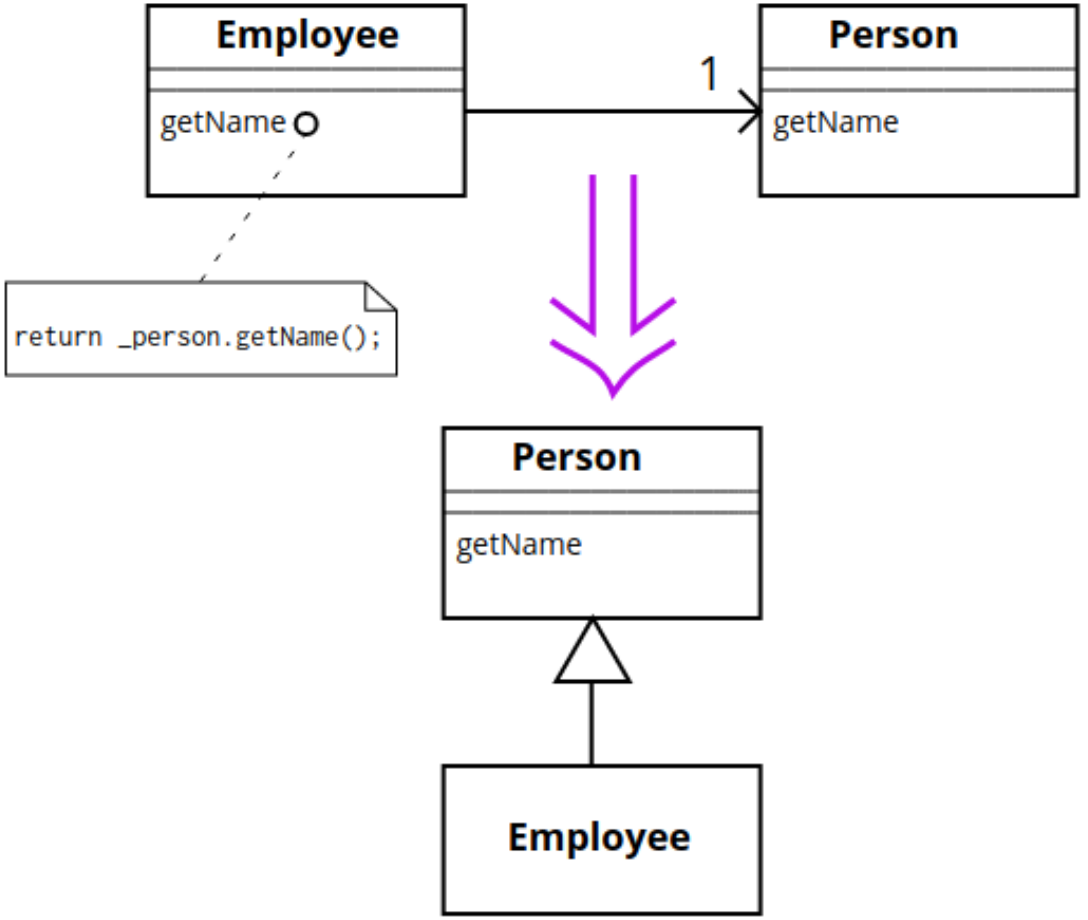
```
if (notSummer(date))
  charge = winterCharge(quantity);
else charge = summerCharge (quantity);
```

Quelle: http://refactoring.com/catalog/decomposeConditional.html

Delegation mit Vererbung ersetzen

Bei Verwendung vieler,
delegierenden Methoden

…besser Vererbung nutzen.

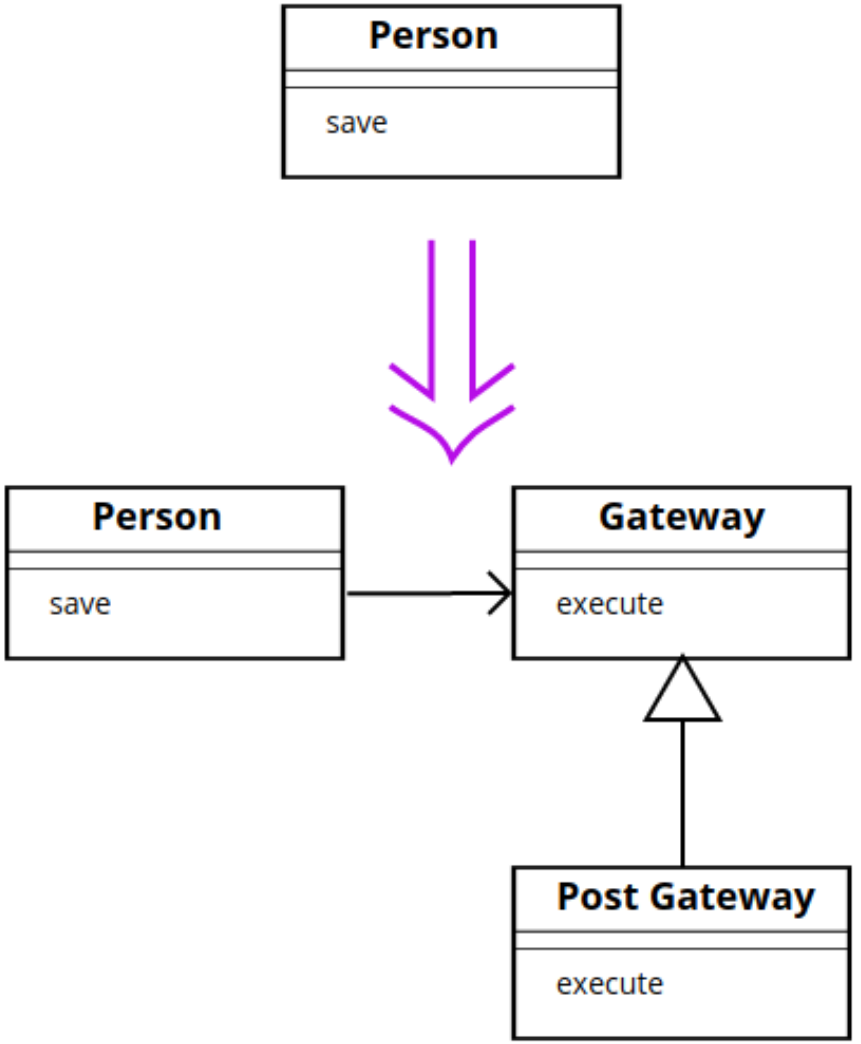


Quelle: http://refactoring.com/catalog/replaceDelegationWithInheritance.html

Gateways verwenden

Wenn komplexe APIs zu externen Systemen bzw. Resourcen verwendet werden müssen

… macht es Sinn, den Zugriff über eine Zwischenschicht bzw. Gateway zu kapseln



Quelle: http://refactoring.com/catalog/introduceGateway.html

Downcast kapseln

Wenn zu jeder klar ist, welchen spezifischen Rückgabetyp eine Methode hat

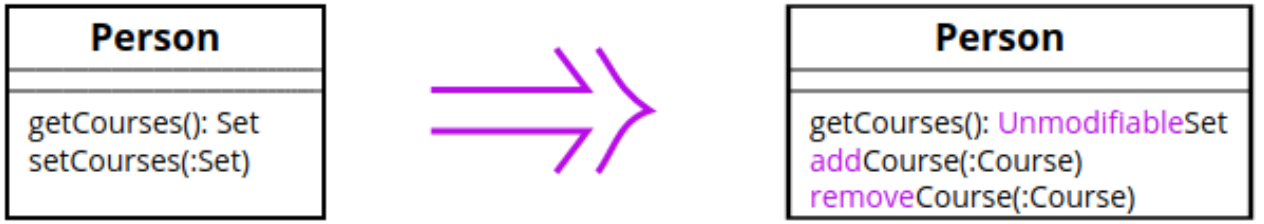
… sollte man ihn auch verwenden und den Downcast ggf. innerhalb der Methode ausführen.

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

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

Quelle: http://refactoring.com/catalog/encapsulateDowncast.html

Collection (Zugriff) kapseln

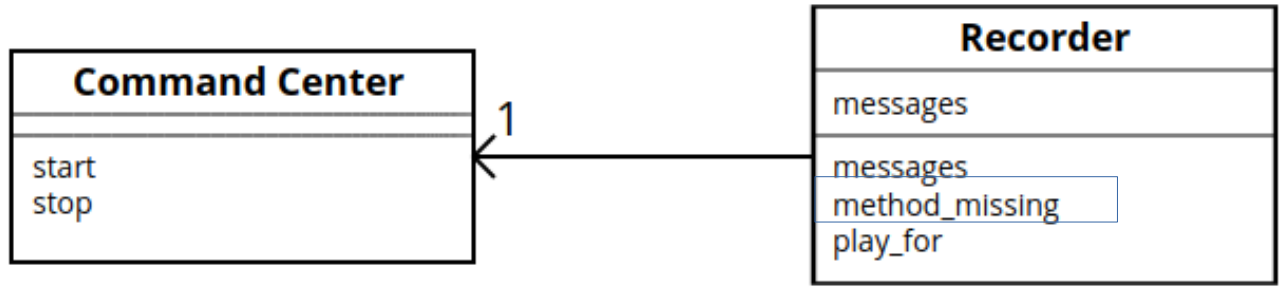


Um zu verhindern, dass eine Collection eines Objekts versehentlich verändert wird → … kann man die Collection als read-only definieren und separate Zugriffsmethoden anbieten

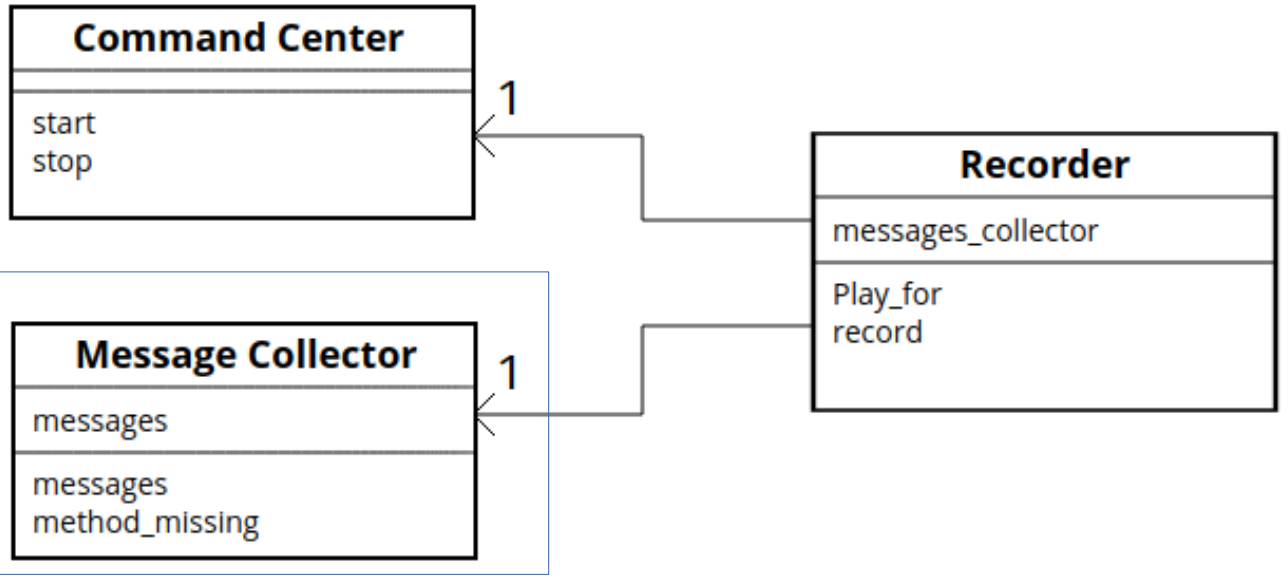
Quelle: http://refactoring.com/catalog/encapsulateCollection.html

Dynamische Rezeptoren isolieren

Wenn Klassen Methoden enthalten, die Berechnungen oder Analysen zu einem dynamischen Objekt dieser Klasse liefern



… kann man diese in eine separate Klasse packen, welche auschließlich dieser Analyse bzw. Berechnung gewidmet ist.



Quelle: http://refactoring.com/catalog/isolateDynamicReceptor.html

Weiterführende Literatur

- „Refactoring: Improving the Design of Existing Code“; Autoren: Martin Fowler; Verlag: Booch Jacobson Rumbaugh.
  - Alternativ sind alle Refactorings online: http://refactoring.com/catalog/
- Interessante Youtube Videos zu Refactorings:

  https://www.youtube.com/playlist?list=PLGLfVvz_LVvSuz6NuHAzpM52qKM6

  (Kanal von Derek Banas)