

---

# Von Object abwärts

---

# Object

- alle Klassen direkt oder indirekt von `Object` abgeleitet
- alle Klassen erben Methoden von `Object`:
  - `String toString()`: Umwandlung in Zeichenkette
  - `boolean equals(Object)`: Vergleich auf Gleichheit
  - `int hashCode()`: gleiche Objekte  $\Rightarrow$  gleiche Hashwerte
  - `Class getClass()`: Klasse des Objekts (= dyn. Typ)
  - `Object clone()`: Erzeugen einer Kopie
  - `void finalize()`: Aufräumen vor Speicherfreigabe
  - `wait, notify, notifyAll`: nebenläufige Programmierung

---

# Implizite Umwandlung in Zeichenkette

- ist ein Operand von + eine Zeichenkette, wird der andere Operand in eine Zeichenkette umgewandelt
- bei Referenztypen Umwandlung mittels toString:

```
String s = "Prüfung " + b + "! ";
```

entspricht

```
String s = "Prüfung " + b.toString() + "! ";
```

- pragmatisch: Überschreiben von toString nötig

---

# equals, hashCode und toString

folgende Bedingungen müssen gelten

- `a.equals(a)`
- `a.equals(b) ⇔ b.equals(a)`
- `a.equals(b) ⇒ a.hashCode() == b.hashCode()`
- `a.equals(b) ⇏ a.hashCode() == b.hashCode()`

Ebenso:

- `a.equals(b) ⇏ a.toString() == b.toString()`

Daher `equals` und `toString` nicht für Objektvergleiche geeignet

---

# Quellcode als Kommunikationsmedium

---

# Kommentare und Namen

- Programmierer kommunizieren über Kommentare
- Qualität der Kommentare entscheidend (nicht Umfang):
  - was man für Verwendung wissen muss
  - keine Beschreibung/Wiederholung der Syntax
- Namen unterstützen die Intuition
- Auffindbarkeit von Information von großer Bedeutung:
  - Faktorisierung soll intuitiv sein
  - wichtigste Informationen an Schnittstellen

---

## Wo Information zu finden ist

**Klasse, Interface:** Zweck, Grobstruktur, Besonderheiten

**Methoden, Konstruktoren:** Parameter, Verhalten, Ergebniswerte, Bedingungen für Aufruf

**Objektvariablen:** Zweck, Eigenschaften für Konsistenz

---

# Kommentare als Zusicherungen

**Vorbedingungen** auf Methoden und Konstruktoren:

was vor Aufruf gelten muss, hauptsächlich Parameterwerte

**Nachbedingungen** auf Methoden und Konstruktoren:

was nachher gilt; was gemacht und zurückgegeben wird

**Invarianten** auf Variablen, Klassen, Interfaces:

müssen in konsistenten Zuständen stets erfüllt sein

**History Constraints** auf Variablen, Klassen, Interfaces:

Entwicklung von Objekten im Laufe der Zeit



---

# Gute Faktorisierung

- alle zusammengehörigen Eigenschaften und Aspekte zu übersichtlichen Einheiten zusammengefasst
- Eigenschaften und Aspekte, die nichts miteinander zu tun haben, unabhängig voneinander änderbar
- Problem: Programmänderungen kaum voraussehbar
- Klassen-Zusammenhalt und Objekt-Kopplung helfen bei der Abschätzung

---

# Klassen-Zusammenhalt

- Grad des Zusammenhangs zwischen Klasseninhalten
- hoher Zusammenhalt = Hinweis auf gute Faktorisierung
- hoher Zusammenhalt erkennbar durch:
  - Variablen und Methoden passen gut zusammen
  - Namen und Kommentare treffend
  - Änderungen verringern Klassenzusammenhalt merklich
- gedankliche Abschätzung schon im Vorhinein möglich

---

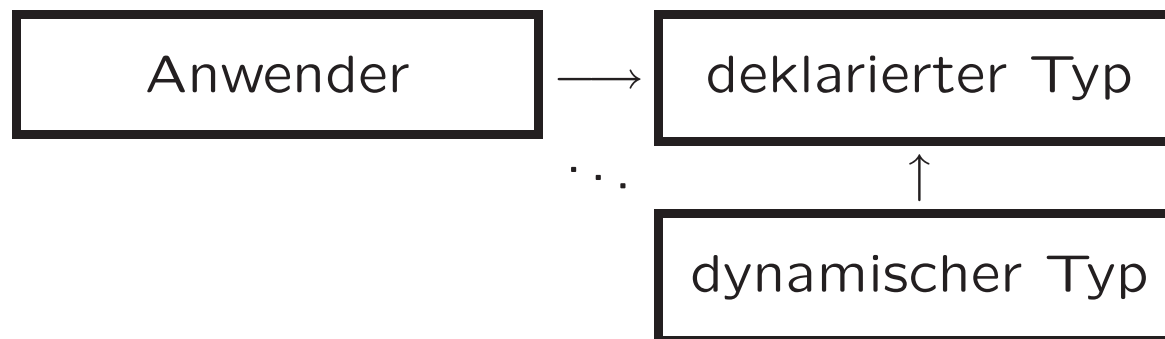
# Objekt-Kopplung

- Stärke der Abhängigkeiten der Objekte voneinander
- schwache Kopplung = Hinweis auf lokale Änderbarkeit
- schwache Kopplung erkennbar durch:
  - wenige nach außen sichtbare Methoden und Variablen
  - wenige Nachrichten an andere Objekte
  - geringe Anzahl an Parametern
- gedankliche Abschätzung schon im Vorhinein möglich

---

# Ersetzbarkeit und Verhalten

- Methoden in Untertypen müssen sich so verhalten, wie von Methoden in Obertypen erwartet
- Kommentare in Untertypen spezifizieren dasselbe Verhalten genauer als in Obertypen
- Ersetzbarkeit entkoppelt Programmteile:



- Ableitung von stabilen Typen (weit oben in Typhierarchie)