Sascha 0graeff

SE2-Paniktutorium

WAAAAAAAAAAAAAAAHHHHHHH

Vertragsmodell

- Dienstleister und Klient haben Vertrag.
- Vertrag steht in der Dienstleister-Klasse.
- Vorbedingungen und Nachbedingungen.
- Wenn Vorbedingungen erfüllt sind, muss der Dienstleister die Nachbedingungen erfüllen.
- Wenn nicht, darf die Methode nicht aufgerufen werden. Wenn doch => Shitstorm

Vertragsmodell

Zusammenfassung:

- Dienstleister legt Vertrag für eine Methode fest.
- Alle Klienten haben sich daran zu halten.
- Dann hält sich auch der Dienstleister daran.

Vertragsmodell

In Java:

- `@require` [Vorbedingung]
- `@ensure` [Nachbedingung]

Vertragsmodell

- Können Vorbedingungen verschärft werden? Siehe Subtyping.

Programmierfehler

Zeller:

- *defect* = Programmierfehler
- *infection* = Fehler im Zustand
  - durch ausgeführten Code mit *defect*
- *failure* = Sichtbar gewordene Infection

Programmierfehler

Zeller:

- *defects* können beliebig lange unentdeckt bleiben.
- *infections* auch, wenn wir nicht darauf testen.

Programmierfehler

- …können im Programm selbst z.B. durch Exceptions behandelt werden.
- Gleiches gilt für Umgebungsfehler.

Exceptions

- …gibt es, damit Fehlerbehandlung und „Normalfall-Code“ nicht gleich aussehen.
- …bieten Fehlerfallunterscheidung.

Java-Exceptions

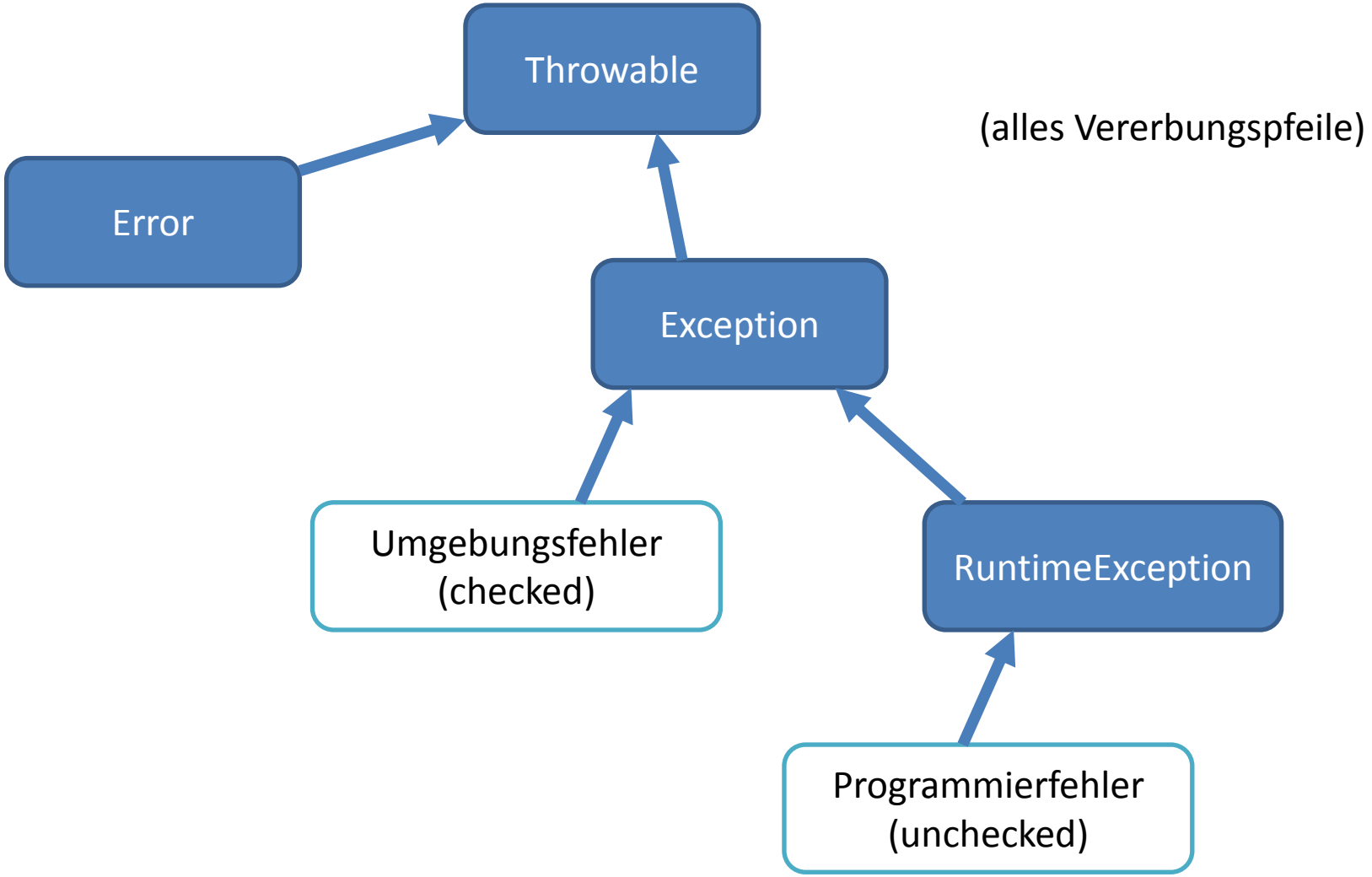
- Geprüft (Checked): Muss in try-catch-Block behandelt werden
- Ungeprüft (Unchecked): Kann einfach blind weitergeleitet werden
  - Wenn nirgendwo gefangen, wird eine ungeprüfte Exception in die Konsole geschmettert.
    + Programmabbruch

Java-Exceptions

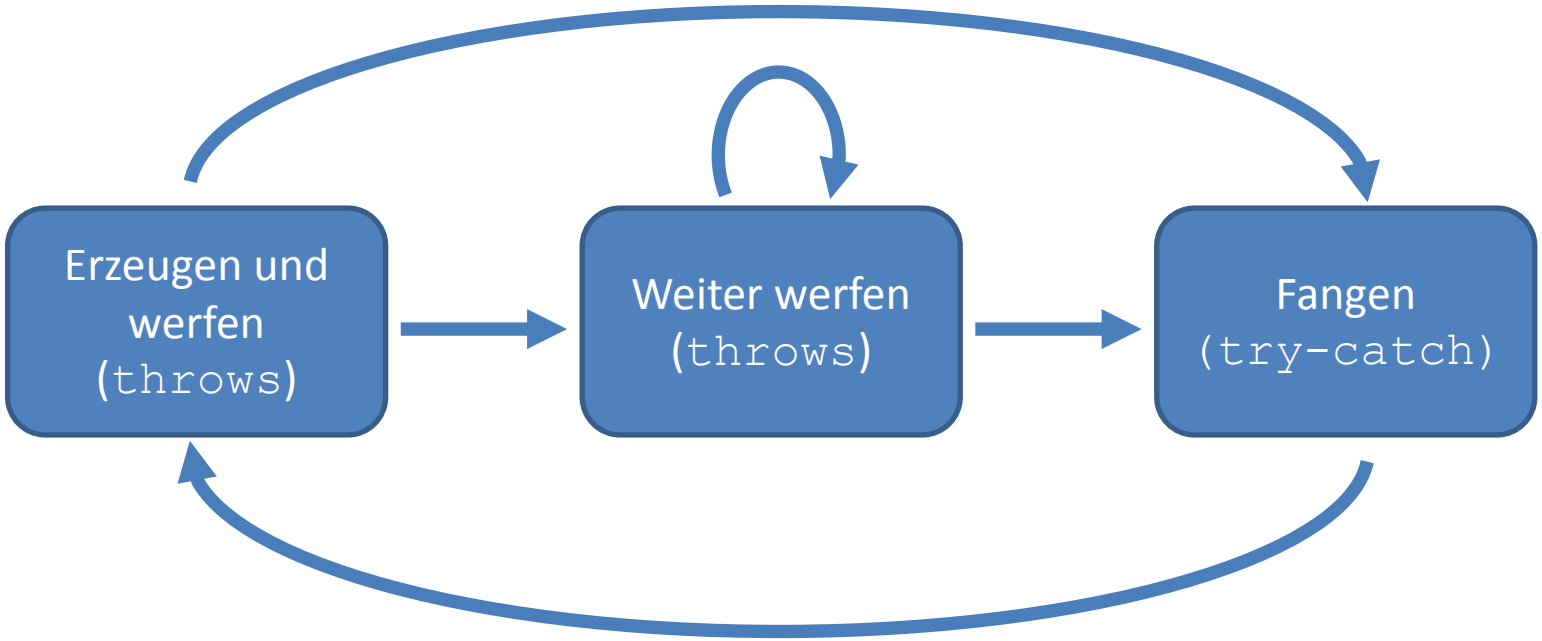
- Geprüft (Checked): **Umgebungsfehler**
- Ungeprüft (Unchecked): **Programmierfehler**

Java-Exceptions

- …sind Exemplare **einer** Exception-Klasse.



Java-Exceptions



Subtyping, dynamisches Binden

- Subtyp: Klasse oder Interface, das ein Interface implementiert und/oder von einer Klasse erbt
- Supertyp: Interface oder Klasse von der geerbt bzw. welche implemtentiert wird

Subtyping, dynamisches Binden

- Exemplare eines Subtypen können überall dort eingesetzt werden, wo ein Exemplar eines Supertyps erwartet wird.

  ```
  Konto k = new Sparkonto();
  ```
  wenn
  ```
  Sparkonto extends Konto
  ```

Subtyping, dynamisches Binden

- Exemplare eines Subtypen können überall dort eingesetzt werden, wo ein Exemplar eines Supertyps erwartet wird.

  ```
  Konto k = new Sparkonto();
  ```
  wenn
  ```
  Sparkonto implements Konto
  ```

Subtyping, dynamisches Binden

- Exemplare eines Subtypen können überall dort eingesetzt werden, wo ein Exemplar eines Supertyps erwartet wird.

```
Konto k = new Sparkonto();
```

wenn

```
Sparkonto extends Foo
Foo extends Konto
```

Subtyping, dynamisches Binden

- Redefinieren: In Subtyp neue Methode für gleiche Operation definieren
- Redeklarieren: Neue Signatur
  + Änderung der Schnittstelle
  (Änderung von Parametern)

Subtyping, dynamisches Binden

- Ein Subtyp hat immer gleich viele oder mehr Operationen als der Supertyp.
- Zum Vertragsmodell:
  - Vorbedingungen verschärfen/entschärfen?
  - Nachbedingungen erweitern/verringern?

Subtyping, dynamisches Binden

- Ein Subtyp hat immer gleich viele oder mehr Operationen als der Supertyp.
- Zum Vertragsmodell:
  - Vorbedingungen verschärfen/entschärfen?
  - Nachbedingungen erweitern/verringern?

Subtyping, dynamisches Binden

- Ein Subtyp hat immer gleich viele oder mehr Operationen als der Supertyp.
- Zum Vertragsmodell:
  - Vorbedingungen verschärfen/entschärfen?
  - Nachbedingungen erweitern/verringern?

  Der Klient will weder plötzlich mehr leiten müssen, noch weniger dafür erhalten!

Subtyping, dynamisches Binden

- Was ist dynamisches Binden? Begriffe:
  - Methode
  - Übersetzungszeit
  - Laufzeit
  - Dynamischer Typ

Subtyping in Java

- Alle Klassen erben von `Object`,
  direkt oder indirekt.
- Damit erbt jede Klasse direkt von genau einer anderen Klasse (außer `Object`).
- Interfaces können aber beliebig viele implementiert werden.
- Ein Interface kann auch von beliebig vielen Interfaces erben!

Subtyping in Java

- Implementiert man mehrere Interfaces, oder erbt von ihnen, werden deren Operationen vereinigt.

Subtyping in Java

- `static` und Subtyping?
  => Kein dynamisches Binden!!!

Subtyping in Java

```
public interface A {
  void m(int a);
  void m2(String s);
}

public interface B {
  void m(boolean b);
  void m2(String s);
}

public interface C extends A,B {}
```

Subtyping in Java

```
public interface A {
   void m(int a);
   void m2(String s);
}

public interface B {
   void m(boolean b);
   void m2(String s);
}

public interface C extends A,B {}
```

- Das Interface C bietet **zwei** Operationen mit dem Namen m an
- Das Interface C bietet **eine** Operation mit dem Namen m an
- Das Interface C bietet **keine** Operationen an

- Das Interface **C** definiert einen **Subtyp** des Interfaces **A**
- Das Interface **B** definiert einen **Subtyp** des Interfaces **A**

- Das Interface C bietet eine Operation mit dem Namen **m2** an

Subtyping in Java

```
public interface A {
   void m(int a);
   void m2(String s);
}

public interface B {
   void m(boolean b);
   void m2(String s);
}

public interface C extends A,B {}
```

- Das Interface C bietet **zwei** Operationen mit dem Namen m an
- Das Interface C bietet **eine** Operation mit dem Namen m an
- Das Interface C bietet **keine** Operationen an

- Das Interface **C** definiert einen **Subtyp** des Interfaces **A**
- Das Interface **B** definiert einen **Subtyp** des Interfaces **A**

- Das Interface C bietet eine Operation mit dem Namen **m2** an

Generizität

- Eine formaler generischer Typparameter ist eine Variable, die einen konkreten Typ als Wert aufnehmen kann.
- `List<Typ>`

Generizität

- Ist `List<Subtyp>` ein Subtyp von `List<Supertyp>`?

Generizität

- Ist `List<Subtyp>` ein Subtyp von `List<Supertyp>`?
- Nein!

```
List<Subtyp> sub = new ArrayList<Subtyp>();
List<Supertyp> super = sub;

super.add(new Supertyp());
Subtyp ding = sub.get(0);
```

- Supertyp wird auf Subtyp gecastet…?

abstract

- Von abstrakten Klassen können keine Exemplare erstellt werden.
- Eine abstrakte Methode darf keinen Rumpf haben.
- Hat eine Klasse eine abstrakte Methode, muss sie als abstract deklariert werden.
- Eine abstrakte Klasse braucht aber keine abstrakte Methode.

final

- Eine als final deklarierte Klasse kann nicht als Supertyp benutzt werden (man kann nicht von ihr erben).
- Eine als final deklarierte Methode kann nicht in Subtypen redeklariert werden.
- Eine als final deklarierte Variable kann höchstens im Konstruktor der Klasse oder bei der Deklaration einen Wert zugewiesen bekommen. Das erübrigt sich bei Klassenkonstanten.

final

- Achtung: Mögliche Fallen mit private!
  - Nur weil eine Methode nicht final ist, heißt das nicht, dass sie redefiniert werden kann…
  - Eine Subklasse muss nicht alle Exemplarvariablen aller Oberklassen übernehmen…

GUI

- Listener-Interfaces werden benutzt, um GUI-Elementen anwendungsspezifisches Verhalten zu geben.
- Beliebig viele (verschiedene und gleiche) Listener können auf eine GUI-Komponente angemeldet werden.
- Layouts in AWT/Swing können ineinander geschachtelt werden, um komplexe Layouts zu erstellen.

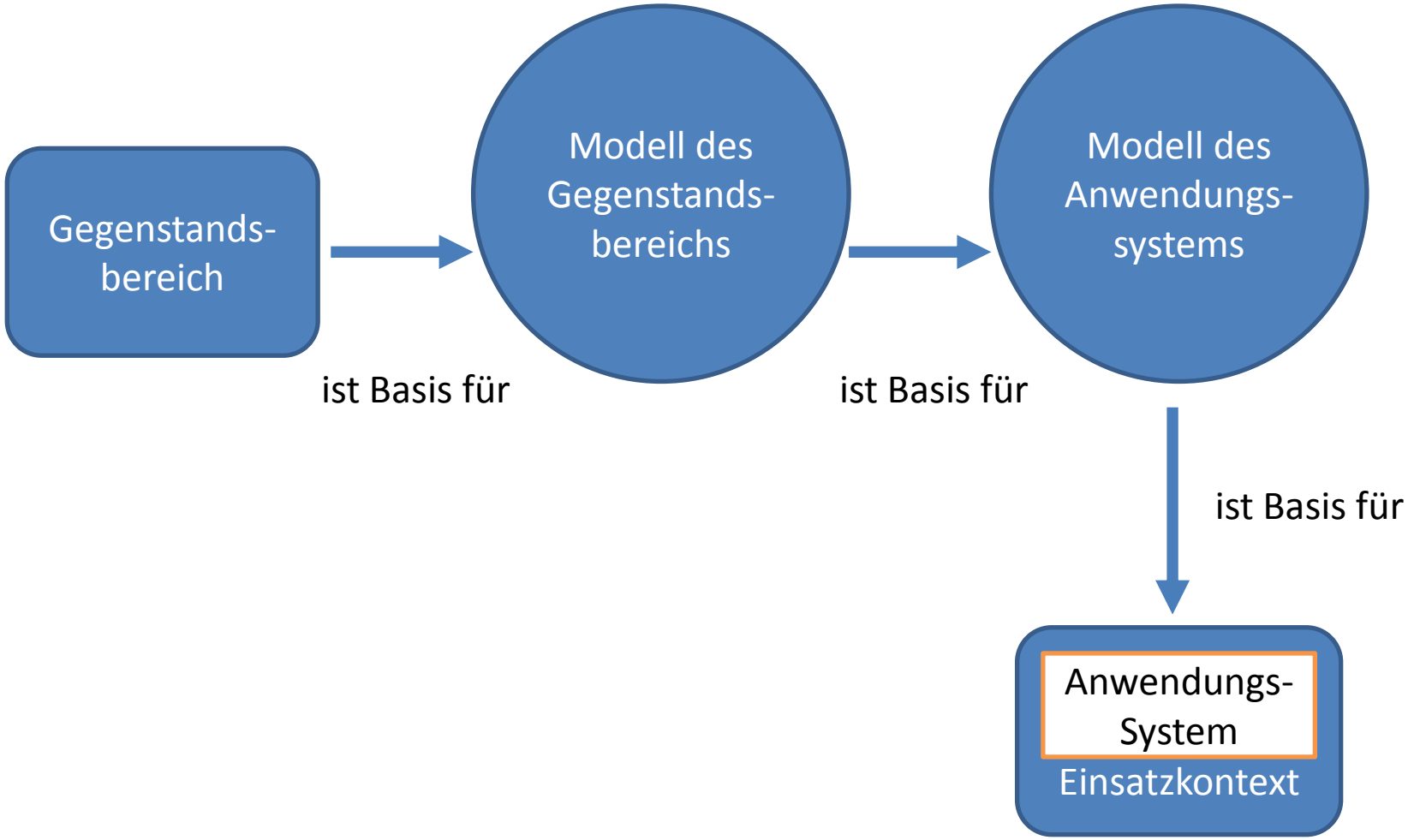
GUI

- Swing-Komponenten basieren teilweise auf AWT-Komponenten.

Modell

- abstrahierte Abbildung des Originals
- lässt subjektive Deutungen zu
- ist pragmatisch (zu einem bestimmten Zweck erstellt)
- ermöglicht Probehandeln
- spiegelt die (gerade interessanten) Zusammenhänge der Realität wieder

Modelle


Gegenstands-
bereich
ist Basis für
Modell des
Gegenstands-
bereichs
ist Basis für
Modell des
Anwendungs-
systems
ist Basis für
Anwendungs-
System
Einsatzkontext

WAM-Ansatz

- *Materialien* werden im Rahmen einer Aufgabe zum Teil des Arbeitsergebnisses.
- Sie werden mit den richtigen *Werkzeugen* unter bestimmten Aspekten bearbeitet.

Werte

- Werte sind zeitlos.
  - …und können daher nie erzeugt werden.
- Werte sind abstrakt.
- Werte sind unveränderlich.
- Werte können unterschiedlich repräsentiert werden (z.B. Zahlensysteme bei Zahlen)

Objekte (im Vergleich zu Werten)

- Objekte behalten auch bei Veränderung ihre Identität.
- Objekte können erzeugt und zerstört werden.
- Zwei gleiche Objekte müssen nicht das selbe sein.

Zugriffsschutz in Java

- Ein Paket legt einen gemeinsamen Namensraum für enthaltene Klassen und Interfaces fest.
- Zugriffsrechte von Klassen und Interfaces werden über `public`, `protected`, `private` oder `package private` (default) definiert.
- Mehr dazu?

Entwurfsmuster

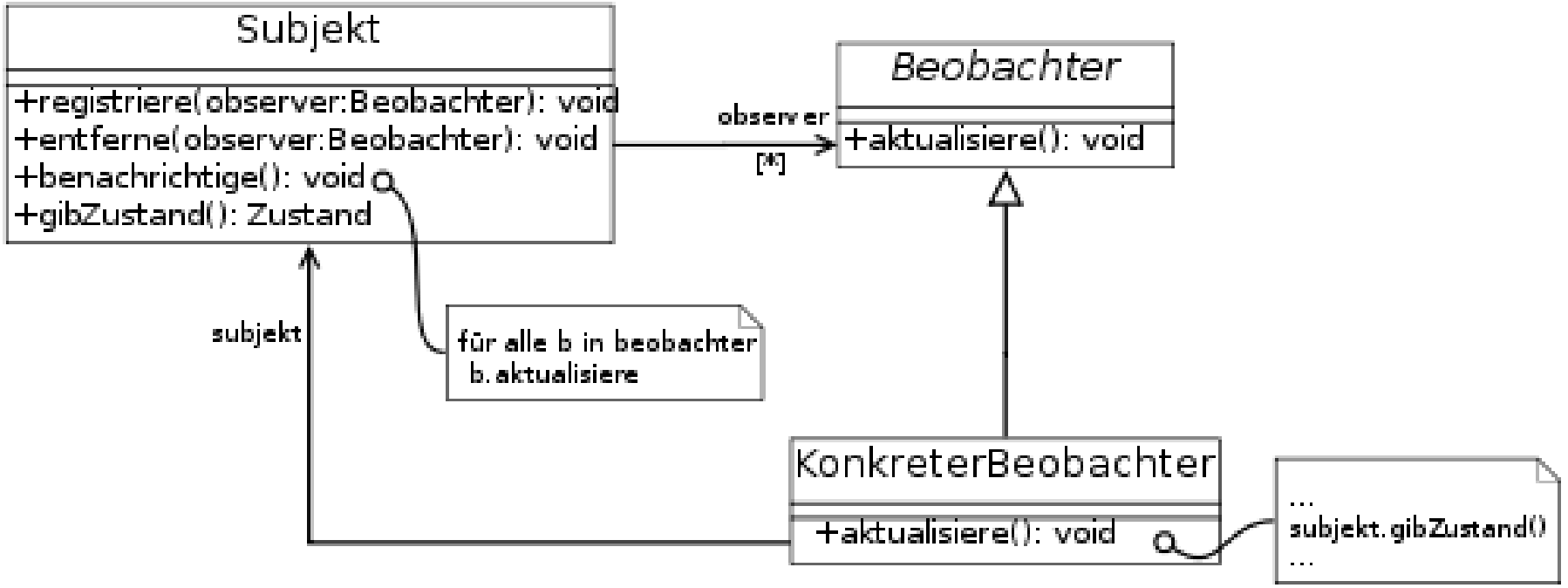
- …werden nicht erfunden, sondern entdeckt!
- …sollen immer mit dem Problem, das sie lösen sollen, beschrieben werden.
- …werden aufgeteilt in:
  - Erzeugungsmuster
  - Strukturmuster
  - Verhaltensmuster

Entwurfsmuster

- …in diesen drei Kategorien gibt es
  - Klassenmuster (strukturell, zur Übersetzungszeit)
  - Objektmuster (zur Laufzeit sichtbar)

Beobachtermuster

Zum Entfernen statischer zyklischer Beziehungen zwischen zwei Klassen



Refactoring

- …beschreibt Vorgehensweise zur Verbesserung des Codes.
- …ändert nur interne Dinge, niemals Aussehen der GUI oder Verhalten des Programms!

Tests

- Ein Klassentest testet die Operationen einer einzelnen Klasse.
- Integrationstests testen das Zusammenspiel mehrerer Komponenten.
- Tests können keine Fehlerfreiheit nachweisen.

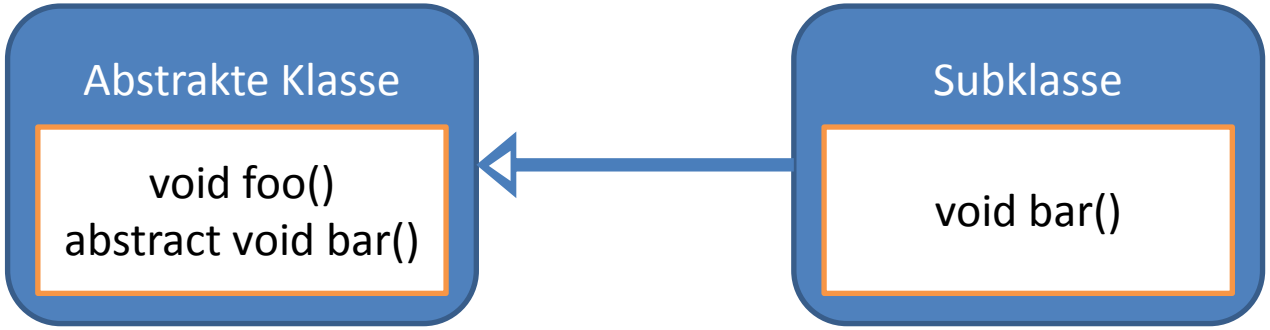
Anonyme innere Klassen

- Exemplare, die bei ihrer Erzeugung Operationen überschreiben, haben als Typ eine neue Klasse.
- Diese Klasse ist anonym, da sie nirgendwo einen Namen erhält
  (wozu auch, wird ja nur ein Mal instanziiert).

Anonyme innere Klassen

```
button1.AddListener(new ActionListener()
   {
      @override
      public void actionPerformed(Parameter)
      {
            Code
      }
   }
);
```

Schablonen- und Einschubmethode



foo ruft bar auf

foo ist eine Schablonenmethode (unvollständig, da bar fehlt)
bar ist eine Einschubmethode (vervollständigt foo, sobald sie implementiert wird)