

Softwareentwicklung II (IB)

Abstrakte Basisklassen

Prof. Dr. Oliver Braun

Fakultät für Informatik und Mathematik
Hochschule München

Letzte Änderung: 24.06.2018 15:20

Inhaltsverzeichnis

Idee	1
Beispiel: ABC Zähler	2
Ableiten einer ABC	2
Vorteile einer ABC	3
Einfache und mehrfache Vererbung	3
Implementieren mehrerer Interfaces	3
Unverträgliche Interfaces	4
Ableiten und Implementieren von Interfaces	4
Blockieren der Vererbung	5

Idee

- Gegensätze:
 - Interfaces** ausschließlich Methodenköpfe, keine Methodenrumpfe, keine Konstruktoren, keinen Objektvariablen
 - Konkrete Basisklassen** vollständig mit Methodenrumpfen, Konstruktoren, Objektvariablen
- Mittelweg: **Abstrakte Basisklassen** (abstract base classes, ABCs)
- Definition mit Modifier **abstract**

```
abstract class Name  
{...}
```

- Methoden in einer ABC wahlweise ...
konkret: mit Rumpf (wie in konkrete Klassen), oder
abstrakt: nur Signatur (wie bei Interfaces), statt Rumpf nur „;“
- Methoden ohne Rümpfe zusätzlich mit Modifier **abstract**

```
abstract class Name {  
    void foo()           // konkrete Methode  
    {...}  
  
    abstract void bar(); // abstrakte Methode  
}
```

Beispiel: ABC Zähler

- Beispiel: ABC Counter für Zähler

```
abstract class Counter { // als ABC markiert  
    private int count = 0;  
    void reset() {  
        count = 0;  
    }  
    int read() {  
        return count;  
    }  
    abstract void step(); // nur Signatur  
}
```

- Methode **step** **abstrakt** = ohne Implementierung
- Methoden **read**, **reset** konkret

Ableiten einer ABC

- ABC kann nicht instanziiert werden (unvollständig, wie ein Interface)
- Einziger Zweck: Ableiten
- Abgeleitete Klassen müssen die fehlenden (abstrakten) Methoden der ABC implementieren
- Wenn nicht oder nicht vollzählig: abgeleitete Klasse selbst ABC, muss noch weiter abgeleitet werden
- Beispiel: **OpenCounter** abgeleitet von **ABC Counter**:

```

class OpenCounter extends Counter {
    void step() {
        count++;
    }
}

```

OpenCounter ist konkret, keine ABC: Liefert Definition der einzigen abstrakten Methode der ABC

Vorteile einer ABC

- ABCs flexibler als Interfaces:

	Interface	ABC
Signaturen	nur <code>public</code>	ohne Einschränkung
Methoden	keine	ohne Einschränkung
Objektvariablen	keine	ohne Einschränkung
Klassenvariablen	nur <code>public static final</code>	ohne Einschränkung
Konstruktoren	keine	für abgeleitete Klassen (<code>super</code>), oft <code>protected</code>
Ableitung	von Interfaces	ohne Einschränkung

Einfache und mehrfache Vererbung

- ABCs mit ausschließlich abstrakten Methoden = **rein abstrakte Basisklasse** (pure ABC)
- Konzeptionell ähnlich zu Interfaces
- Aber: kein Ersatz für Interfaces
- Eine Klasse kann ...
 - ... von einer Basisklasse erben (konkret, abstrakt oder pure abstract)
 - ... beliebig viele Interfaces implementieren
- **Einfache Vererbung** = maximal eine Basisklasse
- Keine mehrfache Vererbung = zwei oder mehr Basisklassen

Implementieren mehrerer Interfaces

- Einfache Vererbung betrifft Basisklassen, nicht Interfaces
- Implementierung mehrerer Interfaces zulässig

```

interface Numbered {
    int getNumber();
}
interface Counted {
    int getCount();
}
class Thing implements Numbered, Counted {
    public int getNumber() {...} // für Numbered
    public int getCount() {...} // für Counted
}

```

- Klasse ist kompatibel zu allen implementierten Interfaces

```

Thing t = new Thing;
Numbered n = t;           // ok, kompatibel
Counted c = t;           // ok, kompatibel

```

- Gleiche Methoden in mehreren Interfaces: einmal implementieren, alle Interfaces bedient

```

interface Numbered {
    int getNumber();
}
interface Serial {
    int getNumber();           // wie in Numbered
}
class Thing implements Numbered, Serial {
    public int getNumber() {...} // für Numbered und Serial
}

```

Unverträgliche Interfaces

- Methoden mit widersprüchlichen Ergebnistypen in verschiedenen Interfaces

```

interface IntNumbered {
    int getNumber();
}
interface DoubleNumbered {
    double getNumber();
}

```

- Können nicht beide implementiert werden:

```

class Thing implements
    IntNumbered,    // verlangt int getNumber()
    DoubleNumbered // verlangt double getNumber() - Fehler!
    {...}

```

Ableiten und Implementieren von Interfaces

- Einfache Vererbung kombinierbar mit Implementieren von Interfaces

```
class OpenCounterDeluxe extends OpenCounter
    implements Numbered, Serial {
    public int getNumber() // für Numbered, Serial
    {...}
}
```

- Klasse ist kompatibel zu allen implementierten Interfaces und zur Basisklasse

```
OpenCounterDeluxe d = new OpenCounterDeluxe();
OpenCounter o = d;           // ok, kompatibel
Numbered n = d;             // ok, kompatibel
Serial s = d;               // ok, kompatibel
```

- Fehler falls Interface und Basisklasse inkompatibel

```
interface Stepper {
    int step();
}
```

```
class OpenCounterDeluxe
    extends OpenCounter // vererbt void step()
    implements Stepper // verlangt int step() - Fehler!
{...}
```

Blockieren der Vererbung

- In seltenen Fällen sinnvoll: Aktives Verhindern der Ableitung
- Modifier `final` der ganzen Klasse erlaubt keine abgeleiteten Klassen mehr

```
// kann nicht abgeleitet werden
final class FinalLastWords {...}
```

- Populäres Beispiel: Klasse `String`

```
class SuperString extends String {...} // Fehler!
```

- Feinere Dosierung: Modifier `final` verhindert Redefinition einer einzelnen Methode

```
class Rational {
    // das letzte Wort zur Addition
    final Rational add(Rational r)
    {...}
}
```

- final-Klasse beendet Folge von Ableitungen

```
class Base {...}
```

```
// ok
```

```
final class Child extends Base {...}
```

```
// Fehler! Ableitung nicht mehr zulässig
```

```
class GrandChild extends Child {...}
```

Entsprechend: final-Methode beendet Folge von Redefinitionen