Objektorientierte Programmierung

WS 2006/2007

Klassen und Vererbung in Java

Walter Binder
Universität Lugano

Überblick

- Konstruktoren
- Verdecken versus Überschreiben
- Final
- Pakete
- Sichtbarkeit
- Interfaces
- Anwendungsbeispiele

Konstruktoren (1)

```
class Circle {
  int r;
  Circle(int r) { this.r = r; }                          // 1
  Circle(Circle c) { r = c.r; }                          // 2
  Circle() { this(1); }                                  // 3
  …
}

class ColorCircle extends Circle {
  Color c;
  ColorCircle(int r, Color _c) { super(r); c = _c; } // 4
  …
}

Circle a = new Circle(2);                        // Konstr. 1
Circle b = new Circle(a);                        // Konstr. 2
Circle c = new Circle();                         // Konstr. 3
Circle d = new ColorCircle(3, someColor);        // Konstr. 4
```

Konstruktoren (2)

- **Default Konstruktor**, falls kein expliziter Konstr.:

```
class X {
   // public X() { }
   …
}
```

- **`super(…)`** – führt Konstr. der Oberklasse aus
- **`this(…)`** – führt anderen Konstr. derselben Klasse aus
- Konstr. beginnt weder mit **`super(…)`** noch mit **`this(…)`**: **`super()`** implizit am Anfang des Konstruktors eingefügt

Verdecken versus Überschreiben

- Variable gleichen Namens in Ober- und Unterklasse:
  - Variable in Unterklasse **verdeckt** Variable in Oberklasse
  - Verdeckte Variable zugreifbar:

    `super.`*`var`*

    `((`*`Oberklasse`*`)this).`*`var`*
- Methode gleicher Signatur in Ober- und Unterklasse:
  - Unterklassenmethode **überschreibt** Oberklassenmethode
  - Überschriebene Methode zugreifbar: **`super.`*`method`*`(…)`**
  - **Kein Zugriff**: **`((`*`Oberklasse`*`)this).`*`method`*`(…)`**

Final

- Konstanten
  - Klassenkonstante: Zuweisung im Static Initializer
  - Instanzkonstante: Zuweisung im Konstruktor
- Methoden
  - Können nicht überschrieben werden
- Klassen
  - Keine Unterklassen möglich

Pakete

- Eine **`public`** Klasse pro Datei
- Paket: Alle Dateien bzw. Klassen im selben Ordner
- Explizite Paketdeklaration: **`package pName;`**
- Beispiel: Datei **`myclasses/test/AClass.java`**

```
package myclasses.test;
public class AClass {
   public static void f() { }
}
```

- Aufruf von **`AClass.f()`** in anderen Paketen:
  - `myclasses.test.AClass.f();`
  - `import myclasses.test;` `test.AClass.f();`
  - `import myclasses.test.AClass;` `AClass.f();`
  - `import myclasses.test.*;` `AClass.f();`

Sichtbarkeit

	public	**protected**	**default**	**private**
lokal sichtbar	ja	ja	ja	nein
global sichtbar	ja	nein	nein	nein
lokal “vererbbar”	ja	ja	ja	nein
global “vererbbar”	ja	ja	nein	nein

lokal: im selben Paket, auch außerhalb der Klasse

global: auch außerhalb des Pakets

“vererbbar”: bezieht sich auf Sichtbarkeit bei Vererbung

Anwendung der Sichtbarkeit

- **Public**: Methoden, Konstanten, (Variablen), die bei Verwendung der Klasse benötigt werden
- **Protected**: Hilfreich bei Klassenerweiterung
- **Default**: Eng zusammenarbeitende Klassen im selben Paket
- **Private**: Internals

Interfaces

```
interface X {
   static final double PI = 3.14159d;
   double fooX();
}

interface Y {
   double fooY();
}

interface Z extends X, Y {
   double fooZ();
}
```

Unterschiede zu abstrakten Klassen

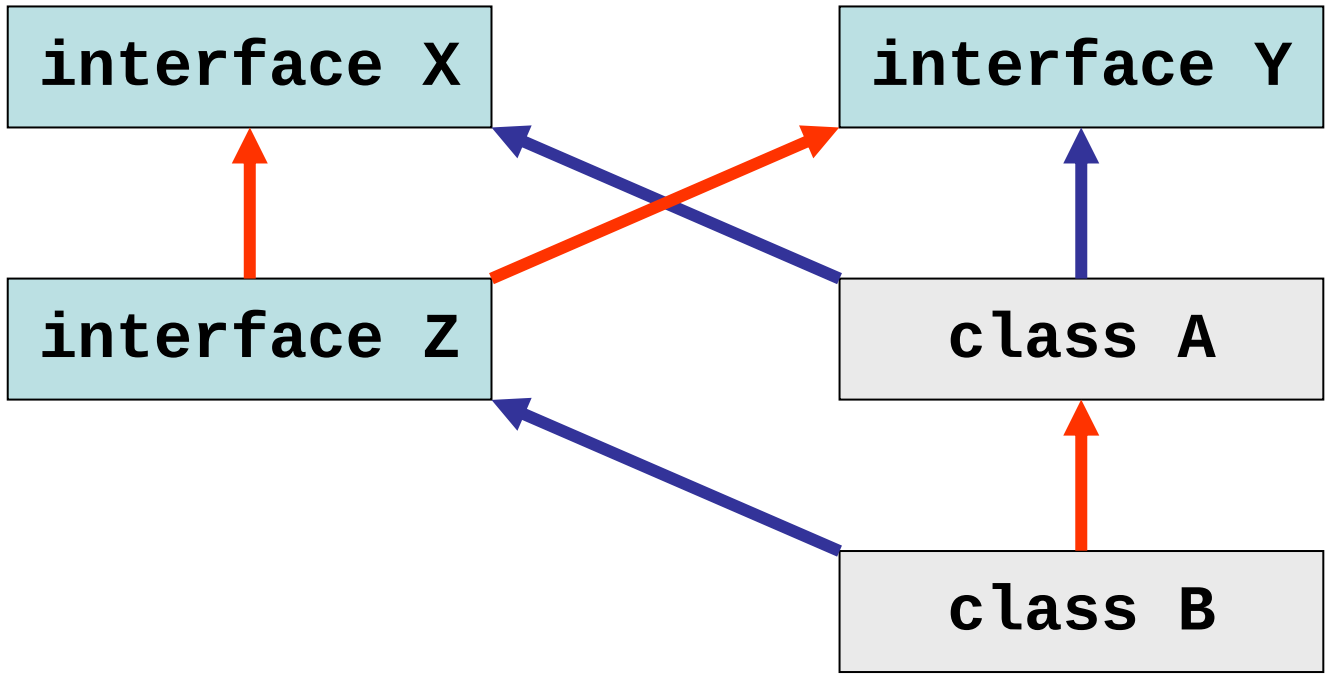
- Schlüsselwort **`interface`** statt **`abstract class`**
- Alles implizit **`public`**
- Alle Methoden implizit **`abstract`**
- Keine Variablen, nur Konstanten
- **`static`** und **`final`** in Methodendeklarationen verboten
- Mehrfachvererbung

Klassen implementieren Interfaces

```
class A implements X, Y {
   double factor = 2.0d;
   public double foo()  { return PI; }
   public double fooX() { return factor * PI; }
   public double fooY() { return factor * fooX(); }
}

class B extends A implements Z {
   public double fooY() { return 3.3d * foo(); }
   public double fooZ() { return factor / fooX(); }
}
```

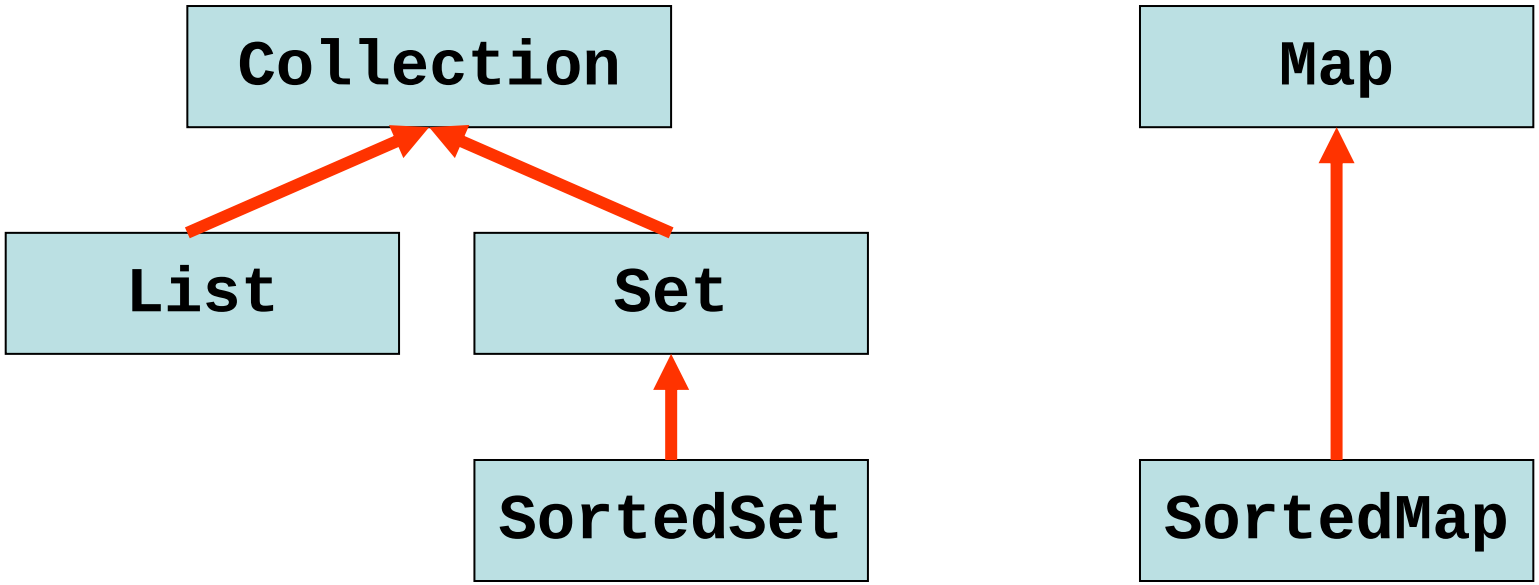
Untertyprelationen



Anwendung von Interfaces

- API Design
- Callbacks
  - Client Code wird bzgl. Basisklasse nicht eingeschränkt
- Libraries möglichst über Interfaces verwenden
- Austausch von Implementierungen einfach möglich

Beispiel 1: Collection Framework



Interface	Implementierung (Class)			
Set	HashSet		TreeSet	
List		ArrayList		LinkedList
Map	HashMap		TreeMap	

Beispiel 2: Locking Framework

```
public interface Lock {
    void lock();
    void unlock();
    …
}
public interface ReadWriteLock {
    Lock readLock();
    Lock writeLock();
}

public class ReentrantLock implements Lock {…}
public class ReentrantReadWriteLock
             implements ReadWriteLock {…}
```

Beispiel 3: RMI