Einführung in die Objektorientierte Programmierung
Vorlesung 10: Zugriffskontrolle

Sebastian Küpper

Namenskapselung durch Pakete

- Klassen können in Java in Paketen gebündelt werden, Schlüsselwort `package`
- Namen, die in Paketen vergeben werden, sind nur innerhalb der Pakete reserviert
- Zugriff auf Klassen anderer Pakete durch Einbinden der Pakete, Schlüsselwort `import`

Beispiel Definition eines Pakets, importieren einer Klasse eines Pakets

```
package kunde;
import waren.Artikel;
import waren.ArtikelListe;
class Bestellung {
   // ...
}
```

Falls man alle Klassen eines Pakets benötigt...

```
package kunde;
import waren.*;
class Bestellung {
   // ...
}
```

Achtung: Unterpakete müssen separat importiert werden

```
import a.*;
import a.b.*; // Diese Zeile ist nicht redundant
```

Wozu Zugriffsbeschränkungen?

- Kontrolle des eigenen Zustandes (Kohärenz)
- (Größere) Unabhängigkeit von Implementierung
- Übersichtlichkeit des Interfaces und der Dokumentation
- Sicherstellen all dieser Punkte bereits zum Kompilationszeitpunkt

Welche Sichtbarkeiten wollen wir beschränken?

- Klassen
- Methoden
- Attribute

Zugriffskontrolle bei Klassen

Es gibt zwei mögliche Zugriffsbeschränkungen:

- Die Klasse ist von jedem Paket aus (nach Einbindung) sichtbar. Zugriffsmodifikator: `public`
  Klasse muss dann in eigener Datei gleichen Namens stehen.
- Die Klasse ist nur im eigenen Paket sichtbar, die Standardsichtbarkeit. Kein Zugriffsmodifikator notwendig.

Beispiel für Sichtbarkeit von Klassen

```
package a;
class K { // Hier sind K, L, M und N sichtbar
}
public class L { // Hier sind K, L, M und N sichtbar
}
package a.c;
public class M { // Hier sind L, M und N sichtbar
}
package b;
public class N { // Hier sind L, M, N und P sichtbar
}
class P { // Hier sind L, M, N und P sichtbar
}
```

Zugriffsmodifikatoren für Klassenelemente

Es gibt vier Sichtbarkeitsstufen für Klassenelemente:

- `public`: Überall sichtbar
- `protected`: Im Paket und abgeleiteten Klassen sichtbar
- Standardzugriff: Im Paket sichtbar
- `private`: In der eigenen Klasse sichtbar

Zugriffsmodifikatoren werden Deklaration vorangestellt und sind vor `static` oder `final` zu nennen.

Sonderfall `protected`

Definition von `protected` weicht von UML und Industriestandard ab:
In UML: `protected` bedeutet Sichtbarkeit nur in eigener Klasse und abgeleiteten Klassen. Verwendung wie in UML beispielsweise in C#, Delphi, Visual Basic

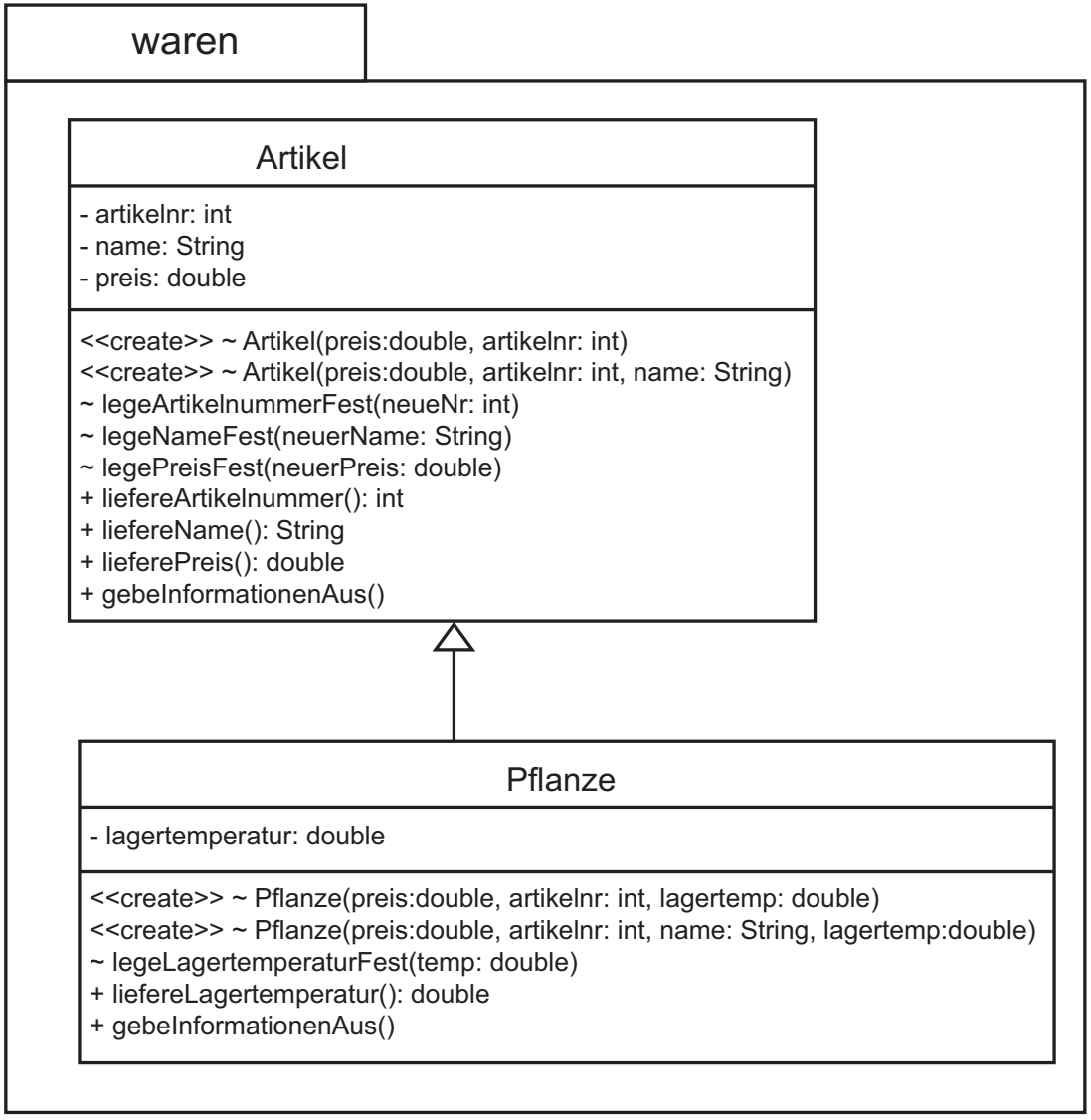
Beispiel für Sichtbarkeit von Klassenmembern

```
package a;
class K {
   public int a; // Sichtbar in K, L
}
public class L {
   public int a; // Sichtbar in K, L, M, N
   protected int b; // Sichtbar in K, L, N
   private static int c; // Sichtbar in L
   final int d = 5; // Sichtbar in K, L
}
package b;
public class M { }
public class N extends L { }
```

Sichtbarkeitsmodifikatoren in UML

`public`	$+$
`protected`	$\#$
Standardzugriff	$\sim$
`private`	`-`

Beispiel Zugriffsmodifikatoren in UML



Zugriffskontrolle und Vererbung

Wird ein Klassenmember bei Vererbung überschrieben, dürfen die Zugriffsrechte nur erweitert werden:
`public` > `protected` > Standardzugriff > `private`

Beispiel: Vorteile des Geheimnisprinzips

```
public class ComplexNumbers {
   private double real;
   private double imaginary;
   public double getReal() {
      return real;
   }
   public void setReal(double real) {
      this.real = real;
   }
   public double getImaginary() {
      return imaginary;
   }
   public void setImaginary(double imaginary) {
      this.imaginary = imaginary;
   }
```

Beispiel: Vorteile des Geheimnisprinzips

```
public double getRadius() {
   return Math.sqrt(real * real + imaginary *
       imaginary);
}
public double getAngle() {
   return Math.atan2(real, imaginary);
}
```

Beispiel: Vorteile des Geheimnisprinzips

```
    public void setRadius(double radius) {
        double oldRadius = getRadius();
        if(oldRadius == 0) {
            real = radius;
            imaginary = 0;
            return;
        }
        real = real / oldRadius * radius;
        imaginary = imaginary / oldRadius * radius;
    }
    public void setAngle(double angle) {
        double radius = getRadius();
        real = radius * Math.cos(angle);
        imaginary = radius * Math.sin(angle);
    }
}
```

Alternative Implementation

```
public class ComplexNumbers {
   private double radius;
   private double angle;
   public double getRadius() {
      return radius;
   }
   public double getAngle() {
      return angle;
   }
   public void setRadius(double radius) {
      this.radius = radius;
   }
   public void setAngle(double angle) {
      this.angle = angle;
   }
```

Alternative Implementation

```
    public double getReal() {
        return radius * Math.sin(angle);
    }
    public void setReal(double real) {
        double imaginary = getImaginary();
        radius = Math.sqrt(real*real + imaginary*imaginary);
        angle = Math.atan2(real, imaginary);
    }
    public double getImaginary() {
        return radius * Math.cos(angle);
    }
    public void setImaginary(double imaginary) {
        double real = getReal();
        radius = Math.sqrt(real*real + imaginary*imaginary);
        angle = Math.atan2(real, imaginary);
    }
}
```

Fragen zur Vorlesungseinheit

1. Wozu sollte man Zugriffsbeschränkungen vornehmen?
2. Von welchen Klassen aus kann auf einen Member mit Standardzugriff zugegriffen werden?
3. Was ist der Unterschied zwischen der UML- und der Java-Interpretation der Zugriffsklasse `protected`?