HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Java lernen – Sichtbarkeit

HPI-Team

Was war? Was kommt?

Woche 1:

- Einführung Java
- Datentypen
- Objektorientierung

Woche 2:

- Parameter
- Attribute
- Kontrollflussmuster

Dieses Video:

- Sichtbarkeit

Was ist Sichtbarkeit und wozu braucht man sie?

Sichtbarkeit: Eigenschaft von Klassen, Attributen, Methoden

- Beschränkte Sichtbarkeit eines Elements
  - Sichtbar: lesen/verändern ist möglich
  - Nicht Sichtbar: Element kann nicht gelesen und verändert werden
  - Innerhalb einer Klasse sind alle Elemente der Klasse sichtbar

Sichtbarkeit für Methoden und Attribute

- Sichtbarkeit eingrenzen:
  - Globale Sichtbarkeit vermeiden
  - Daten gehören einer Klasse allein
  - Komplexität verringern (Modularisierung)

Sichtbarkeit – an Beispielen

```java
public class Papagei {
    public String name;
}
```

```java
public class Pinguin {
    private String name;
}
```

```java
public class Zoo {
    public static void main(String[] args) {
        Papagei alex = new Papagei();
        Pinguin tux = new Pinguin();

        alex.name = "Alex";
        tux.name = "Tux";
    }
}
```

Wie grenzt man die Sichtbarkeit ein?

Nutzung von **Sichtbarkeitsmodifizierern** (Schlüsselwörter)

- **public**
  - Uneingeschränkter Zugriff
- **protected**
  - Zugriff aus der eigenen Klasse und deren Subklassen
- (Kein Modifizierer angegeben)
  - Zugriff für Klassen des Pakets
- **private**
  - Kein Zugriff für andere Klassen

Public – Offene Daten und Methoden

Public – uneingeschränkte Sichtbarkeit

- Lesen & Schreiben von überall
- Verwenden bei Öffentliche Methoden

```
1  public class Papagei {
2      public String name;
3  }
```

Beispiel

- Name kann geändert werden

```
1  public static void main(String[] args) {
2      Papagei alex = new Papagei();
3      // Name kann gelöscht werden
4      Papagei.name = null;
5  }
```

Private – Stark eingeschränkte Sichtbarkeit

Begrenzung auf eigene **Klasse**

- Kein Zugriff von außen
- Zugriff optional über Getter/Setter möglich
- Vollständige Kontrolle
- Konvention: Alle Attribute private

```
1  public class Papagei {
2      private name = "Alex";
3      public void setName(String neuerName) {
4          if (null != neuerName) {
5              name = neuerName;
6          }
7      }
8  }
```

```
2  public static void main(String[] args) {
3      Papagei alex = new Papagei();
4      // Name kann nicht gelöscht werden
5      alex.setName(null);
6  }
```

Private – Sichtbarkeit zweier Objekte einer Klasse

Zwei Objekte einer Klasse - gegenseitiger Zugriff möglich

```
1  public class Papagei {
2      private String name = "Alex";
3
4      public boolean istNameGleich(Papagei anderer) {
5          if(name.equals(anderer.name)) {
6              return true;
7          } else {
8              return false;
9          }
10     }
11 }
12
13 public static void main(String[] args) {
14     Papagei alex1 = new Papagei();
15     Papagei alex2 = new Papagei();
16     if(alex1.istNameGleich(alex2)) {
17         System.out.println("Die Namen sind gleich!");
18     }
19 }
```

Zusammenfassung

Schlüsselwörter zur Sichtbarkeitsbegrenzung:

- public - öffentlich sichtbar
- protected - für Klasse und Subklassen
- (keins) - im Paket/Ordner
- private - nur für Objekte der Klasse

Zusammenfassung

Schlüsselwörter zur Sichtbarkeitsbegrenzung:

- public - öffentlich sichtbar
- protected - für Klasse und Subklassen
- (keins) - im Paket/Ordner
- private - nur für Objekte der Klasse

Java lernen – Überladen von Methoden

HPI-Team

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Rückblick auf Woche 1 - Methoden

Methoden

- Repräsentieren Verhalten eines Objektes
- kann auf Objekten aufgerufen werden
- können Rückgabewerte haben
- können Parameter haben

Beispiel

```
1  class Papagei {
2      String sagHallo() {
3          return "Hallo";
4      }
5  }
```

Überladen von Methoden

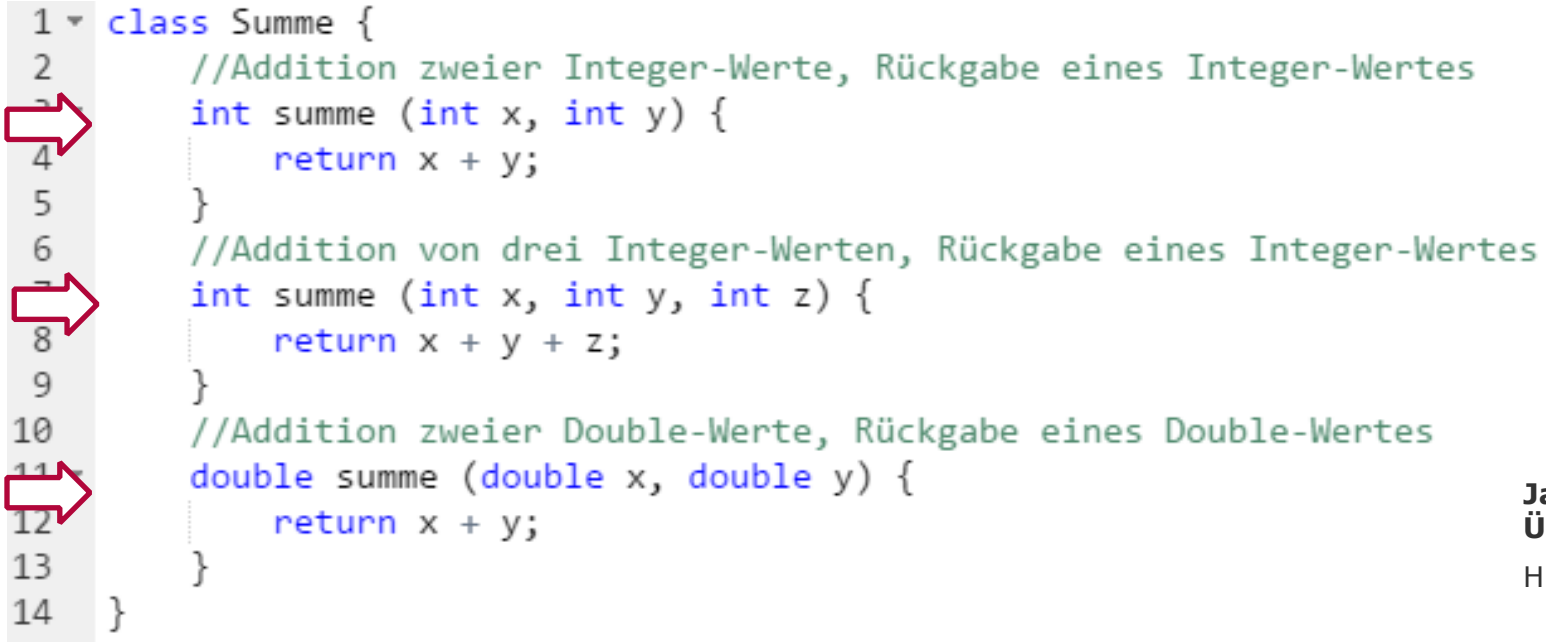
Überladen

- Gleicher Name
- unterschiedliche Parameterliste
  - Anzahl der Parameter
  - Typen der Parameter
- Rückgabetyp nicht entscheidend
- Parameternamen nicht entscheidend

Es wird die am Besten übereinstimmende Methode ausgeführt.

Überladen von Methoden - Beispiel

Beispiel



```
1  class Summe {
2      //Addition zweier Integer-Werte, Rückgabe eines Integer-Wertes
3      int summe (int x, int y) {
4          return x + y;
5      }
6      //Addition von drei Integer-Werten, Rückgabe eines Integer-Wertes
7      int summe (int x, int y, int z) {
8          return x + y + z;
9      }
10     //Addition zweier Double-Werte, Rückgabe eines Double-Wertes
11     double summe (double x, double y) {
12         return x + y;
13     }
14 }
```

Überladen von Methoden - Beispiel

Beispiel

- Aufruf: `summe(3, 2);`
- Rückgabewert: `5`

```
1  class Summe {
2      //Addition zweier Integer-Werte, Rückgabe eines Integer-Wertes
3      int summe (int x, int y) {
4          return x + y;
5      }
6      //Addition von drei Integer-Werten, Rückgabe eines Integer-Wertes
7      int summe (int x, int y, int z) {
8          return x + y + z;
9      }
10     //Addition zweier Double-Werte, Rückgabe eines Double-Wertes
11     double summe (double x, double y) {
12         return x + y;
13     }
14 }
```

Überladen von Methoden - Beispiel

Beispiel

Aufruf: `summe(3.0, 2);`

- Rückgabewert: `5.0`

```
1  class Summe {
2      //Addition zweier Integer-Werte, Rückgabe eines Integer-Wertes
3      int summe (int x, int y) {
4          return x + y;
5      }
6      //Addition von drei Integer-Werten, Rückgabe eines Integer-Wertes
7      int summe (int x, int y, int z) {
8          return x + y + z;
9      }
10     //Addition zweier Double-Werte, Rückgabe eines Double-Wertes
11     double summe (double x, double y) {
12         return x + y;
13     }
14 }
```

Überladen von Methoden - Verwendung

Falsche Verwendung – Compiler-Fehler!

```
1  class Summe {
2      //Addition zweier Integer-Werte, Rückgabe eines Integer-Wertes
3      int summe(int x, int y) {
4          return x + y;
5      }
6
7      //Addition zweier Integer-Werte, Rückgabe eines Integer-Wertes
8      int summe(int y, int x) {
9          return y + x;
10     }
11 }
```

```
Summe.java:8: error: method summe(int,int) is already defined in class Summe
    int summe(int y, int x) {
        ^
1 error
make: *** [run] Error 1
```

Überladen von Methoden - Verwendung

Falsche Verwendung – Compiler-Fehler!

```
1  class Summe {
2      //Addition zweier Integer-Werte, Rückgabe eines Integer-Wertes
3      int summe(int x, int y) {
4          return x + y;
5      }
6
7      //Addition zweier Integer-Werte, Rückgabe einer Zeichenkette
8      String summe(int y, int x) {
9          return "Die Summe ist" + (y + x);
10     }
11 }
```

```
Summe.java:8: error: method summe(int,int) is already defined in class Summe
    String summe(int y, int x) {
           ^
1 error
make: *** [run] Error 1
```

Überladen von Methoden – Weitere Beispiele

print bzw. println ist überladen

```
1  class PrintStream {
2      void print(int arg) { ... }
3      void print(String arg) { ... }
4      void print(char[] arg) { ... }
5      ...
6  }
```

Vorteile des Überladens von Methoden:

- Methodennamen wiederverwenden, nicht printString, printInt, etc.
- Variieren der Ausführung (z.B. Berechnung) abhängig von Parameter

Überladen von Methoden – Weitere Beispiele

print bzw. println ist überladen

```
1  class PrintStream {
2      void print(int arg) { ... }
3      void print(String arg) { ... }
4      void print(char[] arg) { ... }
5      ...
6  }
```

Vorteile des Überladens von Methoden:

- Methodennamen wiederverwenden, nicht printString, printInt, etc.
- Variieren der Ausführung (z.B. Berechnung) abhängig von Parameter

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Java lernen – Vererbung

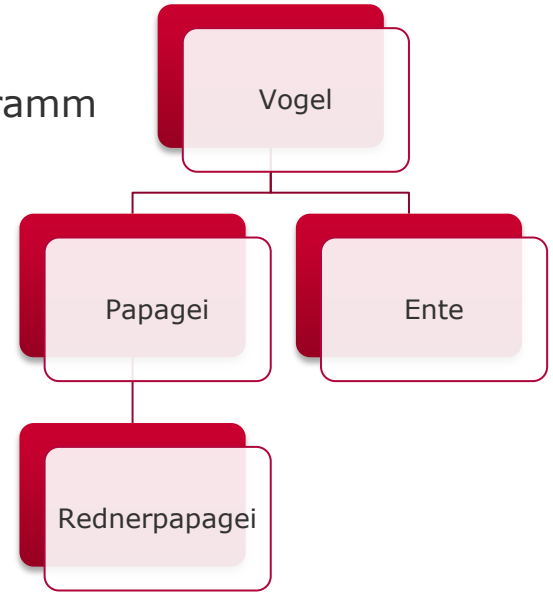
HPI-Team

Bisher

- Dinge der echten Welt werden in **Klassen** beschrieben
- Klassen beschreiben die **Struktur der Objekte**

Eine Hierarchie im Tierreich

- Papagei ist ein Vogel
- Ein Rednerpapagei ist ein Papagei, der alles dreimal sagt und seinen Namen immer sagt
- Ente ist ein Vogel
- Nachbilden solcher Hierarchien im Programm



Die Klasse Vogel

Vogel.java

```
1  public class Vogel {
2
3      String name;
4      boolean kannFliegen = true;
5
6      void sagHallo(){
7      }
8
9      String getName() {
10         return name;
11     }
12
13 }
```

Die Klasse Ente

Ente.java

```
public class Ente extends Vogel {
    boolean kannSchwimmen = true;
    void sagHallo() {
        System.out.println("Quak");
    }
}
```

- Enthält implizit alle Methoden der Oberklasse (getName)
- Enthält implizit alle Attribute der Oberklasse (name, kannFliegen)
- Kann Methoden und Attribute überschreiben (sagHallo)
- Kann Attribute und Methoden ergänzen (kannSchwimmen)

Die Klasse Papagei

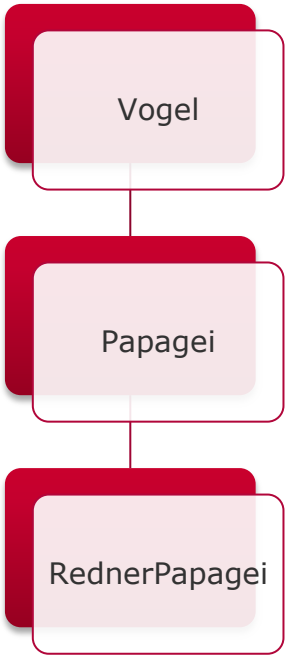
```
1 public class Papagei extends Vogel {
2     void sagHallo() {
3         System.out.println("Hallo");
4     }
5 }
```

Die Klasse Rednerpapagei

Rednerpapagei.java

```
public class RednerPapagei extends Papagei {
    void sagHallo() {
        for(int i = 0; i < 3; i++) {
            System.out.println("Hallo");
        }
    }

    String getName() {
        System.out.println(name);
        return name;
    }
}
```



- Überschreibt Methode der Oberklasse (sagHallo)
- Kann auch Methoden und Attribute einer höheren Oberklasse überschreiben (getName)

Polymorphie

- In der Variable vogel kann ein Objekt vom Typ Vogel gespeichert werden

```
1  public class Zoo {
2      public static void main(String[] args){
           Vogel vogel;
           vogel = new Ente();
           vogel = new Papagei();
           vogel = new RednerPapagei();
7      }
8  }
```

- Ebenso alle Unterklassen von Vogel

Polymorphie

- Aufrufen einer Methode ruft die speziellste Methode auf

Zoo.java

```
1  public class Zoo {
2      public static void main(String[] args) {
3          Vogel vogel1, vogel2, vogel3;
4          vogel1 = new Ente();
5          vogel1.sagHallo();
6          vogel2 = new Papagei();
7          vogel2.sagHallo();
8          vogel3 = new RednerPapagei();
9          vogel3.sagHallo();
10     }
11 }
```

Ausgabe

```
Quak
Hallo
Hallo Hallo Hallo
```

Polymorphie

- Aufrufen einer Methode ruft die speziellste Methode auf

Zoo.java

```
1  public class Zoo {
2      public static void main(String[] args) {
3          Vogel vogel;
4          vogel = new Ente();
5          vogel.kannSchwimmen = false;
6      }
7  }
```

- Programm weiß nicht, dass Vogel eine Ente ist (Typ der Variable ist Vogel)

```
Zoo.java:5: error: cannot find symbol
        vogel.kannSchwimmen = false;
             ^
  symbol:   variable kannSchwimmen
  location: variable vogel of type Vogel
```

Konvertieren in Unterklasse - Typecasting

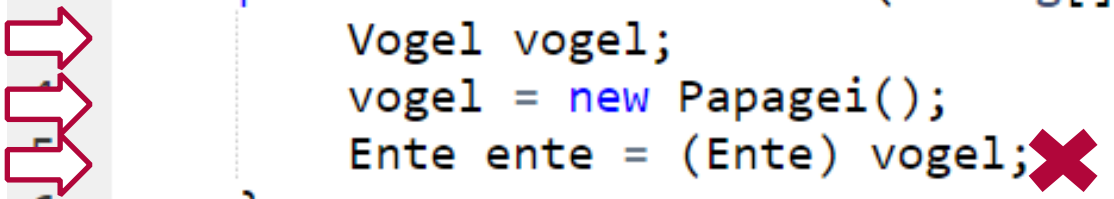
- Das das Objekt das in der Variable gespeichert ist tatsächlich den Typ der Unterklasse hat in die gecastet werden soll

```
1  public class Zoo {
2      public static void main(String[] args) {
3          Vogel vogel;
4          vogel = new Ente();
5          Ente ente = (Ente) vogel;
6          ente.kannSchwimmen = false;
7      }
8  }
```

Casting zwischen Unterklassen ist nicht möglich

Zoo.java

```
1 public class Zoo {
2     public static void main(String[] args) {
          Vogel vogel;
          vogel = new Papagei();
          Ente ente = (Ente) vogel;
6     }
7 }
```



```
Exception in thread "main" java.lang.ClassCastException:
 Papagei cannot be cast to Ente
        at Zoo.main(Zoo.java:6)
make: *** [run] Error 1
```

Casting zwischen Unterklassen ist nicht möglich

Zoo.java

```java
public class Zoo {
    public static void main(String[] args) {
        Vogel vogel;
        vogel = new Papagei();
        Ente ente = (Ente) vogel;
    }
}
```

```
Exception in thread "main" java.lang.ClassCastException:
 Papagei cannot be cast to Ente
        at Zoo.main(Zoo.java:6)
make: *** [run] Error 1
```