Java Einführung

Methoden in Klassen

Lehrziel der Einheit

- Methoden
  - Signatur (=Deklaration) einer Methode
  - Zugriff/Sichtbarkeit
  - Rückgabewerte
  - Parameter
  - Aufruf von Methoden (Nachrichten)
- Information Hiding
- Konstruktoren

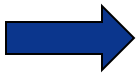
Klassendefinition

In der Designphase wurden

- die Eigenschaften (Variablen) und
- **das Verhalten (Methoden) definiert.**

Im Folgenden wird das Implementieren der Methoden behandelt.

KlassenName
-Variablen
+Methoden()



```
class KlassenName {
   variablenDeklarationen;
   methodenDeklarationen(){
  }
}
```

Methoden

- Objekte interagieren miteinander durch das gegenseitige Aufrufen von Methoden (=Nachrichtenaustausch).
- Methoden sind Programmteile, die bestimmte Teilaufgaben lösen (und sollten nach diesen selbstsprechend benannt werden).
- Eine Methode (Nachricht) besteht aus zwei Teilen:
  - **der Signatur (bzw. Methodendeklaration)**, die Angaben über Sichtbarkeit, Rückgabetyp, Name der Methode und Parameter macht.
  - **dem Methodenrumpf**, in dem die Deklarationen der lokalen Variablen und die eigentlichen Anweisungen stehen.

Zweck von Methoden

- Zustand eines Objektes abfragen

```
point1.getX();
aString.length();
```

- Aufforderung an ein Objekt, etwas zu tun (z.B. Instanzvariable verändern)

```
point1.setInvisible(true);
aString.toUpperCase();
```

Methodendeklaration

Methodenname

Bezeichner des Parameters

Zugriff/Sichtbarkeit

```
public double setX(double xValue)
```

Typ des Rückgabewerts

Typ des Parameters

Beispiel zu Methode

```
class Point {
    public double x, y;
//Methodendeklaration:
    public double distance(Point pkt) {
        double xdiff = x - pkt.x;
        double ydiff = y - pkt.y;
        return Math.sqrt(xdiff*xdiff +ydiff*ydiff);
    }
...

//Methodenaufruf:
Class PointTester {
    public static void main(String[] args) {
       Point lowerLeft = new Point(); lowerLeft.x = 0.1; …
       Point upperRigth = new Point(); upperRight.x = 0.1; …
       double d = lowerLeft.distance(upperRight);
}
```

danach enthält Variable d die Euklidische Distanz zwischen `lowerLeft` und `upperRight`.

Variablenzugriff

- Innerhalb der Methode einer Instanz können die eigenen Variablen und Methoden direkt (d.h. ohne Angabe des Instanznamens) aufgerufen werden.
- z.B. Variable `x` in der Instanz `lowerLeft` bei Aufruf `lowerLeft.distance(upperRight)`

Instanz lowerLeft

```
x, y

public double distance(Point pkt) {
  double xdiff = x - pkt.x;
  double ydiff = y - pkt.y;
  return Math.sqrt(xdiff*xdiff
        +ydiff*ydiff);
}
```

Instanz upperRight

```
x, y
```

Standardvariable „this“

- `this` ist eine Referenz zum aktuellen Objekt, oder anders ausgedrückt
- `this` ist eine Referenz auf die aktuelle Instanz der Klasse in der man sich gerade befindet.
- Über `this` kann auf alle Variablen und Methoden der Instanz zugegriffen werden.

this 2

```
class Point {
  double x,y;
  void setX(double x) {
   this.x = x;
   //Instanzvariable x wird gesetzt
  }} …
```

Instanzvariablenname

Parametername

Innerhalb einer Methode überschreiben bei Namenskonflikten (=gleicher Bezeichnung) die Parameternamen die Variablennamen der Klasse.
Lösung: Verwendung von `this` oder anderen Parameterbezeichner
wählen, z.B. `newName`.

Statische Elemente

- Variablen und Methoden, die nicht zu einer bestimmten Instanz sonder zur Klasse gehören.
- Statische Variablen/Methoden sind auch dann verfügbar, wenn noch keine Instanz der Klasse erzeugt wurde.
- Statische Variablen/Methoden können über den Klassennamen aufgerufen werden.
- Deklaration durch das Schlüsselwort: **`static`**
- Bsp:

```
class Point {
  double x, y;
  static int count;
}
```

- `Point.count` kann dann zum Beispiel benutzt werden um die Anzahl der Instanzen von Punkt zu speichern. Auf diese Variable kann von jeder Instanz der Klasse `Point` aus zugegriffen werden.

Statische Elemente Programmaufbau

```
class Programm {
  public static void main(String[] args) …
```

- Jede Applikation braucht **eine** Klasse, wo die Applikation startet. Diese Klasse hat eine `main`-Methode. Das Programm startet mit dieser Klasse bei der `main`-Methode.
- Der `main`-Methode werden die Eigenschaften `public` und `static` zugewiesen.
- `static` bedeutet, wie wir wissen, dass keine Instanz der Klasse angelegt werden muss. Und zu Beginn einer Applikation existieren ja auch noch keine Instanzen.
- `public` bedeutet hier, dass die Java Virtual Machine die Main-Methode des Programms aufrufen darf, um das Programm zu starten.

Konstruktoren

Jede Klasse benötigt einen oder mehrere Konstruktoren.
Konstruktoren:

- reservieren den Speicherplatz für eine neu zu erzeugende Instanz,
- weisen den Instanzvariablen initiale Werte zu,
- haben denselben Namen wie die Klasse,
- werden wie Methoden deklariert, aber *ohne Rückgabewert* `KlassenName(Parameter){..}`.

Zu einer Klasse können mehrere Konstruktoren mit verschiedenen Parametern deklariert werden.

Konstruktoren

Beispielklasse Student

```
class Student {
  private String matrNr;
  private String name;
  private int semester;

Student(String name, String matrNr) {
   this.name = name;
   this.matrNr = matrNr;
  }
}
```

Konstruktoren - Anwendung

- Jedes Objekt der Klasse `Student` kann nun durch den Konstruktor mit zwei Werten initialisiert werden:
  - Eine **Zeichenkette**, um den Namen zu initialisieren.
  - Eine **Zahl** um die Matrikelnummer zu setzen.

```
Student myStudent =
   new Student("Else Maier", "0524343");
```

Spezialisierte Konstruktoren

Zweiter Konstruktor (in Klassendefinition) mit anderer Anzahl bzw. anderen Typen von Parametern:

```
Student(String name, String matrNr, int semester){
      this(name, matrNr);
      this.semester=semester;
    }
```

Aufruf (aus einem Programm):

```
Student myStudent =
    new Student("Else Maier", "9324343", 5);
```

- Abhängig von der Parameterzahl und ihren Datentypen wird der entsprechende Konstruktor aufgerufen.
- Innerhalb dieses Konstruktors wird der erste Konstruktor über `this(...);` aufgerufen.

Spezialisierte Konstruktoren

Beispielklasse Student

```
class Student {
   private String name, matrNr;
   private int semester;

Student(String studName, String studMatrNr) { //Konstruktor 1

    name = studName;
    matrNr = studMatrNr;
   }
Student(String name, String matrNr, int semester) {
   // Konstruktor 2
        this(name, matrNr); //Aufruf Konstruktor 1
    this.semester=semester;
   }
}
```

Default-Konstruktor

```
//Konstruktor 1 verwenden
Student s1 =
   new Student("Petra Schmidt", "0051998");
//Konstruktor 2 verwenden
Student s2 =
   new Student("Peter Baum", "9988764", 2);
```

- Wenn **kein Konstruktor erstellt wurde**, wird von Java default-mäßig ein Konstruktor (ohne Parameter) zur Verfügung gestellt.

```
Student s = new Student();
```

Zugriffskontrolle

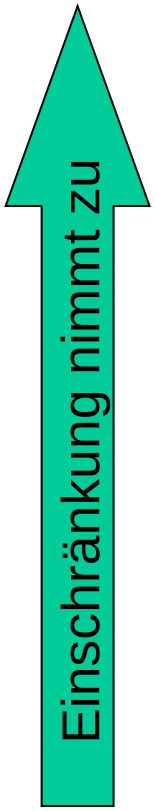
In JAVA gibt es drei Attribute und eine Standardeinstellung, die den Gültigkeitsbereich von Klassen, Variablen und Methoden regeln, d.h. festlegen, ob bzw. welche anderen Objekte auf eine Klasse, Variable oder Methode der programmierten Klasse zugreifen können:

- `private`
- `protected`
- `public`
- *package* (Standardeinstellung)

Zugriffskontrolle 2

Grundsätzlich: alle Methoden oder Variablen können innerhalb der Klasse, in der sie geschrieben sind, verwendet oder aufgerufen werden.

- `private`
  Methoden oder Variablen können **nur** innerhalb der eigenen Klasse verwendet oder aufgerufen werden.
- keine Angabe **(=Standardeinstellung)**
  Methoden oder Variablen sind innerhalb der Eigenen Klasse und innerhalb ihres *Pakets** zugreifbar.
- `protected`
  Wie Standardeintellung. Zusätzlich können abgeleite Klassen** (auch außerhalb des Pakets*) diese Methoden und Variablen verwenden.
- `public`
  Methoden oder Variablen lassen sich generell von allen anderen Klassen und Paketen aus aufrufen (Interface von Klassen in Paketen).



* Pakete sind Sammlungen von zusammengehörigen Klassen.

** Vererbung wird später behandelt.

Zugriffskontrolle 3

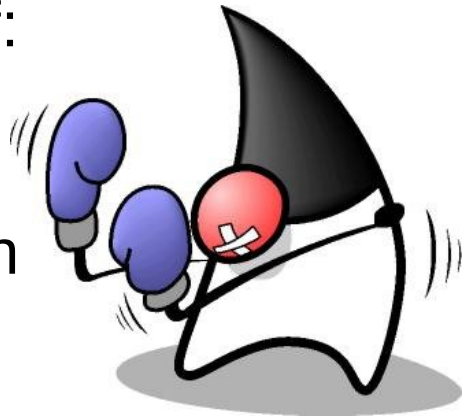
```
class Student {
   private String name;
}
   // ...
class Uni {
  public static void main(String[] args) {
    Student stud1 = new Student();
    stud1.name = “Peter“; /* Zugriff außerhalb der Klasse
  Student auf private-Variable nicht möglich, kann nur innerhalb
  der Klasse Student gesetzt werden */
  }
}
```

Information Hiding

- Verstecken der Implementierung einer Klasse hinter den Methoden, die auf ihre internen Daten ausgeführt werden.
- Das heißt: kein direkter Zugriff auf die Variablen, sondern nur via Methodenaufruf:

```
stud1.setName("Peter Meyer");
String name = stud1.getName();
```

- Dieses wichtige Konzept der ooP wird auch „Data Encapsulation“ (Datenkapselung) genannt.



Information Hiding - Code 1

Die Deklaration der Variablen als **`private`** verhindert den unerwünschten Zugriff auf die Variablen von außerhalb der Klasse.

```
class  Student
{
   private String   name;
   private int      matNr;

...
}
```

Information Hiding - Code 2

Auf die Variablen kann nur über die Methoden `get` und `set` zugegriffen werden. Diese Methoden müssen **`public`** deklariert werden.

```
...
    public String getName()
    {
        return name;
    }
    public int getMatNr()
    {
        return matNr;
    }
...
```

Information Hiding - Code 3

```
...
public void setName(String name)
{
  this.name = name;
}
public void setMatNr(int matNr)
{
  this.matNr = matNr;
}
```

Mit den Methoden `set`… können den entsprechenden Variablen Werte zugewiesen werden.

Klasse Student gesamt

```
class  Student {
   private String name;
   private int  matNr;

   Student(String name, int matNr){
     this.name = name; this.matNr = matNr;
   }


   public String getName(){ return name; }
      public int getMatNr(){ return matNr;}

   /* keine set-Methoden: Die Variablen
      können nur gelesen werden */
}
```

Information Hiding - Vorteile

- Die Implementierung der Daten kann verändert werden, ohne dass Programmcode außerhalb der Klasse geändert werden muss.

- Der Zugriff auf die verschiedenen Variablen kann eingeschränkt werden. Dies verhindert sowohl Datenmanipulationen als auch irrtümliche Veränderung oder Abfrage der Daten. Außerdem können damit die Programmierfehler eingeschränkt werden.

Nach dieser Einheit sollten Sie ...

- die Zugriffskontrolle von Methoden kennen,
- die Signatur verschiedener Methoden interpretieren können,
- die Bedeutung der Konstruktoren kennen,
- Methoden implementieren können,
- Information Hiding und statische Methoden kennen und anwenden können.