Westfälische
Wilhelms-Universität
Münster

Programmieren in Java

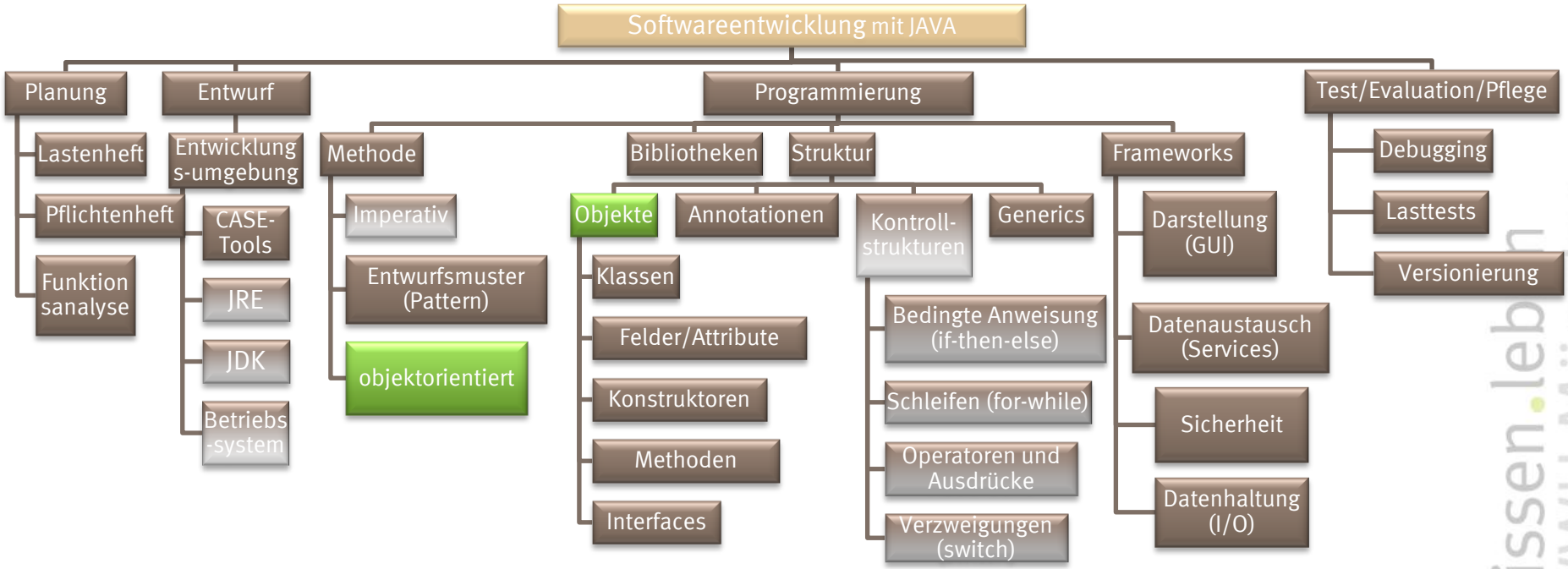
Einführung in die objektorientierte Programmierung

wissen.leben
WWU Münster

Patrick Förster, Thorsten Hovenga

ZENTRUM FÜR
INFORMATIONS
VERARBEITUNG

Einordnung in den Softwareentwicklungsprozess



Motivation und Geschichte

- 70er-Jahre: prozedurale Programmierung
  - Anweisungen werden nach festgelegter Reihenfolge abgearbeitet
  - Strukturierung durch Prozeduren und Funktionen
  - Datenstrukturen auf anderer Hierachieebene
- Ziele:
  - Abbildung von Problemen aus der „Natur“
  - Wartbarkeit herstellen
  - Datenstrukturen in Programmlogik einbeziehen

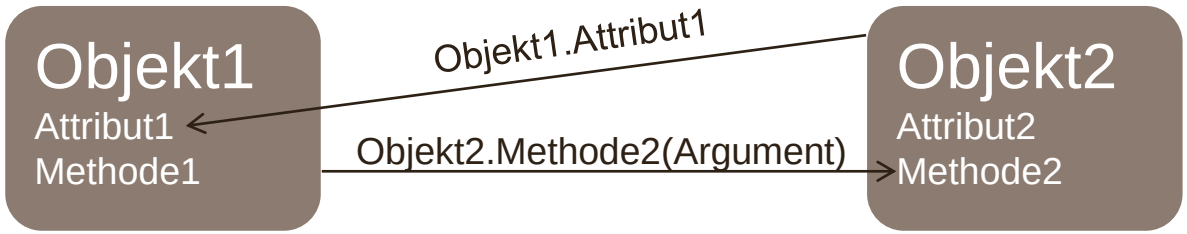
Definition und Begriffszusammenhänge

- Objekt: enthält zusammengehörige Attribute (Daten) und Methoden (Anweisungen)
- Methode: Umsetzung der Objektfunktionalität
- Attribut: Dateneinheit (z.B.Variable)

- Datenkapselung: Objekt von aussen nur durch seine (Attribute und) Methoden sichtbar, Funktionalität ist gekapselt

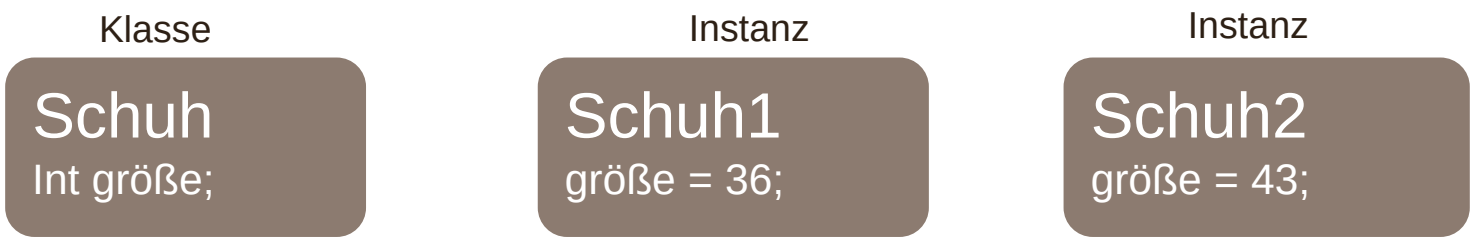
Kommunikation zwischen Objekten

- Zwischen verschiedenen Objekten muss es eine Möglichkeit zur Kommunikation geben
- Attribute und Methoden müssen adressierbar sein
- Punktnotation zum trennen von Objektname und Attribut/Methode

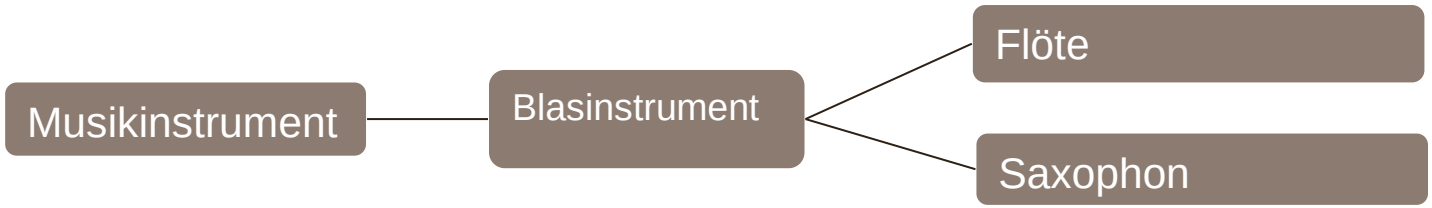


Klassen und Instanzen

- Klasse = Gruppierung von Objekten mit gleicher „Schablone“
- Einzelnes Objekt = Instanz einer Klasse

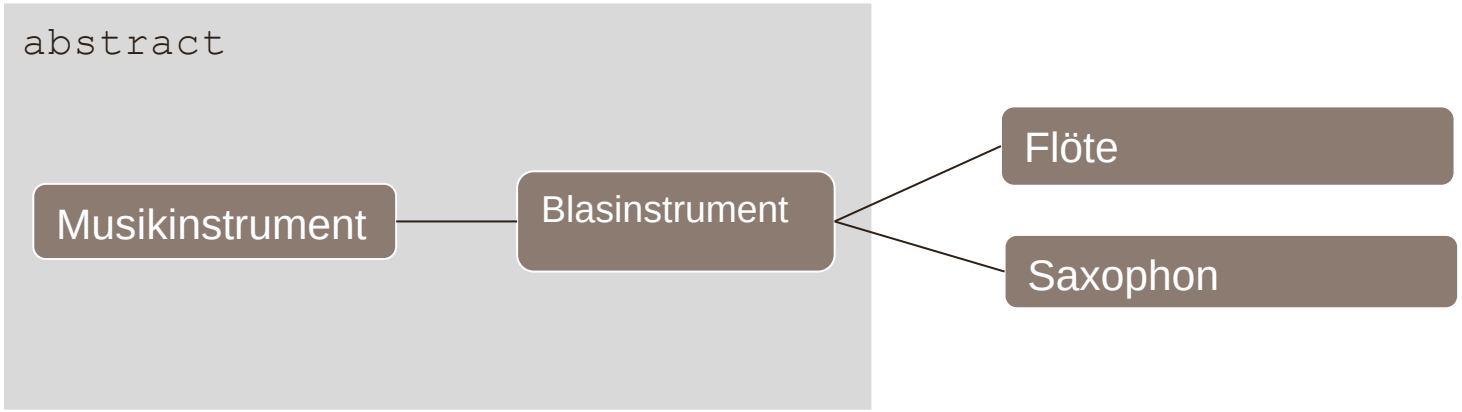


- Klassen können in einer Hierarchie geordnet werden („Ist-ein-Beziehung“)



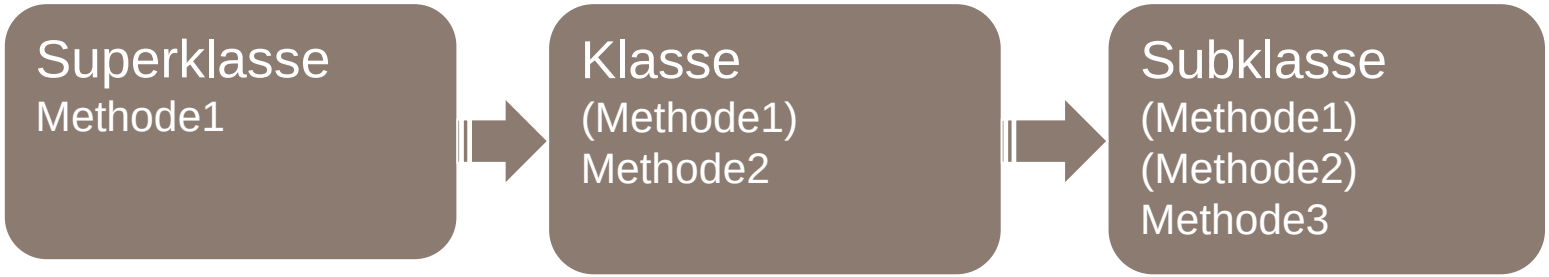
abstrakte Klassen

- Implementieren keine Methoden
- Werden nicht zum erzeugen von Instanzen benutzt
- Dienen zur Strukturierung von Klassenhierarchien
- Ermöglichen das Zusammenfassen von Eigenschaften von Klassen ohne explizite Definition welche Klasse ober- bzw. Unterklasse ist.



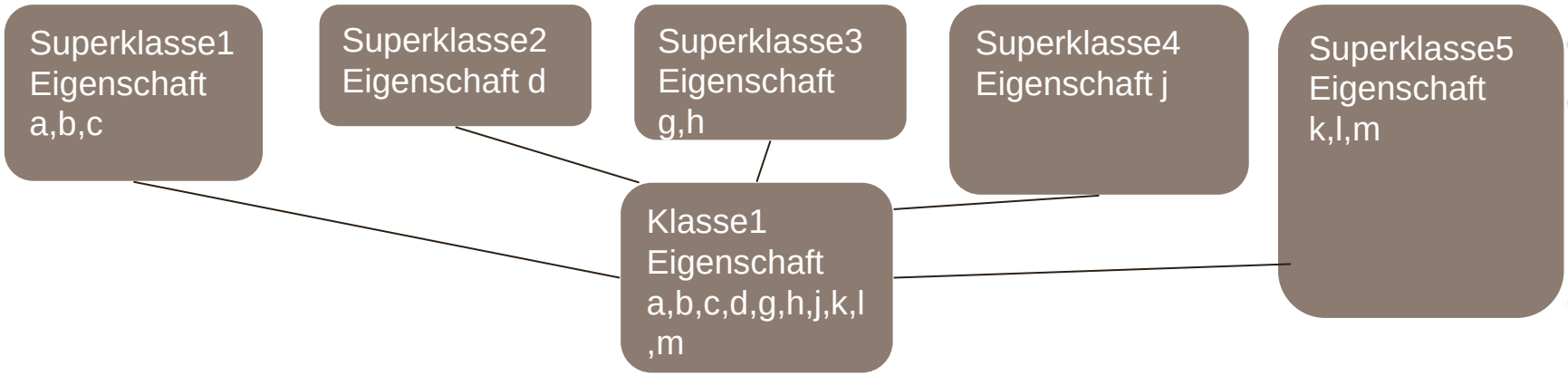
Prinzip der Vererbung

- Beziehung zwischen den Klassen einer Hierarchie lässt sich durch Superklasse wird abgeleitet zu betrachteten Klasse und gegebenenfalls weiter abgeleitet in Subklasse abbilden
- Dabei gilt: abgeleitete Klassen übernehmen die Eigenschaften und Methoden aller übergeordneten Klassen
- Ererbte Eigenschaften und Methoden werden wie eigene Eigenschaften und Methoden behandelt bei der Kommunikation mit einem Objekt einer Klasse
- Vererbungsregel: ist eine Methode/Eigenschaft in der aufgerufenen Klasse nicht vorhanden wird in der nächsthöheren Superklasse danach gesucht und dann in deren Superklasse usw. bis zum ersten Auffinden der Methode/Eigenschaft



Mehrfachvererbung

- Wird in Java nicht unterstützt kommt aber in der „Natur“ durchaus vor
- Prinzip: Ein Objekt erbt von mehreren Superklassen die voneinander unabhängig sind
- Erlaubt eine einfache und vollständige Beschreibung von einzelnen Objekten
- Komplexität steigt unverhältnismäßig bei größeren Projekten



Überschriebene Methoden und Polymorphismus

- Geerbte Methoden können überschrieben werden um eine Spezialisierung in einer Unterklasse darzustellen
- Daraus folgt, dass es namensgleiche Methoden gibt die unterscheidbare Funktionalitäten implementieren
- Bei der Entscheidung welche Methode denn ausgeführt wird gibt es dann einen Konflikt wenn die Klasse mit der ein Objekt erzeugt wird sich von dem Objekttyp unterscheidet
- Wenn der Objekttyp auschlaggebend ist, führt der Methodenaufruf zu verschiedenem Verhalten trotz Namensgleichheit
- Dieses polymorphe Verhalten ist in Java der Standardfall

Klassen in Java

- Gründgerüst einer Klasse um Daten und Methoden in einem Datentyp zu kapseln

```
class test1 {
          ELEMENTDEKLARATION
          ELEMENTDEKLARATION
          ELEMENTDEKLARATION
          …
          ELEMENTDEKLARATION
}
```

- Element einer Klasse sind Datenfelder, Methoden und Konstruktoren

```
//Datenfelder
int zahl1;
int zahl2;
```

```
//Konstruktor
test1(int param1,
int param2){
zahl1 = param1;
zahl2 = param2;
}
```

```
//Methoden
int Summe() {
return zahl1 + zahl2;
}
int Produkt() {
return zahl1 * zahl2;
}
```

„optionale“ Angaben bei der Klassendefinition

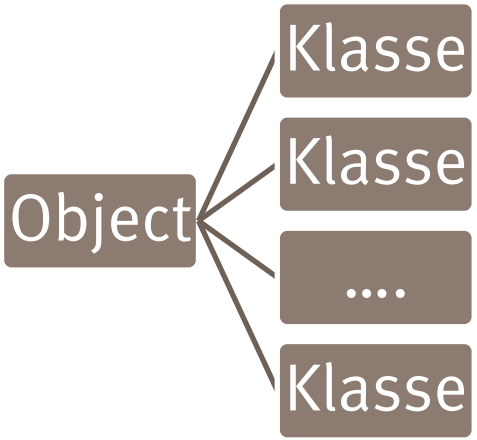
- Bei der Klassendeklaration sind weitere Angaben möglich, die wichtige Eigenschaften der Klasse festlegen

```
[<modifiers>] class <Klassenname> [extends <SuperklassenName>]
[implements <Schnittstellenname1>, <Schnittstellenname2>, …]
```

- Es kann nur ein Superklasse aber mehrere Schnittstellen angegeben werden
- Zugelassene Modifier sind:
- Keine Modifizierer: Die Klasse ist in allen Klassen des selben Pakets verfügbar
- `public`: Die Klasse ist auch in anderen Paketen verfügbar
- `abstract`: Die Klasse wird nur als Basisklasse für Subklassen verwendet
- `final`: Die Klasse kann nicht als Basisklasse für weitere Klassen verwendet werden
- `protected`,`private`,`strictfp`,`static`: auch zugelassen aber nur für innere Klassen oder nur für spezielle Klassen benötigt

Die Klasse Object

- Grundsätzliche Regel in objektorientierten Sprachen: alle Konstrukte werden als Objekte beschrieben.
- Das bedeutet dass auch Klassen selbst Objekte sein müssten
- Deshalb wird in Java die Metaklasse `Object` definiert
- Alle Klassen werden als Instanzen von `Object` abgeleitet
- Vererbt werden Methoden zum Erzeugen von Objekten oder zur Initialisierung von Klassenvariablen



Instanzen

- Bei der objektorientierten Programmierung wird nicht mit Klassen sondern mit Objekten gearbeitet. Diese müssen als Instanz ihrer Klasse erzeugt werden können.

```
Klassentyp objVar = new Klassentyp();
```

- `Klassentyp` bezeichnet hier den Klassennamen, `objVar` ist der Name der erzeugten Objektreferenz und `new` ist eine Klassenmethode von `Object` zum erzeugen von Objekten. `Klassentyp()` bezeichnet den Konstruktor der Klasse Klassentyp der aufgerufen werden soll.

Konstruktoren

- Konstruktoren sind spezielle Methoden die zum Initialisieren von Eigenschaften eines Objekts bei der Erstellung benutzt werden.
- Konstruktoren haben immer denselben Namen wie ihre Klasse.
- Konstruktoren haben keinen Rückgabewert, da sie beim Erstellen der Instanzen aufgerufen werden und Das Objekt selbst der Rückgabewert ist.
- Konstruktoren unterscheiden sich untereinander beim Aufruf durch verschieden Argumente und Typen der Argumente
- Wird in einer Klasse kein Konstruktor definiert wird der Standardkonstruktor der keine Argumente hat benutzt.

```
//Konstruktor1
test1(int param1, int param2){
zahl1 = param1;
zahl2 = param2;
}
```

```
//Konstruktor2
test1(double param1, int param2){
zahl1 = param1;
zahl2 = param2;
}
```

Konstruktoren(2)

- Das Festlegen konstanter Werte mittels Konstruktor ist nicht sinnvoll. Besser ist es konstante Werte direkt zuzuweisen.

```
class test1{
    int zahl1;
    int zahl2;
    test1(){
        zahl1 = 5;
        zahl2 = 10;
    }
}
```

```
class test1{
    int zahl1 = 5;
    int zahl2 = 10;
    test1(){
    }
}
```

Das Definieren mehrerer Konstruktoren stellt bereits ein Überladen einer Methode dar!

Attribute/Felder/Daten

- Attribute sind die Datenelemente einer Klasse. Sie werden Durch Angabe des Datentyps und durch ihren Namen beschrieben.
- Auch hier sind Modifizierer erlaubt: Um den Zugriff zu spezifizieren kann `private`, `protected` oder `public` verwendet werden. Außerdem möglich:
- `final`: Konstantes Attribut dessen Anfangswert nicht mehr geändert werden kann
- `transient`: Attribut das nicht persistent gespeichert wird
- `static`: Statisches Attribut das nicht über das Objekt sondern über die Klasse angesprochen wird
- `volatile`: Thread-sicheres Attribut

```
int feld1;
private int feld2;
private final feld3 = 5;
```

Attribute (2)

- Attribute können nicht nur primitive Datentypen sondern auch Klassen sein.
- Es muss darauf geachtet werden das eine Instanz erzeugt wird bevor über das Attribut auf ein Element der Klasse zugegriffen wird.

```
class EineKlasse {
int wert;
}
class Test {
EineKlasse feld = new EineKlasse();
...
}
```

- Bei konstanten Attributen muss der Anfangswert spätestens im Konstruktor gesetzt werden

Instanzvariablen und Klassenvariablen

- Als Instanzvariablen werden alle Attribute bezeichnet, die nicht-statisch sind.
- statische Attribute werden auch Klassenvariablen genannt.
- statische Attribute werden mit `static` definiert.
- Von einem statischen Attribut existiert immer nur eine Kopie, bei der Instanzierung wird das statische Attribut nicht einem Objekt zugeordnet.
- statische Attribute können benutzt werden ohne das ein Objekt der Klasse erzeugt wurde
- außer mit statischen Klassenmethoden kann man nur mittels des Klassennamens auf statische Methoden zugreifen
- Daraus ergibt sich die Möglichkeit des Daten-Austauschs zwischen Objekten einer Klasse über ein statisches Attribut

Methoden

- Wurden bisher zwar schon verwendet, aber noch nicht als Methode einer Klasse sondern eher als Funktion zum Strukturieren des Programmcodes
- Methoden enthalten Anweisungen und die bisher vorgestellten Sprachkonstrukte
- Methoden werden benutzt um den Zustand eines Objektes zu ändern oder um eine Aufgabe des Objekts zu erledigen.
- Syntax der Methodendefinition:

```
[Zugriff][Modifizierer] typDesRückgabewertes
Methodenname([Parameter]) [throws] [<Exception-Liste>]
```

Methoden(2)

- Als Zugriffsspezifizierer sind dieselben zugelassen wie bei den Klassen. Nicht jede Kombination von Zugriffseinschränkungen macht Sinn.
- Als Modifizierer sind erlaubt:
- `Abstract`: Methode die nur als Signatur ohne Anweisungsblock definiert wird
- `Final`: Konstante Methode die nicht überschrieben oder überdeckt werden kann
- `Static`: sog. Klassenmethode; wird über den Klassennamen angesprochen
- `Synchronized`, `strictfp`, `native`: vorerst nicht benötigt
- `throws` und eine Exceptionliste wird für die Fehlerbehandlung benötigt und wird an entsprechender Stelle der Vorlesung beschrieben, vorerst nicht benutzt

- Methoden können innerhalb anderer Methoden aufgerufen werden und können sich auch selbst aufrufen (Rekursion)

Rückgabewerte/Parameter

- Grundsätzlich kann man jeden Typ, also auch eine Klasse als Rückgabewert definieren oder wenn keiner benötigt wird `void`.
- Rückgabe erfolgt mittels return-Anweisung
- Die return-Anweisung beendet auch die Ausführung der Methode
- Hinter dem return kann ein Wert oder ein ganzer Ausdruck stehen

- Parameter werden durch Kommata getrennt angegeben, zuerst der Typ des Parameters dann der Name.
- Wenn Parameter elementare Datentypen sind wird durch die Manipulation der Parameter in der Methode der Ausgangswert nicht geändert. Werden jedoch Objekte als Parameter übergeben wirken sich Änderungen an den Parametern auch auf das Ursprungsobjekt aus da nur eine Referenz übergeben wird.

Vielen Dank für die Aufmerksamkeit!