Methoden

In ihrem formalen Aufbau entsprechen Java-Methoden den Funktionen in C und C++. Eine Methode dient dazu, eine häufig benötigte Folge von Anweisungen unter einem Namen zusammenzufassen und zum Aufruf anzubieten. Eine Methode kann beliebig viele Werte (Übergabe-Parameter) übernehmen, aber nur einen einzigen Rückgabewert zurückgeben. Eine Methode muss explizit aufgerufen werden, damit sie zur Ausführung kommt.

Im Unterschied zu C und C++ müssen Java-Methoden *innerhalb einer Klasse* definiert werden, weil prinzipiell keinerlei externe Deklarationen erlaubt sind. Eine Methode gehört also immer zu einer bestimmten Klasse und kann auch nur von Objekten dieser Klasse benutzt werden. Objekte anderer Klassen können Methoden nur unter ganz bestimmten Umständen nutzen – dann nämlich, wenn es sich um static-Methode handelt und wenn eine Assoziation zwischen den Klassen besteht (siehe Kapitel Klassenbeziehungen).

```
class Student {

    String name;
    String matrikel;

    void setName( String n ) {

            this.name = n;
    }


}
```

In Java müssen Methoden prinzipiell innerhalb einer Klasse deklariert und zugleich auch definiert werden

Es bestehen weitere Unterschiede, die in der folgenden Tabelle zusammengefaßt dargestellt wurden

C-Funktion	C++-Funktion	Java-Methode
eigenständiger Anweisungsblock außerhalb von main()	Eigenständiger Anweisungsblock *Oder:* Methode einer Klasse	Nur innerhalb einer Klasse möglich
void-Funktion darf kein return enthalten	Void-Funktion darf kein return enthalten	void-Methode darf return enthalten
Übergabe wahlweise per-value oder per-reference	Übergabe wahlweise Per-value oder per-reference	Übergabe durch Datentyp festgelegt
wird als eigenständige Funktion aufgerufen benötigt kein Objekt	wird als eigenständige Funktion aufgerufen Oder: Benötigt Objekt	Eigenständige Methode muss als static gekennzeichnet werden, Ansonsten Objekt benötigt

Da eine Methode immer zu einer bestimmten Klasse gehört, kennt sie grundsätzlich nur die Attribute und Methoden, die zu dieser Klasse gehören. Methoden anderer Klassen können nur unter bestimmten Umständen aufgerufen werden:

- Wenn eine Assoziation zu der anderen Klasse besteht
- Static-Methoden können über den Klassennamen aufgerufen werden,
- Methoden innerhalb der Klassen-Hierarchie (`super.methode()`).

Eine Objekt-Methode wird über den **Objektnamen** aufgerufen

```
Objekt.methode();
```

Eine Klassen-Methode (static-Methode) wird über den **Klassennamen** aufgerufen

```
Klasse.methode();
```

Methodenaufruf - Signatur

Der Methodenaufruf muss exakt der Signatur der Methode entsprechen. Die Signatur einer Methode besteht aus dem Namen und den Übergabeparametern. Der Rückgabetyp gehört nicht dazu.

```
void zeige();
void zeige(String name);
int  zeige(String name); // Fehler!
```

In diesem Beispiel werden die ersten beiden Methoden einer Klasse als verschiedene und getrennt voneinander aufrufbare Methoden gewertet. Die dritte Methode, die sich nur durch den Rückgabetyp (z.B. int) unterscheidet und ansonsten die gleichen Parameter hat, wird vom Java-Compiler nicht akzeptiert, sondern als verbotene Redeklaration einer vorhandenen Methode betrachtet.

Übergabe-Parameter

In Java gibt es keine Möglichkeit, die Art der Parameter-Übergabe beim Methodenaufruf (**per-value** oder **per-reference**) selbst zu bestimmen, da es keinen Adress- und Pointeroperator gibt. Java orientiert sich prinzipiell an der Art des übergebenen Datentyps (Werttyp oder Objekttyp) und legt die Übergabeart entsprechend fest.

a.) Übergabe eines Werttyps

Die Parameter-Übergabe eines elementaren Werttyps (int, double, char, boolean: siehe folgendes Kapitel) erfolgt automatisch **call-by-value**, d.h. es wird eine lokale Kopie des Originals erstellt. Wenn in der Methode dieser Wert verändert wird hat dies keine automatischen Auswirkungen auf das Original. Werttypen müssen deshalb immer per Rückgabe (return) an den Aufrufer zurückgegeben werden, wenn Veränderungen dauerhaft sein sollen.

b.) Übergabe eines Objektdatentyps

Objektdatentypen werden automatisch **per-reference** übergegeben. Das bedeutet, dass Veränderungen an dem Objekt nicht per Rückgabewert an den Aufrufer zurückgegeben werden müssen, sondern sofort ausgeführt werden.

Rückgabetyp

In Java darf eine **`void`**-Methode eine **`return`**-Anweisung enthalten, die die Ausführung des Anweisungsteils abbricht. Die return-Anweisung darf jedoch keinen Wert zurückgeben.

Beispiel

```
class IntegerTest {

    static void pruefen(int j) {  // void-Methode mit int-Parameter

        if (j == 0) {
            System.out.println("Falsch: Sie haben 0 eingegeben");
            return;                 // Methode abbrechen
         }

         System.out.println("Sie haben eine korrekte Zahl eingegeben: "+j);
    }
    ////////////////////////////////////////////////

    public static void main(String a[]) {

        System.out.println("Geben Sie eine Zahl (keine NULL) ein");
        int i = In.readInt();
       pruefen(i);     // Funktionsauruf
    }
}
```

Wenn ein Rückgabetyp angegeben wird, muss der `return`-Typ diesem genau entsprechen. Es wird **keine implizite Typkonvertierung** vorgenommen, die zu einem fehlerhaften Ergebnis führen könnte, sondern der Compiler meldet einen Fehler.

```
class DoubleTest {

    static int pruefen(double j) {
         System.out.println("Sie haben eingegeben: " + j);
         return j;
    }

    public static void main(String a[]) {
        System.out.println("Geben Sie eine Dezimalzahl ein");
        double i = In.readDouble();
        System.out.println(pruefen(i));
    }
}
```

`ERROR: Cannot implicitly convert double to int`
`Oder : possibly loss of precision`

Anmerkung: Der hier gezeigte Quelltext verwendet zum Einlesen die Komfortklasse "In", die in einem späteren Kapitel genauer beschrieben wird. Zum erfolgreichen Übersetzen muss die Klasse "In.class" in das aktuelle Arbeitsverzeichnis kopiert werden.

Mehrere gleichnamige Methoden

Innerhalb einer Klasse darf es mehrere gleichnamige Methoden geben, wenn diese sich durch ihre Signatur unterscheiden. Ein unterschiedlicher Rückgabe-Typ alleine genügt nicht als Unterscheidungskriterium. Der Compiler sucht die passende Methode und führt diese aus. Dies gilt auch für KONSTRUKTOREN.

```
class integerTest {

    static void pruefen(double j) {
         System.out.println("Sie haben double eingegeben: " + j);
         return;
    }

    static void pruefen(int j) {
         System.out.println("Sie haben int eingegeben: " + j);
         return;
    }
    static void pruefen(int j, double k) {
         System.out.println("Sie haben INT eingegeben: " + j);
         System.out.println("und DOUBLE " + k);
         return;
    }
    public static void main(String a[]) {

        System.out.println("Geben Sie eine Ganzzahl‟);
        System.out.println(„ und eine Dezimalzahl ein");
        int i    = In.readInt();
        double j = In.readDouble();
        pruefen(i);
        pruefen(j);
        pruefen(i, j);
    }
}
```