Objekt-Datentypen

Alle Variablen, die keine elementaren Datentypen sind, werden in Java grundsätzlich als Objekte behandelt. Dies gilt insbesondere für alle Arrays, selbst wenn diese aus elementaren Datentypen bestehen (siehe Kapitel Arrays).

Für Objekte gilt übrigens dasselbe *Deklarationsprinzip* wie für elementare Datentypen. Ein Objektdatentyp, der auf Klassenebene deklariert wird, ist ein Attribut. Wenn ein Objektdatentyp innerhalb einer Methode deklariert wird, dann handelt es sich um ein lokales Objekt, das nur im Gültigkeitsbereich der Methode bekannt ist.

```
class Student {

      int matrikelnr;
      String name;              // Objekt-Attribut:
                                // "Student.name"
      void tuWas() {
        String irgendwas = „blabla“; // Lokale Variable
        ...
    }
}
```

Ein häufig verwendeter Objektdatentyp ist beispielsweise String. Für Zeichenketten gelten besondere Regeln, die den Umgang wesentlich erleichtern (siehe Kapitel Zeichenketten). Strings müssen nicht mit `new` erzeugt werden und können direkt initialisiert werden. Diese Erleichterung gilt nicht für andere Objektdatentypen.

Für Objektdatentypen gelten insgesamt dieselben Regeln wie für alle selbstdefinierten Objekte. Ein Objektdatentyp wird nie direkt, sondern immer über eine **Referenz** (den Objektnamen) angesprochen. Die Referenz beinhaltet nicht das Objekt, sondern sie verweist nur auf die individuelle Adresse im dynamischen Speicher (heap), an der sich die Objektdaten befinden. Über den Objektnamen kann auf die Attribute und Methoden des Objekts zugegriffen werden.

Übergabe von Objekten an eine Methode

Da es in Java keinen Pointer- und Adressoperator gibt, entfällt auch die Möglichkeit, bei einem Methodenaufruf die Art der Argumentübergabe zu bestimmen. In Java werden Argumente voreinstellungsgemäß stets **per-Value** (als Wertkopie) übergeben. Es gibt keine Möglichkeit, diese Voreinstellung programmtechnisch zu verändern.

Allerdings wirkt sich die Übergabe per-Value unterschiedlich aus, je nachdem ob es sich um einen Wert- oder Objektdatentyp handelt. Elementare Datentypen (Wertdatentypen) werden immer als Wertkopie übergeben, d.h. Veränderungen, die in der Funktion vorgenommen werden, haben prinzipiell keine Rückwirkung auf das Original (siehe voriges Kapitel).

Objektdatentypen hingegen bestehen bekanntlich aus zwei Bestandteilen, der Referenzvariablen und den eigentlichen Objektdaten. Dementsprechend lassen sich zwei verschiedene Auswirkungen bei der Argumentübergabe unterscheiden.

1. Änderungen, die innerhalb der Methode an der Referenz (und nicht am Objekt) vorgenommen werden (z.B. globale Neuzuweisung oder NULL-Zuweisung), haben keine Auswirkungen auf das Original-Objekt. Nach Verlassen der Funktion ist das ursprüngliche Objekt unverändert über seinen Namen ansprechbar.

```
void tauschen( Objekt a , Objekt b) {

     a = null;     // nicht
     b = null;     // permanent
}
```

```
class Objekt {
    int i;

    public Objekt(int i) {
        this.i = i;
    }

    public void show() {
        System.out.println(i);
    }

    ////////////////////////////////////////////////////////
    public static void main(String args[]) {

        Objekt eins = new Objekt(1);
        Objekt zwei = new Objekt(2);

        System.out.println("Vor Funktion:");
        eins.show();
        zwei.show();

        tausche(eins,zwei);

        System.out.println("Nach Funktion:");
        eins.show();
        zwei.show();
     }

     ////////////////////////////////////////////////////////
     public static void tausche(Objekt a, Objekt b) {

            Objekt temp= a;
            a = b;              // Zuweisung
            b = temp;
            System.out.println("In Funktion:");
            a.show();
            b.show();
      }
}
```

```
Vor Funktion:
1
2
In Funktion:
2
1
Nach Funktion:
1
2
```

2. Wird in der Funktion hingegen auf Attribute oder Methoden des Objekts zugegriffen, dann sind die Änderungen auch nach Verlassen der Funktion tatsächlich gültig. Der Hintergrund ist, dass hierbei auf die Objektdaten (und nicht nur auf den Objektnamen) zugegriffen wird.

```
void tauschen( Objekt a , Objekt b) {

     a.i = 1;      // permanent
     b.i = 2;
}
```

```
class Objekt {

    private int i;

    public Objekt(int i) {
        this.i = i;
    }

    public void show() {
        System.out.println(i);
    }

    ////////////////////////////////////////////////////////
    public static void main(String args[]) {

        Objekt eins = new Objekt(1);
        Objekt zwei = new Objekt(2);

        System.out.println("Vor Funktion:");
        eins.show();
        zwei.show();

        tausche(eins,zwei);

        System.out.println("Nach Funktion:");
        eins.show();
        zwei.show();
     }

     ////////////////////////////////////////////////////////

     public static void tausche(Objekt a, Objekt b) {

            int temp = a.i;
            a.i      = b.i;
            b.i      = temp;

            System.out.println("In Funktion:");
            a.show();
            b.show();
      }
}
```

```
Vor Funktion:
1
2
In Funktion:
2
1
Nach Funktion:
2
1
```