Verschiedene Einteilungen (Kategorien) von Variablen

1 Einteilung von Variablen anhand ihres Gültigkeitsbereichs

1.1 Objektattribute (Objektvariablen) versus Klassenattributen (Klassenvariablen)

Im Allgemeinen werden Attribute ganz am Anfang einer Klasse definiert. Die Attribute einer Klasse beschreiben den Zustand (die Daten) einer Klasse bzw. der einzelnen Objekte der Klasse. Man unterscheidet hierbei Objektattribute (Objektvariablen) von Klassenattributen (Klassenvariablen).

1.1.1 Objektattribute (Objektvariablen bzw. Instanzvariablen)

- Werden ohne das Schlüsselwort `static` definiert.
- werden für jedes einzelne Objekt einer Klasse angelegt und können somit in jedem Objekt einen anderen Wert haben.
- werden außerhalb von Methoden angelegt und sind deshalb in allen Methoden der Klasse sichtbar.
- sind erst verfügbar, nachdem ein Objekt (eine "Instanz") der entsprechenden Klasse erzeugt wurde.
- zur Umsetzung des Geheimnisprinzips werden Objektattribute im Allgemeinen mit dem Zugriffsmodifizierer `private` definiert.

Beispiel: Die Objektattribute `sName` und `sVorname` der Klasse `Kunde`:

```
private String sName;
private String sVorname;
```

1.1.2 Klassenattribute (Klassenvariablen)

- Werden mit dem Schlüsselwort `static` definiert.
- werden für die ganze Klasse nur einmal angelegt und haben somit für alle Objekte einer Klasse den selben Wert.
- werden außerhalb von Methoden angelegt und sind deshalb in allen Methoden der Klasse sichtbar.
- sind auch verfügbar, wenn noch kein Objekt der entsprechenden Klasse erzeugt wurde.
- zur Umsetzung des Geheimnisprinzips werden Klassenattribute im Allgemeinen mit dem Zugriffsmodifizierer `private` definiert.
- sie werden im UML Klassendiagramm durch Unterstreichung des Namens gekenntzeichnet.

Beispiel: Anzahl Kunden der Klasse `Kunde`:

```
private static int iAnzahlKunden = 0;
```

1.1.3 Lokale Variablen

- werden innerhalb einer Methode angelegt, sie sind nur in dieser Methode sichtbar.
- werden erst angelegt, wenn die betreffende Methode ausgeführt wird. Nach Verlassen der Methode sind sie wieder ungültig.

2 Einteilung nach der Art des Zugriffs: Variablen einfacher Datentypen und Variablen von Referenztypen

In Java existieren zwei große Gruppen von Datentypen: einfache (oder primitive) Datentypen sowie Referenztypen (oder Objektdatentypen).

2.1 Variablen einfacher Datentypen

Wie bereits im zugehörigen Dokument beschrieben gibt es in Java 8 einfache oder "primitive" Datentypen. Diese zeichnen sich dadurch aus, dass sie eine genau festgelegte Speichergröße haben, die auf allen Plattformen gleich ist.
Der Wert, der einer entsprechenden Variablen zugewiesen wird, wird direkt in der Speicherzelle der Variablen gespeichert.

2.2 Variablen von Referenztypen (Referenzvariablen)

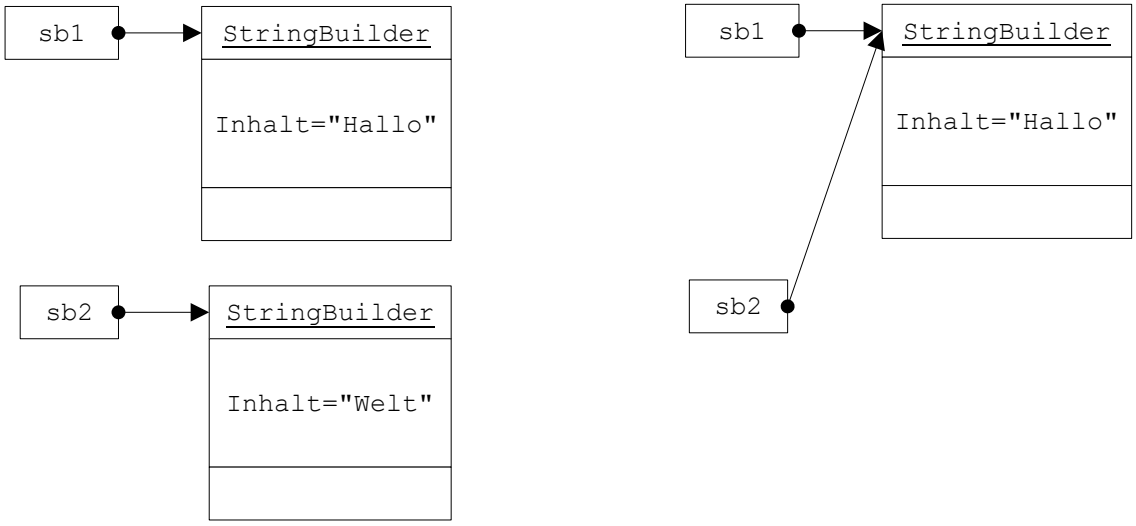
Referenztypen sind Datentypen, auf die nur indirekt über eine "Referenz", also einen Verweis zugegriffen werden kann. Variablen von Referenztypen (sogenannte Referenzvariablen) enthalten somit nicht den Wert einer Variablen, wie bei einfachen Datentypen, sondern den Verweis auf den Speicherplatz eines Objektes.

Referenzvariablen werden in Java für Objekte, Strings und Arrays[1] verwendet. Die Konstante `null` stellt eine leere Referenz dar. Eine Referenzvariable deren Inhalt `null` ist, verweist aktuell ins „Nichts“.
Ein Beispiel soll den Zusammenhang erläutern:

```
   public class ReferenzDemo
   {
     public static void main (String [] args)
     {
1      StringBuilder sb1, sb2;   // Deklaration von Referenzvariablen
2      sb1 = new StringBuilder ("Hallo");
3      sb2 = new StringBuilder ("Welt");
4      sb2 = sb1;
5      sb1.replace (1, 3, "_sb1_");
6      System.out.println ("sb2: " + sb2); // "sb2: H_sb1_lo"
     }
   }
```

1	Deklaration der Referenzvariablen sb1 und sb2 vom Typ StringBuilder; Inhalt ist zunächst `null`
2	Erzeugung eines neuen StringBuilder-Objekts mit dem Inhalt `"Hallo"`. Die Adresse (=Referenz) des Objekts wird der Referenzvariablen `sb1` zugewiesen.
3	dito mit `sb2`.
4	Der Inhalt von `sb1` (also die Adresse des StringBuilder-Objekts mit dem Inhalt `"Hallo"`) wird `sb2` zugewiesen. Damit verweisen beide Referenzvariablen auf dasselbe Objekt (siehe Schaubild unten).
5	Wird nun der Inhalt des Objekts, auf das `sb1` verweist, verändert (z.B. durch Einfügen eines Textes – hier `"_sb1_"`), so hat dies auch Auswirkung auf `sb2`, da diese Referenzvariable ja auf dasselbe Objekt zeigt.

[1] Anmerkung: Strings und Arrays sind zwar auch Objekte, nehmen aber eine gewisse Sonderstellung ein. Zu Strings siehe Merkblätter "Arbeiten mit Standardklassen" und "Datentypen", zu Arrays später.




```
StringBuilder sb1 = new StringBuilder ("Hallo");
StringBuilder sb2 = new StringBuilder ("Welt");
```

```
sb2 = sb1;
```

3 Garbage Collector

Das Beispiel aus Kapitel 2.2 lässt die Frage aufkommen, was denn mit dem `StringBuilder`-Objekt passiert, auf das `sb2` ursprünglich zeigte.

Dieses Objekt (mit dem Inhalt `"Welt"`) existiert immer noch im Hauptspeicher, wird jedoch von keiner Referenzvariablen mehr referenziert, d.h. es ist nicht mehr auffindbar und damit "Müll". Damit durch solche Programme der Speicher nicht langsam aber sicher zuge"müll"t wird, gibt es in Java den sog. *Garbage Collector*, der von Zeit zu Zeit diesen Müll einsammelt, sprich den von nicht mehr referenzierten Objekten belegten Speicherplatz wieder frei gibt. Der Programmierer braucht sich nicht um die Freigabe von Speicherplatz zu kümmern.

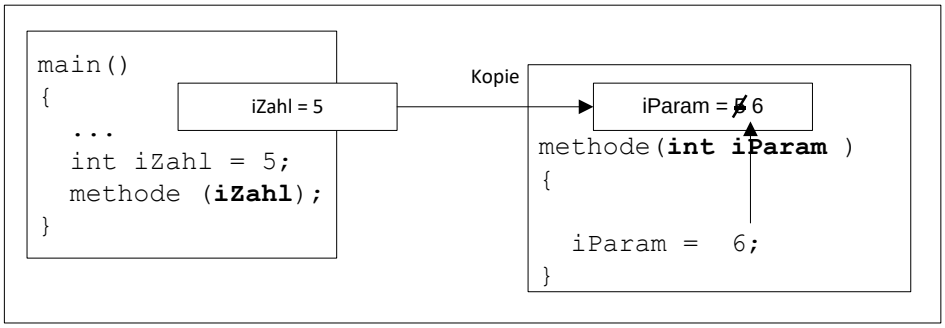
4 Übergabe an Methoden

Werden an eine Methode Parameter übergeben, so werden diese ***grundsätzlich als Kopie*** übergeben. Das bedeutet, dass der Wert der Variablen an einen anderen Speicherplatz im Hauptspeicher kopiert wird. Dieses Verfahren heißt ***call by value***. Es gilt unabhängig vom Datentyp des Parameters für alle einfachen Datentypen und für die Referenztypen.
In Bezug auf die Auswirkungen der Übergabe von einfachen Datentypen und Referenztypen gibt es jedoch einen wesentlichen Unterschied.

4.1 Call by value – einfache Datentypen

Werden Parameter mit einfachen Datentypen übergeben, so gilt call by value "in Reinform":

Veränderungen des Inhalts einer Parametervariablen innerhalb der Methode wirken sich nur auf die Kopie aus und sind daher ***für den Aufrufer der Methode nicht sichtbar.***



Beispiel:

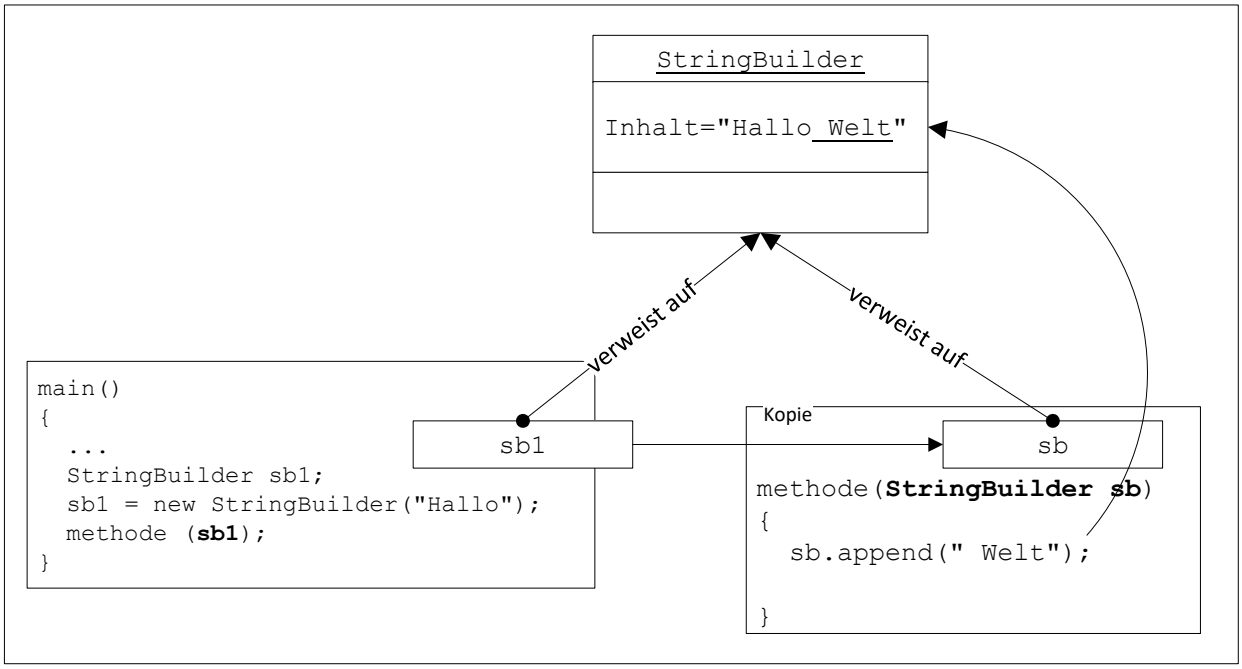
```
public class CallByValueDemo
{
  public static void main(String[] args)
  {
  /*
   * Einfache Datentypen werden als Kopie (CallByValue) übergeben.
   * Daher ist die Veränderung in der aufrufenden Methode nicht sichtbar:
   */
    int iZahl = 5;
    vCallByValue(iZahl);
    System.out.println("call by value: "+iZahl);
  }
  public static void vCallByValue(int iParam)
  {
    /*
     * die übergebene Variable wird verändert:
     */
    iParam = 6;
  }
}
```

Ausgabe:

```
call by value: 5
```

4.2 Call-by-value – mit Referenzdatentypen

Werden Parameter mit Referenzdatentypen übergeben (Objekte, Arrays), so gilt ebenfalls Call-by-value, d.h. es wird eine Kopie des Wertes der Referenzvariablen übergeben.
Da die Referenzvariable jedoch ein Verweis auf das Originalobjekt ist, wirken sich Veränderungen an dem Objekt natürlich direkt auf das Originalobjekt aus und sind somit ***für den Aufrufer der Methode sichtbar***.



Die Übergabe von Objekten an Methoden hat damit zwei wichtige Konsequenzen:

- die Methode erhält zwar eine Kopie der Referenz, diese verweist aber auf ein und das selbe Objekt, damit arbeitet die Methode mit dem Originalobjekt
- die Übergabe von Objekten ist performant, gleichgültig wie groß sie sind.
- *Achtung Fehlerquelle*: Da die aufgerufene Methode mit dem Originalobjekt arbeitet, kann sie dieses verändern, ohne dass der Aufrufer es merkt. Soll dies verhindert werden, muss das Objekt vorher mit der Methode clone() dupliziert werden

Beispiel:

```
public class ReferenceDemo
{
  public static void main(String[] args)
  {
  /*
   * Bei Referenzdatentypen wird als Wert ein Verweis
   * übergeben. Daher ist die Veränderung auch in der
   * aufrufenden Methode sichtbar:
   */
    StringBuilder sb1 = new StringBuilder("Hallo");
    methodeReference(sb1);
    System.out.println("Ergebnis: "+sb1.toString()); // Hallo Welt
  }

// Fortsetzung nächste Seite
```

```
  public static void methodeReference(StringBuilder sb)
  {
    /*
     * das referenzierte Objekt wird verändert:
     */
    sb.append(" Welt");
  }
}
```

Ausgabe:

```
Ergebnis: Hallo Welt
```