Methoden in C#: Werte- und Referenzparameter

Schauen wir uns folgendes Programmstück an:

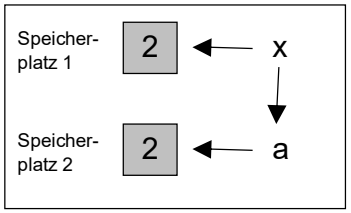
```
1  void vertausche(int a, int b){
2        int hilf=a; a=b; b=hilf;
3  }
4
5  void Button1Click(object sender, System.EventArgs e)
6  {
7        int x=2;
8        int y=3;
9        vertausche(x, y);
10       MessageBox.Show(x.ToString()+"  "+y.ToString());
11 }
```

Dieses Programmstück legt ein merkwürdiges Verhalten an den Tag. Im Meldungsfenster wurde 2 3 ausgegeben statt 3 2.

Die Werte der Variablen x und y wurden in Zeile 7 *also nicht vertauscht*, obwohl die Methode `vertausche` (Zeilen 1-3) logisch richtig programmiert worden ist. Wie man schnell nachprüfen kann, wurden die vertauschten Werte lediglich in den lokalen Variablen a und b innerhalb der Methode, nicht aber in den Variablen x und y in Button1Click gespeichert!

Was ist passiert?

In dem Moment, in dem man der Methode vertausche die Werte von x und y als aktuelle Parameter übergeben hat (Zeile 9), werden Kopien von x und y angefertigt, die innerhalb der Methode vertausche verändert werden. Die Werte der originalen Variablen x und y bleiben dagegen unverändert!



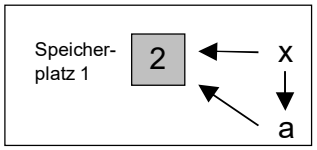
a ist Kopie von x

Diese Art des Methodenaufrufes nennt man **call by value** und die Parameter heißen in diesem Fall **Werteparameter**.

Standardmäßig wird bei einfachen Datentypen (wie **`int`** oder **`double`**) beim Methodenaufruf immer mit einer Kopie der Daten gearbeitet.

Lösung des Problems:

Man muss erreichen, dass die Methode `vertausche` die Werte der originalen Variablen x und y direkt verändern kann. Dazu übergibt man nach Aufrufen der Methode vertausche keine Kopie der Variablen, sondern eine Referenz auf deren Speicherplatz. Damit erreicht man, dass keine weiteren Speicherplätze bereitgestellt werden müssen. Vielmehr wurde ein anderer Name für dasselbe Objekt erstellt.



a ist nur ein anderer Name für das Objekt

Diese Art des Methodenaufrufes nennt man **call by reference** und die Parameter heißen in diesem Fall **Referenzparameter**.

Das obige Programm sieht damit so aus:

```
1  void vertausche(ref int a,ref int b){
2        int hilf=a; a=b; b=hilf;
3  }
4
5  void Button1Click(object sender, System.EventArgs e)
6  {
7        int x=2;
8        int y=3;
9        vertausche(ref x,ref y);
10       MessageBox.Show(x.ToString()+"  "+y.ToString());
11 }
```

Ein Referenzparameter wird in C# mit Hilfe des Schlüsselwortes **`ref`** gekennzeichnet. Dieses muss an zwei Stellen vorhanden sein:

1. im Methodenkopf vor jedem Referenzparameter (hier: in Zeile 1)
2. beim Methodenaufruf vor jeder Variablen, deren Referenz übergeben werden soll (hier: in Zeile 9)

Wann und wie verwendet man Referenzparameter?

1. Die übergebene Variable, bzw. deren Wert, soll innerhalb der Methode direkt geändert werden.
2. Es muss immer eine Variable übergeben werden. Diese muss zudem bereits initialisiert worden sein, d.h. einen Wert haben. Es können keine Ausdrücke übergeben werden! Im obigen Beispiel würde der Methodenaufruf `vertausche(8, 5);` zu einer Fehlermeldung beim Kompilieren führen.

Hinweise für Fortgeschrittene:

- Komplizierte Datentypen (z.B. Klassen) werden aus Performancegründen immer als Referenzparameter an Methoden übergeben.
- C# kennt noch weitere Arten von Parametern: sogenannte out-Parameter und params-Parameter, auf die wir aber in diesem Kurs nicht eingehen werden.

Aufgaben:

Hinweis: Man braucht nicht bei allen Aufgaben Referenzparameter.

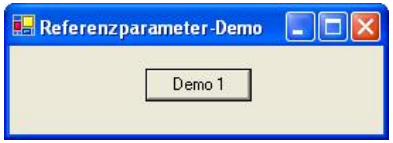
1. Schreibe eine Methode increment, die eine eingegebene Zahl um eins erhöht und diese Zahl in einer MessageBox wieder ausgibt.
   In der Methode soll nur das Erhöhen stattfinden. Das Einlesen und die Ausgabe soll in der Ereignisbehandlung des Buttons stattfinden.
2. Schreibe das Programm zum Heron-Verfahren so um, das die eigentlichen Berechnungen in einer eigenen Methoden namens `heron` ablaufen. Überlege dir vor dem Programmieren, welche Parameter notwendig sind und ob ein Wert zurückgegeben werden soll. Wenn es einen Rückgabewert gibt, welcher Datentyp hat er dann?
3. Was passiert, wenn man im obigen Beispiel die Zeile 9 durch folgende Zeile ersetzt? `vertausche(ref 2,ref 3);`

4. Was wird beim folgenden Programm ausgegeben, wenn der Button gedrückt wird? Erkläre anhand des Quellcodes, wie es zu dieser Ausgabe kommt:

```
int a,b,c;

void loesch(int d, ref int e){
  a=0;
  d=0;
  e=0;
}

void Button1Click(object sender, System.EventArgs e)
{
  a=5;
  b=10;
  c=20;
  loesch(b,ref c);
  MessageBox.Show(a.ToString()+" "+b.ToString()+" "+c.ToString());
}
```



5. Schreibe ein Programm mit einer Methode min, in der das Minimum aus drei übergebenen Zahlen bestimmt wird.
6. Betrachte nochmals das Sortierverfahren „Selection Sort“. Schreibe das Programm so um, dass sinnvolle Methoden verwendet werden.
7. Schreibe ein C#-Programm, das die Oberfläche und das Volumen von bestimmten Körpern berechnet. Verwende dazu entsprechende Methoden.
8. Schreibe ein Programm, das die Quersumme einer Zahl innerhalb einer Methode berechnet.
9. Schreibe ein Programm, das einen Bankautomaten simuliert. Durch die Methoden `abheben` und `einzahlen` soll der Kontostand um einen beliebigen Betrag geändert werden. Ein gewisser Kontostand sei dabei vorgegeben.
   Idee: `kontostand` soll eine globale Variable sein. Die Methoden `abheben` und `einzahlen` sollen jeweils einen Parameter besitzen, der für den entsprechenden Geldbetrag steht.

Zur Erinnerung:

„Globale Variablen“:

- sind Variablen, die außerhalb von Methoden definiert sind.
- gelten für die gesamte Klasse, d.h. in unseren einfachen Beispielen für das gesamte Programm.
- Für Fortgeschrittene: Eigentlich gibt es in C# keine globalen Variablen im klassischen Sinn, da sie immer zu Klassen bzw. Objekten gehören.

Lokale Variablen:

- sind Variablen, die in einem Block definiert sind.
- sind nur in dem Block gültig, in dem sie definiert sind, ebenso wie in allen darin eingeschlossenen Blöcken.
- überdecken globale Variablen mit dem gleichen Namen.

