Methoden in C#

Bisher sprachen wir ausschließlich darüber, wie man den Ablauf eines Programms durch Verzweigungen und Schleifen steuert. Schon bei der Hamsterprogrammierung früher erkannten wir die Vorteile von eigenen Methoden. Auch bei der Einführung in die Objektorientierten Programmierung sahen wir, dass jedes Objekt Eigenschaften und Methoden bzw. Operationen besitzt. In Methoden kann jedes Objekt entsprechend der Werte seiner Eigenschaften handeln. Methoden ähneln den Funktionen aus der Mathematik.

Wir haben bereits vorhandene Methoden verwendet, z. B. `Parse(…)` oder `ToString()` oder vorgegebene „Methodengerüste“ bei Ereignisbehandlungen im Leben erfüllt.

Heute und in den nächsten Stunden geht es darum, eigene Methoden zu erstellen. Wir beginnen mit dem einfachsten Fall:

Methoden ohne Parameter

Alle Methoden werden immer **innerhalb einer Klasse** erstellt. Methoden ohne Parameter haben den folgenden Aufbau:

```
<Rückgabetyp> <Methodenname>()                    Methodenkopf
{
  // Code, der in der Methode ausgeführt wird     Methodenrumpf
}
```

Unter `<Rückgabetyp>` einer Methode versteht man den Datentyp, den die Methode zurückliefern soll. In der Mathematik liefert z. B. die Funktion sin(x) ein Ergebnis vom Typ double zurück. Wenn man eine Methode keinen Wert zurückliefern soll, schreibt man als Typ „`void`“.
Der `<Methodenname>` ist ein Bezeichner (Regeln wie bei Variablen: siehe früher), unter dem die Methode von C# erkannt werden soll.

Beispiel 1: Formular mit Button button1

```
1  void sageHallo(){
2    for(int i=1;i<=3;i++){
3      MessageBox.Show("Hallo!");
4    }
5  }
6
7  void Button1Click(object sender, System.EventArgs e){
8    MessageBox.Show("Button 1 wurde geklickt!");
9    sageHallo();
10   MessageBox.Show("wieder zurück in Button1Click");
11 }
```

Schauen wir uns das Beispiel ein wenig genauer an:

- In den Zeilen 1 bis 5 befindet sich unsere eigene Methode mit dem Namen `sageHallo` mit keinem Parameter und keinem Rückgabewert (erkennbar an `void`). Eine Methode ohne Parameter dient dazu, ein bestimmtes Unterprogramm, das mehrmals in genau der gleichen Weise ablaufen soll, nur einmal programmieren bzw. schreiben zu müssen.
- In der Methode selbst wird dreimal ein Meldungsfenster mit "Hallo" angezeigt.
- In den Zeilen 7 bis 11 befindet sich die Ereignisbehandlung für `Button1Click`, die aufgerufen wird, wenn Button1 gedrückt wird. Zunächst wird ein Meldungsfenster mit dem Text "Button 1 wurde geklickt" ausgegeben.
- In der Zeile 9 wird der Ablauf der Ereignisbehandlung gestoppt und unsere eigene Methode sageHallo aufgerufen. Eine Methode ohne Parameter wird aufgerufen, in dem man ihren Namen und direkt anschließend runde Klammer schreibt. Wenn ihre Abarbeitung beendet ist, wird die Ereignisbehandlung direkt nach dem Methodenaufruf fortgeführt und ein weiteres Meldungsfenster ausgegeben.

Beispiel 2: Formular mit Button button1

```
1  String liefereUhrzeit(){
2        String d;
3        d = DateTime.Now.ToString();
4        return d;
5  }
6
7  void Button1Click(object sender, System.EventArgs e){
8        String s=liefereUhrzeit();
9        MessageBox.Show(s);
10 }
```

- In diesem Beispiel heißt unsere eigene Methode liefereUhrzeit und gibt einen Wert vom Typ String zurück, was man in Zeile 1 erkennt.
- In Zeile 3 wird das aktuelle Datum und die Uhrzeit bestimmt und im String d gespeichert.
- Hinter dem **`return`** in Zeile 4 steht der Wert, der zurückgegeben wird. Statt der Variablen d kann auch ein beliebiger Ausdruck stehen, der aber den im Methodenkopf angegebenen Datentyp ansprechen muss. Wichtig ist, dass **`return`** die letzte in der Methode ausgeführte Anweisung sein muss!

Aufgabe 1:

a) Kompiliere das Programm. Was wird ausgegeben?
b) Lösche die Zeile 4 und versuche das Programm zu kompilieren. Was passiert?
c) Tippe Zeile 4 wieder ein und füge direkt darunter eine weitere Zeile mit `MessageBox.Show("Ausgabe!");` ein. Kompiliere das Programm und führe es aus. Überprüfe außerdem die Meldungen beim Kompilieren. Notiere, was du liest.
d) Entferne die zusätzliche Zeile aus dem c)-Teil und ändere die Methode so, dass als Ausgabe z. B. "Jetzt ist: 23.2.2005 14:00:10" zurückgeliefert wird.
e) Was passiert, wenn du in der Methode `liefereUhrzeit` die Variable d in s umbenennst? *Wichtig: Bitte jedes Auftreten von d ersetzen!*
f) Wie kannst du die Methode so kurz wie möglich formulieren?

Regeln für Methoden:

- **Block-Regel:**

Block:	Bereich zwischen { und }
Gültigkeitsbereich:	Eine Konstante bzw. Variable ist in dem Programmblock gültig, in dem sie vereinbart wurde, sowie in allen untergeordneten Blöcken.
Globaler Name/ Instanzvariable oder Datenfeld:	Eine Variable, die außerhalb von Methoden definiert wird. Sie hat in der gesamten Klasse Gültigkeit. Global heißt „in allen untergeordneten Methoden und Programmteilen bekannt und gültig“. In der Objektorientierten Programmierung spricht man lieber von Instanzvariable oder Datenfeld.
Lokaler Name:	Eine Konstante bzw. Variable, die in einem Block vereinbart wurde. Diese hat nur innerhalb dieses Blockes ihre Gültigkeit. Wir kennen dieses Verhalten von Schleifen. Wenn innerhalb des Schleifenrumpfes eine Variable definiert wird, kann man außerhalb dieses Schleifenblockes nicht mehr auf sie zugreifen.

- **Ausblenden-Regel:** Treten in einem Programm globale und lokale Größen mit gleichem Namen auf, so gilt innerhalb einer Methode nur der **lokale Name**. Die globale Größe wird vorübergehend ausgeblendet (ignoriert). Nach Verlassen der Methode hat die globale Größe noch ihren vorherigen Wert.
- Methoden können **nicht ineinander geschachtelt** werden.

Schnittstelle zwischen Methode und übergeordnetem Programm:

Um eine sauber definierte Schnittstelle zwischen Methode und übergeordnetem Programm zu erhalten, sollten innerhalb einer Methode möglichst keine bzw. wenige globale Variablen verwendet werden. Auf diese Weise können unerwünschte Nebeneffekte, d.h. nicht geplante Veränderungen der Werte der Variablen, vermieden werden. Außerdem erhöht sich dadurch die Wartbarkeit von Programmen.

Deshalb verwenden wir folgende Regel für Variablen und Konstanten:

So lokal wie möglich - nur so global wie unbedingt nötig.

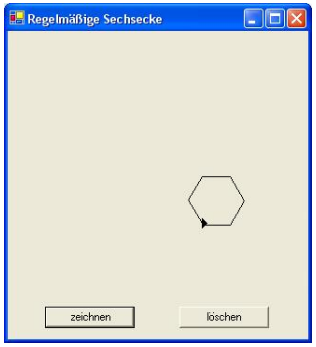
Warum erstellt man Methoden?

Anhand von Methoden oder allgemein Unterprogrammen kann man größere Programme besser aufbauen. Die Vorteile dieser Vorgehensweise sind zum Beispiel:

1. Größere Ökonomie:	Ein Unterprogramm kann an mehreren Stellen eines Programms aufgerufen werden, d.h. es muss nur einmal implementiert werden.
2. Übersichtlichere Gliederung:	Ein Programm wird durch Unterprogramme gegliedert und damit leichter verständlich.
3. Wiederverwendbarkeit:	Ein Unterprogramm kann als Baustein in mehreren Programmen eingebunden werden. Die Bausteine werden in C# in einer Assembly (früher: DLL) gesammelt.

Beispiel 3: „Refactoring“

Ein Programm mit den InfoKursTools zeichnet mit Hilfe der Turtle ein Sechseck:




```
void Button1Click(object sender, System.EventArgs e){
  // Startposition setzen
  turtle1.MoveTurtleTo(200,200);
  // regelmäßiges Sechseck zeichnen
  for(int i=1;i<=6;i++)
  {
    // zeichne eine Seite
    turtle1.DrawTurtle(30);
    // Innenwinkel im 6-Eck ist 120°
    // der Außenwinkel ist hier 180° - 120°= 60°
    // Dieser muss fuer die Drehung angegeben werden
    turtle1.TurnLeft(60);
  }
}

void Button2Click(object sender, System.EventArgs e){
  turtle1.Clear();
}
```

Jetzt stellt man fest, dass man nicht ein, sondern mehrere Sechsecke zeichnen möchte. Die einfachste und unflexibelste Möglichkeit ist, dass man die Schleife so oft kopiert, wie sie benötigt wird. Eleganter geht es mit Hilfe einer Methode:

```
void zeichneSechseck()
{
   // regelmäßiges Sechseck zeichnen
   for(int i=1;i<=6;i++)
   {
     turtle1.DrawTurtle(30);
     turtle1.TurnLeft(60);
   }
}
```



```
void Button1Click(object sender, System.EventArgs e)
{
  // Startposition setzen
  turtle1.MoveTurtleTo(200,200);
  // Methode aufrufen
  zeichneSechseck();
  // neue Startpositionen
  turtle1.MoveTurtleTo(100,100);
  zeichneSechseck();
  turtle1.MoveTurtleTo(150,150);
  zeichneSechseck();
}
```

Die Schleife wurde in eine eigene Methode ausgelagert, die in der Ereignisbehandlung mehrfach aufgerufen wird. Das Auslagern war in diesem Fall einfach, normalerweise muss man noch einige Anpassungen vornehmen, um die oben genannten Regeln zu berücksichtigen.

Aufgaben:

2. Schreibe ein C#-Programm, mit dem man nach Belieben **Rechtecke, Fünfecke und/oder Sechsecke** zeichnen kann.
3. Schreibe ein Programm mit einer Methode **`zeichneStrichmännchen`**, die ab einer vorgegebenen Position ein Strichmännchen zeichnen soll.

Methoden mit Parameter

Leider ist die Methode `zeichneSechseck()` aus Beispiel 3 noch nicht flexibel genug. Geschickter wäre es, wenn man die Startposition nicht im Hauptprogramm setzen muss, sondern direkt beim Aufruf der Methode angibt. Dies macht man folgendenmaßen:

Beispiel 4: Methode zeichneSechseck mit Parameter

```
1  void zeichneSechseck(int x, int y)
2  {
3    // Startposition setzen
4    turtle1.MoveTurtleTo(x,y);
5    // regelmäßiges Sechseck zeichnen
6     for(int i=1;i<=6;i++)
7     {
8       turtle1.DrawTurtle(30);
9       turtle1.TurnLeft(60);
10    }
11 }
12
13 void Button1Click(object sender, System.EventArgs e)
14 {
15   // Methode mehrmals aufrufen (inkl. Startposition setzen)
16   zeichneSechseck(200,200);
17   zeichneSechseck(100,100);
18   zeichneSechseck(150,150);
19 }
```

Die Zeile 1 wurde im Vergleich zu Beispiel 3 geändert. In der Klammer sind die Variablen `x` und `y` vom Typ `int` hinzugekommen. Diese bekommen beim Methodenaufruf die in der Ereignisbehandlung angegebenen Werte. x steht für die x-Koordinate der Startposition, y ist die y-Koordinate der Startposition.
Die Zeile 4 ist neu. Sie entspricht den entsprechenden Zeilen aus `Button1Click`.

Ausgehend von diesem Beispiel beschäftigen wir uns nun mit solchen Methoden allgemein:

Methoden mit Parameter verwendet man, wenn man Werte verarbeiten will, die aus dem restlichen Programm kommen,

Eine Methode mit Parameter hat folgenden allgemeinen Aufbau:

```
<Rückgabetyp> <Methodenname>(<Parameterliste>){
  // Code, der in der Methode ausgeführt wird
}
```

Neu ist die <Parameterliste> innerhalb der runden Klammern. Die `<Parameterliste>` ist eine Menge von Werten, die der Methode übergeben werden. Mit diesen Werten arbeitet die Methode dann. Die Deklaration eines Parameters entspricht der einer Variablen, abgesehen von Initialisierungswerten (wie z: B. bei `int x = 0;` Hier ist „= 0“ nicht erlaubt).

Mehrere Parameter werden durch Kommata getrennt und zu jedem Parameternamen muss eine Typbezeichnung angegeben werden.

Parameter sind in den Methoden Variablen und die entsprechenden Regeln gelten.

Methoden kann man gut mit Funktionen in der Mathematik vergleichen:
Betrachten wir z. B. die Funktion $f(x) = x^2 + 4$. Aus der Mathematik wissen wir, dass wir für x Werte einsetzen (z. B. x = 3) und den Funktionswert (hier: f(3) = 13) bestimmen. Mit Methoden macht man im Prinzip nichts Anderes. Sie sind nur viel flexibler!

Parameterübergabe bei Methoden

Die Parameter in der Definition der Methoden bezeichnet man auch als *formale Parameter*.

Beim Aufruf einer Methode passiert folgendes:

- Man ruft die Methode mit ihrem Namen auf und übergibt die Werte der aktuellen Variablen oder Konstanten. Diese werden von links nach rechts ausgewertet. Man bezeichnet diese Parameter als *aktuelle Parameter*. Sie müssen mit den Datentypen der formalen Parameter übereinstimmen oder sich in diese umwandeln lassen („zuweisungskompatibel“).
- Das Ergebnis der Auswertung wird einer lokalen Variablen mit dem Namen des formalen Arguments abgelegt.

In der Methode wird bei einfachen Datentypen also immer mit einer Kopie der übergebenen Daten gearbeitet. Deshalb nennt man die Parameter auch „*Werteparameter*“ und die Aufrufkonvention „*call by value*“.

C# kennt noch eine andere Art, mit Parametern umzugehen. Wir werden auf sie beim nächstem Mal zu sprechen kommen.

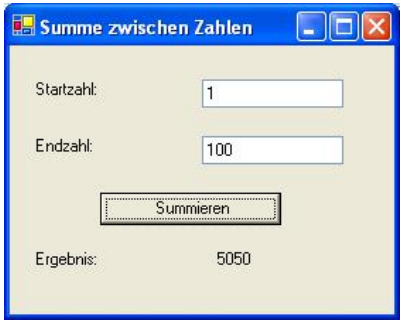
Beispiel 5: Formular mit Button button1

```
1  double Wurzel(double x){
2        return Math.Sqrt(x);
3  }
4
5  void Button1Click(object sender, System.EventArgs e)
6  {
7        double x = Wurzel(2);
8        MessageBox.Show(x.ToString());
9  }
```

- Die Methode Wurzel liefert einen double-Wert zurück und hat einen Parameter vom Typ double. (siehe Zeile 1)
- Die Methode wird in Zeile 7 aufgerufen, in dem man den Methodennamen und in runden Klammern den Wert des Parameters schreibt.

Natürlich ist dieses Beispiel nicht ganz so sinnvoll, da es in C# schon eine fertige Methode Math.Sqrt gibt!

Beispiel 6: Ganze Zahlen mit Hilfe einer Methode addieren




```
1  int summieren(int anfangszahl,int endzahl){
2        int summe=0;
3        for(int i=anfangszahl;i<=endzahl;i++){
4              summe=summe+i;
5        }
6
7        // Summe zurückliefern
8        return summe;
9  }
10
11 void Button1Click(object sender,
12                   System.EventArgs e)
13 {
14       // Anfangs- u. Endzahl aus Textboxen lesen
15       int anfang=int.Parse(textBox1.Text);
16       int ende=int.Parse(textBox2.Text);
17       // Methode summieren aufrufen und den Rückgabewert in der
18       // Variablen Ergebnis speichern
19       int ergebnis=summieren(anfang,ende);
20       label4.Text=ergebnis.ToString();
21 }
```

- In diesem Beispiel verwenden wir eine Methode mit zwei Parametern vom Typ `int` und einem Rückgabewert vom Typ `int` (siehe Zeile 1). Dies ist nur Zufall. Natürlich dürfen sich die Parameter in den Datentypen unterscheiden. Wichtig ist nur, dass der von **`return`** zurückgelieferte Wert mit dem im Methodenkopf definierten übereinstimmt.
- Wie oben schon erwähnt, werden die beiden Parameter mit Hilfe von Kommas aufgezählt. Vor jedem Parameter muss wie bei einer Variablendeklaration der jeweilige Datentyp stehen.
- In Zeile 19 wird die Methode aufgerufen: Wieder folgen dem Methodennamen in runden Klammern die Parameter. Wenn mehrere Parameter übergeben werden, sind diese durch Kommas getrennt. Entscheidend ist hier die Reihenfolge: Der erste übergebene Wert wird dem ersten Parameter zugeordnet. Der zweite Wert dem zweiten Parameter usw.

Aufgaben:

4. Schreibe ein C#-Programm mit einer Methode namens `hoch` für ganze Zahlen, die einen Ausdruck der Form Basis $^{\text{Exponent}}$ berechnet und das Ergebnis zurückliefert. Das Hauptprogramm soll zwei Zahlen von der Tastatur einlesen, die Methode `hoch` aufrufen und das Ergebnis ausgeben.
   Verwende keine Standardmethoden von C# für die Berechnung der Potenz!
5. Schreibe ein Programm, das mit Hilfe einer Methode f mit einem Parameter vom Typ double und dem Rückgabewert vom Typ double eine Wertetabelle ausgibt.
   f ist dabei folgendermaßen definiert: $f(x) = \frac{\sin(x)}{\sqrt{x+5}}$
   Hinweis: Der Sinuswert einer Zahl z (im Bogenmaß) wird in C# durch Math.Sin(z) berechnet und die Wurzel einer Zahl x durch Math.Sqrt(x).
6. Schreibe ein Programm mit einer sinnvoll benannten Methode, die die Minuten zurückliefert, die am heutigen Tag bis zum Aufruf der Methode vergangen sind. Dieses Ergebnis soll ausgegeben werden. *Tipp: Nutze Informationen aus Beispiel 2!*

