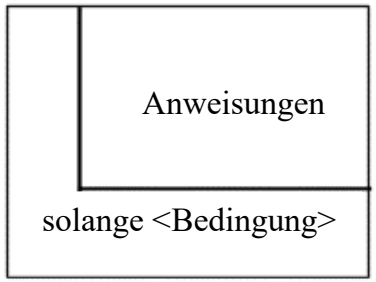
do-while-Schleifen in C#

Bei der do-while-Schleife wird erst am Schleifenende überprüft, ob eine Bedingung erfüllt ist:

Führe aus
Anweisung 1 ⎫
Anweisung 2 ⎬ Schleifenrumpf
.... ⎭
Anweisung n
solange *Bedingung*

Dies steht im Gegensatz zur while-Schleife, denn dort bei ihr wird schon zu Beginn der Schleife eine Bedingung geprüft. In beiden Fällen gilt, dass solange die Bedingung erfüllt, also wahr (true), ist, die Schleife durchlaufen wird. Wenn die Bedingung irgendwann nicht mehr erfüllt, also falsch (false) ist, wird die Schleife verlassen.
Auch bei der do-while-Schleife muss eine entsprechende Anweisung im Schleifenrumpfsicherstellen, dass die Abbruchbedingung nach endlich vielen Schleifendurchläufen den Wert false annimmt. Eine do-while-Schleife wird mindestens einmal durchlaufen, da die Schleifenbedingung erst nach jedem Schleifendurchlauf geprüft wird. Man nennt die do-while-Schleife daher eine **annehmende Schleife**.

Struktogramm:



Formulierung in C#:

```
do
   Anweisung;
while(Bedingung)
```

bzw.

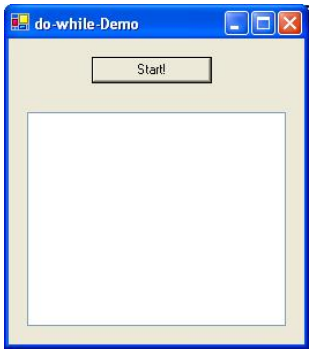
```
do{
   Anweisung1;
   Anweisung2;
      ...
}
while(Bedingung)
```

So wie in C# üblich gilt, dass wenn die Anweisung aus mehreren Einzelanweisungen zusammengesetzt ist, müssen. diese durch { und } zusammengefasst werden.

Beispiel:

```
void Button1Click(object sender, System.EventArgs e)
{
   // Initialisierung
   int x=20;

   // Schleife
   do{
     // Ausgabe des aktuellen Wertes von x
     listBox1.Items.Add(x);
     x=x/2;
   }
   while(x>1);
}
```



Das Programm DoWhileDemo gibt die Zahlen 20, 10, 5, 2 in jeweils einer neuen Zeile aus und entspricht damit dem Beispiel WhileDemo auf dem Blatt zur while-Schleife. Jedoch wurde diesmal eine Listbox statt der Messageboxen verwendet.

Beispiel: Umgang mit Folgen

Folgenglieder a_{n+1} sollen solange berechnet werden, bis die Differenz zweier Folgenglieder kleiner ist als eine vorgegebene Genauigkeit.

Definition der Zahlenfolge: $a_0 = 1020$ $a_{n+1} = a_n / 2$

```
void Button1Click(object sender, System.EventArgs e)
{
      double an;       // entspricht a_n
      double an1;           // entspricht a_n+1
      double Genauigkeit=0.5;   // vorgegebene Genauigkeit

      listBox1.Items.Clear();  // Listbox löschen
      an1=1020;        // Startwert setzen

      do{
            // an übernimmt den Wert von letzten neuen Folgenglied
            an=an1;
            // neues Folgenglied berechnen
            an1=an/2.0;
            // neues Folgenglied ausgeben
            listBox1.Items.Add(an1);
      }
      // Prüfen, ob noch weitergerechnet werden soll
      while(Math.Abs(an1-an)>Genauigkeit);
}
```

Aufgaben:

1. Schreibe ein C#-Programm, dass die Zahlenfolge $a_0 = 4$, $a_{n+1} = a_n/1.5$ bis zu einer eingegeben Genauigkeit berechnet und jedes Folgenglied ausgibt.
2. Eine Tasse Kaffee, die gerade auf den Tisch gestellt worden ist, hat eine Temperatur von 85 °C. Die Zimmertemperatur beträgt 21°C. In jeder Minute verringert sich die Temperatur des Kaffees um ein Zehntel der Differenz von Kaffeetemperatur und Zimmertemperatur.
   Schreibe ein C#-Programm, dass die Kaffeetemperatur nach 0,1,2,3,4,... Minuten ausgibt.
3. *Alternative Formulierung des Heron-Verfahrens*:
   Zur näherungsweisen Berechnung aus einer Quadratwurzel aus einer Zahl a gibt es folgendes Verfahren, das nach Heron von Alexandria (60 n. Chr.) benannt ist:
   - Gib dir eine Zahl a vor (Einlesen mit Hilfe von Textbox). Halbiere die Zahl a und nenne das Ergebnis x.
   - Dividiere die Zahl a durch die Zahl x und nenne das Ergebnis z
   - Wiederhole, solange bis (z-x)(z-x)<1/100000:
     – Addiere zu der Zahl x die Zahl z und nenne die Hälfte der Summe x.
     – Dividiere die Zahl a durch die Zahl x und nenne das Ergebnis z.
     – Gebe die Zahl z aus.

   Schreibe ein C#-Programm, das die Quadratwurzel nach diesem Verfahren berechnet und zur Kontrolle die Quadratwurzel mit der entsprechenden C#-Methode `Math.Sqrt` ausgibt.
4. Löse die Aufgaben vom Übungsblatt „while-Schleifen in C#“ unter Verwendung der do-while-Schleife, insbesondere die Aufgaben 1, 2 und 6.