ZUSAMMENFASSUNG Variablen und Datentypen

WAS SIND VARIABLEN?

Variablen sind Speicherplätze für Daten innerhalb unseres Programmcodes. Sie sind lediglich zur Laufzeit unseres Programms da, um bestimmte Daten im Arbeitsspeicher zu halten. So können wir sie jederzeit in unserem Projekt verwenden.

Beispiel 1:
Bei der Entwicklung eines Videospiels, in dem Gegner existieren gegen die der Spieler kämpfen muss, benötigen wir beispielsweise eine Variable, in die wir die Anzahl der Leben des Spielers speichern. Wird der Spieler von einem Gegner getroffen, kann im Code von dieser Variable eins (= 1 Leben) abgezogen werden.

Beispiel 2:
Wir programmieren eine Anwendung, bei der nach dem Login der Benutzername, der gerade eingegeben wurde, nochmals mit einer Begrüßung angezeigt werden soll.
Um den individuellen Benutzernamen ausgeben lassen zu können, benötigen wir auch hierfür eine Variable, in die wir diesen hineinspeichern.

Der Inhalt von Variablen kann sowohl gelesen als auch **überschrieben** werden.
Des Weiteren sind Variablen in C# **typisiert**. Das bedeutet, dass nicht jede Art von Wert in jede Variable gespeichert werden kann. Man muss beispielsweise zwischen Text und Fließkommazahlen unterscheiden. Mehr dazu wirst du im späteren Verlauf beim Thema Datentypen erfahren.

Variablen haben einen sogenannten **Bezeichner**. Das ist der Name der Variable, über den wir sie im Code ansprechen können.

PRAXISBEISPIEL

Abbildung 1, Beispiel

Um den Namen „Janek“ in die Variable username zu schreiben, müssen wir die folgende Zeile Code verfassen:

string username = “Janek“;

Das Wort **string** bezeichnet den Datentyp für Textwerte.

„username“ ist der **Name (Bezeichner)** der Variable und mit dem Zuweisungsoperator (=) weisen wir der Variable den Wert „Janek“ zu. Das bedeutet, dass nun der Name **Janek** in der Variable steht.

Wenn wir diese Variable nun auf der Konsole ausgeben möchten, schreiben wir die folgende Zeile Code:

Console.WriteLine(username);

Da der Computer die Anweisung erhalten hat, **username** auszugeben, überprüft er, was in der Variable gespeichert wurde. Somit wird der Name Janek ausgegeben, weil sich dieser schließlich darin befindet.

VARIABLEN DEKLARIEREN UND INITIALISIEREN

Unter der Deklaration einer Variable versteht man ihre Bekanntmachung im Programm. Um eine Variable zu deklarieren, müssen wir sowohl ihren Datentyp als auch ihren Bezeichner (Namen) angeben. Eine Deklaration sieht wie folgt aus:

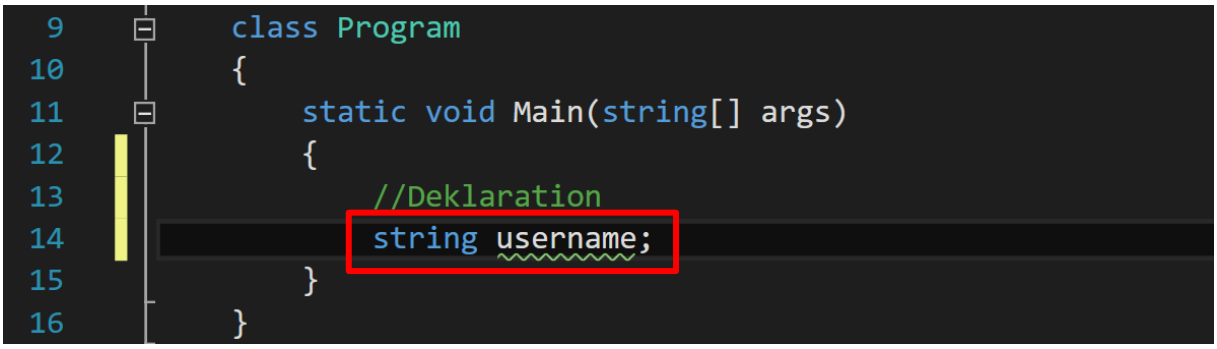

Abbildung 2, Deklaration einer Variable

Eine Variable zu **initialisieren** bedeutet, ihr einen ersten Wert zuzuweisen. (s. Abbildung 3)

Abbildung 3, Initialisierung einer Variable

Deklaration und Initialisierung können auch in einem Schritt erfolgen:

```
9   class Program
10  {
11      static void Main(string[] args)
12      {
13          //Deklaration und Initialisierung
14          string username = "Janek";
15      }
16  }
```

Abbildung 4, Deklaration und Initialisierung einer Variable

Wenn wir auf eine Variable zugreifen möchten, die bereits deklariert wurde, müssen und sollten wir den Datentyp (in unserem Beispiel String) **nicht** mehr vorne angeben.

VARIABLEN ÜBERSCHREIBEN

Wenn wir eine Variable überschreiben möchten, weisen wir dieser im nächsten Schritt einfach einen anderen Wert zu:

```
9   class Program
10  {
11      static void Main(string[] args)
12      {
13          string username;
14          username = "Janek";
15
16          Console.WriteLine(username);
17
18          username = "Sandra";
19
20          Console.WriteLine(username);
21
22          Console.ReadKey();
23      }
24  }
```

Abbildung 5, Überschreibung einer Variable

Im nachfolgenden Beispiel (Abbildung 6) legen wir eine neue Variable vom Typ Integer an. Dieser Datentyp kann **Ganzzahlen** speichern. Anschließend verbinden wir einen Text, den wir in die WriteLine-Methode schreiben mit den Variablen, die wir bereits erstellt haben, um beides gemeinsam auszugeben. (Zeile 17 und 18)

Kleiner Tipp:
Um Console.WriteLine schneller einzugeben, tippt man zunächst die Buchstaben **cw** und drückt anschließend zweimal auf die Tab-Taste.

```
 9      class Program
10      {
11          static void Main(string[] args)
12          {
13              string username;
14              username = "Janek";
15              int age = 18;
16
17              Console.WriteLine("Name: " + username);
18              Console.WriteLine("Alter: " + age);
19
20              Console.ReadKey();
21
```

Abbildung 6, Integer initialisieren, Text mit Variablen verbinden

WAS SIND DATENTYPEN?

Datentypen bestimmen, welche Art von Daten in eine Variable gespeichert werden kann.

Arten von Datentypen

- Numerische Datentypen
- Text Datentypen
- und viele mehr…

In Abbildung 7 erhältst du einen Überblick über die Ganzzahlen Datentypen (numerische Datentypen) sowie deren Wertebereich und Speicherverbrauch:

Ganzzahlen Datentypen

Datentyp	Niedrigster Wert	Höchster Wert	Speicherverbrauch
byte	0	255	8 Bit
short	-32.768	32.767	16 Bit
int	-2,147,483,648	2,147,483,647	32 Bit
long	-9,223,372,036,854,775,808	9,223,372,036,854,775,807	64 Bit
sbyte	-128	127	8 Bit
ushort	0	65.535	16 Bit
uint	0	4.294.967.295	32 Bit
ulong	0	18.446.744.073.709.551.615	64 Bit

Abbildung 7, Ganzzahlen Datentypen

Bei **Ganzzahlen** handelt es sich um Zahlen, die **keine** Nachkommastelle besitzen.

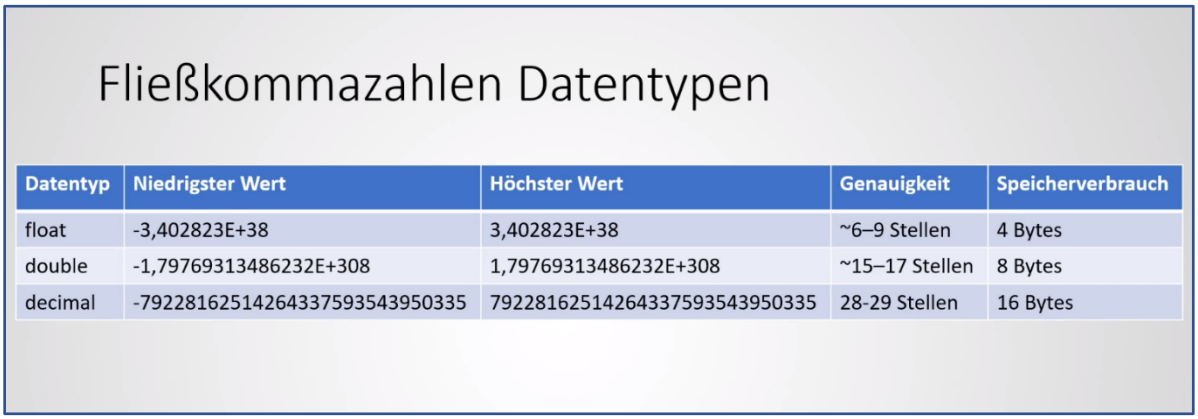

Abbildung 8, Fließkommazahlen Datentypen

Bei Fließkommazahlen handelt es sich um Zahlen, die Nachkommastellen enthalten. Für das Speichern von Fließkommazahlen stehen uns die Datentypen aus Abbildung 8 zur Verfügung. Fließkommazahlen unterscheiden sich nicht nur in ihrem Wertebereich, sondern vor allem in ihrer **Genauigkeit** voneinander. An der Genauigkeit lässt sich erkennen, nach wie vielen Stellen die Zahl gerundet wird. Somit ist die richtige Wahl des Datentyps in manchen Fällen für ein korrektes Ergebnis entscheidend, da es andernfalls zu Rundungsfehlern kommen kann. Zur Rechnung mit Währungen und Geldwerten zum Beispiel, sollte man den Typ Decimal heranziehen. Bei den angegebenen Stellen handelt es sich sowohl um Vorkomma- als auch Nachkommastellen.

Auf der nächsten Abbildung sind noch die weiteren Datentypen veranschaulicht:

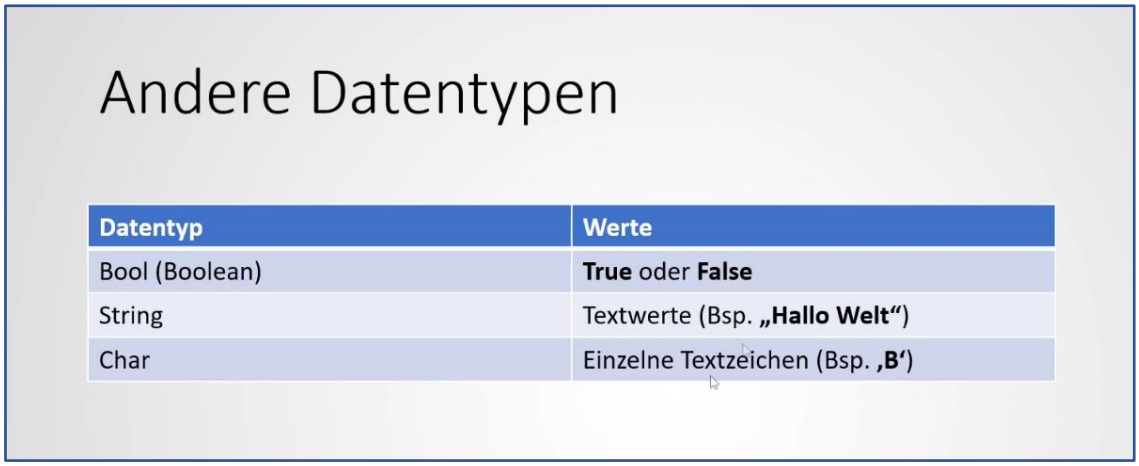

Abbildung 9, Boolean, String und Char

Boolean: Ein Wahrheitswert, mit dem ein Zustand beschrieben werden kann, der entweder wahr (true) oder falsch (false) sein soll.

String: In Strings werden Textwerte gespeichert. Diese müssen immer zwischen zwei doppelte Anführungszeichen geschrieben werden. Da ein String im Grunde eine Aneinanderreihung von Chars darstellt, wird dieser auch Zeichenkette genannt.

Char: Das Wort char steht für character. In diesen Datentyp können einzelne Zeichen gespeichert werden. Chars werden nicht wie Strings zwischen doppelte Anführungszeichen geschrieben, sondern zwischen einfache.

GANZZAHLEN DATENTYPEN IN DER PRAXIS

```
 9      class Program
10      {
11          static void Main(string[] args)
12          {
13              byte zahl1 = 200;
14              short zahl2 = -20000;
15              int zahl3 = -2000000;
16              long zahl4 = 200000000;
17
18              Console.WriteLine(zahl1);
19              Console.ReadKey();
20          }
21      }
```

Abbildung 10, Ganzzahlen Datentypen

In Abbildung 10 sehen wir die Ganzzahlen Datentypen einmal in der Praxis. Wichtig ist, dass man in diese keine Zahlen schreibt, die die jeweiligen Wertebereiche über- oder unterschreiten, da sonst Fehlermeldungen auftreten.

Kleiner Tipp:

Um den minimalen sowie den maximalen Wert eines Datentyps ausgeben zu lassen, schreibt man die folgenden Zeilen:

```
Console.WriteLine(int.MinValue);      //gibt den minimalen Wert vom Typ Integer aus
Console.WriteLine(int.MaxValue);      // gibt den maximalen Wert vom Typ Integer aus
```

FLIEßKOMMAZAHLEN ALS DATENTYPEN IN DER PRAXIS

Wenn wir in C# Kommazahlen schreiben, ohne ein sogenanntes **Suffix** am Ende der Zahl anzuhängen, werden diese immer als Typ **Double** erkannt. Bei den Datentypen **Float** und **Decimal** muss dieses Suffix also angehängt werden. Das kann man auf Abbildung 11 erkennen.

Übersicht der Suffixe

Suffix für den Typ Float:	f oder F
Suffix für den Typ Decimal:	M
Suffix für den Typ Double:	D (muss nicht gesetzt werden)

```
 9    class Program
10    {
11        static void Main(string[] args)
12        {
13            float zahl1 = 2.25F;
14            double zahl2 = 33.3245;
15            decimal zahl3 = 32.45M;
16
17            Console.WriteLine(zahl2);
18            Console.ReadKey();
19        }
```

Abbildung 11, Datentypen und ihre Suffixe

DIE ANDEREN DATENTYPEN IN DER PRAXIS

Auf den nachfolgenden Bildern sehen wir die Datentypen String, Char und Boolean einmal in der Praxis:

```
 9    class Program
10    {
11        static void Main(string[] args)
12        {
13            string text = "Hallo Welt!";
14            char character = 'B';
15
16            Console.WriteLine(character);
17            Console.ReadKey();
18        }
19    }
```

Abbildung 12, String und Char

```
11        static void Main(string[] args)
12        {
13            string text = "Hallo Welt!";
14            char character = 'B';
15
16            bool wahrheitswert = true;
17
18            Console.WriteLine(wahrheitswert);
19            Console.ReadKey();
20        }
```

Abbildung 13, Boolean

DIE EINGABE VOM BENUTZER ENTGEGENNEHMEN

Um die Eingabe vom Benutzer entgegenzunehmen, verwenden wir die ReadLine-Methode. Diese liest beim Aufruf die Zeile, die der Benutzer über seine Tastatur eingibt. Die Nutzereingabe endet, sobald ein Zeilenumbruch gesetzt wird.
In der Praxis kann der Code folgendermaßen aussehen:

```
11          static void Main(string[] args)
12          {
13              Console.Write("Gebe deinen Namen ein: ");
14              string name = Console.ReadLine();
15
16              Console.WriteLine(name);
17              Console.ReadKey();
18          }
```

Abbildung 14, Die ReadLine-Methode

```
Gebe deinen Namen ein: Janek
Janek
```

Abbildung 15, Ausgabefenster

Die Ausgabe des Codes sehen wir auf Abbildung 15. Nachdem der Text „Gebe deinen Namen ein:“ ausgegeben wurde, wartet das Programm nun so lange, bis ein Text vom Benutzer eingegeben wird. In unserem Beispiel ist das der Name Janek. Drückt der Benutzer nach Eingabe auf Enter, um einen Zeilenumbruch einzuleiten, wird der Name in der nächsten Zeile ausgegeben.

Info:

Console.Write(“Text“);	lässt den Cursor am Ende in der gleichen Zeile vom Text stehen
Console.WriteLine(“Text“);	setzt den Cursor nach dem Text in die nächste Zeile

Wie funktioniert das nun, wenn wir eine Zahl entgegennehmen möchten?
Da ein String, der durch die ReadLine-Methode entgegengenommen wird, nicht in einen Integer gespeichert werden kann, müssen wir die Methode in einen Integer umwandeln. Wie das funktioniert, kann man auf Abbildung 16 erkennen:

```
11          static void Main(string[] args)
12          {
13              Console.Write("Gebe deinen Namen ein: ");
14              string name = Console.ReadLine();
15
16              Console.Write("Gebe dein Alter ein: ");
17              int age = Convert.ToInt32(Console.ReadLine());
18
19              Console.ReadKey();
20          }
```

Abbildung 16, Umwandlung zu einem Integer

```
11      static void Main(string[] args)
12      {
13          Console.Write("Gebe deinen Namen ein: ");
14          string name = Console.ReadLine();
15
16          Console.Write("Gebe dein Alter ein: ");
17          int age = Convert.ToInt32(Console.ReadLine());
18
19          Console.WriteLine("Hallo {0} du bist {1} Jahre alt!", name, age);
20
21          Console.ReadKey();
22      }
```

Abbildung 17, Einsatz von Platzhaltern

Erläuterung:

{0}	Der **erste** Platzhalter im WriteLine-Befehl
{1}	Der **zweite** Platzhalter im WriteLine-Befehl
name	Diese Variable wird in den ersten Platzhalter gespeichert
age	Diese Variable wird in den zweiten Platzhalter gespeichert

Eine gewöhnliche Stringverkettung, ohne Platzhalter, sieht folgendermaßen aus:

```
11      static void Main(string[] args)
12      {
13          Console.Write("Gebe deinen Namen ein: ");
14          string name = Console.ReadLine();
15
16          Console.Write("Gebe dein Alter ein: ");
17          int age = Convert.ToInt32(Console.ReadLine());
18
19          Console.WriteLine("Hallo " + name + " du bist " + age + " Jahre alt!");
20
21          Console.ReadKey();
22      }
```

Abbildung 18, Verkettung von Strings