

DEPARTMENT WIRTSCHAFTSINFORMATIK

FACHBEREICH WIRTSCHAFTSWISSENSCHAFT

# Programmieren für Wirtschaftswissenschaftler

## SS 2015

Lucian Ionescu

Blockveranstaltung 16.03–27.3.2015

2. Variablen

# Agenda

---

- **Variablen**
  - Notwendigkeit
  - Variablennamen
  - Deklaration und Initialisierung
- Datentypen
- Operatoren
- Variablen in Objekten

# Aktueller Stand

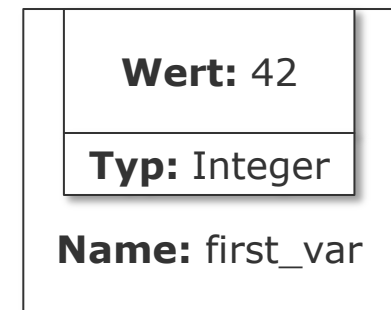
---

- Wir können bisher
  - eine Programmierumgebung in Visual Studio einrichten
  - Eingaben des Benutzers entgegennehmen
  - Ausgaben über die Konsole ausgeben
- Wir können bisher nicht
  - Eingaben zwischenspeichern
  - Berechnungen ausführen
- Warum nicht?

# Variablen

---

- Ein Programm soll Werte (Namen, Zahlen etc.) annehmen und mit diesen Berechnungen durchführen
- Eingaben, z.B. per `Console.ReadLine()`, müssen dafür **gespeichert** werden
- Variablen dienen als **Container** für zu speichernde Werte
- Variablen behalten ihren Namen und Typ
- Variablen werden im Arbeitsspeicher gespeichert und verfallen nach Beenden des Programms
- Ihnen kann aber jederzeit ein neuer Wert des **gleichen Typs** zugewiesen werden
- Wichtig: **aussagekräftige** Namen wählen!



# Variablennamen

---

- Variablennamen/Bezeichner
  - müssen mit einem Unterstrich oder einem Buchstaben beginnen
  - dürfen Buchstaben (nach Unicode), Ziffern und den Unterstrich enthalten
  - dürfen keine Leerzeichen enthalten
  - dürfen keine reservierten C# Schlüsselwörter sein (engl. *keywords*)
    - Auflistung: <http://msdn.microsoft.com/en-us/library/x53a06bb.aspx>
- Variablen sollten so benannt werden, dass ihr Zweck deutlich wird
  - erheblich bessere **Lesbarkeit** des Codes
  - man entwickelt schneller
- Erinnerung: C# unterscheidet zwischen **Groß-** und **Kleinschreibung**
  - engl. *case-sensitive*
  - Beispiel für drei unterschiedliche Variablen!
    - `int`      `meinevariable;`
    - `string`   `meineVariable;`
    - `bool`      `MeineVariable;`

# Variablennamen – Konvention

---

- lowerCamelCase (für Variablen)
  - deutsch: Binnenmajuskel
  - Beispiele:
    - personAge
    - personName
  - mehrere Wörter **zusammenschreiben**
  - Anfangsbuchstaben werden groß geschrieben
  - allererste Buchstabe wird **klein** geschrieben
- Upper Camel Case (für Funktionen, Klassen und Namensräume → später)
  - Beispiele:
    - GetAge()
    - GetName()
  - wie lowerCamelCase, aber allererster Buchstabe wird ebenfalls **groß** geschrieben
  - Auch Pascal Case genannt



# Deklaration und Initialisierung

---

- Für jede Variable wird bei **Deklaration** ein Bereich im Arbeitsspeicher reserviert:

`<datatype> <identifier>;`

Bsp.: `int a;`  
`string b;`

- Die erste Belegung der Variable wird als **Initialisierung** bezeichnet
- Eine Variable muss deklariert worden sein, bevor sie initialisiert werden kann

`<identifier> = <value>;`

Bsp.: `a = 10;`  
`b = "krawehl";`

- Deklaration und Initialisierung kann auch **gleichzeitig** geschehen

`<datatype> <identifier> = <value>;`

Bsp.: `int a = 10;`  
`string b = "krawehl";`

- Deklaration und Initialisierung mehrerer **gleichartiger** Variablen möglich

`<datatype> <identifier> = <value>;`

Bsp.: `int a, x = 10, y;`  
`string b = "krawehl", c = "hitzefrei";`

# Initialisierung

---

- Was wird hier ausgegeben?

```
int a, b = 10;  
int c = a + b;  
Console.WriteLine("{0} + {1} = {2}", a, b, c);
```

```
int a = 5, b = 10;  
string c = a + b;  
Console.WriteLine("{0}, {1}, {2}", a, b, c);
```

```
int a = 5, b = 10 + a;  
int c = b + 10;  
Console.WriteLine("{0}, {1}, {2}", a, b, c);
```



# Agenda

---

- Variablen
- **Datentypen**
  - Zeichenketten
  - Wertebereiche
- Operatoren
- Variablen in Objekten

# Datentypen

---

- Der Speicher ist grundsätzlich begrenzt → **effiziente** Nutzung des Speichers
  - ein Wahr-/Falsch-Wert braucht weniger Platz als eine Zahl von -128 bis 127
  - natürliche Zahlen benötigen keine 1.000 Nachkommastellen
  - deshalb: bei Deklaration einen möglichst kleinen, aber hinreichend großen Speicherbereich reservieren!
- Die Variable kann nur Werte speichern, die ihrem **Datentyp** entsprechen!

Bsp.:     `int a = 10;`  
          `string b = "krawehl";`  
          `a = b; // leads to an error (a is an integer and b is a string)`

# Wichtige Datentypen

---

- Zeichenketten: String
  - `string` message = "Press Enter to close the application."
  - Zahlen in **Anführungsstrichen** werden ebenfalls als String gewertet
- Einzelne Zeichen: Char
  - `char` character = '1'; // andere Anführungsstriche beachten!
- Boolescher Datentyp (wahr/falsch)
  - `bool` truth = false;
- Ganzzahlige Datentypen
  - Unterscheidungsmerkmale sind **Wertebereich** und **Speicherverbrauch**
  - Bsp.: `byte` , `int` , `long` usw.
- Fließkommazahlen
  - Unterscheidungsmerkmale sind Wertebereich, Speicherverbrauch, **Genauigkeit nach dem Komma**
  - Bsp.: `float` , `double` , `decimal` usw.

# Ganzzahlige Typen

---

Datentyp	Speicherbedarf in Byte	Wertebereich	
sbyte	1	$-2^7$	$2^7 - 1$
byte	1	0	$2^8 - 1$
short	2	$-2^{15}$	$2^{15} - 1$
ushort	2	0	$2^{16} - 1$
int	4	$-2^{31}$	$2^{31} - 1$
uint	4	0	$2^{32} - 1$
long	8	$-2^{63}$	$2^{63} - 1$
ulong	8	0	$2^{64} - 1$
char	2	0	$2^{16} - 1$

# Fließkommazahlen

---

Datentyp	Speicher	Genauigkeit	Wertebereich
float	4	7 Stellen	$\pm 1.5e-45$ bis $\pm 3.4e38$
double	8	15–16 Stellen	$\pm 5.0e-324$ bis $\pm 1.7e308$
decimal	16	28–29 Stellen	$(-7.9 \times 10^{28}$ bis $7.9 \times 10^{28}) / (100$ bis $28)$

Weiterführend: <http://msdn.microsoft.com/en-us/library/9ahet949.aspx>

# Übung – Wahl des Datentyps

---

- Wähle den **kleinstmöglichen** Datentyp, der für alle Werte der jeweiligen Zeile zulässig ist!

Werte	Typ?
2; 3.4; -5.6	
4; 2; ?; 1	
C; r; w; tz; \$%; g	
0; 3; -4; -2000	
Hallo; Name; Ahorn	
0.34638267344; -1234.5768	
$10^{12}$ ; $3.5673 \times 10^{13}$	
true; false	
(1; 2; 3); (5; 9; -2); (2; 3; 1)	
0; 1	

# Erweitert: Escape-Zeichen

---

- Wie können Sonderzeichen dargestellt werden?
  - Leerzeichen (Zeilenumbrüche, Tabulatoren)
  - Zeichen, die in C# bereits eine andere Bedeutung haben (z.B. ; oder ")
  - Internationale Zeichen (z.B. chinesische Schriftzeichen)
- Escape-Zeichen
  - \\ Backslash
  - \" Anführungsstriche
  - \n Zeilenumbruch
  - \t Tabulator
  - \v Vertikaler Tabulator
  - \u Unicode Zeichen, Bsp:  $\pi \triangleq \backslash u03a0$ ,  $\sigma \triangleq \backslash u03a3$
  - \x ASCII Zeichen



# Escape Zeichen

---

- Was wird hier ausgegeben?

```
Console.Write(" * \n ***\n*****\n");
```

```
Console.Write("*\n**\n***");
```

```
Console.WriteLine("2\\5=0.4");
```

```
Console.WriteLine("\"\\u03a0\" is exactly 3!");
```

```
Console.WriteLine("\tEnte\tEnte\tEnte\tEnte");
```

```
Console.WriteLine("{0}{1}", "Hello!", "\n");
```



# Erweitert: Variablen technisch betrachtet

---

- C# ist eine **streng-typisierte** Programmiersprache (engl. *strongly typed*)
  - der Typ einer Variablen ist vorher festzulegen (Deklaration) und bleibt fest!
  - Gegenteil: „schwach-typisiert“ (engl. *weak typed*)
    - `b = "hello"; b = 12;`
    - Vertreter: PHP, Visual Basic 6 usw.
- Auch als **statische Typisierung** vs. **dynamische Typisierung** bezeichnet

```
class Program
{
    static void Main(string[] args)
    {
        // Welche Zuweisungen sind falsch? Warum?
        char c = 'ü';
        c = 3;
        c = '3';
        c = "3";
        string c = "33";
    }
}
```

# Typumwandlung

---

- Oft muss man einen Wert in einen anderen Typ **konvertieren**, z.B. bei Konsoleneingaben
  - `Convert`-Klasse bietet viele Methoden zur Umwandlung
  - `int` zahl = `Convert.ToInt32(Console.ReadLine());`
- `ToString()` ist für jeden Datentyp definiert und wird häufig implizit aufgerufen
  - `Console.WriteLine("1+1 = " + 1);`
- Umwandeln zwischen verschiedenen Zahlentypen
  - **Implizite** Umwandlung:
    - Verlustfrei vom kleineren Typ in größeren Typ
  - **Explizite** Umwandlung:
    - **Verluste** können auftreten, daher ist eine explizite Angabe nötig
      - `double` d = 2.1;
      - `int` i = (`int`) d;
    - **Nachkommastellen** fallen weg (abrunden)
    - Was geschieht bei `double` d = 2/5;?

# Agenda

---

- Variablen
- Datentypen
- **Operatoren**
  - Prioritäten
  - Arithmetische Operatoren
  - Zusammengesetzte Operatoren
- Variablen in Objekten

# Assoziation und Priorität

---

- Definition: Ausdruck
  - ein Ausdruck ist ein **Konstrukt**, das in Bezug auf einen Kontext ausgewertet werden kann und einen **Wert zurückliefert**
  - auch eine **Verknüpfung** von Variablen, Konstanten und Funktionen durch Operatoren
  - Beispiel:  $a$ ,  $a + b$ ,  $c + 2$ ,  $1 + 4 * 3$
- Auswertung von Ausdrücken
  - **Priorität** von Operatoren (engl. *precedence*)
  - Die Priorität entspricht unseren „normalen“ Rechenregeln
    - Beispiel: Punktrechnung vor Strichrechnung
  - **Klammerungen** haben stets Vorrang

# Arithmetische Operatoren

Operator	Funktion	Ausdruck	Ergebnis
+	Addition	$1 + 2$	3
-	Subtraktion	$1 - 2$	-1
*	Multiplikation	$1 * 2$	2
/	Division	$1 / 2$	0
%	Restdivision („Modulo“)	$1 \% 2$	1

- Erläuterung zur Division: 0,5 ist keine ganze Zahl!
- Lösungen:
  - $1.0 / 2$
  - $1 / 2.0$
  - `(double) 1 / 2`
  - $1 / (\text{double}) 2$

# Operatoren sind kontextabhängig

---

- Anwendung des +-Operators auf einen String nennt man **Konkatenation**
  - `"1 + 1 = " + 1;`
  - `"1 + 1 = " + 1 + 1;`
  - `"Ich heiße" + "Reinsch.";`
- Anwendung des +-Operators auf Zahlen ergibt eine **Addition**
  - `2 + 1`
- ...und hier?
  - `"1 + 1 = " + (1 + 1);`

# Zusammengesetzte Zuweisungs-Operatoren

Zuweisungs-Operator	Anwendung	Eigentlicher Ausdruck
<code>+=</code>	<code>a += 1</code>	<code>a = a + 1</code>
<code>-=</code>	<code>a -= 1</code>	<code>a = a - 1</code>
<code>*=</code>	<code>a *= 1</code>	<code>a = a * 1</code>
<code>/=</code>	<code>a /= 1</code>	<code>a = a / 1</code>
<code>%=</code>	<code>a %= 1</code>	<code>a = a % 1</code>

## Inkrement- und Dekrementoperatoren

Operator	Anwendung	Eigentlicher Ausdruck
<code>++</code>	<code>a++</code> oder <code>++a</code>	<code>a = a + 1</code>
<code>--</code>	<code>a--</code> oder <code>--a</code>	<code>a = a - 1</code>

Welche Ausgabe erzeugt

```
int a = 10;  
Console.WriteLine(a++);  
Console.WriteLine(++a);
```

# Agenda

---

- Variablen
- Datentypen
- Operatoren
- **Variablen in Objekten**



# Variablen in Objekten

- Bisher haben wir Variablen in der Main-Methode deklariert
- Auch Objekte können Variablen besitzen, um Eigenschaften/Werte zu speichern
- Zur Erinnerung: Farbe und Größe eines Hundes (vgl. erste Vorlesung)
- Neues Beispiel: Eine Klasse Konto benötigt?
  - ein Feld für den Kontostand
  - den Namen oder die Kundennummer des Besitzers
  - ...



```
class Account
{
    public float balance; // a variable for the current account balance

    public Account() // constructor
    {
        //...
    }
}
```

# Variablen in Objekten (2)

---

```
class Program
{
    static void Main(string[] args)
    {
        Account first = new Account();
        first.balance += 10;
    }
}

class Account
{
    public float balance; // a variable for the current account balance

    public Account() // constructor initializes the balance variable
    {
        balance = 0;
        Console.WriteLine("A new account has been created.");
    }
}
```