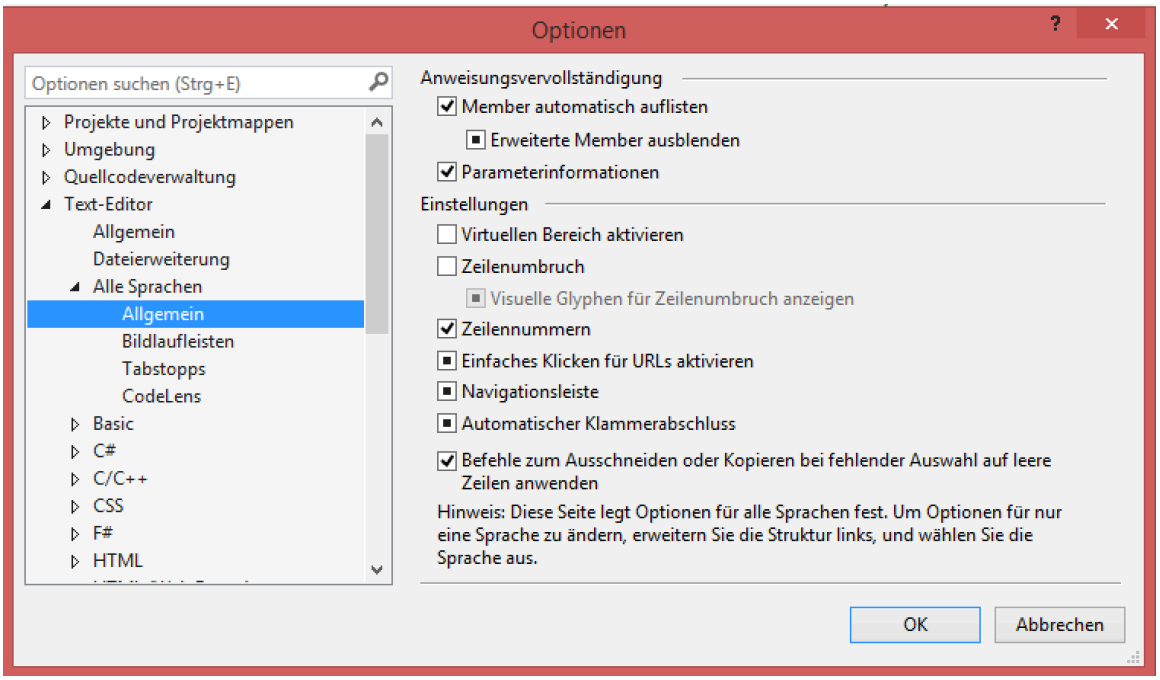
Modul 403: Programmabläufe prozedural implementieren

M403-Codebeispiel-2 C#: Fläche eines Rechteckes berechnen

Erstellen wir ein nächstes Programm, welches die Fläche eines Rechteckes berechnet:

Hierzu wollen wir erstmals die Zeilennummer im Code-Editor anzeigen lassen:
Gehen Sie ins Menu *Extras → Optionen → Texteditor → Alle Sprachen → Zeilennummer aktivieren* ☺



- Gehen Sie in Menu DATEI → Wählen Sie Neu → Projekt
- Wählen Sie Visual C# → Konsolenanwendung
- Bezeichnen Sie das Projekt mit `RechteckFlaeche`

Erfassen Sie folgendes Programm und führen Sie es aus:

Program.cs

RechteckFlaeche.Program | Main(string[] args)

```
1  // Modul 403
2  // gino.colombo@gibb.ch
3  // file: RechteckFlaeche.sln
4
5  using System;
6
7  namespace RechteckFlaeche
8  {
       0 Verweise
9      class Program
10     {
11         static double laenge;
12         static double breite;
13         static double resultat;
14
               1-Verweis
15         static double FlaecheBerechnen ()
16         {
17             return laenge * breite;
18         }
19
               1-Verweis
20         static void Anzeigen ()
21         {
22             Console.WriteLine ("Länge: {0}", laenge);
23             Console.WriteLine ("Breite {0}", breite);
24             Console.WriteLine ("Fläche {0}", resultat);
25         }
26
               0 Verweise
27         static void Main(string[] args)
28         {
29             laenge = 16.5;
30             breite = 30;
31
32             resultat = FlaecheBerechnen();
33             Anzeigen();
34
35             Console.ReadKey();
36         }
37 }
38 }
```

133 %

Betrachten wir wichtige Programmbestandteile anhand des Beispiels
`RechteckFlaeche`

1. Identifiers (Bezeichner)
Es gibt Wörter im Programmiercode, welchen wir Softwareentwickler/innen selber bestimmen können. Da wären in unserem Beispiel die Bezeichnung – auch Identifier genannt – des Namensraumes: `RechteckFlaeche`

Im Code wird der **Namensraum** wie folgt definiert `namespace RechteckFlaeche`.

Das Keyword `namespace` ist reserviert und gehört zur Syntax von C# und darf nicht als Identifier genutzt werden. Es deklariert, dass der folgende Identifier ein Namenraum ist.

Gleiches gilt für die Klassenbezeichnungen und für die Namen der Variablen und Methoden. Alle diese Bezeichnungen nennt man auch Identifier.

- `class Programm`
- `static double laenge;`
- `static double breite;`
- `static double resultat;`
- `static double FlaecheBerechnen()`
- `static void Anzeigen()`

Keywords in C# werden von Visual Studio automatisch mit blauer Farbe gekennzeichnet. Eine farbliche Unterlegung von Programmiercode nennt man auch „Code Highlighting“ oder „Syntax Highlighting“.

Wir konnten aber auch die Variablen frei bezeichnen:
- `double laenge;`
- `double breite;`
- `double resultat;`

In dieser Liste sind die Schlüsselwörter (Keywords) von C# blau markiert, alles andere sind frei definierbare Identifiers.

Keywords C#	
abstract	namespace
as	new
base	null
bool	object
break	operator
byte	out
case	out (generic modifier)
catch	override
char	params
checked	private
class	protected
const	public
continue	readonly
decimal	ref
default	return
delegate	sbyte
do	sealed
double	short
else	sizeof
enum	stackalloc
event	static
explicit	string
extern	struct
false	switch
finally	this
fixed	throw
float	true
for	try
foreach	typeof
goto	uint
if	ulong
implicit	unchecked
in	unsafe
in (generic modifier)	ushort
int	using
interface	virtual
internal	void
is	volatile
lock	while
long	

2. Abschnittsweise Erklärungen zum Beispiel `RechteckFlaeche`
Während sich das prozedurale Programmierparadigma hauptsächlich, mit der Frage der Aufteilung, eines grösseren Problems in Teilprobleme mit Teillösungen befasst, bietet die Objektorientierung **Methoden und Konzepte der Strukturierung von Programmcode für komplexere Anwendungen.**

Objektorientierung ist ein Ansatz, wo Daten und dazugehörige Funktionen in Klassen von Realobjekten zusammengefasst werden.

Einige Ziele der Objektorientierung:

1. Wiederverwendbarkeit von Programmcode → verminderter Programmieraufwand
2. Bessere Lesbarkeit und Verständlichkeit von Programmcode
3. Einfachere Wartbarkeit von Programmcode

Eine **Klasse** ist wie ein allgemeiner *Bauplan* eines konkreten **Realobjekts**. Die Realisierung vom Plan in reales Objekt nennt man **Instanziierung**. Eine Objekt heisst auch *Instanz* einer bestimmten Klasse. Eine Klasse ist mit **Methoden** (Funktionen) und **Eigenschaften** (Daten) definiert. Die Idee einer Klasse ist Daten und dazugehörige Operationen zu einer Einheit zusammenzufassen.

Programmcode-Teil	Erklärung
`using System;`	**Bibliothek** `System` Mit `using` werden unserem Programm `RechteckFlaeche` Funktionen (auch Bibliothek genannt) aus der Befehlsmenge `System` zur Verfügung gestellt. Programmierer und Programmiererinnen können Methoden aus der Bibliothek im Programmcode einfach wiederverwenden. Man spricht auch von „importieren“. `using System;` ist ein abschliessender Programmierbefehl und steht auf einer Zeile und muss mit Semikolon abgeschlossen.

`namespace RechteckFlaeche`
`{`
`...`
`}`
	Namensraum
	Der Namensraum umklammert den gesamten Programm-code mit geschweiften Klammern und ist das äusserste Element.

`class Programm`
`{`
`...`
`}`
	Klasse `class`
	Das Keyword `class` definiert die Klasse mit Namen oder Identifier `Programm`.
	Ein Klasse besteht aus Eigenschaften (Variablen) und Methoden (Aktionen welche das Objekt ausführen kann).

`double laenge;`
`double breite;`
`double resultat;`
	Datentyp: `double` **und Variablen** `laenge, breite` **und** `resultat`
	Die drei Zeilen beschreiben die Eigenschaften von `class Programm`. Das Keyword `double` ist ein Datentyp. Jeder Speicherort in einem C# Programm muss typisiert sein. Ein Speicherort nennt man auch **Variable**. Die Identifier `laenge, breite` und `resultat` sind Variablen. Ein `double` ist eine Kommazahl. Im Anhang i. finden Sie Referenz mit den einfachen Datentypen von C#. *Auf Datentypen wird*

	später noch genau eingegangen. Mit den Identifier `laenge, breite` und `resultat` sagen wir dem Computer, dass unser Programm zwei Eigenschaften in der `class Programm` hat.

`static double FlaecheBerechnen()`
`{`
`return laenge * breite;`
`}`
	`FlaecheBerechnen()` ist der Identifier der Funktion. `double` besagt, dass die Funktion `FlaecheBerechnen()` einen Rückgabewert vom Datentyp `double` hat.
	Das Keyword `return` definiert den Rückgabewert, der aus der Multiplikation `laenge * breite` berechnet wird. `*` steht für eine artihmetische Operation. Weitere Operatoren sind im Anhang ii. Die Datentypen der Eigenschaften `laenge` und `breite` müssen mit dem Datentyp des Rückgabewertes der Funktion `FlaecheBerechnen()` übereinstimmen, in diesem Fall `double`.

`static void Anzeigen()`
`{`
`...`
`}`
	`Anzeigen()` ist der Identifier der Funktion.
	Beschreiben Sie `Anzeigen()`selber:

Die MAIN Methode

```
class RechteckAusfuehren
{
    static void Main(string[] args)
   {
     ...
    }
}
```

Zitiert aus der Microsoft C# Referenz:
*„Die **Main**-Methode ist der Einstiegspunkt einer C#-Konsolenanwendung oder einer Windows-Anwendung. (Für Bibliotheken und Dienste ist keine **Main**-Methode als Einstiegspunkt erforderlich.). Wenn die Anwendung gestartet wird, ist die **Main**-Methode die erste Methode, die aufgerufen wird. Die **Main**-Methode ist der Einstiegspunkt eines EXE-Programms. Dies ist der Punkt, an dem die Programmsteuerung beginnt und endet.*
***Main** wird in einer Klasse oder einer Struktur deklariert.*
***Main** kann entweder einen **void**-Rückgabetyp oder einen **int**-Rückgabetyp aufweisen.*“

`static void Main (string[] args) Main` ist eine **statische Methode** (`static`) ohne Rückgabewert (`void`). In Analogie gibt es auch statische Klassen. Statische Klassen sind gleich wie nicht statische Klassen mit dem einen Unterschied, dass sie nicht instanziiert werden können.“

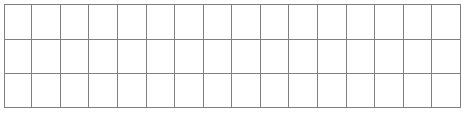
Das Array:
`string[] args` ist eine nummerierte Liste oder indexiertes Array. Die Elemente sind vom Datentyp `string`. Ein `string` ist eine Folge von beliebigen Zeichen (Ziffern, Buchstaben und Sonderzeichen auch Zeichenkette genannt).
Beispiel in C#:

```
string[] Woche = {"Mo","Di","Mi",...,"Fr"};
                  \_____________________/
                    Datenwerte des Arrays
PrintArray(Woche);     //Druckt alle Wochentage
PrintArray(Woche[0]); //Druckt Mo
PrintArray(Woche[1]); //Druckt Di
....
PrintArray(Woche[4]); //Druckt Fr
```

Die Nummerierung der Elemente innerhalt eines Arrays nennt mach auch Indexierung. **Sie beginnt bei 0 und geht nicht bei 1.**

Wie lange ist das Array `Woche`? In C#

```
Console.WriteLine(Woche.Length);
```



Datenwerte als Parameter von Funktionen und formale Parameter in Funktionsdefinitionen

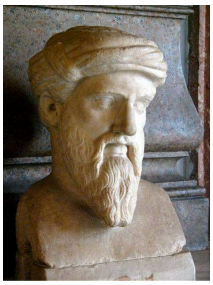
Sie erinnern sich an die einfache Funktionsdefinition `quadrat(x) = x²` aus S. 4.
Die Funktion `quadrat` erhält einen realen Zahlenwert zur Berechnung.
Zum Beispiel den Zahlenwert 6: `quadrat(6) = 36`
Den Identifier `x` in der Funktionsdefinition `quadrat(x) = x²` nennt man **formalen Parameter**. `x²` ist der Rückgabewert der Funktion.

Erklären Sie nun selber die
`static void Main (string[] args):`
Was heisst genau `void` in der Methode
`static void Main (string[] args)`?

Wie nennt man `string[] args` auch in der Methode
`static void Main (string[] args)`?

Von welchen Datentypen ist der formale Parameter `string[] args` in der Methode `static void Main (string[] args)` und wie heisst der Identifier des formalen Parameters?

M403-Codebeispiel-3 C#: Einfach den Pythagoras berechnen



Herme des Pythagoras (um 120 n. Chr.); Kapitolinische Museen, Rom

Erstellen wir ein nächstes Programm, welches nun den Pythagoras berechnet:

- Gehen Sie in Menu DATEI → Wählen Sie Neu → Projekt
- Wählen Sie Visual C# → Konsolenanwendung
- Bezeichnen Sie das Projekt mit `Pythagoras`

Erfassen Sie folgendes Programm und führen Sie es aus:

```
using System;
namespace Pythagoras
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("---------------------------------");
            Console.WriteLine("Pythagoras: c^2 = a^2 + b^2");
            Console.WriteLine("---------------------------------");

            double c, a, b, quadrat_a, quadrat_b, quadratsumme;

            Console.WriteLine("Geben Sie einen Wert für a ein:");
            a = Convert.ToDouble(Console.ReadLine());

            Console.WriteLine("Geben Sie einen Wert für b ein:");
            b = Convert.ToDouble(Console.ReadLine());

            quadrat_a = a * a;
            Console.WriteLine("a im Quadrat = {0}", quadrat_a);

            quadrat_b = b * b;
            Console.WriteLine("b im Quadrat = {0}", quadrat_b);

            quadratsumme = quadrat_a + quadrat_b;
            Console.WriteLine("Quadratsumme von a und b = {0}", quadratsumme);

            c = Math.Sqrt(quadratsumme);
            Console.WriteLine("Hypotenuse c = {0}", c);

            Console.ReadLine();
        }
    }
}
```

Aufgabe 1:
Programmieren Sie das Pythagoras Beispiel neu und versuchen Sie geeignete Bestandteile aus der Main()-Methode ausserhalb dieser zu definieren. Nennen Sie das Projekt `Pythagoras_Prozedural:`

Hinweis:
Sie können einer Methode auch einen Parameter übergeben, der dann in der Berechnung Verwendung findet:

```
using System;

namespace Pythagoras_helper
{

    class Program
    {
        static void drucken(string Text)
        {
            Console.WriteLine("{0}", Text);
            Console.ReadKey();
        }
        static void Main(string[] args)
         {
             drucken("Hallo Welt!");
         }
    }
}
```

Anhang:

i. Einfache Datentypen aus der Microsoft C# Referenz

C# Datentypen	Grösse (bit)	Beschreibung	Wertebereich
sbyte	8	Singed byte	-128 to 127
byte	8	Unsigned byte	0 to 255
short	16	Short integer	-32,768 to 32,767
ushort	16	Unsigned short integer	0 to 65,535
int	32	Integer	-2,147,483,648 to 2,17483,648
uint	32	Unsigned integer	0 to 4,294,967,295
long	64	Long integer	-9223372036854775808 to 9223372036854775808
ulong	64	Unsigned long integer	0 to 18,446,744,073,709,551,615
char	16	Unicode character	any valid character, e.g., a,*, \x0058 (hex), or\u0058 (Unicode)
float	32	Floating point integer	
double	64	Double floating point integer	
bool	1	Logical true/false value	True/False
decimal	128	Used for financial and monetary calculations	

ii. Operatoren aus der Microsoft C# Referenz

Multiplicative	`* / %`
Additive	`+ -`
Shift	`<< >>`
Relational and type testing	`< > <= >= is as`
Equality	`== !=`
Logical	`AND &`
Logical	`XOR ^`
Logical	`OR |`
Conditional	`AND &&`
Conditional	`OR ||`
Conditional	`?:`
Assignment	`= *= /= %= += -= <<= >>= &= ^= |=`