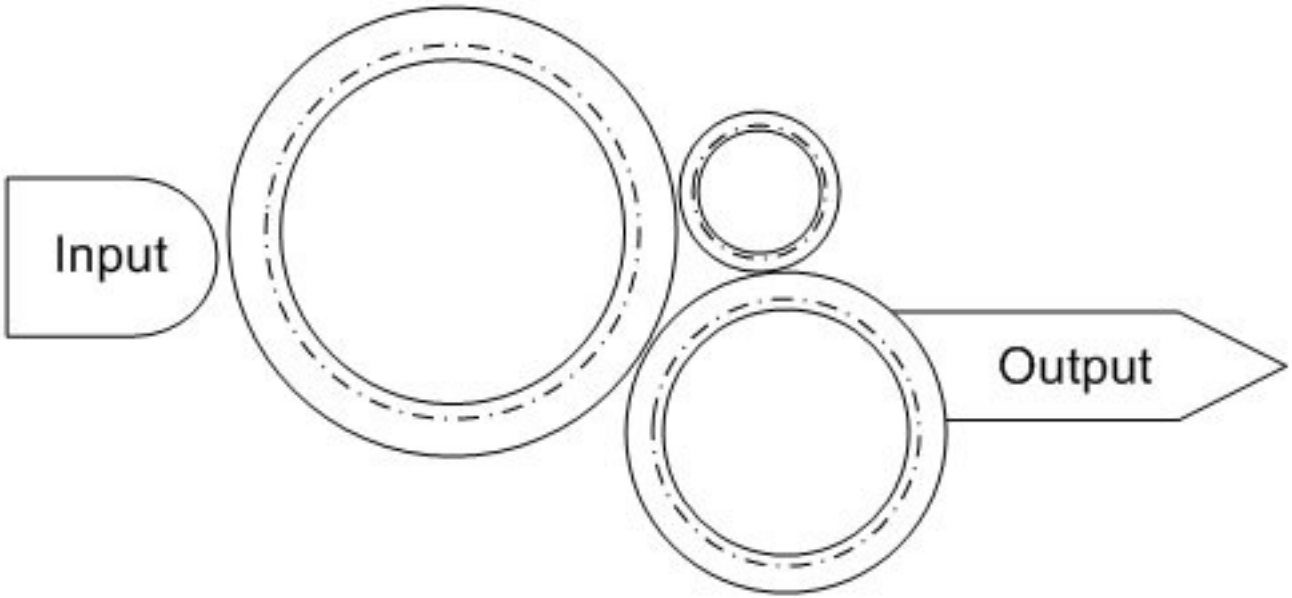
C# - Einführung in die Programmiersprache

„Methoden“



Methoden ...

- sind Subroutinen in einer Klasse.
- können einen Wert an den Aufrufer zurückgeben.
- verändern die Eigenschaften eines Objekts.
- fassen Code zusammen, der von verschiedenen Stellen im Programm aufgerufen werden kann.
- sind eigenständige Codeabschnitte.

Methode „Main“

```
static void Main(string[] args)
{
  int zahl = 0;

  try{
      zahl = int.Parse(Eingabe());
      Ausgabe("Eingegebener Wert: " + zahl);
  }
  catch (Exception exc) {
       Console.WriteLine("Fehler: " + exc.Message);
  }
}
```

Erläuterung

- Einstiegspunkt für jede Konsolenanwendung.
- `void Main`: Die Methode gibt keinen Wert an den Aufrufer zurück.
- `void Main(string[] args)`: Der Methode werden die Befehlszeilenargumente übergeben. Die Parameterliste für eine Methode beginnt und endet mit den runden Klammern.
- Der, zu der Methode gehörende Codeabschnitt beginnt und endet mit den geschweiften Klammern.

Statische Methoden ...

- werden mit dem Schlüsselwort `static` gekennzeichnet.
- werden als Klassenmethoden bezeichnet.
- werden unabhängig von einem Objekt genutzt.
- können nur statische Variablen und Methoden aufrufen.

Benutzerdefinierte Methoden

```
static string Eingabe()
{
  string resulat;

  Console.WriteLine("Bitte geben Sie eine Zahl ein: ");
  resulat = Console.ReadLine();
  return resulat;
}

static void Ausgabe(string message)
{
  Console.WriteLine(message);
}
```

… bestehen aus den ...

- Methodenkopf. Der Kopf definiert den Aufruf einer Methode. Der Methodenkopf bietet eine Schnittstelle zu den Nutzern.
- Methodenrumpf. Der Rumpf wird durch die geschweiften Klammern begrenzt. Innerhalb der Klammern stehen die, zur Methode gehörenden Anweisungen.

Methodenkopf

```
void Ausgabe(string message)
[Datentyp] [Name] (parameter01, parameter02, …)
```

- Der Datentyp der Methode legt den Rückgabetyp fest. Der Typ `void` kennzeichnet eine Methode ohne Rückgabewert.
- Der Name der Methode ist frei wählbar.
- In den runden Klammern werden der Methode Parameter, getrennt durch ein Kommata übergeben. Falls die Klammer leer ist, werden der Methode keine Werte übergeben. Die Parameter können in der Methode verarbeitet werden.

Der Name der Methode ...

- wird aus den Zeichen A..Z, a..z, 0..9 und dem Unterstrich gebildet.
- sollte immer mit einem Buchstaben oder dem Unterstrich beginnen.
- beginnt häufig mit einem Großbuchstaben.
- stellt eine zusammenhängende Zeichenfolge dar.
- kommt einmal in einem Programm vor.
- sollte die Funktion widerspiegeln. Häufig werden Verben im Methodennamen genutzt.

Rückgabewert

```
static string Eingabe()
{
  string resulat;

  Console.WriteLine("Bitte geben Sie eine Zahl ein: ");
  resulat = Console.ReadLine();

  return resulat;
}
```

Erläuterung

- Mit Hilfe des Schlüsselwortes `return` wird ein Wert an den Aufrufer zurückgegeben.
- Der Datentyp des Rückgabewertes entspricht dem Datentyp der Methode. Andernfalls muss der Rückgabewert in den entsprechenden Datentyp konvertiert werden.
- Eine Methode kann immer nur einen Wert zurückgeben.

Methoden aufrufen

```
static void Main(string[] args)
{
  int zahl = 0;

  try{
       zahl = int.Parse(Eingabe());
       Ausgabe("Eingegebener Wert: " + zahl);
  }
  catch (Exception exc) {
       Console.WriteLine("Fehler: " + exc.Message);
  }

}
```

Erläuterung

- Methoden werden immer mit ihren Namen aufgerufen.
- Falls die Parameterliste nicht leer ist, werden in den runden Klammern die Werte für die Parameter in runden Klammern übergeben.
- Falls die Methode einen Rückgabewert hat, kann dieser in einer Variablen gespeichert werden.

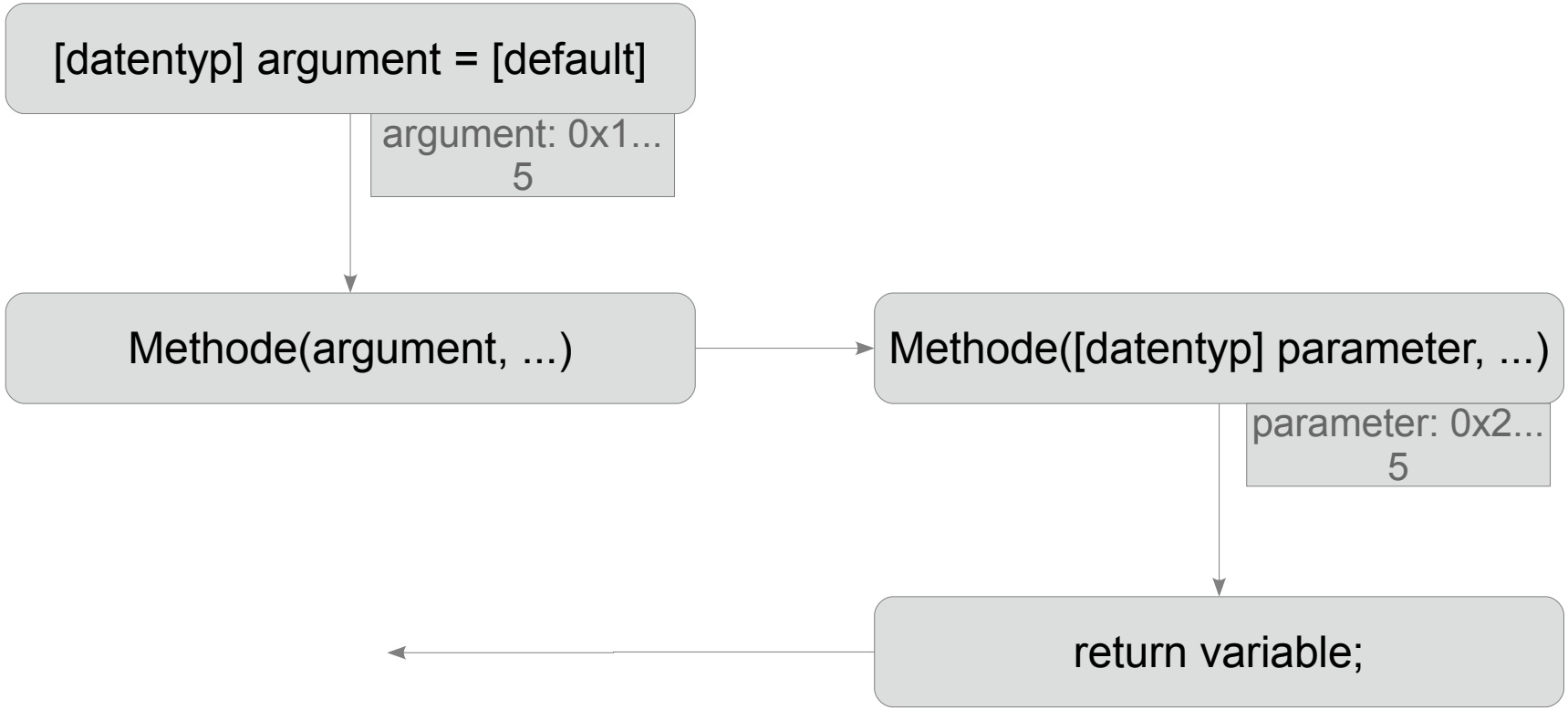
Parameter ...

- sind Platzhalter für Werte im Speicher.
- werden vom Aufrufer an die Methode übergeben.
- bekommen einen Wert oder eine Speicheradresse (`out`, `ref`) zugewiesen.
- werden in Abhängigkeit ihrer Position oder des Namens übergeben.

Wertparameter

- Das Argument übergibt einen Wert an den Parameter.
- Der Parameter ist eine Kopie des Arguments.
- Das Argument kann nicht durch die aufgerufene Methode verändert werden.

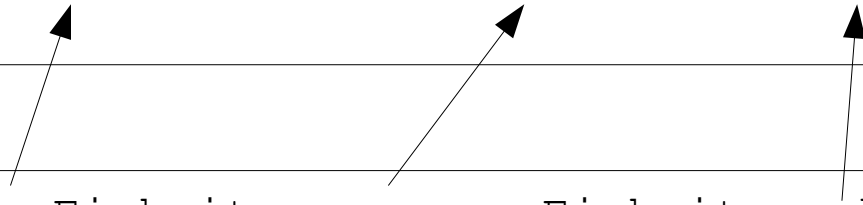
Ablauf


[datentyp] argument = [default]
argument: 0x1...5
Methode(argument, ...)
Methode([datentyp] parameter, ...)
parameter: 0x2...5
return variable;

… in C#

```
static float Change(string jetzt, string neu, float wert)
```

```
neu_Zahl = Umwandlung(massEinheit, neu_massEinheit, zahl);
```



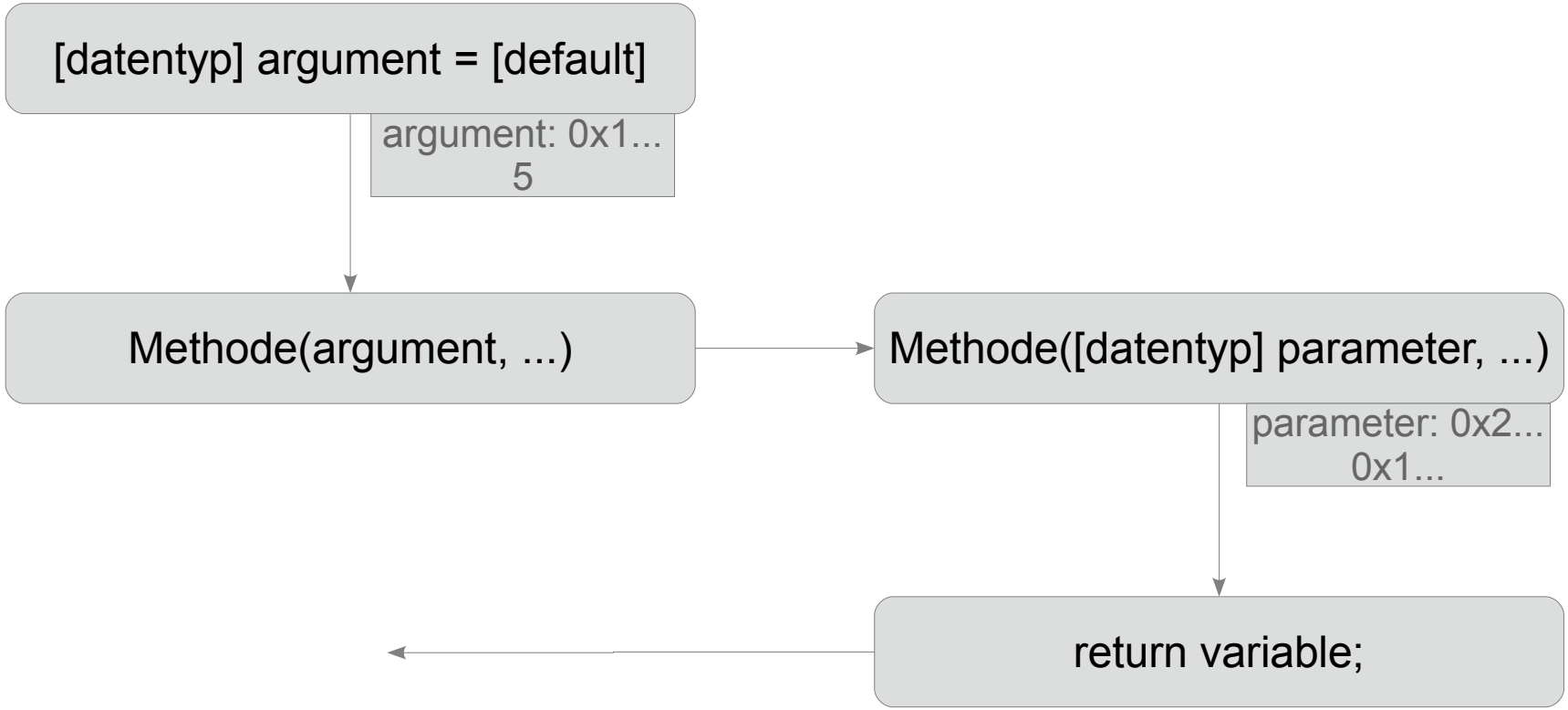
Referenzparameter (call by reference)

- Verweis auf eine bestimmte Speicherstelle.
- Parameter und Argument zeigen auf die gleiche Schublade.
- Innerhalb der aufgerufenen Methode kann der Wert des Arguments verändert werden.

Schlüsselwort ref

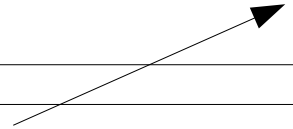
- Das Argument muss vor der Nutzung initialisiert werden.
- Im Methodenkopf und im Aufruf werden die Variablen mit dem Schlüsselwort gekennzeichnet.

Ablauf


[datentyp] argument = [default]
argument: 0x1... 5
Methode(argument, ...)
Methode([datentyp] parameter, ...)
parameter: 0x2... 0x1...
return variable;

… in C#

```
static void Ausgabe(ref string message){}
```

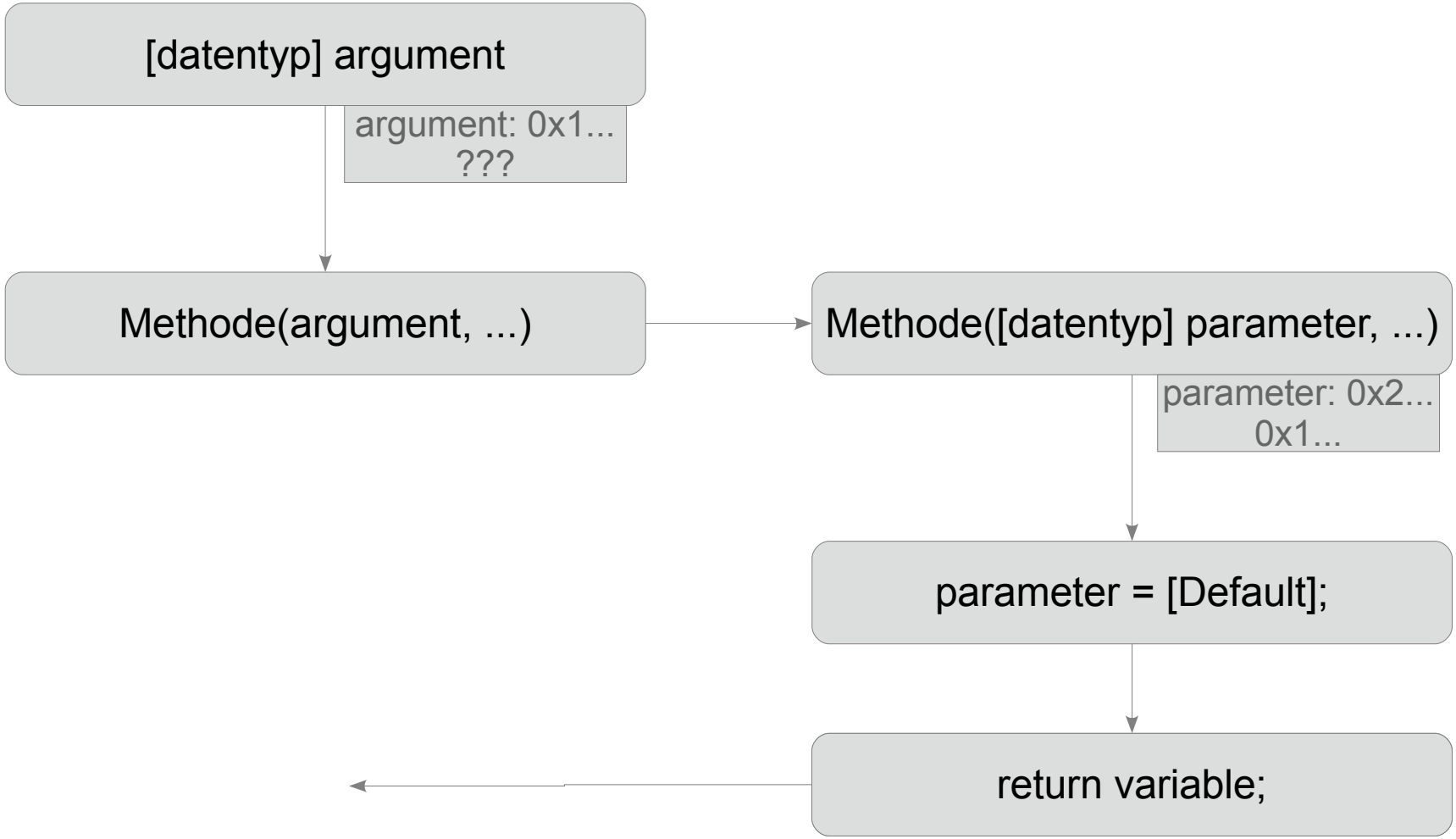


```
Ausgabe(ref message);
```

Schlüsselwort out

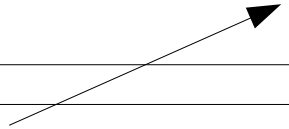
- Das Argument muss vor der Nutzung definiert werden.
- In der aufgerufenen Methode muss der dazugehörige Parameter vor der Nutzung initialisiert werden.
- Im Methodenkopf und im Aufruf werden die Variablen mit dem Schlüsselwort gekennzeichnet.

Ablauf


[datentyp] argument
argument: 0x1...
???
Methode(argument, ...)
Methode([datentyp] parameter, ...)
parameter: 0x2...
0x1...
parameter = [Default];
return variable;

… in C#

```
static void Ausgabe(out string message){}
```

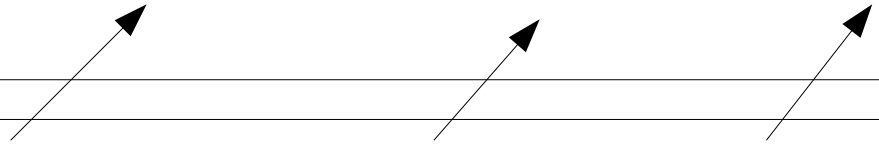


```
Ausgabe(out message);
```

Mischform

```
static string lesen_Adresse(string person, ref string str, out string ort){}
```

```
kunde = lesen_Adresse(kundeName, ref Strasse, out Ort);
```

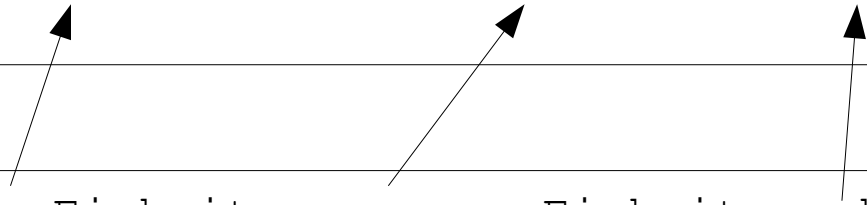


- Die Parameter können in beliebiger Form gemischt werden.

Positionsargumente

```
static float Change(string jetzt, string neu, float wert)
```

```
neu_Zahl = Umwandlung(massEinheit, neu_massEinheit, zahl);
```



- In Abhängigkeit der Position werden die Argumente an die Parameter übergeben.
- Die Anzahl der Argumente entspricht der Anzahl der Parameter.

Benannte Argumente

```
static float Change(string jetzt, string neu, float wert)
```

```
neu_Zahl = Umwandlung(jetzt: massEinheit, neu;
                      neu_massEinheit, wert: zahl);
```

- `parameter: argument`.
- Die Argumente werden den Parametern in Abhängigkeit des Namens zugeordnet.
- Die Reihenfolge der Parameter ist beliebig.
- Ein benanntes Argument kann einem Positionsargument folgen, aber nicht umgekehrt.

Optionale Parameter ...

```
static float Change(float wert,
                    string jetzt = "m", string neu = "cm")
```

```
neu_Zahl = Umwandlung(jetzt: massEinheit, neu;
                      neu_massEinheit, wert: zahl);
```

- [datentyp]parameter = [wert].
- werden in der Liste initialisiert.
- haben einen Standardwert.
- stehen immer am Ende der Parameterliste.

Anweisungsblock ...

- beginnt und endet immer mit den geschweiften Klammern.
- fasst Anweisungen zusammen.
- kann lokale Variablen nutzen.
- können verschachtelt werden.

Lokale Variablen

- sind nur innerhalb eines Blockes definiert.
- können nur innerhalb des Blockes genutzt werden, in dem sie definiert sind.
- kommen in einem Block exakt einmal vor.
- werden beim Verlassen des Blockes zerstört.

… in C#

```
class Program
{
        static string message;

        static string Eingabe()
        {
            string result;

            Console.WriteLine(message);
            result = Console.ReadLine();
            return result;
        }
}
```

… in C#

```
 for (int zahlR = 1; ; zahlR++)
{
  if ((zahlR > 10) || (zahlL > 10)){
      break;
  }
  if (((zahlL % 2) > 0) && ((zahlR % 2) > 0)){
     ergebnis = zahlL * zahlR;
  }
  else{
       continue;
  }

}
```

Attribute einer Klasse ...

- werden innerhalb einer Klasse, aber außerhalb einer Methode definiert.
- kann von jeder Methode der Klasse genutzt werden.
- werden definiert, wenn mehr als eine Methode die Variable nutzen.
- werden durch das Schlüsselwort `static` einmal für eine Klasse angelegt.

… in C#

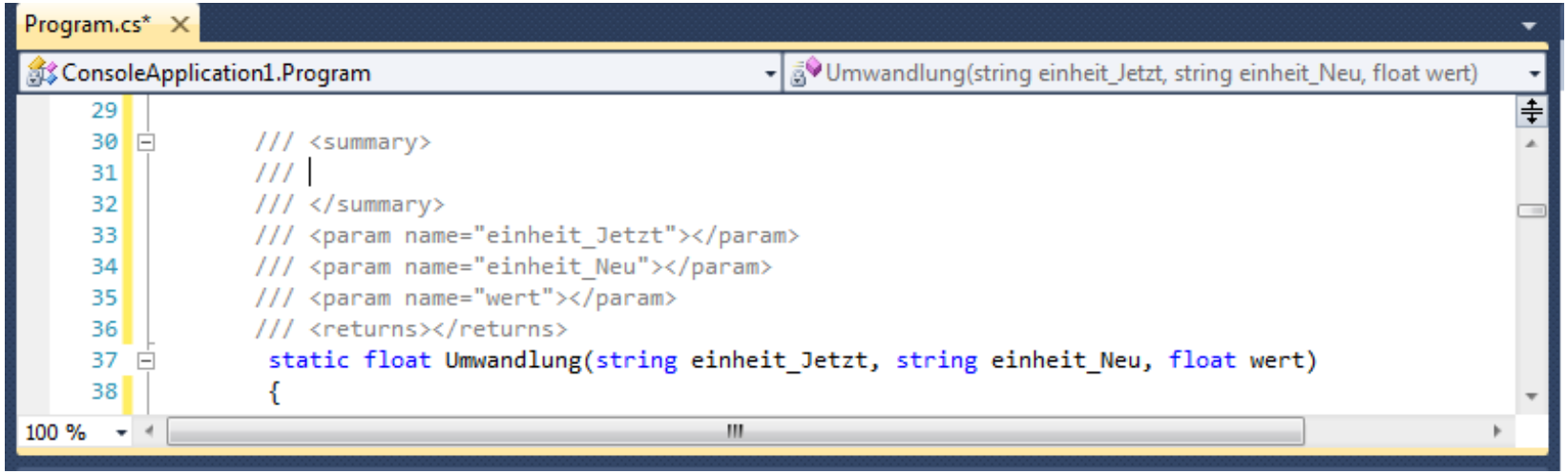
```
class Program
{
        static string message;

        static string Eingabe()
        {
            string result;

            Console.WriteLine(message);
            result = Console.ReadLine();
            return result;
        }
}
```

Methodenbeschreibung

- Eingabe von drei Schrägstrichen in der Zeile vor dem Methodenkopf.
- Sobald der dritte Schrägstrich eingetippt ist, werden automatisch weitere Zeilen eingefügt.



Aufbau

- Funktionsweise der Methode:

```
/// <summary>
/// [Beschreibung]
/// </summary>.
```

- Beschreibung der Parameter:

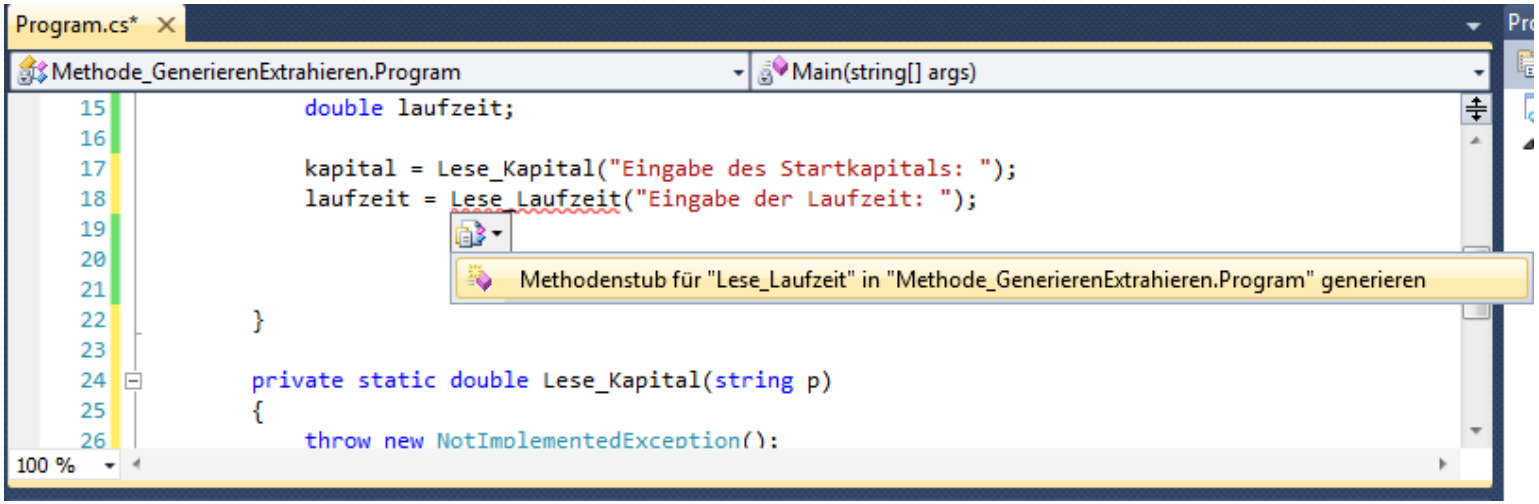
```
/// <param name="name">[Beschreibung]</param>
```

- Beschreibung des Rückgabewertes:

```
/// <returns>[Beschreibung]</returns>
```

Methode automatisch generieren

- Der Aufruf der Methode wird geschrieben.
- Mausklick auf den Smarttag der Methode.
- Mausklick auf die Methode *Methodenstub für … generieren*.
- Das dazugehörige Methodengerüst wird automatisch erstellt.



Generische erzeugte Methode

```
private static double Lese_Kapital(string p)
{
    throw new NotImplementedException();
}
```

- In der generierten Methode wird eine Exception geworfen (`throw`).

Methode aus Code erzeugen

- Der Code für die Methode ist markiert.
- Rechter Mausklick, um das Kontextmenü zu öffnen.
- Auswahl des Befehls *Umgestalten – Methode extrahieren*.
- In dem Dialog *Methode extrahieren* wird ein passender Methodenname eingegeben.