

Funktionen oder statische Methoden

in der Programmiersprache C#

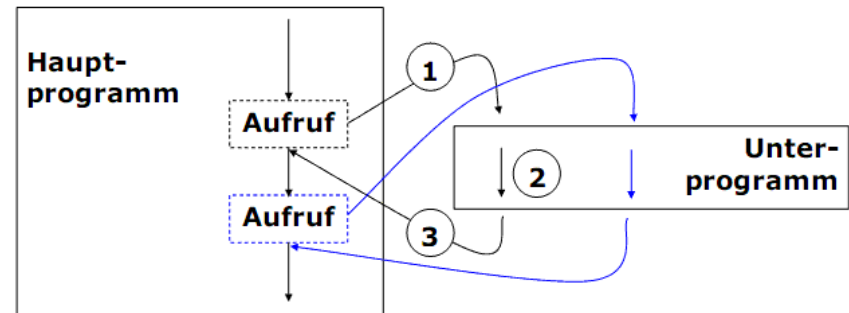
```
DO
  GOSUB Subroutine_Name
  DEBUG "Next command"
LOOP

Subroutine_Name:
  DEBUG "This is a subroutine..."
  PAUSE 3000
RETURN
```

The diagram illustrates a subroutine call mechanism. It shows a main loop structure with a `DO` statement, followed by a `GOSUB Subroutine_Name` statement, a `DEBUG "Next command"` statement, and a `LOOP` statement. A curved arrow points from the `GOSUB` statement to the start of the subroutine definition, which begins with `Subroutine_Name:`. Inside the subroutine, there is a `DEBUG "This is a subroutine..."` statement, a `PAUSE 3000` statement, and a `RETURN` statement. A curved arrow points from the `RETURN` statement back to the line immediately following the `GOSUB` statement, indicating the return path.

Überblick

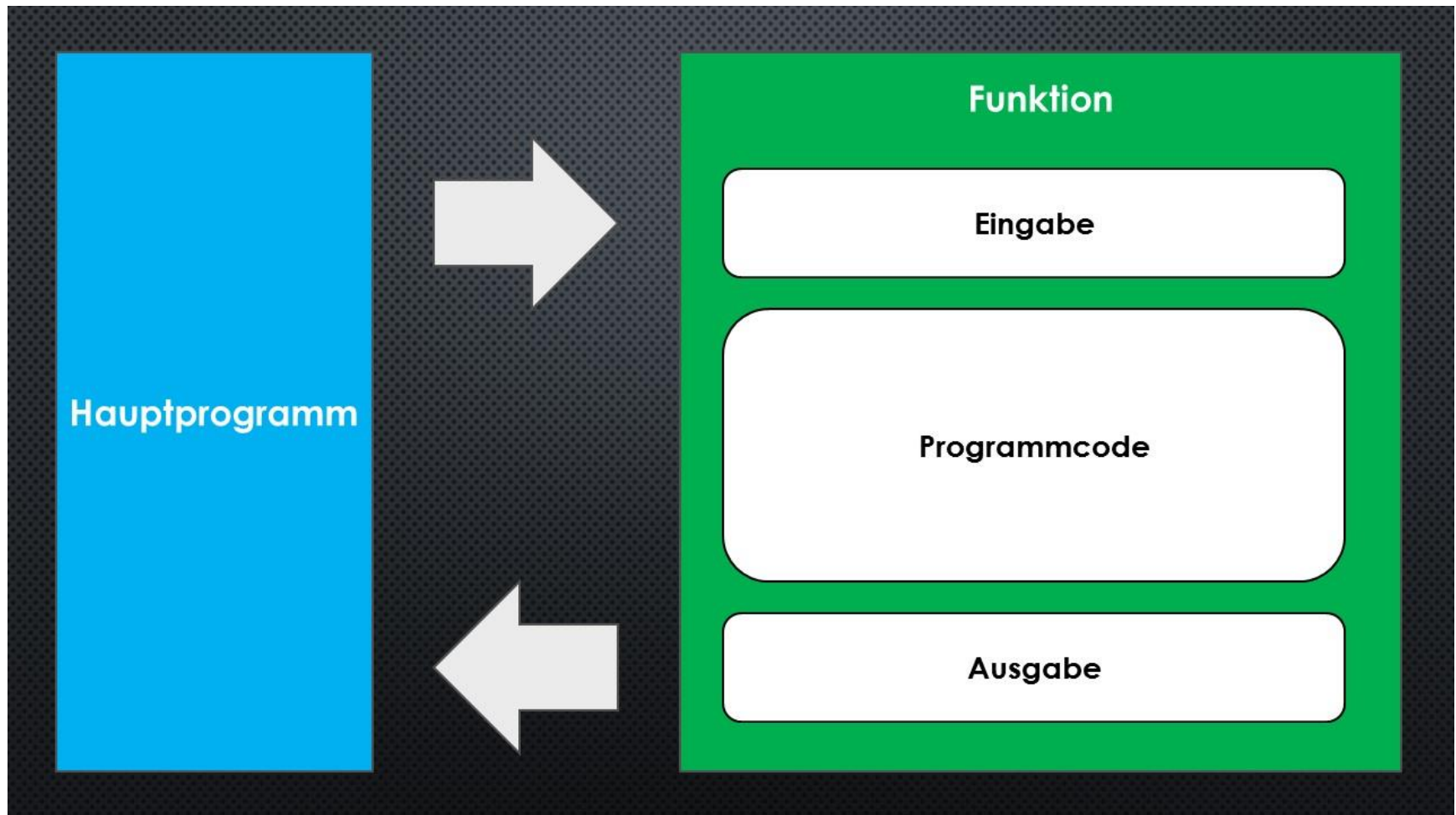
- Definition von Funktion
- Syntax von Funktionen
- Zugriffsmodifizierer
- Statische Funktionen
- Aufruf von Funktionen
- Rückgabewert
- Return-Anweisung
- Lokale Variablen
- Arrays als Parameter



Beispiele:

Taschenrechner, Power

Funktion



Funktion

- Programme in übersichtliche Teile zerlegen
- Teile können weiter unterteilt werden
- usw.

- Entwurf dieser Hierarchie nennt sich Prinzip der schrittweisen Verfeinerung
- -> **TOP-DOWN Design**

- Funktion
 - einmal definiert
 - beliebig oft durch Nennung ihres Namens aufrufen

Eine Funktion ist **wiederverwendbar!**

Funktionen

- sind Teilprogramme
- **erledigen** eine abgeschlossene **Teilaufgabe**
- verarbeitenden Werte werden mitgegeben
sogenannte **Argumente** bzw. **Parameter**
- **liefern** das **Ergebnis** zurück = **Rückgabewert**

```
Random r = new Random();
```

```
int number = r.Next(0,100);
```

- Next ist eine Funktion, die Grenzen als Parameter erhält und eine Zufallszahl als Ergebnis zurück gibt.

Syntax einer Funktion

```
[Modifizierer] Typ Bezeichner([Parameterliste]) {  
    ...  
    return Wert;  
}
```

- Funktionen erhalten
 - einen Funktionsnamen (Bezeichner)
 - können Werte (Argumente/Parameter) erhalten
 - können Ergebniswerte zurück geben (Rückgabewert)
 - Modifizierer ist optional: (Standardwert private)
 - beeinflussen die Sichtbarkeit

Zugriffsmodifizierer

Zugriffsmodifizierer	Beschreibung
public	Der Zugriff unterliegt keinerlei Einschränkungen.
private	Der Zugriff auf ein als private definiertes Mitglied ist nur innerhalb der Klasse möglich, die das Member definiert. Alle anderen Klassen sehen private Member nicht. Deshalb ist darauf auch kein Zugriff möglich.
protected	Der Zugriff auf protected Member ähnelt dem Zugriff auf als private definierte Member. Die Sichtbarkeit ist in gleicher Weise eingeschränkt, jedoch sind als protected definierte Mitglieder in abgeleiteten Klassen sichtbar. Zu diesem Thema folgt später in diesem Kapitel noch mehr.
internal	Der Zugriff auf internal Member ist nur aus den Klassen heraus möglich, die sich in derselben Anwendung befinden.
protected internal	Stellt eine Kombination aus den beiden Modifizierern protected und internal dar.

Funktion ohne Parameter & ohne Rückgabe:

- Erstelle eine Funktion die eine Zeichenkette in der Konsole ausgibt.

```
public static void PrintName()
{
    Console.WriteLine("*****");
    Console.WriteLine("*** Hello Mike Rohsoft ***");
    Console.WriteLine("*****");
}
```


Funktion mit Parameter & ohne Rückgabewert

- Erstelle eine Funktion die einen Namen als Parameter erhält und eine Zeichenkette in der Konsole ausgibt

```
//Funktion mit Parameter ohne Rückgabewert|
public static void PrintName(String name)
{
    Console.WriteLine("*****");
    Console.WriteLine("*** Hello {0} ***", name);
    Console.WriteLine("*****");
}
```

Funktion mit 2 Parametern & ohne Rückgabewert

- Übergib Name und Ort als Parameter, gib beides in der Console aus
- Setze die passende Anzahl an Sternen, und gib diese aus.

```
//Funktion mit 2 Parametern und ohne Rückgabewert
public static void PrintPerson(String name, String location)
{
    String s = String.Format("*** Hallo {0} aus {1} ***", name, location);
    String stars = new String('*', s.Length);
    Console.WriteLine(stars);
    Console.WriteLine(s);
    Console.WriteLine(stars);
}
```

Funktion mit 2 Parametern & mit Rückgabewert

- Funktion die 2 Zahlen addiert / multipliziert
- Das Ergebnis wird retour gegeben

```
//Funktion mit 2 Int als Parameter und der Summe als Rückgabewert  
public static int Sum(int a, int b)  
{  
    return a + b;  
}
```

```
//Funktion mit 2 Int als Parameter und dem Produkt als Rückgabewert  
public static int Mul(int a, int b)  
{  
    return a * b;  
}
```

Aufruf von statischen Funktionen

- Echo: eine Nachricht wird als Parameter übergeben und wird mit „Echo + Nachricht“ zurück gegeben

```
public static String Echo(String message)
{
    return "Echo: " + message;
}
```

```
static void Main(string[] args)
{
    String message = Echo("Hallo Welt");
    Console.WriteLine(message);
}
```

Ausgabe:

Aufgabe Addition

- Addiere 2 Zahlen in der Add Funktion, gib das Ergebnis retour

```
static void Main(string[] args) {  
    TestAdd();  
}  
#region Add  
static void TestAdd() {  
    int result = Add(3, 4);  
    Console.WriteLine(result);  
}  
static int Add(int a, int b) {  
    return a + b;  
}  
#endregion
```

Rückgabewert

- Funktionen können einen Wert zurückliefern

```
public static int DoSomething() {  
    return 7;  
}
```

- Funktionen die keinen Wert zurückliefern werden Prozeduren genannt

```
public static void DoSomething() { /*...*/ }
```

- Syntax von Prozeduren:

```
[Modifizierer] static void Bezeichner([Parameterliste]) { ... }
```

Funktion vs Prozedur

- **Funktion** hat einen Rückgabewert

- `public static int DoSomething() { /*...*/ }`
→ Aufruf: `int x = DoSomething();`

- **Prozedur** hat keinen Rückgabewert

- `public static void DoSomethingElse() { /*...*/ }`
→ Aufruf: `DoSomethingElse();`

- **Methoden**

- **sind das Verhalten von Objekten in der OOP**

Return-Anweisung

- return-Anweisung muss in Funktionen passend zum Datentyp des Rückgabewertes vorhanden sein.

```
public static int DoSomething() {  
    return 7;  
}
```

- Optional kann eine Return-Anweisung auch in Prozeduren genutzt werden um diese vorzeitig zu verlassen.

Taschenrechner

- Einlesen auslagern
- Add/ Mul/ Sub / Div Funktionen schreiben

```
static void Main(string[] args)
{
    int x = GetValue();
    int y = GetValue();
    int result = Add(x, y);

    Console.WriteLine(result);
}
```

Aufruf in der Main:

- 2 Werte einlesen
- Funktion mit Berechnung
- Ausgabe des Rückgabewerts

Einlesen eines Wertes

- Erstelle eine Funktion, die eine Zahl einliest, und diese als Rückgabewert retour gibt:

```
public static int GetValue()  
{  
    Console.WriteLine("Bitte geben Sie eine Zahl ein");  
    return Convert.ToInt16(Console.ReadLine());  
}
```

Funktion: Add, Sub, Mul, Div

- Erstelle pro Rechenoperation eine Funktion, die 2 Werte erhält, die gewünschte Berechnung laut Funktionsnamen durchführt und das Ergebnis retour gibt:

```
public static int Add(int a, int b)
{
    return a + b;
}
```

```
public static int Mul(int a, int b)...
```

0 Verweise

```
public static int Sub(int a, int b)...
```

0 Verweise

```
public static double Div(double a, double b)...
```

```
static void Main(string[] args) {  
    Console.WriteLine("Bitte wählen Sie zwischen Add/Mul/Div/Sub");  
  
    string read = Console.ReadLine();  
    int number1 = GetNumber();  
    int number2 = GetNumber();  
    double result;  
    switch (read) {  
        case "Add":  
            result = AddNumbers(number1, number2); break;  
        case "Sub":  
            result = SubNumbers(number1, number2); break;  
        case "Mul":  
            result = MulNumbers(number1, number2); break;  
        case "Div": {  
            result = DivNumbers(number1, number2); break;  
        }  
        default:  
            Console.WriteLine("Falsche Eingabe");  
            result = 0.0;  
            break;  
    }  
    Console.WriteLine("Das Ergebnis lautet {0}", result);  
}
```

Taschenrechner mit Main

```
static int GetNumber() {  
    Console.WriteLine("Bitte geben  
    int number = Int32.Parse(Console  
    return number;  
}  
  
static int AddNumbers(int x, int y)  
    return x + y;  
}  
static int SubNumbers(int x, int y)  
    return x - y;  
}  
static int MulNumbers(int a, int b)  
    return a * b;  
}  
static double DivNumbers(int num1,  
    return num1 / num2;  
}
```

Lokale Variablen

- sind Variablen innerhalb einer Funktion

```
public static void MyMethod() {  
    long x = 34;  
    ...  
}
```

- **nur in der Methode sichtbar**, in der sie deklariert sind
- außerhalb der Funktion „existiert“ die Variable nicht
 - kein Zugriff, Auswertung oder Veränderung möglich
- **Lebensdauer** einer lokalen Variablen
ist auf die Dauer der **Methodenausführung** begrenzt

Schlüsselwort static

- Funktionale Programmierung
 - enthält eine Liste von Funktionen (= Funktionalitäten)
- Objektorientierte Programmierung enthält Klassen mit
 - Eigenschaften (Attribute) und
 - Funktionalitäten (Methoden)

C# ist objektorientiert

- um funktional programmieren zu können benötigt man das Schlüsselwort „**static**“
- damit kann die Methode ohne Instanz genutzt werden
 - `public static void DoSomething() { /*...*/ }`
 - `public static void DoSomethingElse() { /*...*/ }`

Gliedern von Methoden

Wieviel Code kommt in ein Teilprogramm?

Wie benennt man Teilprogramme?

Welchen Rückgabewert wählt man für ein Teilprogramm?



Programmbeispiel: Power ($x*x$)

- Erstelle ein Programm, das eine Zahl vom Benutzer einliest und diese dann potenziert: $x = x*x$. Das Ergebnis soll in der Konsole ausgegeben werden:

```
Enter a number 5  
Power of 5 = 25
```

```
//Version 1:  
static void Main(string[] args)  
{  
    int num;  
    Console.Write("Enter a number\t");  
    num = Convert.ToInt32(Console.ReadLine());  
    Console.WriteLine("Power of {0} = {1}", num, num * num);  
}
```


Version 2: alles auslagern

```
namespace Funktionen
{
    class Program
    {
        //Version 2:
        //Create static method
        public static void Power()
        {
            int num;
            Console.Write("Enter a number\t");
            num = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Power of {0} = {1}", num, num * num);
        }
        static void Main(string[] args)
        {
            Power();
        }
    }
}
```

Version3:

Berechnung und Ausgabe auslagern

```
class Program
{
    //Version 3:
    //Create static method
    public static void Power(int num)
    {
        Console.WriteLine("Power of {0} = {1}", num, num * num);
    }
    static void Main(string[] args)
    {
        int num;
        Console.Write("Enter a number\t");
        num = Convert.ToInt32(Console.ReadLine());
        Power(num);
    }
}
```

Version4: Nur Berechnung auslagern

```
class Program
{
    //Version 4:
    public static int Power(int num)
    {
        return num * num;
    }
    static void Main(string[] args)
    {
        int num, result;
        Console.Write("Enter a number\t");
        num = Convert.ToInt32(Console.ReadLine());
        result = Power(num);
        Console.WriteLine("Power of {0} = {1}", num, result);
    }
}
```

Teilprogramm - Funktion

- Wieviel Code kommt in ein Teilprogramm?
 - So wenig wie möglich pro Funktion
- Wie benennt man Teilprogramme?
 - Nutze ein Verb, das die Funktionalität beschreibt
 - `PrintArray(int[] arr)`
 - `InitializeArray(int[] arr)`
 - Methodennamen sind Groß und in CamelCase zu schreiben
- Welchen Rückgabewert wählt man?
 - Rückgabedatentyp und Datentyp der das Ergebnis entgegennimmt müssen zusammenpassen.

```
public int GetMaxValue(int[] arr) {  
    int maxValue = arr[0];  
    foreach (int element in arr)  
        if (element > maxValue)  
            maxValue = element;  
    return maxValue;  
}
```

Array als Parameter

Es können mehrere Werte in Form von Arrays an eine Funktion übergeben werden:

Array als Parameter

- Array können als Parameter mitgegeben werden
- Anschließend kann in der Funktion darauf zugegriffen werden

```
static void PrintArray(int[] arr) {  
    for (int i = 0; i < arr.Length; i++) {  
        Console.Write($"{arr[i]}, ");  
    }  
}
```

```
}
```

Beispiel: Summe vom Array

- Erstelle eine Funktion zum Initialisieren eines beliebig großen Arrays
- Erstelle eine Funktion für die Ausgabe eines beliebigen Arrays
- Erstelle Funktion für die Berechnung der Summe eines beliebigen Arrays

Arrays

//Initialisieren des Array mit einer Funktion

 (Array als Rückgabewert und Anzahl der Elemente als Parameter

//Initialisiern mit Random Werten zwischen 0 und 100

```
public static int[] InitializeArray(int amount)
{
    int[] arr = new int[amount];

    Random rand = new Random();
    int min = 0;
    int max = 101;

    for (int i = 0; i < amount; i++)
        arr[i] = rand.Next(min, max);
    return arr;
}
```

//Funktion mit einem Int-Array als Parameter
//und der Summe aller Werte als Rückgabewert

```
public static int SumArray(int[] arr)
{
    int sum = 0;
    foreach (int item in arr)
        sum += item;
    return sum;
}
```

//Funktion für die Ausgabe eines Arrays

```
public static void PrintArray(int[] arr)
{
    foreach(int item in arr)
        Console.Write(item + ", ");
    Console.WriteLine();
}
```

```
static void Main(string[] args)
{
```

//Array als Parameter

```
int[] myArray = InitializeArray(5);
```


```
PrintArray(myArray);
```

```
int sum = SumArray(myArray);
```

```
Console.WriteLine("Die Summe beträgt: {0} ", sum);
```

```
}
```


Initialisiere Array

```
//Initialisieren des Array mit einer Funktion  
 (Array als Rückgabewert und Anzahl der Elemente als Parameter  
//Initialisiern mit Random Werten zwischen 0 und 100  
public static int[] InitializeArray(int amount)|  
{  
    int[] arr = new int[amount];  
  
    Random rand = new Random();  
    int min = 0;  
    int max = 101;  
  
    for (int i = 0; i < amount; i++)  
        arr[i] = rand.Next(min, max);  
    return arr;  
}
```

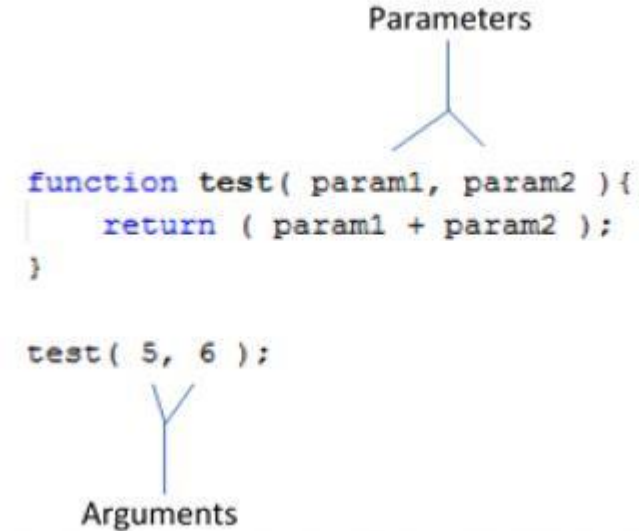
Summe eines Arrays berechnen

```
//Funktion mit einem Int-Array als Parameter  
//und der Summe aller Werte als Rückgabewert  
public static int SumArray(int[] arr)  
{  
    int sum = 0;  
    foreach (int item in arr)  
        sum += item;  
    return sum;  
}
```

Ausgabe

```
//Funktion für die Ausgabe eines Arrays
public static void PrintArray(int[] arr)
{
    foreach(int item in arr)
        Console.Write(item + ", ");
    Console.WriteLine();
}
```

```
static void PrintArray(int[] arr) {
    for (int i = 0; i < arr.Length; i++) {
        Console.Write($"{arr[i]}, ");
    }
}
```



Parameterübergabe

Primitive Datentypen vs Referenzdatentypen als Parameter

Parameterübergabe als KOPIE

- Call by Value
 - Parameter werden kopiert
 - neue Variable wird im Stack angelegt
 - Funktion erhält eine Kopie des Wertes

```
static void Main(string[] args) {  
    int x = 3;  
    DoSomething(x);  
    Console.WriteLine(x);  
}  
  
static void DoSomething(int a) {  
    a = 4;  
    Console.WriteLine(a);  
}
```

Stack
a = 4
x = 3



Parameter übergeben

```
static void TestPara(int x, int y) {  
    x = 5;  
    y = 4;  
    Console.WriteLine("Methode");  
    Console.WriteLine($"x={x} y={y}");  
}
```

```
static void Main(string[] args) {  
    int x = 3;  
    int y = 2;  
    TestPara(x, y);  
    Console.WriteLine("Main");  
    Console.WriteLine($"x={x} y={y}");  
}
```

Wie lautet die Ausgabe?

```
Methode  
x=5 y=4  
Main  
x=3 y=2
```

Parameter übergeben

```
static void TestPara() {  
    int x = 2;  
    int y = 3;  
    TestPara1(x, y);  
    Console.WriteLine("Main");  
    Console.WriteLine($"x={x} y={y}");  
    int res = TestPara2(x, y);  
    Console.WriteLine("Main");  
    Console.WriteLine($"x={x} y={y} res={res}");  
}  
  
static void TestPara1(int x, int y) {  
    x = 5;  
    y = 4;  
    Console.WriteLine("Methode1");  
    Console.WriteLine($"x={x} y={y}");  
}  
  
static int TestPara2(int a, int b) {  
    a = 6;  
    b = 3;  
    Console.WriteLine("Methode2");  
    Console.WriteLine($"a={a} b={b}");  
    return a + b;  
}
```

```
static void Main(string[] args) {  
    TestPara();  
}
```

- Wie lautet die Ausgabe?

```
Methode1  
x=5 y=4  
Main  
x=2 y=3  
Methode2  
a=6 b=3  
Main  
x=2 y=3 res=9
```

Parameter übergeben

```
#region intPrameter
public static void TestPara() {
    int x = 2;
    int y = 3;
    TestPara1(x, y);
    Console.WriteLine("Main");
    Console.WriteLine($"x={x} y={y}");
    int res = TestPara2(x, y);
    Console.WriteLine("Main");
    Console.WriteLine($"x={x} y={y} res={res}");
}

static void TestPara1(int x, int y) {
    x += 5;
    y += 4;
    Console.WriteLine("Methode1");
    Console.WriteLine($"x={x} y={y}");
}

static int TestPara2(int a, int b) {
    a += 6;
    b += 3;
    Console.WriteLine("Methode2");
    Console.WriteLine($"a={a} b={b}");
    return a + b;
}
#endregion
```

```
static void Main(string[] args) {
    TestPara();
}
```

- Wie lautet die Ausgabe?

```
Methode1
x=7 y=7
Main
x=2 y=3
Methode2
a=8 b=6
Main
x=2 y=3 res=14
```


Parameter & Rückgabewert

mit eigener Klasse/Datei gelöst

```
static void Main(string[] args) {
    ParameterClass.TestPara();
}
```

Wie lautet die
Ausgabe?

```
Methode1
x=6 y=3
Main
x=2 y=3
Methode2
a=4 b=4
Main
x=2 y=3 res=16
```

```
public class ParameterClass {
    #region intPrameter
    public static void TestPara() {
        int x = 2;
        int y = 3;
        TestPara1(x, y);
        Console.WriteLine("Main");
        Console.WriteLine($"x={x} y={y}");
        int res = TestPara2(x, y);
        Console.WriteLine("Main");
        Console.WriteLine($"x={x} y={y} res={res}");
    }
    static void TestPara1(int y, int x) {
        y = 3;
        x = x+y;
        Console.WriteLine("Methode1");
        Console.WriteLine($"x={x} y={y}");
    }
    static int TestPara2(int a, int b) {
        a += 2;
        b += 1;
        Console.WriteLine("Methode2");
        Console.WriteLine($"a={a} b={b}");
        return a * b;
    }
}
#endregion
```

Parameterwünsche...

Es soll die Möglichkeit geben zu wählen ob ...

- Eingangsparameter
 - eine Funktion einen Wert erhält um ihn zu benutzen, weiterzuverwenden, beliebig zu verändern ohne, dass es den Originalwert beeinflusst
- Ausgangsparameter
 - eine Funktion einen Wert erhält den man in der Funktion „befüllt“ damit man mehrere Rückgabewerte haben kann
- Übergangsparameter
 - eine Funktion einen Wert erhält, den verändert und der Originalwert der außerhalb der Funktion existent ist wird mit aktualisiert.

Call By Value

- Parameterübergabe per Wert
- Kopie der Variable wird an die Methode übergeben

genannt:

- Eingangsparmeter

```
static void Method( int i)
{
    i = 44;
}
static void Main()
{
    int value=0;
    Method(value);
    // value is now 0
    Console.WriteLine(value);
}
```

Call By Referenz

- Parameterübergabe per Referenz mit **ref** / **out**
- Speicherplatz wo die Variable gespeichert wird, wird an die Methode übergeben

genannt:

- **ref** – Übergangsparemeter
- **out** - Ausgangsparemeter

```
static void Method( ref int i)
{
    i += 44;
}
static void Main()
{
    int value=22;
    Method( ref value);
    // value is now 66
    Console.WriteLine(value);
}
```

Arten von Parameter in C#

Call by

- Value mit Wertübergabe
- Referenz mit Adressübergabe
 - Ausgangsparameter:
 - Schlüsselwort: out
 - Übergangsparameter:
 - Schlüsselwort: ref
 - Unterschied zu out:
Variable **muss** vor dem
Methodenaufruf initialisiert sein

```
static void Method( out int i)
{
    i = 44;
}
static void Main()
{
    int value;
    Method( out value);
    // value is now 44
    Console.WriteLine(value);
}
```

Ausgangsparameter:

```
static void Method(out int i, out string s1, out string s2)
{
    i = 44;
    s1 = "I've been returned";
    s2 = null;
}
static void Main()
{
    int value;
    string str1, str2;
    Method(out value, out str1, out str2);
    Console.WriteLine("V: {0}, S1: {1}, S2: {2}",
        value, str1, str2);
}
```

Wie lautet die Ausgabe?

Arrays als Parameter

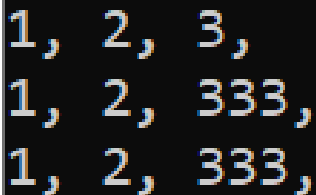
ref & out bei Arrays als Parameter?!

Arrays sind Referenzdatentypen

- Wertzuweisung:
 - die Adresse im Heap wird gespeichert

```
static void Main(string[] args) {  
    int[] arr = { 1, 2, 3 };  
    int[] brr = arr;  
    PrintArray(arr);  
    brr[2] = 333;  
    PrintArray(arr);  
    PrintArray(brr);  
}  
  
static void PrintArray(int[] arr) {  
    for (int i = 0; i < arr.Length; i++) {  
        Console.Write($"{arr[i]}, ");  
    }  
    Console.WriteLine();  
}
```

- Wie lautet die Ausgabe?



```
1, 2, 3,  
1, 2, 333,  
1, 2, 333,
```


Parameterübergabe mit Arrays

```
static void Main(string[] args) {
    TestArrPara();
}
```

- Wie lautet die Ausgabe

```
Main
2, 3, 4, 5,
Methode
2, 3, 333, 777,
Main
2, 3, 333, 777,
```

```
#region ArrayParamter
static void TestArrPara() {
    int[] arr = { 2, 3, 4, 5 };
    Console.WriteLine("Main");
    PrintArr(arr);
    TestArrPara1(arr);
    Console.WriteLine("Main");
    PrintArr(arr);
}
```

```
static void TestArrPara1(int[] arr) {
    arr[2] = 333;
    arr[3] = 777;
    Console.WriteLine("Methode");
    PrintArr(arr);
}
```

```
static void PrintArr(int[] arr) {
    for (int i = 0; i < arr.Length; i++) {
        Console.Write(arr[i] + ", ");
    }
    Console.WriteLine();
}
#endregion
```

Array als Parameter

- In der Main wird ein Array instantiiert, welches anschließend verworfen wird in der FillArray-Methode -> Sinnvoller out verwenden!
- Wie lautet die Ausgabe?
- Array elements are:
- 1 2 3 4 5

```
static void FillArray(int[] arr)
{
    // Initialize the array:
    arr = new int[5] { 1, 2, 3, 4, 5 };
}

static void Main()
{
    int[] theArray = new int[5]; // Initialization is not required

    // Pass the array to the callee using out:
    FillArray(theArray);

    // Display the array elements:
    System.Console.WriteLine("Array elements are:");
    for (int i = 0; i < theArray.Length; i++)
    {
        System.Console.Write(theArray[i] + " ");
    }
}
```

Eingangsparameter mit Array

- Array elements are:
- 00000

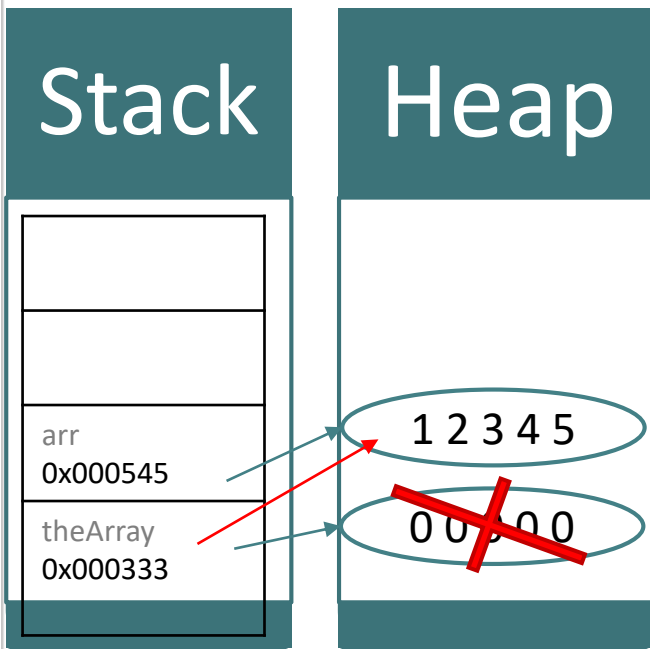
```
static void FillArray(int[] arr)
{
    // Initialize the array:
    arr = new int[5] { 1, 2, 3, 4, 5 };
}

static void Main()
{
    int[] theArray = new int[5]; // Initialization is not required

    // Pass the array to the callee using out:
    ➔ FillArray(theArray);

    // Display the array elements:
    System.Console.WriteLine("Array elements are:");
    for (int i = 0; i < theArray.Length; i++)
    {
        System.Console.Write(theArray[i] + " ");
    }

    // Keep the console window open in debug mode.
    System.Console.WriteLine("Press any key to exit.");
    System.Console.ReadKey();
}
```



Ausgangsparameter mit Array

- Mit Hilfe von out kann ein Array in der Funktion instantiiert werden!
- Wie lautet die Ausgabe?
- Array elements are:
- 1 2 3 4 5

```
static void FillArray(out int[] arr)
{
    // Initialize the array:
    arr = new int[5] { 1, 2, 3, 4, 5 };
}

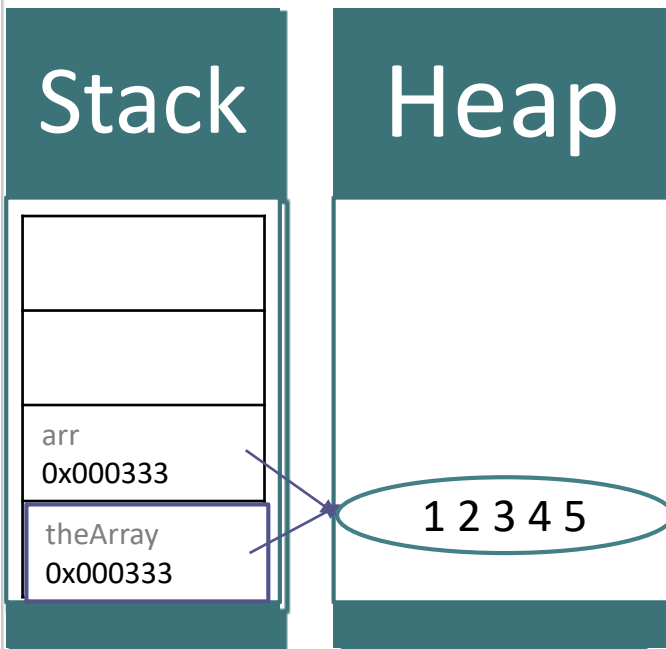
static void Main()
{
    int[] theArray; // Initialization is not required

    // Pass the array to the callee using out:
    FillArray(out theArray);

    // Display the array elements:
    System.Console.WriteLine("Array elements are:");
    for (int i = 0; i < theArray.Length; i++)
    {
        System.Console.Write(theArray[i] + " ");
    }
}
```

Array im Speicher – mit out

- Array elements are:
- 1 2 3 4 5



```
static void FillArray(out int[] arr)
{
    // Initialize the array:
    arr = new int[5] { 1, 2, 3, 4, 5 };
}

static void Main()
{
    int[] theArray; // Initialization is not required

    // Pass the array to the callee using out:
    FillArray(out theArray);

    // Display the array elements:
    System.Console.WriteLine("Array elements are:");
    for (int i = 0; i < theArray.Length; i++)
    {
        System.Console.Write(theArray[i] + " ");
    }

    // Keep the console window open in debug mode.
    System.Console.WriteLine("Press any key to exit.");
    System.Console.ReadKey();
}
```

Übergangsparmeter mit Arrays?

- Es macht keinen Sinn Arrays mit „ref“ zu übergeben, auch ohne ref ändern sich die Werte im Array, da es sich um einen Referenzdatentyp handelt.
- Wie lautet die Ausgabe?
- Array elements are:
- 1111 2 3 4 5555

```
static void FillArray(ref int[] arr)
{
    // Create the array on demand:
    if (arr == null)
    {
        arr = new int[10];
    }
    // Fill the array:
    arr[0] = 1111;
    arr[4] = 5555;
}

static void Main()
{
    // Initialize the array:
    int[] theArray = { 1, 2, 3, 4, 5 };

    // Pass the array using ref:
    FillArray(ref theArray);

    // Display the updated array:
    System.Console.WriteLine("Array elements are:");
    for (int i = 0; i < theArray.Length; i++)
    {
        System.Console.Write(theArray[i] + " ");
    }
}
```

Arrays als Parameter

- Ein Array kann ohne ref an eine Funktion übergeben werden, es gibt nur eine Version des Arrays am Heap, diese wird direkt verändert.
- Als Parameter wird eine Kopie der Adresse des Arrays übergeben.
- Wie lautet die Ausgabe?
- Array elements are:
10 20 30 111 555

```
static void FillArray(int[] arr)
{
    // Create the array on demand:
    if (arr == null) {
        arr = new int[10];
    }
    // Fill the array:
    arr[3] = 111;
    arr[4] = 555;
}

static void Main()
{
    // Initialize the array:
    int[] theArray = { 10, 20, 30, 40, 50 };

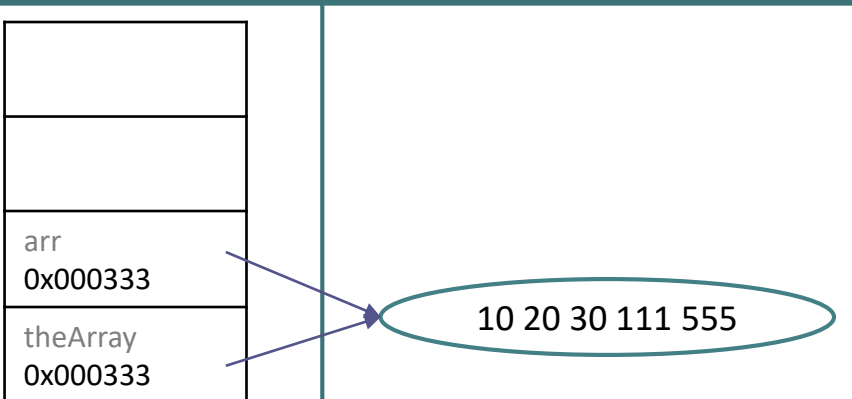
    // Pass the array using ref:
    FillArray(theArray);

    // Display the updated array:
    System.Console.WriteLine("Array elements are:");
    for (int i = 0; i < theArray.Length; i++)
    {
        System.Console.Write(theArray[i] + " ");
    }
}
```

Array als Parameter

- Array elements are:
- 10 20 30 111 555

Stack Heap



```
static void FillArray(int[] arr)
{
    // Create the array on demand:
    if (arr == null) {
        arr = new int[10];
    }
    // Fill the array:
    arr[3] = 111;
    arr[4] = 555;
}

static void Main()
{
    // Initialize the array:
    int[] theArray = { 10, 20, 30, 40, 50 };

    // Pass the array using ref:
    → FillArray(theArray);

    // Display the updated array:
    System.Console.WriteLine("Array elements are:");
    for (int i = 0; i < theArray.Length; i++)
    {
        System.Console.Write(theArray[i] + " ");
    }

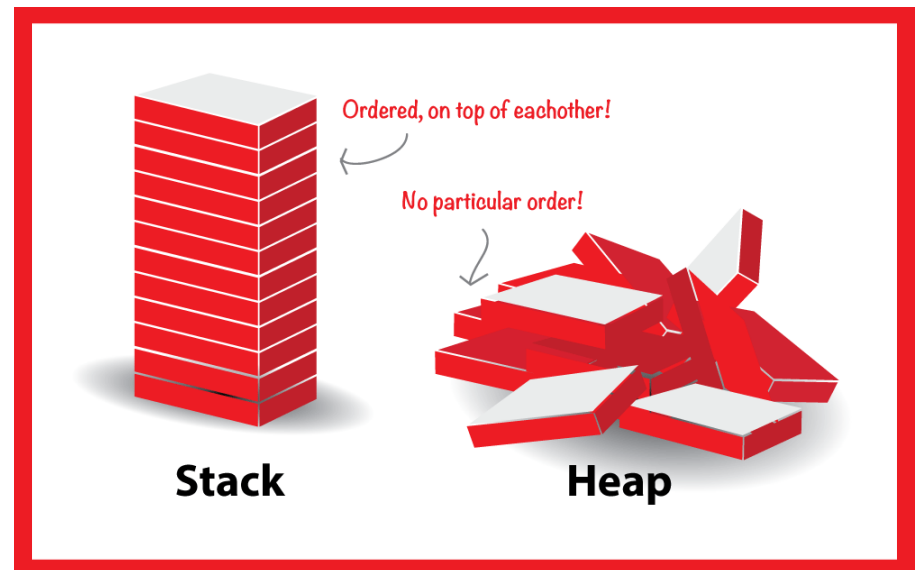
    // Keep the console window open in debug mode.
    System.Console.WriteLine("Press any key to exit.");
    System.Console.ReadKey();
}
```


Ref & Out bei Arrays

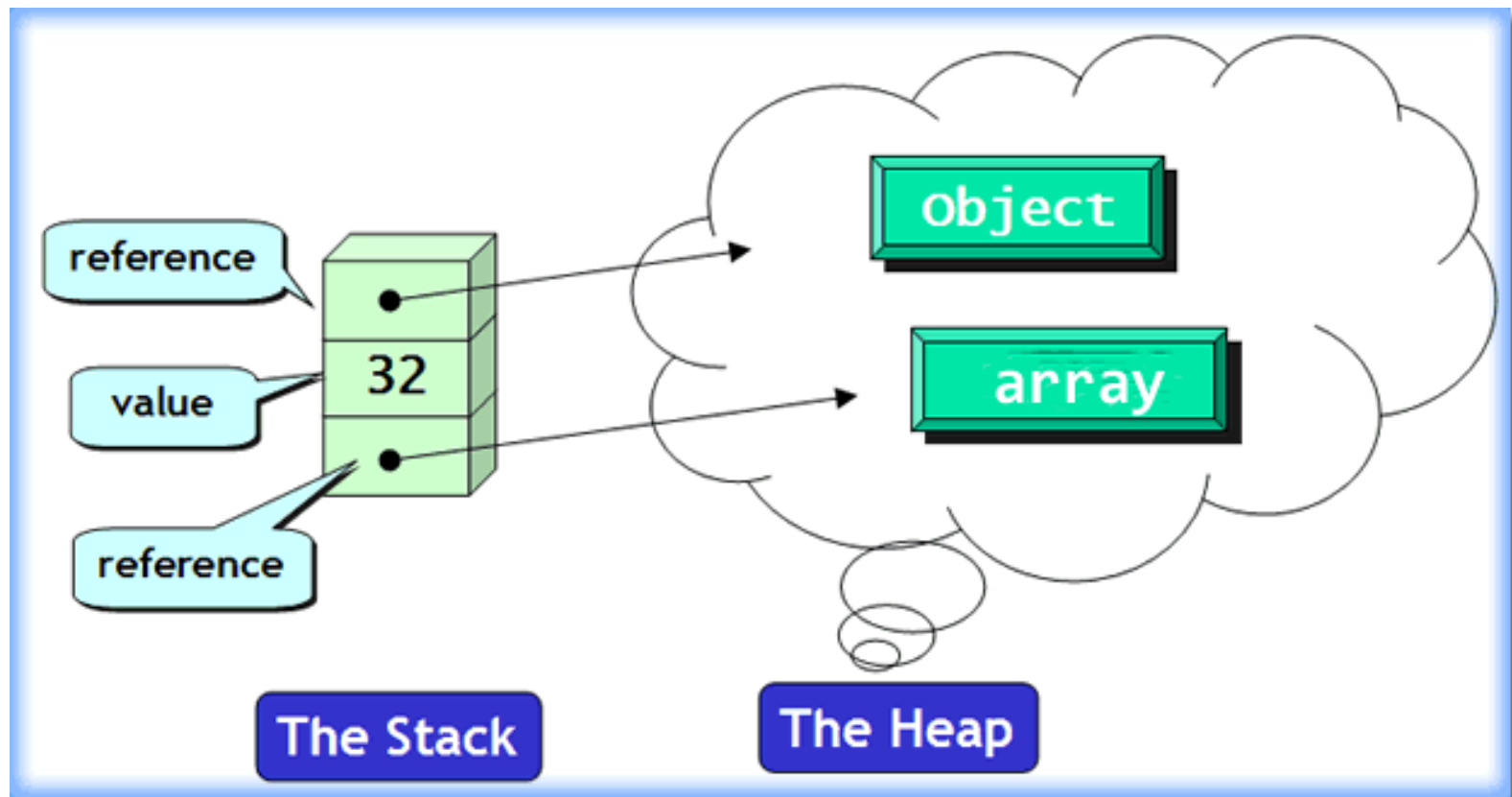
- Auch Arrays kann man mit ref und out übergeben, allerdings sind Arrays Referenzdatentypen.
- Es wird die „Adresse des Arrays im Heap“ übergeben
- Ref macht für die Übergabe von Arrays keinen Sinn, kann weggelassen werden
- Out hat den Vorteil, dass man ein Array, welches nur deklariert jedoch nicht instantiiert ist übergeben kann – ohne out würde ein Compilerfehler auftreten...

Speicherverwaltung

Stack & Heap

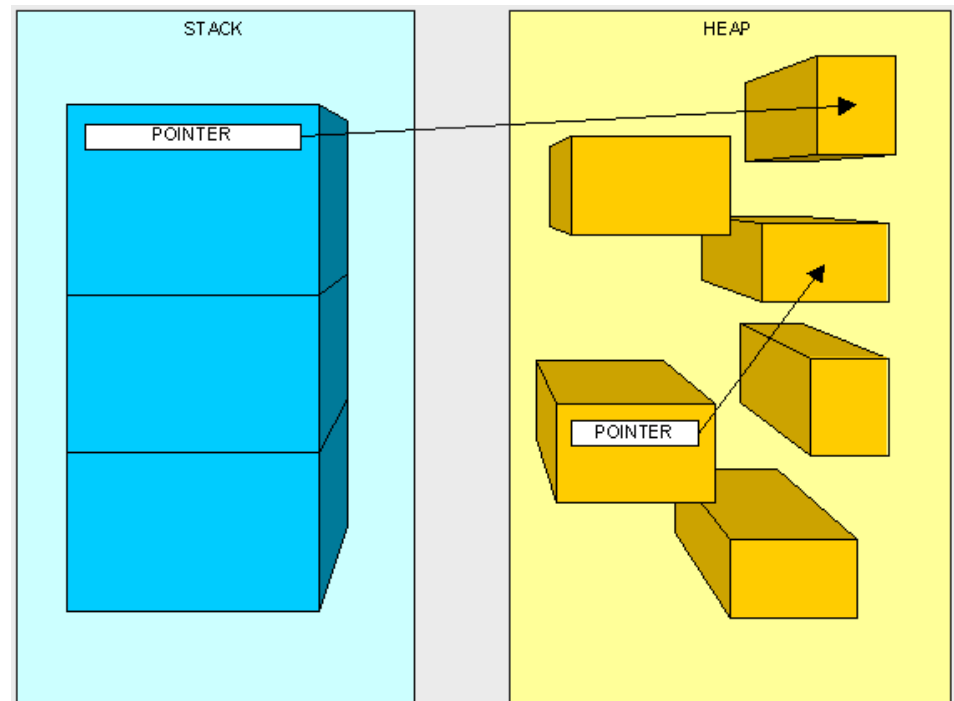


Stack vs Heap



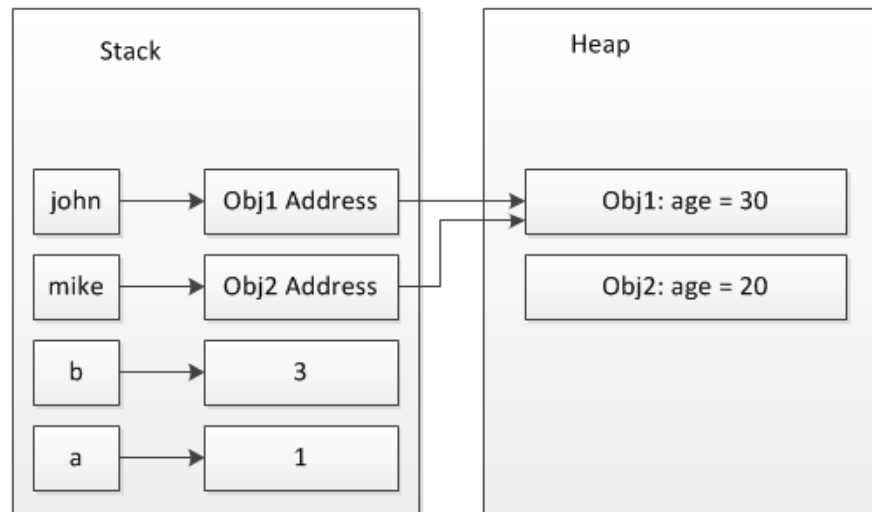
Wie wird entschieden, was landet wo?

- Zwei goldene Regeln:
 - **Referenztypen** landen immer am HEAP (einfache Regel)
 - **Wertetypen** und Referenzen landen im STACK
 - oder dort wo sie deklariert werden



Stack & Heap

- String, StringBuilder & Random sind Referenztypen und entsprechen „Objekten“
- Zahlen(int) in Arrays landen am Heap



Unterscheidung Referenz- und Wertetypen

Wertetypen

primitive

Datentypen:

- bool
- byte
- char
- decimal
- double
- enum
- float
- int
- long

- sbyte
- short
- struct
- uint
- ulong
- ushort

- Datentypen von
System.ValueType

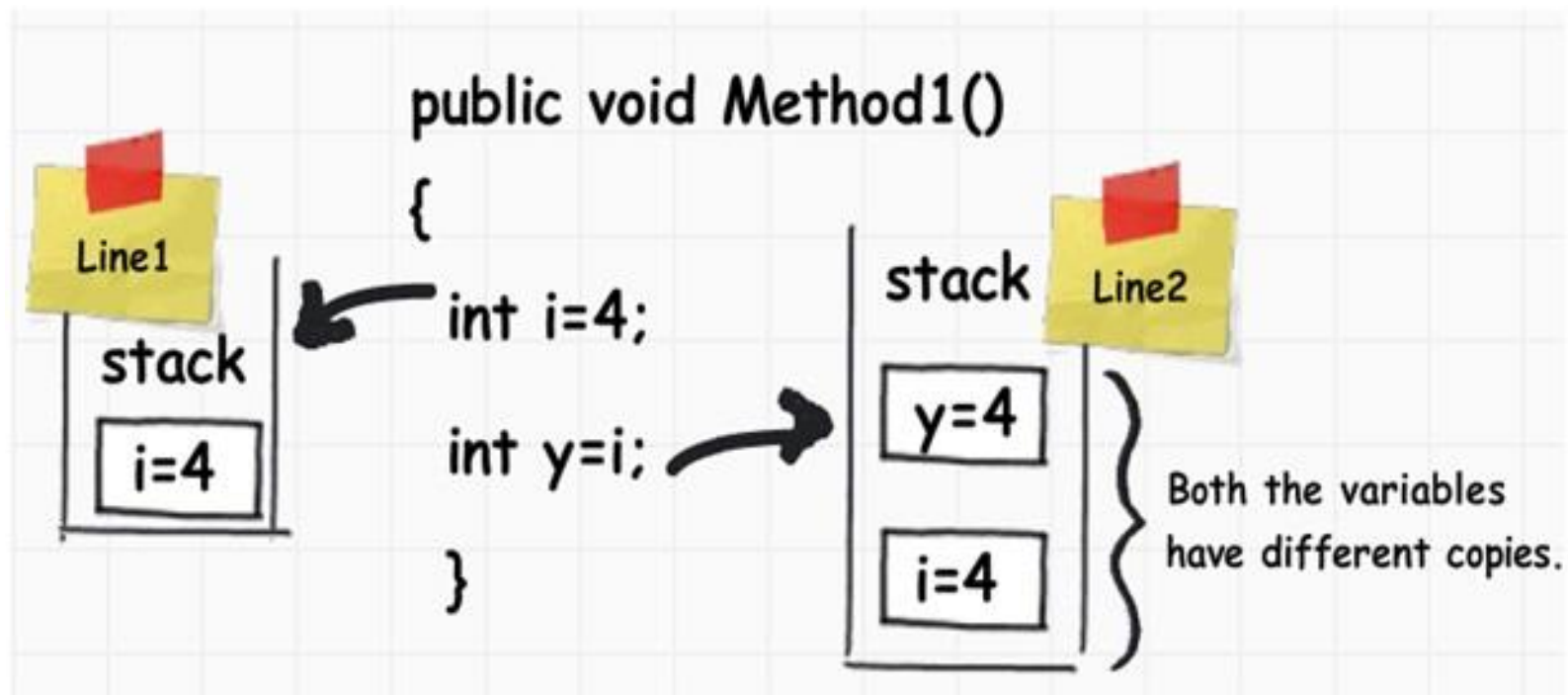
Referenztypen

zB:

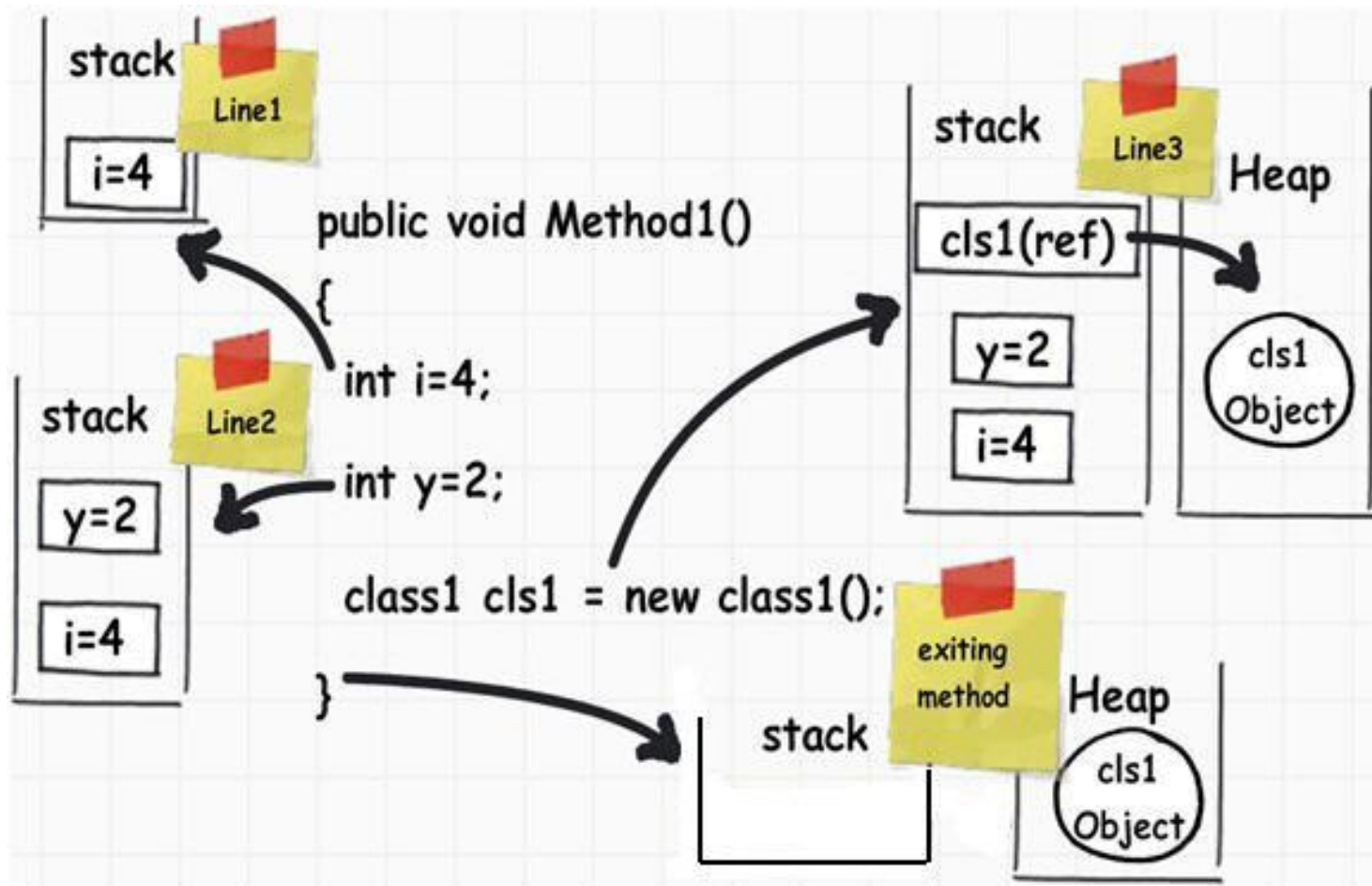
- String
 - Random
 - StringBuilder
 - Arrays
 - alle Objekte
- class
- interface
- delegate
- object
- string

Wertedatentypen

- Primitive Datentypen landen am Stack:

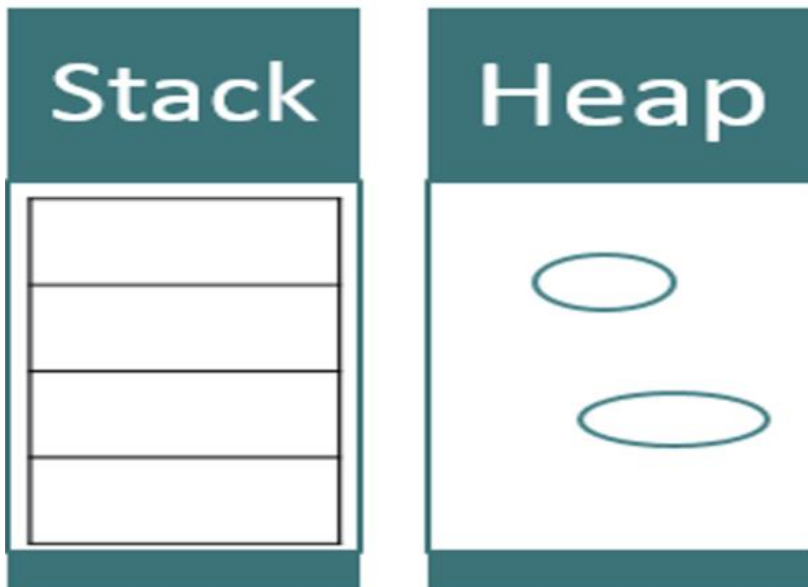


Wertetypen & Referenztypen



Arrays im Speicher

Zeichne Stack und Heap
nach dem Methodenaufruf
`FillArray(theArray);`



```
static void FillArray(int[] arr)
{
    // Create the array on demand:
    if (arr == null) {
        arr = new int[10];
    }
    // Fill the array:
    arr[3] = 111;
    arr[4] = 555;
}

static void Main()
{
    // Initialize the array:
    int[] theArray = { 10, 20, 30, 40, 50 };

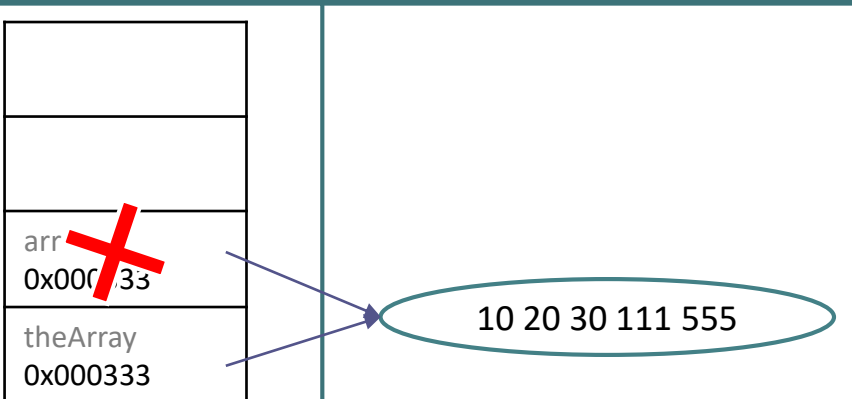
    // Pass the array using ref:
    FillArray(theArray);

    // Display the updated array:
    System.Console.WriteLine("Array elements are:");
    for (int i = 0; i < theArray.Length; i++)
    {
        System.Console.Write(theArray[i] + " ");
    }

    // Keep the console window open in debug mode.
    System.Console.WriteLine("Press any key to exit.");
    System.Console.ReadKey();
}
```

Arrays im Speicher

Stack Heap



```
static void FillArray(int[] arr)
{
    // Create the array on demand:
    if (arr == null) {
        arr = new int[10];
    }
    // Fill the array:
    arr[3] = 111;
    arr[4] = 555;
}

static void Main()
{
    // Initialize the array:
    int[] theArray = { 10, 20, 30, 40, 50 };

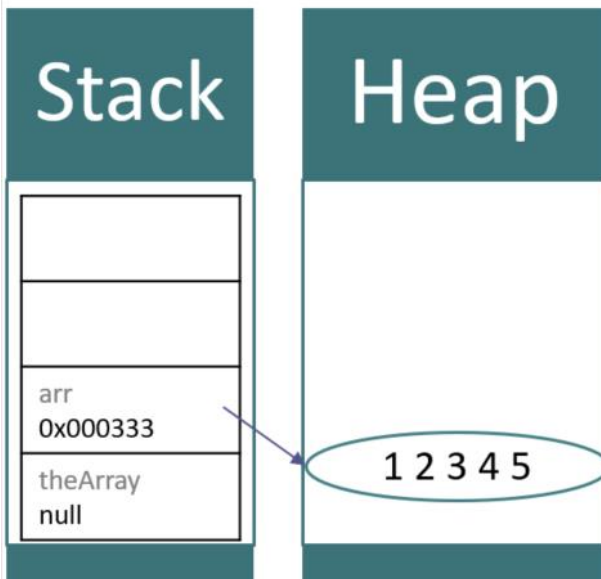
    // Pass the array using ref:
    FillArray(theArray);

    // Display the updated array:
    System.Console.WriteLine("Array elements are:");
    for (int i = 0; i < theArray.Length; i++)
    {
        System.Console.Write(theArray[i] + " ");
    }

    // Keep the console window open in debug mode.
    System.Console.WriteLine("Press any key to exit.");
    System.Console.ReadKey();
}
```

Array im Speicher – mit out

- Array elements are:
- 1 2 3 4 5



```

static void FillArray(out int[] arr)
{
    // Initialize the array:
    arr = new int[5] { 1, 2, 3, 4, 5 };
}

static void Main()
{
    int[] theArray; // Initialization is not required

    // Pass the array to the callee using out:
    FillArray(out theArray);

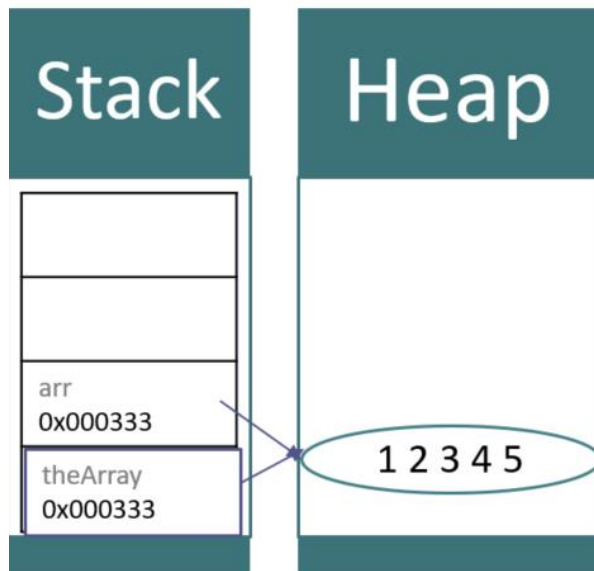
    // Display the array elements:
    System.Console.WriteLine("Array elements are:");
    for (int i = 0; i < theArray.Length; i++)
    {
        System.Console.Write(theArray[i] + " ");
    }

    // Keep the console window open in debug mode.
    System.Console.WriteLine("Press any key to exit.");
    System.Console.ReadKey();
}

```

Array im Speicher – mit out

- Array elements are:
- 1 2 3 4 5



```
static void FillArray(out int[] arr)
{
    // Initialize the array:
    arr = new int[5] { 1, 2, 3, 4, 5 };
}

static void Main()
{
    int[] theArray; // Initialization is not required

    // Pass the array to the callee using out:
    FillArray(out theArray);

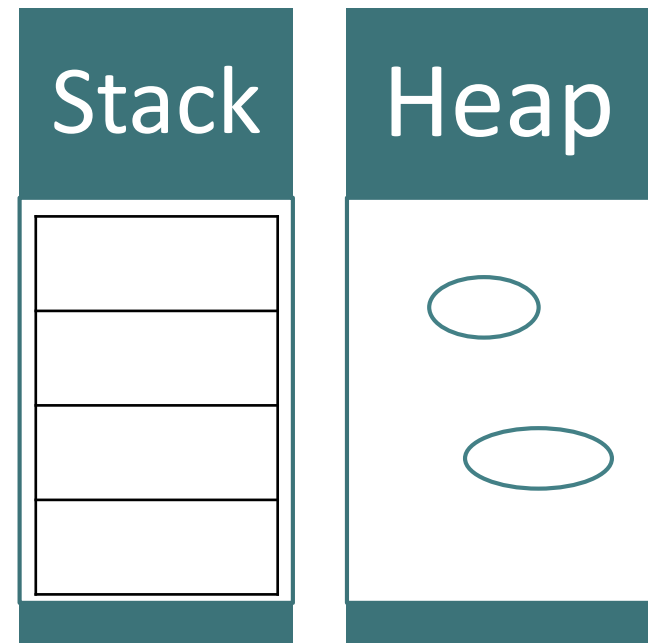
    // Display the array elements:
    System.Console.WriteLine("Array elements are:");
    for (int i = 0; i < theArray.Length; i++)
    {
        System.Console.Write(theArray[i] + " ");
    }

    // Keep the console window open in debug mode.
    System.Console.WriteLine("Press any key to exit.");
    System.Console.ReadKey();
}
```

Zusammenfassung

- Speicher ist in Stack und Heap geteilt:

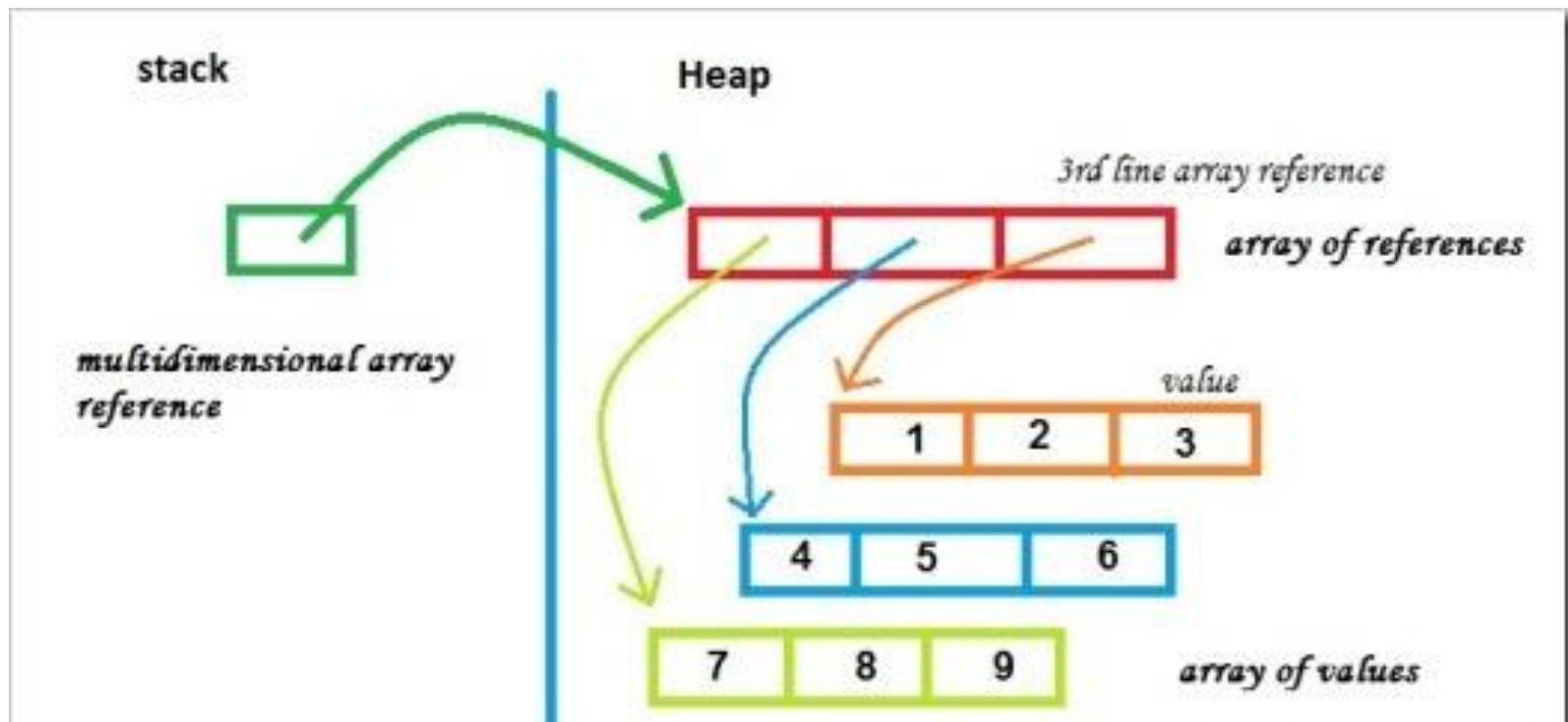
- Referenztypen landen immer am HEAP (einfache Regel)



- Wertetypen und Referenzen landen dort wo sie deklariert werden

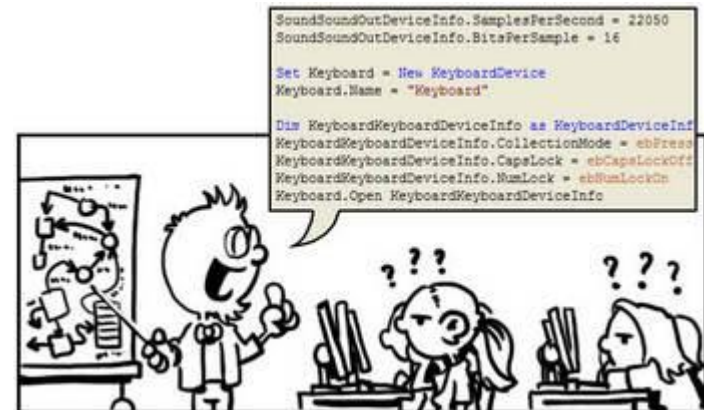
Mehrdimensionale Arrays

- im Speicher:



Wiederholungsfragen

Zu Funktionen



Nutzen von Funktionen

- Was ist eine Funktion?
 - Ein Bezeichner
 - für mehrere Anweisungen
 - der mehrere Anweisungen zusammenfasst
 - kann benutzt werden um diese Anweisungen auszuführen.
- Nutzen einer Funktion?
 - Übersichtlicher
 - Kann beliebig oft genutzt/aufgerufen werden

Eigenschaften von Funktionen

- Variablen einer Funktion können innerhalb der Funktion genutzt werden, und sind nur innerhalb dieser Funktion gültig
 - Lebensdauer einer Variable -.- > Funktionsvariable
- Lebensdauer einer Variable
 - Von der Speicherreservierung bis zum Ende des Gültigkeitsbereichs
 - Gültigkeitsbereich einer Variable von { bis }
 - `public static void` Funktion() { `int` x = 1; x++; CW(x); }
 - Es gibt Schleifenvariablen, Funktionsvariablen und Klassenvariablen

Parameter von Funktionen

- Wie können Variablenwerte zwischen Funktionen ausgetauscht werden?
 - Mit Hilfe von Parameter
- Wie bekommt man ein Ergebnis von der Funktion?
 - Mit Hilfe des Return-Werts
 - (Später) auch über Ausgangsparameter oder Übergangsparameter möglich

Nutzen von Parameter

- Aufgabe 1

- Funktion1 berechnet die Summe eines Arrays
- Funktion2 gibt alle Werte des Arrays in der Console aus
- Wie können beide Funktionen mit dem selben Array arbeiten?

- Aufgabe 2

- Funktion1 liest einen Wert vom Benutzer ein.
- Diese Funktion wird 2 mal aufgerufen
- Funktion2 addiert beide Werte und gibt die Summe in der Console aus.

Die einfache und suboptimale Lösung

- Globale Variablen ----> iiiigitttt ;-)
 - Innerhalb einer Klasse eine (statische, public) Variable deklarieren
 - Gültigkeitsbereich dieser Variable ist auf die gesamte Klasse ausgedehnt und kann von allen Funktionen verändert werden.
- **Alternative Lösung:**
- Für eine bessere Struktur und schönere Lösung -> Arbeiten mit Parametern (Werte die einer Funktion übergeben werden)

Syntax

- Syntax einer Funktion

Modifizierer `static` Rückgabetyt Bezeichner(Parameterliste){}

- Rückgabewert einer Funktion

- Datentyp der nach Aufruf der Funktion zurück gegeben wird

- Parameter einer Funktion

- Werte die an eine Funktion übergeben werden
- In weiterer Folge: Eingangs- Übergangs- oder Ausgangsparameter