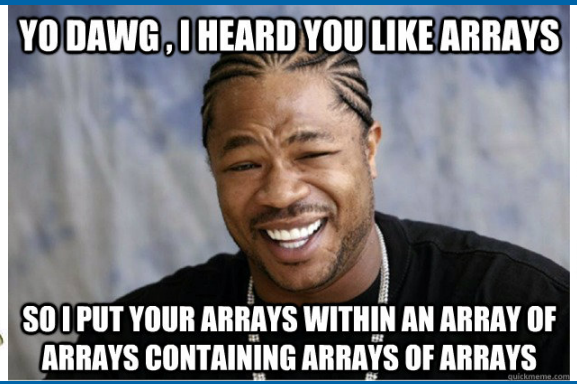
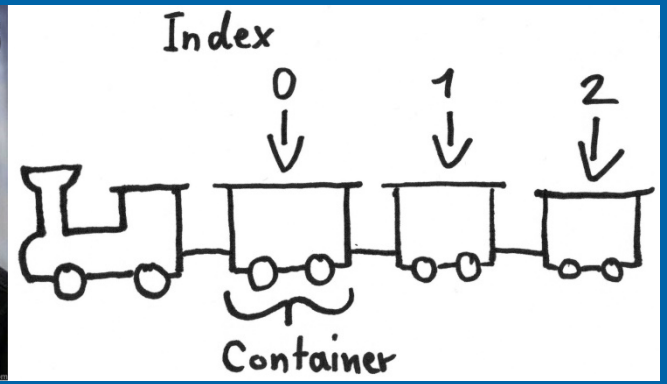
TECHNISCHE UNIVERSITÄT
BERGAKADEMIE FREIBERG
Die Ressourcenuniversität. Seit 1765.

Klassen II
Konstruktoren, static

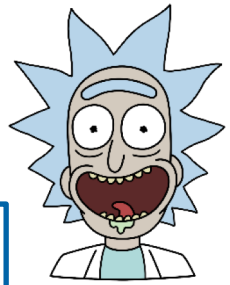




Konstruktoren
statische Klassen
Einschub: Arrays

Klassen sind grobe Pläne und Objekte sind die Entitäten davon. Darum müssen wir immer erst Objekte der Klassen erstellen.

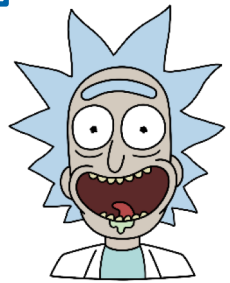
Absolut korrekt. So ist es



Was ist dann sowas wie

```
System.Console.WriteLine("Hallo Welt");
```

Hier geht es um statische Klassen, Methoden und Felder.
System ist jedoch ein **namespace**. Das dürfen wir nicht vergessen.



statische Felder

Die Felder und Methoden von Klassen können entweder

- in jedem Objekt oder
- nur einmal pro Klasse

angelegt werden.

static ist das Schlüsselwort für solche Member

```
class StaticExample
{
    public static void FunStuff() =>
        System.Console.WriteLine("Ich bin statisch");
    public void BoringStuff() =>
        System.Console.WriteLine("Ich bin nicht statisch");

    public static int CoolThing = 2;
    public int OtherCoolThing = 4;
}
```

```
class App
{
    static void Main()
    {
        StaticExample exampleObject = new StaticExample();
        // Treating as object:
        exampleObject.BoringStuff();
        exampleObject.OtherCoolThing = 1;
        // Treating as class:
        StaticExample.FunStuff();
        StaticExample.CoolThing = 3;
    }
}
```

Statische Klassen

- statische Klassen enthalten ausschließlich statische Member

- statische Klassen können nicht instanziiert werden

```
static class StaticClass
{
    public static void PrintStuff() { }
    public static int CalculateStuff() { return 0; }

    public static int CoolNumber = 5;
}
```

Spezielle Methoden in Klassen

Klassen besitzen einige Methoden mit speziellen Aufgaben:

- Konstruktoren (Initialisierungsmethoden)
- (Finalizer, auch: Destruktoren)
- *Properties
- …

Konstruktor

Bisher immer:

```
static void Main()
{
    Rectangle rect = new Rectangle();
}
```

- Konstruktoren: spezielle Methoden zur Initialisierung
- **Rectangle()** ist der Standard-Konstruktor

eigene Konstruktoren

```
class Rectangle
{
    int Width, Length;
    public int Area;

    public Rectangle(int inputWidth, int inputLength)
    {
        Width = inputWidth;
        Length = inputLength;
        CalculateArea();
    }

    public Rectangle()
    {
        Width = 2;
        Length = 3;
        CalculateArea();
    }

    void CalculateArea() => Area = Width * Length;
}
```

- Besitzen den Namen der Klasse selbst
- können beliebig **überladen** werden
- haben **keinen** Rückgabewert
- werden bei Initialisierung der Objekte aufgerufen.

Aufruf des Konstruktors

```
static void Main()
{
    Rectangle rect1 = new Rectangle(5,6);
    Rectangle rect2 = new Rectangle();
    Console.WriteLine("Fläche ist: " + rect1.Area.ToString());
    Console.WriteLine("Fläche ist: " + rect2.Area.ToString());
}
```

- Konstruktoren ermöglichen eine schnelle Initialisierung

- beim Anlegen des Objektes muss ein passender Konstruktor existieren

Schlüsselwort this

```
public Name(string vorname, string nachname)
{
    vorname = vorname;
    nachname = nachname;
}
```

Erzeugt nur eine Warnung

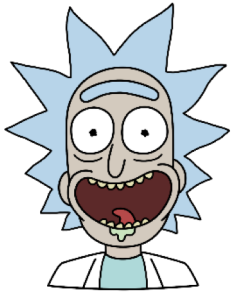
```
public Name(string vorname, string nachname)
{
    this.vorname = vorname;
    this.nachname = nachname;
}
```

ist wesentlich eindeutiger

- die Referenz **this** verweist auf die aktuelle Instanz der Klasse
- nützlich bei Konstruktoren

Wie kann man eigentlich in statischen Klassen Werte definieren?

Statische Klassen bzw. Klassen mit statischen Membern rufen zum Zeitpunkt der ersten Nutzung einen statischen Konstruktor auf.
Falls keiner existiert, wird der parameterlose Standard-Konstruktor benutzt, der alle Werte auf den Default-Wert setzt.



statische Konstruktoren

```
class Student
{
    string vorname, nachname;
    int Matrikelnummer;
    static int Matrikel;
    public static readonly string university;

    public Student(string vorname, string nachname)
    {
        this.vorname = vorname;
        this.nachname = nachname;
        this.Matrikelnummer = Matrikel;
        Matrikel++;
    }
```

```
    static Student()
    {
        university = "TU BA Freiberg";
        Matrikel = 50000;
    }

    public void PrintDetails()
    {
        System.Console.WriteLine(
            nachname + ", " +
            vorname + ", Matrikelnummer: " +
            Matrikelnummer.ToString());
    }
}
```

```
class App
{
    static void Main()
    {
        Console.WriteLine(Student.university);
        Student stud1 = new Student("Micky", "Mouse");
        Student stud2 = new Student("Donald", "Duck");
        stud1.PrintDetails();
        stud2.PrintDetails();
    }
}
```

```
>constructorExample03.exe
TU BA Freiberg
Mouse, Micky, Matrikelnummer: 50000
Duck, Donald, Matrikelnummer: 50001

>
```

Einschub: Arrays

Erinnerung:

- Arrays repräsentieren eine feste Anzahl an Elementen eines Typs
- Arrays werden in einem einzelnen Block Speicher gelegt, was schnellen und effizienten Zugriff ermöglicht

Deklaration und Definition

```
using System;

class App
{
    static void Main()
    {
        char[] vowels;
        vowels = new char[5];
        // einzelner Zugriff:
        vowels[0] = 'a';
        vowels[1] = 'e';
        vowels[2] = 'i';
        vowels[3] = 'o';
        vowels[4] = 'u';
        // Zugriff über Zählschleife:
        for(int i = 0; i < 5; i++)
            Console.Write(vowels[i]);
        Console.WriteLine();
    }
}
```

- Arrays sind generell Referenztypen
- Arrays liegen immer auf dem Heap
- Arrays werden grundlegend zur Laufzeit erst erstellt
- Arrays sind Objekte der Klasse System.Array
- Arrays unterstützen keine Pointer-Arithmetik

Standardwerte und Kurz-Definition

```
using System;

class App
{
    static void Main()
    {
        int[] intArray = new int[3]; // definiert mit 0
        char[] charArray = new char[5]; // definiert mit '\0'
        bool[] boolArray = new bool[5]; // definiert mit false
        string[] stringArray = new string[22]; // definiert mit null

        int intArray1 = new int[] { 0, 4, 22, 56 };
        int[] initializedArray = { 3, 8, 6, 99 };
    }
}
```

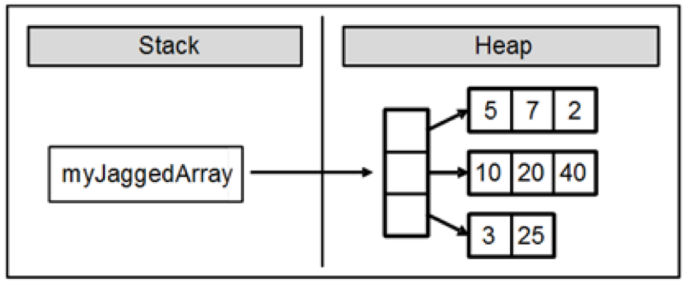
- Arrays haben Standardwerte

- Arrays belegen erst durch den **new** Operator Speicherplatz

mehrdimensionale Arrays

C# unterstützt zwei Arten von mehrdimensionalen Arrays:

- Jagged Arrays (ausgefranzte Arrays)



- Rectangular Arrays (rechteckige Arrays)

[0,0]	[0,1]	[0,2]	[0,3]	[0,4]
[1,0]	[1,1]	[1,2]	[1,3]	[1,4]
[2,0]	[2,1]	[2,2]	[2,3]	[2,4]

```
static void Main()
{
    int[][] jaggedArray = new int[2][];
    jaggedArray[0] = new int[2] { 0, 3 };
    jaggedArray[1] = new int[5];

    jaggedArray[1][0] = 4;

    int[,] rectArray1 = new int[2, 3];
    int[,] rectArray2 = { { 1, 2, 3 }, { 4, 5, 6 } };
    int[,,] boxArray1 = new int[2, 3, 4];

    rectArray1[1, 2] = 1;
    boxArray1[0, 0, 0] = 1;
}
```

- beide Formen haben Vorteile und Nachteile
- Bevorzugung je nach Anwendung

Arrays als Objekte

Arrays sind Objekte und haben damit einige tolle Eigenschaften:

- sie besitzen Methoden, welche Informationen liefern
- sie werfen **Ausnahmen (Exceptions)**, um Fehlermanagement zu ermöglichen
- sie können in Verbindung mit **foreach** verwendet werden

Array-Methoden

```
int[,] integerArray = new int[5,7];
Console.WriteLine("Length: " + integerArray.Length.ToString());
Console.WriteLine("rank: " + integerArray.Rank.ToString());
Console.WriteLine("Length in 0: " + integerArray.GetLength(0).ToString());
Console.WriteLine("Length in 1: " + integerArray.GetLength(1).ToString());
```

```
int[] smallArray = new int[4] { 5, 1, 99, -4 };
Array.Sort(smallArray);
Array.Reverse(smallArray);

// Ausgabe:
for (int i = 0; i < smallArray.Length; i++)
    Console.Write(smallArray[i].ToString() + " ");
Console.WriteLine();

// noch besser:
foreach (int element in smallArray)
    Console.Write(element.ToString() + " ");
Console.WriteLine();
```

```
Length: 35
rank: 2
Length in 0: 5
Length in 1: 7
99 5 1 -4
99 5 1 -4
```

Arrays of Objects

In C# können Arrays auch komplexere Daten enthalten:

```
public double Length =1;
public double Width = 2;

public void printArea()
    => Console.WriteLine(
        "Area: " + (Length * Width).ToString()
        );
```

```
Rectangle[] rectArray = new Rectangle[3];
for (int i = 0; i < rectArray.Length; i++)
    rectArray[i] = new Rectangle();
foreach (Rectangle actualOne in rectArray)
    actualOne.printArea();
```

```
Area: 2
Area: 2
Area: 2
```

Typische Array-Exception

```
int[] myArray = new int[] { 1,2,3,4,5};
try
{
    for (int i = 0; i < 15; i++)
        Console.WriteLine(myArray[i].ToString());
}
catch(IndexOutOfRangeException e)
{
    Console.WriteLine(e.Message);
}
```

```
1
2
3
4
5
Der Index war außerhalb des Arraybereichs.
```

Übergabe von Arrays an Methoden

```
static void PrintArray(string[] arr)
{
    foreach (string element in arr)
        Console.Write(element + " ");
    Console.WriteLine();
}


static void Main()
{
    string[] theArray = { "Hallo", "I", "bims", "1", "array" };
    PrintArray(theArray);
}
```

```
>arraysExample07.exe
Hallo I bims 1 array

>
```

Oder als Rückgabewert

```
int[] GetArray()
{
    int[] newArray =  { 1, 2, 3, 4};
    return newArray;
}
```

```
int[] myArray = GetArray();
```

```
static void ChangeArray1(int[] input)
{
    input[0] = 0;
}

static void ChangeArray2(ref int[] input)
{
    input[0] = 0;
}

static void Main()
{
    int[] theArray = { 1, 2, 3 };
    ChangeArray1(theArray);
    Console.WriteLine(theArray[0]);
    ChangeArray2(ref theArray);
    Console.WriteLine(theArray[0]);
}
```

Echte Änderungen werden immer übernommen, egal ob mit oder ohne ref gearbeitet wird.

```
>arraysExample08.exe
0
0

>
```

Aber!

```
static void ChangeArray1(int[] input)
    {
        input = new int[] { 5, 6, 12, 25};
    }

    static void ChangeArray2(ref int[] input)
    {
        input = new int[] { 5, 6, 12, 25 };
    }

    static void Main()
    {
        int[] theArray = { 1, 2, 3 };
        ChangeArray1(theArray);
        Console.WriteLine(theArray[0]);
        ChangeArray2(ref theArray);
        Console.WriteLine(theArray[0]);
    }
```

```
>arraysExample08-2.exe
1
5

>
```

Nutzung von Kommandozeilenargumenten

```
using System;

class App
{
    static void Main(string[] args)
    {
        foreach(string argument in args)
            Console.WriteLine(argument);
    }
}
```

weitere Konvertierungen dann mittels **Parse** Methoden

```
>arraysExample09.exe

>arraysExample09.exe Hallo Welt
Hallo
Welt

>
```

Zusammenfassung

- statische Klassen
- Konstruktoren, Standard-Konstruktor, Überladung von Konstruktoren
- Arrays: Definition, Deklaration, Übergabe, mehrdimensionale Arrays, Nutzen statischer Methoden der Array-Klasse