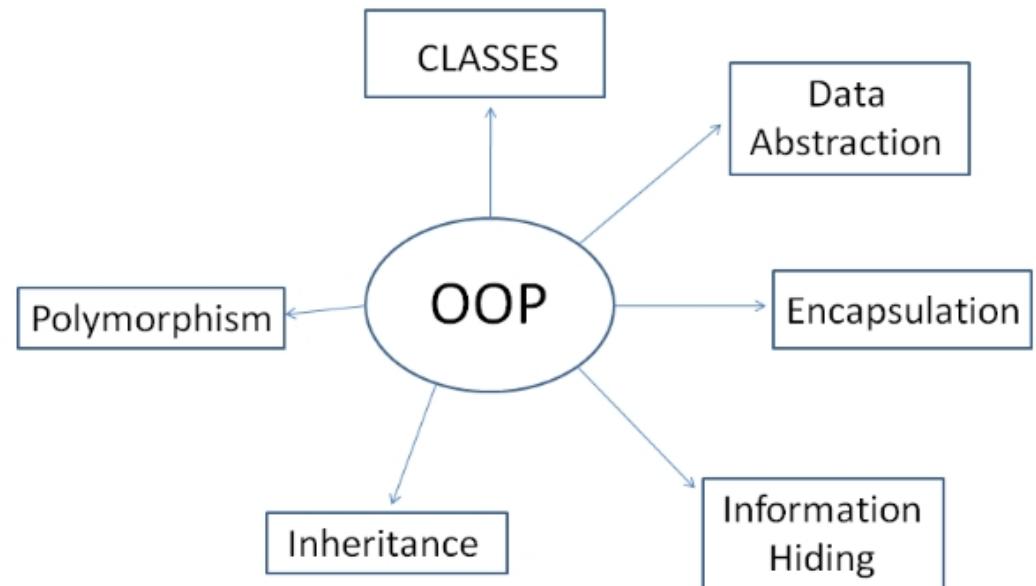


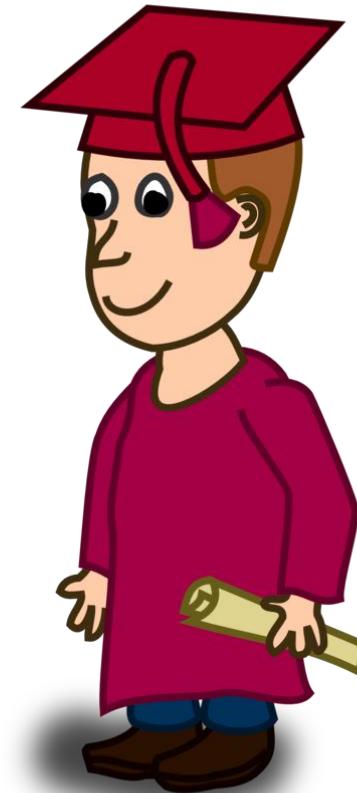
OOP Fragen und Aufgabenstellungen

Softwareentwicklung SEW2



Übersicht

- Klassen & Instanzen
- Arrays von selbst geschriebenen Klassen
- Sichtbarkeiten
- Statische vs dynamische Methoden
- Exkurs in Kommandozeilenparameter
- Konstruktoren - Konstruktorenüberladung
- Vererbung & ToString
- Get Set Methoden
- Konstruktorkette
- Properties
- Object Initializer
- Abstrakte Klassen
- Polymorphie



Klasse – Instanz/Object

Ersten Klassen erstellen





```
class FirstClass {  
    public string name;  
    public void MyMethod() {  
        Console.WriteLine($"Hallo {name}");  
    }  
}
```

FirstClass

- Eine Klasse in der selben Datei erzeugen und nutzen

```
class Program {  
  
    static void Main(string[] args) {  
        FirstClass fc = new FirstClass();  
        fc.name = "Kurt";  
        fc.MyMethod();  
    }  
}
```



Neue Klasse in neuer Datei

The screenshot shows the context menu of a project in the Project Explorer of Visual Studio. The 'Klasse...' option is selected, which has opened a 'Neues Element hinzufügen' (Add New Item) dialog.

Project Explorer Context Menu:

- Erstellen
 - Neu erstellen
 - Bereinigen
 - Analysieren und Code bereinigen
 - Paket
- Veröffentlichen...
 - Ansicht auf dieses Element beschränken
- Neue Projektmappen-Explorer-Ansicht
- Auf Code Map anzeigen
- Projektdatei bearbeiten

Add New Item Dialog:

The dialog lists various item types under the 'Visual C#-Elemente' category. A new class named 'SecondClass' is being added.

Kategorie	Element	Sub-Kategorie
Visual C#-Elemente	Klasse	Visual C#-Elemente
	Schnittstelle	Visual C#-Elemente
	Formular (Windows Forms)	Visual C#-Elemente
	Benutzersteuerelement (Windows Forms)	Visual C#-Elemente
	Komponentenklasse	Visual C#-Elemente
	Benutzersteuerelement (WPF)	Visual C#-Elemente
	Anwendungskonfigurationsdatei	Visual C#-Elemente
	Anwendungsmanifestdatei	Visual C#-Elemente
	Assemblyinformationsdatei	Visual C#-Elemente
	Benutzerdefiniertes Steuerelement (Windows Fo...)	Visual C#-Elemente
Online	Bitmapdatei	Visual C#-Elemente
	Codedatei	Visual C#-Elemente
	Cursordatei	Visual C#-Elemente

Dialog Fields:

- Name: SecondClass
- Hinzufügen (Add)
- Abbrechen (Cancel)



Second Class

```
class SecondClass {
    public string name;
    public int number;

    public void MyMethod(string x) {
        Console.WriteLine($"Method {x} - {name} + {number}");
    }
}
```



Second Class Instanziieren

```
class SecondClass {  
    public string name;  
    public int number;  
  
    public void MyMethod(string x) {  
        Console.WriteLine($"Method {x}" +  
            $" - {name} + {number}");  
    }  
}
```

```
class Program {  
    static void TestSecondClass() {  
  
        SecondClass sc = ...  
        sc.number = 15;  
        sc.name = "Kurt";  
        sc.MyMethod("Beliebige Zeichenkette");  
  
        SecondClass sc2 = ...  
        sc2.name = "Max";  
        sc2.number = 25;  
        sc2.MyMethod("x");  
    }  
  
    static void Main(string[] args) {  
        FirstClass fc = ...  
        fc.name = "Kurt";  
        fc.MyMethod();  
  
        //Eigene statische Methode  
        //zum Testen der Objekte  
        TestSecondClass();  
    }  
}
```



Klasse Person

- Mit den Attributen: Name + Alter
 - Methode: Greet
-
- Erstelle eine Instanz der Klasse Person
 - Setze Werte für Name und Alter
 - Rufe die Greet-Methode auf



Klasse Person

```
class Person {  
    //Attribut Name  
    public string name;  
  
    //Attribut Alter  
    public int age;  
  
    //Methode Begrüßen  
    public void Greet() {  
        Console.WriteLine($"Begrüße {name} - {age}");  
    }  
}
```



Nutze die Klasse Person

```
class Program {  
    static void Main(string[] args) {  
  
        //Deklarieren ein Person  
        Person p;  
        //Instanz == Objekt einer Klasse  
        p = new Person();  
        //Initialisiere die Person  
        p.name = "Susanne";  
        p.age = 25;  
        //Methodenaufruf eines Objekts  
        p.Greet();  
    }  
}
```



Klasse Student

- Klasse Student: Name, Alter, Klasse, Lieblingsgegenstand
- Methode: Vorstellen (Name, Alter, Klasse)
- Methode: Lernen: Lernt Lieblingsgegenstand
- **Teil 1:** Erstelle 5 Objekte der Klasse Student und Initialisiere diese mit den Werten deiner Mitschüler. Speichere diese jeweils in einer Referenz der Klasse Student.
- **Teil 2:** Erstelle ein Array vom Typ Student und speichere 5 deiner Mitschüler. Gib diese in einer Schleife aus.
- **Teil 3:** Erstelle eine List<Student> und speichere 5 deiner Mitschüler. Gib diese in einer Schleife aus.
-

Student

```
class Student {  
    public string name;  
    public int age;  
    public string schoolclass;  
    public string favorite;  
    public void PresentMyself() {  
        Console.WriteLine($"{name} {age} {schoolclass}");  
    }  
    public void Lerning() {  
        Console.WriteLine($"{name} learns {favorite}");  
    }  
}
```

```
static void Main(string[] args) {  
    //Deklarieren ein Student  
    Student student;  
    //Instanz == Objekt einer Klasse  
    student = new Student();  
    //Initialisiere die Person  
    student.name = "Susanne";  
    student.age = 15;  
    student.schoolclass = "2AHIT";  
    student.favorite = "SEW";  
    //Methodenaufruf eines Objekts  
    student.PresentMyself();  
    student.Lerning();  
}
```





Mehrere Instanzen einer Klasse

```
public static void TestStudents1() {
    Student s1 = new Student();
    s1.name = "Susi";
    s1.age = 15;
    s1.schoolclass = "2B HIT";
    s1.favorite = "SEW";

    Student s2 = new Student();
    s2.name = "Maxl";
    s2.age = 15;
    s2.schoolclass = "2B HIT";
    s2.favorite = "SEW";

    Student s3 = new Student();
    s3.name = "Kurt";
    s3.age = 15;
    s3.schoolclass = "2B HIT";
    s3.favorite = "SEW";

    s1.PresentMyself();
    s2.PresentMyself();
    s3.PresentMyself();
}
```

```
class Student {
    public string name;
    public int age;
    public string schoolclass;
    public string favorite;
    public void PresentMyself() {
        Console.WriteLine($"{name} {age} {schoolclass}");
    }
    public void Lerning() {
        Console.WriteLine($"{name} learns {favorite}");
    }
}
```

```
Susi 15 2B HIT
Maxl 15 2B HIT
Kurt 15 2B HIT
```



Objekte in Array speichern

```
public static void TestStudents2() {
    Student[] students = new Student[3];
    students[0] = new Student();
    students[0].name = "Susi";
    students[0].age = 15;
    students[0].schoolclass = "2B HIT";
    students[0].favorite = "SEW";

    students[1] = new Student();
    students[1].name = "Maxl";
    students[1].age = 15;
    students[1].schoolclass = "2B HIT";
    students[1].favorite = "SEW";

    students[2] = new Student();
    students[2].name = "Kurt";
    students[2].age = 15;
    students[2].schoolclass = "2B HIT";
    students[2].favorite = "SEW";

    foreach (Student s in students)
        s.PresentMyself();
}
```

```
class Student {
    public string name;
    public int age;
    public string schoolclass;
    public string favorite;
    public void PresentMyself() {
        Console.WriteLine($"{name} {age} {schoolclass}");
    }
    public void Lerning() {
        Console.WriteLine($"{name} learns {favorite}");
    }
}
```

```
Susi 15 2B HIT
Maxl 15 2B HIT
Kurt 15 2B HIT
```

```
using System;
using System.Collections.Generic;
```

Objekte in List<Student> speichern



```
public static void TestStudents3() {
    List<Student> students = new List<Student>();
    Student s1 = new Student();
    s1.name = "Susi";
    s1.age = 15;
    s1.schoolclass = "2B HIT";
    s1.favorite = "SEW";
    students.Add(s1);

    Student s2 = new Student();
    s2.name = "Maxl";
    s2.age = 15;
    s2.schoolclass = "2B HIT";
    s2.favorite = "SEW";
    students.Add(s2);

    Student s3 = new Student();
    s3.name = "Kurt";
    s3.age = 15;
    s3.schoolclass = "2B HIT";
    s3.favorite = "SEW";
    students.Add(s3);

    foreach (Student s in students)
        s.PresentMyself();
}
```

```
class Student {
    public string name;
    public int age;
    public string schoolclass;
    public string favorite;
    public void PresentMyself() {
        Console.WriteLine($"{name} {age} {schoolclass}");
    }
    public void Lerning() {
        Console.WriteLine($"{name} learns {favorite}");
    }
}
```

```
Susi 15 2B HIT
Maxl 15 2B HIT
Kurt 15 2B HIT
```



Sichtbarkeiten: public vs private

- Klasse Student und sichtbare Methoden und Attribute:

```
class Student {  
    public string name;  
    public int age;  
    public string schoolclass;  
    public string favorite;  
  
    public void PresentMyself() {  
        Console.WriteLine($"{name} {age} {schoolclass}");  
    }  
    public void Lerning() {  
        Console.WriteLine($"{name} learns {favorite}");  
    }  
}
```



Attribute private

Private Attribute sind nicht verfügbar im instantiierten Objekt

```
class Student {
    string name;
    int age;
    string schoolclass;
    string favorite;
```

```
public void PresentMyself() {
    Console.WriteLine($"{name} {age} {schoolclass}");
}
public void Lerning() {
    Console.WriteLine($"{name} learns {favorite}");
}
```

Methoden public

```
static void Main(string[] args) {
    Student s = new Student();
    s.|
```

als " einzufügen.

}

- Equals
- GetHashCode
- GetType
- Lerning
- PresentMyself
- ToString

Student erbt von Klasse Object implizit



Private explizit hinschreiben

- Es bleibt nur Learning als **public** Methode sichtbar

```
class Student {
    private string name;
    private int age;
    private string schoolclass;
    private string favorite;
```

```
private void PresentMyself() {
    Console.WriteLine($"{name} {age} {schoolclass}");
}
public void Lerning() {
    Console.WriteLine($"{name} learns {favorite}");
}
```

```
class Program {
    void TestSecondClass() ...
}

static void Main(string[] args) {
    Student s = new Student();
    s.
}

object. t "Equals" einzufügen.
    }
```

The screenshot shows a code editor with a tooltip for the `s` variable. The tooltip lists several methods from the `Object` class: `Equals`, `GetHashCode`, `GetType`, `Lerning` (which is highlighted in yellow), and `ToString`. The `Lerning` method is described as being added by the user.

Private Attribute und private Methoden sind nicht verfügbar im instanziierten Objekt



Object ist die Basisklasse

- Object ist die Basisklasse aller Referenzdatentypen:
- Object beinhaltet 4 Methoden:
 - Equals
 - GetHashCode
 - GetType
 - ToString

```
object o = new object();  
o.  
  
    □ ★ ToString  
    □ ★ GetType  
    □ ★ GetHashCode  
    □ Equals  
    □ GetHashCode  
    □ GetType  
    □ ToString
```

string? object.ToString()
Returns a string that represents the current object.
★ IntelliCode-Vorschlag basierend auf diesem Kontext



Statische Methoden über Main

- **Private** und **Public** statische Methoden innerhalb der Klasse Program sind in der Main verfügbar

```
class Program {  
    static void TestStudent() ...  
    static void Main(string[] args) {  
  
        Student s = new Student();  
        s.Lerning();  
        s.PresentMyself();  
  
        Student.TestStudents1();  
  
        Program.TestStudent();  
    }  
}
```

The screenshot shows an IDE interface with Java code. A tooltip is displayed over the call to `Program.TestStudent();`. The tooltip contains four entries: `Equals`, `Main`, `ReferenceEquals`, and `TestStudent`. The `TestStudent` entry is highlighted with a yellow background, indicating it is the intended method call.



Statische Methoden in Klassen

- Public Methoden der Klasse sind nach dem Klassennamen in der Main verfügbar

```
class Student {
    public string name;
    public int age;
    public string schoolclass;
    public string favorite;

    public void PresentMyself() {
        Console.WriteLine($"{name} {age} {schoolclass}");
    }
    public void Lerning() {
        Console.WriteLine($"{name} learns {favorite}");
    }

    public static void TestStudents1() ...
    public static void TestStudents2() ...
    public static void TestStudents3() ...
}
```

```
static void Main(string[] args) {
    Student s = new Student();
    s.Lerning();
    s.PresentMyself();

    Student. ...
    idered equal.
    chnitt "Equals" einzufügen.
}

    Equals
    ReferenceEquals
    TestStudents1
    TestStudents2
    TestStudents3
}
```



Private Statische Methoden

- sind in der Main **nicht** sichtbar
hinter dem Klassennamen:

```
class Student {
    public string name;
    public int age;
    public string schoolclass;
    public string favorite;

    public void PresentMyself() {
        Console.WriteLine($"{name} {age} {schoolclass}");
    }
    public void Lerning() {
        Console.WriteLine($"{name} learns {favorite}");
    }

    public static void TestStudents1() ...
    private static void TestStudents2() ...
    public static void TestStudents3() ...
}
```

```
static void Main(string[] args) {
    Student s = new Student();
    s.Lerning();
    s.PresentMyself();

    Student.TestStudents1();
    Student.~
}

    Equals
    ReferenceEquals
    TestStudents1
    TestStudents3
void Student.TestStude
```



Eingabeaufforderung
App

-
- Öffnen
 - Als Administrator ausführen
 - Dateispeicherort öffnen
 - An "Start" anheften
 - An Taskleiste anheften

Konsolenparameter

Starten von Programmen in der Commandline

Einlesen von Werten aus der Commandline

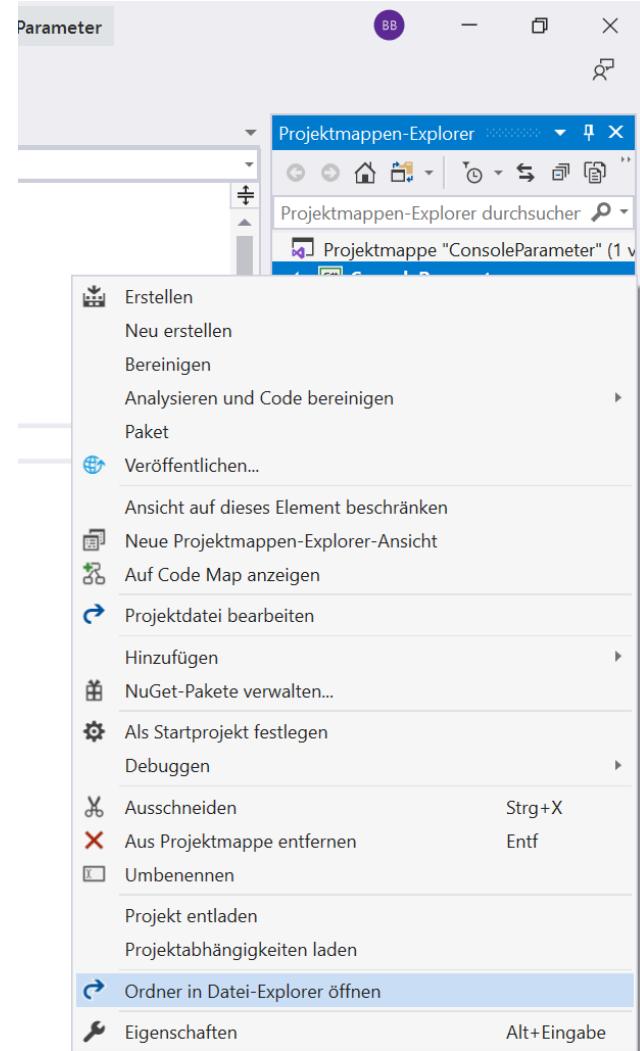
Einlesen von Werten in Visual Studio

In der Main über String[] args darauf zugreifen

Pfad finden für EXE

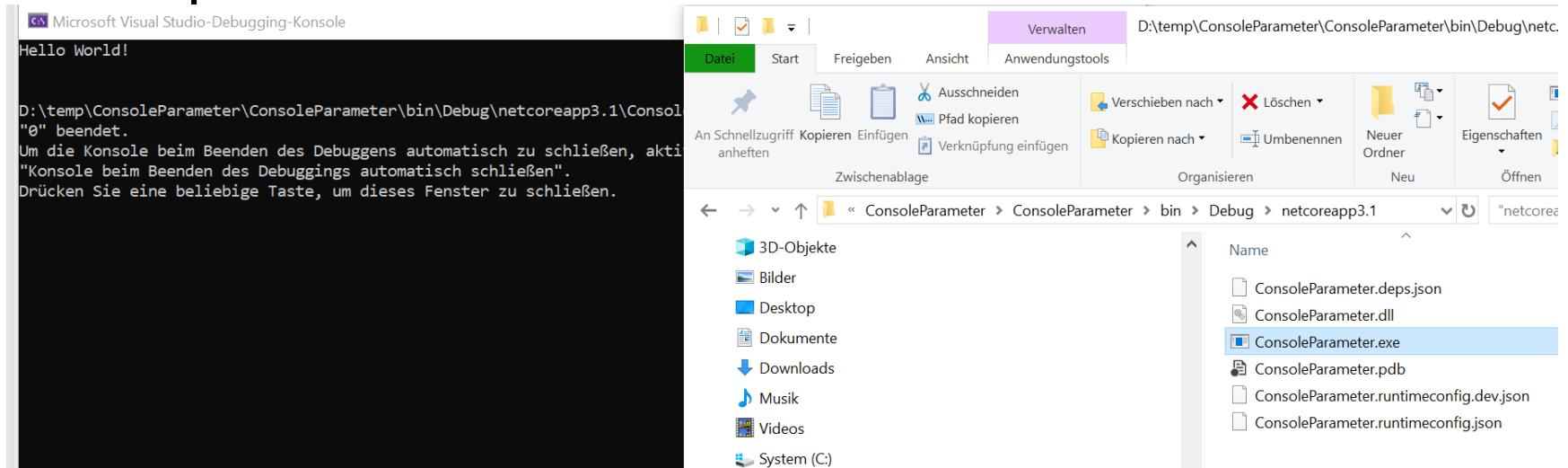
- Rechtsklick auf Projektnamen
- Ordner im Datei Explorer Öffnen

```
namespace ConsoleParameter {
    class Program {
        static void Main(string[] args) {
            Console.WriteLine("Hello World!");
            Console.ReadLine();
        }
    }
}
```



Compilieren

- Zuerst das Projekt ausführen, damit wird es kompiliert und die EXE erzeugt.
- Ausführen der Exe mit Doppelklick
- Fenster schließt falls kein Console.Read() implementiert wurde.



Eingabeaufforderung CMD

- Win-Taste + CMD
- Eingabeaufforderung starten
- Verzeichnis wechseln mit
- D: Enter
- cd Pfad eingeben Enter



Eingabeaufforderung
App

- ⇨ Öffnen
- ⇨ Als Administrator ausführen
- ⇨ Dateispeicherort öffnen
- ⇨ An "Start" anheften
- ⇨ An Taskleiste anheften

Ausführen des Programms

```
Eingabeaufforderung - ConsoleParameter.exe
Microsoft Windows [Version 10.0.18362.1082]
(c) 2019 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\dubdi>d:
D:\>cd D:\temp\ConsoleParameter\ConsoleParameter\bin\Debug\netcoreapp3.1
D:\temp\ConsoleParameter\ConsoleParameter\bin\Debug\netcoreapp3.1>dir
Datenträger in Laufwerk D: ist Data
Volumeseriennummer: A06F-E7CE

Verzeichnis von D:\temp\ConsoleParameter\ConsoleParameter\bin\Debug\netcoreapp3.1

22.09.2020 07:59    <DIR>        .
22.09.2020 07:59    <DIR>        ..
22.09.2020 08:15            440 ConsoleParameter.deps.json
22.09.2020 08:14            4 608 ConsoleParameter.dll
22.09.2020 08:14            174 592 ConsoleParameter.exe
22.09.2020 08:14            540 ConsoleParameter.pdb
22.09.2020 07:59            236 ConsoleParameter.runtimeconfig.dev.json
22.09.2020 07:59            154 ConsoleParameter.runtimeconfig.json
                           6 Datei(en),      180 570 Bytes
                           2 Verzeichnis(se), 89 448 517 632 Bytes frei

D:\temp\ConsoleParameter\ConsoleParameter\bin\Debug\netcoreapp3.1>ConsoleParameter.exe
Hello World!
```

Laufwerksbuchst
aben wählen mit
zB: **D:**

Verzeichnis
wechseln mit **cd**
Verzeichnis

Auflistung vom
Inhalt mit **dir**

Ausführen des
Programms mit
Programmname.
exe

Tab zum
Durchsteppen der
Dateinamen im
Verzeichnis

Zugriff auf Parameter vom User

```
D:\temp\ConsoleParameter\ConsoleParameter\bin\Debug\netcoreapp3.1  
>ConsoleParameter.exe Hallo das ist ein Text zum Testen
```

Hello World!

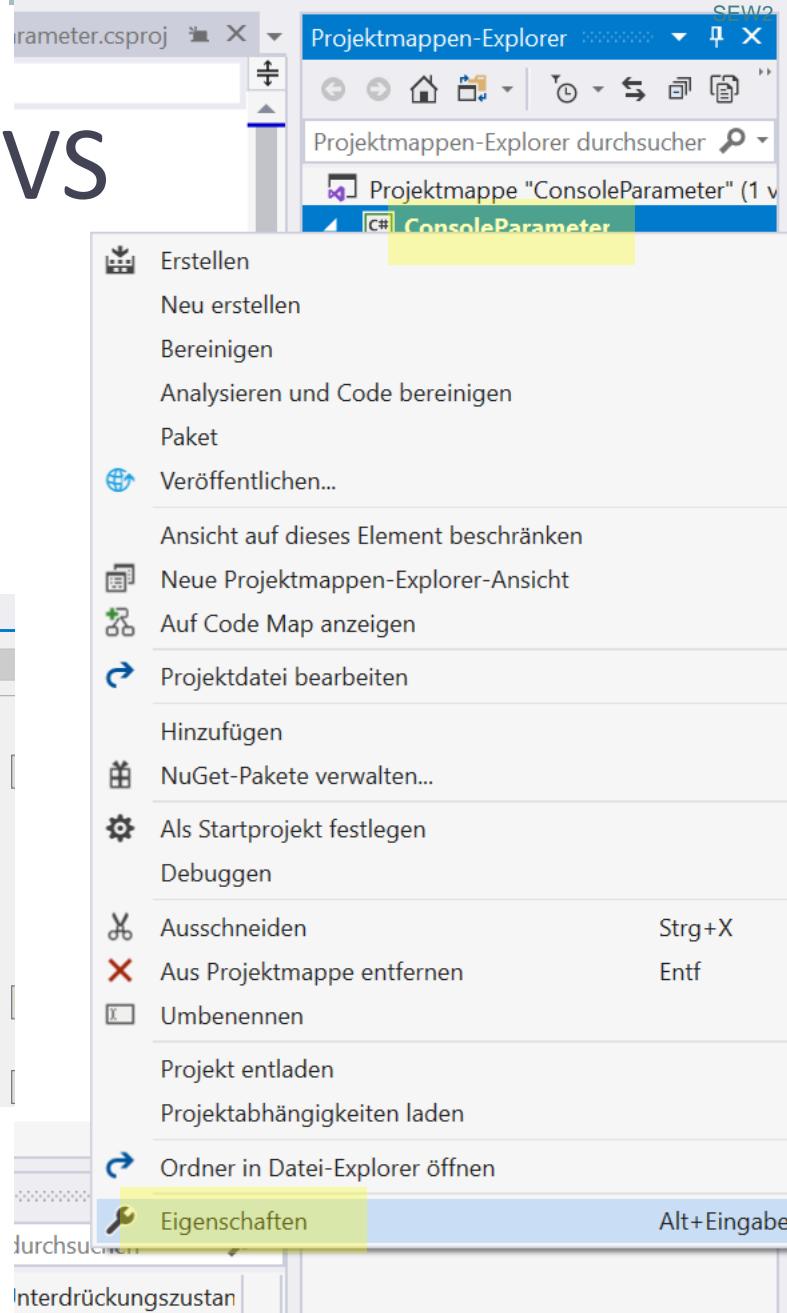
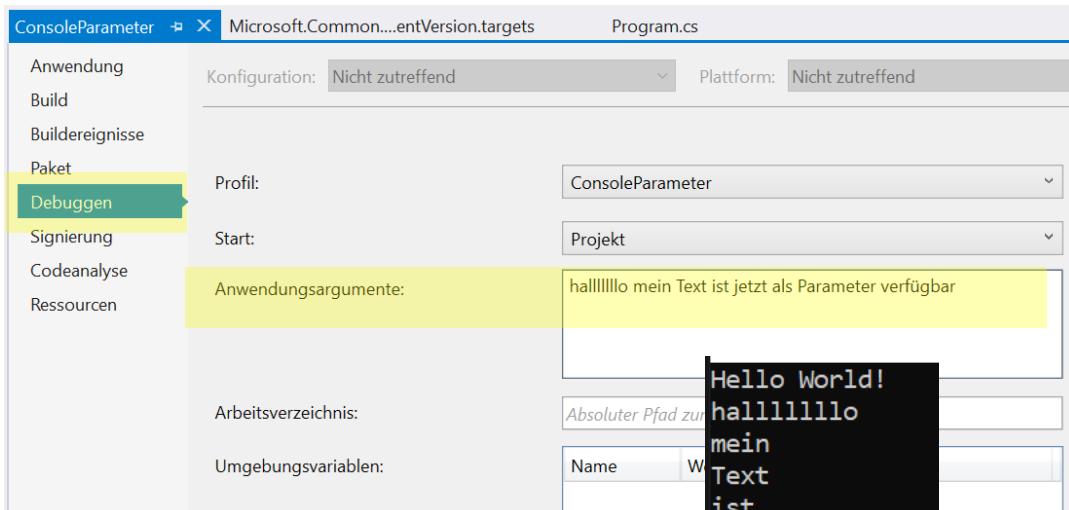
Hallo
das
ist
ein
Text
zum
Testen

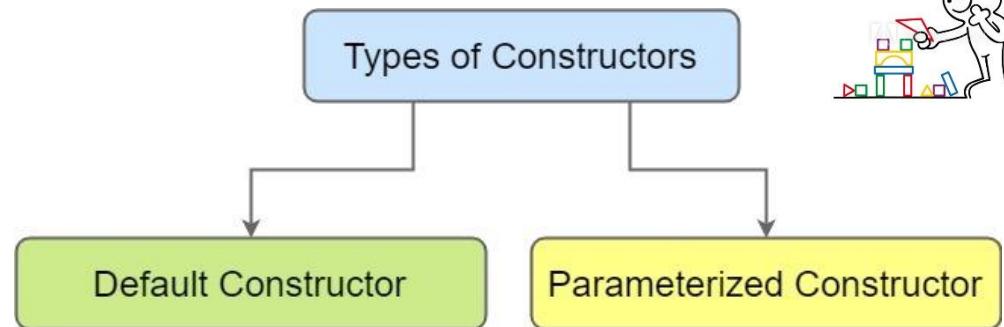
```
class Program {  
    static void Main(string[] args) {  
        Console.WriteLine("Hello World!");  
        foreach (var item in args) {  
            Console.WriteLine(item);  
        }  
        Console.ReadLine();  
    }  
}
```

- Durchlaufe das args-Array und erhalte die Parameter die beim Aufruf des Programms übergeben werden:

Konsolenparameter im VS

- Rechtsklick auf Projektmappe
- Eigenschaften / Debuggen
- Konsolenparameter eingeben





Konstruktoren

Wenn eine **Klasse** erstellt wird, wird deren Konstruktor aufgerufen, dabei können Standardwerte festlegt werden.

Konstruktoren haben den gleichen Namen wie die Klasse und sie initialisieren normalerweise die Datenmember (Attribute) des neuen Objekts.

Konstruktor mit Parameter



```
class Student {  
    public string name;  
    public int age;  
    public string schoolclass;  
    public string favorite;  
  
    //Konstruktor  
    public Student(string name) {  
        //Attribut der Klasse = Eingangsparameter  
        this.name = name;  
    }  
}
```

Gibt es einen selbst geschriebenen Konstruktor, verschwindet der parameterlose Standardkonstruktor, dieser muss wenn gewünscht selbst wieder hinzugefügt werden.

```
Student s1 = new Student();  
Student s2 = new Student("Susi");
```

Konstruktor mit Parameter



```
class Student {  
    public string name;  
    public int age;  
    public string schoolclass;  
    public string favorite;  
  
    //Konstruktor (Standardkonstruktor)  
    public Student() {  
        this.name = "Kurt";  
    }  
  
    //Konstruktor mit Parameter  
    public Student(string name) {  
        //Attribut der Klasse = Eingangsparameter  
        this.name = name;  
    }  
  
    public static void TestStudentKonstruktor1() {  
        Student s1 = new Student();  
        Console.WriteLine(s1.name);  
        Student s2 = new Student("Susi");  
        Console.WriteLine(s2.name);  
    }  
}
```

```
        static void Main(string[] args) {  
            Student.TestStudentKonstruktor();  
        }
```

Kurt
Susi



Konstruktor mit mehreren Parametern

```
class Student {
    public string name;
    public int age;
    public string schoolclass;
    public string favorite;

    //Konstruktor (Standardkonstruktor)
    public Student() { }

    //Konstruktor mit Parameter
    public Student(string name) {
        //Attribut der Klasse = Eingangsparameter
        this.name = name;
    }
}
```

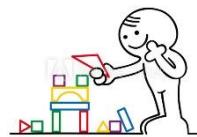
```
public static void TestStudentKonstruktor() {
    Student s1 = new Student();
    Student s2 = new Student("Susi");
    Console.WriteLine(s2.name);
    Student s3 = new Student("Susi", 14, "1B HIT", "SEW");
    s3.PresentMyself();

    static void Main(string[] args) {
        Student.TestStudentKonstruktor();
    }
}
```

Susi
Susi 14 1B HIT

```
public Student (string name, int age, string schoolclass, string favorite) {
    this.name = name;
    this.age = age;
    this.schoolclass = schoolclass;
    this.favorite = favorite;
}
```

Studentenarray



```
class Student {
    public string name;
    public int age;
    public string schoolclass;
    public string favorite;

    //Konstruktor (Standardkonstruktor)
    public Student() { }

    //Konstruktor mit Parameter
    public Student(string name) {
        //Attribut der Klasse = Eingangsparameter
        this.name = name;
    }
    //Konstruktor mit allen Attributen
    //als parameter zum Initialisieren
    public Student (string name, int age,
                   string schoolclass, string favorite) {
        this.name = name;
        this.age = age;
        this.schoolclass = schoolclass;
        this.favorite = favorite;
    }
}
```

```
public static void TestStudentKonstruktor2() {
    Student s1 = new Student("Susi", 14, "1B HIT", "SEW");
    Student s2 = new Student("Karl", 15, "1B HIT", "SEW");
    Student s3 = new Student("Franz", 17, "3B HIT", "SEW");
    Student s4 = new Student("Sepp", 12, "1B HIT", "SEW");
    Student[] students = new Student[8];
    students[0] = s1;
    students[1] = s2;
    students[2] = s3;
    students[3] = s4;
    students[4] = new Student("Kurt", 16, "3B HIT", "SEW");
    students[5] = new Student("Max", 18, "4B HIT", "SEW");
    students[6] = new Student("Edeltraud", 14, "1B HIT", "SEW");
    students[7] = new Student("Hildegard", 14, "1B HIT", "SEW");

    foreach (var item in students) {
        item.PresentMyself();
        item.Lerning();
    }

    static void Main(string[] args) {
        Student.TestStudentKonstruktor1();
        Student.TestStudentKonstruktor2();
    }
}
```

```
Susi
Susi 14 1B HIT
Susi 14 1B HIT
Susi learns SEW
Karl 15 1B HIT
Karl learns SEW
Franz 17 3B HIT
Franz learns SEW
Sepp 12 1B HIT
Sepp learns SEW
Kurt 16 3B HIT
Kurt learns SEW
Max 18 4B HIT
Max learns SEW
Edeltraud 14 1B HIT
Edeltraud learns SEW
Hildegard 14 1B HIT
Hildegard learns SEW
```

Konstruktor

Initialisierung mit Schleife

```
class Student {
    public string name;
    public int age;
    public string schoolclass;
    public string favorite;

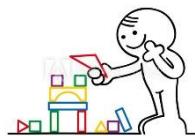
    //Konstruktor (Standardkonstruktor)
    public Student() { }

    //Konstruktor mit Parameter
    public Student(string name) {
        //Attribut der Klasse = Eingangsparameter
        this.name = name;
    }
    //Konstruktor mit allen Attributen
    //als Parameter zum Initialisieren
    public Student (string name, int age,
        string schoolclass, string favorite) {
        this.name = name;
        this.age = age;
        this.schoolclass = schoolclass;
        this.favorite = favorite;
    }
}
```

```
public static void TestStudentKonstruktor3(int size) {
    Student[] students = new Student[size];
    for (int i = 0; i < size; i++) {
        students[i] = new Student("Person" + i,
            10 + i, "2B HIT", "SEW");
    }
    foreach (var item in students) {
        item.PresentMyself();
    }
}
```

Person0	10	2B HIT
Person1	11	2B HIT
Person2	12	2B HIT
Person3	13	2B HIT
Person4	14	2B HIT
Person5	15	2B HIT
Person6	16	2B HIT
Person7	17	2B HIT
Person8	18	2B HIT
Person9	19	2B HIT

```
static void Main(string[] args) {
    Student.TestStudentKonstruktor1();
    Student.TestStudentKonstruktor2();
    Student.TestStudentKonstruktor3(10);
}
```





Klassen & Vererbung

Eine Klasse kann eine bestehende Klassen erweitern

Die Eigenschaften und Funktionalitäten der Basisklasse werden übernommen und in der abgeleiteten Klasse werden zusätzliche Eigenschaften und Funktionalitäten ergänzt.

Es herrscht eine >>ist eine<< Beziehung.

Person & Student

- Doppelter Quellcode
- Name & Alter

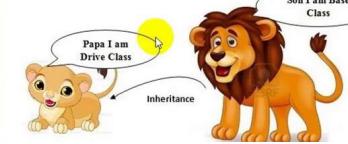
```
class Person {
    //Attribut Name
    public string name;

    //Attribut Alter
    public int age;

    //Methode Begrüßen
    public void Greet() {
        Console.WriteLine($"Begrüße {name} - {age}");
    }
}
```

```
class Student {
    public string name;
    public int age;
    public string schoolclass;
    public string favorite;

    public void PresentMyself() {
        Console.WriteLine($"{name} {age} {schoolclass}");
    }
    public void Lerning() {
        Console.WriteLine($"{name} learns {favorite}");
    }
}
```



Ein Student ist eine Person, hat damit alle Eigenschaften die eine Person hat.



Vererbung: Student erbt von Person

```

class Person {
    public string name;
    protected int age;
    private bool isMale;
}

class Program {
    static void Main(string[] args) {
        Person p = new Person();
        p.name = "Kurt";
        Student s = new Student();
        s.name = "Max";

        Console.WriteLine(p.name);
        Console.WriteLine(s.name);
    }
}

```

Alle Eigenschaften von Person sind auch im Studenten verfügbar



Vererbung: Student erbt von Person

```
class Person {
    public string name;
    protected int age;
    private bool isMale;
}
```

```
static void Main(string[] args) {
    Person p = new Person();
    p.name = "Susi";
    p.
    St new Student();
    s._Equals
    s. GetHashCode
    s. GetType
    s. name (Feld) string Person.name
    s. ToString
    s. 
```

```
class Student : Person {
}
```

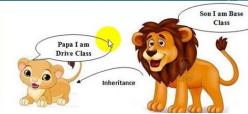
public Eigenschaften von Person
sind in der Main nach der
Instanziierung abrufbar.

```
static void Main(string[] args) {
    Person p = ...
    p.name = "Susi";

    Student s = new Student();
    s.name = "Martina";
    s.

    } 
```

Equals
GetHashCode
GetType
name (Feld) string Person.name
ToString



Person erweitern & Student erbt von Person

```
class Person {
    public string name;
    protected int age;
    private bool ...;
```

```
public Person() {
    //this verweist auf Attribute
    //innerhalb der Klasse
    this.name = "Hille";
    this.age = 16;
    this.isMale = false;
}
```

```
public void ThatsMe() {
    string gender = (isMale) ? "Mann" : "Frau";
    Console.WriteLine($"{gender} - {age} - {name}");
}
```

```
class Student : Person {
}
```

```
class Program {
    static void Main(string[] args) {
        Person p = new Person();
        p.name = "Susi";
        Student s = new Student();
        s.name = "Martina";

        Console.WriteLine(p.name);
        Console.WriteLine(s.name);
        p.ThatsMe();
        s.ThatsMe();
    }
}
```

public Methoden von Person sind in der Main nach der Instanziierung aufrufbar.

Student erweitern

```
class Student : Person {
    public string schoolclass;
    public string favorite;

    public Student() {
        schoolclass = "2CHIT";
        favorite = "SEW";
    }

    public void PresentStudent() {

        //public Attribut name ist verfügbar
        //protected Attribut age ist verfügbar
        //private Attribut isMale ist NICHT verfügbar
        Console.WriteLine($"{name} - {age} " +
            $"- {schoolclass} - {favorite}");
        //public Methode ThatsMe ist verfügbar
        ThatsMe();
    }
}
```

Susi
Martina
Frau - 16 - Susi
Martina - 16 - 2CHIT - SEW
Frau - 16 - Martina

```
class Person {
    public string name;
    protected int age;
    private bool isMale;

    public Person() {
        //this verweist auf Attribute
        //innerhalb der Klasse
        this.name = "Hille";
        this.age = 16;
        this.isMale = false;
    }

    public void ThatsMe() {
        string gender = (isMale) ? "Mann" : "Frau";
        Console.WriteLine($"{gender} - {age} - {name} ");
    }
}
```

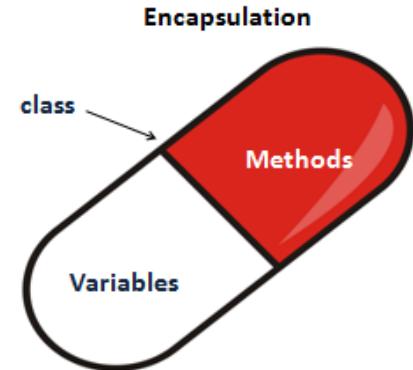
```
static void Main(string[] args) {
    Person p = new Person();
    p.name = "Susi";
    Student s = new Student();
    s.name = "Martina";
    Console.WriteLine(p.name);
    Console.WriteLine(s.name);
    p.ThatsMe();
    //PresentStudent ruft ThatsMe auf
    s.PresentStudent();
    s.
}
```

favorite
GetHashCode
GetType
name
PresentStudent
schoolclass
ThatsMe
ToString

void Student.PresentStudent()

Age und IsMale sind in der Main nicht verfügbar



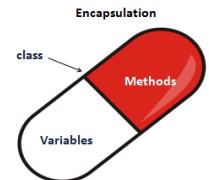


Get- Set-Methoden

Eine Möglichkeit Datenkapselung zu realisieren,
wird in C# selten verwendet – C# bietet Properties als Alternative

Dient als wohldefinierte Schnittstelle, um jene Attribute mit Schreib- und Leserechten auszustatten passend für die Klasse.

Get & Set Methode für Attribute



- Zugriff auf **private Attribute** von Außen:

```
class MyClass {  
  
    //private Attribute  
    private string attribut;  
  
    //Über Get Methode abfragen  
    public string GetAttribut() {  
        return attribut;  
    }  
    //Über Set Methode mit einem neuen Wert versehen  
    public void SetAttribut(string attribut) {  
        this.attribut = attribut;  
    }  
}
```

Erstelle für die Klasse Person Get und Set Methoden für die Attribute Name, Age und IsMale.

Erstelle für die Klasse Student Get und Set Methoden für die Attribute Schoolclass und Favorite

Get & Set Methoden

```
class Person {
    protected string name;
    protected int age;
    protected bool isMale;

    public Person() ...
}
```

```
public string GetName() { return name; }
public int GetAge() { return age; }
public bool IsMale() { return isMale; } //oder
public string GetMale() { return (isMale) ? "Mann" : "Frau"; }
```

```
public void SetName(string name) { this.name = name; }
public void SetAge( int age) { this.age = age; }
public void SetIsMale(bool isMale) { this.isMale = isMale; }
```

```
public void ThatsMe() ...
}
```

```
class Student : Person {

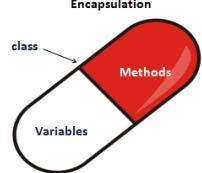
    protected string schoolclass;
    protected string favorite;
```

```
public string GetSchoolClass() { return schoolclass; }
public string GetFavorite() { return favorite; }
```

```
public void SetSchoolClass(string schoolclass) { this.schoolclass = schoolclass; }
public void SetFavorite(string favorite) { this.favorite = favorite; }
```

```
public void PresentStudent() ...
}
```

```
Frau - 25 - Susi
Kurt - 22 - 2BHT - SEW
Mann - 22 - Kurt
Susi ist 25 Jahre alt
Kurt hat SEW als Lieblingsgegenstand
```



The diagram shows a red and white capsule. The word 'Variables' is written in the white section, and 'Methods' is written in the red section. An arrow points from the word 'class' to the capsule.

```
static void Main(string[] args) {
    TestGetSet2Person2();
}

public static void TestGetSet2Person2() {
    Person p = new Person();
    p.SetName("Susi");
    p.SetAge(25);
    p.SetIsMale(false);
    p.ThatsMe();
    Student s = new Student ();
    s.SetName("Kurt");
    s.SetAge(22);
    s.SetIsMale(true);
    s.SetFavorite("SEW");
    s.SetSchoolClass("2BHT");
    s.PresentStudent();
    //oder
    Console.WriteLine($"{p.GetName()} ist " +
        $"{p.GetAge()} Jahre alt");
    Console.WriteLine($"{s.GetName()} hat " +
        $"{s.GetFavorite()} als Lieblingsgegenstand");
```

ToString

Zum Darstellen einer Instanz / eines Objekts in der Console wird die `ToString()` Methode der Klasse `Object` überschrieben.

```
public override string ToString() {
    return "Zeichenkette mit Attributen der Klasse";
}
```

```
//Darstellung von Objekten als Zeichenkette
public override string ToString() {
    return base.ToString() + $" - {Schoolclass} - {Favorite}";
}
```

ToString Methode

Person

- Erstelle eine ToString Methode für Person und gib als Zeichenkette Name – Alter und Frau/Herr retour.

```
//Darstellung von Objekten als Zeichenkette
public override string ToString() {
    string gender = (IsMale) ? "Mann" : "Frau";
    return
}
```

Student

- Erstelle eine ToString Methode für Student.
- Nutze die ToString Methode von Person mit Hilfe vom Aufruf: base.ToString() und
- ergänze diese um die Attribute Schoolclass und Favorite

```
//Darstellung von Objekten als Zeichenkette
public override string ToString() {
    return base.ToString()
}
```

```

class Person {
    protected string name;
    protected int age;
    protected bool isMale;

    public string GetName() { return name; }
    public int GetAge() { return age; }
    public string IsMale() { return (isMale) ? "Mann" : "Frau"; }

    public void SetName(string name) { this.name = name; }
    public void SetAge(int age) { this.age = age; }
    public void SetMale(bool isMale) { this.isMale = isMale; }

    //Darstellung von Objekten als Zeichenkette
    public override string ToString() {
        string gender = (isMale) ? "Mann" : "Frau";
        return $"{gender} - {name} - {age} ";
    }
}

class Student : Person {
    protected string schoolclass;
    protected string favorite;

    public string GetSchoolClass() { return schoolclass; }
    public string GetFavorite() { return favorite; }

    public void SetSchoolClass(string schoolclass) { this.schoolclass = schoolclass; }
    public void SetFavorite(string favorite) { this.favorite = favorite; }

    //Darstellung von Objekten als Zeichenkette
    public override string ToString() {
        return base.ToString() + $"- {schoolclass} - {favorite}";
    }
}

```

Frau - Susi - 25
 Frau - Susi - 25
 Mann - Kurt - 22 - 2BHT - SEW

```

class Program {
    public static void TestInheritancePerson1() ...
    public static void TestGetSetPerson2() ...
    public static void TestToStringPerson3() {
        Person p = new Person();
        p.SetName("Susi");
        p.SetAge(25);
        p.SetMale(false);
        Console.WriteLine(p.ToString());
        Console.WriteLine(p);
        Student s = new Student();
        s.SetName("Kurt");
        s.SetAge(22);
        s.SetMale(true);
        s.SetFavorite("SEW");
        s.SetSchoolClass("2BHT");
        Console.WriteLine(s);
    }
    static void Main(string[] args) {
        TestToStringPerson3();
    }
}

```

ToString-Methode der Klasse Object für die eigene Klasse anpassen:

Überschreiben der ToString Methode – stellt die Instanz der Klasse als Zeichenkette dar.

Zeichenkette der Basisklasse kann mit base.ToString()
abgerufen und genutzt werden.

ToString überschreiben



Konstruktorkette

Die Weiterleitung von einem Konstruktor an einen anderen Konstruktor wird Konstruktorkette genannt.

- Dies passiert automatisch oder explizit vom Programmierer
 - Aufruf eines Überladenen Konstruktors der selben Klasse ist mit **this** möglich.
 - Aufruf eines Konstruktors der Basisklasse ist mit **base** möglich.

Konstruktoren überladen

```
class Person {
    protected string name;
    protected int age;
    protected bool isMale;

    public Person() {
        Console.WriteLine("Konstruktor: Person()");
        this.name = "Dummy";
        this.age = -1;
    }
    public Person(string name, int age) {
        Console.WriteLine("Konstruktor: Person(string name, int age)");
        this.age = age;
        this.name = name;
    }
}

//Darstellung von Objekten als Zeichenkette
public override string ToString() ...
}
```

```
public static void TestKonstruktorkette1Person4() {
    Person p1 = new Person();
    Console.WriteLine(p1);
    Person p2 = new Person("Kurt", 24);
    Console.WriteLine(p2);
}
```

Wie lautet die Ausgabe?

```
Konstruktor: Person()
Frau - Dummy - -1
Konstruktor: Person(string name, int age)
Frau - Kurt - 24
```

Konstruktorkette - this

```
class Person {
    protected string name;
    protected int age;
    protected bool isMale;

    public Person() {
        Console.WriteLine("Konstruktor: Person()");
        this.name = "Dummy";
        this.age = -1;
    }
    public Person(string name, int age) {
        Console.WriteLine("Konstruktor: Person(string name, int age)");
        this.age = age;
        this.name = name;
    }
    public Person (string name, int age, bool isMale):this(name, age) {
        Console.WriteLine("Konstruktor: Person(string name, int age, bool isMale)");
        this.isMale = isMale;
    }
    //Darstellung von Objekten als Zeichenkette
    public override string ToString() ...
}
```

```
public static void TestKonstruktorkette1Person4() {
    Person p1 = new Person();
    Console.WriteLine(p1);
    Person p2 = new Person("Kurt", 24);
    Console.WriteLine(p2);
    Person p3 = new Person("Sissi", 15, false);
    Console.WriteLine(p3);
}
```

Innerhalb einer Klasse mit **this**

Wie lautet die Ausgabe?

```
Konstruktor: Person()
Frau - Dummy - -1
Konstruktor: Person(string name, int age)
Frau - Kurt - 24
Konstruktor: Person(string name, int age)
Konstruktor: Person(string name, int age, bool isMale)
Frau - Sissi - 15
```

Konstruktorkette mit base

Mit **base** auf Basisklasse weiterleiten
Beachte die AUSGABE!!

```
class Student : Person {
    protected string schoolclass;
    protected string favorite;

    public Student() {
        Console.WriteLine("Konstruktor: Student()");
    }
    public Student(string s, string f) {
        Console.WriteLine("Konstruktor: Student(string s, string f)");
        this.schoolclass = s;
        this.favorite = f;
    }
    public Student(string n, int a, bool m, string s, string f)
        :base (n, a, m) {
        Console.WriteLine("Konstruktor: " +
            "Student(string n, int a, bool m, string s, string f)");
        this.schoolclass = s;
        this.favorite = f;
    }
    //Darstellung von Objekten als Zeichenkette
    public override string ToString() {
        return base.ToString() + $"- {schoolclass} - {favorite}";
    }
}
```

```
Konstruktor: Person()
Frau - Dummy - -1
Konstruktor: Person(string name, int age)
Frau - Kurt - 24
Konstruktor: Person()
Konstruktor: Student()
Frau - Dummy - -1 - -
Konstruktor: Person()
Konstruktor: Student(string s, string f)
Frau - Dummy - -1 - 2BHIT - SEW
Konstruktor: Person(string name, int age)
Konstruktor: Person(string name, int age, bool isMale)
Konstruktor: Student(string name, int age, bool isMale, string s, string f)
Mann - Max - 22 - 2BHIT - SEW
```

```
public static void TestKonstruktorkette2Student4() {
    Person p1 = new Person();
    Console.WriteLine(p1);
    Person p2 = new Person("Kurt", 24);
    Console.WriteLine(p2);
    Student s1 = new Student();
    Console.WriteLine(s1);
    Student s2 = new Student("2BHIT", "SEW");
    Console.WriteLine(s2);
    Student s3 = new Student("Max", 22, true, "2BHIT", "SEW");
    Console.WriteLine(s3);
}
```

```
class Person {
    protected string name;
    protected int age;
    protected bool isMale;

    public Person() {
        Console.WriteLine("Konstruktor: Person()");
        this.name = "Dummy";
        this.age = -1;
    }
    public Person(string name, int age) {
        Console.WriteLine("Konstruktor: Person(string name, int age)");
        this.age = age;
        this.name = name;
    }
    public Person (string name, int age, bool isMale):this(name, age) {
        Console.WriteLine("Konstruktor: Person(string name, int age, bool isMale)");
        this.isMale = isMale;
    }
    //Darstellung von Objekten als Zeichenkette
    public override string ToString() [...]
}
```



Konstruktorkette mit base

```
public static void TestKonstruktorkette2Student4() {
    Console.WriteLine("---- p3 ----");
    Person p3 = new Person("Sissi", 15, false);
    Console.WriteLine(p3);
    Console.WriteLine("---- s1 ----");
    Student s1 = new Student();
    Console.WriteLine(s1);
    Console.WriteLine("---- s2 ----");
    Student s2 = new Student("2BHT", "SEW");
    Console.WriteLine(s2);
    Console.WriteLine("---- s3 ----");
    Student s3 = new Student("Max", 22, true, "2BHT", "SEW");
    Console.WriteLine(s3);
}
```

```
---- p3 ----
Konstruktor: Person(string name, int age)
Konstruktor: Person(string name, int age, bool isMale)
Frau - Sissi - 15
---- s1 ----
Konstruktor: Person()
Konstruktor: Student()
Frau - Dummy - -1 - -
---- s2 ----
Konstruktor: Person()
Konstruktor: Student(string s, string f)
Frau - Dummy - -1 - 2BHT - SEW
---- s3 ----
Konstruktor: Person(string name, int age)
Konstruktor: Person(string name, int age, bool isMale)
Konstruktor: Student(string n, int a, bool m, string s, string f)
Mann - Max - 22 - 2BHT - SEW
```

```
class Student : Person {

    protected string schoolclass;
    protected string favorite;

    public Student() {
        Console.WriteLine("Konstruktor: Student()");
    }
    public Student(string s, string f) {
        Console.WriteLine("Konstruktor: Student(string s, string f)");
        this.schoolclass = s;
        this.favorite = f;
    }
    public Student(string n, int a, bool m, string s, string f)
        :base (n, a, m) {
        Console.WriteLine("Konstruktor: " +
            "Student(string n, int a, bool m, string s, string f)");
        this.schoolclass = s;
        this.favorite = f;
    }
    //Darstellung von Objekten als Zeichenkette
    public override string ToString() {
        return base.ToString() + $"- {schoolclass} - {favorite}";
    }
}

class Person {
    protected string name;
    protected int age;
    protected bool isMale;

    public Person() {
        Console.WriteLine("Konstruktor: Person()");
        this.name = "Dummy";
        this.age = -1;
    }
    public Person(string name, int age) {
        Console.WriteLine("Konstruktor: Person(string name, int age)");
        this.age = age;
        this.name = name;
    }
    public Person (string name, int age, bool isMale):this(name, age) {
        Console.WriteLine("Konstruktor: Person(string name, int age, bool isMale)");
        this.isMale = isMale;
    }
    //Darstellung von Objekten als Zeichenkette
    public override string ToString() ...
```

this Schlüsselwort

This verweist auf die aktuelle Instanz einer Klasse
damit kann man auf alle Member zugreifen
this wird bei Namenskonflikten verwendet

```
this.<Attributname>
this.<Methodenname>
Classname(string x):this(x) {...}
```

base Schlüsselwort

base leitet an die direkte Basisklasse,
oder an eine der Basisklassen einer Instanz weiter.

```
base.ToString()  
Classname(string x):base(x) {...}
```

Special Student - Vererbung

```
class Person {
    protected string name;
    protected int age;
    protected bool isMale;

    public Person() ...
    public Person(string name, int age) ...
    public Person(string name, int age, bool isMale) ...
    //Darstellung von Objekten als Zeichenkette
    public override string ToString() ...
}
```

Vererbung über 3 Ebenen:

```
Person
  |
Student
  |
SpecialStudent
```

```
class SpecialStudent : Student {
    protected string info;
    public SpecialStudent() ...
    public SpecialStudent(string name, int age,
        bool isMale, string sclass, string fav, string info) ...
    public override string ToString() ...
}
```

Klasse SchoolGrade mit Attributen Subject, Description und Grade

```
class Student : Person {
    protected string schoolclass;
    protected string favorite;
    protected SchoolGrade[] grades;

    public Student() { //: base() - optional
        Console.WriteLine("Konstruktor: Student()");
        grades = new SchoolGrade[10];
        Console.WriteLine("Create Array");
    }

    public Student(string name, int age,
        bool isMale, string s, string f) ...

    //Darstellung von Objekten als Zeichenkette
    public override string ToString() ...
}
```

```
class Student : Person {
    protected string schoolclass;
    protected string favorite;
    protected SchoolGrade[] grades;

    public Student() { //: base() - optional
        Console.WriteLine("Konstruktor: Student()");
        grades = new SchoolGrade[10];
        Console.WriteLine("Create Array");
    }
}
```

```
public Student(string name, int age,
    bool isMale, string s, string f) : this () {
    Console.WriteLine("Konstruktor: Student(string n, " +
        " int a, bool m, string s, string f) : this()");
    base.name = name;
    base.age = age;
    base.isMale = isMale;
    this.schoolclass = s;
    this.favorite = f;
}
```

Student & Grades Array

Klasse

SchoolGrade
mit Property

Subject,
Description und
Grade

```
class SchoolGrade {
    public string Subject;
    public string Description;
    public int Grade;
}
```

Special Student & Main

```
class SpecialStudent : Student {  
    protected string info;  
    public SpecialStudent() {  
        Console.WriteLine("SpecialStudent");  
    }  
    public SpecialStudent(string name, int age,  
        bool isMale, string sclass, string fav, string info)  
    : base(name, age, isMale, sclass, fav) {  
        Console.WriteLine("SpecialStudent(name,age,male,class,fav,info)");  
        this.info = info;  
    }  
    public override string ToString() {  
        return base.ToString() + $" - {info}";  
    }  
}  
  
public static void TestSpecialStudent() {  
    Student s = new Student("Kurt", 23, true, "2CHIT", "SEW");  
    Console.WriteLine(s);  
    Console.WriteLine();  
    SpecialStudent ss1 = new SpecialStudent();  
    Console.WriteLine(ss1);  
    Console.WriteLine();  
    SpecialStudent ss2 = new SpecialStudent("Susi", 23, true, "2CHIT", "SEW", "leistungsstark");  
    Console.WriteLine(ss2);  
}  
  
static void Main(string[] args) {  
    TestSpecialStudent6();  
}
```

Ausgabe – Special Student

```
public static void TestSpecialStudent() {  
    Student s = new Student("Kurt", 23, true, "2CHIT", "SEW");  
    Console.WriteLine(s);  
    Console.WriteLine();  
    SpecialStudent ss1 = new SpecialStudent();  
    Console.WriteLine(ss1);  
    Console.WriteLine();  
    SpecialStudent ss2 = new SpecialStudent("Susi", 23, true, "2CHIT", "SEW", "leistungsstark");  
    Console.WriteLine(ss2);  
}
```

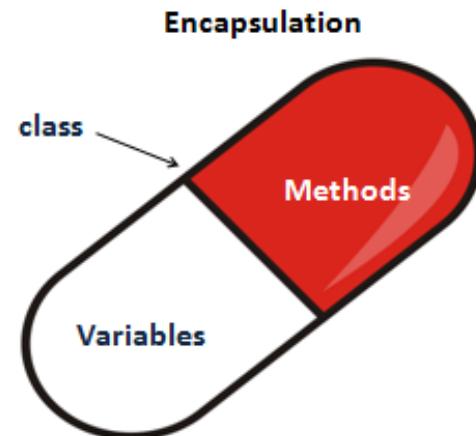
Das Array für die Noten
in Student wird immer
angelegt.

Die Konstruktorkette in
Student ist so
implementiert, dass der
parameterlose
Konstruktor in jedem Fall
aufgerufen wird.

```
[Konstruktor: Person()  
[Konstruktor: Student()  
Create Array  
Konstruktor: Student(string n, int a, bool m, string s, string f) : this()  
Mann - Kurt - 23 - 2CHIT - SEW  
  
Konstruktor: Person()  
Konstruktor: Student()  
Create Array  
SpecialStudent  
Frau - Dummy - -1 - - -  
  
Konstruktor: Person()  
Konstruktor: Student()  
Create Array  
Konstruktor: Student(string n, int a, bool m, string s, string f) : this()  
SpecialStudent(name,age,male,class,fav,info)  
Mann - Susi - 23 - 2CHIT - SEW - leistungsstark
```

Properties

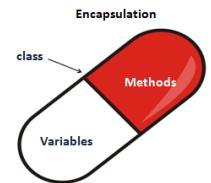
Datenkapselung wird in C# mit Properties statt Get- und Set-Methoden realisiert, diese Technologie ist in anderen Sprachen nicht verfügbar.



Vorteil:

Schreibweise von public Attributes und trotzdem eine Wertüberprüfung möglich.

Properties für Attribute



Zugriff auf Attribute von Außen – zwei Schreibweisen:

Langschreibweise:

- Erstelle ein privates/geschütztes Attribut und stelle ein Property mit gleichem Namen nur großem Anfangsbuchstaben zur Verfügung

```
class MyClass {
    //private Attribute
    private string attribut;

    //Langschreibweise verweist auf ein Attribut
    public string Attribut {
        get { return attribut; }
        set { this.attribut = value; }
    }
}
```

Erstelle für die Klasse Person Properties in der Langschreibweise für die Attribute Name, Age und IsMale.

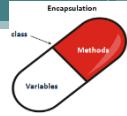
Kurzschreibweise:

- Erstelle ein Property – Attribut wird im Hintergrund automatisch erzeugt,
 - Wertüberprüfung nicht möglich
 - nur Leserechte setzen möglich

```
class MyClass {
    //Kurzschreibweise
    public string Attribut1 { get; set; }
    //Kurzschreibweise und nur Leserechte
    public string Attribut2 { get; private set; }
    public string Attribut3 { get; }

}
```

Erstelle für die Klasse Student Properties in der Kurzschreibweise für die Attribute Schoolclass und Favorite



Properties

**Attribute schreibt man klein
Properties schreibt man Groß**

```
class Person {
    //Attribute
    protected string name;
    protected int age;
    protected bool isMale;
    //Properties
    public string Name { get { return name; } set { name = value; } }
    public int Age { get { return age; } set { age = (value>0) ? age = value : age = 0; } }
    public bool IsMale { get { return isMale; } set { isMale = value; } }
```

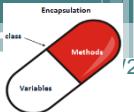
Properties in der Langschreibweise
benötigen ein Attribut und bieten
eine Wertüberprüfung im **set**

```
class Student : Person {

    //Keine Attribute bei Kurzschreibweise von Properties
    //wird es zusätzlich deklariert, existiert es doppelt
    //protected string schoolclass;
    //protected string favorite;

    //Properties mit Kurzschreibweise
    //das Attribut wird automatisch im Hintergrund erstellt,
    public string Schoolclass { get; set; }
    public string Favorite { get; private set; } //get;}
```

Properties in der Kurzschreibweise,
Wertüberprüfung nicht möglich –
nur Leserechte implementierbar



Teste Properties

```

class Person {
    //Attribute
    protected string name;
    protected int age;
    protected bool IsMale;
    //Properties
    public string Name { get { return name; } set { name = value; } }
    public int Age { get { return age; } set { age = (value>0) ? age = value : age = 0; } }
    public bool IsMale { get { return IsMale; } set { IsMale = value; } }

    //Darstellung von Objekten als Zeichenkette
    public override string ToString() {
        string gender = (IsMale) ? "Mann" : "Frau";
        return($"{gender} - {Name} - {Age}");
    }
}

class Student : Person {

    //Keine Attribute bei Kurzschreibweise von Properties
    //wird es zusätzlich deklariert, existiert es doppelt
    //protected string schoolclass;
    //protected string favorite;

    //Properties mit Kurzschreibweise
    //das Attribut wird automatisch im Hintergrund erstellt,
    public string Schoolclass { get; set; }
    public string Favorite { get; private set; } //get;

    //Darstellung von Objekten als Zeichenkette
    public override string ToString() {
        return base.ToString() + $"- {Schoolclass} - {Favorite}";
    }
}

```

```
Frau - Sue Permarkt - 0
Mann - Ro Mantik - 14 - 2BHTI -
```

```
static void Main(string[] args) {
    TestPropertiesPerson5();
}
```

```

public static void TestPropertiesPerson5() {
    Person p1 = new Person();
    p1.Name = "Sue Permarkt";
    p1.Age = -33;
    p1.IsMale = false;

    Student s1 = new Student();
    s1.Name = "Ro Mantik";
    s1.Age = 14;
    s1.IsMale = true;
    //s1.Favorite = "SEW"; - keine Schreibrechte
    s1.Schoolclass = "2BHTI";

    Console.WriteLine(p1);

    Console.WriteLine(s1);
}
```

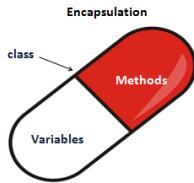
Properties können wie public Attribute genutzt werden

```
class MyClass {  
    public string Attribut1 { get; set; }  
}  
  
MyClass c = new MyClass() { Attribut1 = "Hallo Welt" };
```

Objekt Initialisierer

Standardkonstruktor kann kombiniert werden mit den Properties einer Klasse, wobei beim Erstellen des Objekts in geschwungenen Klammern alle verfügbaren Properties gesetzt werden können.

Erstelle eine Testmethode, welche ein Objekt der Klasse Person und ein Objekt der Klasse Student instanziiert – nutze zum Initialisieren der Werte den Objekt Initialisierer.



Properties in der Main

```

public static void TestPropertiesPerson5() {
    Person p1 = new Person();
    p1.Name = "Sue Permarkt";
    p1.Age = -33;
    p1.IsMale = false;
    Person p2 = new Person() { Name = "Mike Rohsoft", Age = 33, IsMale = true };
    Student s1 = new Student();
    s1.Name = "Ro Mantik";
    s1.Age = 14;
    s1.IsMale = true;
    //s1.Favorite = "SEW"; - keine Schreibrechte
    s1.Schoolclass = "2BHT";
    Student s2 = new Student() {
        Name = "Kurt",
        Age = 16,
        IsMale = true,
        Schoolclass = "2BHT"
    };
    Console.WriteLine(p1);
    Console.WriteLine(p2);
    Console.WriteLine(s1);
    Console.WriteLine(s2);
}

```

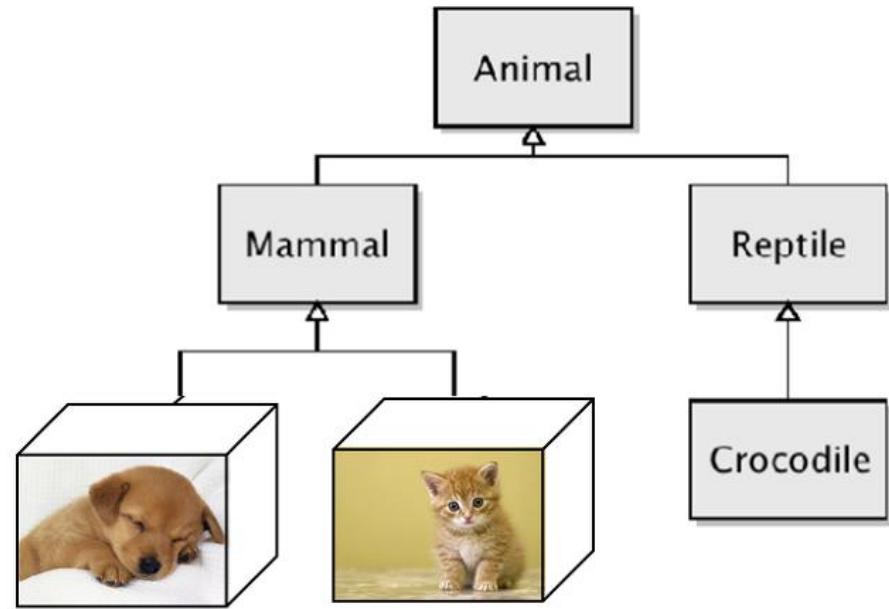
```

Frau - Sue Permarkt - 0
Mann - Mike Rohsoft - 33
Mann - Ro Mantik - 14 - 2BHT -
Mann - Kurt - 16 - 2BHT -

```

Properties können wie public Attribute genutzt werden,

oder direkt nach Aufruf des Konstruktors gesetzt werden.



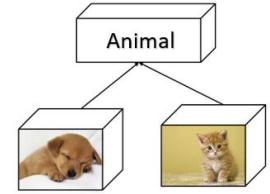
Abstrakte Klassen

Dinge die es im echten Leben nicht gibt können als abstrakte Klassen implementiert werden:

Animal: Dog, Cat, Tiger, ...

Form: Circle, Triangle, Rectangle, ...

Abstrakte Klasse Animal

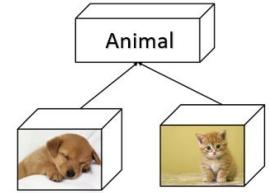


- Abstrakte Klassen beginnen mit A
 - Erstelle eine Klasse AAnimal
 - Mit einem Property Name und einer Methode Sound
-
- Animal kann nicht instanziert werden!

```
public abstract class AAnimal {  
    public string Name { get; set; }  
    public void Sound(string sound) {  
        Console.WriteLine($"Das Tier {sound}");  
    }  
}
```

```
AAnimal a.. = new AAnimal();
```

Erstelle Elefant und Katze



- Erbe von der Klasse Animal.
- Erstelle Objekte der Klasse Elefant und Katze, setze Namen & nutze die Methode Sound

```

public abstract class AAnimal {
    public string Name { get; set; }
    public void Sound(string sound) {
        Console.WriteLine($"{Name} {sound}");
    }
}
  
```

```

public class Elefant : AAnimal { }
public class Cat : AAnimal { }
  
```

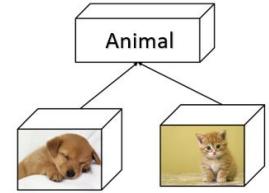
```

Dumbo trörööt
Mia miaut
  
```

```

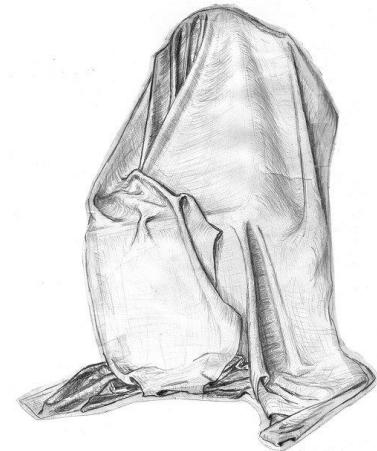
static void TestAnimal() {
    //AAnimal instanziieren nicht möglich
    //AAnimal a = new AAnimal();
    Elefant e = new Elefant();
    e.Name = "Dumbo";
    e.Sound("trörööt");
    Cat c = new Cat();
    c.Name = "Mia";
    c.Sound("miaut");
}
  
```

Array von Tieren



- Deklarierter Datentyp (Compilezeit) ist Animalarray
- Datentyp zur Laufzeit (Instanz der Klasse) ist eine davon abgeleitete Klasse
 - Methoden von Animal können aufgerufen werden
 - Abgeleitete Klasse stellt gleichnamige Methode neu implementiert zur Verfügung

Werden die Methoden
der Basisklasse (Compilezeit) oder
der abgeleiteten Klasse (Laufzeit) verwendet?!



Statische Bindung

Methode der Basisklasse
wird auch in der abgeleiteten Klasse ausimplementiert...

Statische Bindung

Nicht Polymorphes Verhalten

Typ des Objekts zur Compilezeit

Schlüsselwort new

Überdecken



Statische Bindung

- Wenn man eine Methode in der Basisklasse erstellt
 - und eine gleichnamige Methode in der abgeleiteten Klasse zur Verfügung stellt wird ein Hinweis angezeigt:

```
public class Elefant : AAnimal {  
    public void Move() {  
        Console. if   void Elefant.Move()  
    }  
}
```

""Elefant.Move()" blendet den vererbten Member "AAnimal.Move()" aus. Verwenden Sie das new-Schlüsselwort, wenn das Ausblenden vorgesehen war.

Mögliche Korrekturen anzeigen (Alt+Eingabe oder Strg+.)

Verwenden Sie das new-Schlüsselwort,
wenn das Ausblenden vorgesehen war.

Array von Tieren

```

public class Elefant : AAnimal {
    public new void Move() {
        Console.WriteLine($"Der Elefant {Name} spaziert");
    }
}

public class Cat : AAnimal {
    public new void Move() {
        Console.WriteLine($"Die Katze {base.Name} springt");
    }
}

public class Dog : AAnimal {
    public new void Move() {
        Console.WriteLine($"Der Hund {base.Name} rennt");
    }
}

public class Horse : AAnimal {
    public new void Move() {
        Console.WriteLine($"Das Pferd {base.Name} galoppiert");
    }
}

public class Mouse : AAnimal {
    public new void Move() {
        Console.WriteLine($"Die Maus {base.Name} eilt");
    }
}

```

```

public abstract class AAnimal {
    public string Name { get; set; }

    public void Move() {
        Console.WriteLine($"Das Tier bewegt sich ... ");
    }
}

static void TestAnimalArray() {
    AAnimal[] animals = new AAnimal[5];
    animals[0] = new Elefant() { Name = "Dumbo" };
    animals[1] = new Cat() { Name = "Minka" };
    animals[2] = new Dog() { Name = "Bello" };
    animals[3] = new Horse() { Name = "Pegasus" };
    animals[4] = new Mouse() { Name = "Cherry" };

    foreach (AAnimal a in animals) {
        a.Move();
    }
}

```

Wie lautet die Ausgabe?





Problematik erkennen

- Ist das die gewünschte Ausgabe?

```
static void TestConcreteInstance() {  
    Elefant a1 = new Elefant() { Name = "Dumbo" };  
    Cat a2 = new Cat() { Name = "Minka" };  
    Dog a3 = new Dog() { Name = "Bello" };  
    Horse a4 = new Horse() { Name = "Pegasus" };  
    Mouse a5 = new Mouse() { Name = "Cherry" };  
    a1.Move();  
    a2.Move();  
    a3.Move();  
    a4.Move();  
    a5.Move();  
}
```

```
Der Elefant Dumbo spaziert  
Die Katze Minka springt  
Der Hund Bello rennt  
Das Pferd Pegasus galoppiert  
Die Maus Cherry eilt
```

```
static void TestAnimalArray() {  
    AAnimal[] animals = new AAnimal[5];  
    animals[0] = new Elefant() { Name = "Dumbo" };  
    animals[1] = new Cat() { Name = "Minka" };  
    animals[2] = new Dog() { Name = "Bello" };  
    animals[3] = new Horse() { Name = "Pegasus" };  
    animals[4] = new Mouse() { Name = "Cherry" };  
    foreach (AAnimal a in animals) {  
        a.Move();  
    }  
}
```

```
Das Tier macht ein Geräusch ...  
Das Tier macht ein Geräusch ...
```

Polymorphie

Vielgestaltigkeit

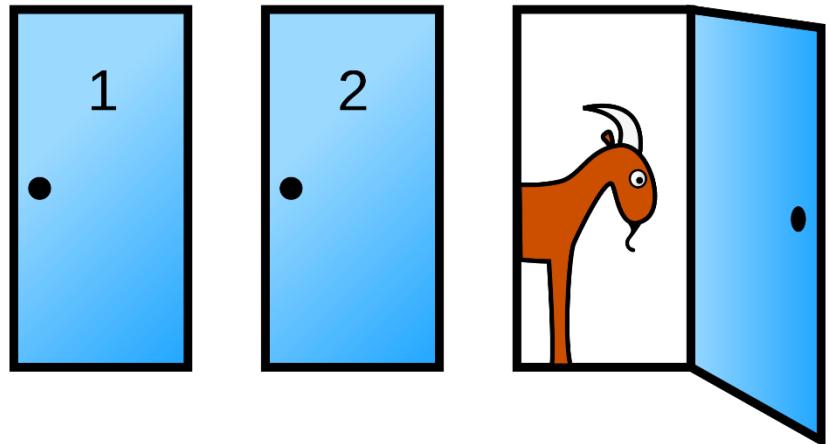
Polymorphes Verhalten

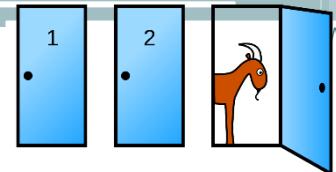
Dynamische Bindung

Typ des Objekts zur Laufzeit

Überschreiben einer Methode

Schlüsselwörter `virtual` & `override`





Methode überschreiben

```
public abstract class AAnimal {
    public string Name { get; set; }

    public void Sound(string sound) {
        Console.WriteLine($"{Name} {sound}");
    }
    public virtual void Sound() {
        Console.WriteLine("Das Tier macht ein Geräusch ... ");
    }
}

public class Elefant : AAnimal {
    public override void Sound() {
        Console.WriteLine($"Der Elefant {Name} tröttet");
    }
}
public class Cat : AAnimal {
    public override void Sound() {
        Console.WriteLine($"Die Katze {base.Name} miaut");
    }
}
```

Sound wird in AAnimal **überladen**
(2 Methoden gleichen Namens
unterschiedliche Parameterliste)

Sound wird in Elefant
und Cat **überschrieben**
Schlüsselwörter virtual
und override

Überschreiben

```

public class Elefant : AAnimal {
    public override void Sound() {
        Console.WriteLine($"Der Elefant {base.Name} tröttet");
    }
}

public class Cat : AAnimal {
    public override void Sound() {
        Console.WriteLine($"Die Katze {base.Name} miaut");
    }
}

public class Dog : AAnimal {
    public override void Sound() {
        Console.WriteLine($"Der Hund {base.Name} bellt");
    }
}

public class Horse : AAnimal {
    public override void Sound() {
        Console.WriteLine($"Das Pferd {base.Name} wiehert");
    }
}

public class Mouse : AAnimal {
    public override void Sound() {
        Console.WriteLine($"Die Maus {base.Name} piepst");
    }
}

```

```

public abstract class AAnimal {
    public string Name { get; set; }

    public void Sound(string sound) {
        Console.WriteLine($"{Name} {sound}");
    }

    public virtual void Sound() {
        Console.WriteLine($"Das Tier macht ein Geräusch ... ");
    }

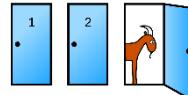
    public void Move() {
        Console.WriteLine($"Das Tier bewegt sich ... ");
    }
}

static void TestOverrideAnimalArray() {
    AAnimal[] animals = new AAnimal[5];
    animals[0] = new Elefant() { Name = "Dumbo" };
    animals[1] = new Cat() { Name = "Minka" };
    animals[2] = new Dog() { Name = "Bello" };
    animals[3] = new Horse() { Name = "Pegasus" };
    animals[4] = new Mouse() { Name = "Cherry" };

    foreach (AAnimal a in animals) {
        a.Sound();
    }
}

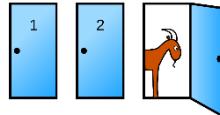
```

Wie lautet die Ausgabe?



Überschreiben vs Überdecken

Überschreiben

- Polymorphes Verhalten
- Dynamische Bindung
- Typ des Objekts zur Laufzeit
- virtual & override
- Überschreiben 
- Sinnvoll zu nutzen, man verwendet die Methoden der Instanz einer Klasse, nicht die Methoden laut Deklaration der Variable

Überdecken

- Nicht Polymorphes Verhalten
- Statische Bindung
- Typ des Objekts zur Compilezeit
- Schlüsselwort new
- Überdecken
- Einsatzgebiet fraglich



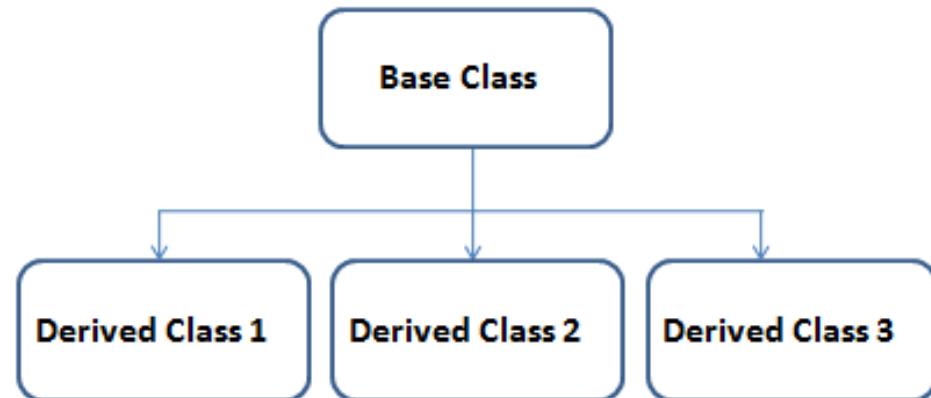
Überladen

- Methode Sound zwei mal innerhalb der Klasse
 - mit unterschiedlicher Parameterliste

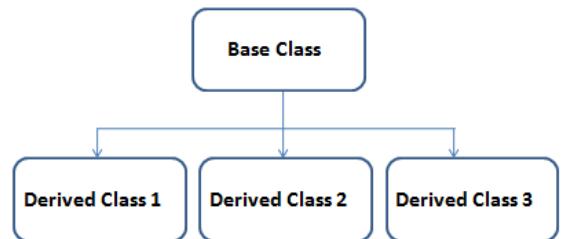
```
public class Animal {  
    public string name;  
  
    public void Sound() {  
        Console.WriteLine($"Das Tier macht ein Geräusch ... ");  
    }  
  
    public void Sound(string sound) {  
        Console.WriteLine($"{name} {sound}");  
    }  
}
```

Beispiel Baseclass & Childclass

Vererbung
Sichtbarkeiten
Properties
Konstruktor
Konstruktorkette



Aufgabenstellung Klassen



- Erstelle eine Basisklasse „Baseclass“
 - mit 3 Attributen „attribute_x“ vom Datentyp Integer – setze die Zugriffsmodifizierer auf private, protected und public.
 - Erstelle 3 Methoden – verwende die Zugriffsmodifizierer private, protected und public.
- Erstelle eine abgeleitete Klasse „ChildClass“
 - die von Baseclass erbt
 - mit 3 Attributen „text_x“ vom Datentyp String – setze die Zugriffsmodifizierer auf private, protected und public.

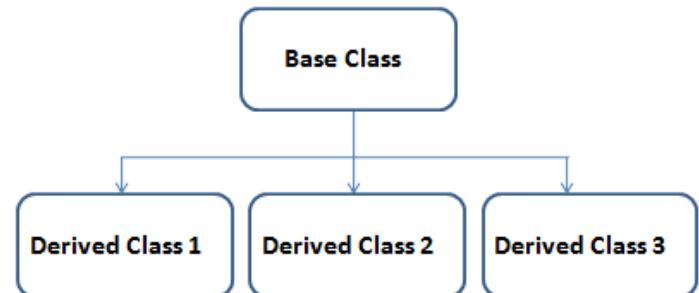
BaseClass &

```
public class BaseClass
{
    private int attribute1;
    protected int attribute2;
    public int attribute3;

    private void method1 ()
    {
        Console.WriteLine("Methode eins" );
    }

    protected void method2 ()
    {
        Console.WriteLine("Methode zwei" );
    }

    public void method3()
    {
        Console.WriteLine("Methode drei");
    }
}
```

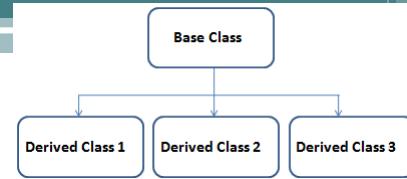


ChildClass

```
public class ChildClass : BaseClass
{
    private string text1;
    protected string text2;
    public string text3;

}
```

Childclass



Inheritance - Microsoft Visual Studio

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP

Sign in

Solution Explorer Start Any CPU Program.cs

Search Solution Explorer (Ctrl+ü)

Solution 'Inheritance' (1 project)

- Inheritance
 - Properties
 - References
 - App.config
 - BaseClass.cs
 - BaseClass
 - ChildClass.cs
 - Program.cs

ChildClass.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

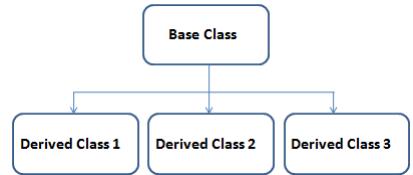
namespace Inheritance
{
    public class ChildClass : BaseClass
    {
        private string text1;
        protected string text2;
        public string text3;

        public ChildClass()
        {
            att
        }
    }
}
```

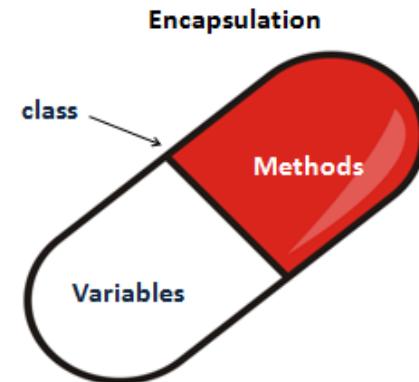
Quick Launch (Ctrl+Q)

The code editor shows the ChildClass.cs file. It defines a class ChildClass that inherits from BaseClass. The ChildClass contains three string fields: text1, text2, and text3. It also has a constructor that includes a placeholder "att". A tooltip or dropdown menu is open at the bottom right, showing various .NET attributes like AssemblyLoadEventHandler, async, AsyncCallback, Attribute, attribute2, attribute3, AttributeTargets, AttributeUsageAttribute, and await.

Fragen zu Childclass



- Wieso ist attribut1 nicht zu sehen?
 -
- Welche Möglichkeiten gibt es auf dieses Attribut zuzugreifen?
 -
- Erstelle für das erste Attribut ein Property, das Leserechte jedoch keine Schreibrechte hat.
- Welche Namenskonvention ist hier einzuhalten?
 -



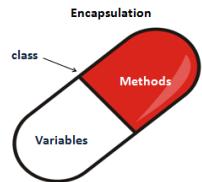
Properties

Um den Zustand des Objektes gewährleisten zu können, müssen die Eigenschaften (Attribute) des Objekts „geschützt“ werden.

Dies wird mit Properties oder Get- und Set-Methoden realisiert.

Ergänzende Informationen:

http://openbook.rheinwerk-verlag.de/visual_csharp_2010/visual_csharp_2010_03_005.htm#mjb3365746ed7c845fdc414b50306a0693



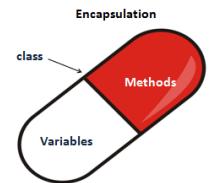
Aufgabenstellung Properties

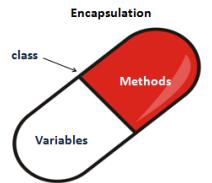
- Erstelle für alle 3 Attribute in der Basisklasse public Properties.
- Setzte beim ersten Attribut nur Leseberechtigung.
- Beim zweiten Attribut darf der Wert nur größer 0 sein.
- Beim dritten Attribut darf die Schnellschreibweise von Properties benutzt werden.

Properties

- Wie ist der Unterschied zwischen der Schnellschreibweise und der Langschreibweise von Attributen?

```
private int attribute1;  
protected int attribute2;  
  
//Nur Leseberechtigung  
public int Attribute1 { get { return attribute1; } }  
  
//Langschreibweise mit Überprüfungsmöglichkeit  
public int Attribute2  
{  
    get  
    {  
        return attribute2;  
    }  
    set  
    {  
        if (value > 0)  
            attribute2 = value;  
    }  
}  
  
//Schnellschreibweise  
public int Attribute3 { get; set; }
```



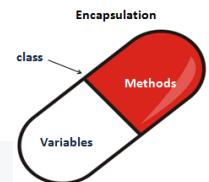


Properties – Lese & Schreibzugriff

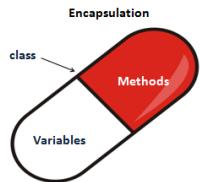
- Was fällt bei Attribut3 auf?

- Nachteil der Schnellschreibweise:

Langschreibweise vs Kurzschreibweise



```
protected int attribute2;  
//Langschreibweise mit Überprüfungsmöglichkeit  
public int Attribute2  
{  
    get { return attribute2; }  
    set { attribute2 = value; }  
}  
  
//Schnellschreibweise  
public int Attribute3 { get; set; }
```



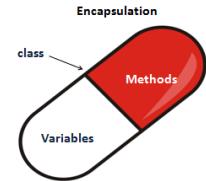
Frage

- Was passiert wenn beides Attribut und Property in der Klasse geschrieben werden? (Teste diesen Fall!)

```
public int attribute3;  
//Schnellschreibweise  
public int Attribute3 { get; set; }
```

- Beide Variablen sind verfügbar – in Summe hat die Klasse dann 4 Attribute:
 - attribute1, attribute2 (je mit dazugehörigen Property)
 - attribute3 (public) und Property Attribute3

Testen der Properties



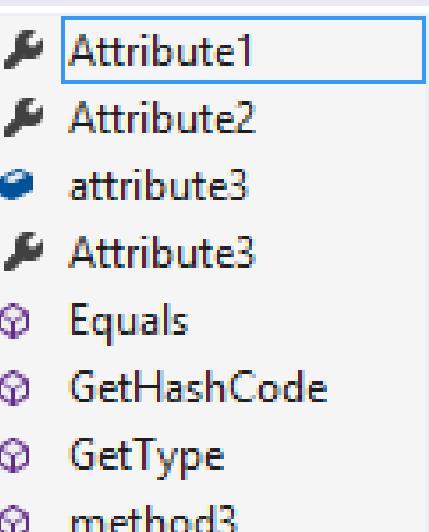
- Erzeuge eine Klasse – TestClass mit einer Main.
- Erstelle ein Objekt der Klasse BaseClass und finde heraus welche Werte gesetzt werden können und welche Werte abgefragt werden können.
- Erstelle ein Objekt der Klasse ChildClass und finde heraus welche Werte gesetzt werden können und welche Werte abgefragt werden können.

TEST BaseClass

- Bei der Baseclass stehen alle Properties und das öffentliche Attribut „attribute3“ zur Verfügung

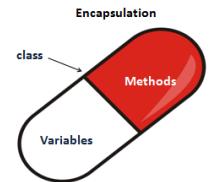
```
class TestClass
{
    public static void Main(String[] args)
    {
        BaseClass bc = new BaseClass();
        ChildClass cb = new ChildClass();

        bc.
    }
}
```



The screenshot shows an IDE's code editor with a dropdown menu open at the cursor position. The menu lists several members of the `BaseClass` type, each preceded by a small icon indicating its nature (e.g., attribute, method). The member `Attribute1` is highlighted with a blue border.

- Attribute1
- Attribute2
- attribute3
- Attribute3
- Equals
- GetHashCode
- GetType
- method3
- ToString



TEST ChildClass

- Bei der Childclass stehen alle Properties der Basisklasse und das öffentliche Attribut der Basisklasse „attribute3“ zur Verfügung, sowie das öffentliche Attribut „text“ der Childclass

```
public static void Main(String[] args)
{
    BaseClass bc = new BaseClass();
    ChildClass cb = new ChildClass();
```

The screenshot shows a code editor with the following code:

```
cb.
```

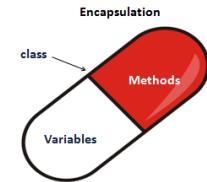
A completion dropdown menu is open, listing the following members:

- Attribute1
- Attribute2
- attribute3
- Attribute3
- Equals
- GetHashCode
- GetType
- method3
- text3

The member "Attribute1" is highlighted with a blue border. To the right of the dropdown, a portion of another class definition is visible:

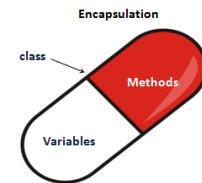
```
int BaseClass
```

Teste selbstständig Lese- und Schreibrechte



- Setzte Werte für alle Properties (soweit möglich)
- Setze für die öffentlichen Attribute Werte
- Gib alle Werte in der Konsole aus

Frage zur Leseberechtigung

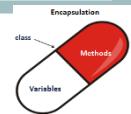


- Was passiert wenn man Attribute1 einen Wert zuweisen möchte?

A screenshot of a code editor showing a C# program. The code defines a `TestClass` with a `Main` method. Inside the `Main` method, two objects are created: `BaseClass bc` and `ChildClass cc`. Below this, two assignments are made: `bc.Attribute1 = 3;` and `cc.Attribute1 = 2;`. Both assignments are underlined with red, indicating they are errors. The code editor's status bar shows "100 %".

Error List

Description
1 Property or indexer 'Inheritance.BaseClass.Attribute1' cannot be assigned to -- it is read only
2 Property or indexer 'Inheritance.BaseClass.Attribute1' cannot be assigned to -- it is read only



```
public static void Main(String[] args)
{
    BaseClass bc = new BaseClass();
    ChildClass cc = new ChildClass();

    //bc.Attribute1 = 3;
    //cc.Attribute1 = 2;

    bc.attribute3 = -4;
    cc.attribute3 = 3;

    bc.Attribute2 = -5;
    cc.Attribute2 = -3;

    cc.text3 = "sew macht spass!";
    Console.WriteLine("Attribut1: {0} und {1}", bc.Attribute1, bc.Attribute1);
    Console.WriteLine("Attribut3: {0} und {1}", bc.Attribute3, cc.Attribute3);
    Console.WriteLine("attribut3: {0} und {1}", bc.attribute3, cc.attribute3);
    Console.WriteLine("Attribut2: {0} und {1}", bc.Attribute2, cc.Attribute2);
    Console.WriteLine(cc.text3);

}
```

**Attribut1: 0 und 0
Attribut3: 0 und 0
attribut3: -4 und 3
Attribut2: 0 und 0
sew macht spass!
Drücken Sie eine beliebige Taste . . .**



Konstruktor

- Was ist ein Konstruktor?

▫

- Gibt es in jeder Klasse einen Konstruktor?

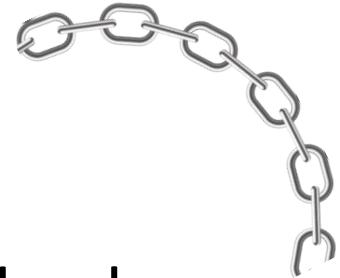
▫

- Kann mehrere Konstruktoren in einer Klasse geben?

▫

- Wenn ein Konstruktor mit 2 Parameter erstellt wurde, gibt es dann auch noch den parameterlosen Konstruktor?

▫



Fragen – Schlüsselwörter

- Mit welchem Schlüsselwort kann auf Methoden bzw Attribute der Basisklasse zugegriffen werden?
 -
- Mit welchem Schlüsselwort referenziert man auf die aktuelle Instanz einer Klasse?
 -
- Mit welchem Schlüsselwort kann sichergestellt werden, dass eine Klasse keine Basisklasse mehr sein kann?
 -

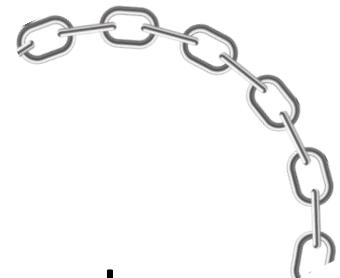


Konstruktoren

- Erstelle eine Klasse GrandChildClass
- dieses erbt von ChildClass und
- hat ein privates Attribut sum vom Datentyp Integer
- Erstelle ein Property zum Abfragen dieses Wertes.

```
class GrandChildClass : ChildClass
{
    private int sum;

    public int Sum
    {
        get {
            sum = base.Attribute1 +
                  base.Attribute2 +
                  base.attribute3 +
                  base.Attribute3;
            return sum;
        }
    }
}
```



Konstruktor

- Erstelle in jeder Klasse: BaseClass, ChildClass und GrandChildClass einen Konstruktor ohne Parameter und erzeuge jeweils eine Konsolenausgabe.
- Erzeuge für alle Klassen ein Objekt und analysiere die Konsolenausgabe.

```
public Class()
{
    Console.WriteLine("Konstruktor von Class");
}
```



Testfälle zu Konstruktoren

- Nur Baseclass
 - Ein Aufruf

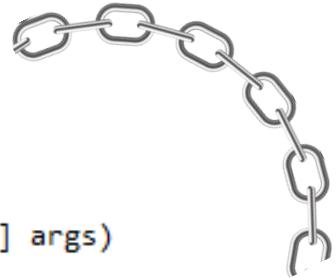
```
public static void Main(String[] args)
{
    BaseClass bc = new BaseClass();
```

Konstruktor von BaseClass
Drücken Sie eine beliebige Taste . . .

- Nur ChildClass
 - Aufruf von BaseClass und ChildClass

```
public static void Main(String[] args)
{
    //BaseClass bc = new BaseClass();
    ChildClass cc = new ChildClass();
```

Konstruktor von BaseClass
Konstruktor von ChildClass
Drücken Sie eine beliebige Taste . . .



Testfälle Konstruktorkette

- Nur GrandChildClass
 - Aufruf aller Konstruktoren aller Basisklassen

```
public static void Main(String[] args)
{
    //BaseClass bc = new BaseClass();
    //ChildClass cc = new ChildClass();
    GrandChildClass gcc = new GrandChildClass();
```

Konstruktor von BaseClass
Konstruktor von ChildClass
Konstruktor von GrandChildClass

- Testen alle 3 Klassen gleichzeitig, erzeuge eine übersichtliche Konsolenausgabe.



Testfall Konstruktorenverkettung:

```
public static void Main(String[] args)
{
    Console.WriteLine("\nBasisklasse wird getestet:");
    BaseClass bc = new BaseClass();
    Console.WriteLine("\nKindklasse wird getestet:");
    ChildClass cc = new ChildClass();
    Console.WriteLine("\nEnkerlklaasse wird getestet:");
    GrandChildClass gcc = new GrandChildClass();
```

Wie lautet die Ausgabe?

Ein Objekt der Klasse ruft bei der Erstellung immer zuerst den Konstruktor der eigenen Basisklasse auf, danach den eigenen Konstruktor!

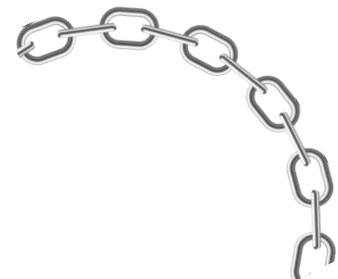


Konstruktorenüberladung

- Erstelle für die Klasse BaseClass
 - einen Konstruktor der alle 3 Attribute initialisiert.
- Erstelle in der Klasse ChildClass
 - einen Konstruktor der alle 3 Attribute der Basisklasse und alle 3 Zeichenketten der eigenen Klasse verwendet.
- Erstelle für die GrandChildClass
 - einen Konstruktor der alle Attribute der Basisklassen initialisiert, und die Summe berechnet.

HINWEIS:

- Nutze wenn vorhanden immer die Properties!



Konstruktoren von BaseClass

```
public BaseClass()
{
    Console.WriteLine("Konstruktor von BaseClass");
}

public BaseClass(int attribute1)
{
    Console.WriteLine("Konstruktor von BaseClass mit einem Parameter");
    this.attribute1 = attribute1;
}

public BaseClass(int attribute1, int attribute2, int attribute3)
{
    Console.WriteLine("Konstruktor von BaseClass mit Parameter");
    this.attribute1 = attribute1;
    this.Attribute2 = attribute2;
    this.Attribute3 = attribute3;
}
```

- Standardkonstruktor
- Konstruktor mit einem Parameter
- Konstruktor für alle 3 Attribute

Konstruktoren von ChildClass



```
public class ChildClass : BaseClass
{
    private string text1;
    protected string text2;
    public string text3;

    public ChildClass()
    {
        Console.WriteLine("Konstruktor von ChildClass");
    }

    public ChildClass(int attribute1)
        : base(attribute1)
    {
        Console.WriteLine("Konstruktor von ChildClass mit einem Parameter.");
    }

    public ChildClass(int attribute1, int attribute2, int attribute3, String text1)
        : base(attribute1, attribute2, attribute3)
    {
        Console.WriteLine("Konstruktor von ChildClass mit Parametern: {0}, {1}, {2}, {3}",
            Attribute1, Attribute2, Attribute3, text1);
        this.text1 = text1;
    }
}
```



Konstruktoren von GrandChildClass

```
public GrandChildClass()
{
    Console.WriteLine("Konstruktor von GrandChildClass");
    sum = Sum;
}

public GrandChildClass(int attribute1)
    :base(attribute1)
{
    Console.WriteLine("Konstruktor von GrandChildClass mit einem Parameter");
}

public GrandChildClass(int attribute1, int attribute2, int attribute3, String text1)
    : base(attribute1, attribute2, attribute3, text1)
{
    Console.WriteLine("Konstruktor von GrandChildClass mit allen Parametern: Sum = {0}", Sum);
}
```



Teste die Konstruktoren mit Parameter

```
Console.WriteLine("\nBasisklasse wird getestet:");
BaseClass bc = new BaseClass(2,3,4);
Console.WriteLine("\nKindklasse wird getestet:");
ChildClass cc = new ChildClass(5,6,7,"hallo");
Console.WriteLine("\nEnkerlklassen wird getestet:");
GrandChildClass gcc = new GrandChildClass(8,9,10,"hallo");
```

Basisklasse wird getestet:

Konstruktor von BaseClass mit Parameter: 2, 3, 4

Kindklasse wird getestet:

Konstruktor von BaseClass mit Parameter: 5, 6, 7

Konstruktor von ChildClass mit Parametern: 5, 6, 7, hallo

Enkerlklassen wird getestet:

Konstruktor von BaseClass mit Parameter: 8, 9, 10

Konstruktor von ChildClass mit Parametern: 8, 9, 10, hallo

Konstruktor von GrandChildClass mit allen Parametern: Sum = 27

Drücken Sie eine beliebige Taste . . .

Definiere die Begriffe

Methodenüberladen

Methodenüberschreiben

Methodenverdecken



Welche Schlüsselwörter werden im Falle verwendet?



Begriff – Überladung

- **Konstruktorenüberladung**
- Konstruktoren einer Klasse mit unterschiedlichen Parametern werden überladen.
- **Methodenüberladung**
- Zwei gleichnamige Methoden einer Klasse unterscheiden sich nur durch ihre Parameterliste



Begriff - Methodenüberschreiben

- **Überschreiben einer virtuellen Methode**
 - Soll eine Subklasse polymorphes Verhalten zeigen, ist in der Basisklasse der Methode `virtual` anzugeben.
 - In der Subklasse ist die geerbte Methode neu zu implementieren und mit dem Modifizierer `override` zu signieren.
- Dies erzeugt **dynamische Bindung**, der Datentyp wird zur Laufzeit bestimmt und die Methode des Objekts und nicht der statischen Klasse wird verwendet.
- Dies wird auch **polymorphes Verhalten** genannt (Polymorphie).



Begriff - Methodenüberdecken

- Nichtpolymorphes Verhalten
Überdecken einer Methode
 - Soll eine Subklasse eine geerbte Methode überdecken
 - wird in der Subklasse die überdeckte Methode mit **new** neu implementiert
- Es entsteht **statische Bindung**,
eine Referenz der Basisklasse, zeigend auf ein Objekt der
Subklasse, nutzt die Methode der Basisklasse
- Der Datentyp zur Compilezeit wird verwendet um die dazugehörige
Methode laut deklarierten Datentyp wird verwendet.



Aufgabe: Überladen vs Überdecken

- Erstelle in der Basisklasse 3 Methoden, alle public mit einer Ausgabe in der Console: Klasse + Methodename
- Überschreibe in der abgeleiteten Klasse die 2te Methode - erstelle eine neue Ausgabe
 - Füge in der Basisklasse `virtual` und in der abgeleiteten Klasse `override` ein
- Überschreibe in der abgeleiteten Klasse die 3te Methode – erstelle eine neue Ausgabe.
 - Es erscheint ein Hinweis – füge das Schlüsselwort „`new`“ ein



BaseClass – 3 Methoden

- Methode 2 mit `virtual` signieren:

```
public class BaseClass
{
    public void method1 ()
    {
        Console.WriteLine("Methode eins" );
    }

    public virtual void method2()
    {
        Console.WriteLine("Methode zwei" );
    }

    public void method3()
    {
        Console.WriteLine("Methode drei: BaseClass");
    }
}
```



Childclass

- Überschreiben und überdecken der Methoden:

```
//Methodenüberschreiben mit override -> polymorph
public override void method2()
```

```
{
```

```
    Console.WriteLine("Methode zwei: ChildClass");
```

```
}
```

```
//Methodenüberdecken mit new -> nicht polymorph
```

```
public new void method2 method3()
```

```
{
```

```
    Console.WriteLine("Methode drei: ChildClass");
```

```
}
```



Testen der Polymorphie

```
BaseClass bc = new BaseClass();
ChildClass cc = new ChildClass();
BaseClass bc2 = new ChildClass();

Console.WriteLine("Virtual & Override für BC, CC und Statisch BC Dynamisch CC");
bc.method2();
cc.method2();
bc2.method2();

Console.WriteLine("Methode 3 mit new für BC, CC und Statisch BC Dynamisch CC");
bc.method3();
cc.method3();
bc2.method3();
```

```
Virtual & Override für BC, CC und Statisch BC Dynamisch CC
Methode zwei
Methode zwei: ChildClass
Methode zwei: ChildClass
Methode 3 mit new für BC, CC und Statisch BC Dynamisch CC
Methode drei: BaseClass
Methode drei: ChildClass
Methode drei: BaseClass
Drücken Sie eine beliebige Taste . . .
```



Polymorphismus - Konsequenz

- Basis für dynamische Bindung:
 - Behandlung von dynamischen Objekten abgeleiteter (neuer) Klassen mit bereits übersetzten Programmteilen ist möglich, da diese Programmteile wenn sie mit Objekten der Klasse X arbeiten können, auch mit Objekten einer von X abgeleiteten Klasse arbeiten können
- Sammlung von Objekten unterschiedlicher Klassen in einem Behälter (Array / List)



Aufgabe: Array of Shapes

- Erstelle eine Klasse Shape – mit der Methode Anzeigen() mit einer Konsolenausgabe.
- Erzeuge abgeleitete Klassen: Rechteck, Kreis, Dreieck, Stern, ...
- Erstelle ein Array of Shapes,
befülle diese mit unterschiedlichen konkreten Formen.
- Rufe die Methode Anzeigen()
beim Durchlaufen des Arrays auf.
- Erläutere deine Beobachtungen...

Luftfahrzeuge

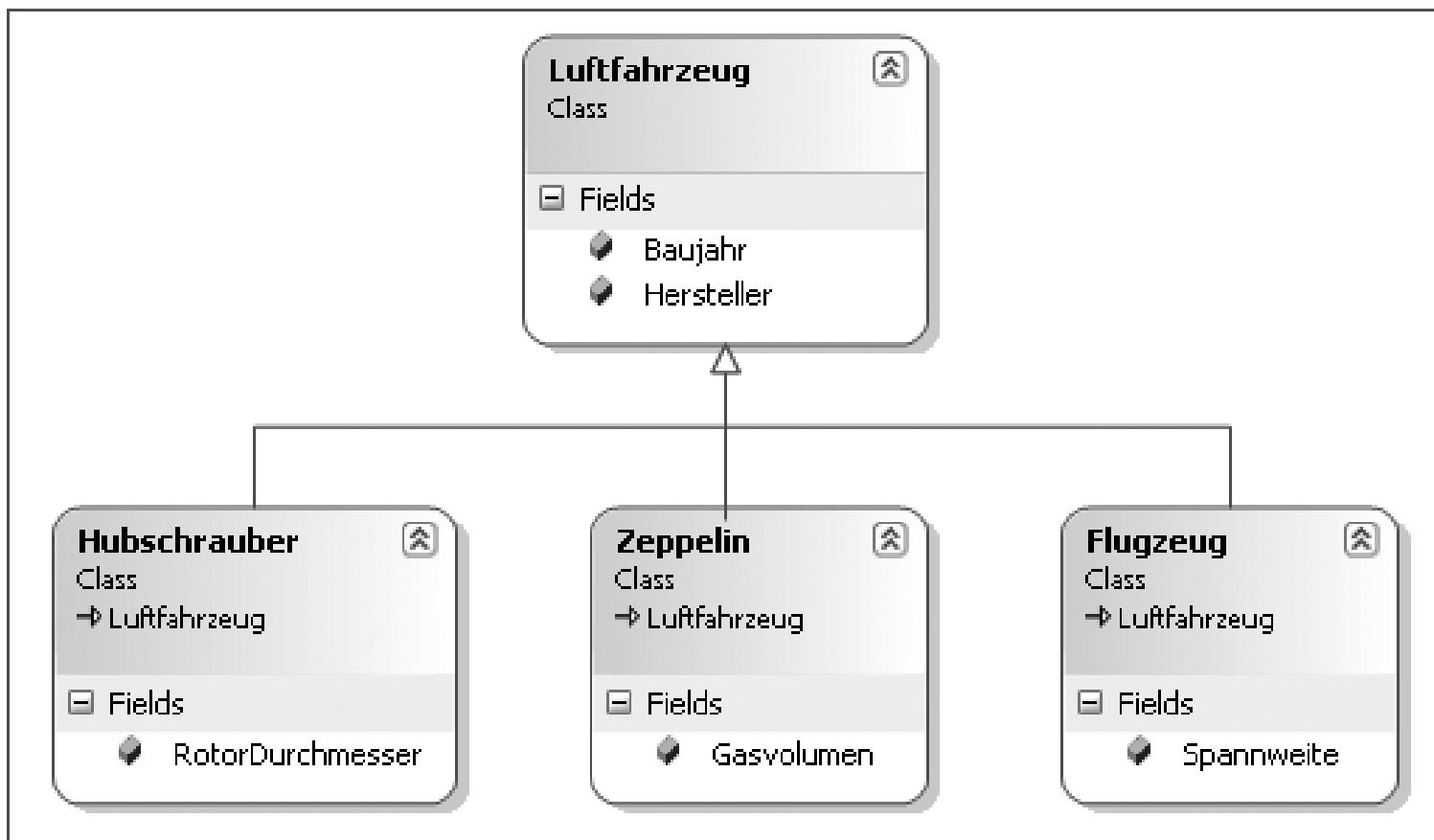


Erstellen von Klassen
Realisieren von Vererbung
Datenkapselung - Properties
Methodenüberschreiben -
Polymorphie
Überladen von Konstruktoren -
Konstruktorkette

SEW2



Klassendiagramm Beispiel Luftfahrzeug





Angabe 1: Luftfahrzeug

Erstelle eine Klasse Luftfahrzeug
mit den öffentlichen Attributen Hersteller
(String) und Baujahr (int).

**Leite die Klasse Flugzeug von Luftfahrzeug
ab**
und füge das öffentliche Attribut
Spannweite (double) hinzu.

**Leite die Klasse Hubschrauber von
Luftfahrzeug ab**
und füge das öffentliche Attribut
RotorDurchmesser (double) hinzu.

**Leite die Klasse Zeppelin von Luftfahrzeug
ab**
und füge das öffentliche Attribut
Gasvolumen (double) hinzu.



Luftfahrzeug – Aufgabe (1)

- Erstelle eine Klasse Luftfahrzeug mit den öffentlichen Attributen Hersteller (String) und Baujahr (int).
 - Leite die Klasse Flugzeug von Luftfahrzeug ab und füge das öffentliche Attribut Spannweite (double) hinzu.
 - Leite die Klasse Hubschrauber von Luftfahrzeug ab und füge das öffentliche Attribut RotorDurchmesser (double) hinzu.
 - Leite die Klasse Zeppelin von Luftfahrzeug ab und füge das öffentliche Attribut Gasvolumen (double) hinzu.



Bsp Luftfahrzeug

```
class Luftfahrzeug
{
    int ...baujahr;
    string ...hersteller;
}
class Flugzeug : Luftfahrzeug
{
    double ...spannweite;
}
class Hubschrauber : Luftfahrzeug
{
    double ...rotordurchmesser;
}
class Zeppelin : Luftfahrzeug
{
    double ...gasvolumen;
}
```

```
Luftfahrzeug myLuftfahrzeug=new Luftfahrzeug();
Flugzeug myFlugzeug=new Flugzeug();
Hubschrauber myHubschrauber=new Hubschrauber();
Zeppelin myZeppelin=new Zeppelin();
```



Fragen (1)

- Kann auf hersteller und baujahr zugegriffen werden in den abgeleiteten Klassen?
- Welchen Standardzugriffsmodifizierer haben die Klassenelemente wenn es nicht explizit angegeben wurde?
- Welchen Zugriffsmodifizierer müsste man verwenden um auf die Klassenelemente der Basisklasse in den abgeleiteten Klassen zugreifen zu können?
- Welche weiteren Möglichkeiten gibt es um auf private Klassenelemente von außen zugreifen zu können.



Angabe 2: Luftfahrzeug

Get und Set-Methoden

Erstelle für den hersteller der Klasse Luftfahrzeug Get und Set-Methoden.

Properties

Erstelle für das baujahr get und set Properties, überprüfe auf korrekte Jahreszahlen.

Erstelle für die Klassenelemente der abgeleiteten Klassen Properties mit nur Leserechte (Lösung pro Property ist ein Einzeiler)

Instanzen der Klasse

Erstelle jeweils eine Instanz der Klassen und setze passende Werte. Speichere nun die aktuelle Version nochmal als Kopie unter „VererbungA2“ bevor du weiterarbeitest.



Luftfahrzeug – Aufgabe (2)

- **Get und Set-Methoden**
 - Erstelle für den hersteller der Klasse Luftfahrzeug Get und Set-Methoden.
- **Properties**
 - Erstelle für das baujahr get und set Properties, überprüfe auf korrekte Jahreszahlen.
- **Properties**
 - Erstelle für die Klassenelemente der abgeleiteten Klassen Properties mit nur Leserechte
 - (Lösung pro Property ist ein Einzeiler)
- **Instanzen der Klasse**
 - Erstelle jeweils eine Instanz der Klassen und setze passende Werte. Speichere nun die aktuelle Version nochmal als Kopie unter „VererbungA2“ bevor du weiterarbeitest.



Fragen (2)

- Wie sieht der Methodenaufruf aus um einer Variable einen neuen Wert zu übergeben?
- Welchen Vorteil hat es dafür Methoden oder Properties zu verwenden?
- Wie kann man Properties schnell erstellen?
- Welchen Nachteil hat die schnelle Erstellung von Properties?
- Wie kann man auf Properties zugreifen?



Lösung (2)

```
public static void TestMethod() {  
    Luftfahrzeug myLuftfahrzeug = new Luftfahrzeug();  
    Flugzeug myFlugzeug = new Flugzeug();  
    Hubschrauber myHubschrauber = new Hubschrauber();  
    Zeppelin myZeppelin = new Zeppelin();  
    myLuftfahrzeug.SetHersteller("Hersteller");  
    myLuftfahrzeug.Baujahr = 2003;  
    myFlugzeug.SetHersteller("Hersteller");  
    myFlugzeug.Baujahr = 2005;  
    myHubschrauber.SetHersteller("Hersteller");  
    myHubschrauber.Baujahr = 2001;  
    myZeppelin.SetHersteller("Hersteller");  
    myZeppelin.Baujahr = 2010;  
}
```



Angabe 3: Luftfahrzeug

Konstruktoren in Basisklasse

Erstelle für die Klasse Luftfahrzeug einen Konstruktor und schreibe in diesen Konstruktor eine Console.WriteLine(„“) Mitteilung die angibt in welchem Konstruktor man sich gerade befindet.

Konstruktor in abgeleiteten Klassen

Erstelle für die von der Klasse Luftfahrzeug abgeleiteten Klassen auch jeweils einen Konstruktor und schreibe in diesen Konstruktor eine Console.WriteLine(„“) Mitteilung die angibt in welchem Konstruktor man sich gerade befindet.



Aufgabe (3)

- **Konstruktoren in Basisklasse**
 - Erstelle für die Klasse Luftfahrzeug einen Konstruktor und schreibe in diesen Konstruktor eine Console.WriteLine(„“) Mitteilung die angibt in welchem Konstruktor man sich gerade befindet.
- **Konstruktor in abgeleiteten Klassen**
 - Erstelle für die von der Klasse Luftfahrzeug abgeleiteten Klassen auch jeweils einen Konstruktor und schreibe in diesen Konstruktor eine Console.WriteLine(„“) Mitteilung die angibt in welchem Konstruktor man sich gerade befindet.



Fragen (3)

- Erzeuge jeweils von allen Klassen eine Instanz
 - Wie oft wurde der Konstruktor der Basisklasse aufgerufen?
-
- Erzeuge jeweils von allen Klassen eine Instanz
 - Wann wird welcher Konstruktor aufgerufen?

Lösung (3)

```
Luftfahrzeug myLuftfahrzeug = new Luftfahrzeug();
Flugzeug myFlugzeug = new Flugzeug();
Hubschrauber myHubschrauber = new Hubschrauber();
Zeppelin myZeppelin = new Zeppelin();
Luftfahrzeug myLuftfahrzeug2 = new Luftfahrzeug(2002, "Hersteller");
Flugzeug myFlugzeug2 = new Flugzeug(100, 2004, "Hersteller");
Hubschrauber myHubschrauber2 = new Hubschrauber(16.5, 2011, "Hersteller");
Zeppelin myZeppelin2 = new Zeppelin(700.0, 2010, "Hersteller");
myLuftfahrzeug.Hersteller = "Hersteller";
myLuftfahrzeug.Baujahr = 2003;
myFlugzeug.Hersteller = "Hersteller";
myFlugzeug.Baujahr = 2005;
myHubschrauber.Hersteller = "Hersteller";
myHubschrauber.Baujahr = 2001;
myZeppelin.Hersteller = "Hersteller";
myZeppelin.Baujahr = 2010;
Console.ReadKey();
```



Angabe 3 Erweiterung: Luftfahrzeug

Mehr Konstruktoren

Erstelle für die Klasse Luftfahrzeug und deren davon abgeleiteten Klassen jeweils einen Konstruktor mit einem, zwei und in den abgeleiteten Klassen mit 3 Parametern. Schreibe in diesen Konstruktor eine `Console.WriteLine("")` Mitteilung die angibt in welchem Konstruktor man sich gerade befindet.

Konstruktorkette

Nutze hierbei mit „`: this(hersteller)`“ oder „`:base(hersteller)`“ jeweils den passenden Konstruktor um nicht alle Werte in jedem Konstruktor nochmals zu initialisieren.



Erweiterung Aufgabe (3)

- **Mehr Konstruktoren**

- Erstelle für die Klasse Luftfahrzeug und deren davon abgeleiteten Klassen jeweils einen Konstruktor mit einem, zwei und in den abgeleiteten Klassen mit 3 Parametern. Schreibe in diesen Konstruktor eine Console.WriteLine(„“) Mitteilung die angibt in welchem Konstruktor man sich gerade befindet.

- **Konstruktorkette**

- Nutze hierbei mit „: this(hersteller)“ oder „:base(hersteller)“ jeweils den passenden Konstruktor um nicht alle Werte in jedem Konstruktor nochmals zu initialisieren.



Konstruktorverkettung

http://openbook.galileocomputing.de/visual_csharp/visual_csharp_03_008.htm#mj0724422b9c46345b32bb84ee346fab0d

1. Erzeuge jeweils von allen Klassen eine Instanz
2. Wann wird welcher Konstruktor aufgerufen?



Angabe 4: Luftfahrzeug

Methode Fliegen

Implementiere die Methode fliegen in der Klasse Luftfahrzeug mit der Ausgabe „Das Luftfahrzeug fliegt.“ Lege Objekte der Klassen an und betrachte das Ergebnis.

Schreibe nun in den von Luftfahrzeug abgeleiteten Klassen die Methode fliegen und verändere die Ausgabe in der Methode passend. zB: „Das Flugzeug fliegt..“

Array von Luftfahrzeugen die Fliegen

Erzeuge nun ein Array mit Luftfahrzeugen und befülle es mit den unterschiedlichen Objekten gemischt aus Luftfahrzeugen und davon abgeleiteten Klassen.

Verwende das Schlüsselwort new zum Überdecken von gleichlautenden Methoden.

Erstelle eine Methode „Ausgabe“ die ein Array von Luftfahrzeugen übergeben bekommt und jeweils vom Objekt der Klasse die Methode fliegen aufruft. Arbeitet mit einer foreach-Schleife.



Aufgabe (4)

- **Methode Fliegen in der Basisklasse**
 - Implementiere die Methode fliegen in der Klasse Luftfahrzeug mit der Ausgabe „Das Luftfahrzeug fliegt.“ Lege Objekte der Klassen an und betrachte das Ergebnis.
- **Und in allen Abgeleiteten Klassen:**
 - Schreibe nun in den von Luftfahrzeug abgeleiteten Klassen die Methode fliegen und verändere die Ausgabe in der Methode passend. zB: „Das Flugzeug fliegt.“



Aufgabe (4)

- **Array mit Luftfahrzeugen**
 - Erzeuge nun ein Array mit Luftfahrzeugen und befülle es mit den unterschiedlichen Objekten gemischt aus Luftfahrzeugen und davon abgeleiteten Klassen.
- **Überdecken**
 - Verwende das Schlüsselwort new zum Überdecken von gleichlautenden Methoden.



Fragen (4)

- Welchen Hinweis gibt der Compiler aus?
- Was bedeutet dieser Hinweis?



Antworten (4)

Beschreibung	
⚠ 1	"ClassLibrary1.Flugzeug.fliegen()" blendet den vererbten Member "ClassLibrary1.Luftfahrzeug.fliegen()" aus. Verwenden Sie das new-Schlüsselwort, wenn das Ausblenden vorgesehen war.
⚠ 2	"ClassLibrary1.Hubschrauber.fliegen()" blendet den vererbten Member "ClassLibrary1.Luftfahrzeug.fliegen()" aus. Verwenden Sie das new-Schlüsselwort, wenn das Ausblenden vorgesehen war.
⚠ 3	"ClassLibrary1.Zeppelin.fliegen()" blendet den vererbten Member "ClassLibrary1.Luftfahrzeug.fliegen()" aus. Verwenden Sie das new-Schlüsselwort, wenn das Ausblenden vorgesehen war.

- Die Methode Fliegen von der Klasse Flugzeug blendet die Methode Fliegen von der Klasse Luftfahrzeug aus. Sollte das Ausblenden vorgesehen sein, dann new verwenden.



Lese: Methoden in einer abgeleiteten Klasse

http://openbook.galileocomputing.de/visual_csharp/visual_csharp_03_011.htm#mj00cf0d329dfc1a409f6737d6cffbdecf



Aufgabe (4)

- **Array durchlaufen und Objekte fliegen lassen**
 - Erstelle eine Methode „Ausgabe“ die ein Array von Luftfahrzeugen übergeben bekommt und jeweils vom Objekt der Klasse die Methode fliegen aufruft. Arbeitet mit einer foreach-Schleife.

```

static void Main(string[] args)
{
    Luftfahrzeug myLuftfahrzeug = new Luftfahrzeug();
    Flugzeug myFlugzeug = new Flugzeug();
    Hubschrauber myHubschrauber = new Hubschrauber();
    Zeppelin myZeppelin = new Zeppelin();
    Luftfahrzeug myLuftfahrzeug2 = new Luftfahrzeug(2002, "Hersteller");
    Flugzeug myFlugzeug2 = new Flugzeug(100, 2004, "Hersteller");
    Hubschrauber myHubschrauber2 = new Hubschrauber(16.5, 2011, "Hersteller");
    Zeppelin myZeppelin2 = new Zeppelin(700.0, 2010, "Hersteller");
    Luftfahrzeug[] arr = new Luftfahrzeug[4];
    arr[0] = new Luftfahrzeug();
    arr[1] = new Flugzeug();
    arr[2] = new Hubschrauber();
    arr[3] = new Zeppelin();
    myLuftfahrzeug.Hersteller = "Hersteller";
    myLuftfahrzeug.Baujahr = 2003;
    myFlugzeug.Hersteller = "Hersteller";
    myFlugzeug.Baujahr = 2005;
    myHubschrauber.Hersteller = "Hersteller";
    myHubschrauber.Baujahr = 2001;
    myZeppelin.Hersteller = "Hersteller";
    myZeppelin.Baujahr = 2010;
    Ausgabe(arr);
    Console.ReadKey();
}
static void Ausgabe(Luftfahrzeug[] arr)
{
    foreach (Luftfahrzeug l in arr)
    {
        l.Fliegen(); ;
    }
}

```

Lösung (4)



Ausgabe:

- a. Das Luftfahrzeug fliegt
- b. Das Luftfahrzeug fliegt
- c. Das Luftfahrzeug fliegt
- d. Das Luftfahrzeug fliegt



Lösung (4)

Betrachte das Ergebnis.

Welches Ergebnis würdest du dir wünschen?

- **Ausgabe:**

- Das Luftfahrzeug fliegt
- Das Luftfahrzeug fliegt
- Das Luftfahrzeug fliegt
- Das Luftfahrzeug fliegt

- **Ausgabe Wunsch:**

- Das Luftfahrzeug fliegt
 - Das Flugzeug fliegt
 - Der Hubschrauber fliegt
 - Der Zeppelin fliegt
-



Fragen (4)

- Was wird in der Console ausgegeben?
- Was passiert wenn wir das Schlüsselwort new wieder entfernen?
- Welches Ergebnis würdest du dir bei der Ausgabe wünschen?



Angabe 5: Luftfahrzeug

Überschreiben

Füge das Schlüsselwort `virtual` in der Basisklasse vor den Rückgabewert der Methode `Fliegen()` und in den abgeleiteten Klassen das Schlüsselwort `override` vor den Rückgabewert der Methode `Fliegen()`.

Rufe nochmals die Methode `Ausgabe` auf und betrachte das Ergebnis.



Aufgabe (5)

- **Überschreiben**

- Füge das Schlüsselwort `virtual` in der Basisklasse vor den Rückgabewert der Methode `Fliegen()` und in den abgeleiteten Klassen das Schlüsselwort `override` vor den Rückgabewert der Methode `Fliegen()`.
- Rufe nochmals die Methode `Ausgabe` auf und betrachte das Ergebnis.



Fragen (5)

- Entspricht das Ergebnis den oben beschriebenen Vorstellungen?
- Was passiert wenn man die Schlüsselwörter virtual und override benutzt?
- Was ist eine statische Bindung?
- Was ist eine dynamische Bindung?



Antworten (5)

- Ja, Ergebnis entspricht den Vorstellungen
- Wenn bei der Basisklasse „virtual“ kann sie von einer UnterkLASSE mit dem Schlüsselwort „override“ überschrieben werden.
- Polymorphie arbeitet mit dynamischer Bindung. Der Aufrufcode wird nicht zur Kompilierzeit erzeugt, sondern erst zur Laufzeit der Anwendung, wenn die konkreten Typinformationen vorliegen, um die tatsächlich aufzurufende Methode zu bestimmen.
- Im Gegensatz dazu legt die statische Bindung die auszuführende Operation bereits zur Kompilierzeit fest.

```

static void Main(string[] args)
{
    Luftfahrzeug myLuftfahrzeug = new Luftfahrzeug();
    Flugzeug myFlugzeug = new Flugzeug();
    Hubschrauber myHubschrauber = new Hubschrauber();
    Zeppelin myZeppelin = new Zeppelin();
    Luftfahrzeug myLuftfahrzeug2 = new Luftfahrzeug(2002, "Hersteller");
    Flugzeug myFlugzeug2 = new Flugzeug(100, 2004, "Hersteller");
    Hubschrauber myHubschrauber2 = new Hubschrauber(16.5, 2011, "Hersteller");
    Zeppelin myZeppelin2 = new Zeppelin(700.0, 2010, "Hersteller");
    Luftfahrzeug[] arr = new Luftfahrzeug[4];
    arr[0] = new Luftfahrzeug();
    arr[1] = new Flugzeug();
    arr[2] = new Hubschrauber();
    arr[3] = new Zeppelin();
    myLuftfahrzeug.Hersteller = "Hersteller";
    myLuftfahrzeug.Baujahr = 2003;
    myFlugzeug.Hersteller = "Hersteller";
    myFlugzeug.Baujahr = 2005;
    myHubschrauber.Hersteller = "Hersteller";
    myHubschrauber.Baujahr = 2001;
    myZeppelin.Hersteller = "Hersteller";
    myZeppelin.Baujahr = 2010;
    Ausgabe(arr);
    Console.ReadKey();
}
static void Ausgabe(Luftfahrzeug[] arr)
{
    foreach (Luftfahrzeug l in arr)
    {
        l.Fliegen(); ;
    }
}

```

Lösung (5)



```

class Luftfahrzeug
{
    ...
    public virtual void Fliegen()
    {
        Console.WriteLine("Das Luftfahrzeug fliegt");
    }
}
class Flugzeug : Luftfahrzeug
{
    ...
    public override void Fliegen()
    {
        Console.WriteLine("Das Flugzeug fliegt");
    }
}

```

Luftfahrzeug

```
public class Luftfahrzeug  
{  
    protected string hersteller;  
    protected int baujahr;
```



```
public class Flugzeug : Luftfahrzeug  
{  
    protected double spannweite;
```

Flugzeug



CoolClips.com