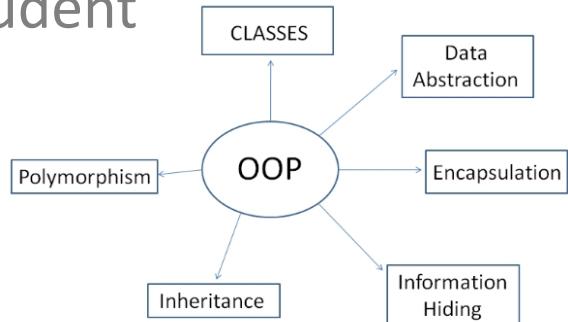




Objekt Orientierte Programmierung

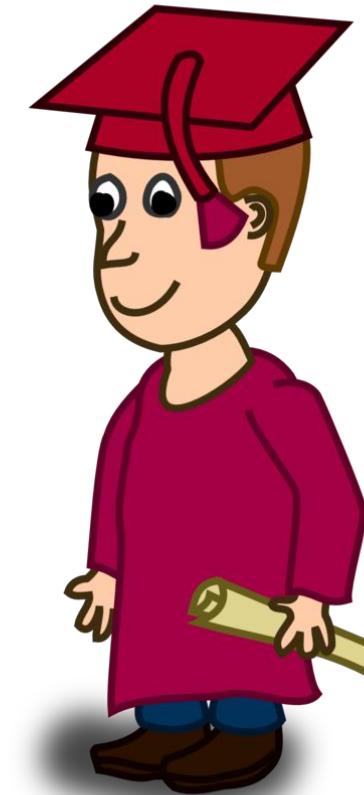
Person & Student
Animal



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 - OopFirstProgram' 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

Selected Option: Hinzufügen > Klasse

Neues Element hinzufügen - OopFirstProgram Dialog:

Sortieren nach: Standard

Typ: Visual C#-Elemente
Eine leere Klassendefinition.

Kategorie	Element	Typ
Installiert	Allgemein	Visual C#-Elemente
Visual C#-Elemente	Code	Visual C#-Elemente
	Daten	Visual C#-Elemente
	Windows Forms	Visual C#-Elemente
	WPF	Visual C#-Elemente
	.NET Core	Visual C#-Elemente
	SQL Server	Visual C#-Elemente
Online		
	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
	Bitmapdatei	Visual C#-Elemente
	Codedatei	Visual C#-Elemente
	Cursordatei	Visual C#-Elemente

Name: SecondClass

Hinzufügen Abbrechen



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();  
}
```

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

```
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}");  
    }  
}
```



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 variable 's'. 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;

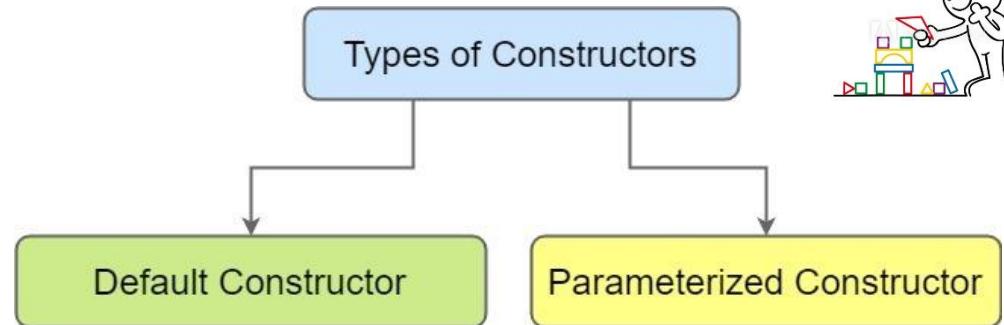
    public void PresentMyself() {
        Console.WriteLine($"{name} {age} {schoolclass}");
    }
    public void Lerning() {
        Console.WriteLine($"{name} learns {favorite}");
    }

    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
```



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");
```

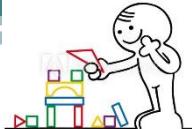
Konstruktor mit Parameter



```
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;
    }
}
```

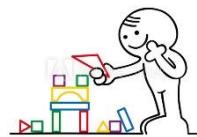
```
public static void TestStudentKonstruktor() {
    Student s1 = new Student();
    Student s2 = new Student("Susi");
    Console.WriteLine(s2.name);
    Student s3 = new Student("Susi", 14, "1B HIT", "SEW");
    s3.PresentMyself();

    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;

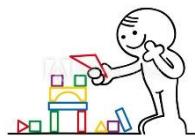
    //Konstruktor (Standardkonstruktor)
    public Student() { }

    //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, "2BHT", "SEW");
    }
    foreach (var item in students) {
        item.PresentMyself();
    }
}
```

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

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



Überladen

- Mehrere Konstruktoren innerhalb der Klasse
 - mit unterschiedlicher Parameterliste

```
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;  
    }  
}
```

Standardkonstruktor

- Konstruktor der verfügbar ist, wenn kein Konstruktor selbst geschrieben wurde
 - ohne Parameter
 - automatisch in jeder Klasse verfügbar
 - muss ersetzt werden durch selbst geschriebenen Konstruktor ohne Parameter, wenn es mehrere Konstruktoren gibt

```
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;  
    }  
}
```



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

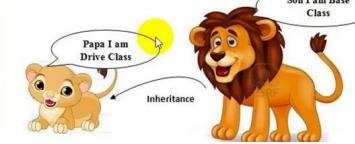
```
class Person {
    //Attribut Name
    public string name;

    //Attribut Alter
    public int age;

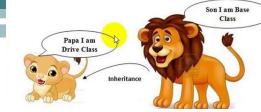
    //Methode Begrüßen
    public void Greet() {
        Console.WriteLine($"Begrüße {name} - {age}");
    }
}
```

```
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



Object ist die Basisklasse

- Object ist die Basisklasse aller Klassen
- Object beinhaltet 4 Methoden,
welche für alle Instanzen verfügbar sind
 - 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



Vererbung: Student erbt von Person

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

Person erbt
automatisch von Object

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

```
p.  
St ...  
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 = ...;
s.name = "Martina";
s.
```

```
}  
s. ⚭ Equals  
s. ⚭ GetHashCode  
s. ⚭ GetType  
s. ⚭ name (Feld) string Person.name  
s. ⚭ 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} ");
    }
}
```

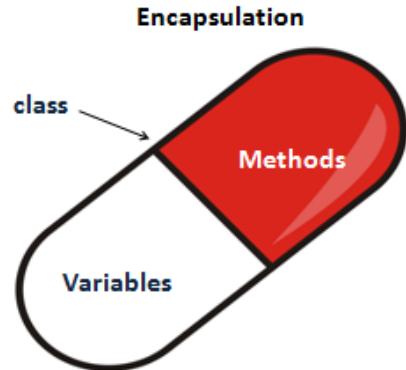
```
static void Main(string[] args) {
    Person p = new Person();
    p.name = "Susi";
    Student s = new Student();
    s.name = "Martina";
    Console.WriteLine(p.name);
    Console.WriteLine(s.name);
    p.ThatsMe();
    //PresentStudent ruft ThatsMe auf
    s.PresentStudent();
    s.
}
```

favorite
GetHashCode
GetType
name
PresentStudent
schoolclass
ThatsMe
ToString

void Student.PresentStudent()

Age und IsMale sind in der Main nicht verfügbar



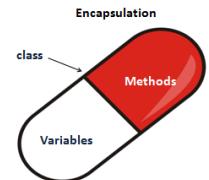


Get- Set-Methoden

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 {

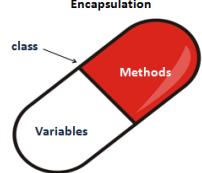
    protected string schoolclass;
    protected string favorite;
```

```
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
```



```
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() ...
}

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

```

Vererbung über 3 Ebenen:

```

Person
  |
Student
  |
SpecialStudent

```

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");
    }
}

```

Student & Grades 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;
}

```

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);  
}
```

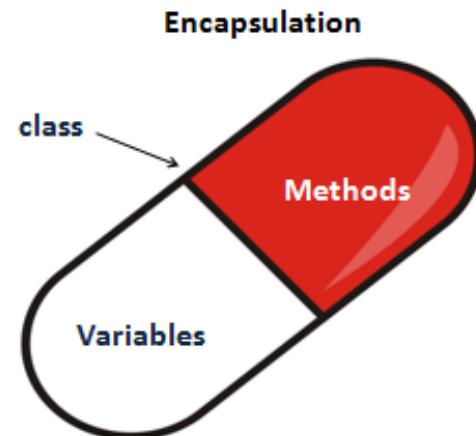
Das Array für die Noten
in Student wird immer
angelegt.

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

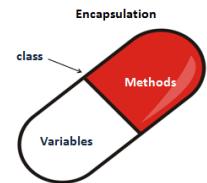
Datenkapselung wird in C# mit Properties statt Get- und Set-Methoden realisiert, diese Technologie ist in anderen Sprachen nicht verfügbar.



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.

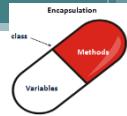
Kurzschreibweise:

- 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; } }
```

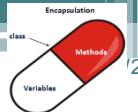
Properties in der Langschreibweise
benötigen ein Attribut und bieten
eine Wertüberprüfung im **set**

```
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);
}
```

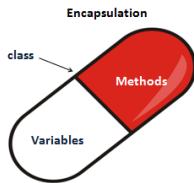
Properties können wie public Attribute genutzt werden

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

Objekt Initialisierer

Konstruktor 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 instanziert – 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.

```
public object this[int index]  
{  
    get { /* return the specified index here */ }  
    set { /* set the specified index to value here */ }  
}
```

Indexer einer Klasse

Ermöglicht das Zugreifen auf eine Collection innerhalb einer Klasse, gleich komfortabel als wäre die Klasse eine Collection

Erstelle für die Klasse Student einen Indexer, der es ermöglicht auf die Noten eines Student zuzugreifen.

Zugriff auf ein Array (Collection)

- Wie greift man auf einen Wert eines Arrays zu, das sich innerhalb einer Klasse befindet?!
- Property auf das Array erstellen...

```
class Student : Person {  
    public string Schoolclass { get; set; }  
    public string Favorite { get; private set; }  
    protected SchoolGrade3[] grades;  
    public SchoolGrade3[] Grades {  
        get { return grades; }  
        set { grades = value; }  
    }  
}
```

```
//Property auf Grades  
s.Grades[3] = new SchoolGrade3() {  
    Subject = "D",  
    Description = "Deutsch",  
    Grade = 2  
};  
  
s.Grades[4] = new SchoolGrade3() {  
    Subject = "M",  
    Description = "Mathematik",  
    Grade = 2  
};  
Console.WriteLine(s.Grades[3]);  
Console.WriteLine(s.Grades[4]);
```

- Geht's noch einfacher?!

Indexer

- Zugriff auf ein Array in Student
- mit einem Indexer für Noten

```
class Student : Person {  
    public string Schoolclass { get; set; }  
    public string Favorite { get; private set; } //{{get;}  
    protected SchoolGrade[] grades;  
  
    //Indexer  
    public SchoolGrade this[int index] {  
        get { return grades[index]; }  
        set { grades[index] = value; }  
    }  
}
```

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

Student mit Indexer

```

class Student : Person {
    public string Schoolclass { get; set; }
    public string Favorite { get; private set; } // {get;}
    protected SchoolGrade[] grades;

    //Indexer
    public SchoolGrade this[int index] {
        get { return grades[index]; }
        set { grades[index] = value; }
    }

    //Konstruktor
    public Student() {
        grades = new SchoolGrade[10];
        Favorite = "SEW";
    }

    //Darstellung von Objekten als Zeichenkette
    public override string ToString() {
        StringBuilder sb = new StringBuilder(base.ToString());
        sb.Append($" - {Schoolclass} - {Favorite}");
        sb.Append($" - Noten: \n");
        //nur den Teil des Arrays aus wo Noten gespeichert sind
        for(int i = 0; i < grades.Length && grades[i] != null; i++)
            sb.Append(grades[i].ToString());
        return sb.ToString();
    }
}

```

Mann - Kurt - 15 - 2BHT - SEW - Noten:
 Software Entwicklung(SEW): 1
 Medientechnik(MEDT): 2
 Netzwerktechnik(NWTK): 1

```

class SchoolGrade {
    public string Subject { get; set; }
    public string Description { get; set; }
    private int grade;
    public int Grade {
        get { return grade; }
        set { grade = (value > 0 && value < 6) ? value : 0; }
    }
    public override string ToString() {
        return String.Format("\t{1}({0}): {2} \n",
            Subject, Description, Grade);
    }
}

public static void TestIndexerStudent7() {
    Student s = new Student() {
        Name = "Kurt", Age = 15 };
    s.IsMale = true;
    s.Schoolclass = "2BHT";
    s[0] = new SchoolGrade() {
        Subject = "SEW",
        Description = "Software Entwicklung",
        Grade = 1
    };
    s[1] = new SchoolGrade() {
        Subject = "MEDT",
        Description = "Medientechnik",
        Grade = 2
    };
    s[2] = new SchoolGrade() {
        Subject = "NWTK",
        Description = "Netzwerktechnik",
        Grade = 1
    };
    Console.WriteLine(s);
}

static void Main(string[] args) {
    TestIndexerStudent7();
}

```

Syntax vom Indexer

- Spezielle Art von Properties
- Erlaubt einer Klasse benutzt zu werden wie ein Array

Syntax:

```
<datatype> array;  
public <datatype> this[ <parameter type> index] {  
    get { return array[index]; }  
    set { array[index] = value; }  
}
```

- Ergänzende Erklärungen & Beispiele:
 - <http://www.tutorialsteacher.com/csharp/csharp-indexer>

Property vs Indexer

Property auf Array

```
class Student : Person {
    public string Schoolclass { get; set; }
    public string Favorite { get; private set; }
    protected SchoolGrade3[] grades;
    public SchoolGrade3[] Grades {
        get { return grades; }
        set { grades = value; }
    }
}
```

```
Student s = new Student() {
    Name = "Kurt", Age = 15 };
s.IsMale = true;
s.Schoolclass = "2BHT";

//Property auf Grades
s.Grades[3] = new SchoolGrade3() {
    Subject = "D",
    Description = "Deutsch",
    Grade = 2
};
```

Indexer ist etwas komfortabler um auf ein Array innerhalb einer Klasse zuzugreifen.

Indexer

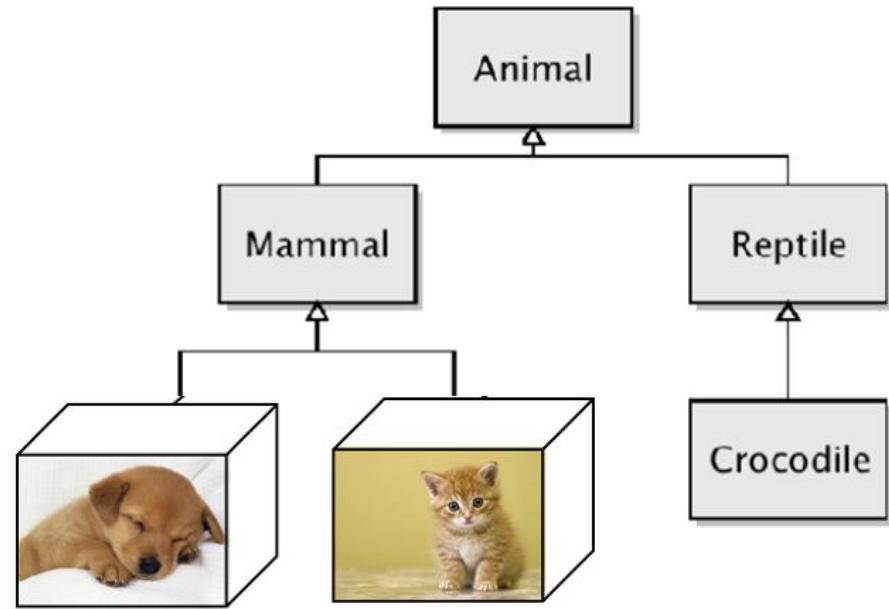
```
class Student : Person {
    public string Schoolclass { get; set; }
    public string Favorite { get; private set; } // {get; }
    protected SchoolGrade[] grades;

    //Indexer
    public SchoolGrade this[int index] {
        get { return grades[index]; }
        set { grades[index] = value; }
    }
}
```

```
Student s = new Student() {
    Name = "Kurt", Age = 15 };
s.IsMale = true;
s.Schoolclass = "2BHT";
s[0] = new SchoolGrade() {
    Subject = "SEW",
    Description = "Software Entwicklung",
    Grade = 1
};
```

Indexer Zusammengefasst:

- Indexer ermöglichen es Objekten, in ähnlicher Weise **wie Arrays indiziert** zu werden.
- Ein **get-Accessor** gibt einen Wert zurück.
Ein **set-Accessor** weist einen Wert zu.
- Das **this-Schlüsselwort** wird zum Definieren des Indexers verwendet.
- Das **value-Schlüsselwort** wird verwendet, um den Wert zu definieren, der vom set-Accessor zugewiesen wird.
- Indexer müssen **nicht durch einen Ganzzahlwert indiziert** werden.
Sie können entscheiden, wie Sie den spezifischen Suchmechanismus definieren möchten.



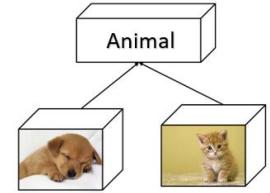
Abstrakte Klassen

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

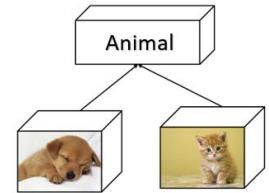


- Abstrakte Klassen beginnen mit A
 - Erstelle eine Klasse AAnimal
 - Mit einem Property Name und einer Methode Sound
-
- Animal kann nicht instanziert werden!

```
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}");
    }
}
  
```

```

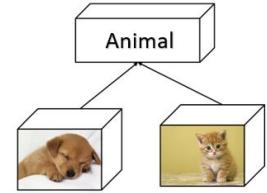
public class Elefant : AAnimal { }
public class Cat : AAnimal { }
  
```

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

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

- Erstelle eine Methode Move in AAnimal und in Elefant – es erscheint ein Hinweis...

```
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 bewegt sich ...  
Das Tier bewegt sich ...
```

Polymorphie

Vielgestaltigkeit

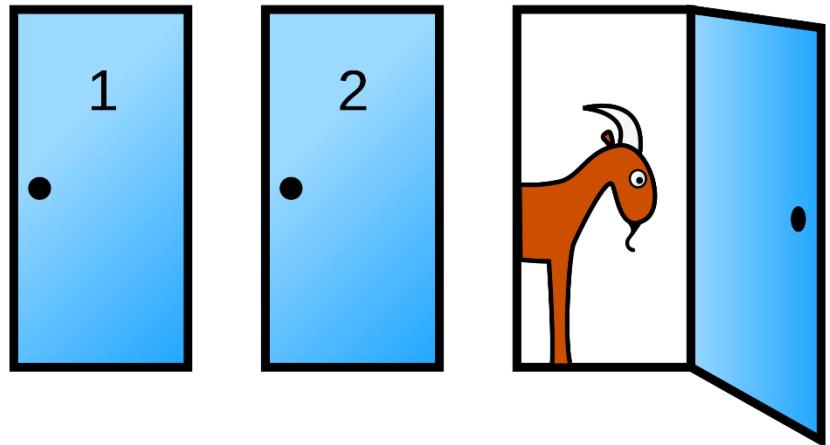
Polymorphes Verhalten

Dynamische Bindung

Typ des Objekts zur Laufzeit

Überschreiben einer Methode

Schlüsselwörter `virtual` & `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 ... ");
    }

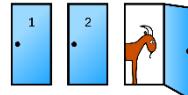
    public void Move() {
        Console.WriteLine($"Das Tier bewegt sich ... ");
    }
}

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();
    }
}

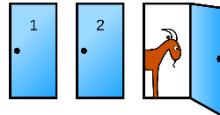
```

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

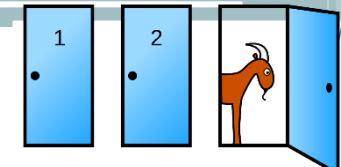


Überladen

- 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}");  
    }  
}
```

Methode überladen vs 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 ... ");
    }
}
```

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

```
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 Elefant
und Cat **überschrieben**
Schlüsselwörter virtual
und override