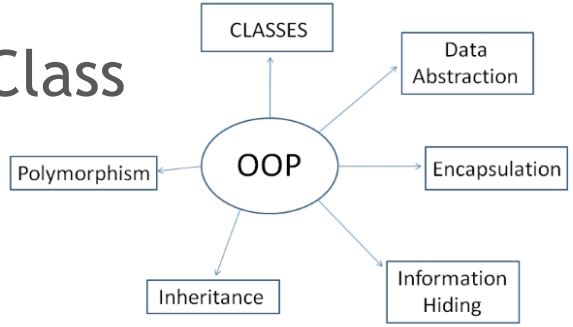
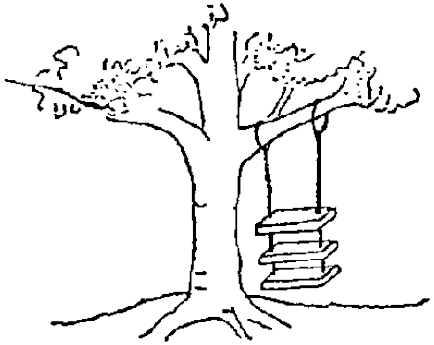
Objektorientierte Programmierung

BaseClass

ChildClass

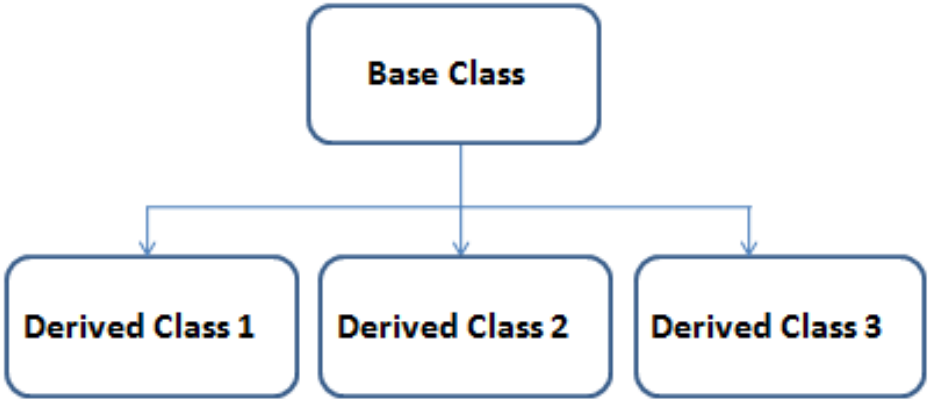
GrandChildClass

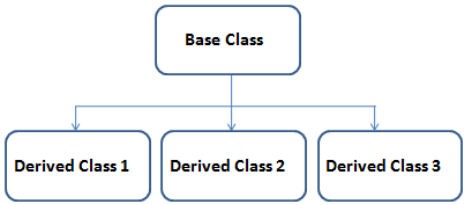




Überblick

- Vererbung
- Sichtbarkeiten
- Properties
- Konstruktor
- Konstruktorkette
- Polymorphie



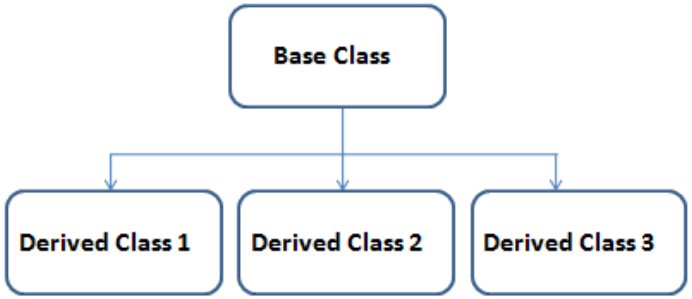


Aufgabenstellung Klassen

- Erstelle eine Basisklasse „Baseclass“
  - mit 3 Attributen „attribute_x“ vom Datentyp Integer
    verwende die Zugriffsmodifizierer
    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

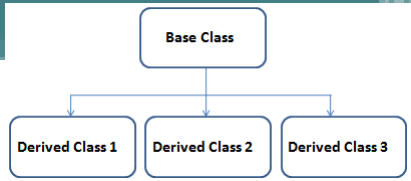
BaseClass &

```
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

Quick Launch (Ctrl+Q)

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

Start Debug Any CPU

Solution Explorer

Search Solution Explorer (Ctrl+ü)

Solution 'Inheritance' (1 project)
- Inheritance
  - Properties
  - References
  - App.config
  - BaseClass.cs
    - BaseClass
  - ChildClass.cs
  - Program.cs

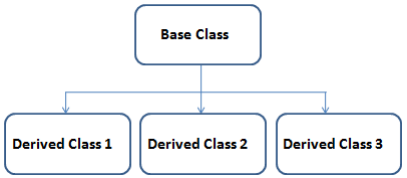
ChildClass.cs* BaseClass.cs Program.cs

Inheritance | Inheritance.ChildClass | ChildClass()

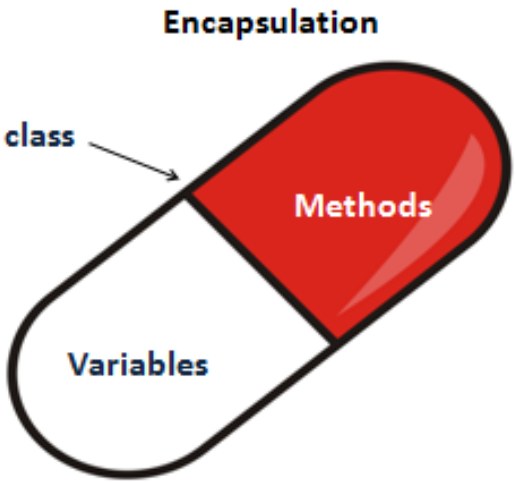
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

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

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

AssemblyLoadEventHandler
async
AsyncCallback
Attribute
attribute2
attribute3
AttributeTargets
AttributeUsageAttribute
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?
  -

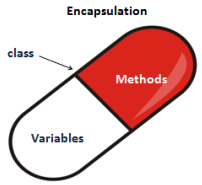


Properties

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.

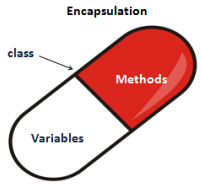
Ergänzende Informationen:
http://openbook.rheinwerk-verlag.de/visual_csharp_2010/visual_csharp_2010_03_005.htm#mjb3365746ed7c845fdc414b50306a0693



Aufgabenstellung Properties

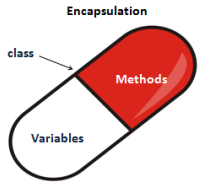
- Erstelle für alle 3 Attribute in der Basisklasse public Properties.
- Setzte beim ersten Attribut nur Leseberechtigung.
- Beim zweiten Attribut darf der Wert nur größer 0 sein.
- Beim dritten Attribut darf die Schnellschreibweise von Properties benutzt werden.

Properties



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

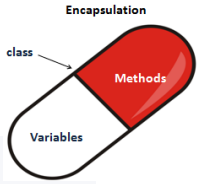
```
private int attribute1;
protected int attribute2;

//Nur Leseberechtigung
public int Attribute1 { get { return attribute1; } }

//Langschreibeweise mit Überprüfungsmöglichkeit
public int Attribute2
{
    get
    {
        return attribute2;
    }
    set
    {
        if (value > 0)
            attribute2 = value;
    }
}

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



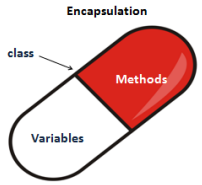
Properties – Lese & Schreibzugriff

- Was fällt bei Attribut3 auf?
  -
  -
- Nachteil der Schnellschreibeweise:
  -
  -

Langschreibweise vs Kurzschreibweise




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

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

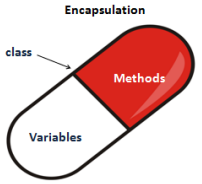


Frage

- 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



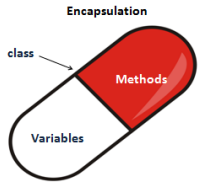
Testen der Properties

- 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

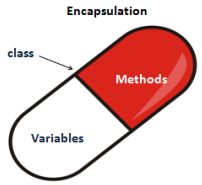
- Bei der Baseclass stehen alle Properties und das öffentliche Attribut „attribute3“ zur Verfügung



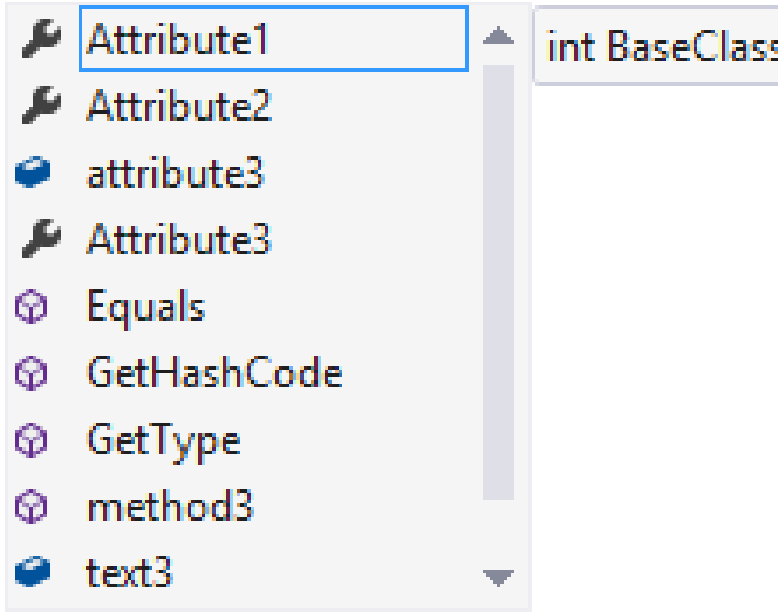

```
class TestClass
{
    public static void Main(String[] args)
    {
        BaseClass bc = new BaseClass();
        ChildClass cb = new ChildClass();

        bc.
    }

}
```

TEST ChildClass

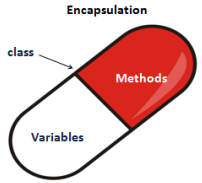
- Bei der Childclass stehen alle Properties der Basisklasse und das öffentliche Attribut der Basisklasse „attribute3“ zur Verfügung, sowie das öffentliche Attribut „text“ der Childclass



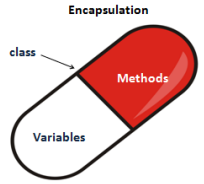

```
public static void Main(String[] args)
{
    BaseClass bc = new BaseClass();
    ChildClass cb = new ChildClass();

    cb.
}
```



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?

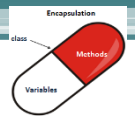
```
class TestClass
{
    public static void Main(String[] args)
    {
        BaseClass bc = new BaseClass();
        ChildClass cc = new ChildClass();

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

100 %

Error List

2 Errors | 0 Warnings | 0 Messages

	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



Möglicher Testfall

```
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?
  -

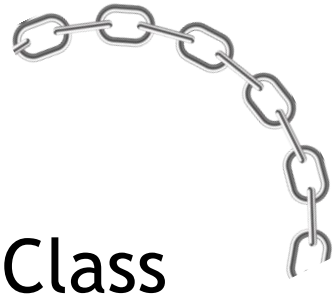
Fragen – Schlüsselwörter

- Mit welchem Schlüsselwort kann auf Methoden bzw Attribute der Basisklasse zugegriffen werden?
  -
- Mit welchem Schlüsselwort referenziert man auf die aktuelle Instanz einer Klasse?
  -

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;
            }
    }
}
```

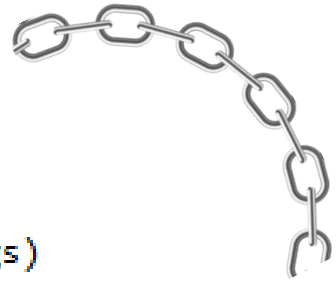


Konstruktor

- Erstelle in jeder Klasse: BaseClass, ChildClass und GrandChildClass einen Konstrkutor 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
```

- 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
```

Testfälle Konstruktorkette

- Nur GrandChildClass
  - Aufruf aller Konstrutoren 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("\nEnkerlklasse 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");
}
```

- Konstruktor parameterlos

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

- Konstruktor mit einem Parameter

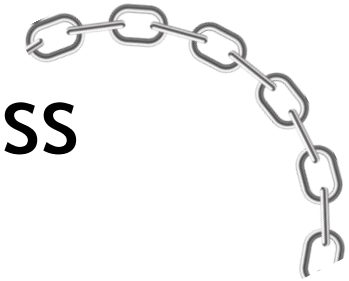
- Konstruktor für alle 3 Attribute

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

Konstruktoren von ChildClass

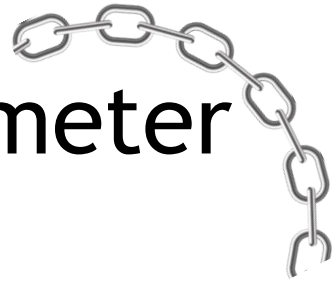
```
public class ChildClass : BaseClass
{
    private string text1;
    protected string text2;
    public string text3;

    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



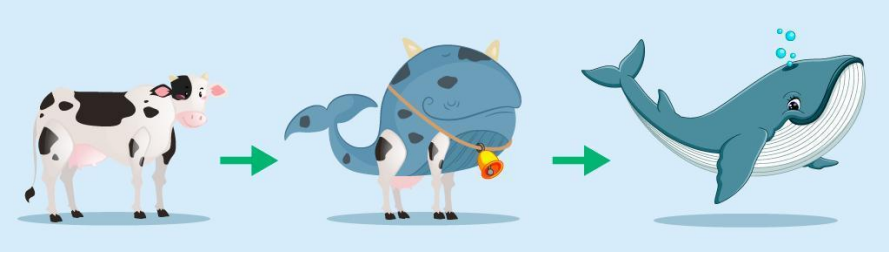
```
public GrandChildClass()
{
    Console.WriteLine("Konstruktor von GrandChildClass");
    sum = Sum;
}

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("\nEnkerlklasse 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

Enkerlklasse 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 + Methodenname

- **Überschreibe** in der abgeleiteten Klasse die 2te Methode - erstelle eine neue Ausgabe
  - Füge in der Basisklasse **virtual** und in der abgeleiteten Klasse override ein

- **Überdecke** in der abgeleiteten Klasse die 3te Methode – erstelle eine neue Ausgabe.
  - Es erscheint ein Hinweis: füge das Schlüsselwort „**new**“ in der abgeleiteten Klasse 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 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

- Teste nun BaseClass, ChildClass und GrandChildClass mit einen Array.

- Erstelle von BaseClass ein Array und instantiiere alle 3 unterschiedlichen Klassen.

- Rufe in einer Schleife Methode2 und Methode3 auf.
- Beschreibe die Wörter: Überladen, Überschreiben & Überdecken