



# Software Development

## Quellenangaben:

- Jürgen Kotz, München, Visual C# 2019, Carl Hanser Verlag München
- Michael Bonacina, C# Programmieren für Einsteiger (2. Aufl.), BMU Verlag
- <https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide>

# Kurseinheit 5

- Weiterführende Prinzipien der OOP
  - Sichtbarkeit von Klassen und Mitgliedern
  - Vererbung
  - Überladen (Overriding)
  - Überschreiben (Overloading)
  - Spezielle Klassen

# Weiterführende Prinzipien der OOP

# Sichtbarkeit von Klassen und Mitgliedern

- Kapselung von Daten
- Schutz vor Manipulation
- Klar definierte Schnittstellen
- Erhöht die Übersichtlichkeit des Programms

# Sichtbarkeit Klassen

Modifizierer	Sichtbarkeit
<code>public</code>	Der Zugriff ist nicht beschränkt. Auch von anderen Assemblierungen aus können Objekte der Klasse erstellt werden
<code>internal</code>	Nur innerhalb des aktuellen Projekts. Ausserhalb des Projekts ist kein Zugriff auf diese Klasse möglich. Gilt als Standard, falls kein Modifizier vorangestellt wird.
<code>private</code>	Nur innerhalb einer anderen Klasse

# Sichtbarkeit Klassenmit- glieder

Modifizierer	Sichtbarkeit
<b>public</b>	Der Zugriff ist nicht beschränkt
<b>protected</b>	Der Zugriff ist auf die enthaltende Klasse oder auf Typen beschränkt, die von der enthaltenden Klasse abgeleitet sind
<b>internal</b>	Der Zugriff ist auf das aktuelle Projekt beschränkt
<b>internal protected</b>	Der Zugriff ist auf das aktuelle Projekt oder auf Typen beschränkt, die von der enthaltenden Klasse abgeleitet sind
<b>private</b>	Der Zugriff ist auf die enthaltende Klasse beschränkt. Gilt als Standard, falls kein Modifizier vorangestellt wird.
<b>private protected</b>	Der Zugriff ist auf die enthaltende Klasse oder auf Typen beschränkt, die von der enthaltenden Klasse innerhalb des aktuellen Projektes abgeleitet sind. Verfügbar seit C# 7.2

# Vererbung

- Klasse (Subklasse) von anderen Klasse (Basisklasse) ableiten
- Subklasse erhält automatisch alle Methoden und Attribute
- Weitere Spezifizierung in Subklasse möglich



# Vererbung

## Schreibweise

- Die abgeleitete Klasse hängt den Namen der Basisklasse mit einem Doppelpunkt an.

C#

```
class AbgeleiteteKlasse: BasisKlasse  
{  
    ...  
}
```

# Vererbung

## Beispiel

Die Basisklasse hat zum Beispiel ein Feld und eine Methode


C #

```
class BasisKlasse
{
    protected int feld = 2;
    protected int Methode1(int a)
    {
        return a + feld;
    }
}
```

Die abgeleitete Klasse "erbt" alle Felder und Methoden und kann diese benutzen

C #

```
class AbgeleiteteKlasse: BasisKlasse
{
    public float Methode2(float a)
    {
        return a + feld + Methode1(5);
    }
}
```



# Vererbung Felder

## Beispiel

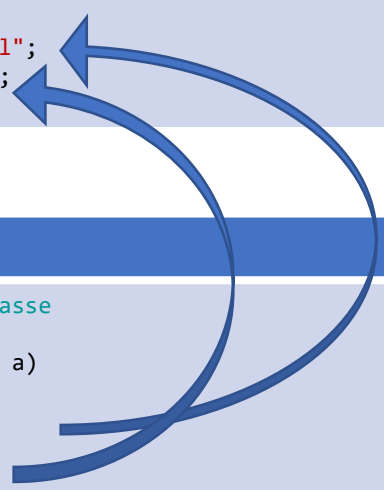
Jedes nicht private Feld, welches in der Basisklasse deklariert wird, existiert auch in der abgeleiteten Klasse

C #

```
class BasisKlasse
{
    protected string name = "Bill";
    protected double wert = 1.41;
}
```

C #

```
class AbgeleiteteKlasse: BasisKlasse
{
    public double Methode(double a)
    {
        Console.WriteLine(name);
        return a + wert;
    }
}
```



# Vererbung Felder

## Beispiel

Zur besseren Erkennung, woher das Feld stammt, kann jeweils mit den Schlüsselwörtern `this` und `base` gearbeitet werden

C #

```
class BasisKlasse
{
    protected double wert1 = 1.41;
}
```

base

C #

```
class AbgeleiteteKlasse: BasisKlasse
{
    private int wert2 = 4;
    public double Methode(double a)
    {
        return a + base.wert1 + this.wert2;
    }
}
```

# Überladen (Overloading)

- Überladen heisst, dass ein Methodename in einer Klasse für mehrere Methoden verwendet wird
- Damit dies möglich ist, müssen folgende zwei Voraussetzungen erfüllt sein:
  - Der Datentyp mindestens eines Übergabeparameters ist anders als in den anderen gleichnamigen Methoden
  - Die Anzahl der Übergabeparameter ist unterschiedlich

C #

```
int Add(int a, int b) { return a + b; }  
float Add(int a, int b, float c) { return a + b + c; }
```

# Überladen (Overloading)

## Beispiel

Die Klasse Console hat verschiedene WriteLine Methoden mit verschiedenen Argumenten:

C #

```
Console.WriteLine();  
Console.WriteLine("Hallo");  
Console.WriteLine(6.55);  
Console.WriteLine("Ausgabe {0}", 22);  
Console.WriteLine(true);
```

Die Klasse string hat verschiedene Format Methoden mit verschiedenen Argumenten:

C #

```
string.Format("Argument: {0}", a1);  
string.Format("Argument: {0}, {1}", a1, a2);  
string.Format("Argument: {0}, {1}, {3}", a1, a2, a3);
```

# Method-Overriding

- Eine Methode kann in einer davon abgeleiteten Klasse **überschrieben** werden
- Beim Überschreiben einer Methode müssen die beiden Methodensignaturen (Methodenname, Parameterliste, Rückgabewert) **exakt** übereinstimmen

C#

```
public class BasisKlasse
{
    public virtual int MachEtwas(int a)
    { return a + a; }
}
public class AbgeleiteteKlasse: BasisKlasse
{
    public override int MachEtwas(int a)
    { return a * a; }
}
```

# Method-Overriding

## virtual

Die Methode der Basisklasse welche durch die Methode der abgeleiteten Klasse überschrieben werden soll, muss dies durch das Schlüsselwort **virtual** zulassen

C #

```
public class BasisKlasse
{
    public virtual int MachEtwas(int a)
    {
        return a + a;
    }
}
```



# Method-Overriding

## override

Die Methode der abgeleiteten Klasse, welche die Methode der Basisklasse überschreibt, muss die gleiche Signatur haben und das Überschreiben durch das Schlüsselwort **override** anzeigen

C#

```
public class AbgeleiteteKlasse: BasisKlasse
{
    public override int MachEtwas(int a)
    {
        return a + a;
    }
}
```

# Method-Overriding

## nicht überschriebene Methoden

Virtuelle Methoden müssen nicht zwingend überschrieben werden

C #

```
public class BasisKlasse
{
    public virtual int MachEtwas (int a)
    { return a + a; }

    public virtual int MachEtwasAnderes (int a)
    { return 5 * a; }
}

public class AbgeleiteteKlasse: BasisKlasse
{
    public override int MachEtwas (int a)
    { return a * a; }
}
```

Die abgeleitete Klasse kann, muss aber die virtuellen Methoden der Basis-Klasse nicht zwingend überschreiben

Fehlende, nicht überschriebene Methoden werden vererbt.

# Method-Overriding

## nicht überschriebene Methoden

Virtuelle Methoden müssen nicht zwingend überschrieben werden

C #

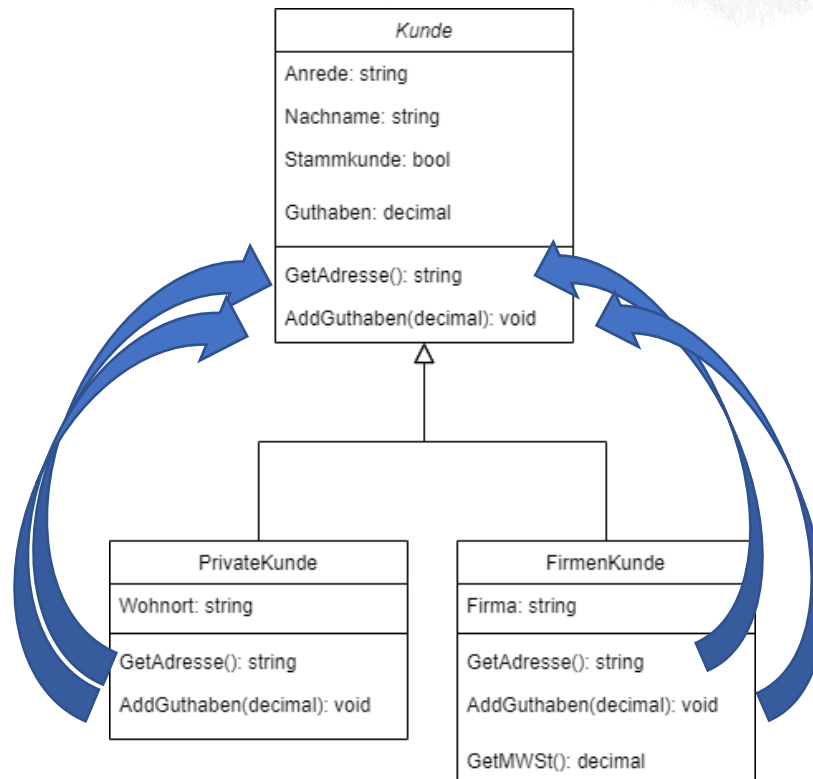
```
static void Main(string[] args)
{
    BasisKlasse b = new BasisKlasse();
    Console.WriteLine($"MachEtwas von Basisklasse: {b.MachEtwas(4)}");
    AbgeleiteteKlasse a = new AbgeleiteteKlasse();
    Console.WriteLine($"MachEtwas von AbgeleiteteKlasse: {a.MachEtwas(4)}");
    Console.WriteLine($"MachEtwasAnderes von AbgeleiteteKlasse: {a.MachEtwasAnderes(4)}");
}
```

```
MachEtwas von Basisklasse: 8
MachEtwas von AbgeleiteteKlasse: 16
MachEtwasAnderes von AbgeleiteteKlasse: 20
```

MachEtwasAnderes wurde in der abgeleiteten Klasse nicht überschrieben, also erbt AbgeleiteteKlasse die Methode von der BasisKlasse.

# Method-Overriding

## Beispiel



# Method-Overriding

## Beispiel

- Subklassen `PrivateKunde` und `FirmenKunde` können auf sämtliche Eigenschaften und Methoden der Basisklasse zugreifen
- Fügen selbst eigene Methoden (auch Eigenschaften möglich) hinzu
- *`GetAdresse()`* und *`AddGuthaben()`* sind überschriebene Methoden. D.h. Adresse und Guthaben werden für Privatkunden anders implementiert als für Firmenkunden





# Allgemeine Hinweise und Regeln zur Vererbung

- Alle öffentlichen Eigenschaften und Methoden der Basisklasse sind auch über die abgeleiteten Subklassen verfügbar.
- Methoden der Basisklasse, die von den abgeleiteten Subklassen überschrieben werden dürfen (sogenannte virtuelle Methoden), müssen mit dem Schlüsselwort **virtual** deklariert werden.
- Fehlt das Schlüsselwort **virtual** bei der Methodendeklaration, so bedeutet das, dass dies die einzige Implementierung der Methode ist.
- Methoden der Subklassen, welche die gleichnamige Methode der Basisklasse überschreiben, müssen mit dem Schlüsselwort **override** deklariert werden.
- Wenn Sie das **override**-Schlüsselwort in der Subklasse vergessen, wird angenommen, dass es sich um eine „Schattenfunktion“ der originalen Funktion handelt. Eine solche Funktion hat denselben Namen wie das Original, überschreibt dieses aber nicht. Bitte solche Fälle dringend vermeiden.
- Private Felder der Basisklasse, auf die die Subklassen zugreifen dürfen, müssen mit **protected** deklariert werden.
- Die Basisklasse wird der Subklasse durch einen der Klassendeklaration nachgestellten Doppelpunkt bekanntgemacht:

## Syntax:

```
class SubKlasse: BasisKlasse
{
    // ... Implementierungscode
}
```

# Allgemeine Hinweise und Regeln zur Vererbung

- Eine Subklasse kann immer nur von einer einzigen Basisklasse abgeleitet werden (keine Mehrfachvererbung möglich). - Mit dem `base`-Objekt kann von den Subklassen auf die Basisklasse zugegriffen werden, mit dem `this`-Objekt auf die eigene Klasse.
- Wenn die Basisklasse einen oder mehrere eigene Konstruktoren mit Parametern verwendet, so müssen in den Subklassen ebenfalls eigene Konstruktoren definiert werden (Konstruktoren können nicht vererbt werden!).
- Der nicht parameterlose Konstruktor einer Subklasse muss einen passenden Konstruktor seiner Basisklasse aufrufen (`base`-Schlüsselwort).
- Falls aber die Basisklasse über keinen eigenen Konstruktor verfügt, wird der Standard Konstruktor automatisch aufgerufen, wenn ein Objekt aus einer Subklasse erzeugt wird.
- Wenn Sie mit Vererbung arbeiten, sollten Sie Folgendes beachten:
  - Es gibt keinerlei Beschränkung bezüglich der Stufenanzahl der Vererbungshierarchie. Sie können die Hierarchie so tief wie nötig staffeln, die Eigenschaften/Methoden werden trotzdem durch alle Vererbungsstufen hindurchgereicht. Allgemein gilt, je weiter unten sich eine Klasse in der Hierarchie befindet, umso spezialisierter ist ihr Verhalten. Zum Beispiel eine `HochschulKunden`-Klasse, die von einer Klasse `SchulKunden` erbt und diese wiederum von der `Kunden`-Klasse.
  - Um die Komplexität zu minimieren und die Wartbarkeit des Codes zu vereinfachen, sollten Sie die Vererbungshierarchie nicht tiefer als ca. vier Stufen staffeln.
  - Jede Subklasse kann nur von einer Basisklasse erben! So kann z. B. eine `HochSchulKunden`-Klasse nicht sowohl von der `Kunden`-Klasse als auch von einer `SchulKunden`-Klasse erben. Das ist in Ordnung so, denn eine solche multiple Vererbung könnte sehr schnell zu einem komplexen und unübersichtlichen Problem werden.



# Allgemeine Hinweise und Regeln zur Vererbung

Es gibt zwei primäre Anwendungsfälle für Vererbung in Anwendungen:

- Sie verwenden Objekte unterschiedlichen Typs mit ähnlicher Funktionalität. So erben z. B. SchulKunden-Klasse und StaatsKunden-Klasse von der Kunden-Klasse.
- Sie haben gleiche Prozesse mit einer Menge von Objekten auszuführen. So erbt z. B. jeder Typ eines Geschäftsobjekts von einer Business Object(BO)-Klasse.

Sie sollten in folgenden Fällen auf Vererbung verzichten:

- Sie brauchen nur eine einzige Funktion von der Basisklasse. In diesem Fall sollten Sie die Funktion in die eigene Klasse delegieren, statt von einer anderen zu erben.
- Sie möchten alle Funktionen überschreiben. In einem solchen Fall sollten Sie eine Schnittstelle (Interface, wird in der nächsten Kurseinheit erklärt) statt Vererbung verwenden.

# Übungen

Uebungen\Kurseinheit 5\Übung  
Vererbung.pdf

Uebungsaufgaben 10.5 s.142