



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>

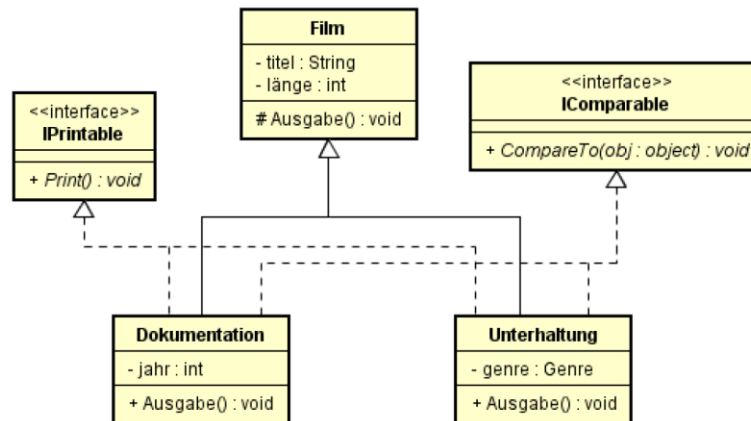
Interfaces

Definition

- Ein Interface dient dazu, Schnittstellen klar zu definieren (Methoden müssen implementiert werden)
- Interfaces dürfen ausschliesslich Deklarationen enthalten, Implementierungen sind nicht erlaubt
- Alle Members des Interface sind automatisch *public*
- Ein Interface ist eine Art Vertrag, welcher mit den implementierenden Klassen vereinbart wird
- Alle Methoden des Interface müssen in der Klasse, welche das Interface nutzt, implementiert werden

Mehrfach Vererbung

- Eine Klasse darf jeweils nur genau von einer anderen Klasse abgeleitet werden (keine Mehrfach-Vererbung)
- Eine Klasse kann aber beliebig viele Interfaces implementieren



Schreibweise

- Die Deklaration des Interface enthält alle vorgeschriebenen Methoden

C #

```
public interface IMeinInterface
{
    int Methode1(int a);
    void Methode2(int a, string b);
    string Methode3();
}
```

Schreibweise

- Die Deklaration der implementierenden Klasse hängt den Namen des Interface mit einem Doppelpunkt an (gleich wie bei der Vererbung).
- Dadurch wird die Klasse gezwungen, alle Methoden des Interface zu implementieren

C #

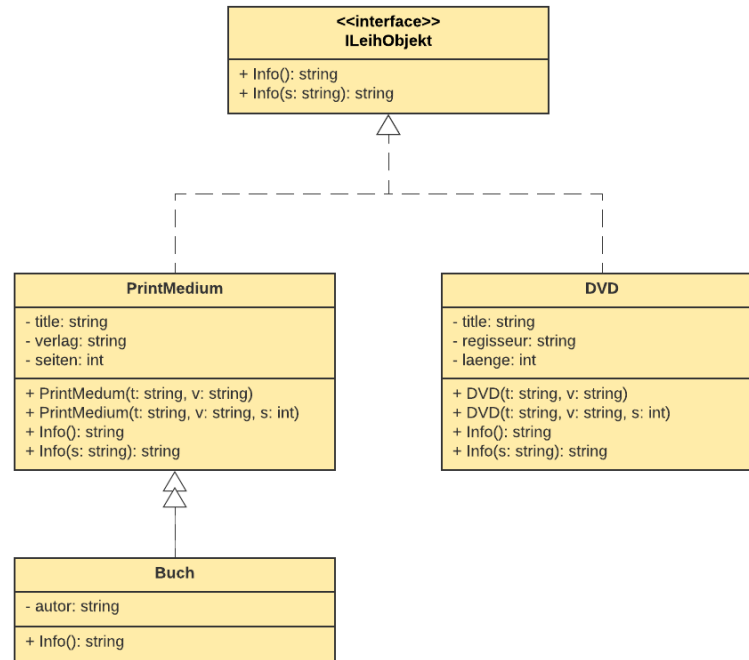
```
public class MeineKlasse : IMeinInterface
{
    public int Methode1(int a)
    { // Code }

    public void Methode2(int a, string b)
    { // Code }

    public string Methode3()
    { // Code }
}
```

Beispiel Bibliothek

Klassendiagramm



Beispiel Bibliothek

Interface: ILeihobjekt

- Die Deklaration des Interface ILeihObjekt schreibt vor, welche Mehtoden implmentiert werden müssen

C #

```
public interface ILeihObjekt
{
    void Info();
    string Info(string s);
}
```

Beispiel Bibliothek

Klasse DVD implementiert ILeihObjekt

C #

```
public class DVD : ILeihObjekt
{
    private int laenge;
    private string titel;

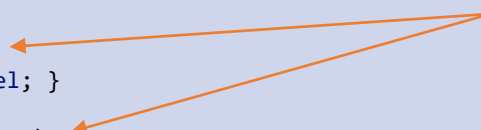
    public DVD(string t)
    {
        this.titel = t;
        Console.WriteLine("--> DVD");
    }

    public DVD(string t, int l) : this(t)
    { this.laenge = l; }

    public string Info()
    { return "\n" + this.titel; }

    public string Info(string s)
    { return this.Info() + s + this.laenge; }
}
```

Müssen implementiert werden,
um die Bedingungen des
Interface zu erfüllen



Beispiel Bibliothek

Klasse PrintMedium implementiert ILeihObjekt

C #

```
public class PrintMedium : ILeihObjekt
{
    private string title;
    private string verlag;
    private int seiten;

    public PrintMedium(string t, string p)
    {
        this.title = t; verlag = p;
        Console.Write("--> PrintMedium");
    }

    public PrintMedium(string t, string p, int s) : this(t, p)
    { seiten = s; }

    public virtual string Info()
    { return $"\\n {this.title} des Verlags {this.verlag}"; }

    public virtual string Info(string s)
    { return this.Info() + s + this.seiten; }
}
```

Müssen implementiert werden,
um die Bedingungen des
Interface zu erfüllen

Beispiel Bibliothek


Klasse Buch erbt von PrintMedium

C #

```
public class Buch : PrintMedium
{
    private string autor;
    public Buch(string t, string p, string a) : base(t, p)
    {
        this.autor = a;
        Console.Write("--> Buch");
    }

    public override string Info()
    { return $"{base.Info()}, Autor: {autor}"; }
}
```

Die Methode Info() kann überschrieben werden, da Info() bereits in PrintMedium implementiert ist, ist aber nicht zwingend nötig



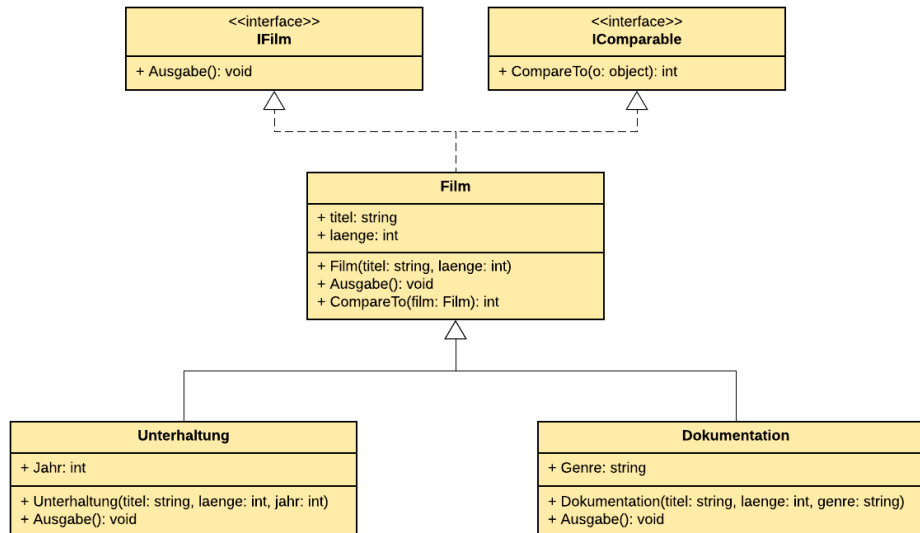
Definierte Interfaces

Im Namespace System von C# existieren eine Reihe von nützlichen Schnittstellen

- `ICloneable` verlangt eine `Clone()` Methode, welche eine exakte Kopie (ein Clone) eines Objektes herstellt.
- `IComparable<Type>` verlangt eine `int Compare(Type other)` Methode, welche zwei Objekte vom Typ `Type` vergleicht.
- `IEquatable<Type>` verlangt eine `bool Equals(Type other)` Methode, welche entscheidet, ob zwei Objekte vom Typ `Type` gleich sind.

Mehrfach-Vererbung

- Da eine Klasse von einer anderen Klasse erben und zusätzlich ein- oder mehrere Interfaces implementieren kann, ist es möglich die Vorteile dieser Interfaces zu nutzen



Die Interfaces IFilm und IComparable

Das Interface IFilm verlangt für alle Filme eine Ausgabe Funktion:

C #

```
interface IFilm
{
    void Ausgabe();
}
```

Das Interface IComparable verlangt eine Vergleichsfunktion, damit die Filme geordnet werden können

C #

```
int CompareTo(Object obj)
```

Die Klasse Film

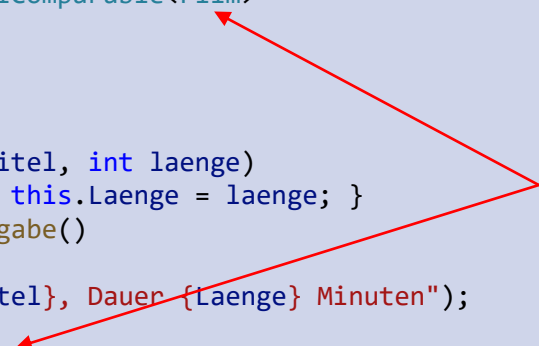
Die Klasse Film soll beide Interfaces implementieren. Also muss sie mindestens eine Funktion Ausgabe und eine Funktion CompareTo haben:

C#

```
public class Film : IFilm, IComparable<Film>
{
    string Titel { get; }
    int Laenge { get; }

    protected Film(string titel, int laenge)
    { this.Titel = titel; this.Laenge = laenge; }
    public virtual void Ausgabe()
    {
        Console.WriteLine($"{Titel}, Dauer {Laenge} Minuten");
    }
    public int CompareTo(Film film)
    {
        return this.Titel.CompareTo(film.Titel);
    }
}
```

Wir wollen nur Objekte vom Typ
"Film" vergleichen können



Die Klasse Dokumentation

Die Klasse Dokumentation erbt von Film und braucht darum die Methoden des Interfaces nicht mehr zu implementieren. Sie kann aber die Ausgabe Methode überschreiben:

C #

```
public class Dokumentation : Film
{
    string Genre { get; }

    public Dokumentation(string titel, int laenge, string genre): base(titel,laenge)
    {
        this.Genre = genre; }

    public override void Ausgabe()
    {
        base.Ausgabe();
        Console.WriteLine($"", aus dem bereich {this.Genre}");
    }
}
```

Die Klasse Unterhaltung

Die Klasse Unterhaltung erbt von Film und braucht darum die Methoden des Interfaces nicht mehr zu implementieren. Sie kann aber die Ausgabe-Methode überschreiben:

C#

```
public class Unterhaltung : Film
{
    int Jahr { get; }

    public Unterhaltung(string titel, int laenge, int jahr) : base(titel, laenge)
    {
        this.Jahr = jahr;
    }

    public override void Ausgabe()
    {
        base.Ausgabe();
        Console.WriteLine($"aus dem Jahr {this.Jahr}");
    }
}
```

Die Main Methode

C #

```
static void Main(string[] args)
{
    Film[] liste = new Film[6];
    liste[0] = new Dokumentation("Der 13.", 100, "Sozial und Kultur");
    liste[1] = new Dokumentation("Durch die Wand", 100, "Sport");
    liste[2] = new Dokumentation("Jane", 90, "Natur");
    liste[3] = new Unterhaltung("Das Leben des Brian", 94, 1979);
    liste[4] = new Unterhaltung("The Big Lebowski", 117, 1998);
    liste[5] = new Unterhaltung("Inglourious Basterds", 153, 2009);
```

```
    Array.Sort(liste);
```

Alle Film-Objekte können verglichen werden, darum funktioniert die Sort()-Funktion

```
    foreach (Film film in liste)
```

```
    {
```

```
        film.Ausgabe();
```

```
    }
```

```
}
```

Alle Film-Objekte haben eine Ausgabe()-Funktion

Die Ausgabe der Main Methode

Die Filme werden durch den Befehl

```
Array.Sort(liste);
```

nach den Film-Titeln sortiert und danach durch den Aufruf

```
foreach (Film film in liste)  
{    film.Ausgabe(); }
```

auf die Console geschrieben.

```
Das Leben des Brian, Dauer 94 Minuten, aus dem Jahr 1979  
Der 13., Dauer 100 Minuten, aus dem bereich Sozial und Kultur  
Durch die Wand, Dauer 100 Minuten, aus dem bereich Sport  
Inglourious Basterds, Dauer 153 Minuten, aus dem Jahr 2009  
Jane, Dauer 90 Minuten, aus dem bereich Natur  
The Big Lebowski, Dauer 117 Minuten, aus dem Jahr 1998
```



Übungen

Uebungsaufgaben

"Uebungen\Kurseinheit 8\Interface\Uebung Interface.pdf"