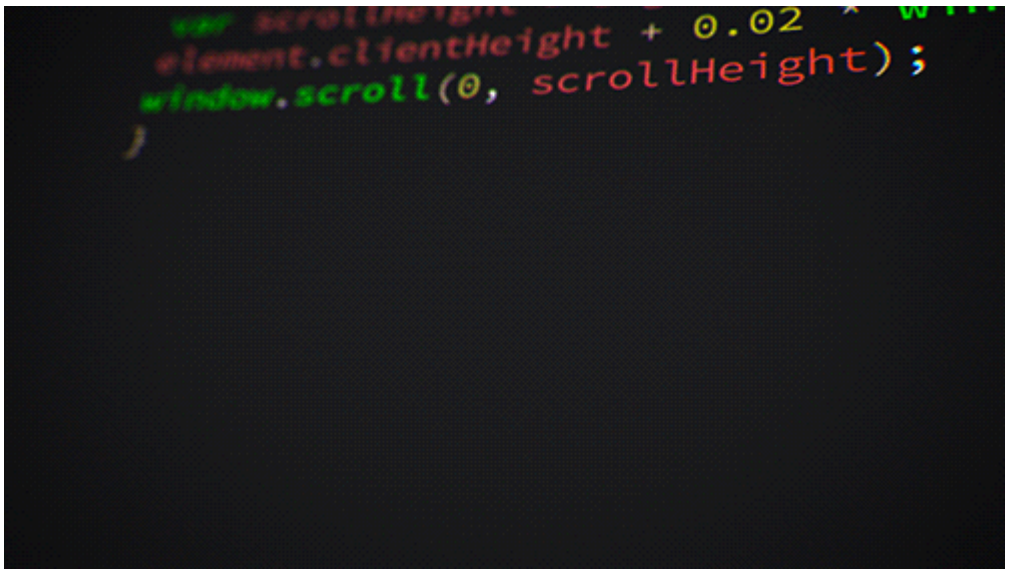
Abstrakte Klassen und Interfaces

Parameter	Kursinformationen
Veranstaltung:	`Vorlesung Softwareentwicklung`
Teil:	`10/27`
Semester	`Sommersemester 2025`
Hochschule:	`Technische Universität Freiberg`
Inhalte:	Lösungsansatz und Formen der Versionsverwaltung, Strategien der Konfliktvermeidung, Git
Link auf den GitHub:	https://github.com/TUBAF-IfI-LiaScript/VL_Softwareentwicklung/blob/master/10_AbstrakteKlassenUndInterfaces.md
Autoren	Sebastian Zug, Galina Rudolf, André Dietrich, `Lina` & `Florian2501`



Abstrakte Klassen / Abstrakte Methoden

> Mit `virtual` werden einzelne Methoden spezifiziert, die durch die abgeleiteten Klassen implmentiert werden. Die Basisklasse hält aber eine "default" Implementierung bereit.

Letztendich kann man diesen Gedanken konsequent weiter treiben und die Methoden der Basisklasse auf ein reines Muster reduzieren, dass keine eigenen Implementierungen umfasst.

Diese Aufgabe übernehmen abstrakte Klassen und abstrakte Methoden. Eine abstrakte Klasse:

- kann nicht instantiiert werden
- kann abstrakte Methoden umfassen
- ist oft als Startpunkt(e) einer Vererbungshierarchie gedacht sind.

Innerhalb der Klasse können abstrakte Methoden integriert werden, die

- implizit als virtuelle Methode implementiert angelegt werden
- entsprechend keinen Methodenkörper umfassen

Eine nicht abstrakte Klasse, die von einer abstrakten Klasse abgeleitet wurde, muss Implementierungen aller geerbten abstrakten Methoden und Accessoren enthalten.

abstractClass

```
using System;

public abstract class Animal
{
  public string Name;
  public Animal(string name){
    Name = name;
  }
  public abstract void makeSound();
}

public class Corcodile : Animal{
  public Corcodile(string name) : base(name){
    Name = name;
  }
  public override void makeSound(){
    Console.WriteLine("I'm a Crocodile");
  }
}

public class Program
{
  public static void Main(string[] args){
    Corcodile A = new Corcodile("Tuffy");
    A.makeSound();
  }
}
```

```
I'm a Crocodile
I'm a Crocodile
```

Abstrakte Klassen dienen somit als Template für nachgeordnete Unterklassen. Neben Methoden können auch Properties und Indexer als abstrakt deklariert werden.

> Warum macht es keinen Sinn eine abstrakte Klasse als `sealed` zu deklarieren?

Interfaces

Interfaces setzen die Idee der abstrakten Klassen konsequent fort und umfassen nur abstrakte Member. Sie bilden die Signatur einer Klasse, in der Methoden, Properties, Indexer und Events erfasst werden.

Merke: Interfaces umfassen keine Felder!

Charakteristik von Interfaces:

- alle Bestandteile aus einem Interface müssen implementiert werden
- Klassen „implementieren“ Interfaces und „erben“ von Basisklassen
- Interfaces haben das Schlüsselwort `interface` und fangen im allgemeinen mit dem Buchstaben I an
- alle Elemente sind implizit `abstract` und `public`

InterfaceExample

```
1  using System;
2  using System.Reflection;
3  using System.ComponentModel.Design;
4
5  interface IShape
6  {
7    double Area();
8    double Scope();
9  }
10
11 class Rectangular : IShape   // Rectangular implementiert das Interfa
     IShape
12 {
13   double area;
14   double scope;
15   public double Area() => area;
16   public double Scope() => scope;
17   public Rectangular(double sideA, double sideB)
18   {
19     area = sideA * sideB;
20     scope = 2 * sideA + 2 * sideB;
21   }
22 }
23
24 public class Program
25 {
26   public static void Main(string[] args)
27   {
28     Rectangular rect = new Rectangular(2, 3);
29     Console.WriteLine("Area: {0}, " + "Scope: {1}", rect.Area(), rect
         ());
30   }
31 }
```

```
Area: 6, Scope: 10
Area: 6, Scope: 10
```

Eine Klasse kann nur von einer anderen Klasse erben, aber beliebig viele Interfaces implementieren.

Schnittstellen werden verwendet:

- um eine lose Kopplung zu erreichen.
- um eine vollständige Abstraktion zu erreichen.
- um komponentenbasierte Programmierung zu erreichen
- um Mehrfachvererbung und Abstraktion zu erreichen.

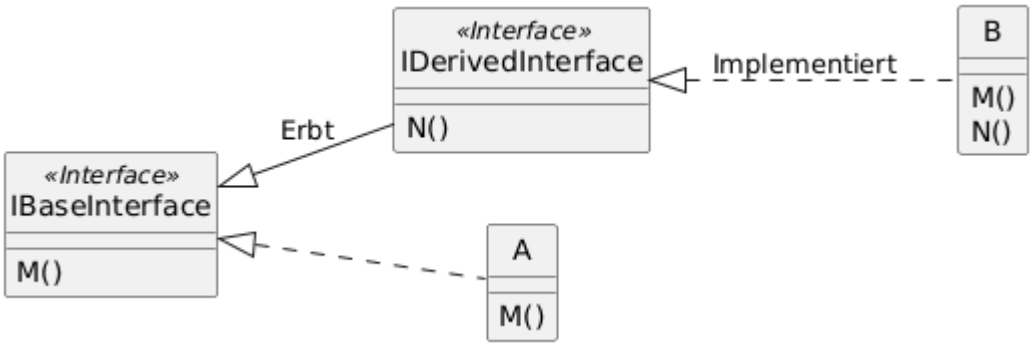
Vererbung

ImplementingInterface

```
1  using System;
2  using System.Reflection;
3  using System.ComponentModel.Design;
4
5  interface IBaseInterface { void M(); }
6  interface IDerivedInterface : IBaseInterface { void N(); }
7
8  class A : IBaseInterface
9  {
10   public void M()
11   {
12     Console.WriteLine("Methode M in {0}", this.GetType().Name);
13   }
14 }
15
16 class B : IDerivedInterface
17 {
18   public void M()
19   {
20     Console.WriteLine("Methode M in {0}", this.GetType().Name);
21   }
22   public void N()
23   {
24     Console.WriteLine("Methode N in {0}", this.GetType().Name);
25   }
26 }
27
28 public class Program
29 {
30   public static void Main(string[] args)
31   {
32     IBaseInterface t1 = new A();    // Statischer Typ IBaseInterface,
         dynamischer class A
33     IBaseInterface t2 = new B();    // Statischer Typ IBaseInterface,
         dynamischer class B
34     t1.M();
35     t2.M();
36     Console.WriteLine(t2 is IDerivedInterface);
37     (t2 as IDerivedInterface).N();
38     (t2 as B).N();
39   }
40 }
```

```
Methode M in A
Methode M in B
True
Methode N in B
Methode N in B
Methode M in A
Methode M in B
True
Methode N in B
Methode N in B
```

Es besteht keine Vererbungshierarchie zwischen den beiden Klassen `A` und `B`! Vielmehr ergibt sich ein neuer Zusammenhang, die gemeinsame Implementierung eines Musters an Membern.



Die Visualisierung von Klassen und deren Abhängigkeiten mit plantUML ist eine Möglichkeit einen raschen Überblick über bestimmte Zusammenhänge zu gewinnen. In den folgenden Materialien wird dies intensiv genutzt.

Interfaces vs. Abstrakte Klassen

interface	**abstract class**
viele Interfaces möglich	immer nur eine Basisklasse
speichert keine Daten	kann Felder umfassen
keine Konstruktorensignaturen	kann Konstruktoren umfassen
beinhaltet nur Methodensignaturen	kann Signaturen und Implementierungen integrieren
keine Zugriffsmodifizierer	beliebige Zugriffsmodifizierer
keine statischen Member	statische Member möglich

> Merke: Interfaces geben keine Struktur vor, sondern nur ein Verhalten!

Bedeutung von Interfaces

Die C# Bibliothek implementiert eine Vielzahl von Interfaces, die insbesondere für die Handhabung von Datenstrukturen in jedem Fall genutzt werden sollten.

Informieren Sie sich unter Link über die wichtigsten davon wie:

- IEnumerable, IEnuerator
- IList
- IComparable
- ICollection
- ...

UpCastExample

```
using System;
using System.Reflection;
using System.ComponentModel.Design;

public class Cat: IComparable
{
    public string Name {get; set;}
    public int CompareTo(object obj)
    {
        if (!(obj is Cat))
        {
            throw new ArgumentException("Compared Object is not of Ca
        }
        Cat cat = obj as Cat;
        return Name.CompareTo(cat.Name);
    }
}

public class Program
{
  public static void Main(string[] args)
  {
    Cat[] cats = new Cat[]
    {
        new Cat()  {Name = "Mizekatze"},
        new Cat()  {Name = "Beethoven"},
        new Cat()  {Name = "Alex"},
    };
    Array.Sort(cats);
    Array.ForEach(cats, x => Console.WriteLine(x.Name));
  }
}
```

```
Alex
Beethoven
Mizekatze
Alex
Beethoven
Mizekatze
```

Auflösung von Namenskonflikten

UpCastExample

```
using System;

interface IInterfaceA{
  void M();
}

interface IInterfaceB{
  void M();
}

public class SampleClass : IInterfaceA, IInterfaceB
{
    // Hier ist die zuordnung nicht eindeutig
    public void M()
    {
        Console.WriteLine("Gib irgendwas aus!");
    }
}

public class Program
{
  public static void Main(string[] args)
  {
    SampleClass sample = new SampleClass();
    sample.M();
    IInterfaceA A = sample;
    IInterfaceB B = sample;
    A.M();
    B.M();
  }
}
```

```
Gib irgendwas aus!
Gib irgendwas aus!
Gib irgendwas aus!
Gib irgendwas aus!
Gib irgendwas aus!
Gib irgendwas aus!
```

Wenn zwei Schnittstellenmember nicht dieselbe Funktion durchführen sollen muss diese separat implementiert werden. Hierzu wird ein Klassenmember erstellt, der sich explizit auf das Interface bezieht und den Namen der Schnittstelle benennt.

```
public class SampleClass : IInterfaceA, IInterfaceB
{
    // Hier ist die zuordnung nicht eindeutig
```

```
    // Hier ist die Zuordnung nicht eindeutig
    void IInterfaceA.M()
    {
        Console.WriteLine("IInterfaceA - Gib irgendwas aus!");
    }

    void IInterfaceB.M()
    {
        Console.WriteLine("IInterfaceB - Gib irgendwas aus!");
    }
}
```

Allerdings kann diese Funktion dann nur über die Schnittstelle und nicht über die Klasse aufgerufen werden.

Beispiel der Woche

Aufgabe

Nehmen wir an, das Prüfungsamt engagiert Sie für einen Auftrag. Im Laufe mehrerer Wochen sind viele Prüfungszertifikate eingegangen. Unglücklicherweise geht aus den Dateinamen nicht hervor, auf welche Lehrveranstaltung diese sich beziehen. Ihr Auftrag besteht darin, diese Frage automatisiert zu beantworten.

Dabei existiert eine erste Lösung, das Beispielprojekt mit allen Textdateien finden Sie im Repository.

```
using System;
using System.IO;

public class CertificateEvaluator{
    private string fileName;
    public CertificateEvaluator(string fileName)
    {
        this.fileName = fileName;
    }
    public void RunEvaluation(string patter)
    {
        bool result = false;
        using (StreamReader file = File.OpenText(fileName))
        {
            string line = file.ReadLine();
            result = line.Contains(patter);
        }
        Console.Write("{0:-20} - ", fileName);
        if (result) Console.WriteLine($"references {patter}!");
        else Console.WriteLine("contains unknown certificate");
    }
}

public class RunCode
{
    public static void Main(string[] args)
```

```
    public static void Main(string[] args)
    {
        string fileName = "./files/textfile_0.txt";
        const string pattern = "VL Softwareentwicklung";
        CertificateEvaluator CertProcessor = new CertificateEvaluator(fileN
        CertProcessor.RunEvaluation(pattern);
    }
}
```

Welche Verbesserungsmöglichkeiten sehen Sie?

1. `RunEvaluation` mischt zwei Dinge, das Management aller Dateien und die eigentlichen Business-Logik - die "Textanalyse"
2. Es wird nur ein Typ von Dateien überhaupt unterstützt, zudem ist die Art hart codiert - `txt`
3. Es existiert keinerlei Fehlerhandling, weder in Bezug auf die Prüfung der Dateinamen noch mit Blick auf die eingelesen Informationen (`Nullable` Check für das Streamreader Objekt)
4. Die Parameter - Ordner und Dateiname - werden im Code hinterlegt.
5. ...

Was müssen wir anpassen, wenn nun auch plötzlich `docx` Dateien zusätzlich auftauchen? Diese können wir nicht als Streamobjekt lesen!.

Start.cs

```
using System;
using System.IO;
using DocumentFormat.OpenXml;
using DocumentFormat.OpenXml.Packaging;
using DocumentFormat.OpenXml.Wordprocessing;

public abstract class Certificate{      // <- Warum nutzen wir hier kein
  Interface?
    public string? fileName;
    public string? folderName;
    public abstract string getFirstLineContent();
}

public class CertificateTxt : Certificate
{
    public CertificateTxt(string fileName)
    {
        this.fileName = fileName;
    }
    public override string getFirstLineContent(){
        string line = "";
        using (StreamReader file = File.OpenText(fileName))
        {
            line = file.ReadLine();
        }
        return line;
    }
}

public class CertificateDocx : Certificate
{
    public CertificateDocx(string fileName)
    {
        this.fileName = fileName;
    }
    public override string getFirstLineContent(){
        string line = "";
        using (WordprocessingDocument wordDocument = WordprocessingDocument
          (fileName, false))
        {
            var firstParagraph = wordDocument.MainDocumentPart.RootElement
              .Descendants<Paragraph>().First();
            line = firstParagraph.InnerText;
        }
        return line;
    }
}
```

```
public class RunCode
{
    // Business Logik für unseren Anwendungsfall
    public static void CheckCertificates(Certificate cert, string pattern){
        // Datenaggregation
        string line = cert.getFirstLineContent();
        // Patternüberprüfung
        bool result = line.Contains(pattern);
        // Ausgabe des Resultates
        Console.Write("{0:-20} - ", cert.fileName);
        if (result) Console.WriteLine($"references {pattern}!");
        else Console.WriteLine("contains unknown certificate");
    }

    public static void Main(string[] args)
    {
        string fileName = "./files/docxfile_1.docx";
        const string pattern = "VL Softwareentwicklung";

        CertificateDocx certTxtFile = new CertificateDocx(fileName);
        CheckCertificates(certTxtFile, pattern);
    }
}
```

Für die Nutzung der `DocumentFormat` Bibliothek müssen wir diese im Projekt noch als Dependency installieren.

```
dotnet add package DocumentFormat.OpenXml
```

Achtung! Die Lösung ignoriert eine Vielzahl von Hinweisen des Compilers auf mögliche `null references`. In einer realen Implementierung sollte dies berücksichtigt werden.

Aufgaben

- [ ] Setzen Sie sich mit den Konzepten von Interfaces auseinander!