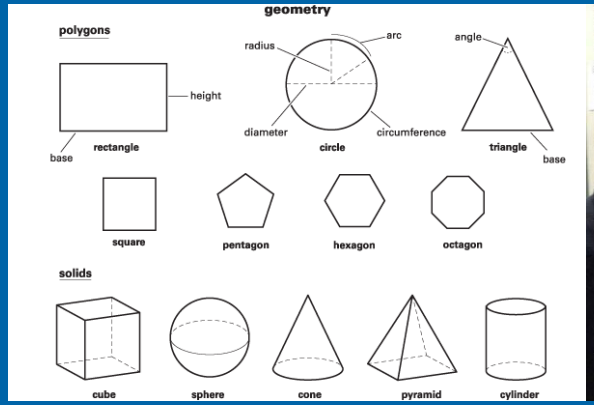
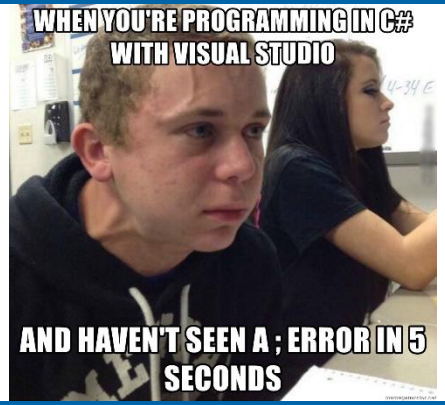
Großes Beispiel: Klassenbibliothek





Eigene Exceptions

Aufbau einer Klassenbibliothek mit VS 2017

Nutzung einer Klassenbibliothek

Einschub: Exceptions

- die Klasse **Exceptions** ist eine ableitbare Basisklasse
- eigene Exceptions können durch Vererbung erstellt werden
- mit **throw** können Exceptions explizit ausgeworfen werden
- Exceptions können erneut Exceptions werfen

```
class MyException : Exception
{
    public override string Message
        => "Oje. Hier ist etwas schief gelaufen";
}
```

```
class MyException : Exception
{
    public override string Message
        => "Oje. Hier ist etwas schief gelaufen";
}
```

```
static void Main(string[] args)
{
    try {
        for (int i = 0; i < 10; i++)
        {
            if (i == 5) throw new MyException();
            if (i == 8) SomeFeature();
        }
    } catch( Exception e ) {
        if (e is MyException)
        {
            Console.WriteLine(e);
            throw new Exception();
        }
        if (e is NotImplementedException)
        {
            Console.WriteLine("Feature is not implemented yet");
        }
    }
}
```

```
} catch( Exception e ) {
    switch (e)
    {
        case MyException ex:
            Console.WriteLine(e);
            throw new Exception();
            break;
        case NotImplementedException ex:
            Console.WriteLine("Feature is not implemented yet");
            break;
        default: throw e;
    }
}
```

```
} catch( MyException e) {
    Console.WriteLine(e);
    throw new Exception();
} catch (NotImplementedException) {
    Console.WriteLine("Feature is not implemented yet");
}
```

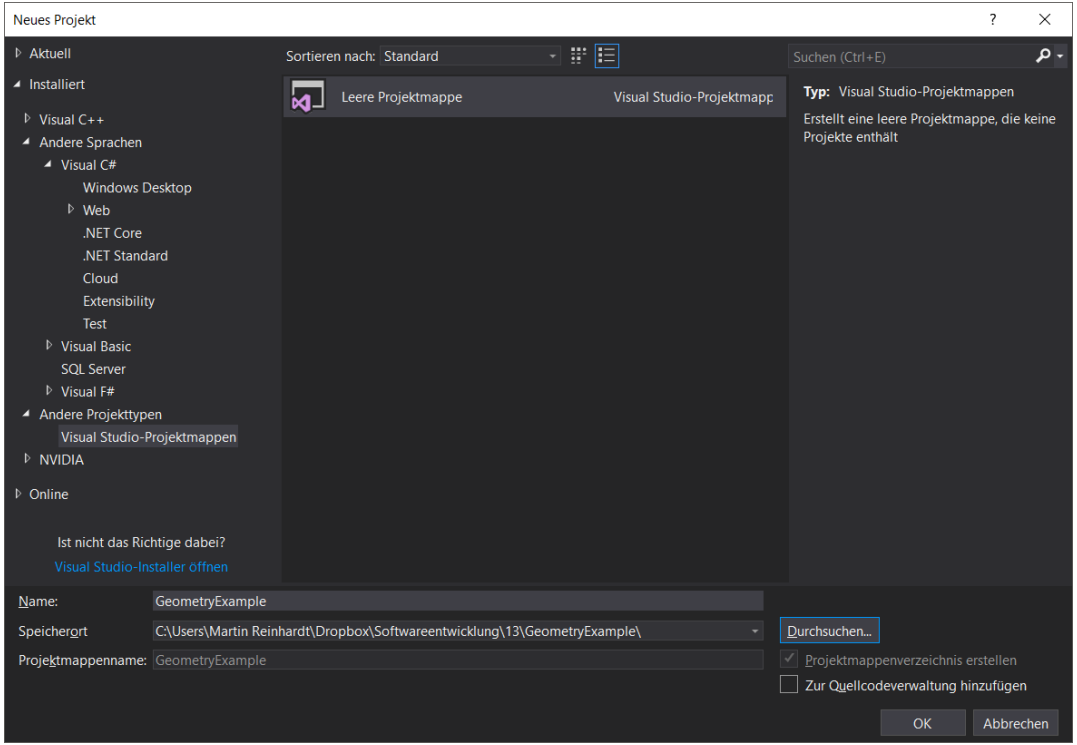
Eine Klassenbibliothek für geometrische Formen

- 2D und 3D Formen sollen behandelt werden
- 2D Formen sollen Flächeninhalt zurückgeben können
- 3D Formen sollen Volumen zurückgeben können
- Alle Formen sollen einen Namen haben
- Ecken sollen gezählt werden können
- Ziel ist eine C# Bibliothek zur Weitergabe an Andere Benutzer

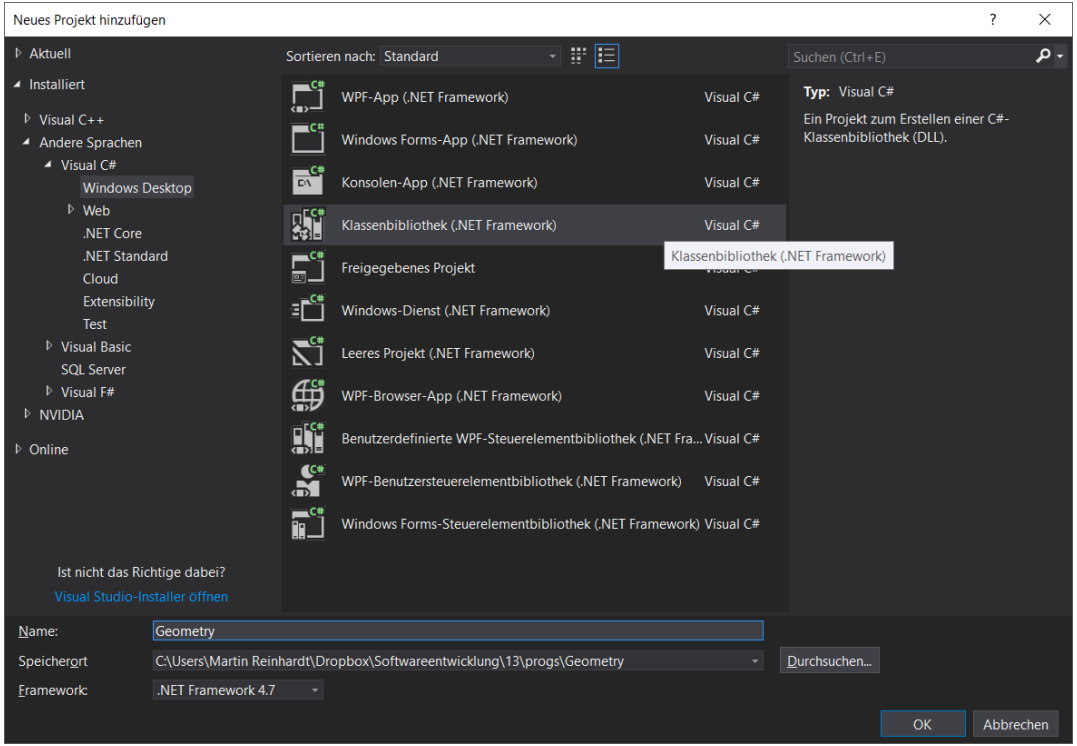
Hierarchie / Architektur

→ Tafel

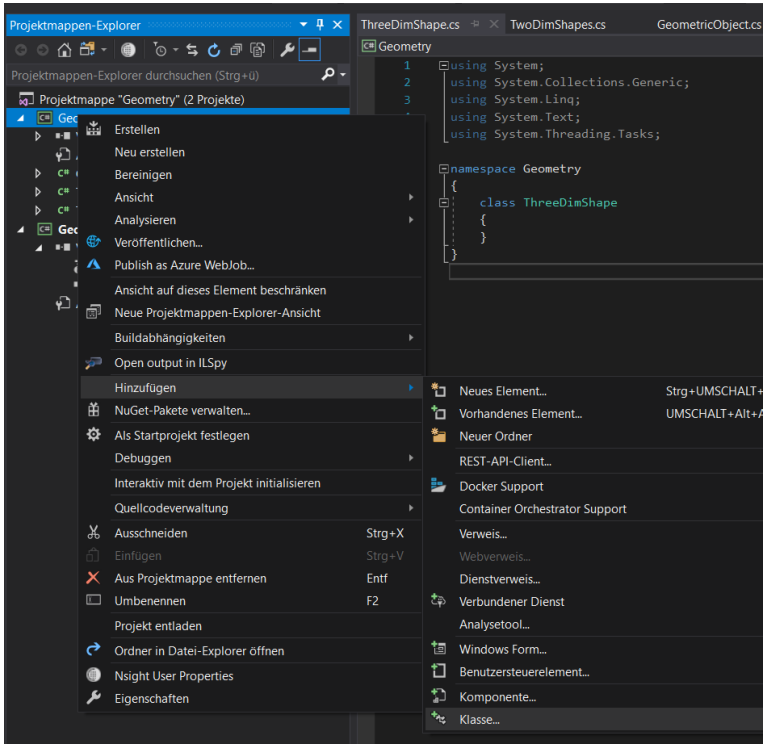
Zuerst leere Projektmappe anlegen



Klassenbibliothek als Projekt hinzufügen

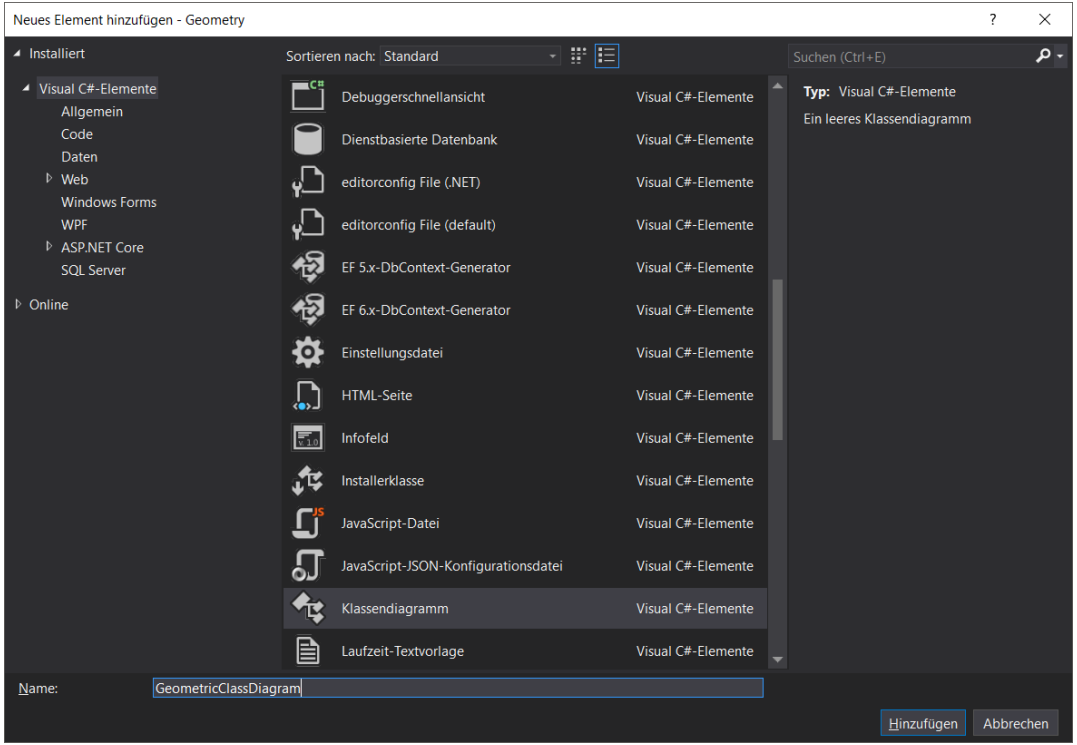


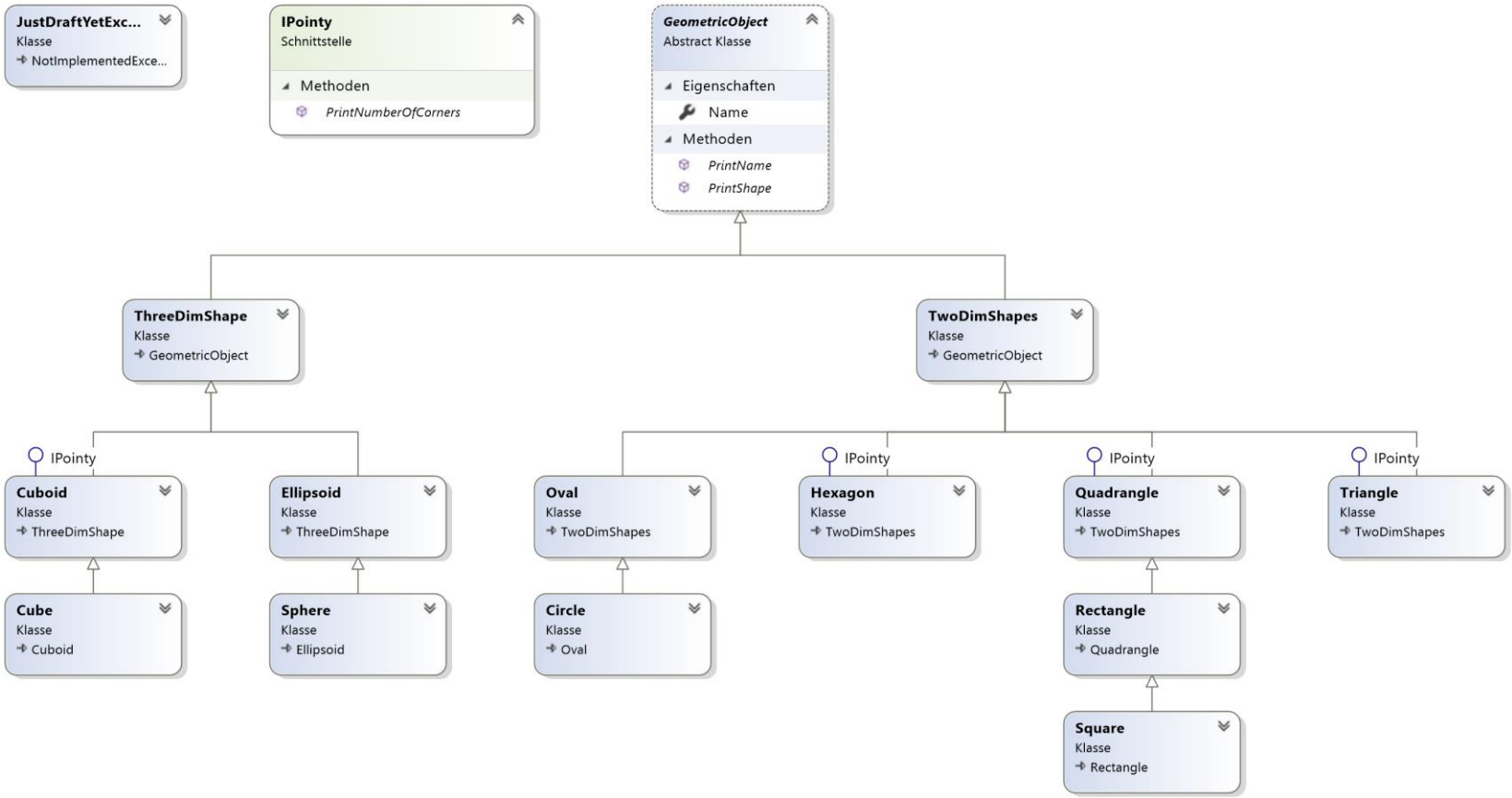
Hinzufügen von Klassen zur Bibliothek

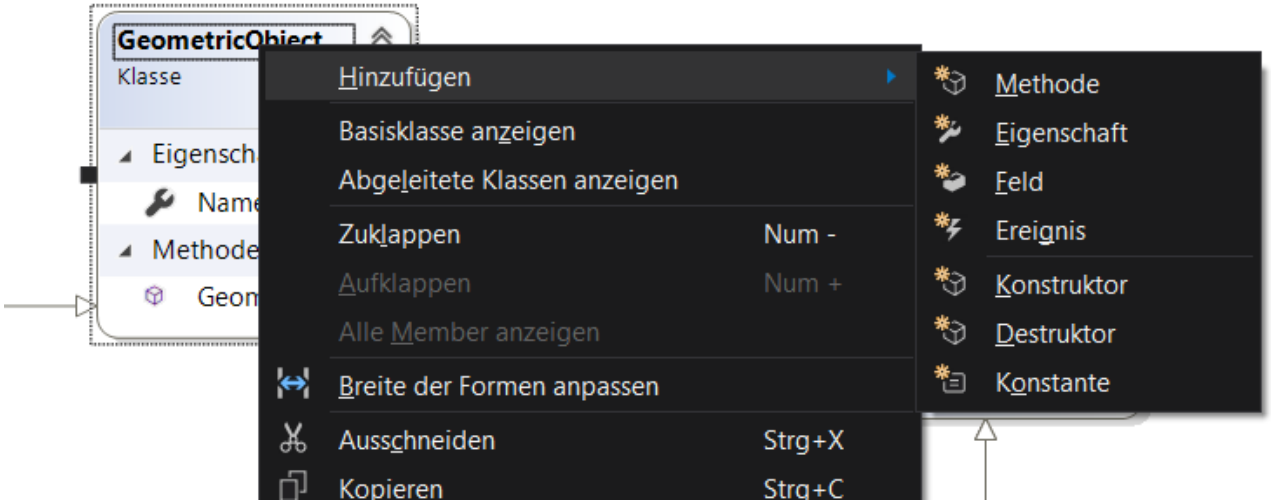


- Alle Klassen und Interfaces erst einmal leer lassen
- jede Klasse sollte eigene Datei haben
- gemeinsamen Namensraum nicht vergessen
- Vorsicht: Klassen müssen **außerhalb** der Assembly nutzbar sein

Klassendiagramm erstellen




JustDraftYetExc...
Klasse
NotImplementedExce...
IPointy
Schnittstelle
Methoden
PrintNumberOfCorners
GeometricObject
Abstract Klasse
Eigenschaften
Name
Methoden
PrintName
PrintShape
ThreeDimShape
Klasse
GeometricObject
TwoDimShapes
Klasse
GeometricObject
IPointy
Cuboid
Klasse
ThreeDimShape
Ellipsoid
Klasse
ThreeDimShape
Oval
Klasse
TwoDimShapes
IPointy
Hexagon
Klasse
TwoDimShapes
IPointy
Quadrangle
Klasse
TwoDimShapes
IPointy
Triangle
Klasse
TwoDimShapes
Cube
Klasse
Cuboid
Sphere
Klasse
Ellipsoid
Circle
Klasse
Oval
Rectangle
Klasse
Quadrangle
Square
Klasse
Rectangle


GeometricObject
Klasse
Eigensch
Name
Methode
Geom
Hinzufügen
Basisklasse anzeigen
Abgeleitete Klassen anzeigen
Zuklappen Num -
Aufklappen Num +
Alle Member anzeigen
Breite der Formen anpassen
Ausschneiden Strg+X
Kopieren Strg+C
Methode
Eigenschaft
Feld
Ereignis
Konstruktor
Destruktor
Konstante

```
namespace Geometry
{
    public abstract class GeometricObject
    {
        public string Name { get; protected set; }
        public virtual void PrintName()
        {
            System.Console.WriteLine(Name);
        }
        public abstract void PrintShape();
        public GeometricObject() => Name = "unbekannt";
    }
}
```

- abstrakte Basisklasse **Geometry**
- eigene Exception oder Standard-Version von **NotImplementedException**

```
namespace Geometry
{
    public class JustDraftYetException : NotImplementedException
    {
        public JustDraftYetException() :
            base("It's just a draft yet." +
                "This feature is not implemented!")
        { }
    }
}
```

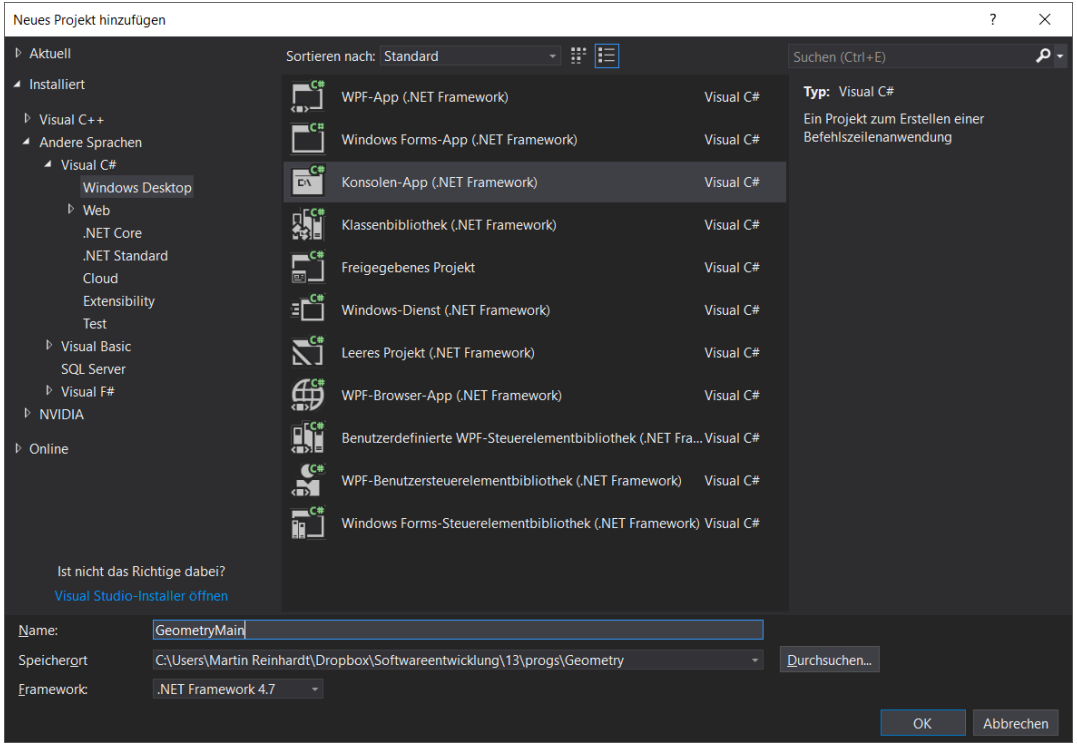
```
namespace Geometry
{
    public class TwoDimShapes : GeometricObject
    {
        public virtual double CalculateArea()
        {
            throw new JustDraftYetException();
        }

        public override void PrintShape()
        {
            throw new JustDraftYetException();
        }
    }
}
```

- Klassen zuerst leer lassen
- das Grundkonstrukt erst einmal schaffen
- Good Practice: **NotImplementedException**
- alles überschreibbar belassen

Zwischenstand

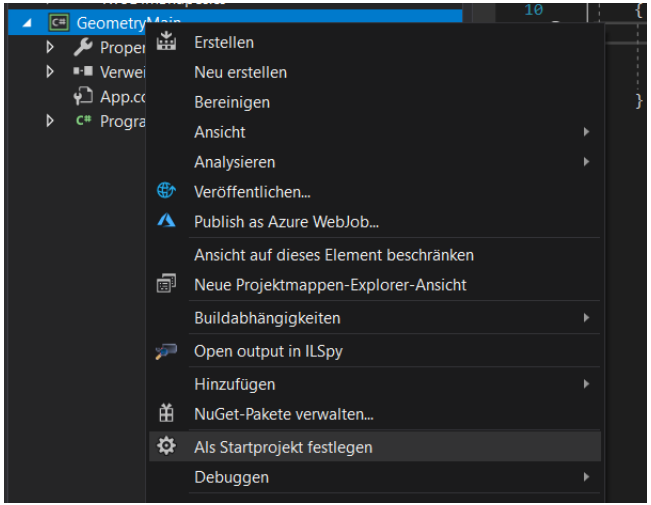
- Projektmappe
- Klassenbibliothek mit allen Klassen und Vererbung und Methodensignaturen
- alle Methoden werfen Exceptions: **NotImplementedException**
- Klassendiagramm zur besseren Übersicht

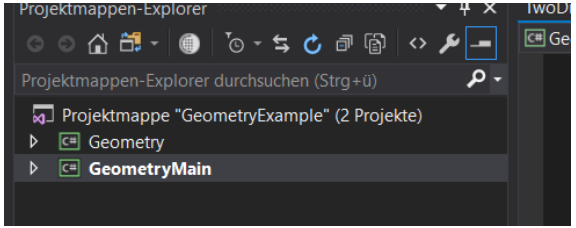
Hinzufügen einer Konsolen-App



- Ergebnis: 2 Projekte in der Projektmappe
- Projekte müssen aufeinander abgestimmt werden
- Reihenfolge der Kompilierung ist entscheidend

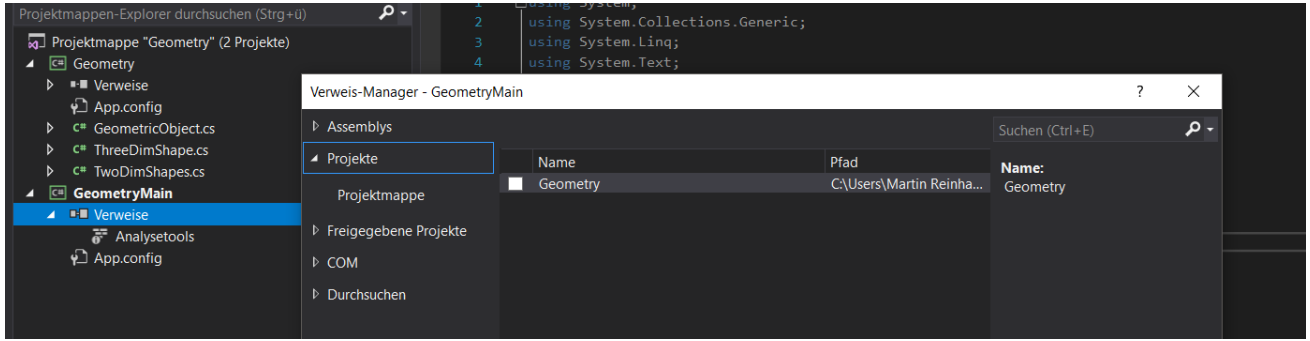
Startprojekt festlegen





- Startprojekt einer Lösung muss explizit bestimmt werden
- Projekt **Geometry** muss nun **GeometryMain** bekannt gemacht werden

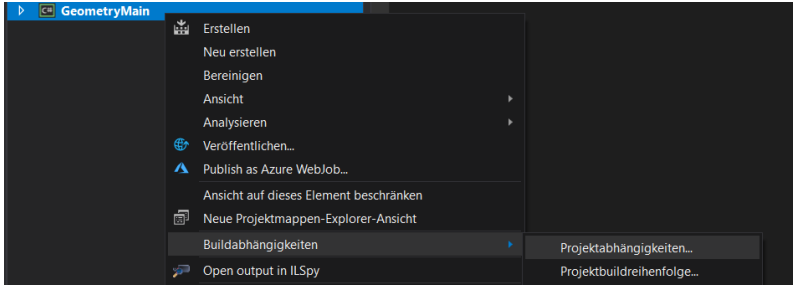
Verweise einbinden



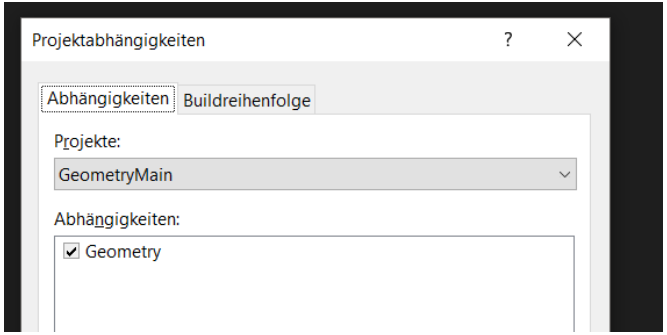

```
using System;
using Geometry;

class Program
{
    static void Main(string[] args)
    {
    }
}
```

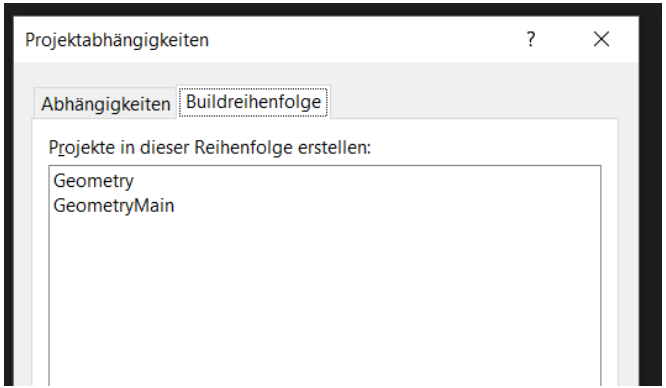
- **Assembly** muss als Verweis bekannt gemacht werden
- Nützlich: Einbinden des Namensraums

Projektabhängigkeit und Reihenfolge festlegen



- Abhängigkeit und Reihenfolge überprüfen
- VS 2017 sollte Reihenfolge alleine bestimmen können





Nähere Implementierung am Beispiel

```
public class TwoDimShapes : GeometricObject
{
    public virtual double CalculateArea()
    {
        throw new JustDraftYetException();
    }

    public override void PrintShape()
    {
        Console.WriteLine("I'm flat");
    }
}
```

- erste Generation: nur wenig Anpassung
- **CalculateArea()** nicht allgemein implementierbar

```
public class Quadrangle : TwoDimShapes, IPointy
{
    public virtual void PrintNumberOfCorners()
        => Console.WriteLine(
            "A Quadrangle has 4 Corners");

    private double a, b, c, d;

    public Quadrangle(string name, double a, double b, double c, double d)
    {
        this.a = a; this.b = b; this.c = c; this.d = d; Name = name;
    }

    protected Quadrangle() { }

    public override void PrintShape()
        => Console.WriteLine("I'm a Quadrangle");

    public override double CalculateArea()
        => base.CalculateArea();
}
```

```
public class Rectangle : Quadrangle
{
    private double a, b;

    public Rectangle(string name, double a, double b)
    {
            this.a = a; this.b = b; Name = name;
    }

    public override double CalculateArea()
    {
        return a * b;
    }

    public override void PrintShape()
        => Console.WriteLine("I'm a Rectangle");

    public override void PrintNumberOfCorners()
        => Console.WriteLine(
            "A Rectangle has 4 corners with 90 degree");

    protected Rectangle() { }
}
```

```
public class Square : Rectangle
{
    public Square(string name, double a) : base(name, a, a)
    { }

    public override void PrintNumberOfCorners()
    {
        base.PrintNumberOfCorners();
        System.Console.WriteLine(
            "Additionaly I'm a square and my sides have the same Length.");
    }
```

- Hier fast vollständige Wiederverwertung des zuvor geschriebenen Codes

Bibliothek testen

- Nutzung des Interface
- Nutzung von Basisklassen
- Nutzung der 3.Generation
- Dynamische Bindung

```
static void PrintingFun(IPointy item)
    => item.PrintNumberOfCorners();

static void PrintingArea(Quadrangle item)
    => Console.WriteLine($"Area: {item.CalculateArea()}");
```

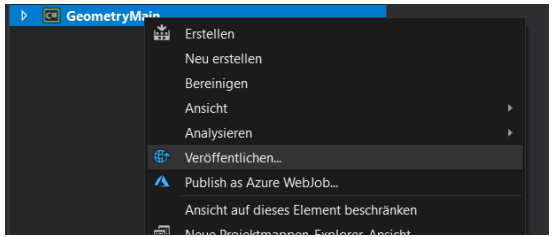
```
static void Main(string[] args)
{
    Quadrangle[] geometryList = new Quadrangle[4];
    geometryList[0] = new Rectangle("Rect-1", 5, 2);
    geometryList[1] = new Square("MyBox5", 3);
    geometryList[2] = new Rectangle("Rect-2", 4, 1);
    geometryList[3] = new Quadrangle("Quad", 1, 2, 3, 4);

    for (int i = 0; i < 4; ++i)
    {
        geometryList[i].PrintName();
        PrintingFun(geometryList[i]);
        try {
            PrintingArea(geometryList[i]);
        } catch (NotImplementedException e) {
            Console.WriteLine(e.Message);
        }
        Console.WriteLine();
    }
    Console.ReadKey();
}
```

- Ausgabe entspricht der Erwartung:

```
>GeometryMain.exe
Rect-1
A Rectangle has 4 corners with 90 degree
Area: 10

MyBox5
A Rectangle has 4 corners with 90 degree
Additionaly I'm a square and my sides have the same Length.
Area: 9

Rect-2
A Rectangle has 4 corners with 90 degree
Area: 4

Quad
A Quadrangle has 4 Corners
It's just a draft yet.This feature is not implemented!


>
```

Deployment

- **GeometryMain.exe** und **Geometry.dll** können ausgeliefert werden
- Bei Updates: Einfach Bibliothek austauschen
- Bei Nutzung der DLL in anderen Projekten: Verweis auf die Bibliothek in der Projektdatei setzen
- Für größere Programme: Deployment-Tool von VS 2017



A word about Comments

Beispiel für Kommentare:

```
// This interface is especially for geometric objects with corners
public interface IPointy
{
    void PrintNumberOfCorners();
}
```

Nachteile:

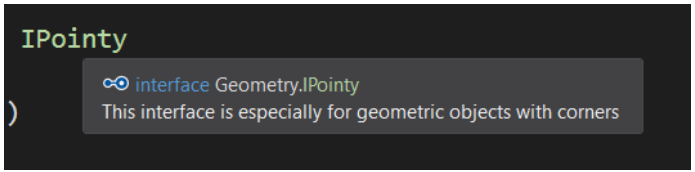
- Keine Möglichkeit die Kommentare außerhalb der Klasse zu sehen
- Keine saubere Exportmöglichkeit

XML Kommentare

Besser: XML Kommentare

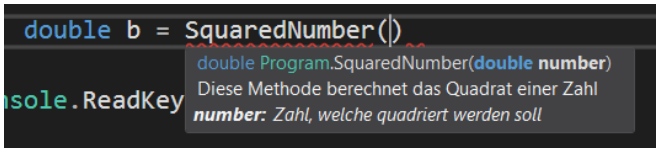
```
/// <summary>
/// This interface is especially for geometric objects with corners
/// </summary>
public interface IPointy
{
    void PrintNumberOfCorners();
}
```

- Können von IntelliSense gelesen und ausgewertet werden:



wichtigste Elemente

```
/// <summary> Diese Methode berechnet das Quadrat einer Zahl </summary>
/// <remarks> Hier steht genau, wie das funktioniert </remarks>
/// <param name="number">Zahl, welche quadriert werden soll</param>
/// <returns>Der Rückgabewert beinhaltet das Quadrat der ursprünglichen Zahl</returns>
/// <code> double testVariable = SquaredNumber(3.5)</code>
/// <exception cref="NotImplementedException">
/// Die Methode ist für Inputs kleiner als 0 nicht implementiert
/// </exception>
public static double SquaredNumber(double number)
{
    if (number < 0) throw new NotImplementedException();
    return number * number;
}
```




```
>csc /target:library /doc:CommentTest.xml CommentTest.cs
Microsoft (R) Visual C# Compiler Version 2.8.2.62916 (2ad4aabc)
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.


>
```

Ergebnis ist unformatiert:

```
<?xml version="1.0"?>
<doc>
    <assembly>
        <name>CommentTest</name>
    </assembly>
    <members>
        <member name="T:CommentTest">
            <summary>
            Klasse mit der statischen Methode SquaredNumber
            </summary>
        </member>
        <member name="M:CommentTest.SquaredNumber(System.Double)">
            <summary> Diese Methode berechnet das Quadrat einer Zahl </summary>
            <remarks> Hier steht genau, wie das funktioniert </remarks>
            <param name="number">Zahl, welche quadriert werden soll</param>
            <returns>Der Rückgabewert beinhaltet das Quadrat der ursprünglichen Zahl</returns>
            <code> double testVariable = SquaredNumber(3.5)</code>
            <exception cref="T:System.NotImplementedException">
            Die Methode ist für Inputs kleiner als 0 nicht implementiert
            </exception>
        </member>
    </members>
</doc>
```

Dokumentation

Bessere Darstellung mittels Stylesheets oder externen Programmen

Documentation Preview

Method SquaredNumber

Diese Methode berechnet das Quadrat einer Zahl

Syntax

C#

```
public static double SquaredNumber(double number)
```

Parameters

number

Zahl, welche quadriert werden soll

Return Value

Der Rückgabewert beinhaltet das Quadrat der ursprünglichen Zahl

Exceptions

Exception	Condition
NotImplementedException	Die Methode ist für Inputs kleiner als 0 nicht implementiert

Remarks

Hier steht genau, wie das funktioniert

Zusammenfassung

- grober Ablauf bei der Erstellung einer Klassenbibliothek
- Nutzung von eigenen Exceptions
- Deployment von Programmen
- Verwendung von geeignetetn Kommentaren