

```

/// <summary>
/// Gets a list with available T types.
/// Loads all assemblies from executing application.
/// searches for T types.
/// </summary>
/// <typeparam name="T">Type to use search</typeparam>
/// <returns>
/// List of found T types
/// </returns>
public static IEnumerable<T> GetAvailableTypes()
{
    List<T> foundTypes = new List<T>();

    Assembly asm = Assembly.GetExecutingAssembly();
    FileInfo fi = new FileInfo(asm.Location);

    //get all DLL assemblies from current directory
    List<FileInfo> fileList = new List<FileInfo>();
    string[] filters = fileFilters.Split(';');
    foreach (string filter in filters) {
        fileList.AddRange(fi.Directory.GetFiles(filter));
    }

    foreach (FileInfo fiItem in fileList) {
        //load each found file and look up for IDataIO types
        Assembly tmpAsm = Assembly.LoadFile(fiItem.FullName);
        Type[] definedTypes = tmpAsm.GetTypes();
        foreach (Type type in definedTypes) {
            try {
                //try to create a IDataIO instance
                T dataIO = Activator.CreateInstance(type) as T;
                if (dataIO != null)
                    foundTypes.Add(dataIO);
            } catch {
                //do nothing!
            }
        }
    }

    return foundTypes;
}

```

DeepFileSearcher by Atilla Kati ©2011

```

Usage:
DeepFileSearcher basePath FileSearchFilter [actions]

basePath:      Start path to search
FileSearchFilter:  File search pattern (eg: *.txt)

-ver:          Tries to set the value of AssemblyVersion() and Assembly-
               tags within found files
-disp:         Displays file attributes in a listview if no further param-
               eter sets the attribute.
-lc:           Lower-case clears, upper-case sets the corresponding fil-
               e available flags: <A> = ReadOnly, <B> = Archive, <H> = Hidden
-gui:          Starts the graphical editor (GUI)
-changepath:   Change the target platform of Visual Studio CS project
Usage: -changepath:vsdk\jdk1.6\src\
Remove all found duplicate filenames from list.
-disp:         Displays found files matching search settings.
-copy:        Copy files to a destination folder. eg: -copy:dst\temp
-delete:       Delete all files found regarding defined filter settings
-pathfilter:   Found file path have to contain the folderFilter. eg:
               -pathfilter:src
-rename:       Rename found file extensions. eg: -extRename:tmp
-mail:        Sends the found file list to mail receiver.
               eg: -mail:atilla.kati@tut.by
-clip:         zips/unzips a file into file's defined directory. eg: -zip:src\temp
               /z Add files into new ziparchive. Usage: -zip:src\temp\newzip.zip
               /u Unzip file into directory. Usage: -zip:src\temp


```

Zeitdauer Berechnung

Start- und Endzeit für die Dauerberechnung eingeben:

Startzeit (hh:mm:ss):

Endzeit:



C# Grundlagen

Kursleiter: Atilla Kati

- Grundlagen der Programmierung in C#
- OOP Konzept
- Anwendungen in C#
- *selbstständiges Aneignen von Wissen*



C# Grundlagen – Zeitplanung

- 84 Stunden (21 Abende) insgesamt
 - 1 Abende Grundlagen Visual Studio 2019
 - 6 Abende C# Grundlagen
 - 6 Abende Einstieg in die OOP in C#
 - 8 Abende Architektur & GUI (Forms)
 - Beispiel Playlist Editor
 - Design Patterns, REST Services, Generics, Lambda, Linq, Xml/Json



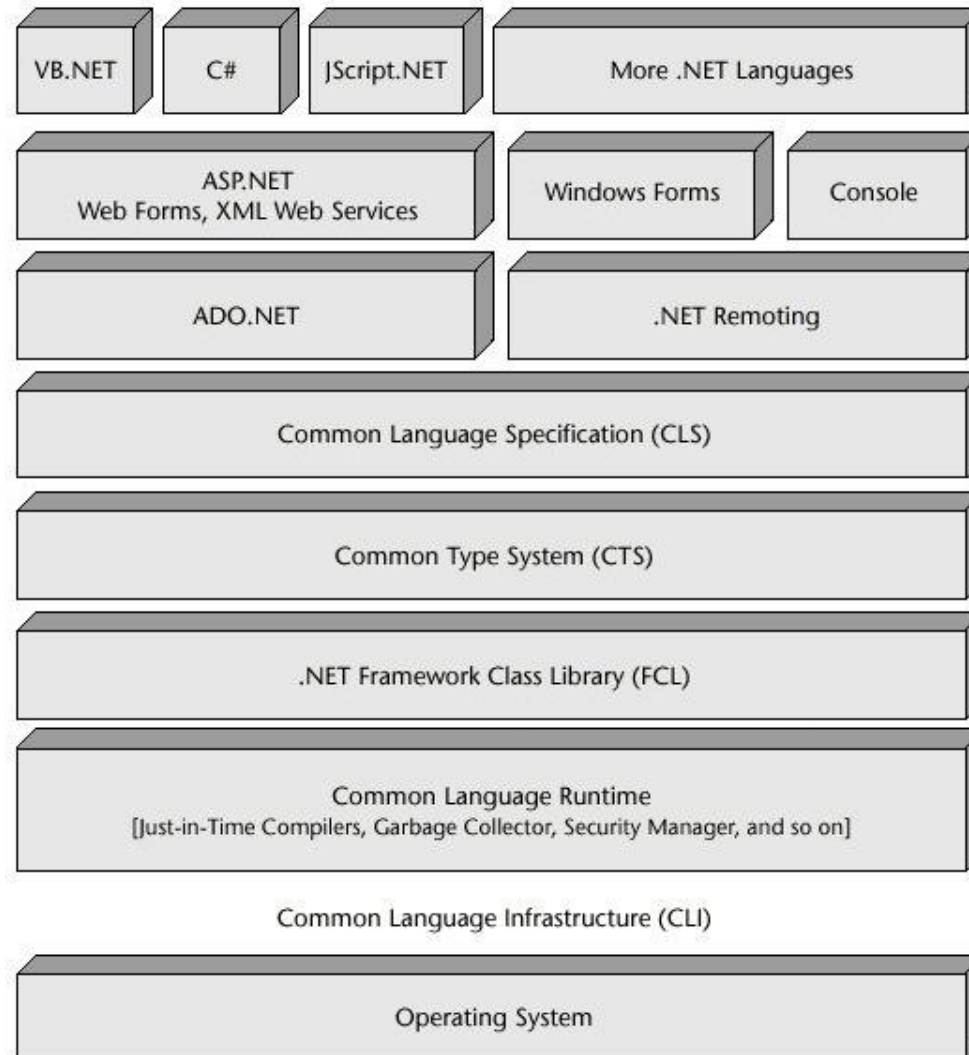
C# Grundlagen – Tools

- Unterlagen
 - MSDN & Google
 - eBook Gallery for Microsoft Tech ([eBook Gallery for Microsoft Tech](#))
 - Objektorientierte Programmierung ([Objektorientierte Programmierung](#))
 - **Visual C# 2010** ([Visual C# 2010](#))
 - **Buch zum Unterricht**
- Compiler
 - VS 2022 Community Edition ([VS2019 Community Edition Download](#))

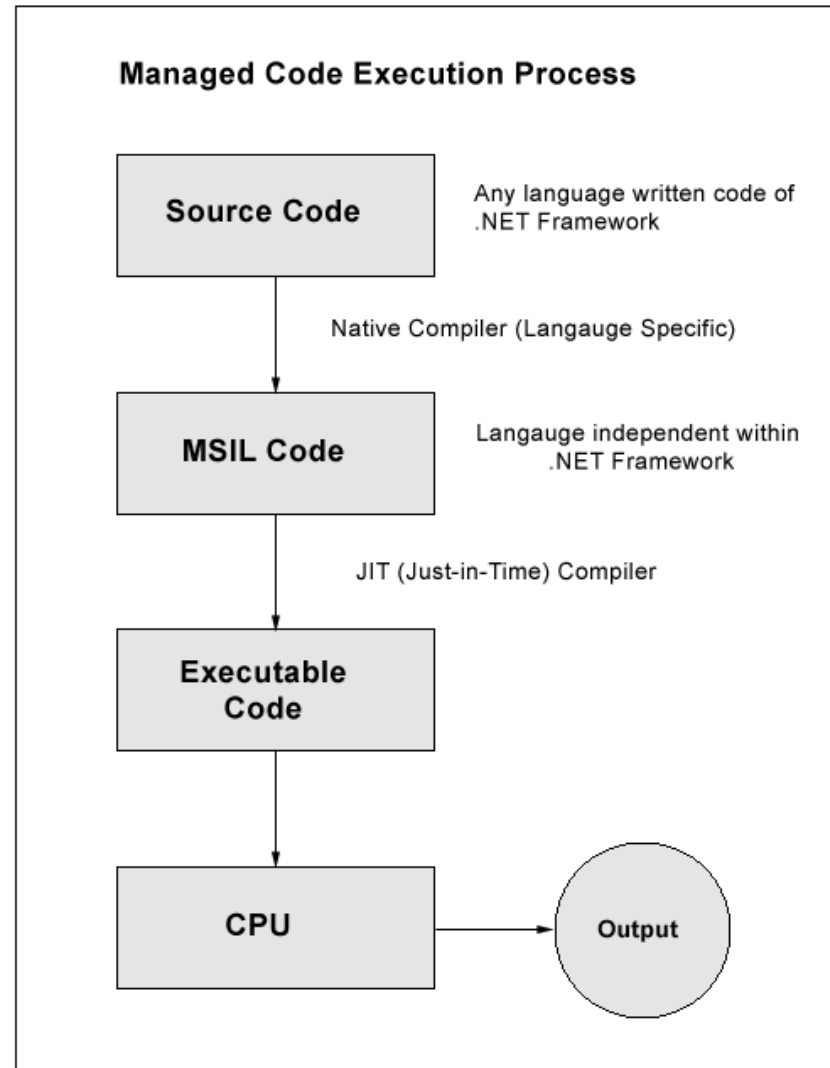
Editieren, Kompilieren, Ausführen



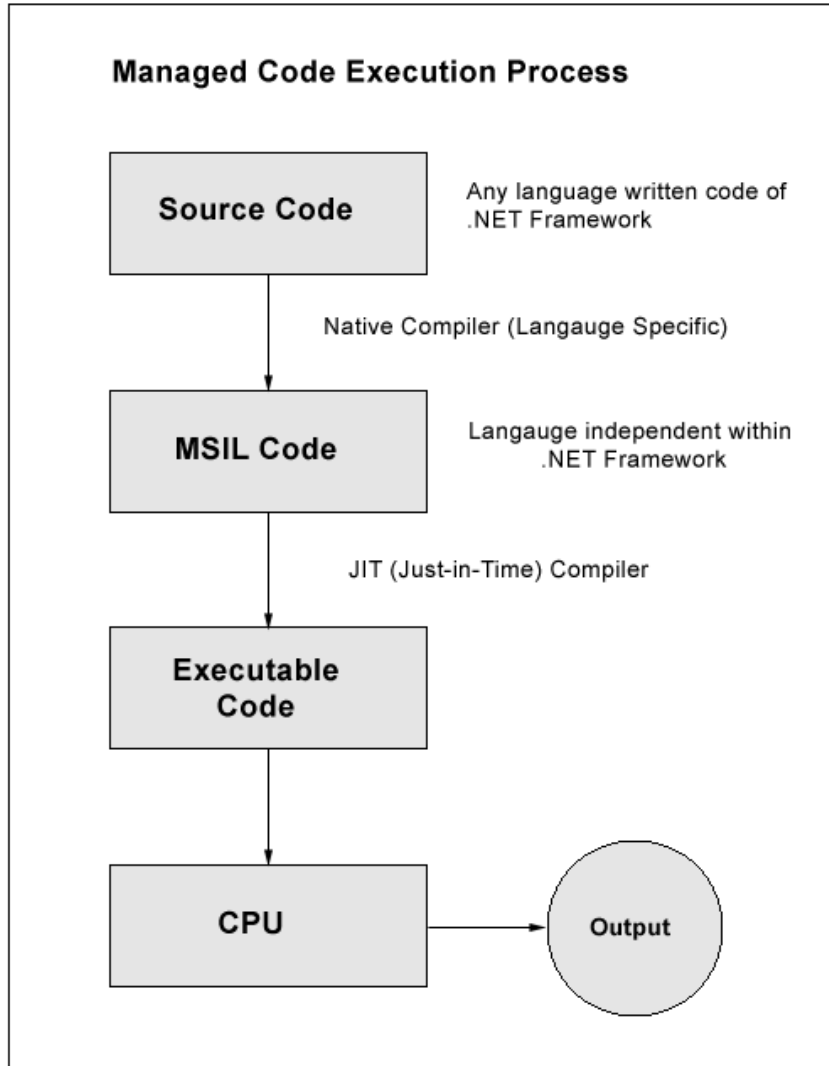
Managed Code / .NET



Managed Code / .NET



Managed Code / .NET



HelloWorld.cs mit Editor erstellen

HelloWorld.cs mit CSC.exe
compilieren

HelloWorld.exe mit ILDASM.exe
analysieren

HelloWorld.exe in der Console
ausführen

Konsolenausgabe



*Console.WriteLine()
Console.Write()*

Die Daten



Operatoren



Buch S. \geq 160

Strings



„\nHallo Leute {0} \n!\n“

Konsoleneingabe



Console.ReadLine()

Console.Read()

Console.ReadKey()

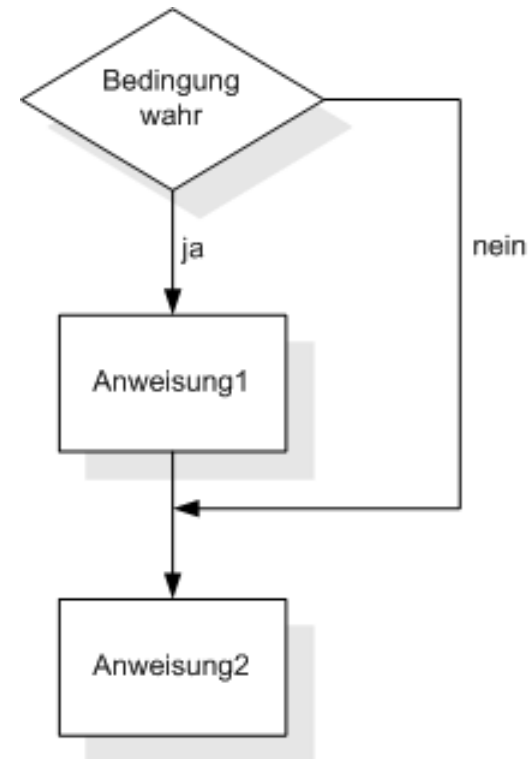
Fehlerbehandlung



- If Anweisung

```
if (BEDINGUNG==wahr)  
{  
    Anweisung1;  
}
```

```
Anweisung2;
```



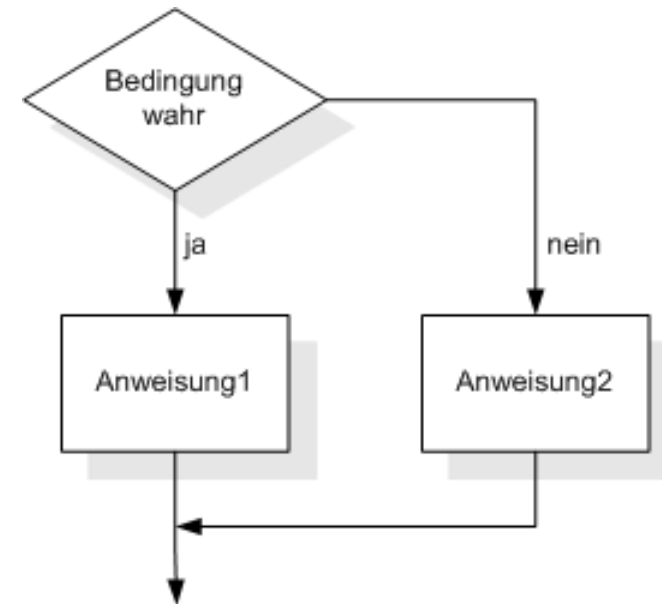
- Vergleichsoperatoren

<i>Vergleichsoperator</i>	<i>Bedeutung</i>
$a < b$	<i>Wahr, wenn a kleiner als b</i>
$a \leq b$	<i>Wahr, wenn a kleiner oder gleich b</i>
$a > b$	<i>Wahr, wenn a größer als b</i>
$a \geq b$	<i>Wahr, wenn a größer oder gleich b</i>
$a == b$	<i>Wahr, wenn a gleich b</i>
$a != b$	<i>Wahr, wenn a ungleich b</i>

- If Anweisung

```
if(BEDINGUNG1==wahr)  
{  
    Anweisung1;  
}  
else  
{  
    Anweisung2;  
}
```

Anweisungen;



- switch – case Anweisung

```
switch(AUSDRUCK)  
{  
    AUSDRUCK_1 :  
        anweisung_1;  
        break;  
    AUSDRUCK_2 :  
        anweisung_2;  
        break;  
    ...  
  
    default:  
        Anweisung3;  
        break;  
}
```

- Iterationskonstrukte
 - *for Schleife*
 - *while Schleife*
 - *do-while Schleife*
 - *foreach / in Schleife*

- For Schleife

```
for(Initialisierung; Bedingung; Reinitialisierung)  
{  
    /* Anweisungen */  
}
```

- while Schleife

```
while(Bedingung == wahr)  
{  
    //Abarbeiten von Befehlen bis Bedingung ungleich wahr  
}
```

- Topics

- Bedingungen mit bool Variable

```
if(inputIsValid)
{
    ...
}
```

- Konstanten

```
const int MAX_AGE = 150;
```

Arrays (S. 98)



Bezeichnung	Wert
myArray[0]	321
myArray[1]	12
myArray[2]	4
myArray[3]	0
myArray[4]	54

- foreach Schleife

```
using System;
class Class1
{
    static void Main(string[] args)
    {
        string[] aBooks = new string[] { "C# und die .NET Plattform",
                                          "Die Kunst zu programmieren",
                                          "ASP.NET Kochbuch",
                                          "Softwareentwicklung in C#" };

        foreach (string s in aBooks)
        {
            Console.WriteLine("Buchtitel: {0}", s);
        }
    }
}
```


Strukturen



Übung

- Aufzählungen

```
enum DayOfWeek
{
    Montag = 1,
    Dienstag,
    Mittwoch,
    Donnerstag,
    Freitag,
    Samstag,
    Sonntag
}
```

Ungarische Notation



Microsoft
Kodierungs Standards

„Methoden“



Stack & Heap Werttypen & Verweistypen



- Unterschiede zw. Wert & Referenztypen
 - *Parameterübergabe*
 - *Vergleiche*
 - *Der Wert null*
 - *Initialisierung*

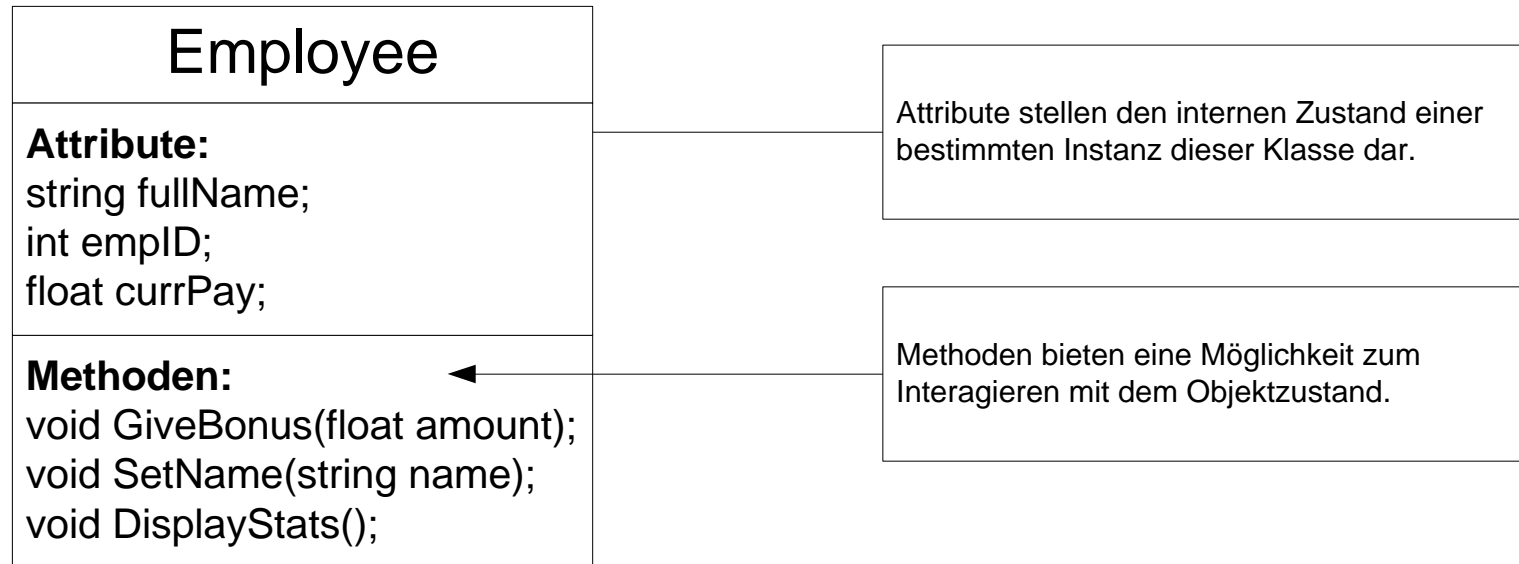
- Ich == Du oder Ich != Du
 - *Werttypen*
 - *Werttypen sind numerische Datentypen(int, float, Strukturen usw.)*
 - *Werttypen werden im Stack (Stapel) zugewiesen und unabhängig voneinander bearbeitet*
 - *Wird ein Wert einem anderen Wert zugewiesen, wird eine bitweise Kopie erstellt.*
 - *Verweistypen*
 - *Sind Klassen und Schnittstellen und werden auf dem Heap reserviert*
 - *Kopien eines Verweistypes führen zu einer „flachen Kopie“, d.h. mehrere Verweise zeigen auf dieselbe Speicheradresse*

Grundlagen



Kapitel 6

Eine einfache Klassendefinition



- Öffentliche std. Schnittstellen
 - *Unter öffentliche Member versteht man jene Klassenmember, die von einer Objektinstanz aus erreichbar sind*
 - *Bei einer öffentlichen std. Schnittstelle handelt es sich um jenes Element in einer Klasse, dass mit dem Schlüsselwort „public“ deklariert wurde.*

Die Säulen der objektorientierten Programmierung

- *Kapselung*
- *Vererbung*
 - *Ist-Ein Beziehung*
 - *Hat-Ein Beziehung*
- Polymorphismus

Kapselungsdiens

Vererbung

Polymorphie

- Folgende Member können in einer Klasse aufgenommen werden:
 - *Methoden*
 - *Bestimmen das Verhalten in einer Klasse*
 - *Eigenschaften*
 - *Verborgene Veränderungs- und Zugriffsfunktionen*
 - *Felder*
 - *Öffentliche Daten/Variablen (nicht empfehlenswert)*
 - *Ereignisse*

- Feld
- Methode
- Instanz
- Was ist eine Instanzmethode?
- Was ist eine statische Methode?

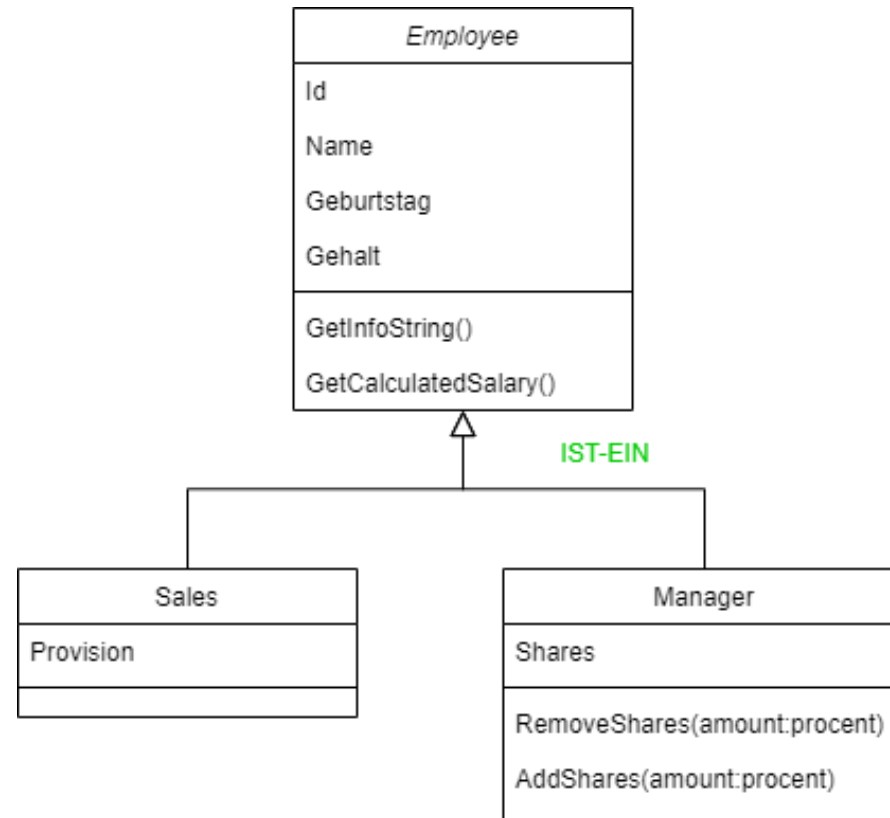
- 1. Säule: Kapselungsdienst
 - *Interne Daten eines Objekts sind von einer Objektinstanz aus nicht direkt zugänglich.*
 - *Kapselung wird mit den Schlüsselwörtern „public“, „private“ und „protected“ erzwungen.*
 - *Öffentliche Datenpunkte („public“) werden als Felder bezeichnet.*

- public
 - Felder, sprich öffentliche Datenpunkte, sollten generell vermieden werden, da die Elemente keine Kenntnisse darüber besitzen, ob der aktuell zugewiesene Wert den Regeln des Objekts entspricht.
 - BlackBox → die funktionalen Details sollten gegenüber der Aussenwelt verborgen werden
 - Spezielle *Zugriffs-* und *Änderungsmethoden* bzw. Klasseneigenschaften verwenden

- Klasseneigenschaften
 - *Werden verwendet um öffentlich zugängliche Datenpunkte(Felder) zu simulieren.*
 - *Sie bestehen aus einem Paar verborgener Methoden („get“, „set“).*
 - *„get“ und „set“ entsprechen immer echten Zugriffs- und Veränderungsmethoden.*
 - *Schreibgeschützte Eigenschaften erreicht man, indem man den „set“-Block weglässt.*

- 2. Säule: Vererbung
 - *Vererbung erleichtert die Wiederverwendung von Code*
 - *Man unterscheidet zwischen:*
 - *Klassischer Vererbung (Ist-Ein Beziehung)*
 - *Container/Delegate Methode (Hat-Ein Beziehung)*

- Klassische Vererbung



- Klassische Vererbung
 - *Neue Klassen nutzen die Funktionalität anderer Klassen und erweitern sie gegebenenfalls. Eine Klasse kann nur direkt eine Basisklasse besitzen. Mehrfachvererbung ist in C# nicht möglich!!!*
 - *Die Subklasse erbt alle öffentlichen Member der Basisklasse. Das Erweitern einer Klasse(Subklasse) wird mit dem „Doppelpunkt-Operator“ („:“) erreicht.*
 - *Jede Subklasse kann das Verhalten der Basisklasse erweitern.*

- Besonderheiten von Konstruktoren
 - *Ein Subklassenkonstruktor ruft automatisch immer den Standardkonstruktor der Basisklasse auf.*
 - *Dies ist zwar technisch zulässig, wenngleich nicht optimal.*

Warum?

Es wird zuerst der Standardkonstruktor der Basisklasse aufgerufen, bevor die Logik des Subklassenkonstruktor ausgeführt wird.

- Zauberwort „***base***“

```
public class Manager : Employee
{
    private ulong numOfOptions;

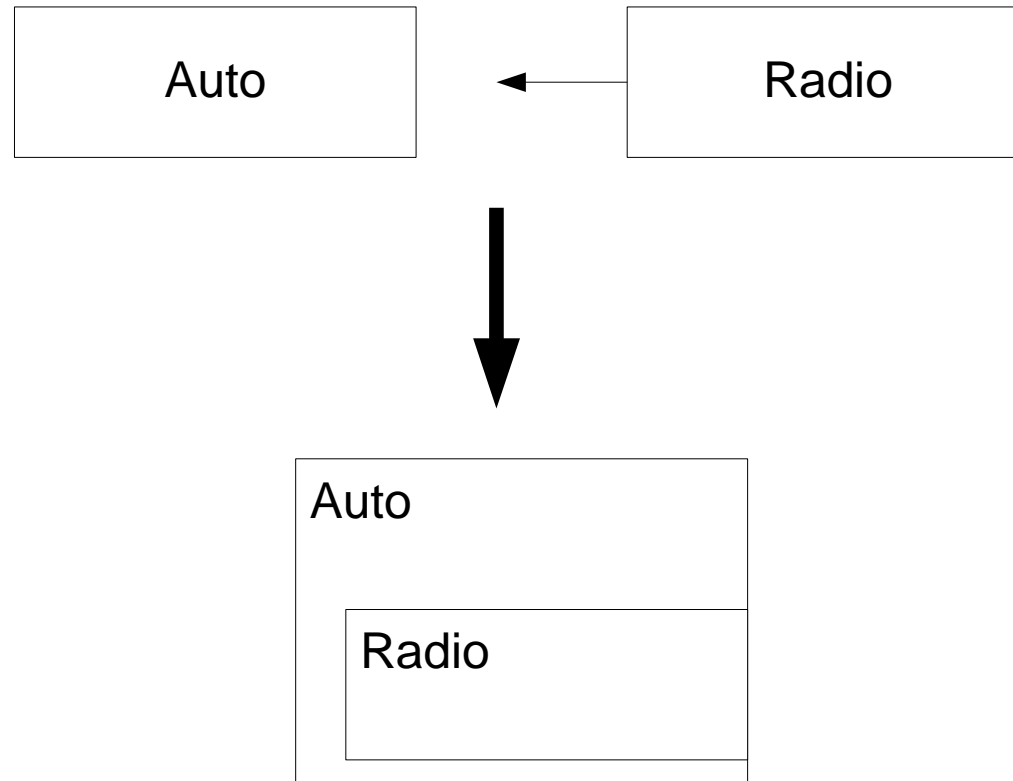
    public Manager(string name, int ID,
        double currPay, ulong NumOfOptions) : base(name, ID, currPay)
    {
        this.numOfOptions = NumOfOptions;
    }

    public ulong NumOpt
    {
        get{return this.numOfOptions;}
        set{this.numOfOptions = value;}
    }
}
```

- „*protected*“ – Geschützte Daten
 - *öffentliche Elemente sind direkt aus jeder Suklasse zugänglich*
 - *private Elemente sind nicht von Objekten außerhalb des Objektes erreichbar.*
 - *geschützte Elemente sind nur direkt aus abgeleiteten Klassen erreichbar, können von Objekten außerhalb des Objektes jedoch nicht erreicht werden.*

- „*sealed*“ – Versiegelte Klassen
 - *Von versiegelten Klassen können keine weiteren Klassen abgeleitet werden.*
 - *Eignet sich besonders für das Erstellen von eigenständigen Hilfsklassen.*

- Container/Delegate Modell



- Container/Delegate Modell
 - *Das Container/Delegate Modell wird auch als „Hat-Ein“ Beziehung bezeichnet.*
 - *Basis sind min. 2 unabhängige Klassen („Auto“ und „Radio“), zw. denen eine Beziehung erstellt werden soll. Das Ziel ist der Abstrakte Ausdruck: Ein Auto hat ein Radio!*
 - *Eine Klasse die eine andere Klasse enthält nennt man „übergeordnete“ Klasse („Auto“). Die enthaltene Klasse wird als „untergeordnete“ Klasse bezeichnet („Radio“).*

- Verschachtelte Typdefinition
 - *Man spricht von verschachtelten Typdefinition, wenn innerhalb einer bestehenden Klasse, eine weitere Klasse definiert wird.*
 - *Vorteil liegt darin, dass die verschachtelte Typdefinition von der Aussenwelt nicht erreichbar ist (Hilfsklasse).*
 - *Der verschachtelte Typ kann sowohl als „private“ oder „public“ deklariert werden.*

- 3. Säule: Polymorphismus
 - *Polymorphismus bedeutet, dass vererbte Methoden einer Basisklasse in Subklassen überschrieben werden können.*
 - *Dies wird mit folgenden Schlüsselwörtern erreicht:*
 - „virtual“

*Wird eine Methode in der Basisklasse mit „virtual“ festgelegt,
kann diese in einer beliebigen Subklasse überschrieben werden.*
 - „override“

*In der Subklasse kann eine „virtuelle“ Methode der Basisklasse
mit „override“ neu implementiert werden.*

- Abstrakte Klassen
 - *Abstrakte Klassen, definieren im Prinzip den „Bauplan“, d.h. Standardzustandsdateien und –verhalten (Methoden) für Subklassen.*
 - *Von abstrakten Klassen können direkt keine Instanzen erzeugt werden.*
 - *Abstrakte Klassen können mit dem Schlüsselwort „abstract“ festgelegt werden.*
 - *Eine abstrakte Klasse kann eine beliebige Anzahl von abstrakten Members definieren.*

- Abstrakte Klassen
 - *Durch abstrakte Member, wird quasi ein reines polymorphes Verhalten aller abgeleiteter Typen (Methoden) erzwungen.*
 - *Für jede Subklasse muss eine spezielle Implementierung (override) definiert werden, da in der Basisklasse, aufgrund der abstrakten Member, keine Standardimplementierung möglich ist.*
 - *Abstrakte Member (Methoden) haben keine „geschwungenen“ ({...}) Klammern.*

OOP - Ausnahmebehandlung

- Ausnahmebehandlung
 - *Die .NET Plattform stellt ein einheitliches Verfahren zur Fehlerbehandlung zur Verfügung.*
 - *SEH – strukturierte Ausnahmebehandlung (Structured Exception Handling)*
 - *Das Konzept von SEH ist bei allen .NET Sprachen gleich, egal ob C#, VB.NET, C++ usw...*
 - *Ausnahmen sind echte Objekte, die von der Klasse „System.Exception“ abgeleitet werden.*



OOP - Ausnahmebehandlung

- Ausnahmebehandlung
 - *Ausnahmen können mit dem Schlüsselwort „throw“ explicit ausgelöst werden. Dabei muss eine Instanz der Exception-Klasse erstellt und konfiguriert werden.*

Beispiel:

throw new Exception(„Fehler!“);

- Hinweis:

Ausnahmen sollten generell nur dann ausgelöst werden, wenn ein definitiver Endstand erreicht wurde.

D.h. der Programmablauf nicht mehr weitergeführt werden kann.

OOP - Ausnahmebehandlung

- Mit dem ***try/catch/finally*** Block kann eine Ausnahme, die durch den Aufruf einer Methode auftreten kann, abgefangen werden.

```
public class mainAPP
{
    static void Main()
    {
        Auto vw = new Auto("Polo", 100, 0);
        vw.Musik(true);

        try
        {
            for(int i=0; i<7; i++)
                vw.SpeedUp(25);
        }
        catch(Exception e)
        {
            Console.WriteLine("MESSAGE: " + e.Message);
            Console.WriteLine("Stacktrace: " + e.StackTrace);
        }
    }
}
```


OO P - Ausnahmebehandlung

- Der „*try*“ Block kann mehrere „*catch*“ Blöcke enthalten, die die jeweilige „Ausnahme“ behandeln.
- Der „*finally*“ Block wird immer ausgeführt. Er dient vor allem dazu, dass reservierte Ressourcen freigegeben werden, auch dann, wenn eine Ausnahme den normalen Ablauf stört.

- Grundlagen
 - *Bei einer Schnittstelle handelt es sich lediglich um eine Auflistung semantisch verwandter abstrakter Methoden!*
 - *Eine Schnittstelle drückt ein Verhalten aus, dass eine Klasse unterstützen soll.*
 - *Schnittstellen bieten eine weitere Möglichkeit das Systemverhalten polymorph zu gestalten.*
 - *Schnittstellen definieren niemals Datentypen und stellen niemals eine Standard-Implementierung der Methoden zur Verfügung.*

- Grundlagen
 - *Schnittstellen werden mit dem Schlüsselwort „interface“ erstellt.*
 - *Schnittstellen können auch Eigenschaften und Ereignisse unterstützen.*
 - *Wegen den abstrakten Member der Schnittstelle, muss jede Klasse oder Struktur, die Details der einzelnen Member festlegen.*
 - *Jeder Member einer Schnittstelle ist automatisch „abstrakt“!*

- Zugriff auf Schnittstellenverweise
 - *1. Mittels einer expliziten Typumwandlung*

```
static void Main()  
{  
    Hexagon egon = new Hexagon("Egon");  
  
    IPointy itfPt = (IPointy) egon;  
    Console.WriteLine(itfPt.GetNumberOfPoints());  
}
```

*Wird auf eine von der Klasse nicht unterstützte Schnittstelle zugegriffen, wird eine „**InvalidCastException**“-Ausnahme ausgelöst.*

- Zugriff auf Schnittstellenverweise
 - *2. Mittels dem Schlüsselwort „as“*

```
static void Main()
{
    Hexagon hex = new Hexagon("Hexagon");
    IPointy itfPt;

    itfPt = hex as IPointy;
    if(itfPt != null)
        Console.WriteLine("Anzahl der Punkte: " + itfPt.GetNumberOfPoints());
    else
        Console.WriteLine("OOOOPS! Hat keine Punkte!");
}
```

*Wird die Schnittstelle vom Objekt nicht unterstützt, dann legt der **as-Syntax** die Schnittstellenvariable auf **null** fest.*

- Zugriff auf Schnittstellenverweise
 - *3. Mittels dem Schlüsselwort „is“*

```
static void Main()
{
    Triangle t = new Triangle("Marianne");

    if(t is IPointy)
        Console.WriteLine("Anzahl der Punkte: {0}", t.GetNumberOfPoints());
    else
        Console.WriteLine("OOOOPS! Hat keine Punkte.");
}
```

- Zugriff auf Schnittstellenverweise
 - **3. Mittels dem Schlüsselwort „is“**

```
static void Main()
{
    //Sinnvolle Anwendung
    Shape[] myshapes = { new Hexagon("Hex"), new Kreis("Kreis"),
                        new Triangle("Trinagle"), new Oval("Oval")
                        };

    foreach(Shape shape in myshapes)
    {
        Console.WriteLine();
        shape.Draw();

        //Punkte vorhanden?
        if(shape is IPointy)
            Console.WriteLine("Punkte: {0}", ((IPointy) shape).GetNumberOfPoints());
        else
            Console.WriteLine(shape.Name + " hat keine Punkte.");
    }
}
```

- Schnittstellen als Parameter
 - *Schnittstellen sind streng typisierte Variablen, weshalb Methoden erstellt werden können, die Schnittstellen als Parameter oder Rückgabewerte verwenden.*
 - *Wenn eine Methode als Parameter eine Schnittstelle verlangt, können alle Objekte übergeben werden, die diese Schnittstelle unterstützen.*

- Benutzerdefinierte Auflistungen
 - *C# Typen implementieren eine Vielzahl von Standardschnittstellen. Für benutzerdefinierte Typen können diese Schnittstellen natürlich auch verwendet werden.*
 - *IEnumerable – Schnittstelle (System.Collections Namespace)
Damit bekommt der benutzerdefinierte Typ das Verhalten einer Aufzählung*
 - *IEnumerable definiert GetEnumerator(), welche in den benutzerdefinierten Typ implementiert werden muss. (siehe MSDN!)*

- Benutzerdefinierte Auflistungen
 - *„GetEnumerator()“ gibt eine weitere Schnittstelle namens IEnumerator zurück.*
 - *IEnumerator*
Auf IEnumerator kann von einem Objekt aus zugegriffen werden, um eine interne Auflistung von Typen zu durchlaufen.
 - *IEnumerator definiert 3 Member, die implementiert werden müssen:*
 - *MoveNext() (=Methode)*
 - *Current (= Eigenschaft)*
 - *Reset() (= Methode)*

Überladen von Operatoren

- Schlüsselwort *operator*
- Einfacher geht's nicht!

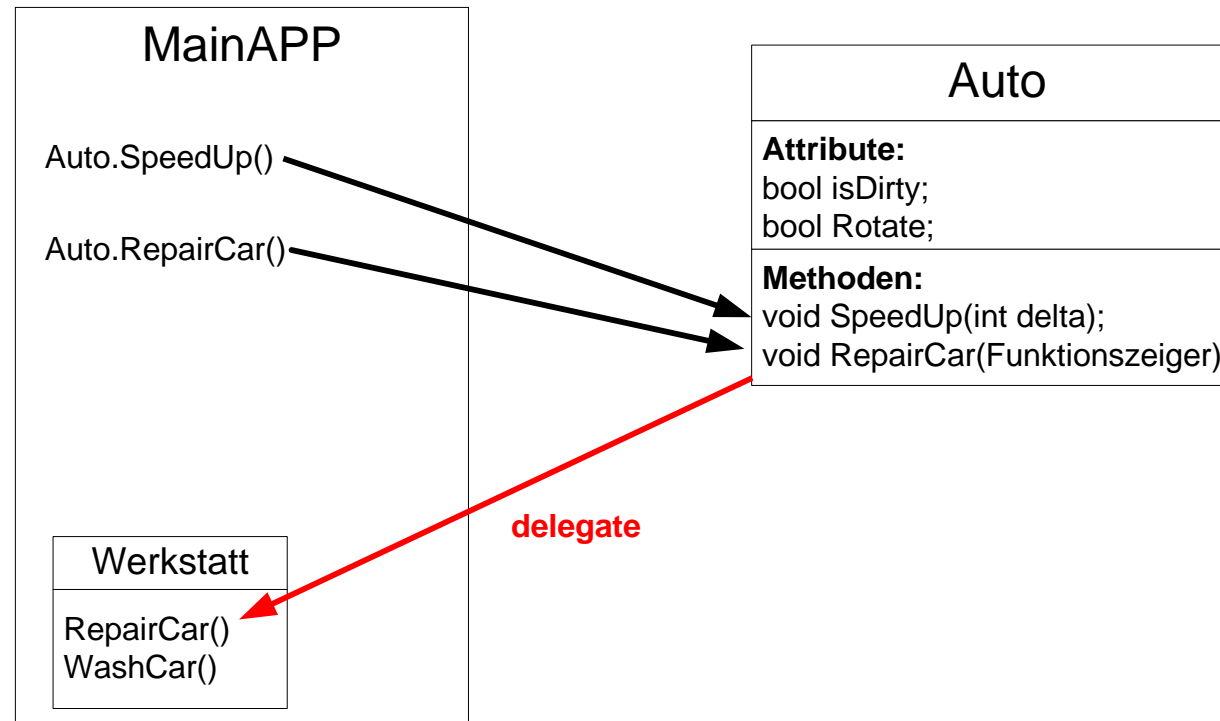
```
public class Punkt
{
    private int x,y;
    public Punkt(){}
    public Punkt(int xPos, int yPos)
    {
        this.x = xPos;
        this.y = yPos;
    }

    public override string ToString()
    {
        return "Xpos: " + this.x + " Ypos: " + this.y;
    }
}
```



- *delegates* → typisierte Rückruffunktionen
- Kommunikation zwischen Objekten
- Funktionen können so konfiguriert werden, daß diese eine weitere Funktion in der Anwendung aufrufen können!
- `public delegate void CarDelegate(Car c);`

- Wo setze ich *delegates* ein?



Übersicht Aufbau Themen

