

```

/// <summary>
/// Gets a list with available T types.
/// Loads all assemblies from executing application directory
/// 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<T>()
{
    List<T> foundTypes = new List<T>();

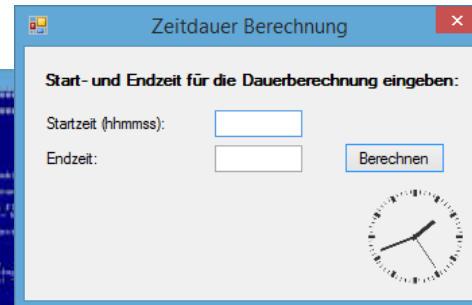
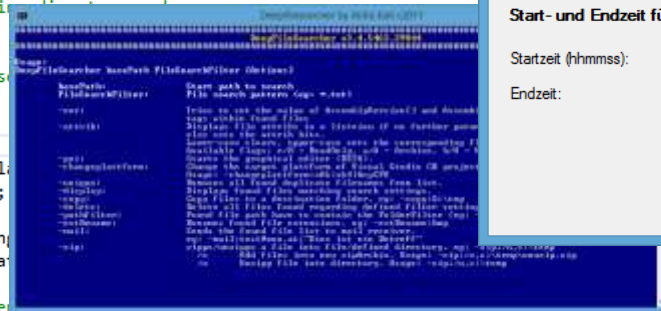
    Assembly asm = Assembly.GetExecutingAssembly();
    FileInfo fi = new FileInfo(asm.Location);

    //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 fileItem in fileList) {
        //load each found file and look up for IDataIO types
        Assembly tmpAsm = Assembly.LoadFile(fileItem.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;
}

```



C# Basics

Tutor: Atilla Kati

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

C# Basics – Time schedule

- 88 Stunden (22 Abende) insgesamt
 - 1 Abende Grundlagen Visual Studio 2019
 - 6 Abende C# Grundlagen
 - 6 Abende Einstieg in die OOP in C#
 - 9 Abende Architektur & GUI (Forms)
 - Design Patterns, REST Services, Generics, Lambda, Linq, Xml/Json

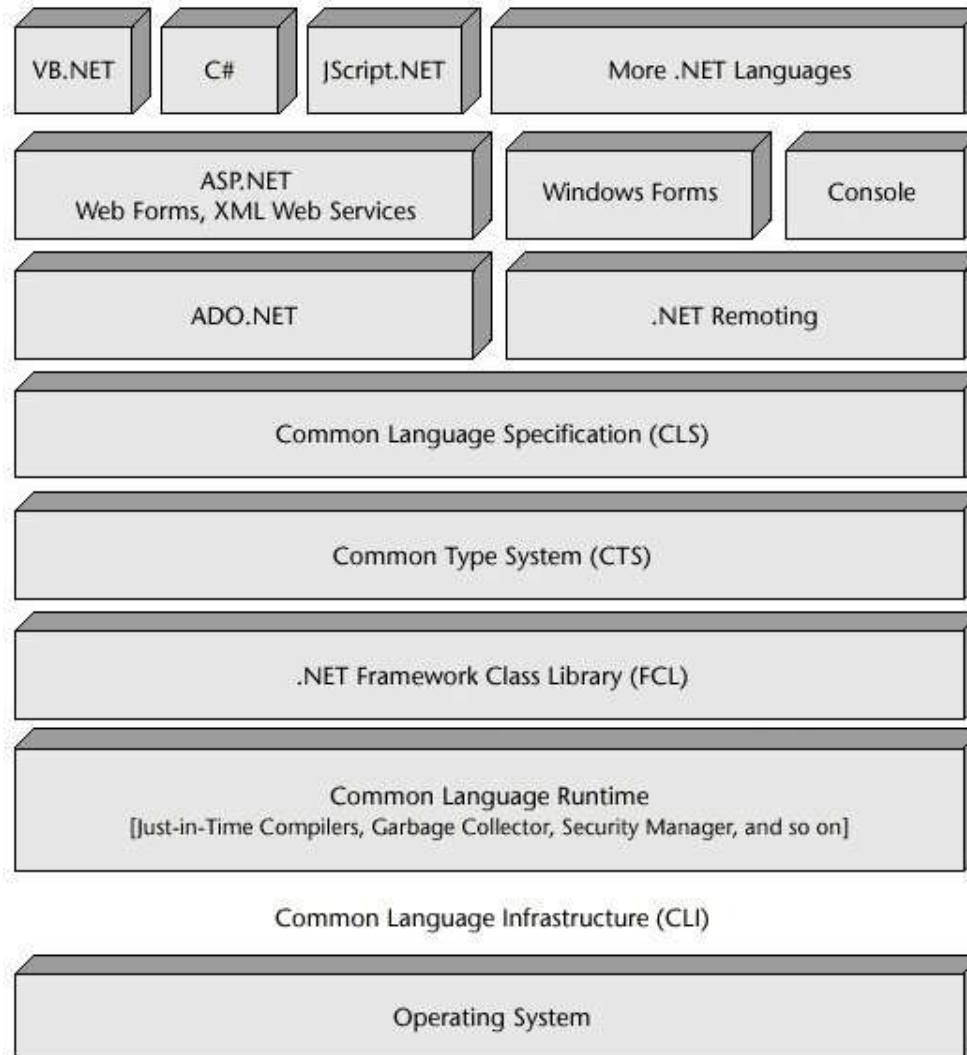
- Unterlagen
 - MSDN & Google
 - eBook Gallery for Microsoft Tech ([eBook Gallery for Microsoft Tech](#))
 - Objektorientierte Programmierung ([Objektorientierte Programmierung](#))
 - **Visual C# 2010** ([Visual C# 2010](#))
- Compiler
 - VS 2019 Community Edition ([VS2019 Community Edition Download](#))

Indroduction into C#

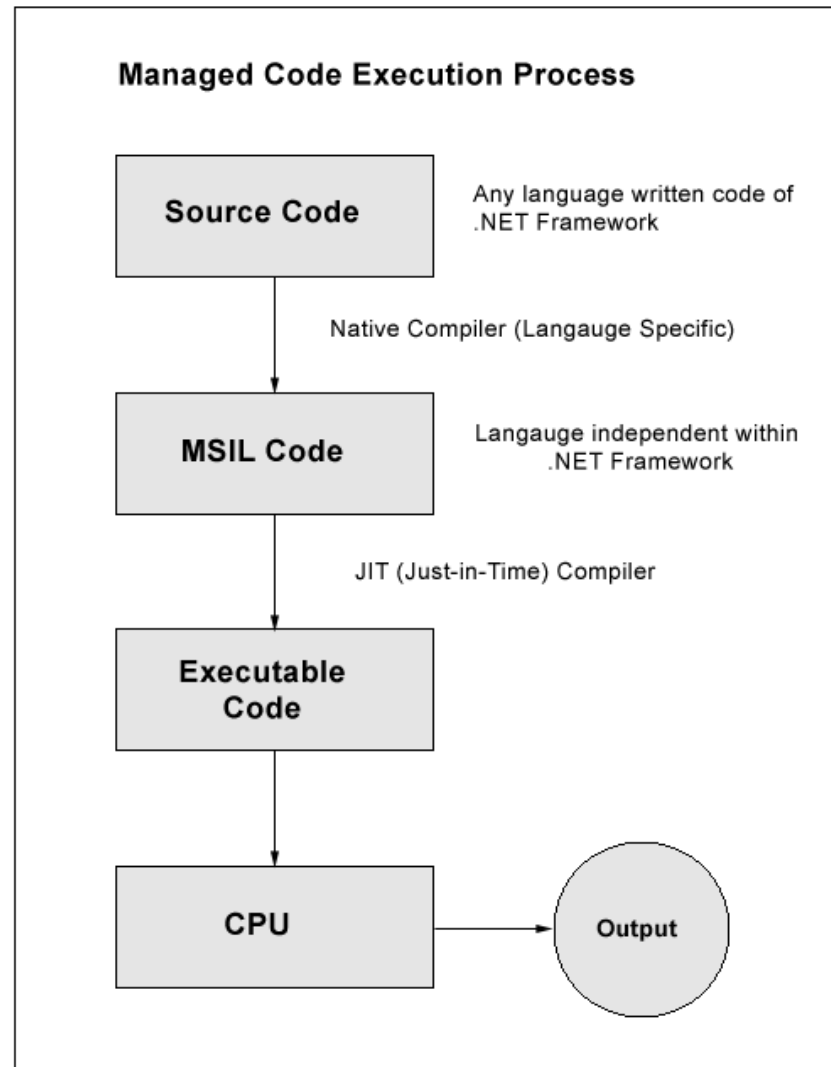
Edit, Compile, Execute



Managed Code / .NET

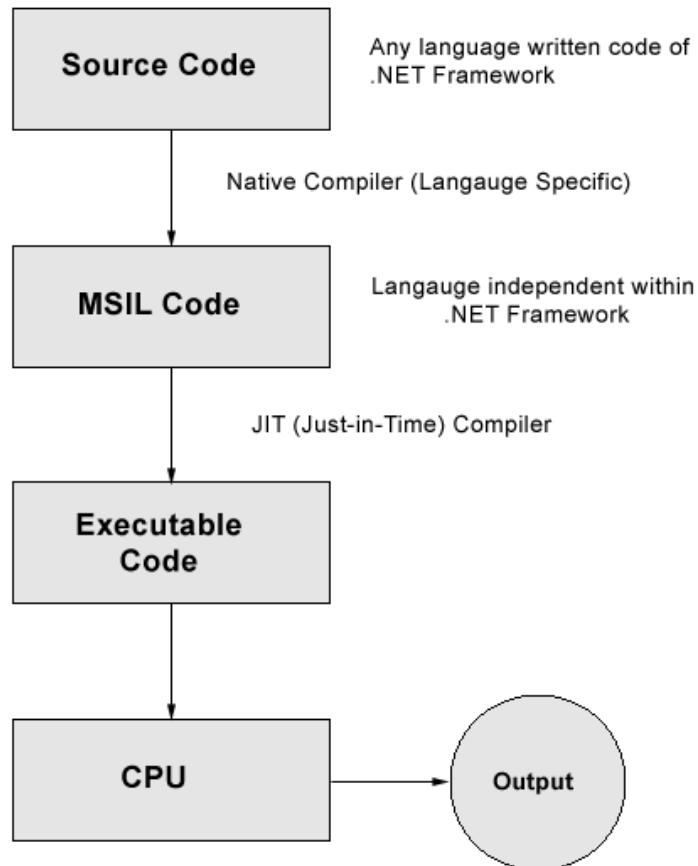


Managed Code / .NET



Managed Code / .NET

Managed Code Execution Process



Create HelloWorld.cs using
Windows text editor

Compile HelloWorld.cs using
CSC.exe

Open HelloWorld.exe using
ILDASM.exe tool

Execute HelloWorld.exe in
Console

Introduction into C#

Console outputs



Console.WriteLine()
Console.Write()

Introduction into C#

Working with data



Introduction into C#

Operators



Introduction into C#

Strings



„\nHi folks, {0} \a\n!“

Introduction into C#

Console inputs



Console.ReadLine()

Console.Read()

Console.ReadKey()

Introduction into C#

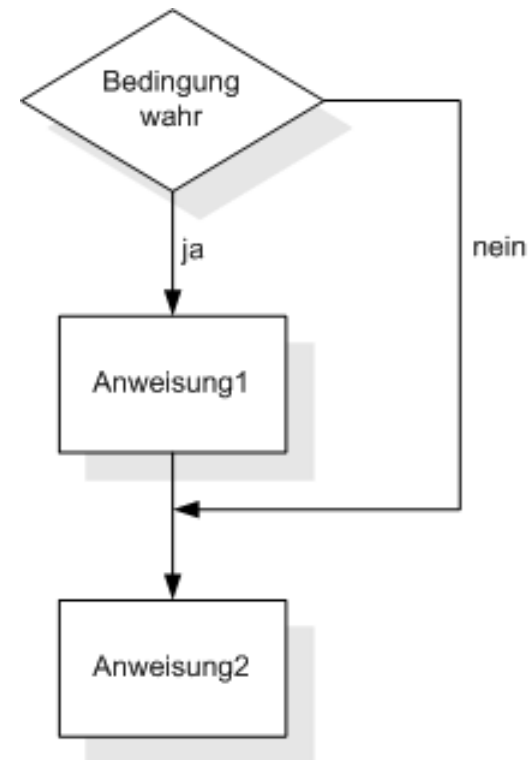
Exception handling



- If conditions

```
if (CONDITION==true)  
{  
    Line of code;  
    Line of code;  
    ...  
}
```

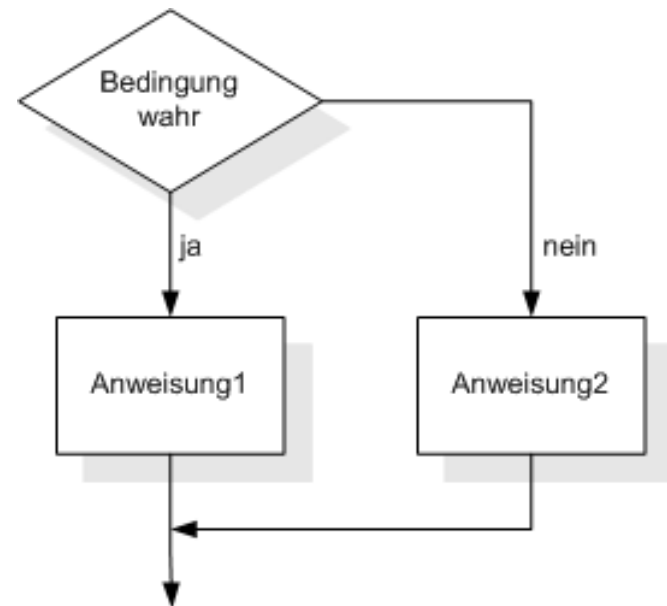
Other line of code;



- If conditions

```
if(CONDITION == true)
{
    Line of code;
    Line of code;
    ...
}
else
{
    Line of code;
}
```

Other line of code;



- Comparative operators

<i>Operator</i>	<i>Description</i>
$a < b$	True if a is smaller than b
$a \leq b$	True if a is less than or equal to b
$a > b$	True if a is greater than b
$a \geq b$	True if a is greater than or equal to b
$a == b$	True if a equals b
$a != b$	True if a is not equal to b

- switch – case statement

```
switch(expression)  
{  
    EXPRESSION_VALUE_1:  
        line of code;  
        line of code;  
        break;  
    EXPRESSION_VALUE_2:  
        line of code;  
        break;  
    ...  
    default:  
        line of code;;  
        break;  
}
```

- Iteration constructs
 - *for iteration*
 - *while iteration*
 - *do-while iteration*
 - *foreach iteration*

- for iteration

```
for (Init; Condition; Reinit;)
{
    /*
        Statements
    */
}
```

- while iteration

```
while (condition == true)
{
    /*
        Statements
    */
}
```

- Conditions with boolean variables

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

- Constants

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

Introduction into C#

Arrays



Name	Value
myArray[0]	321
myArray[1]	12
myArray[2]	4
myArray[3]	0
myArray[4]	54

- foreach iteration

```
using System;

class Demo
{
    static void Main(string[] args)
    {
        string[] books = new string[] {"C# and the .NET platform",
                                         "The art of programming",
                                         "ASP.NET Cookbook",
                                         "Software development in C#"};

        foreach(string book in books)
        {
            Console.WriteLine("Book title: {0}", book);
        }
    }
}
```


Introduction into C#

Structs



Name	<input type="text"/>
City	<input type="text"/>
Birthday	<input type="text"/>
Postal code	<input type="text"/>

- Enumerations

```
enum DayOfWeek
{
    Monday = 1,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday,
    Sunday
}
```

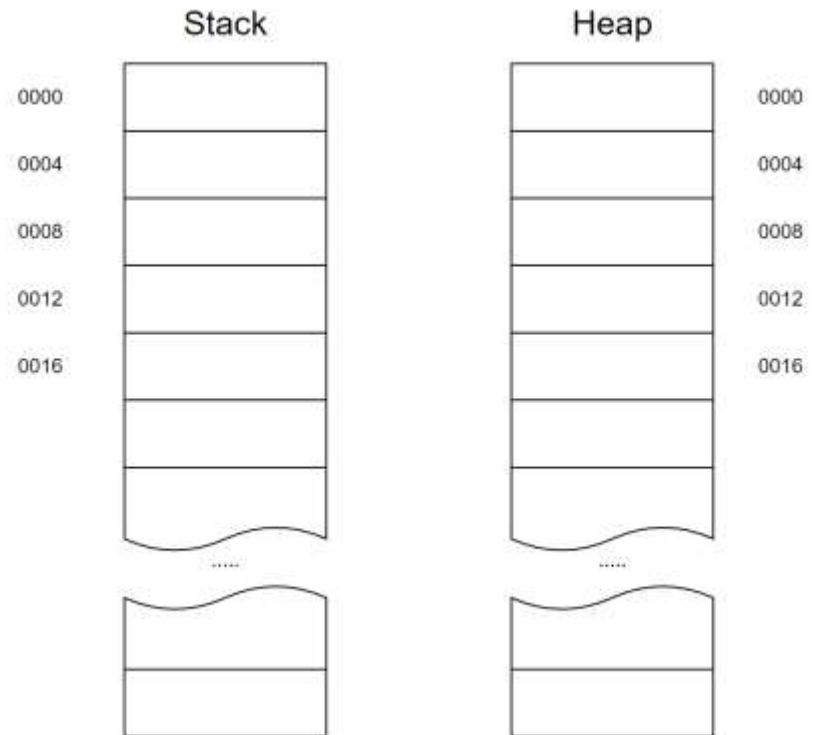
Introduction into C#

Methods



Introduction into C#

Value- and reference types



- Differences between value & reference types
 - *Parameter transfer*
 - *Comparisons*
 - *The value null*
 - *Initialization*



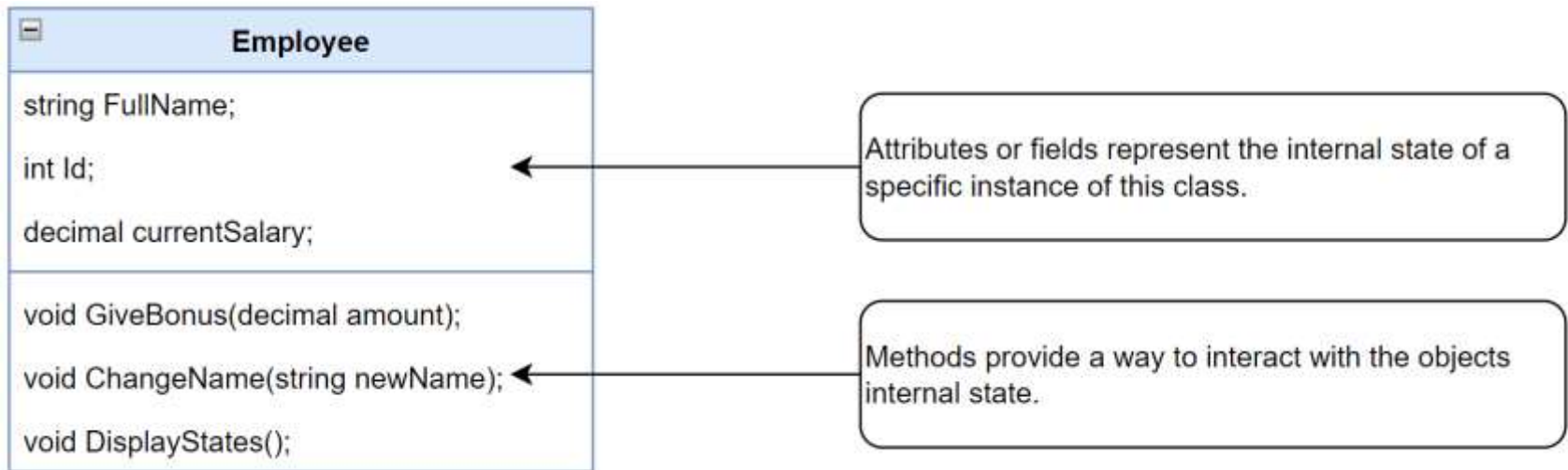
Introduction into C#

- To wrap up things...
 - *Value types*
 - Value types are numerical data types (int, float, structures, etc.)
 - Value types are assigned in the stack and processed independently of each other
 - If a value is assigned to another value, a bit-by-bit copy is created.
 - *Reference types*
 - Are classes and interfaces and are reserved on the heap
 - Copies of a reference type lead to a “flat copy”, i.e. several references point to the same memory address

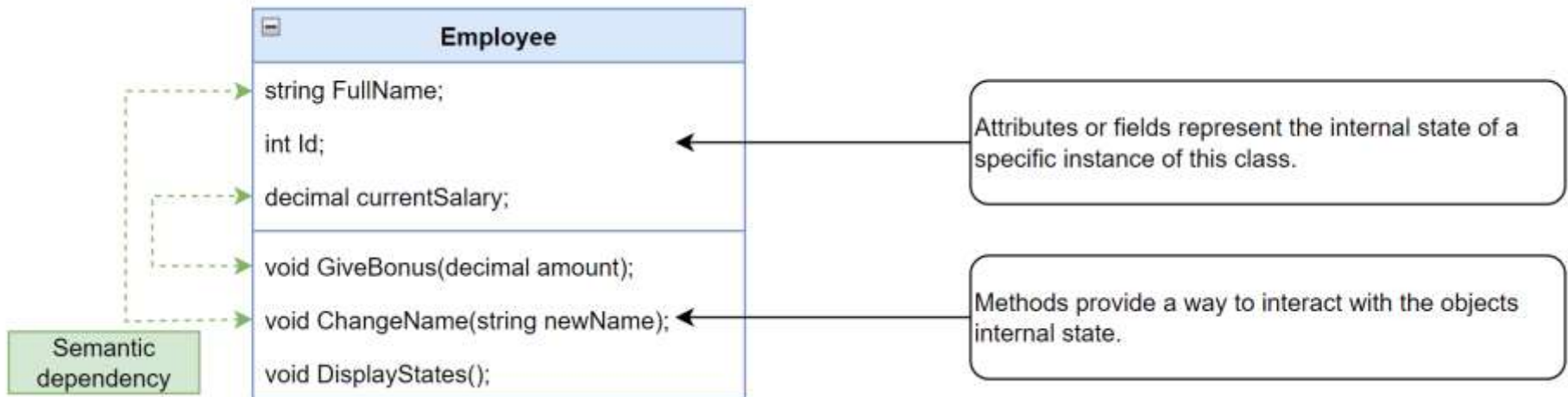
Basics



A simple class definition



A simple class definition



- public class interface
 - Part of the public interface are the elements in a class, which was declared with the keyword “public”.



The following members can be part of a class:

- **Methods**
 - Determine the behavior in a class
- **Properties**
 - Hidden read and write functions
- **Fields**
 - Public data/variables (not recommended)
- **Events**



Introduction to C# - Dictionary

- Field
- Method
- Instance vs. Object vs. Class

- What is an instance method?
- What is a static method?

The pillars of object-oriented programming

- Encapsulation
- Inheritance
 - *Is-One relationship*
 - *Has-One relationship*
- Polymorphism

Kapselungsdienst

Vererbung

Polymorphie

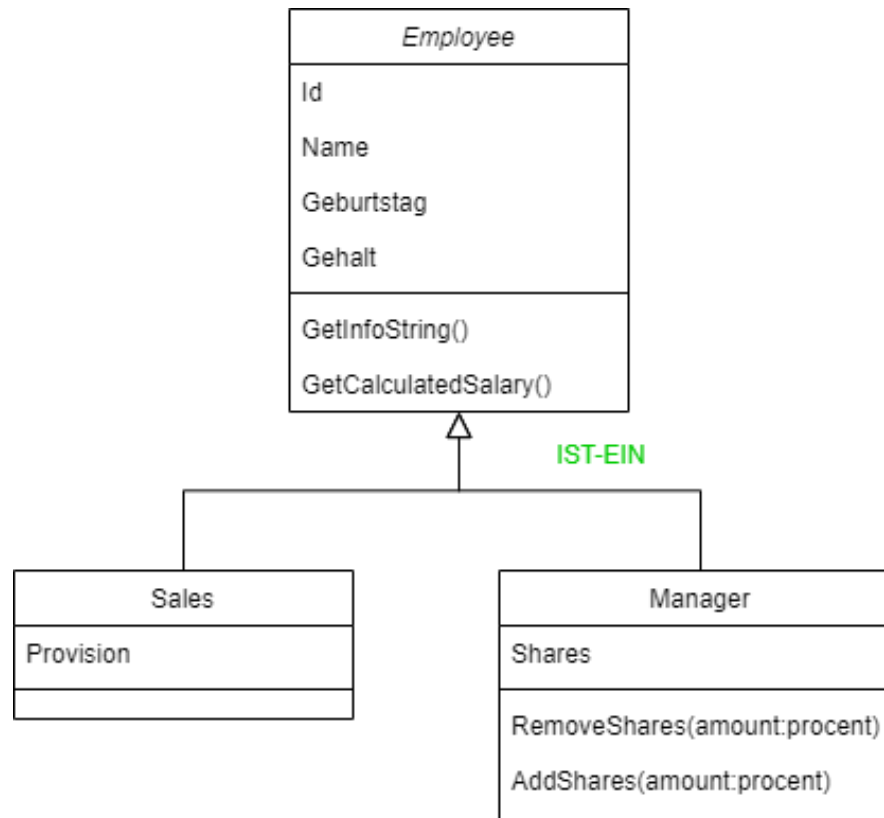
- 1. Säule: Kapselungsdienst
 - *Interne Daten eines Objekts sind von einer Objektinstanz aus nicht direkt zugänglich.*
 - *Kapselung wird mit den Schlüsselwörtern „public“, „private“ und „protected“ erzwungen.*
 - *Öffentliche Datenpunkte („public“) werden als Felder bezeichnet.*

- 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;

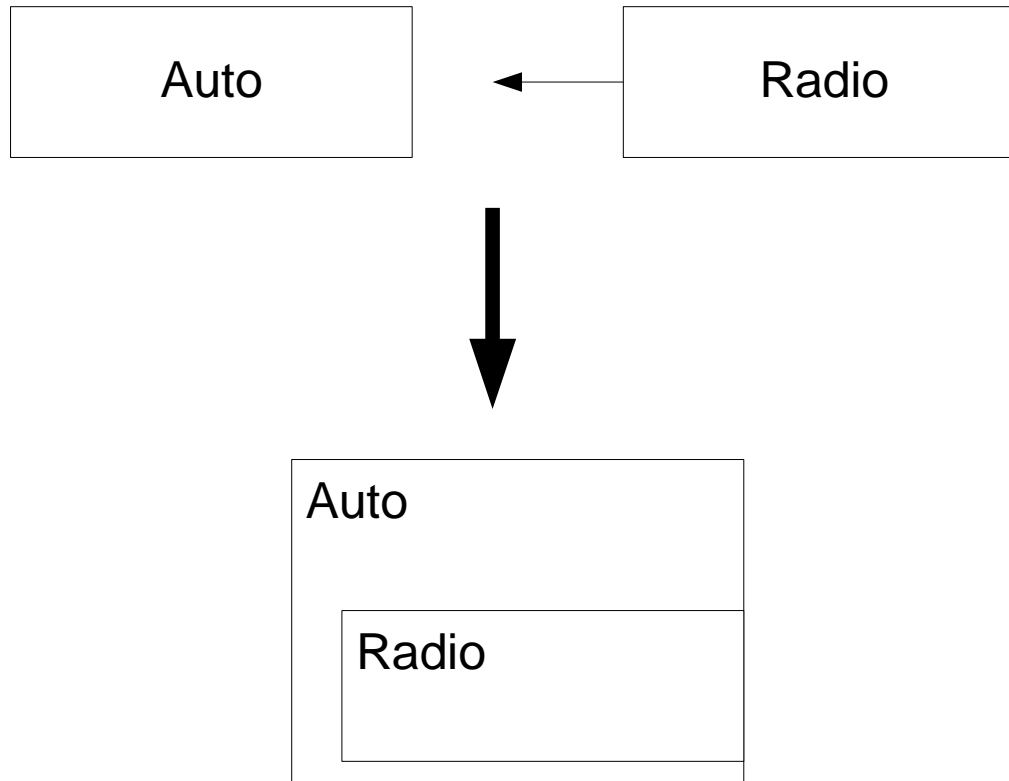
    public Manager(string name, int ID,
        double currPay, ulong NumOfOptions) : base(name, ID, currPay)
    {
        this.numOfOptions = NumOfOptions;
    }

    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. Standardzustandsdateen 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!)*

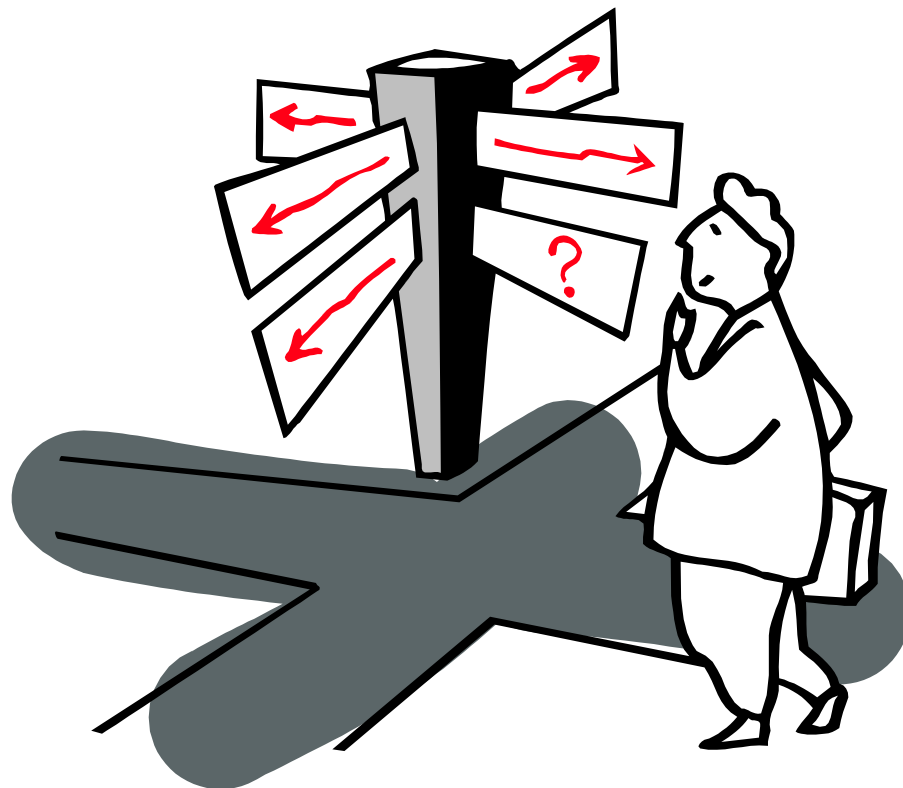
- Benutzerdefinierte Auflistungen
 - *„GetEnumerator()“ gibt eine weitere Schnittstelle namens IEnumerator zurück.*
 - *IEnumerator*
Auf IEnumerator kann von einem Objekt aus zugegriffen werden, um eine interne Auflistung von Typen zu durchlaufen.
 - *IEnumerator definiert 3 Member, die implementiert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

Überladen von Operatoren

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

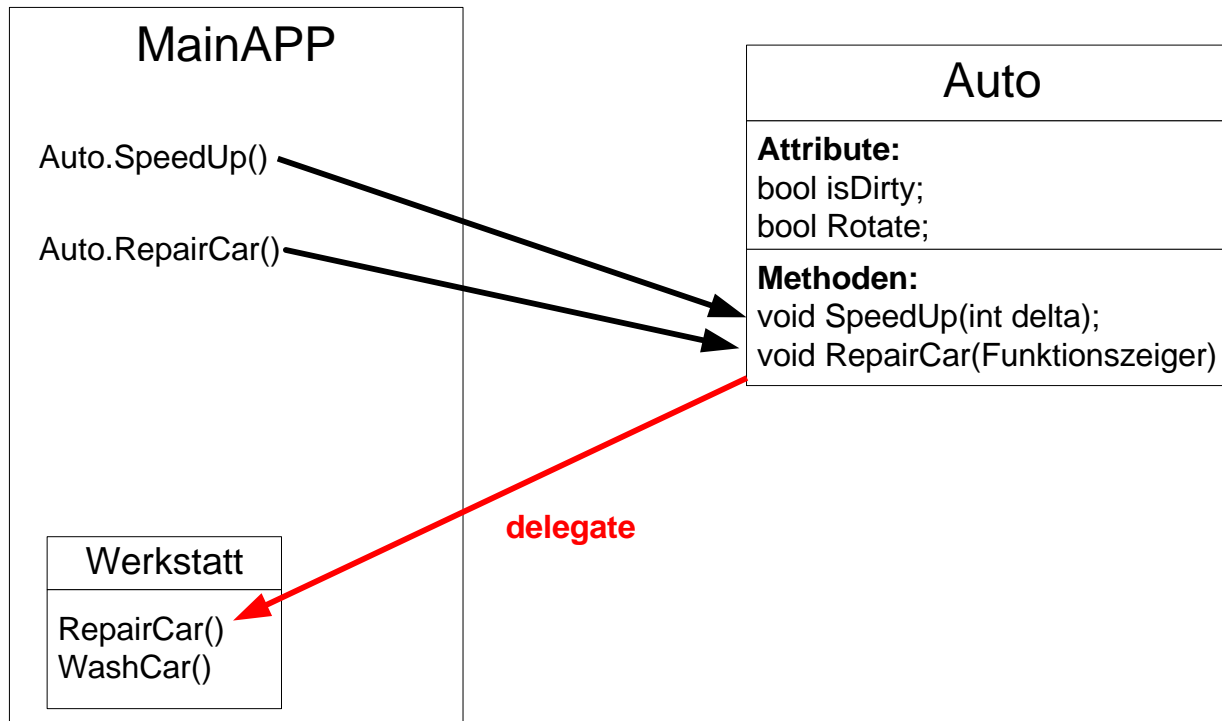
```
public class Punkt
{
    private int x,y;
    public Punkt(){}
    public Punkt(int xPos, int yPos)
    {
        this.x = xPos;
        this.y = yPos;
    }

    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

