Vorlesung
Component Ware und Web-Services

- Komponentenkonzepte -

11. Microsoft .NET

Prof. Dr. Hans-Gert Gräbe, F. Schumacher
Wintersemester 2003/2004

Inhalt

- **Einführung**
- **Das .NET Framework**
- **Common Language Runtime (CLR)**
- **Base Class Library (BCL)**
- **Common Language Specification (CLS)**
- **Microsoft Intermediate Language (MSIL)**
- **Assemblies**
- **Common Type System (CTS)**
- **Administration von .NET-Anwendungen**
- **Vergleich mit JAVA**

Was ist .NET?

"... komplette Neudefinition der Art, wie Microsoft in Zukunft Geschäfte machen will ... und wie Software entwickelt werden soll."

Westphal, 2002

- Bündel von Softwaretechnologien zum Verbinden von Informationen, Menschen, Systemen und Geräten

- Ziele
  - Sicherheit
  - Plattformunabhängigkeit
  - Interoperabilität
  - Homogenität

Standardisierung

- August 2000 – Einreichung zur Standardisierung bei ECMA
  - mit Unterstützung von HP und Intel
  - Standardisierung von C#
  - Standardisierung einer Common Language Infrastructure (CLI)
    - Untermenge der Klassenbibliotheken von .NET
    - Dateiformat
    - Typsystem
    - erweiterbares Metadatensystem
    - Intermediate Language (IL)
    - Zugriff auf die zugrunde liegende Plattform
    - skalierbare Basisklassenbibliothek
  - was fehlt:
    - Graphische Benutzerschnittstelle
    - Datenbankzugriff

Standardisierung

- Dezember 2001 – Fertigstellung des ersten Standards
  - mit Unterstützung von IBM, Fujitsu, Netscape, Sun u.a.
  - Weitergabe an die ISO
- April 2003 – Verabschiedung des ISO-Standards
  - ISO/IEC 23270 (C#)
  - ISO/IEC 23271 (CLI)

- Aktuelle Implementierungen
  - Microsoft (Windows) http://msdn.microsoft.com/net
  - MONO (Linux) http://www.go-mono.com

Die 4 Säulen von .NET

Microsoft .NET Strategie

.NET Framework

Visual Studio .NET
Codeeditor
Fenstereditor
Debugger
Server-Explorer
Entwurfshilfen
C# / VB .NET / J#
ASP.NET

.NET Enterprice Server

Application Center
Exchange Server
BizTalk Server
...

- nicht speziell an .NET angepasst
- Dienste zentral für .NET Anwen-dungen

.NET My Services

konkrete WebServices

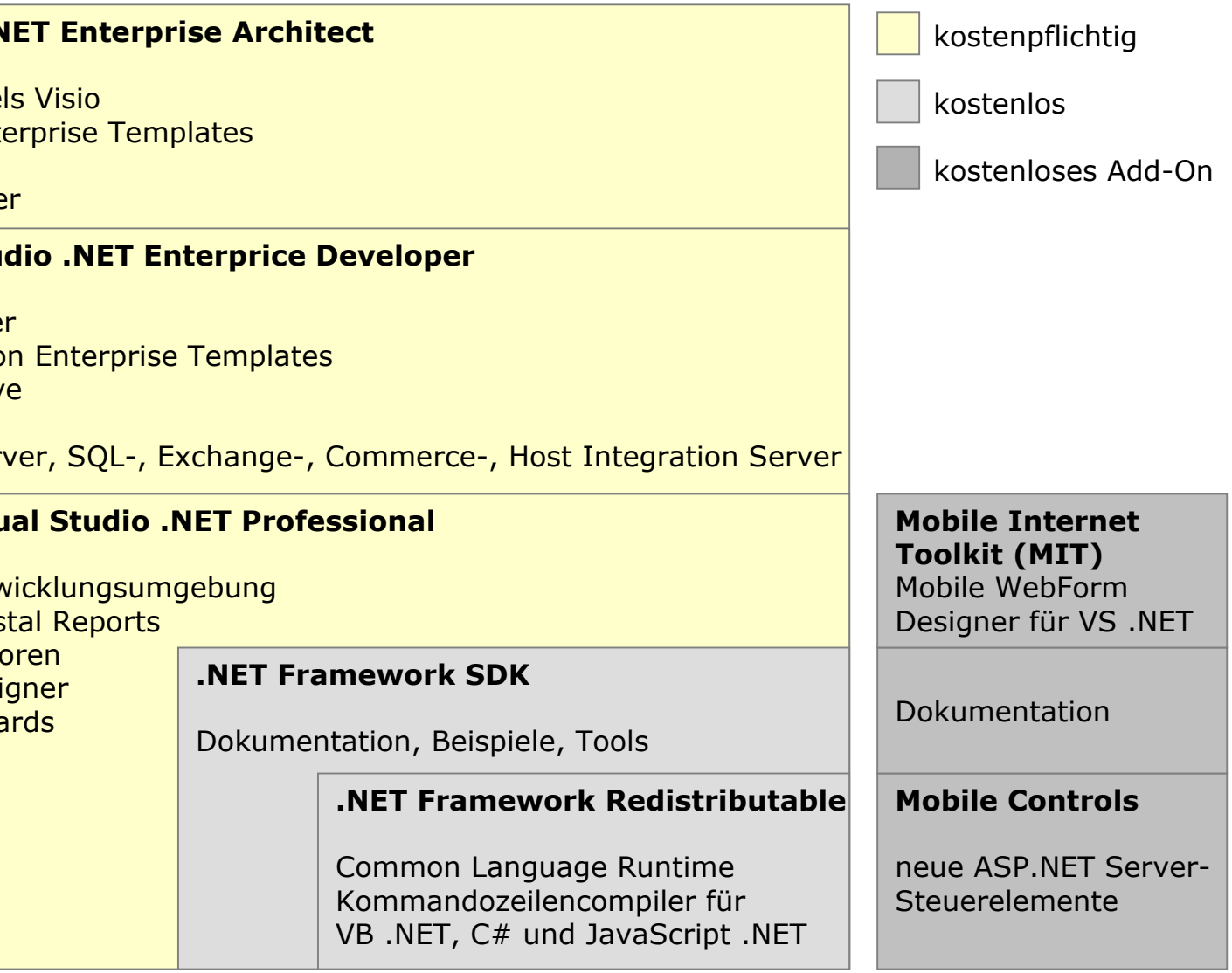
.NET Contacts
.NET Wallet
.NET Lists
Microsoft Passport
...

.NET Devices

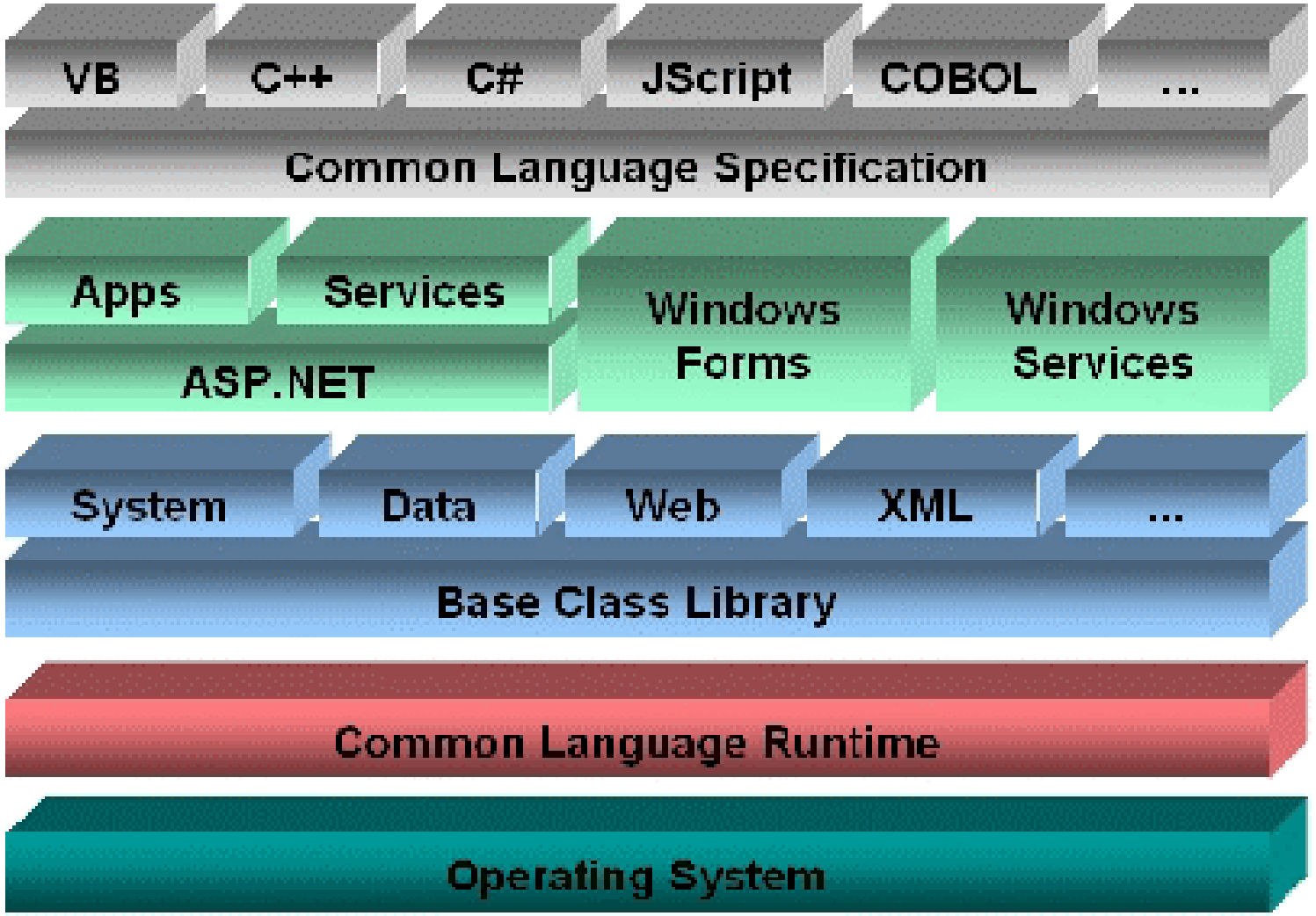
Hardware wie PDA, Handys, Tablet, PC, Auto-PC ...

- Mobile Internet Toolkit (MIT)
- .NET Compact Framework (CF)

Entwicklungsumgebung von .NET



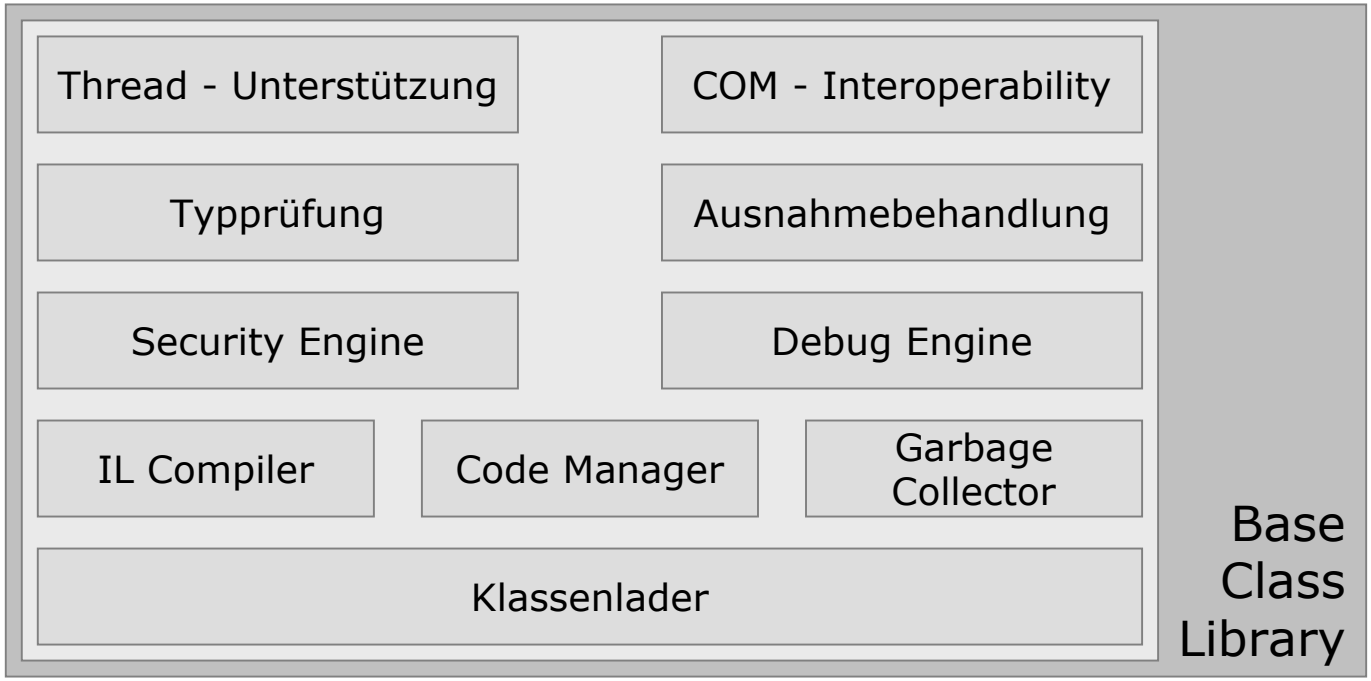
Das .NET Framework



Quelle: Microsoft

Common Language Runtime (CLR)

- Übersetzung von Zwischensprachencode (IL) in Maschinencode
- Speicherverwaltung
- Verwaltung von Prozessen und Threads
- Durchsetzung von Sicherheitsmechanismen
- Laden von Komponenten
- **Alle** .NET-Sprachen setzen auf die CLR als Runtime auf



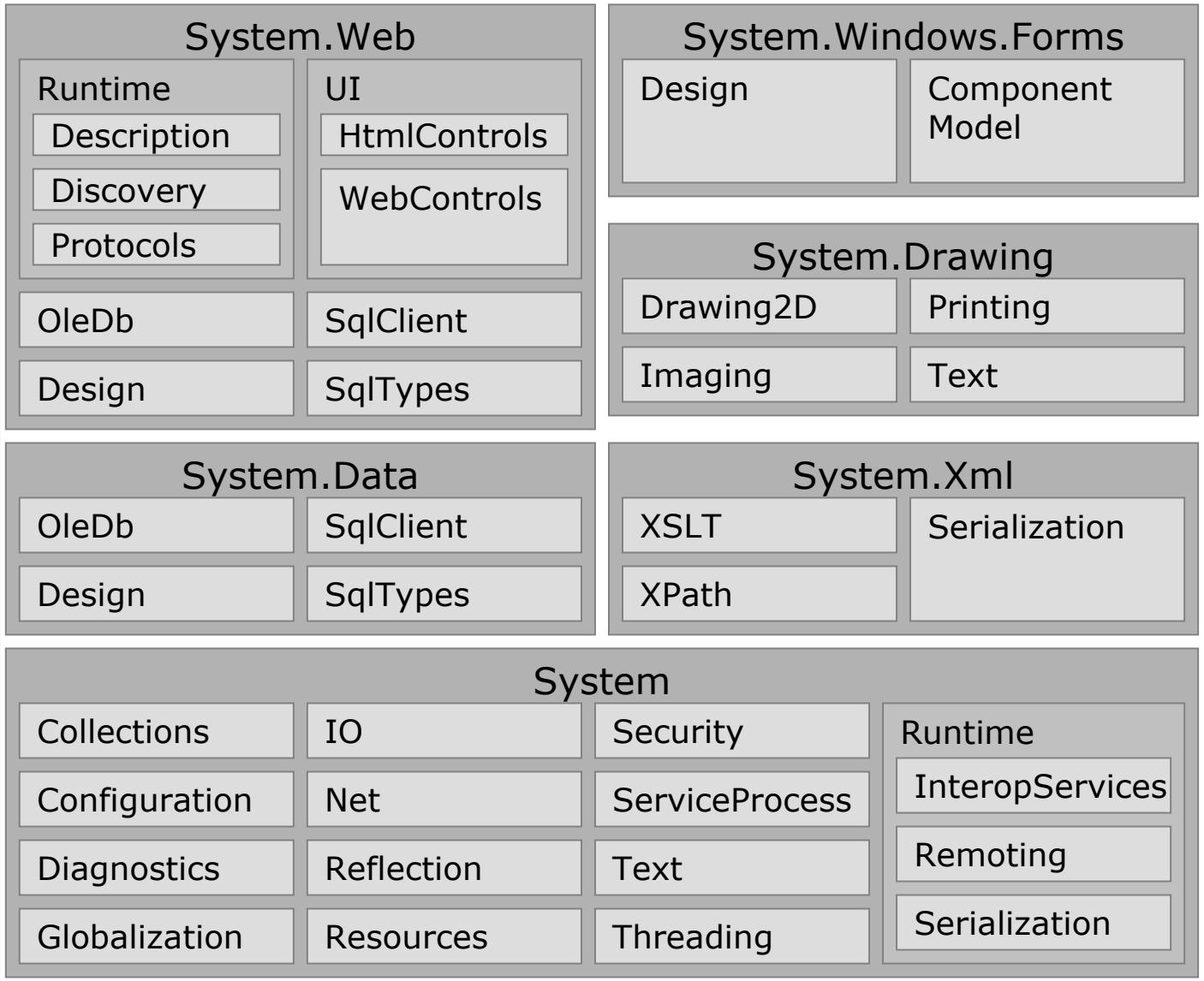
Common Language Runtime (CLR)

- Konsistentes Programmiermodell
  - alle Anwendungsdienste als objektorientiertes Programmiermodell
- Vereinfachtes Programmiermodell
  - keine Registrierung, COM-Schnittstellen, HRESULTs ...
- Stabile Installationen
  - isolierte Anwendungskomponenten
  - keine >DLL-Hölle< mehr
  - Versionierung von Komponenten
- Vereinfachte Installationen
  - Anwendungsdateien einfach in Zielverzeichnis kopieren
  - keine Registry-Einträge nötig
- Viele verfügbare Plattformen
  - Compiler generiert IL-Code
  - Ausführbar auf Maschinen, die über ECMA-kompatible Versionen der CLR verfügen
- Integration verschiedener Programmiersprachen
  - Typen, die in unterschiedlichen Sprachen geschrieben wurden
  - Common Type System

Common Language Runtime (CLR)

- Einfacheres Wiederverwenden von Code
  - durch oben beschriebene Techniken
- Automatische Speicherverwaltung
  - Garbage Collection
- Typsicherheit
  - Zugriff auf Objekte immer auf kompatible Weise
  - Code springt nur an bekannte Stellen (Eintrittspunkt von Methoden)
  - keine Pufferüberläufe
- Komfortables Debuggen
  - Debuggen von Anwendungen unterschiedlicher Sprachen
- Konsistente Fehlerverarbeitung
  - ALLE Fehler werden über Ausnahmen gemeldet
- Sicherheit
  - basierend auf Herkunft des Codes / der Daten
- Interoperabilität
  - Zugriff auf COM-Komponenten

Base Class Library (BCL)



- Schnittstelle zum Betriebssytem
- komplett Objektorientiert
- Allen .NET Sprachen stehen dieselben Dienste zur Verfügung
- Zugriff auf Dateisystem, Fensteranzeige, Druckfunktionen, Remoting, Grafik, Datenbankzugriff

Base Class Library (BCL)

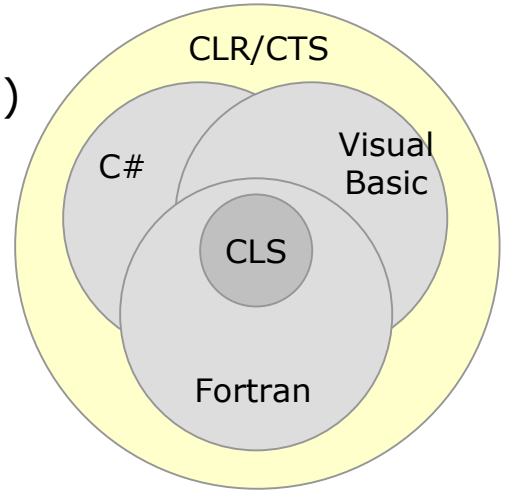
- Namespaces
  - Zusammenfassung von Typen in logische Gruppen
  - **System** ist Basisnamespace

```
class App {
   static void main() {
      System.IO.FileStream fs = new System.IO.FileStream(...);
      System.Collections.Queue q = new System.Collections.Queue();
   }
}
```

  - Unverwechselbarkeit von Namen
  - Namespace sagt nichts über physische Speicherung der Typen
  - Typen eines Namespace können über mehrere physische Dateien verteilt sein

Common Language Specification (CLS)

- kleinster gemeinsamer Nenner der .NET-Sprachen
  - standardisierte Typen
  - selbstbeschreibende Typinformationen (Metadaten)
  - gemeinsame Ausführungsumgebung




```
using System;
[assembly:CLSCompliant(true)] // Compiler soll CLS-Kompatibilität prüfen

// Fehler, weil Klasse öffentlich ist
public class App {
    // Fehler, weil UInt32 nicht CLS-Kompatibel
    public UInt32 Abc() { return 0; }
    // Fehler, weil keine Unterscheidung zwischen Groß- und Kleinschreibung in CLS
    public void abc() {}
    //Kein Fehler, da Methode privat ist
    private UInt32 ABC() { return 0;}
```

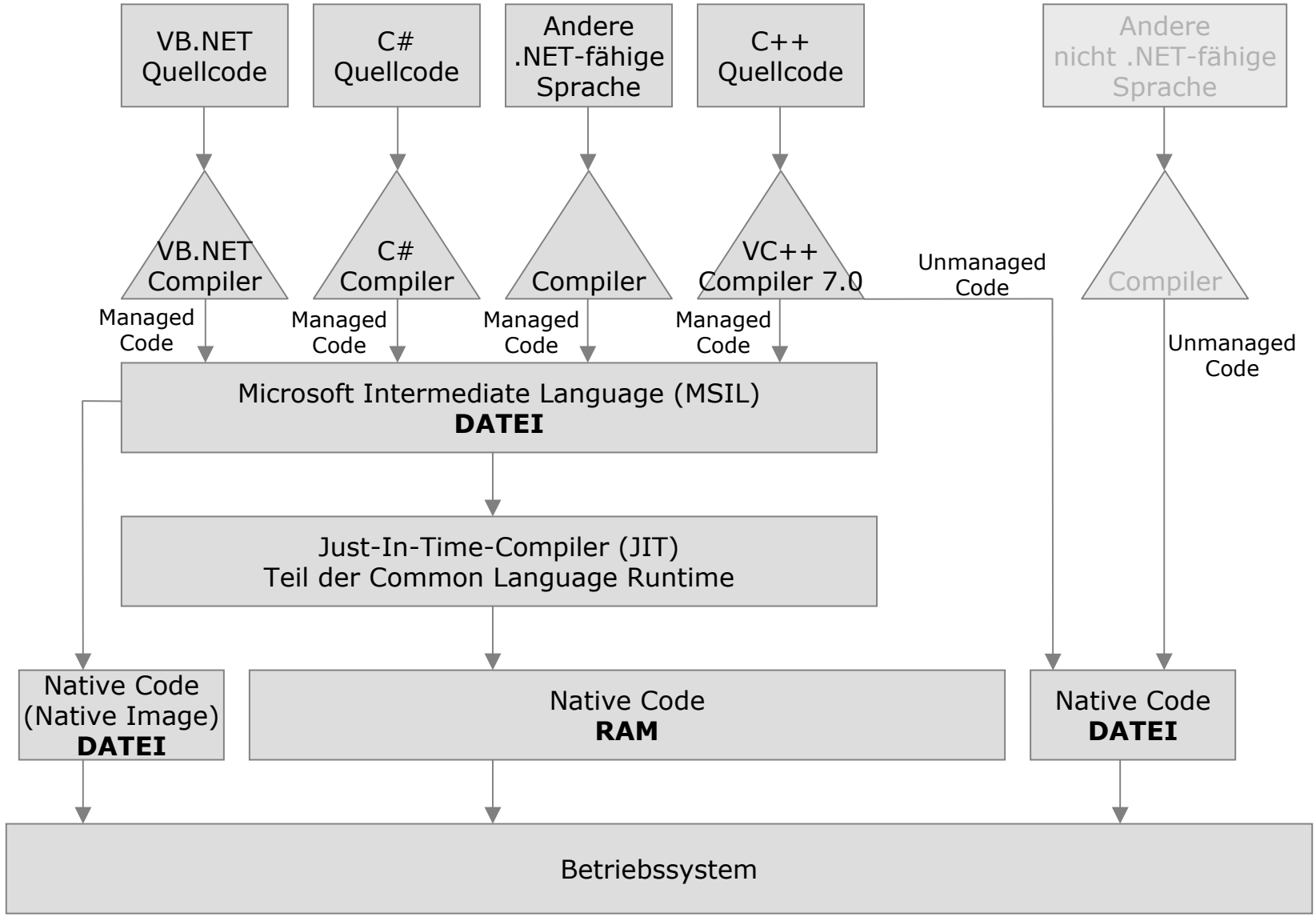
- Vollständige Liste der CLS-Regeln im Abschnitt >Cross-Language Interoperability< in der Dokumentation des .NET Framework SDK
- Vereinfacht: jeder Member eines Typs ist entweder ein Feld (Daten) oder eine Methode (Verhalten)

Common Language Specification (CLS)

Primitive Typen

C#	FCL	CLS	Beschreibung
sbyte	SByte	Nein	8-Bit-Wert mit Vorzeichen
byte	Byte	Ja	8-Bit-Wert ohne Vorzeichen
short	Int16	Ja	16-Bit-Wert mit Vorzeichen
ushort	UInt16	Nein	16-Bit-Wert ohne Vorzeichen
int	Int32	Ja	32-Bit-Wert mit Vorzeichen
uint	UInt32	Nein	32-Bit-Wert ohne Vorzeichen
long	Int64	Ja	64-Bit-Wert mit Vorzeichen
ulong	UInt64	Nein	64-Bit-Wert ohne Vorzeichen
char	Char	Ja	16-Bit-Unicodezeichen
float	Single	Ja	IEEE-32-Bit-float
double	Double	Ja	IEEE-64-Bit-float
bool	Boolean	Ja	Wahrheitswert, true oder false
decimal	Decimal	Ja	128-Bit-Gleitkommawert, hohe Genauigkeit
object	Object	Ja	Basistyp für alle Typen
string	String	Ja	Ein Array mit Zeichen

Microsoft Intermediate Language (MSIL)



Microsoft Intermediate Language (MSIL)

- IL ist eine Art "objektorientierter Maschinencode"
- arbeitet Stack-Orientiert, keine Register
- unabhängig von CPU
- Verifizierung des Codes durch die CLR bei der Übersetzung in nativen Code
  - nur Speicheradressen lesen, in die vorher Daten geschrieben wurden
  - Methoden mit der korrekten Anzahl von Argumenten aufrufen
  - jedes Argument hat richtigen Typ
  - usw.
- wird zur Laufzeit vom Just-In-Time-Compiler kompiliert
- Caching von bereits übersetzten Typen
- Optimierung anhand ausführender Architektur

```
.method private hidebysig static void Main() cil managed
{
        .entrypoint
        .maxstack 3
        .locals ([0] int32 v, [1] object o)
        IL_0000:   ldc.i4.5
        IL_0001:   stloc.0
        IL_0002:   ldloc.0
        IL_0003:   box         [mscorlib]System.Int32
```

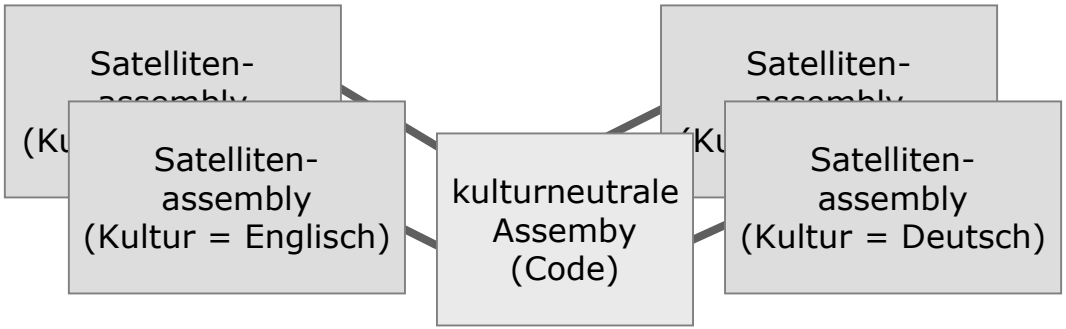
Beispiel für IL-Code

Assemblies

- atomare, selbstbeschreibende Einheit, inklusive Metadaten
- Metadatenstruktur wird **Manifest** genannt
- "single file assembly"
  - Manifest ist Teil des eigentlichen Codes
  - z.B. bei DLLs
- "multi file assembly"
  - Manifest ist eigenständige Einheit
- ein Manifest enthält ...
  - Identität der Assembly (Name, Version, ...)
  - die Namen aller Dateien im Assembly
  - kodierter Hashwert aller Dateien im Assembly
  - Details der vorhandenen Klassen, Methoden und Eigenschaften
  - Namen und Hashwerte aller referenzierten Assemblies
  - Sicherheitseinstellungen

Assemblies

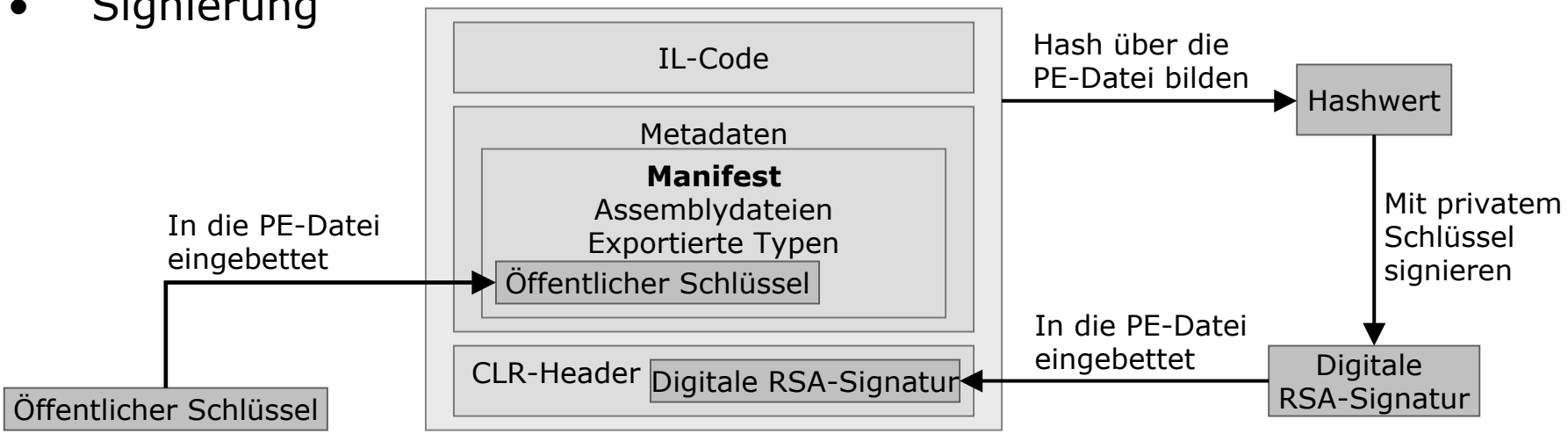
- Kultur einer Assembly
  - sprachlichen Einordnung
  - falls keine kulturspezifischen Merkmale vorhanden sind: *kulturneutral* (Code)



  - Speichern der Satellitenassembly in separaten Unterverzeichnissen (z.B. ..\en-US für US-Englisch)
  - während der Laufzeit Zugriff über System.Resources.ResourceManager

Assemblies

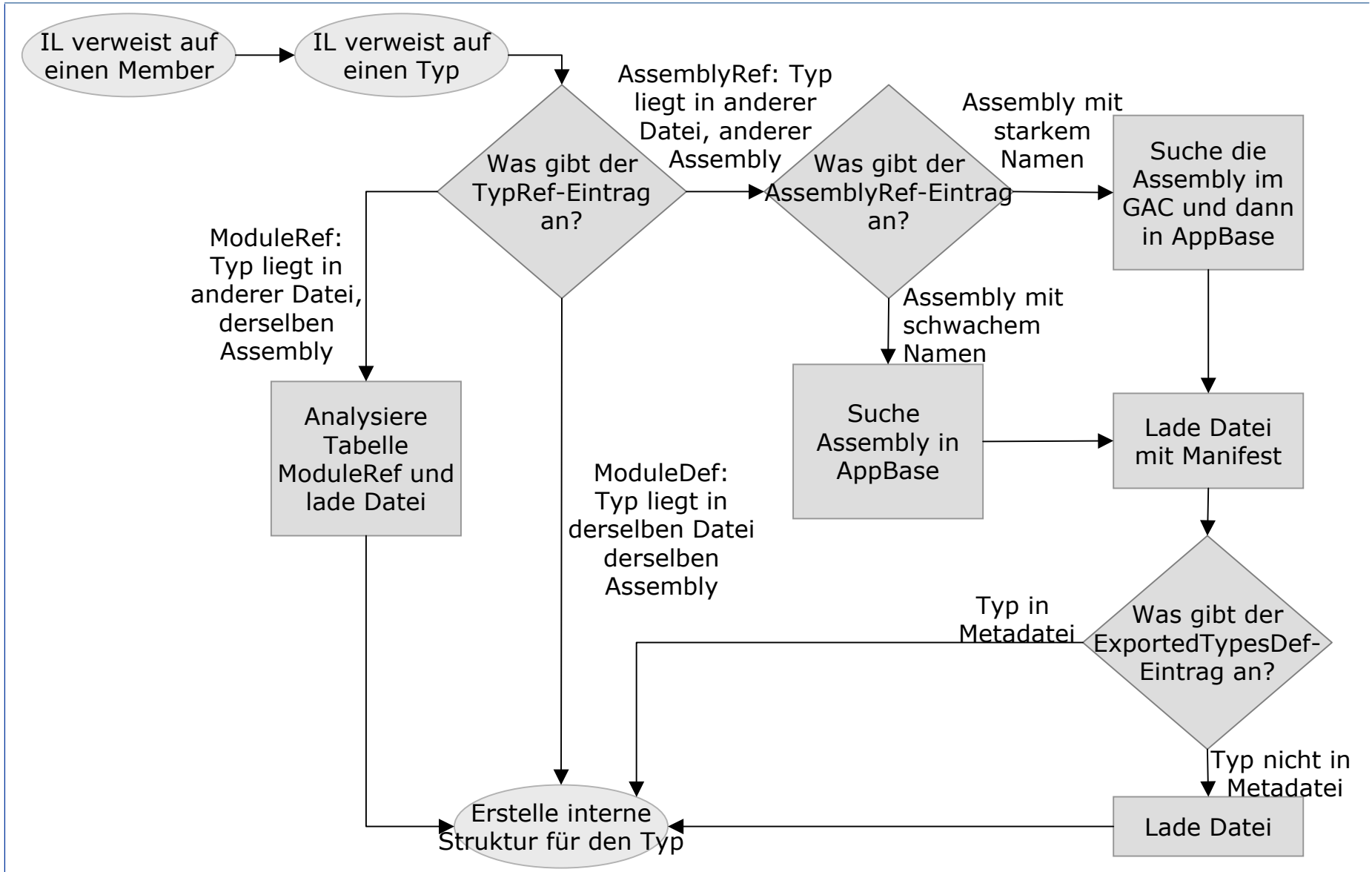
- Starke Namen (strong names)
  - zur eindeutigen Identifizierung einer Assembly
    - Dateinamen (ohne Erweiterung)
    - Versionsnummer
    - Kultur
    - Token für einen öffentlichen Schlüssel
  - kann global weitergegeben werden (Global Assembly Cache)
  - Möglichkeit zur Sicherstellung der Unversehrtheit des Codes
- Signierung



Global Assembly Cache (GAC)

- genau definiertes Verzeichnis, in der globale Assemblies abgelegt werden
  - C:\Windows\Assembly\GAC
  - niemals von Hand Dateien in GAC kopieren
- festdefinierte Struktur:
  - Name
    - Versionsnummer_Kultur_Schlüsseltoken
- Vorteil: Assembly kann global genutzt werden
- Nachteil: Keine einfache Installation mehr möglich

Auflösung von Typverweisen



Elemente eines verwalteten Moduls

- PE-Header
  - Windows-Standardheader für PE-Dateien
  - definiert Dateityp (GUI, CUI, DLL)
  - Zeitstempel
  - Informationen über speziellen CPU-Code
- CLR-Header
  - benötigte Version der CLR
  - spezielle Flags
  - Metadatentoken MethodDef (Einstiegspunkt)
  - Adresse und Größe von Metadaten des Moduls
  - Ressourcen
  - strong names

Elemente eines verwalteten Moduls

- Metadaten
  - Typen und Member, die im Quellcode definiert sind
  - Typen und Member, auf die aus dem Quellcode zugegriffen wird
- IL-Code
  - vom Compiler generierter Code
  - Programmiersprachenunabhängig
  - wird bei der Ausführung von der CLR in CPU-spezifische Befehle kompiliert

Common Type System (CTS)

Formale Spezifikation

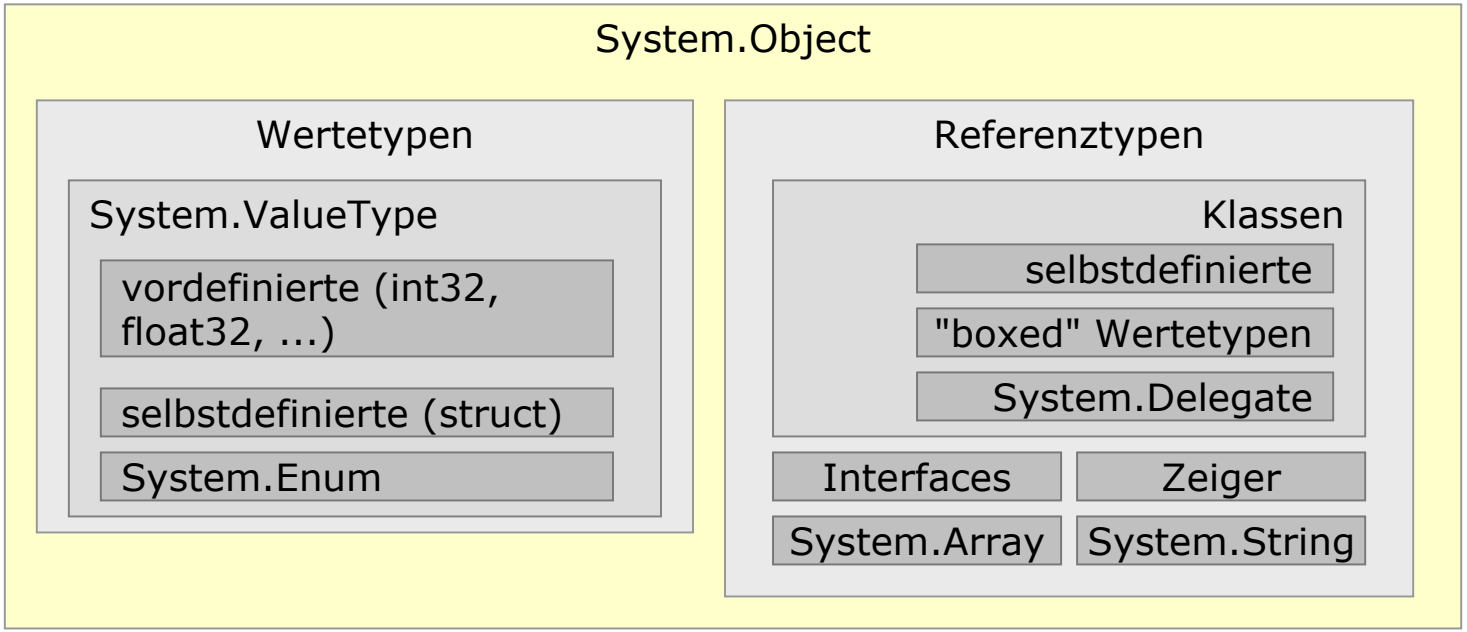
- Feld (Field)
  - Datenvariable, die Teil des Objektzustandes ist
- Methode (method)
  - Funktion, die Operationen mit dem Objekt vornimmt
  - Name, Signatur, Modifizierer (modifier)
- Eigenschaften (property)
  - sieht für Aufrufer wie ein Feld aus
  - für Entwickler: bis zu 2 Methoden (get, set)
- Ereignis (Event)
  - Mechanismus zum Benachrichtigen anderer Objekte

Common Type System (CTS)

Zugriffsmodifizierer

- Privat (Private)
  - Methode/Feld kann nur von Methoden im gleichen Klassentyp aufgerufen werden
- Familie (Family / protected)
  - kann von abgeleiteten Typen aufgerufen werden
- Assembly (internal)
  - kann von Code in derselben Assembly aufgerufen werden
- Familie **und** Assembly
  - kann von abgeleiteten Typen, die in derselben Assembly liegen, aufgerufen werden
  - steht nicht in allen Sprachen zur Verfügung (fehlt in C# und VB)
- Family **oder** Assembly (protected internal)
  - kann von abgeleiteten Typen oder Typen in derselben Assembly aufgerufen werden
- Öffentlich (public)
  - kann von jedem beliebigen Code aufgerufen werden

Common Type System (CTS)



- Wertetypen enthalten Werte (liegen auf dem Stack)
- Referenztypen zeigen auf Werte (Werte liegen auf dem Heap)
- Wertetypen können ausdrücklich als Objekte behandelt (und damit auf dem Heap abgelegt) werden: Boxing

Common Type System (CTS)

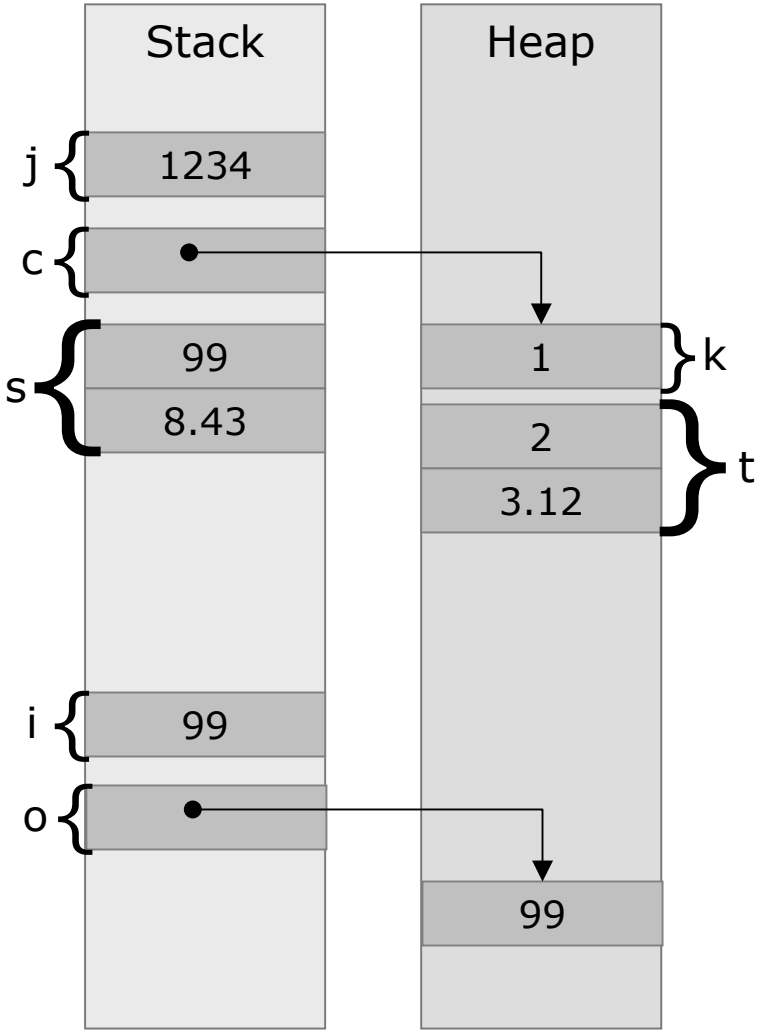
Verweis- und Referenztypen

```
struct MyStruct {
      int i;
      float f;
}
class MyClass {
      int k;
      MyStruct t
}
```

```
int j;
j = 1234;

MyClass c = new MyClass();
c.k = 1; c.t.i = 2; c.t.f = 3.12;

MyStruct s;
s.i = 99; s.f = 8.43
```



Boxing

```
int i;
i = 99;

object o;
o = i;
```

Common Type System (CTS)

Klassendefinition in C#

```
class DatenKlasse
{
      int i;
      float f;
      string s;
      int[] ai;
}
```

Klassendefinition in VB .NET

```
Private Class DatenKlasse
      Dim i As Integer
      Dim f As Single
      Dim s As String
      Dim ai() As Integer
End Class
```

Ausführbare Komponente auf IL-Ebene

```
.class private auto ansi beforefieldinit DatenKlasse
extends [mscorelib]System.Object
{
      .field private int32 i
      .field private float32 f
      .field private string s
      .field private int32[] ai

      .method public hidebysig specialname rtspecialname
      instance void .ctor() cil managed
      { ... }
}
```

Common Type System (CTS)

Alle Typen sind von System.Object abgeleitet

- öffentliche Instanzmethoden

Equals	true, falls zwei Objekte denselben Wert haben	kann überschrieben werden wenn Objektgleichheit nicht ausreicht
GetHashCode	Hashwert für den Wert des Objektes	muss überschrieben werden, falls Objekt in Hashtabelle benutzt werden soll
ToString	Stringobjekt, welches den Zustand (Standardwert) des Objektes enthält	wird üblicherweise überschrieben, sonst vollständiger Name des Typs
GetType	von Type abgeleitete Instanz, die den Typ des Objektes beschreibt	kann nicht überschrieben werden (Typsicherheit)

Common Type System (CTS)

Alle Typen sind von System.Object abgeleitet

- geschützte Methoden

MemberwiseClone	legt eine neue Instanz an und kopiert die Werte in das neue Objekt	Methode ist nicht virtuell, muss nicht überschrieben werden
Finalize	wird aufgerufen, wenn Garbage-Collector Objekt zum Löschen freigegeben hat	Typen, die etwas "aufräumen" müssen, bevor sie freigegeben werden, müssen diese Methode implementieren

Common Type System (CTS)

- Typsicherheit
  - CLR weiß zur Laufzeit **immer** um den Typen eines Objektes
  - Objekt kann seinen Typ nicht manipulieren
- Konvertierung (Type Casting)
  - Objekt kann in einen seiner Basistypen konvertiert werden (z.B. Int32 -> Object) – implizite Konvertierung
  - wenn ein Objekt in einen abgeleiteten Typen umgewandelt werden soll, muss explizit konvertiert werden

```
class SHK : Student { ... }

class App {
    public static void Main() {
        SHK x = new SHK();
        EvaluateStudent(x); // Ok, da SHK vom Typ Student abgeleitet wurde

        DateTime newYear = new DateTime(2001, 1, 1);
        EvaluateStudent(newYear); // Fehler, da DateTime nicht von Student abgeleitet wurde
    }
    public void EvaluateStudent(Object o) {
        Student s = (Student) o; // Compiler weiß nicht, ob Konvertierung erfolgen kann, erst CLR zur Laufzeit
        ...
    }
```

Common Type System (CTS)

Anweisung	OK	CTE	RTE
System.Object o1 = new System.Object();	X		
System.Object o2 = new B();	X		
System.Object o3 = new D();	X		
System.Object o4 = o3;	X		
B b1 = new B();	X		
B b2 = new D();	X		
D d1 = new D();	X		
B b3 = new System.Object();		X	
D d3 = new System.Object();		X	
B b3 = d1;	X		
D d2 = b2		X	
D d4 = (D) d1;	X		
D d5 = (D) b2;	X		
D d6 = (D) b1;			X
B b4 = (B) o1;			X
B b5 = (D) b2;	X		

```
class B
{
   Int32 x;
}

class D : B
{
   Int32 y;
}
```

Anwendungsweitergabe

- Kopieren von Dateien in Zielverzeichnis, z.B. per Batch
  - alle abhängigen Verweise und Typen sind in Assembly enthalten
  - referenzierte Assemblies im Anwendungsverzeichnis
- Konventionelle Installation möglich (cab, msi ...)
  - Verknüpfungen auf Desktop, Startleiste ...
  - Einbindung in Softwareverwaltung von Windows
  - zukünftig eventuell Automation von Verknüpfung
- Private Assemblies
  - werden in Anwendungsverzeichnis installiert
  - werden **nur** von jeweiliger Anwendung benutzt
  - es werden nur Typen gebunden, die für die Anwendung passen

Administration

- Konfigurationsdatei im Anwendungsverzeichnis zur Steuerung der Anwendung
  - Pfadeinstellung für Zusatzassemblies

```
Anwendungsverzeichnis
   App.exe
   App.exe.config
   Zusatzassembly
      zusatz.dll
      xxx.netmodule
      yyy.netmodule
```

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <runtime>
      <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
         <probing privatePath="Zusatzassembly" />
      </assemblyBinding>
   </runtime>
</configuration>
```

  - Suche nach Assemblydateien

```
AppBase\culture\AsmName.dll
AppBase\culture\AsmName\AsmName.dll
AppBase\culture\privatePath1\AsmName.dll
AppBase\culture\privatePath1\AsmName\AsmName.dll
AppBase\culture\privatePath2\AsmName.dll
AppBase\culture\privatePath2\AsmName\AsmName.dll
```

  - bei Misserfolg wird nach .exe gesucht
  - Konfigurationsdatei kann während der Laufzeit durch Klassen aus dem Namespace "System.Configuration" manipuliert werden

Administration

- Systemweite Konfiguration möglich (Machine.config)
  - *C:**Windows*\Microsoft.NET\Framework*version*\CONFIG
  - Einstellungen in Machine.config überschreiben anwendungsspezifische Einstellungen
  - Angaben zu den von der CLR verwendeten Dateien
  - Einstellungen für den Zugriff vom Internet aus
  - Einstellungen für die Kompilierung von IL-Code

Vergleich mit Java

- Sehr ähnlich zu Java
- 70% Java, 10% C++, 5% Visual Basic, 15% neu
- Ähnlichkeiten zu JAVA
  - Objektorientierung (einf. Vererbung)
  - Interfaces
  - Exceptions
  - Threads
  - Namespaces (wie Packages)
  - Strenge Typprüfung
  - Garbage Collection
  - Reflection
  - Dynamisches Laden von Code
  - Spezielle String-Unterstützung
  - ...

Vergleich mit Java

- Was ist neu?
  - Referenzparameter
  - Objekte am Stack (structs)
  - Blockmatrizen
  - Enumerationen
  - Uniformes Typsystem
  - goto
  - Systemnahes Programmieren
  - Versionierung
  - Kompatibel mit anderen .NET-Sprachen

Quellen

- ".NET kompakt" von Ralf Westphal
- "Visual Basic .NET Class Design Handbook" von Andy Olsen et al
- Microsoft .NET Framework Programmierung von Jeffrey Richter
- Softwareentwicklung mit C# von Hanspeter Mössenböck
  - http://dotnet.jku.at.
- Microsoft Website
  - http://www.microsoft.com/net
- it-Visions Website
  - http://www.it-visions.de
- .netXtreme
  - http://www.dotnetextreme.com/articles/assemblies.asp