Repetitionsfragen

Common Language Runtime

.NET Code wird direkt in Maschinencode kompiliert. **Falsch**

Sie können eine in einer anderen .NET- Sprache geschriebene Komponente in ihrem C#-Projekt einbinden. **Wahr**

Welcher Programmiersprache sieht der Microsoft Intermediate Language Code am ähnlichsten? **Assembler**

Wie ist heisst Äquivalent der Microsoft Intermediate Language in Java? **Java Bytecode**

Der Just In Time Compiler kompiliert eine Methode bei jeder Ausführung. **Falsch**

Assemblies / C# Compiler

Assemblies können weitere Assemblies beinhalten. **Falsch**

Wieso sind Metadaten im Assembly überhaupt nötig? Zur Beschreibung der Typen und zur Strukturierung des IL-Codes, welcher ohne diese lediglich eine Sammlung von Anweisungen wäre. Ohne Metadaten könnte der JIT-Compiler keinen Maschinencode erzeugen

Jedes beliebige Assembly kann in den Global Assembly Cache installiert werden. **Falsch**

Aus welchen Elementen setzt sich ein Strong Name zusammen? Bezeichnung, Versionsnummer, Kulturinformation, Public Key Token

Mit dem Tool csc.exe lassen sich auch VB.NET Dateien kompilieren. **Falsch** / CSC steht für CSharp Compiler

```
// File ClassA.cs
namespace A
{
    public class ClassA
    {
        public void DoSomething() { /* ... */ }
    }
}

// File ClassB.cs
namespace B
{
    using A;
    class ClassB
    {
        private void Work()
        {
            ClassA a = new ClassA();
            a.DoSomething();
        }
    }
}
```

Wie muss der C# Compiler angestossen werden, um beide Dateien zu kompilieren?

```
csc.exe /target:library ClassA.cs
csc.exe /target:exe
        /r:ClassA.dll
         ClassB.cs
```

Was bewirkt «/r:ClassA.dll»?

Es wird im Manifest von ClassB.dll eine Referenz auf ClassA.dll erstellt

Common Type System

Welche Komponenten stellen die Sprachneutralität des .NET Frameworks sicher?

Microsoft Intermediate Language, Common Type System, Common Language Runtime, Common Language Specification, Framework Class Library.

	Reference (Class)	Value (Struct)
Speicherort	Heap	Stack
Variable enthält	Objekt-Referenz	Wert
Nullwerte	Möglich	Nie
Default value	null	0 | false | '\0'
Zuweisung / Methodenaufruf	Kopiert Referenz	Kopiert Wert
Ableitung möglich	Ja	Nein (sealed)

Reference Types:
Auf dem Heap gespeichert
Variable enthält Referenz
Garbage Collected
Objekt muss nicht explizit aufgeräumt werden
Konstruktor erzeugt & initialisiert Objekt
Objekt hat Referenz auf seine Typenbeschreibung

Value Types:
Zur Speicherung roher Werte (auf Stack, inline in Reference- oder Value Type
Konstruktur macht nur Initialisierung
sealed
Boxing: Automatische Umwandlung in Reference Type
Unboxing: Automatische Rückumwandlung in Value Type
Vorteile: effiziente Speicherausnutzung, effizienter Zugriff, keine Garbage Collection nötig

Boxing: Kopiert Value Type in Reference Type. Implicit Conversion (Up Cast)
Unboxing: Kopiert Reference Type in Value Type. Explicit Conversion nötig / Exceptions möglich.

Structs haben einen kleineren Memory Footprint als Classes. **Wahr**

String ist ein Reference Type. **Wahr**

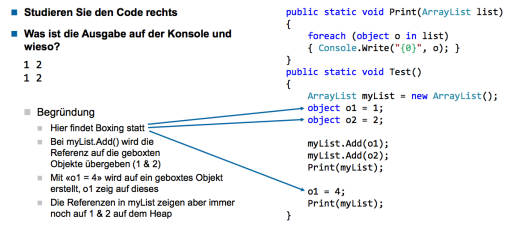
Unboxing erreicht man durch Type Casts. **Wahr**

Beim Boxing kann eine Exception auftreten. **Wahr**

Was passiert, wenn eine Value Type Variable einer anderen Value Type Variable zugewiesen wird? **Wert wird kopiert**

Was passiert, wenn eine Value Type Variable einer Methode übergeben wird? **Wert wird kopiert**

Was passiert, wenn eine Reference Type Variable einer Methode übergeben wird? **Objekt-Referenz wird kopiert**



C# Grundlagen

Kann eine int-Variable einer uint-Variable zugewiesen werden? **Ja, mit vorherigem Type Cast**

Wieso könnte das überhaupt ein Problem sein? **Vorzeichenverlust bei negativem int**

Ist die Klasse String ein Value- oder ein Reference Type? **Reference Type**

Kompiliert folgender Code?

```
internal class Base { }
public class Sub : Base { }
```

Nein, «Base» muss mindestens die gleich hohe Sichtbarkeit haben wie «Sub»

Von welcher Basisklasse leitet enum ab, sofern nichts anderes angegeben wird? **Int32**

Wieso sollte dieses Verhalten manuell übersteuert werden?
Speicherplatz-Effizienz oder Erweiterung des Nummernbereiches

Was ist die Krux dabei, wenn ein Enumerationstyp von byte, sbyte, short, ... ableitet?
Es handelt sich um Value Types welche eigentlich per Definition nicht abgeleitet werden können.
Faktisch wird dies intern auch nicht gemacht, es wird einfach syntaktisch so dargestellt.

Welche zwei Arten von multidimensionalen Arrays bietet .NET?
«Jagged» und Blockmatritzen

Was ist der Vorteil bei der Verwendung von Blockmatritzen?
Speicherplatz-Effizienz und schnellerer Zugriff

Klassen und Structs

```
class CPoint {
  public double X { get; set; }
  public double Y { get; set; }
}
class Rectangle {
  public CPoint P1 { get; set; }
  public CPoint P2 { get; set; }
}

public static void Test() {
  Rectangle r = new Rectangle();
  r.P1.X = 2.1;
  r.P1.Y = 3.5;
}
```

Funktioniert der Code oben? **Nein** / NullReferenceException. «r.P1» ist nicht initialisiert.
Lösung: Initialisierung in Konstruktor oder Properties von Rectangle

Referenzen

```
class Examples {
  static void Foo(string s)  {
    s = s + "abc";
  }
  static void Test() {
    string str = "123";
    Foo(str);
    Console.WriteLine(str);
  }
}
```

Was schreibt die Methode Test() auf die Konsole? **123**

Was muss gemacht werden, damit die Ausgabe «123abc» ist? **Pass by Reference**
Dies lässt zu, dass der Speicherbereich von «str» in Test() mit der Methode Foo(...) geshared wird. In Foo(...)arbeitet «s» also im gleichen Speicherbereich. Der Wert in diesem Speicherbereich wird mit «s = ...» verändert.

```
class Examples {
  private readonly int myProperty1;
  public int MyProperty1 {
    get { return myProperty1; }
  }
  public int MyProperty2 { get; private set; }
  public Examples(){
    myProperty1 = 1;
    MyProperty2 = 1;
  }
}
```

Wie unterscheiden sich MyProperty1 und MyProperty2 semantisch?
MyProperty1: Schränkt stärker ein. Kann nur im Konstruktor oder bei der Definition initialisiert werden
MyProperty2: Kann innerhalb der Klasse «Examples» geändert werden.

```
public class Base
{
    public Base() { Console.WriteLine(this); }
}
public class NuclearReactor : Base
{
    private bool panic;

    public NuclearReactor(int level)
    { panic = level > 10; }

    public override string ToString()
    {
        if (panic)
            return "run for your lives";
        else
            return "don't worry, be happy";
    }
}
public static void Test()
{
    var a = new NuclearReactor(12);
    Console.WriteLine(a.ToString());
}
```

In welcher Reihenfolge werden die Konstruktoren aufgerufen?
Base() -> NuclearReactor()

Was ist die Ausgabe von new NuclearReactor(20)?
don't worry, be happy

Was lernt man daraus? Achtung bei virtuellen Methoden im Konstruktor!

```
class Rectangle {
    public Point P1 { get; set; }
    public Point P2 { get; set; }
    public void Add(Point p){…}
    public static Rectangle Add(Rectangle r, Point p){…}
}

Rectangle r = new Rectangle { . . . }
r.Add(new Point(1,2)); //A
Rectangle r1=Rectangle.Add (r, new Point(1,2)); //B
```

Worin unterscheiden sich A und B?
A: ist eine Methode und verändert das Objekt r.
In B wird die statische Methode Rectangle.Add aufgerufen mit r als immutable und erstellt ein neues Rechtek.

Wie kann B eleganter umgeschrieben werden?
Mit Operator-Overloading kann r1=r+new Point(1,2) geschrieben werden.

Garbage Collection
Wie viele Generationen kennt der Garbage Collector? **Drei**

Erklären Sie, wie der Garbage Collector aufräumt
1. Alle Objekte als Garbage betrachten
2. Alle reachable Objekte markieren
3. Nicht markierte Objekte freigeben
4. Speicher kompaktieren

Wieso hat man keinen Einfluss darauf, wann der Finalizer einer Klasse aufgerufen wird?
Komplexes Blackbox System, abhängig von:
- Gerade verfügbarem Speicher
- Generation des aktuellen Objektes
- Reihenfolge in der «Finalization Queue»
- Manuell oder automatisch getriggert
- Kann auch abhängig von der .NET Runtime Version sein
- Und vieles mehr...

Was ist der Vorteil von Deterministic Finalization?
Entwickler kann gezielt Ressourcen freigeben

Sinnvolle Anwendungsfälle von Deterministic Finalization?
Grundsätzlich bei allen Klassen mit I/O:
- Dateisystem-Zugriffe
- Netzwerk-Kommunikation
- Datenbank-Anbindung
- …

Delegates und Events

Was ist ein Delegate? Typ, z.B. delegate int Calculator(int a, int b);

Was ist eine Delegate-Variable? Calculator add;

Was kann einer Delegate-Variable zugewiesen werden? **Eine Delegate Instanz**

Was ist eine Delegate-Instanz
add = delegate(int a, int b) { return a+b; }

Wie wird eine Delegate-Instanz aufgerufen? int res = add(3,4);

Was ist ein Multicast-Delegate?
Implementieren eine Liste von Delegate- Instanzen

Was ist der Unterschied zwischen normalen und Multicast-Delegates?
Alle Delegates in C# sind Multicast- Delegates

```
Calculator calc =
   delegate (int a, int b) { return a+b)
 + delegate (int a, int b) { return a-b);
int res = calc(3,2);
```

Was wird auf die Konsole ausgegeben? **1**

Wie unterscheiden sich Events und Delegates?
Delegate = Typ, z.B public void DelType(int arg);
Event = Instanz eines Delegates mit spezieller Sichtbarkeit
public event DelType MyEvent;

Welches ist die Standard-Anwendung von Events?
Observer-Pattern

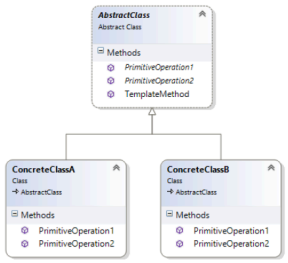
Beispiel Observable:

```
public delegate void TickHandler(Metronome m, int ticks);

public class Metronome
{
    public event TickHandler TickObservers;


    public void Start() {
        int ticks = 0;
        while (true) {
            Thread.Sleep(1000);
            ticks++;
            if (TickObservers != null) {
                TickObservers(this, ticks);
            }
        }
    }
}
```

OnTick-Methode beim Event m.TickObserves:

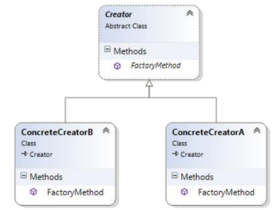
```
public class Listener
{
    public Listener(Metronome m) {
        m.TickObservers += OnTick;
    }

    private void OnTick(Metronome m, int ticks) {
        System.Console.WriteLine("Metronom ticks {0}", ticks);
    }

}
```

Wie können Sie das klassische «Template Method Pattern» mit Delegates implementieren?



TemplateMethod() oder Konstroktor von AbstractClass um 2 Delegate-Parameter erweitern.
Folgendes kann entfernt werden:
AbstractClass.PrimitiveOperation1(), AbstractClass.PrimitiveOperation2(), ConcreteClassA, ConcreteClassB

Wie können Sie das klassische «Factory Pattern» mit Delegates implementieren?



Alles nur noch über Delegates instanzieren oder FactoryMethod() um 1 Delegate-Parameter erweitern. Folgendes kann entfernt werden: Creator (je nach Implementation), ConcreteCreatorA, ConcreteCreatorB

Generics

```
public class Buffer {
  object[] items;
  public void Put(object item) { /* ... */ }
  public object Get() { /* ... */ }
}
```

Ist die Laufzeiteffizienz einer generischen Klasse «Buffer» besser als die Implementation mit «object» oben?

Bei Value Types? Ja. Für jeden verwendeten Value Type wird zur Laufzeit ein Buffer<?> erzeugt und wiederverwendet

Bei Reference Types? Nein. Es wird «nur» ein Buffer<object> erzeugt

```
class MyList { }
class MyList<T> : MyList { }
class MyDict<TKey, TValue> : MyList { }
class Examples {
  public void Test() {
    ??? l1 = new MyList<int>();
  }
}
```

Kompiliert der gegebene Code wenn «???» durch MyList<long> ersetzt wird?
Nein. Generische Typen sind nur miteinander kompatibel, wenn alle Typparameter identisch sind. (Gewisse Konstellationen liessen sich mit Ko-/Kontravarianz aber abbilden)

Was ist dürfte anstelle von «???» alles stehen?
MyList<int>, MyList, object, var, dynamic (nicht behandelt)

```
int a = 1;
int? b = 2;
int? c = null;

Console.WriteLine(a + 1); // 2 (Int32)
Console.WriteLine(a + b); // 3 (Int32)
Console.WriteLine(a + c); // (nothing) (null)
Console.WriteLine(a < b); // True
Console.WriteLine(a < c); // False
Console.WriteLine(a + null); // (nothing) (null of type int?)
Console.WriteLine(a + null < b); // False (null of type int?)
Console.WriteLine(a + null < c); // False (null of type int?)
Console.WriteLine(a + null == c); // True (null of type int?)
```

Iteratoren

Was bedeuted es, wenn eine Collection das IEnumerable<T> Interface implementiert?
Collection implementiert Iteratoren => unterstützt foreach

Welche Methoden definiert IEnumerable<T>?
IEnumerator<T> GetEnumerator<T>() liefert Iterator Objekt

Welche Members definiert IEnumerator<T> ?
bool MoveNext(), T Current, void Reset()

```
static List<T> JedesZweiteElement<T>(List<T> source)
{
    var res = new List<T>();
    int n = 0;
    foreach (T e in source)
    {
        n++;
        if (n % 2 != 0) res.Add(e);
    }
    return res;
}

static IEnumerable<T> Jedes2teElement<T>
    (IEnumerable<T> source)
{
    int n = 0;
    foreach (T e in source)
    {
        n++;
        if (n % 2 != 0) yield return e;
    }
}
```

Vorteile der Iterator-Methode:

- Nur ein Element wird zwischengespeichert
- Deferred Evalution
- Bei Abbruch muss nicht gesamtes Resultat berechnet werden

Damit man

```
foreach (int e in myList.JedesZweiteElement())
```

statt

```
List<int> myList = new List<int>() { 2, 1, ..., 4 };
foreach (int e in Program.JedesZweiteElement(myList))
```

nutzen kann, muss JedesZweiteElement als Erweiterungsmethode definiert werden:

```
static class Extensions {
static IEnumerable<T> JedesZweiteElement<T>
  (this IEnumerable<T> source) { ... }
}
```

Nicht vergessen: using Extensions;

LINQ

LINQ wurde mit CLR Features implementiert. **Falsch**, reines Compiler-Feature

Query Expressions und Lambda Expression sehen im MSIL-Code identisch aus. **Wahr**

Was ist der Unterschied zwischen Statement Lambdas und Expression Lambdas?
Statement Lambdas: Einzelner Ausdruck / Kein «return» nötig
Expression Lambdas: Anweisungsblock {} / Falls nicht «void» ist «return» zwingend nötig

LINQ könnte theoretisch verwendet werden, um Daten von einer Android-gesteuerten Raumsonde abzufragen. **Wahr**

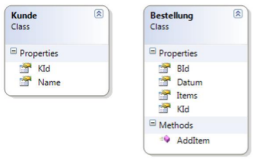
Verschiedene Varianten x1 zu initialisieren:

```
Action x1;

// Standard (Instanz-Methode) C# 1.0 / 2.0
x1 = new Action(this.Print);
x1 = this.Print;

// Standard (Statischer Meth) C# 1.0 / 2.0
x1 = new Action(Examples.PrintStatic);
x1 = Examples.PrintStatic;

// Anonymous Delegate
x1 = delegate(string sender)
        { Console.WriteLine(sender); };

// Anonymous Delegate (Kurzform)
x1 = delegate { Console.WriteLine("Hello"); };

// Lambda Expression (LINQ / später)
x1 = sender => Console.WriteLine(sender);

// Statement Lambda Expr. (LINQ / später)
x1 = sender => { Console.WriteLine(sender); };
```

```
class Examples                          «static» fehlt
{                                       «T» undefiniert
    public IEnumerable<T> HsrWhere(
        this IEnumerable<var> source,
        Predicate<T> predicate          «var» unzulässig
    )
    {
        foreach (object item in source)
        {                               Muss «T» sein
            if(predicate(item))
            {
                return item;
            }
        }
    }                                   «yield» fehlt
}
```

LINQ-Abfrage: Kunden mit mindestens einer Bestellung

```
from kunde in Kunden
join best in Bestellungen
  on kunde.KId equals best.Kid
orderby kunde.Name
select kunde.Name;
```



Oder gruppiert & sortiert nach Datum mit Bestellungen pro Datum:

```
from best in Bestellungen
group best by best.Datum
  into datumGroup
orderby datumGroup.Key
select new
{
  Datum = datumGroup.Key,
  Anzahl = datumGroup.Count()
};
```