.NET Technologien
Repetitionsfragen

Nico Fehr

28. Januar 2024

1 .NET Grundlagen

1.1 Common Language Runtime (CLR)

.NET Code wird direkt in Maschinencode kompiliert. Wahr / Falsch?
Falsch

Sie können eine in einer anderen .NET-Sprache geschriebene Komponente in ihrem C#-Projekt einbinden. Wahr / Falsch?
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 (JIT) Compiler kompiliert eine Methode bei jeder Ausführung. Wahr / Falsch?
Falsch

1.2 Assemblies

Ein Assembly enthält Metadaten und IL-Code von genau einer Klasse. Wahr / Falsch?
Falsch, Assemblies sind Komponenten und enthalten 0-n Typen (Klassen)

Assemblies können weitere Assemblies beinhalten. Wahr / Falsch?
Falsch, aber theoretisch als Resource möglich

Wieso sind Metadaten im Assembly überhaupt nötig?
Zur Beschreibung der Typen JIT-Compiler braucht dies zur Erzeugung von Maschinencode

1.3 Common Type System (CTS)

Structs haben einen kleineren Memory Footprint als Classes. Wahr / Falsch?
Wahr

String ist ein Reference Type. Wahr / Falsch?
Wahr

Unboxing erreicht man durch implizite Type Casts. Wahr / Falsch?
Falsch, muss explizit gecastet werden

Beim Boxing kann eine Exception auftreten. Wahr / Falsch?
Falsch, ausser vielleicht OutOfMemoryException

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

1.3.1 Codebeispiel

Studieren Sie den Code!

```
1  public static void Print(ArrayList list)
2  {
3      foreach (object o in list)
4      {
5          Console.Write("{0}", o);
6      }
7  }
8  public static void Test()
9  {
10     ArrayList myList = new ArrayList();
11     object o1 = 1;
12     object o2 = 2;
13
14     myList.Add(o1);
15     myList.Add(o2);
16     Print(myList);
17
18     o1 = 4;
19     Print(myList);
20 }
```

Was ist die Ausgabe auf der Konsole und wieso?
1 2
1 2

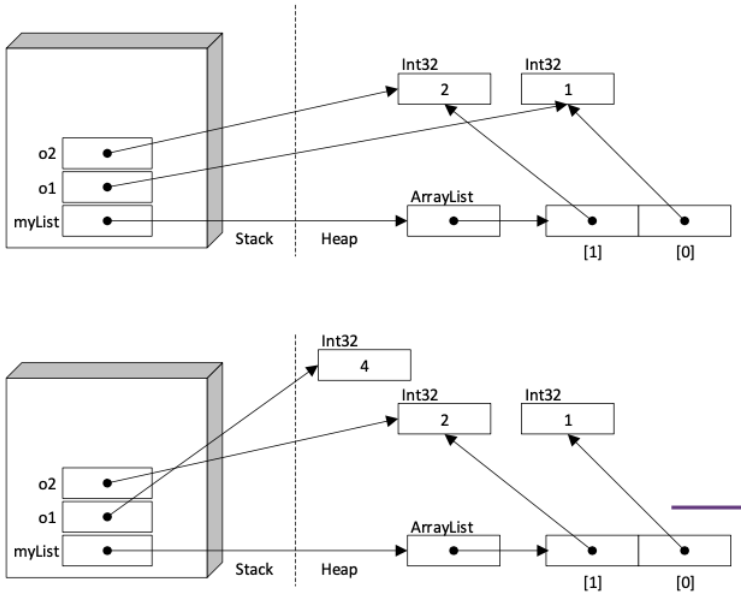

Abbildung 1: Memory Layout des Codes

Begründung:

- *Es findet Boxing statt (Zeilen 10, 11, 18)*
- *Bei `myList.Add()` wird die Referenz auf die geboxten Objekte übergeben (1 & 2)*
- *Mit `o1 = 4` wird auf ein geboxtes Objekt erstellt, `o1` zeigt auf dieses*
- *Die Referenzen in `myList` zeigen aber immer noch auf 1 & 2 auf dem Heap*

1.4 .NET Standard / NuGet

Warum wurde der .NET Standard eingeführt?
Kompatibiltät, API-Konsistenz, Cross-Platform

Was ist NuGet? Welche analogen Services gibt es für andere Sprachen?
Ein Package Manager (haupts. .NET). Analog zu NPM, yearn, etc.

Wie ist ein NuGet Package aufgebaut?
Zip-Datei mit Libraries, Manifest, etc.

2 C# Grundlagen

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

Wieso könnte das überhaupt ein Problem sein?
Wenn der int negativ wäre und dann dem vorzeichenlosen uint zugewiesen wird.

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 gleichhohe 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“ (ausgefranst) und Blockmatrizen

Was ist der Vorteil (+ Nachteil) bei der Verwendung von Blockmatritzen?

- *Speicherplatz-Effizienz*
- *Schnelleres Allozieren*
- *Schnellere Garbage Collection*
- *Kein Schnellerer Zugriff, weil Boundary-Check bei eindimensionalen Arrays viel schneller ist*

3 Klassen und Structs

3.1 Allgemein

Funktioniert folgender Code?

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

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

Nein, `NullReferenceException` „`r.P1`“ ist nicht initialisiert

Wie könnte `Point` und/oder `Rectangle` angepasst werden um dies zu verhindern?

- *Initialisierung in Konstruktor oder Property von Rectangle*
- *Point als `struct` definieren*

3.2 Memory Modell

Skizzieren Sie gemäss folgendem Code das Memory-Layout der Variable `r` unter der Annahme, dass `P1` / `P2` im Konstruktor von `Rectangle` initialisiert werden!

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

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

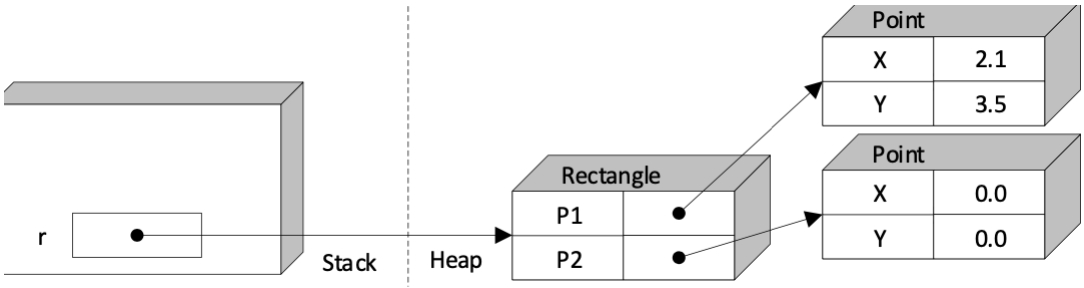

Abbildung 2: Memory Layout

Wie verändert sich das Memory-Layout wenn `Point` von `class` nach `struct` wie folgt ändert?

```
...
struct Point
{
    public double X { get; set; }
    public double Y { get; set; }
}
...
```

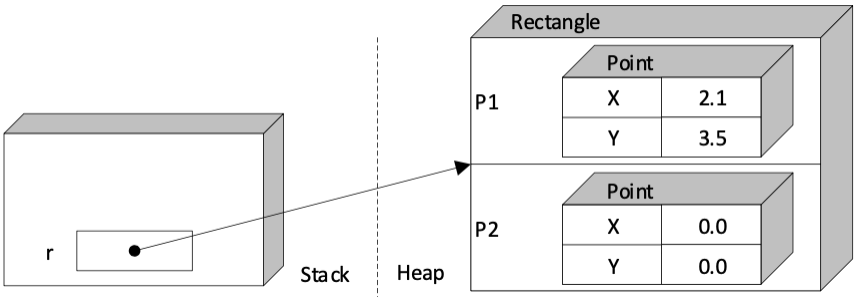

Abbildung 3: Verändertes Memory Layout

3.3 Indexer

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

Was muss im obigen Code verändert werden, dass folgendes Statement funktioniert?

```
int val = r[1].X;
```

```
class Point
{
    public double X { get; set; }
    public double Y { get; set; }
}
class Rectangle
{
    public Point P1 { get; set; }
    public Point P2 { get; set; }
    public Point this[int index]
    {
        get
        {
            if (index == 1) return P1;
            if (index == 2) return P2;
            return default(Point);
        }
    }
}
```

3.4 Referenzen

Was schreibt die Methode `Test()` auf die Konsole (`Person` ist eine Klasse)?

```
static void Foo(Person p)
{
    p.Age += 10;
}
static void Test()
{
    Person pers = new(age: 24);
    Foo(pers);
    Console.WriteLine(pers.Age);
}
```

34

Was schreibt die Methode `Test()` auf die Konsole?

```
static void Foo(string s)
{
    s = s + "abc";
}
```

```
static void Test()
{
    string str = "123";
    Foo(str);
    Console.WriteLine(str);
}
```

123

Was mus geändert werden, damit die Ausgabe „123abc“ ist?

- *Übergabe „by ref“*
- *Dies lässt zu, dass der Speicherbereich von `str` in `Test()` mit der Methode `Foo(...)` geteilt wird.*
- *In `Foo(...)` arbeitet `s` also im gleichen Speicherbereich.*
- *Der Wert in diesem Speicherberich wird mit `s = ...` verändert*

3.5 Read-Only

Wie unterscheiden sich `MyProperty1` und `MyProperty2` semantisch?

```
class Examples
{
    private readonly int _myProperty1;
    public int MyProperty1
    {
        get { return _myProperty1; }
    }

    public int MyProperty2
    { get; private set; }
    public int MyProperty3
    { get; }

    public Examples()
    {
        myProperty1 = 1;
        MyProperty2 = 1;
        MyProperty3 = 1;
    }
}
```

- *`MyProperty1`*

  Schränkt stärker ein

  Kann nur im Konstruktor oder bei der Definition initialisiert werden

- *`MyProperty2`*

  Kann innerhalb der Klasse `Examples` geändert werden

Wie unterscheiden sich `MyProperty1` und `MyProperty3` semantisch?
Identisch

3.6 Konstruktoren / Reihenfolge

In welcher Reihenfolge werden die Konstruktoren aufgerufen?

```
public static void Test()
{
    NuclearReactor a = new(12);
    Console.WriteLine(a.ToString());
}

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";
    }
}
```

1. Base()

2. NuclearReactor()

Was ist die Ausgabe der Methode `Test`?
don't worry, be happy
run for your lives

Was lernen Sie daraus?
Achtung bei virtuellen Method im Konstruktor!

4 Vererbung

4.1 Überschreiben

Funktioniert folgendes Szenario? `Car.Drive()` gibt `void` zurück `ElectroCar.Drive()` gibt `int` zurück.

```
class Vehicle
{
public virtual void Drive() { }
}

class Car : Vehicle
{
public override void Drive() { }
}

class ElectroCar : Car
{
public new virtual int Drive() { return 0; }
}

class SolarElectroCar : ElectroCar
{
public override int Drive() { return 0; }
}
```

Ja, da die Methode überdeckt wird und kein Dynamic Binding angewandt wird.

Würde der Code auch funktionieren, wenn `ElectroCar.Drive()` nicht `new virtual` sondern `override` wäre?
Nein, beim Überschreiben einer Methode (mit Dynamic Binding) muss der Rückgabewert übereinstimmen resp. konkreter als jener von der überschriebenen Methode sein

4.2 Versiegeln

Funktioniert folgendes Szenario? `Car.Drive()` ist versiegelt `ElectroCar.Drive()` ist „`new virtual`“.

```
class Vehicle
{
public virtual void Drive() { }
}

class Car : Vehicle
{
public override sealed void Drive() { }
}

class ElectroCar : Car
{
public new virtual int Drive() { return 0; }
}

class SolarElectroCar : ElectroCar
{
public override int Drive() { return 0; }
}
```

Ja, sealed auf einer Methode unterbricht nur das Dynamic Binding.

Dürfte `ElectroCar.Drive()` auch mit `override` markiert sein?
Nein, genau das soll `sealed` verhindern.

4.3 Interfaces / Abstrakte Klassen

Funktioniert folgender Code obwohl die Klasse `List` keine `Sort()`-Methode definiert?

```
abstract class Sequence
{
    public abstract void Add(object x);
    public virtual void Sort()
    { /* Sort Logic */ }
}
interface ISortableCollection
{
    void Sort();
}
class List : Sequence, ISortableCollection
{
    public override void Add(object x) { }
}
```

Ja, die `Sort()`-Methode wird von der Basisklasse vererbt.

Würde der Code funktionieren, wenn `Sequence.Sort()` ebenfalls `abstract` wäre?
Nein

Wie müsste der Code dann verändert werden? Nennen Sie zwei Möglichkeiten!

- *`List.Sort()` implementieren*
- *`List` ebenfalls als `abstract` markieren*

5 Delegates und Events

5.1 Delegates

Was ist ein Delegate?
Ein Typ

Auf welcher Ebene / wo im Code wird ein Delegate-Typ deklariert?
Auf Namespace-Ebene wie Klassen, Interfaces, etc.

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

Deklarieren Sie ein `Calculator` Delegate mit zwei `int`-Parametern und `int` als Rückgabewert!

```
delegate int Calculator(int a, int b);
```

Deklarieren Sie eine Variable `add` vom Typ `Calculator`!

```
Calculator add;
```

Weisen Sie dem Delegate eine Lambda Expression zu!

```
add = (a, b) => a + b;
```

Führen Sie das Delegate aus!

```
int result = add(3, 4);
```

Was ist ein Multicast-Delegate?
Ein Delegate mit mehereren Callback-Methoden

In welcher Art Liste ist das Multicast Delegate implementiert?
Als Linked List

Was ist der Unterschied zwischen normalen und Multicast-Delegates?
Keiner, Trennung ist nur logischer Natur, technisch sind alles Multicast-Delegates

Was wird auf die Konsole ausgegeben?

```
Calculator calc1 = (a, b) => a + b;
Calculator calc2 = (a, b) => a - b;

Calculator final = calc1 + calc2;

int res = final(3, 2);
Console.WriteLine(res);
```

Ausgabe: 1

5.2 Events

Wie unterscheiden sich Events und Delegates?

- *Delegate ist ein Typ*

  ```
  public delegate void DelType(int arg);
  ```

- *Event ist ein Compiler-Konstrukt für Klassenmember*

  ```
  public event DelType MyEvent;
  ```

- *Member wird `private`*
- *Add / Remove Methode wird generiert*

Welches Entwurfsmuster / Pattern ersetzen Events?
Observer-Pattern

5.2.1 Observable

Erweitern Sie den Code um einen Event `TickObservers`!
Feuern Sie den Event nach jedem Tick!

```
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);
            }
        }
    }
}
```

5.2.2 Observer

Registrieren Sie die `OnTick`-Methode am Event `m.TickObservers`!

```
public class Listener
{
    public Listener(Metronome m) {
        m.TickObservers += OnTick;
    }
    private void OnTick(Metronome m, int ticks) {
        System.Console.WriteLine("Metronom␣ticks␣{0}", ticks);
    }
}
```

6 Generics

Ist die Laufzeiteffizienz einer generischen Klasse ≪Buffer≫ besser als die Implementation mit `object`?

```
public class Buffer
{
    object[] items;

    public void Put(object item)
    { /* ... */ }

    public object Get()
    { /* ... */ }
}
```

Bei Value Types?
Ja

Bei Reference Types?
Nein

Begründen Sie die obigen Antworten!

- *Value Types: Für jeden verwendeten Value Type wird zur Laufzeit ein `Buffer<?>` erzeugt und wiederverwendet*
- *Reference Types: Es wird „nur“ ein `Buffer<object>` erzeugt*

6.1 Generic Type Constraints

Korrigieren Sie folgende Methode

```
public T GetInstance<T>() where T : new()
{
    return new T();
}
```

Die Methode ist so korrekt und funktioniert.

Korrigieren Sie folgende Methode

```
public void Work<T>()
    where T : List<T>, Dictionary<T>
{
    /* ... */
}
```

Die Methode kann wie folgt angepasst werden, `Dictionary<T>` ist nicht möglich

```
public void Work<T>()
    where T : List<T>
{
    /* ... */
}
```

Korrigieren Sie folgende Methode

```
void ForAll<T>(List<T> l, Action<T> a)
    where T : struct
{
    if (a == null) { return; }

    foreach (T e in l)
    {
        if(a != null && e != null)
            a(e);
    }
}
```

Die Methode ist nicht ganz korrekt, das `if`-Statement kann weggelassen werden, da bereits auf `null` geprüft wurde.

```
void ForAll<T>(List<T> l, Action<T> a)
    where T : struct
{
    if (a == null) { return; }

    foreach (T e in l)
    {
        a(e);
    }
}
```

6.2 Zuweisungen

Kompiliert der gegebene Code wenn „???“ durch `MyList<long>` ersetzt wird?

```
class MyList { }
class MyList<T> : MyList { }
class MyDict<TKey, TValue> : MyList { }

class Examples
{
    public void Test()
    {
        ??? l1 = new MyList<int>();
    }
}
```

Nein

Begründen Sie weshalb!
Generische Typen sind nur miteinander kompatibel, wenn alle Typparameter identisch sind. (Gewisse Konstellationen liessen sich mit Ko-/Kontravarianz aber abbilden)

Was wäre anstelle von „???“ erlaubt?

```
MyList<int>
```

```
MyList
```

```
object
```

```
var
```

```
dynamic
```

7 Nullability & Record Types

Was wird beim Einsatz von Records / Record Types automatisch generiert?

- Konstruktor
- Properties (immutable)
- Value equality
- Darstellung (ToString-Methode, etc.)
- Vererbung wird berücksichtigt (z.B. Equality)

Record Types können nicht um „Custom Code“ ergänzt werden. Wahr / Falsch?
Falsch, eine Kombination aus generiertem und Custom Code ist möglich.

Was ist der einfachste Weg, den Wert eines Properties bei einem Record zu verändern?
Nondestructive Mutation mit with-Clause

Vervollständigen Sie die Tabelle unten!

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

Expression	Typ	Resultat
`a + 1`	`int`	`2`
`a + b`	`int?`	`3`
`a + c`	`int?`	`null`
`a < b`	`bool`	`true`
`a < c`	`bool`	`false`
`a + null`	`int?`	`null`
`a + null < b`	`bool`	`false`
`a + null < c`	`bool`	`false`
`a + null == c`	`bool`	`true`

8 Iteratoren

Was bedeutet es, wenn eine Collection das `IEnumerable<T>` Interface implementiert?
Collection implementiert Iterator(en) resp. unterstützt `foreach`

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

Welche Members definiert `IEnumerator<T>`?

- *`bool MoveNext()`*
- *`T Current`*
- *`void Reset()`*

Schreiben Sie die untenstehende Methode als Iterator-Methode um!

```
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;
}
```

Neue Methode:

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

Was sind die Vorteile der Iterator-Methode?

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

8.1 Erweiterungsmethode

Wie müsssen Sie den Code erweitern, so dass anstelle

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

diese Schreibweise möglich ist?

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

`JedesZweiteElement` als Erweiterungsmethode definieren

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

Was müssen Sie für die Verwendung der Erweiterungsmethode beachten?

```
using Extensions;
```

9 LINQ (Language Integrated Query)

LINQ wurde mit CLR Features implementiert. Wahr / Falsch?
Falsch, Reines Compiler-Feature

Query Expressions und Lamda Expression sehen im MSIL-Code identisch aus. Wahr / Falsch?
Wahr

Was ist der Unterschied zwischen Statement Lambdas und Expression Lambdas?

- *Expression Lambdas: Einzelner Ausdruck / Kein* `return` *nötig*
- *Statement Lambdas: Anweisungsblock {} / Falls nicht* `void` *ist* `return` *zwingend nötig*

LINQ könnte theoretisch verwendet werden, um Daten von einer android-gesteuerten Raumsonde abzufragen. Wahr / Falsch?
Wahr

Finden Sie möglichst viele Alternativen, die Variable `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 / spaeter)
x1 = sender => Console.WriteLine(sender);

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

Die Methode «HsrWhere» definiert sich wie folgt:

- Filtert ein `IEnumerable<T>` anhand eines Prädikats
- Extension Methode
- Unterstützt Deferred Evaluation / Iterator-Methode

Wieso kompiliert / funktioniert die Methode `HsrWhere` nicht?

```
1  class Examples
2  {
3      public IEnumerable<T> HsrWhere(
4          this IEnumerable<var> source,
5          Predicate<T> predicate)
6      {
7          foreach (object item in source)
8          {
9              if(predicate(item))
10             {
11                 return item;
12             }
13         }
14     }
15 }
```

- *Line 1: `static` fehlt*
- *Line 3: `static` fehlt, `T` undefiniert*
- *Line 4: `var` unzulässig*
- *Line 7: `object` muss `T` sein*
- *Line 11: `yield` fehlt for `return`*

9.1 Beispiel LINQ-Abfrage 1

Erstellen Sie eine LINQ-Abfrage, die alle Bestellungen wie folgt ausgibt:

- Name des Kunden
- Datum der Bestellung
- Sortiert nach Name

Datenbasis

- Property „Kunden“: Liste von Kunden
- Property „Bestellungen“: Liste von Bestellungen

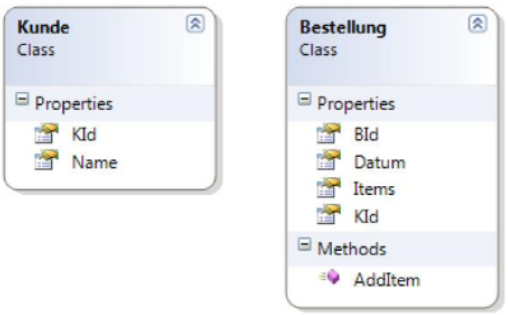

Abbildung 4: Klassen Kunde und Bestellung

Lösung:

```
1 from best in Bestellungen
2 join kunde in Kunden
3     on best.KId equals kunde.KId
4 orderby kunde.Name
5 select new
6 {
7     Name = kunde.Name,
8     Datum = best.Datum
9 };
```

9.2 Beispiel LINQ-Abfrage 2

Wir benutzen die gleiche Datenbasis wie im 1. Beispiel. Erstellen Sie eine LINQ-Abfrage, die Bestellungen wie folgt ausgibt:

- Gruppiert nach Datum
- Anzahl Bestellungen an diesem Datum
- Sortiert nach Datum

Lösung:

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

10 Entity Framework Core

Welche drei konzeptionellen Elemente kennt ein OR-Mapper?

- *Storage Entity*
- *Entity Type*
- *Mapping*

Welche drei Möglichkeiten hat ein Entwickler, um EF Core den Namen einer Tabelle anzugeben?

- *Name der Klasse (by Convention)*
- *Table-Attribut (Data Annotations)*
- *Fluent API*

Wo passiert ein Datenbankzugriff...

- Ohne Lazy Loading?
- Mit Lazy Loading?

```
using (ShopContext context = new ShopContext())
{
    Category category = context
        .Categories
        .Single(c => c.Id == 1);

    category.Name = $"{category.Name} / Changed";
    context.SaveChanges();

    var categories = context.Categories;
    foreach (Category c in categories)
    {
        Console.WriteLine(c.Name);
        Console.WriteLine(c.Products.Count());
    }
}
```

Erklären Sie die den Begriff „Optimistic Concurrency“ in wenigen Worten!

- Annhame: Zwischen Lesen und Speichern wird der Datensatz nicht von jemand drittem verändert
- Falls doch, Erkennung ob dies passiert ist oder nicht

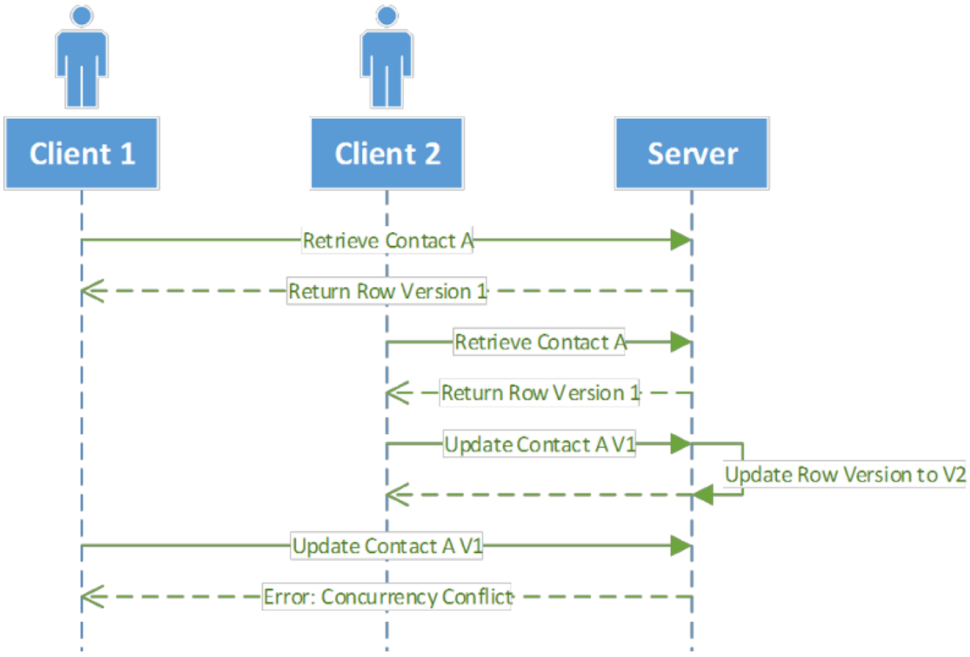

Abbildung 5: Erklärung Optimistic Concurrency