KT 1

C# Dotnet Basics

global.json - used to fix dependencies, fe. dotnet sdk version

LanguageMix is possible, fe.

```
dotnet new classlib --language=VB
```

to create classlib in Visual Basic

Terminal:

```
# New Console Application named PrimeCalc
dotnet new console --output PrimeCalc
# New Solution
dotnet new sln --output PrimeCalc
# New classlib
dotnet new classlib --output PrimeCalc.Math
# Add reference
dotnet add .\PrimeCalc.Client\ reference .\PrimeCalc.Math\PrimeCalc.Math.csproj
# Add package
dotnet add PrimeCalc.Client package Newtonsoft.Json
```

Different kinds of deployments

- SFE Single File Executable
- SCD Self Contained Deployment
    - Includes Runtime
- FDD Framework-dependent Deployment:
    - Default
    - App is published as dll.

Yield Return

...

```
public static IEnumerable<T> Filter<T>(this IEnumerable<T> items, Func<T, bool>
predicate)
{
    foreach (var item in items)
```

```
    {
        if (predicate(item))
        {
            yield return item;
        }
    }
}
```

Delegates

Deklariert ein Funktionsinterface. Dieses kann dann, mit einer Function, die dieses INterface erfüllt, instanziert werden.

```
delegate void GenericProcedure<T>(T value);
static void PrintInteger(int value) => Console.WriteLine(value);
GenericProcedure<int> p7 = PrintInteger;
p7(8);

delegate int StringToIntFunction(string str);
StringToIntFunction f2 = int.Parse;
Console.WriteLine(f2("4711"));
```

Events Delegate registrierung

Events sind quasi eine Sammlung, denen beliebig viele Funktionen zugewiesen werden dürfen, welche das delegate erfüllen, mit welchem das event deklariert wurde. Delegate ist eigentlich das gleiche nur Events können nur mit += und -= Funktionen zuegwiesen werden.

```
//Definiere delegate
public delegate void ExpiredEventHandler(DateTime signaledTime);
//Deklariere ein event, welches Funktionen nach dem delegate registrieren kann
public event ExpiredEventHandler Expired;
//Registriere eine Funktion auf dem event
//time => Console.WriteLine($"Expired at {time:HH:mm:ss}") =
//void ExpiredEventHandler(time)
timer.Expired += time => Console.WriteLine($"Expired at {time:HH:mm:ss}");
// Ruft alle registrierten Funktionen
Expired?.Invoke(DateTime.Now);
```

Function = Funktions delegate mit returnwert
Action = Function ohne return, sprich Methode

```
int anyFunc(Func<T,int> function) {
    int variable = function(new T);
    return variable;
}

void smth(Action<T> action) {
    action(new T);
    //something happened
}
```

Predicate = Function delegate welches einen Wert als parameter nimmt und ein Boolean returned.

```
void anyFunc(Predicate<T> predicate){
    bool boolean = predicate.Invoke(new T());
}
```

Comparison = Nimmt ein comparator delegate.

```
//Comparison: delegate int comparer<T>(T x, T y);
public static T MaxBy<T>(this IEnumerable<T> items, Comparison<T> comparer)
{
    using IEnumerator<T> e = items.GetEnumerator();
    if (!e.MoveNext())
    {
        throw new ArgumentException("Max for empty collection not defined");
    }
    T maxItem = e.Current;
    while (e.MoveNext())
    {
        if(comparer(e.Current,maxItem) > 0)
        {
            maxItem = e.Current;
        }
    }
    return maxItem;
}

persons.MaxBy((person1,person2) =>
person1.DateOfBirth.CompareTo(person2.DateOfBirth))
```

Erweiterungsmethoden

Eine Erweiterungsmethode stellt eine Methode für ein bestehendes Interface. bzw eine Klasse dar.
Zu beachten sind, dass die Methode static sein muss und als ersten Parameter
(this <zu erweiterndes Interface> name) aufführen muss.
Diese Methode muss in einer static Class enthalten sein.

```
public static class StaticExtensionClassForICollection
{
    public static int EMethod<T>(this ICollection<T> target, T param)
    {
        target.Add(param);
        return param;
    }
}
```

Asynchronous Programming

Asynchronous Programming versucht Threads möglichst nützlich auszulasten.
Besipiel: zwei Downloads gleichzeitg laufen lassen.
Aufpassen: alle Methoden/Funktionen async definieren falls await darin verwendet wird.
async ermöglicht verzahnte und paralelle Ausführung.
async Funktionen returnen einen Task oder void (Nur Eventhandler).

```
// Non value task

public async Task DownloadAndSaveMultipleAsync(string url1, string filePath1,
string url2, string filePath2)
{
    Task t1 = DownloadAndSaveAsync(url1, filePath1);
    Task t2 = DownloadAndSaveAsync(url2, filePath2);
    await Task.WhenAll(t1, t2);
    // Automatically returns a task.
}

static async Task Main()
{
    var downloader = new Downloader();
    Task downloadTask = downloader.DownloadAndSaveMultipleAsync(
        URL1, "",
        URL2, "");
    await downloadTask;
}

// Value task

async Task<int> ReturnsInt (int param)
{
```

```
  ++param;
  await Task.Delay(10);
  return param;
}

int result =  await ReturnsInt(5);

Task<int> t1 = ReturnsInt(1);
Task<int> t2 = ReturnsInt(2);
await Task.WhenAll(t1, t2);
var result1 = t1.Result;
var result2 = t2.Result;
```

Linq

Linq ermöglicht durch eigene Syntax, Datentransformationen auf Datenquellen (DB, Collection).

Zwei verschiedene Syntaxen sind möglich. Fluent (Extension Methods) und Query Expression (wie SQL nur select am ende)

- .Select(a => a)
- .Where(a => a == 0)
- .OrderBy(a => a.Prop)
- .OrderByDescending(a => a.Prop)
- .GroupBy(a => a.Prop)
- .First()
- .FirstOrDefault()
- .Take(5) Takes first 5 elements from .Select()
- .Average(a => a.Prop)
- .Min(a => a.Prop)
- .Max(a => a.Prop)
- .Sum(a => a.Prop)

Mit new {
Prop1 = var.Prop1,
Prop2 = var.Prop2
} können anonyme Typen returned werden.
Der anonyme Typ kann nur in var gespeichert werden.
Bsp.

```
var stud = students.Select(s => new{Name = s.Name, Avg = s.Grades.Average()});
```

```
// Fluent
var transformedData = data
    .OrderBy( x => x.OrderProp )
```

```
    .Where( x => x<0 )

var avgRevenueByCountry = customers
    .GroupBy( c => c.Location.Country )
    .Select(
        c => new {
            Country = c.Key,
            AvgRevenue = c.Average( c => c.Revenue )
            }
    )

// Query Expr
var transformedData =
    from d in Data
    orderby d.OrderProp ascending
    where d < 0
    select d

var avgRevenueByCountry =
    from c in customers
    group c by c.Location.Country into countryGroup
    select new {
        Country = countryGroup.Key,
        AvgRevenue = countryGroup.Average( c => c.Revenue )
    };

var another = from c in customers
    group c by c.Location.Country into g
    //now you can use aggregate functions on g
    let average = g.Average(c => c.Revenue)
    orderby average
    select new { Country = g.Key, Revenue = average};
```

Nullable Reference Types

Wenn nullable reference types aktiviert werden, werden warnings gegeben falls ein Wert null gesetzt wird.

```
// Per cs file
#nullable enable
```

Mit Tags in csproj

<Nullable>enable

Potentielle Fragen

zu delegates:
event mit