FPROG - WordCount C#

Maramara, Rieder

FP vs. OOP

Pure Functions - Mit Rückgabewert und keine Side-Effects (Lambdas)

```
public static Func<int,int,Tuple<string, string>> GetPathAndFile = (pathIndex,fileIndex) =>
{
    Tuple<string, string> pathAndFile = new Tuple<string, string>(Environment.GetCommandLineArgs()[pathIndex], Environment.GetCommandLineArgs()[fileIndex]);
    return pathAndFile;
};
```

High-Order Function - Map Reduce - bekommt Funktionen als Parameter übergeben

```
public static IEnumerable<TResult> MapReduce<TSource, TMapped, TKey, TResult>(
this IEnumerable<TSource> source,
Func<TSource, IEnumerable<TMapped>> map,
Func<TMapped, TKey> keySelector,
Func<IGrouping<TKey, TMapped>, IEnumerable<TResult>> reduce)
{
    return source.SelectMany(map).GroupBy(keySelector).SelectMany(reduce);
}
```

Lessons Learned

- Lambdas vereinfachen das Testen
- mit C# LINQ kann man die Konzepte vom Funktionalen Programmieren gut umsetzen
- Durchschnittliche Zeit von einem Durchlauf mit Sample Moodle Testfile (~566.546 Wörtern) exklusive Konsolenausgabe: 0,0067 Sekunden

Possible Side-Effects

```
//lambda with 2 arguments
public static Func<string, string, IEnumerable<string>> GetFilesListFromDirectory = (dirPath, fileExtension) =>
{
    //maybe side-effects
    return Directory.EnumerateFiles(dirPath, "*" + fileExtension, SearchOption.AllDirectories);
};
```

Non-Functional Parts

```
// NON-FUNCTIONAL METHOD
1 Verweis
public static void printWords(IEnumerable<KeyValuePair<string, int>> wordlist)
{
    foreach (var c in wordlist)
    {
        Console.WriteLine($"{c.Key}  {c.Value}");
    }
}

// NON-FUNCTIONAL METHOD
1 Verweis
public static void printTime(Stopwatch stopwatch)
{
    Console.WriteLine("Time elapsed: {0}s", stopwatch.Elapsed.TotalSeconds);
}
```

```
//non functional part
printWords(orderedCounts);
printTime(stopwatch);
```

```
static void Main(string[] args)
{
    if (Environment.GetCommandLineArgs().Length > 2)
    {
```

Code review

Live code review + demo