Stilrichtlinien für die Entwicklung der FG-App

Kierán Meinhardt Daniel Richter Jonas Thelemann

25. August 2015

1 Vorwort

Es folgen die Regeln, nach denen neuer Code in der FG-App formatiert und formuliert werden sollte. Das soll dir und den nachfolgenden mitarbeitenden helfen, den Quellcode zu verstehen und zu erweitern.

2 Stilrichtlinien

2.1 Der Dateikopf (Header)

Im Dateikopf wird jeweils der Name der Datei, des Autors, das Erstell- und Änderungsdatum und der genaue Zweck der jeweiligen Datei angegeben. Ein Beispiel:

```
// Datei: StundenplanDownloadHandler.cs
// Autor: Max Mustermann
// Datum: 01.04.2015 (04.02.2018)
//
// Verwaltet das Download-System fuer die Schuelerstundenplaene.
```

2.2 Imports / `usings`

Die Imports befinden sich alle direkt nach dem Header und sind, wenn möglich nach `namespace` gruppiert. (Und, wenn man extrem ordentlich sein möchte, alphabetisch sortiert.[1]) So beispielsweise:

[1] Das können Programme wie Microsoft Visual Studio® automatisch.

```
// Android
using Android.App;
using Android.Content;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;
// Android.Support
using Android.Support.V4.Widget;
using Android.Support.V4.App;
// Koopakiller
using Koopakiller.NewsFeed;
// System
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.IO;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
```

2.3 Namensgebung

Alle Namen und Dokumentationen werden auf Deutsch geschrieben. Entsprechend der Microsoft C# Richtlinien, sind Klassennamen im sogenannten “CamelCase” zu halten. Das heißt, der Anfang jedes Wortes wird großgeschrieben und die Wörter dann ohne Trennzeichen wie _ aneinandergehängt. So sind beispielsweise `HausTuer` und Banane richtig, aber *`Haus_Tuer`*, *`haus_tuer`* und *banane* nicht.

Bei Methoden- und Funktionsnamen verhält es sich genauso.

Bei Feld- und Variablennamen wird zwischen lokalen und globalen Variablen unterschieden. Eine lokale Variable sei eine Variable, die nur innerhalb einer Methode/Funktion als “Zwischenablage” genutzt wird. Globale Variablen werden so wie Klassennamen formatiert (CamelCase).

Für lokale Variablen gilt das Folgende: alles wird komplett kleingeschrieben. und mehrteilige wörter werden durch unterstriche getrennt.

Lokal	Global
banane	Banane

Lokal	Global
`haus_tuer`	`HausTuer`
`bananen_staude`	`BananenStaude`

Konstante globale Variablen werden komplett großgeschrieben und durch _ zwischen den Wörtern aufgetrennt.

```
const int PERMISSIONS_MULTIPLIKATOR = 4;
```

Konstante lokale Variablen werden hingegen wie normale lokale Variablen behandelt.

2.4 Typisierung

Datentypangaben sind in C# ja bekanntlicherweise optional. Das bedeutet, man kann Datentypen wenn man will durch `var` ersetzen.

```
int x = 0;
// ->
var x = 0;
```

Davon raten wir dringend ab, da es die Verständlichkeit des Codes oft senkt. *Eine Ausnahme* dazu ist jedoch, dass man die Datentypen weglassen kann, wenn sie rechts vom Gleichzeichen noch einmal vorkommen. Zum Beispiel lässt sich folgender unnötig angegebener Typ durch `var` ersetzen.

```
DateTime jetzt = DateTime.Now;
// ->
var jetzt = DateTime.Now;
```

2.5 Einrückung & Klammern ({ und })

Die Einrückung (im sogenannten Allman-Stil) folgt 3 einfachen Regeln:

1. Klammern stehen direkt unter dem ersten Zeichen der oberen Zeile.
2. Nach der Klammer folgt ein Zeilenumbruch.
3. Jede Zeile zwischen zwei Klammern wird um *einen Tab* eingerückt.

Ein Beispiel verdeutlicht diese Regeln:

```
class Banane
{
   public Banane()
   {
```

```
      if (false)
      {
         while (1 == 1)
         {
            Console.WriteLine("Banane!");
         }
      }
      else
      {
         return 0;
      }
   }
}
```

Natürlich kann man bei obigem Beispiel, wenn nur eine Anweisung auf ein `if`, `else`, `while` etc. folgt, viele Klammern weglassen. Also:

```
class Banane
{
   public Banane()
   {
      if (false)
         while (1 == 1)
            Console.WriteLine("Banane!");
      else
         return 0;
   }
}
```

2.6 Kommentare

Für Kommentare wird durchgängig der C#-Dokumentationskommentarstil empfohlen. Es muss wahrscheinlich nicht mehr erwähnt werden, dass zu gutem Code auch gute (d.h. verständliche und erklärende) Kommentare gehören.

```
/// <summary>
/// gibt eine Zufaellige Zahl zurueck
/// </summary>
public int GetRandomNumber()
{
   return 42;
}
```

Kommentarwitze werden bei Programmierern aber auch gerne gesehen. Wenn sie gut sind (!).

```
double penetration = 3.1; // hahaha

//                        _
// _._ _..._ .-', _.._(`))
// '-. `  ' /-._.-' ',/
//   )      \            '.
//  / _  _   |             \
// | a   a   /              |
// \  .-.                  ;
//  '-('' ).-'   ,'       ;
//    '-;        |     .'
//       \        \   /
//       | 7 .__ _.-\ \
//       | | |  ``/ /` /
//      /,_| |  /,_/ /
//        /,_/    '`-'
// DIESER CODE WURDE VON UNSEREM
// SICHERHEITS-SCHWEIN GEPRUEFT.
// NICHT GENAUER ANGUCKEN,
// ER FUNsKTIONIERT SCHON!
```

2.7 Freiwillige Regeln

2.7.1 Gleichzeichen-Alignment

Gleichzeichen in Definitionen können vertikal untereinander ausgerichtet werden. Dies ist natürlich nicht unbedingt nötig, ist aber für die ganz perfektionistischen unter uns ein extrem schönes Detail. Gemeint ist das ganze so:

```
List<string> names = Database.GetNames();
int counter   = 0;
string first_name = names[counter];
```