# Pedantic Patrick's programming pattern

In dit document omschrijf ik de programmeer stijl welke we gaan aanhouden. Doordat we allemaal de zelfde stijl en naamgeving gebruiken blijft de code overzichtelijk en eenduidig. (Hopelijk)

## Terminologie

- camelCase: Naamvoering beginnend met een kleine letter, vervolgens elk volg word beginnend met een hoofdletter.
- PascalCase: Naamvoering beginnend met een hoofdletter, vervolgens elk volg word beginnend met een hoofdletter.

---

## 1. Bestanden en folders

### 1.1. Bronbestanden

Houd classes elk in zijn eigen bestand. Naamgeving van het bestand is gelijk aan de naam van de class. Gebruik hiervoor PascalCase en eindig uiteraard op .cs

### 1.2. Folderstructuur

Houd de folder structuur logisch. Verplaats indien nodig bestanden. Een folder heeft altijd een meervoud van het zelfstandige naamwoord als naam. Er zitten immers meerdere van deze zelfstandige naamworden in de zelfde folder, anders voegt de folder niets toe. Uitzondering: als van te voren bekent is dat er later meerdere bestanden aangemaakt zullen worden. Uitzondering: de volgende namen zijn tevens toegestaan: common, misc en config.

#### 1.2.1. Namespaces

De Naam van de namespace is identiek aan de folderstructuur. Houd ook de zelfde bestanden in de zelfde map als de namespace.

---

## 2. Commentaar

### 2.1. Taal

Elk commentaar is in het Engels, de code is immers ook Engels.

## 2.2. Blok commentaar

Zie af van het gebruik van blok commentaar. Indien je dit nodig denkt te hebben naast het documentatie commentaar dan is je documentatie incompleet of je code te obscuur. Werk een van de twee beter uit.

## 2.3. Lijn commentaar

Gebruik lijn commentaar (//) voor het weg commentaren van een code blok. Tip: Gebruik ALT+SHIFT Geef bij aannamen en onduidelijke code wat aanwijzingen in het commentaar. Zal over het algemeen nooit meer dan één regel overschrijden, zo niet dan is je code te obscuur.

## 2.4. Documentatie commentaar

Zet boven elke methode minstens een summary block, wat doet de methode. N.B. het resultaat is (als het goed is) af te lezen aan de methode naam. Daarop hoeft niet diep te worden ingegaan.

```
/// <summary>
/// This class...
/// </summary>
public void DoSomething(Type var)
{
}
```

Kijk op het internet voor alle mogelijk XML items en gebruik deze zinnig.

---

# 3. Globale stijl

## 3.1. Haakjes

Gebruik de standaard stijl van Visual Studio. Om merge conflicten zo veel mogelijk te voorkomen is het zeer aangeraden om voor elke commit ALT+K+D te duwen om alle code netjes te formatteren.

## 3.2. Spatie en tab gebruik

Zie 3.1.

## 3.3. Sexy Code™

Voor de Rock Star programmeur. Code dient zo te worden geschreven dat deze lijkt op de perfecte vriendin voor Patrick; lang, slank en elegant. Maak de code niet te breed dit leest moeilijk, maak de code bij voorkeur wat langer door deze op de volgende regel door te laten lopen. Een oplossing is bij voorkeur simpel en daardoor elegant. Breek methodes altijd op in korte simpelere methodes.

## 3.4. Lengte

Ondanks punt 3.3. spreekt over lengte dient niet elke lange if met een volle else te worden geschreven indien deze alleen een waarde returned. Alleen de return volstaat.

## 3.5. Herhaling

Herhaal nooit je code, ik herhaal: “herhaal nooit je code”, schrijf een methode om dubbele functionaliteit samen te voegen.

## 3.6. Loops

Geneste loops waarvan de body meer dan drie regels overschrijft dient te worden geplaatst in zijn eigen methode. Dit maakt duidelijker wat er bij elke iteratie gebeurt. Schrijf eventueel in het Documentatie commentaar (2.4.) de tijd complexiteit van de code als dat door punt 3.6. weg genomen word.

### 3.6.1 Voorbeeld

```
foreach(var item in list)
{
    Sometimes();
    foreach(var item in list)
    {
        Often();
        if(GetSomeLogic() == null)
        {
            DoIt();
        }
    }
}
```

Herschreven naar

```
foreach(var item in list)
{
    PerformIteration();
}
...
public void PerformIteration()
{
    Sometimes();
```

```
    foreach(var item in list)
    {
        Often();
        if(GetSomeLogic() == null)
        {
            DoIt();
        }
    }
}
```

## 3.7. Switch

Switch altijd op een IEnumerable of getal. Nooit op eens string dit maakt onduidelijk wat de input voor de switch is.

Indien altijd één van de cases dient te worden behandeld, gooi een Exception in de default: case.

---

# 4. Naamgeving

Volgende zijn aanbevelingen geen verplichtingen.

| Soort | Hoofdletter gebruik | Woordsoort | Opmerking |
|---|---|---|---|
| Class | PascalCase | Zelfstandignaamwoord | |
| Interface | PascalCase | Zelfstandignaamwoord | Hoofdletter I gevolgd door PascalCase. |
| Enum type | PascalCase | Zelfstandignaamwoord | Zelfstandignaamwoord is altijd enkelvoud. |
| Enum waarde | PascalCase | Zelfstandignaamwoord | |
| Methodes | PascalCase | Werkwoord | |
| Namespace | PascalCase | Zelfstandignaamwoord | Zie 1.2. & 1.3. voor uitzonderingen. |
| Private Field | camelCase | Zelfstandignaamwoord | |
| Property | PascalCase | Zelfstandignaamwoord | |
| Parameter | camelCase | Zelfstandignaamwoord | |
| Constants | PascalCase | Zelfstandignaamwoord | |

---

# 5. Vaste waarden

Zet niet veranderende waarde altijd in een const veld. Dan is het altijd duidelijk wat het inhoud (bijvoorbeeld: Height, of PlayersCount) en het vergemakkelijkt onderhoud. Er hoeft immers maar een waarde te worden geüpdatet i.p.v. meerdere waardes in meerdere bestanden.

---

# 6. Region blokken

Maak slim gebruik van #Region blokken, hierdoor is het makkelijk de code te beheren en kun je beter focussen op de code. Plaats bijvoorbeeld alle constructors in een region en/of alle properies

# Voorbeeld

Een classe als voorbeeld:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SimpeProject
{
    enum Colors
    {
        Blue,
        Yellow,
        Red
    };
    class Something
    {
        List<int> list;
        /// <summary>
        /// Deze class doet iets.
        /// </summary>
        public Something()
        {

        }
        public void DoSomething(int something)
        {
            foreach (int item in list)
            {
                PerformIteration();
            }
```

```
        }
        public void PerformIteration()
        {
            Sometimes();
            foreach(int item in list)
            {
                Often();
                if (item == null)
                {
                    DoIt();
                }
            }
        }
        public void Often()
        {
            Console.WriteLine("Often");
        }
        public void DoIt()
        {
            Colors color = Colors.Blue;
            switch (color)
            {
                case Colors.Blue:
                    Console.WriteLine("Blue");
                    break;
                case Colors.Red:
                    Console.WriteLine("Red");
                    break;
                case Colors.Yellow:
                    Console.WriteLine("Yellow");
                    break;
                default:
                    throw new System.ArgumentException("color has wrong value");
            }
        }

        public void Sometimes()
        {
            Console.WriteLine("Sometimes");
        }
    }
}
```