Trabalho de WEB 2

Object Calisthenics

Nome: Murilo da Silva Barbosa

Código: 816577

Curso: Engenharia de Software

Matéria: Programação web 2

# Object Calisthenics

Ele e um conjunto de regras e boas ações em programação que tornam o código melhor.

Essas regras tem como medida melhorar sua manutenibilidade, legibilidade, testável e compreensão do código.

Essas regras são divididas em 9 partes, que são:

## Apenas um nível de identação por método

Não deixe sua função com muitos métodos. A tentativa é de deixar a sua função mais compreensível possível e dividir em outras funções as todas as atividades de seu método.

De:

public String LerDados(string[][] data)

{

    StringBuilder buf = new StringBuilder();    // Primeiro nivel

    for (int i = 0; i < 10; i++)

    {

         // Segundo nivel

         for (int j = 0; j < 10; j++)

         {

              // Primeiro nivel

              buf.Append(data[i][j]);

         }

         buf.Append("\n");

     }     return buf.ToString();

}

Para:

public String LerDados(string[][] data)

{

     StringBuilder buf = new StringBuilder();

     LerLinhas(data, buf);     return buf.ToString();

}private void LerLinhas(string[][] data, StringBuilder buf)

{

     for (int i = 0; i < 10; i++)

     {

        LerColunas(data, buf, i);

     }

}private static void LerColunas(string[][] data, StringBuilder buf, int i)

{

     for (int j = 0; j < 10; j++)

     {

        buf.Append(data[i][j]);

     }

     buf.Append("\n");

}

## Evitar o uso do “else” nos códigos

Evite o uso da palavra condicionada “else” em seu código.

De:

protected function indexAction()

{

    if ($this->security->isGranted('ADMIN')) {

        $view = 'admin/index.html.twig';

    } else {

        $view = 'home/access\_denied.html.twig';

    }

    return $this->render($view);

}

Para:

<?php

    protected function indexAction()

    {

        if ($this->security->isGranted('ADMIN')) {

            return $this->render('admin/index.html.twig');

        }

        return $this->render('home/access\_denied.html.twig');

    }

## Envolva seus tipos primitivos

Deixe suas variáveis encapsuladas.

De:

public class Pessoa

{

public string CPF { get; set; }

}

Para:

public class Pessoa

{

    public CPF CPF { get; set; }

}public class CPF

{

    public string Numero { get; set; }

    public string NumeroFormatado => FormataNumero();    public CPF (string numero)

    {

        if (!EstaValido(numero))

            throw new CPFInvalidoException(numero);        this.Numero = numero;

    }    private void EstaValido(numero)

    {

        //Codigo..

    }    private String FormataNumero()

    {

        //Codigo

    }

}

## Envolva as collection em classes

Qualquer classe que tenha uma coleção não deve ter outras variáveis de membro.

public class ProductCollection : IList<Product>

{

    //Codigo

}

## Use apenas um ponto por linha

Essa regra se diz que você não deve encadear métodos, e sim usar objetos que fazem parte do mesmo contexto.

public class Endereco

{

    public Cidade Cidade { get; set; }

}

public class Cidade

{

    public Estado Estado { get; set; }

}

public class Estado

{

    public string Nome { get; set; }

}

static void Main(string[] args)

{

    var endereco = new Endereco();

    Console.WriteLine(endereco.Cidade.Estado.Nome);

}

## Não abreviar

Não abrevie variáveis difíceis de entender. O uso de abreviações pode acarretar a dificuldade de uma pessoa de entender o código

## Mantenha todas as classes pequenas

É recomendado que as classes não passem de 50 linhas e que os pacotes não tenham mais de 10 arquivos.

## Não tenha mais de 2 variáveis de instancia em sua classe

Esta regra está relacionada a questão de responsabilidade de cada instancia. Se ela estiver com mais variáveis do que deveria, ela poderia estar recebendo mais responsabilidade do que devia.

## Não use Getters e Setters

A retirada dos métodos *getters* e *setters* pode adicionar muitos benefícios quando aplicamos essa ideia. Reduzimos a duplicação de regras e damos uma melhor compreensão àquele objeto. Também enriquecemos nosso objeto com lógica e métodos mais valiosos e significativos, não apenas o usando como uma classe com apenas dados.