INTRODUÇÃO À PROGRAMAÇÃO COM C#

contato@rafaelpadilla.net

BOAS PRÁTICAS COM C#

BOAS PRÁTICAS COM C#

- Ajudam na transferência de conhecimento entre projetos
- Ajudam no aprendizado da linguagem
- Organizam e deixam o código limpo
- Consistência
- Ajudam na manutenção do código

BOAS PRÁTICAS COM C#

- “People First, Computers Second.”
- Use as normas do time ou da empresa
- Quebre trechos em métodos sempre que o trecho de código for reutilizável
- Declare propriedades e variáveis no topo da classe
- Use #region para organizar seu código
- Tenha bom senso!!!

BOAS PRÁTICAS COM C#

Identificadores	Case	Exemplo
Classes	Pascal	AppDomain
Tipos de Enum	Pascal	ErrorLevel
Valores de Enum	Pascal	FatalError
Eventos	Pascal	ValueChanged
Classes de Exceções	Pascal	WebException
Interfaces	Pascal	IDisposable
Métodos	Pascal	ToString
Namespace	Pascal	System.Drawing
Parâmetros	Camel	typeName
Propriedades	Pascal	BackColor
Constantes	Pascal / Upper	MaxValue / MAXVALUE

BOAS PRÁTICAS COM C#

Identificadores	Case	Exemplo
Variáveis privadas	Camel	firstName; _firstName
Nomes de Controles	Camel	btnClose
Variáveis booleanas	Pascal	IsVisible; HasChildren; CanExecute

COMANDOS REPETIÇÃO

- for
- foreach
- do...while
- while

DESAFIO #3

DESAFIO #3 (INVERTER CARACTERES)

Crie um programa (Windows Form) com C# capaz de inverter os caracteres de uma frase informada pelo usuário. Desconsidere os caracteres ".", ";", "!", "?", "," e os espaços " ".
A frase de saída deverá ser totalmente em letras minúsculas.

Ex:
Frase de entrada: "Olá, tudo bem com você?"
Frase de saída: "êcovmocmebodutálo"

contato@rafaelpadilla.net

GENERICS

Muitas vezes ao criarmos uma classe, precisamos que algumas de suas variáveis adotem diferentes tipos, sendo estes definidos de acordo com a sua instância.

Exemplos:

```
List<string> listaDeStrings = new List<string>();
List<int> listaDeInteiros = new List<int>();
List<Carro> listaDeCarros = new List<Carro>();
List<Cliente> listaDeClientes = new List<Cliente>();
```

COLEÇÕES

- List
- Dictionary
- Queue
- SortedList
- Stack

TRATAMENTO DE EXCEÇÕES

- Try
- Catch
- Finally

```
void TestTryBlock()
{
  try
  {
    // code goes here...
  }
  catch (Exception ex)
  {
    // code goes here...
  }
  finally
  {
    // code goes here..
  }
}
```

DESAFIO #4 (FREQUÊNCIA PALAVRAS)

DESAFIO #4 (FREQUÊNCIA PALAVRAS)

Crie um programa (Windows Form) com C# capaz de informar a frequência que cada palavra aparece em um texto de forma ordenada por quantidade de ocorrências.

contato@rafaelpadilla.net

HANDS-ON (PARTE 1)

- ✓ Métodos e Parâmetros
- ✓ Sobrecarga de Métodos
- ✓ C# e a Orientação a Objetos
- ✓ Windows Forms
- • Parsing a web page
- • Acessando um webservice
- • Json e XML
- • Consumindo uma dll externa
- ✓ Encapsulamento
- • Delegates
- • Async & Await

- • Alterar UI de uma outra thread
- ✓ Get/Set
- • Utilizando Variáveis (Tipo Referência)
- • Referência em Memória
- ✓ Utilizando os tipos primitivos
- ✓ Hierarquia dos objetos
- • Herança em C#
- • LINQ e Lambda

DESAFIO #5 (API PIADAS)

DESAFIO #5 (API PIADAS)

Neste desafio, você deverá criar um **Wrapper** de uma página HTML, capaz de mostrar em uma List<T>, os objetos capturados a partir do código HTML.

Para este desafio, não é necessário criar um projeto Windows Forms. A interface com o usuário é opcional.

É necessário obter as 15 primeiras piadas do site http://www.osvigaristas.com.br/piadas/ em forma de objetos (List<Piada>) da seguinte classe:

```
public class Piada
{
    public string Nome { get; set; }
    public string Url { get; set; }
    public string Corpo { get; set; }
    public string Categoria { get; set; }
    public DateTime DataPublicacao { get; set; }
    public int VotosPositivos { get; set; }
    public int VotosNegativos { get; set; }
    public int VotosMedios { get; set; }
}
```

DESAFIO #6 (XML DE CARROS)

DESAFIO #6 (XML DE CARROS)

Neste desafio, você deverá criar um pequeno sistema de uma loja de carros. Com este sistema o usuário poderá:

1) Selecionar um arquivo XML e *deserializar* as informações contidas neste arquivo

1.1) Este arquivo deverá ter a seguinte estrutura:

```
<?xml version="1.0" encoding="utf-8"?>
<ElementoRaiz>
<DataCriacao>15 de maio de 2014</DataCriacao>
    <Carros>
      <Carro>
        <Codigo>0001</Codigo>
        <Marca>Nissan</Marca>
        <Modelo>Sentra</Modelo>
        <Ano>2015</Ano>
        <Preço>50000</Preço>
      </Carro>
      <Carro>
        <Codigo>0002</Codigo>
        <Marca>Toyota</Marca>
        <Modelo>Corolla</Modelo>
        <Ano>2014</Ano>
        <Preço>58000</Preço>
      </Carro>
      <Carro>
        <Codigo>0003</Codigo>
        <Marca>Chevrolet</Marca>
        <Modelo>Chevette</Modelo>
        <Ano>1993</Ano>
        <Preço>12000</Preço>
      </Carro>
    </Carros>
</ElementoRaiz>
```

Lista de carros
(min 10 cadastros)

1.2) Este arquivo deverá ter no mínimo 10 carros cadastrados.

DESAFIO #6 (XML DE CARROS)

2) Visualizar em uma UI (Windows Form) as informações contidas no XML

2.1) Esta UI deverá ter 4 ComboBoxes contendo todas as Marcas, Modelos, Anos e Preços disponíveis no XML. Não poderão ter marcas, modelos, anos e preços repetidos.

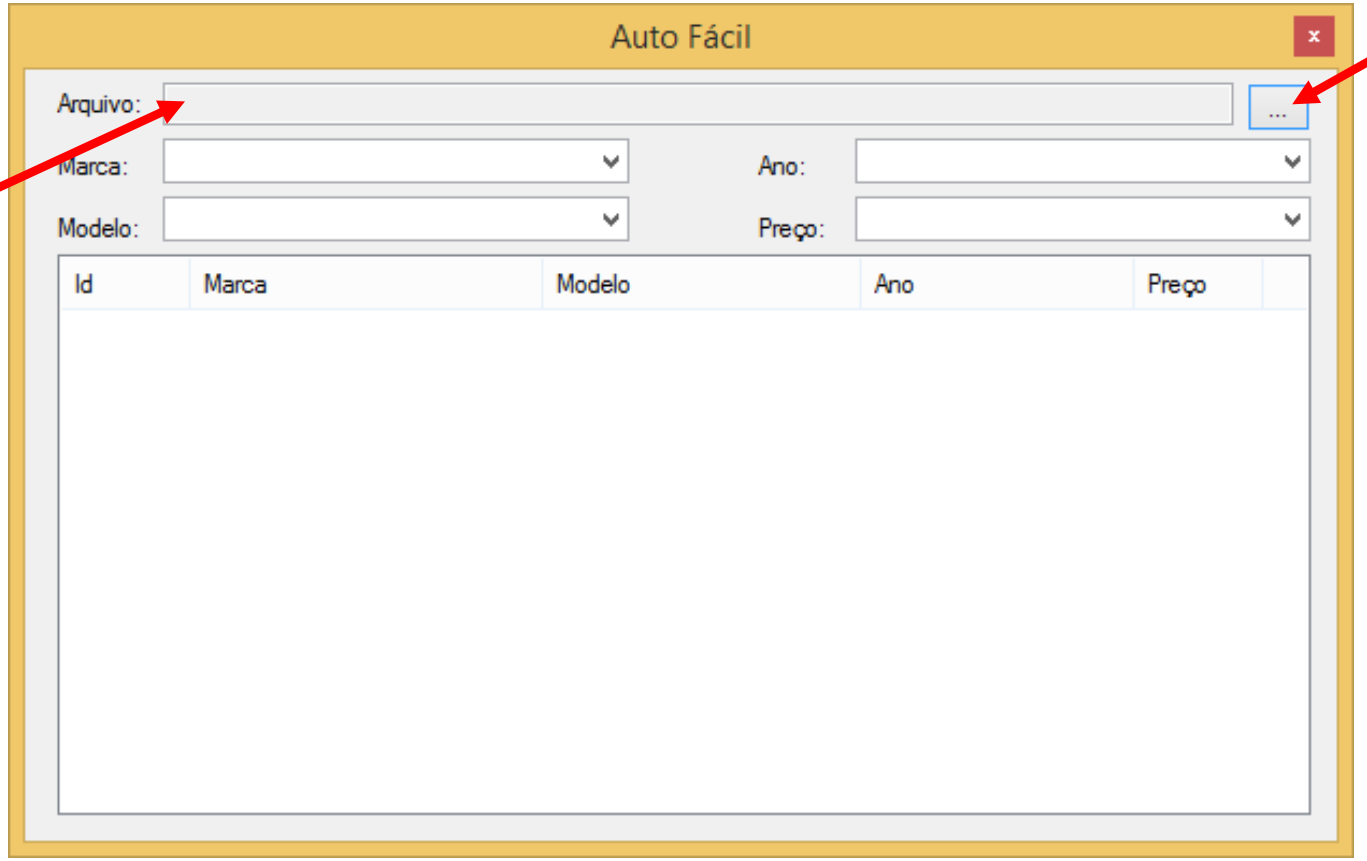
2.2) As ComboBoxes serão filtros para que as informações do XML sejam apresentadas em uma ListView.

2.3) Um item obrigatório em todas as ComboBoxes é uma opção que quando selecionada traga todos os itens desta categoria sem filtros. Esta opção pode ser “(Todas Marcas)”, “(Todos Modelos)”, “(Todos Anos)” e “(Todos Preços)”.

3) Utilize expressões Lambda ou LINQ para realizar a filtragem dos itens.

4) Utilize o exemplo de UI a seguir

DESAFIO #6 (XML DE CARROS)



Botão para escolha do XML

Caminho do XML

ComboBoxes de filtragem

ListView mostrando os resultados

Exemplo de UI Windows Form

MODIFICADORES DE ACESSO

Todos os tipos e membros possuem nível de acesso que controlam se podem ser usados em outras classes no mesmo assembly ou assemblies diferentes.

- public
  Membros ou tipos **public** podem ser acessados na mesma classe ou por outras classes do mesmo assembly ou por outro assembly.
- private
  Membros ou tipos **private** podem ser acessados somente na mesma classe.
- protected
  Membros ou tipos **protected** só podem ser acessados na mesma classe ou em uma classe derivada desta classe. (proteger filhos)
- internal
  Membros ou tipos **internal** podem ser acessados por qualquer classe apenas no mesmo assembly.
- protected internal
  protected + internal

ORIENTAÇÃO A OBJETOS

```
class ClassePrivada
{
    //Propriedades
    private int PropriedadePrivada { get; set; }
    public int PropriedadePublica { get; set; }
    //Propriedades Estáticas
    private static int PropriedadeEstaticaPrivada { get; set; }
    public static int PropriedadeEstaticaPublica { get; set; }

    //Variáveis
    private int variavelPrivada;
    public int variavelPublica;
    //Variáveis Estáticas
    private static int variavelEstaticaPrivada;
    public static int variavelEstaticaPublica;

    //Métodos
    private void MetodoPrivado()
    { }
    public void MetodoPublico()
    { }
    //Métodos Estáticos
    private static void MetodoEstaticoPrivado()
    { }
    public static void MetodoEstaticoPublico()
    { }
}
```

Fora da classe, é possível acessar?

- ✗ propriedade **PropriedadePrivada**
- ✓ propriedade **PropriedadePública**
- ✗ propriedade **PropriedadeEstaticaPrivada**
- ✓ propriedade **PropriedadeEstaticaPublica**

- ✗ variável **variavelPrivada**
- ✓ variável **variavelPublica**
- ✗ variável **variavelEstaticaPrivada**
- ✓ variável **variavelEstaticaPublica**

- ✗ método **MetodoPrivado**
- ✓ método **MetodoPublico**
- ✗ método **MetodoEstaticoPrivado**
- ✓ método **MetodoEstaticoPublico**

MODIFICADORES DE ACESSO-PROTECTED

```
class ClassePai
{
    private int variavelPrivate;
    public int variavelPublic;
    protected int variavelProtected;

    void MetodoProtected()
    {
        /* Lembrando que:
        Variáveis protected podem ser acessadas na mesma classe ou em classe derivada desta classe
        Variáveis public podem ser acessadas na mesma classe ou por outras classes do mesmo ou outro assembly
        Variáveis private podem ser acessadas somente na mesma classe */

        variavelProtected = 1; //Permitido
        variavelPrivate = 2; //Permitido
        variavelPublic = 3; // Permitido
    }
}
```

```
class ClasseFilha1 : ClassePai
{
    void Metodo()
    {
        variavelPublic = 123;
        variavelProtected = 123; //Permitido, pois é uma variável da classe pai
        variavelPrivate = 123; //Não permitido, pois esta variável é privada ao uso exclusivo da ClassePai

        //Acessando através de um um objeto
        ClasseFilha1 obj = new ClasseFilha1();
        obj.variavelPublic = 123; //Permitido
        obj.variavelProtected = 123; //Permitido, pois é uma variável da classe pai
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        ClassePai baseObj = new ClassePai();
        baseObj.variavelPublic = 123; //Permitido
        baseObj.variavelPrivate = 123; //Não permitido, pois esta variável é privada ao uso exclusivo da ClassePai
        baseObj.variavelProtected = 123; //Não permitido, pois só pode ser acessada na ClassePai ou
                                         //em classes filhas da ClassePai
    }
}
```

HANDS-ON (PARTE 2)

- Utilizando Classes Sealed (lacradas)
- Classes Abstratas

CLASSES ABSTRATAS

```
public class EmpregadoComContrato
{
    public int Id { get; set; }
    public string Nome { get; set; }
    public string Sobrenome { get; set; }
    public float SalarioAnual { get; set; }

    public string GetNomeCompleto()
    {
        return this.Nome + " " + this.Sobrenome;
    }

    public float GetSalarioMensal()
    {
        return SalarioAnual / 12;
    }
}
```

```
public class EmpregadoHorista
{
    public int Id { get; set; }
    public string Nome { get; set; }
    public string Sobrenome { get; set; }
    public float SalarioHora { get; set; }
    public int HorasTrabalhadas { get; set; }

    public string GetNomeCompleto()
    {
        return this.Nome + " " + this.Sobrenome;
    }

    public float GetSalarioMensal()
    {
        return this.SalarioHora * this.HorasTrabalhadas;
    }
}
```

CLASSES ABSTRATAS

```
public abstract class BaseEmpregado
{
    public int Id { get; set; }
    public string Nome { get; set; }
    public string Sobrenome { get; set; }

    public string GetNomeCompleto()
    {
        return this.Nome + " " + this.Sobrenome;
    }

    public abstract float GetSalarioMensal();
}
```

Obriga classes que a implementam a fazer um **override** do método **abstrato.**

Não é possível implementar uma classe abstrata!

```
public class EmpregadoComContrato : BaseEmpregado
{
    public float SalarioAnual { get; set; }

    public override float GetSalarioMensal()
    {
        return SalarioAnual / 12;
    }
}

public class EmpregadoHorista : BaseEmpregado
{
    public float SalarioHora { get; set; }
    public int HorasTrabalhadas { get; set; }

    public override float GetSalarioMensal()
    {
        return this.SalarioHora * this.HorasTrabalhadas;
    }
}
```

HANDS-ON (PARTE 2)

- Interfaces
- Criando e Destruindo Objetos (instâncias)
- Threads
- LINQ
- Lambda

contato@rafaelpadilla.net

www.linkedin.com/in/rafael-padilla

https://github.com/rafaelpadilla