Universidade Estadual do Paraná
Campus Apucarana

RAFAEL FRANCISCO FERREIRA

Trabalho Teórico de P.O.O. (C Sharp)

APUCARANA
2017

Sumário

1. História

A linguagem C# faz parte do conjunto de ferramentas oferecidas na plataforma .NET e surge como uma linguagem simples, robusta, orientada a objetos, fortemente tipada e altamente escalável a fim de permitir que uma mesma aplicação possa ser executada em diversos dispositivos de *hardware*, independentemente destes serem PCs, *handhelds* ou qualquer outro dispositivo móvel.

O avanço das ferramentas de programação e dos dispositivos eletrônicos inteligentes, criou problemas e novas exigências. As novas versões de componentes compartilhados eram incompatíveis com o *software* antigo. Os desenvolvedores reconheceram a necessidade de *software* que fosse acessível para qualquer um e disponível por meio de praticamente qualquer tipo de dispositivo. Para tratar dessas necessidades, a *Microsoft* anunciou sua iniciativa .NET e a linguagem de programação C#.

Durante o desenvolvimento da plataforma .NET, as bibliotecas foram escritas originalmente numa linguagem chamada *Simple Managed C* (SMC), que tinha um compilador próprio. Mas, em Janeiro de 1999, uma equipe de desenvolvimento foi formada por Anders Hejlsberg, que fora escolhido pela *Microsoft* para desenvolver a linguagem. Dá-se inicio à criação da linguagem chamada *Cool*. Um pouco mais tarde, em 2000, o projeto .NET era apresentado ao público na *Professional Developers Conference* (PDC), e a linguagem Cool fora renomeada e apresentada como C#.

A criação da linguagem, embora tenha sido feita por vários programadores, é atribuída principalmente a Anders, hoje um *Distinguished Engineer* na Microsoft. Ele fora o arquiteto de alguns compiladores da Borland, e entre suas criações mais conhecidas estão o Turbo Pascal e o Delphi.

A Microsoft submeteu o C# à ECMA para uma padronização formal. Em Dezembro de 2001 a associação liberou a especificação ECMA-334 *Especificação da Linguagem C#*. Em 2003 tornou-se um padrão ISO. Há algumas implementações em desenvolvimento, destacando-se a Mono, implementação open source da Novell, o dotGNU e o Portable.NET, implementações da Free Software Foundation, e o BDS 2008, implementação da CodeGear.

1.1 Origem do nome

Pensava-se que o nome "C#" viria duma sobreposição de quatro símbolos +, dando a impressão de ++++, uma alusão à continuação do C++. Entretanto, a cerquilha de "C#" se refere ao sinal musical sustenido, que aumenta em meio tom uma nota musical.

Porém, devido a limitações técnicas (fontes padrões, navegadores, etc) e o fato do símbolo não estar presente nos teclados, o cerquilha (#) foi escolhido para ser usado no nome escrito. Essa convenção é refletida no *ECMA-334 C# Language Specification*, a especificação técnica da linguagem. Entretanto, em determinados lugares, como em propagandas e capas de livros, é usado o símbolo de sustenido.

2. Principais características

O C# é uma linguagem de programação visual dirigida por eventos e totalmente orientada a objetos. Permite um novo grau de intercâmbio entre linguagens (componentes de software de diferentes linguagens podem interagir). Os desenvolvedores podem empacotar até software antigo, para trabalhar com novos programas C#. Além disso, os aplicativos C# podem interagir pela Internet usando padrões do setor, como *SOAP* (protocolo de acesso a objetos simples) e *XML* (linguagem de marcação extensível).

O C# tem raízes em C, C++ e Java, adaptando os melhores recursos de cada linguagem e acrescentando novas capacidades próprias. Ele fornece os recursos que são mais importantes para os programadores, como programação orientada a objetos,*strings*, elementos gráficos, componentes de interface com o usuário gráfica (GUI), tratamento de exceções, múltiplas linhas de execução, multimídia (áudio, imagens, animação e vídeo), processamento de arquivos, estruturas de dados pré-empacotadas, processamento de banco de dados, redes cliente/servidor com base na Internet e na *World Wide Web* e computação distribuída.

Dentre as características essenciais do C# podemos citar:

- Simplicidade: os projetistas de C# costumam dizer que essa linguagem é tão poderosa quanto o C++ e tão simples quanto o *Visual Basic*;
- Completamente orientada a objetos: em C#, qualquer variável tem de fazer parte de uma classe;
- Fortemente tipada: isso ajudará a evitar erros por manipulação imprópria de tipos e atribuições incorretas;
- Gera código gerenciado: assim como o ambiente .NET é gerenciado, assim também é o C#;
- Tudo é um objeto: *System.Object* é a classe base de todo o sistema de tipos de C#;
- Controle de versões: cada *assembly* gerado, seja como *EXE* ou *DLL*, tem informação sobre a versão do código, permitindo a coexistência de dois assemblies homônimos, mas de versões diferentes no mesmo ambiente;
- Suporte a código legado: o C# pode interagir com código legado de objetos COM e DLLs escritas em uma linguagem não-gerenciada;
- Flexibilidade: se o desenvolvedor precisar usar ponteiros, o C# permite, mas ao custo de desenvolver código não-gerenciado, chamado "*unsafe*";

- Linguagem gerenciada: os programas desenvolvidos em C# executam num ambiente gerenciado, o que significa que todo o gerenciamento de memória é feito pelo *runtime* via o *GC (Garbage Collector*).

3. Exemplos em códigos-fonte

3.1 Exemplo de classe:

```
class Carro
{
   private string cor;

   public Carro(string cor)
   {
      this.cor = cor;
   }

   public string Descricao()
   {
      return "Esse carro é " + Cor;
   }

   public string Cor
   {
      get { return cor; }
      set { cor = “Preto”; }
   }
}
```

3.2 Exemplo de instância de objeto:

```
class Animal
{
  // Atributo
  protected string especie;

  // Construtor
  public Animal(string especie)
  {
    this.especie = especie;
  }

  // Execução
  static void Main(string[] args)
  {
    // Instâncias
    Animal cachorro = new Animal("Canis lupus familiaris");
    Animal gato = new Animal("Felis catus");
    Animal lobo = new Animal("Canis lupus");
  }
}
```

3.3 Exemplo de herança:

```
public class Conta
{
 public virtual void Saca(double valor)
 {
   this.Saldo -= valor;
 }

   // Resto do código da classe
}

// Arquivo ContaPoupanca.cs
public class ContaPoupanca : Conta
{
 public override void Saca(double valor)
 {
   this.Saldo -= (valor + 0.10);
 }
}
```

3.4 Exemplo de polimorfismo:

```
//Interface para validar o documento
namespace Polimorfismo
{
   interface Validador
   {
      Boolean validarDocumento();
   }
}
//Pessoa
namespace Polimorfismo
{
   abstract class Pessoa: Validador
   {
      public string Nome { get; set; }
      public void escreverNome()
      {
         Console.Write(this.Nome);
      }
      public virtual bool validarDocumento()
      {
         throw new NotImplementedException();
      }
   }
}
//Pessoa Física
namespace Polimorfismo
{
   class Fisica : Pessoa
   {
      public string Cpf { get; set; }
      public override bool validarDocumento()
      {
         return this.Cpf.Length == 11;
      }
   }
}
//Pessoa Jurídica
namespace Polimorfismo
{
   class Juridica : Pessoa
   {
      public string Cnpj { get; set; }
      public override bool validarDocumento()
      {
         return this.Cnpj.Length == 14;
      }
   }
}
```

3.5 Exemplo de classe abstrata:

```
public class ContaCorrente : Conta
{
  public override void Saca(double valor)
  {
    this.Saldo -= (valor + 0.10);
  }
//  ...
}

public class ContaPoupanca : Conta
{
  public override void Saca(double valor)
  {
    this.Saldo -= valor;
  }
//  ...
}
```

4. Exemplo de programa em código fonte

```
namespace ExemploCalculoFrete
{
        public partial class Form1 : Form
        {
                public Form1()
                {
                        InitializeComponent();
                }
                public void CalcularFrete()
                {
                        //Declaro as variáveis para o nome, valor e estado
                        string nome = txtNome.Text;
                        decimal valor = decimal.Parse(txtValor.Text);
                        string estado = cboEstado.SelectedItem.ToString();
                        //Declaro e inicializo as variáveis frete e total
                        decimal frete = 0, total = 0;
                        //Faço o teste condicional para calcular o valor do frete
                        if (valor > 1000)
                        {
                                frete = 0;
                        }
                        else
                        {
                                switch (estado)
                                {
                                        case "SP": frete = 5;
                                        break;
                                        case "RJ": frete = 10;
                                        break;
```

```
                                case "AM": frete = 20;
                                break;
                                default: frete = 15;
                                break;
                        }
                }
                //Armazeno na variável total o valor digitado mais o frete que acabei de
                calcular
                total = valor + frete;
                //Armazeno os valores nos respectivos labels
                lblValorCompra.Text = valor.ToString("C");
                lblValorFrete.Text = frete.ToString("C");
                lblValorTotal.Text = total.ToString("C");
        }
        public void LimparCampos()
        {
                //Finalizando, crio este método para limpar as variáveis
                txtNome.Text = string.Empty;
                txtValor.Text = string.Empty;
                cboEstado.SelectedValue = string.Empty;
                lblValorCompra.Text = string.Empty;
                lblValorFrete.Text = string.Empty;
                lblValorTotal.Text = string.Empty;
        }
        private void btnCalcular_Click(object sender, EventArgs e)
        {
                CalcularFrete();
        }
        private void btnLimpar_Click(object sender, EventArgs e)
        {
                LimparCampos();
        }
    }
}
```