

Curso de C#: Criação de Sistemas Desktop

Por: António Mantente

Aula 1: Fundamentos da Programação em C#

Objetivo:

Estabelecer uma base sólida em programação com C#, abordando conceitos fundamentais como variáveis, estruturas de controle, funções e classes.

1. Introdução ao C# e ao .NET Framework

O que é C#?

C# é uma linguagem de programação orientada a objetos criada pela Microsoft. Ela é usada para desenvolver uma ampla gama de aplicativos, incluindo sistemas desktop, web e mobile. A linguagem foi projetada para ser simples, moderna e segura, facilitando o desenvolvimento de aplicações robustas.

História e Evolução:

C# foi lançado em 2000 junto com o .NET Framework. Desde então, passou por várias atualizações, adicionando recursos modernos e poderosos. A combinação de C# com o .NET Framework oferece uma plataforma rica para o desenvolvimento de aplicações.

Arquitetura do .NET Framework:


O .NET Framework fornece dois componentes principais:

- **Common Language Runtime (CLR):** Gerencia a execução de programas e oferece serviços como gerenciamento de memória, segurança e tratamento de exceções.
- **Base Class Library (BCL):** Uma vasta biblioteca de classes que oferece suporte a operações comuns, como entrada/saída, manipulação de dados e gráficos.

2. Instalação e Configuração do Ambiente de Desenvolvimento (VS2013)

Instalando o Visual Studio 2013:

Baixe e instale o Visual Studio 2013. Esta IDE é usada para escrever, compilar e depurar código C#. Ela fornece ferramentas integradas para facilitar o desenvolvimento de software.

 Instalar o visual studio!

Configuração inicial:

1. Crie um novo projeto em C#:

- **Tipo de projeto:** Console Application.
- **Nome do projeto:** PrimeiroProjeto.

2. Estrutura básica do projeto:

- O projeto contém arquivos como **Program.cs** que é o ponto de entrada do programa.

Explorando o Ambiente do Visual Studio:

- **Solution Explorer:** Exibe os arquivos e pastas do projeto.
- **Editor de Código:** Onde o código é escrito e editado.
- **Janela de Saída:** Mostra mensagens de erro e resultados de build.

Primeiro Programa em C#:

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("Hello, World!");
    }
}
```

Explicação:

- **using System;**: Importa o namespace **System**, que contém classes básicas como **Console**.
- **class Program { }:** Define uma classe chamada **Program**.
- **static void Main():** Método principal onde o programa começa a executar.
- **Console.WriteLine("Hello, World!");**: Exibe a mensagem "Hello, World!" no console.

3. Manipulação de Dados em C#

Variáveis e Constantes:

- **Variáveis:** Armazenam dados temporários. Exemplo:

```
int idade = 24;
double salario = 2500.50;
```

- **Constantes:** Valores imutáveis durante a execução do programa. Exemplo:

```
const double PI = 3.14159;
```

Listas:

Uma lista é uma coleção que pode armazenar múltiplos elementos do mesmo tipo.

```
List<string> frutas = new List<string> { "Maçã", "Banana", "Laranja" };
frutas.Add("Manga");
frutas.Remove("Banana");
```

Explicação:

- **List<string>**: Cria uma lista de strings.
- **.Add("Manga")**: Adiciona um novo item à lista.
- **.Remove("Banana")**: Remove um item da lista.

Operações básicas:

Operações matemáticas podem ser feitas diretamente em variáveis:

```
int a = 10;
int b = 5;
int soma = a + b;
Console.WriteLine("Soma: " + soma);
```

4. Estruturas de Controle e Repetição

Estruturas de Controle:

- **if e else:**

```
int idade = 18;
if (idade >= 18)
{
    Console.WriteLine("Você é maior de idade.");
}
else
{
    Console.WriteLine("Você é menor de idade.");
}
```

- **switch:**

```
char letra = 'A';
switch (letra)
{
    case 'A':
```

```
        Console.WriteLine("Letra A");  
        break;  
    case 'B':  
        Console.WriteLine("Letra B");  
        break;  
    default:  
        Console.WriteLine("Outra letra");  
        break;  
}
```

Vamos aprimorar a seção sobre **Estruturas de Repetição** com explicações detalhadas de cada estrutura, incluindo a sintaxe, a declaração de variáveis e exemplos de uso em C#.

Estruturas de Repetição

As estruturas de repetição, também conhecidas como laços ou loops, são usadas para executar repetidamente um bloco de código enquanto uma condição específica for verdadeira. Em C#, temos três estruturas principais de repetição: **for**, **while**, e **do-while**.

1. Estrutura **for**

A estrutura **for** é usada quando você sabe, de antemão, o número de vezes que o loop deve ser executado. Ela é ideal para situações onde o contador de iterações é necessário.

Sintaxe:

```
for (inicialização; condição; incremento/decremento)  
{  
    // Bloco de código a ser repetido  
}
```

- **Inicialização:** Declara e inicializa uma variável de controle (geralmente um contador).
- **Condição:** Verifica se o loop deve continuar. Se for **true**, o loop continua; se for **false**, o loop para.
- **Incremento/Decremento:** Atualiza a variável de controle após cada iteração.

Exemplo:

```
for (int i = 0; i < 5; i++)  
{  
    Console.WriteLine("Valor de i: " + i);  
}
```

Explicação:

- **int i = 0;** Declara e inicializa o contador **i** com 0.

- `i < 5`; Define a condição para que o loop continue enquanto `i` for menor que 5.
- `i++`: Incrementa `i` em 1 a cada iteração.

Resultado:

```
Valor de i: 0
Valor de i: 1
Valor de i: 2
Valor de i: 3
Valor de i: 4
```

2. Estrutura `while`

A estrutura `while` é usada quando o número de iterações não é conhecido com antecedência e a repetição depende de uma condição que será avaliada a cada iteração.

Sintaxe:

```
while (condição)
{
    // Bloco de código a ser repetido
}
```

- **Condição:** Enquanto for `true`, o loop continuará executando; se for `false`, o loop para.

Exemplo:

```
int contador = 0;

while (contador < 3)
{
    Console.WriteLine("Contador: " + contador);
    contador++;
}
```

Explicação:

- `int contador = 0`; Declara e inicializa a variável `contador`.
- `contador < 3`: Define a condição para que o loop continue enquanto `contador` for menor que 3.
- `contador++`; Incrementa `contador` em 1 a cada iteração.

Resultado:

```
Contador: 0
Contador: 1
```

Contador: 2

3. Estrutura **do-while**

A estrutura **do-while** é similar ao **while**, mas com uma diferença importante: o bloco de código é executado ao menos uma vez antes da condição ser avaliada.

Sintaxe:

```
do
{
    // Bloco de código a ser repetido
} while (condição);
```

- **Condição:** Avaliada após a execução do bloco de código. Se for **true**, o loop continua; se for **false**, o loop para.

Exemplo:

```
int numero;

do
{
    Console.WriteLine("Digite um número positivo: ");
    numero = int.Parse(Console.ReadLine());
} while (numero <= 0);
```

Explicação:

- **int numero;**: Declara a variável **numero** sem inicializar.
- **do { ... }:** O bloco de código é executado antes da condição ser avaliada.
- **numero <= 0:** O loop continua enquanto **numero** for menor ou igual a zero.

Neste exemplo, o loop garante que o usuário insira um número positivo.

Comparação entre as Estruturas:

- **for:** Ideal quando você sabe o número exato de iterações.
- **while:** Usado quando a condição de continuação é avaliada no início do loop.
- **do-while:** Garantia de que o bloco de código seja executado ao menos uma vez, com a condição avaliada no final.

- **for:**

```
for (int i = 0; i < 5; i++)
{
    Console.WriteLine("Valor de i: " + i);
}
```

- **while:**

```
int contador = 0;
while (contador < 3)
{
    Console.WriteLine("Contador: " + contador);
    contador++;
}
```

- **do-while:**

```
int numero;
do
{
    Console.WriteLine("Digite um número positivo: ");
    numero = int.Parse(Console.ReadLine());
} while (numero <= 0);
```

5. Funções e Classes em C#

Funções:

Uma função é um bloco de código que executa uma tarefa específica.

```
static int Soma(int a, int b)
{
    return a + b;
}

int resultado = Soma(10, 5);
Console.WriteLine("Resultado da soma: " + resultado);
```

Explicação:

- **static int Soma(int a, int b):** Define uma função que recebe dois parâmetros e retorna a soma deles.
- **return a + b;** Retorna o resultado da soma.

Classes e Objetos:

Classes são moldes para criar objetos que encapsulam dados e comportamento.

```
class Calculadora
{
    public int Soma(int a, int b)
    {
        return a + b;
    }
}

Calculadora calc = new Calculadora();
int resultado = calc.Soma(10, 5);
Console.WriteLine("Resultado da soma: " + resultado);
```

Explicação:

- `class Calculadora { }`: Define uma classe chamada `Calculadora`.
- `public int Soma(int a, int b)`: Método público que realiza uma soma.
- `Calculadora calc = new Calculadora();`: Cria um objeto da classe `Calculadora`.

6. Resumo e Conclusão da Aula

Nesta aula, exploramos os fundamentos da programação em C#, incluindo variáveis, estruturas de controle, repetição, funções e classes. Esses conceitos são essenciais para o desenvolvimento de qualquer aplicação em C#. Certifique-se de praticar esses conceitos para solidificar sua compreensão.

Tarefa Final:

Desenvolva um programa que permita ao usuário adicionar novos produtos com nome, preço e quantidade. O programa deve permitir listar todos os produtos cadastrados e encerrar o programa.