Aulas 7,8

- Programação Orientada à Objectos (POO)
- Exercícios Usando POO

O Professor : Lucas Pazito

Introdução à POO em C#

Nas Classes anteriores, os programas que criámos eram uma lista de comandos seguidos de cima para baixo. Esses comandos são chamados de procedimentos. Um procedimento pode ser uma única ação ou uma função que contém várias ações. Este tipo de programação é chamada de Programação Procedural. A programação procedural existe desde a década de 1970 e continua sendo amplamente usada.

A Programação Orientada à objetos, também conhecida como POO, é uma maneira muito mais moderna de programar. Em vez de se concentrar em procedimentos que fazem coisas, a POO se foca em objetos para os quais podemos fazer coisas. A ênfase da POO é sobre objetos que podemos manipular, e não a lógica necessária para manipulá-los.

Introdução à POO em C#

A Orientação à Objectos é uma ponte entre o mundo real e virtual, a partir desta é possível transcrever a forma como enxergamos os elementos do mundo real em código fonte, a fim de nos possibilitar a construção de Sistemas Complexos baseados em objectos. È um paradigma de programação baseado no conceito de OBJECTO, que podem conter dados na forma de campos (Atributos) e códigos (Métodos)

Introdução à POO em C# - Vantagens

- POO permite que você represente melhor o mundo real em seus programas. Praticamente tudo no mundo pode ser descrito como um conjunto de propriedades e ações, que é exatamente como os objetos são organizados. Ao representar melhor o mundo, os programas POO podem resolver melhor os problemas reais.
- Os programas POO são mais fáceis de ler e entender. Como você não precisa escrever as propriedades e ações para cada personagem ou sprite, seus programas serão muito mais curtos. Isso os torna mais fáceis de ler e entender.
- Pode ser mais rápido programar com POO. POO facilita a reutilização de objetos em outros programas. Em vez de criar código a partir do zero, você pode usar um objeto ou método existente e alterá-lo para se adequar ao seu programa.

Introdução à POO em C# - Vantagens

- É mais fácil criar grandes programas. Como a POO te ajuda a eliminar o código desnecessário, é mais fácil criar programas maiores e mais complexos.
- Os programas POO são mais fáceis de modificar e manter. Com a POO é fácil fazer alterações nos objetos existentes. Você também pode usar objetos existentes para criar novos objetos com pequenas modificações.
- Como os objetos são autônomos, os programas em POO são mais fáceis de depurar.

Os Pilares Básicos da POO

1- Abstracção: Observar comportamentos e estruturas do dia á dia e transformá-los em uma linguagem computacional.

Como o próprio nome diz, consiste em abstrair algo do mundo real e transformá-lo em um objecto na programação. Abstraindo, este objecto ele passará a ter uma identidade, propriedades e métodos.

2-Encapsulamento: Visa a protecção de váriaveis importantes dentro de uma classe que não podem ser manipuladas directamente. Por meio deste pilar da POO, eu posso construir coisas dentro de um objecto que outros não tenham acesso.

Os Pilares Básicos da POO

Exemplo de Encapsulamento:

Quando clicamos no Botão desligar da TV, nós não sabemos o que está acontecendo internamente na TV... apenas sabemos que ao clicar ali, a TV vai desligar. Então, podemos dizer que os métodos que fazem uma TV desligar estão encapsulados.

3-Herança: Um Objecto pode herdar algumas características de outro objecto, facilitando a reutização do código fonte. construir coisas dentro de um objecto que outros não tenham acesso.

Os Pilares Básicos da POO

Exemplo de Herança:

O Objecto **Cachorro** herda as Características do Objecto **Mamífero**. Este Exemplo mostra que o cachorro tem as suas Características Próprias, mas ele é um mamífero e compartilha as mesmas caractrerísticas que um Gato por exemplo, porém cada qual com suas próprias peculiaridades

4-Polimorfismo: **Como o próprio nome diz, consiste em acções para um mesmo método de um objecto, assumir várias formas.**

Os Pilares Básicos da POO

O principal conceito é a propriedade de duas ou mais classes derivadas de uma mesma superclasse responderem a mesma mensagem, cada uma de uma forma diferente.

Ocorre Polimorfismo quando uma subclasse redefine um método existente na superclasse, ou seja, quando temos os métodos sobrescritos (overriding).

Exemplo: Tanto o **Homem** e um **Gato** Herdam muitas caracteristicas da Classe **Mamífero**... Mas eles andam de forma diferente... Neste caso teríamos de dizer que o método **andar** destes mamíferos vai assumir várias formas (Polimorfismo)

Introdução à POO em C# - Classes

Na POO, as Classes são os elementos que provêm a estrutura de um Objecto. Ou seja, a Classe é o desenho ou estrutura em papel de um objecto. No mundo real é como se fosse a planta para a construção de uma casa.

Em C# quando nós estamos programando um objecto com suas características e funcionalidades, estamos na realidade programando uma classe.

Introdução à POO em C# - Classes

Para criarmos uma classe em C# usamos a seguinte sintaxe:

```
[acesso] Class nome_da_classe
  {
      Corpo da classe;
  }
```

Introdução à POO em C# - Classes

O acesso de uma classe pode ser de 2 tipos:

1- Internal: quando uma classe é internal significa que ela só pode ser vista dentro do assembly onde foi criada. Um assembly é um repertório de classes.

2- Public: uma classe quando é public pode ser vista apartir de qualquer assembly.

Introdução à POO em C# - Objectos

Um Objecto é a instância de uma classe, ou seja, é transformar a classe em algo real para poder usar.

Para criar um objecto usamos a seguinte sintaxe:

```
nome_da_classe nome_objecto = new nome_da_classe( );
```

Introdução à POO em C# - Atributos ou Propriedades

Os atributos são as características que definem os objectos.

Ex: A cor, altura, nome, raça, idade, peso, sexo, etc.

Para declarar um atributo de um objecto, declaramos da mesma forma que declaramos variáveis. A sintaxe é:

[acesso] tipo nome_do_atributo;

Os atributos podem ter 5 tipos de acessos:

1- Private: o atributo pode ser acedido a partir da classe onde foi criada.

2- Public: o atributo pode ser acedido a partir de qualquer classe, inclusive fora do assembly onde foi criada.

Introdução à POO em C# - Atributos ou Propriedades

3- Protected: O atributo pode ser acedido a partir da classe onde foi criada, e a partir de classes derivadas (inclusive fora do assemby onde foi criada).

4- Internal: O atributo pode ser acedido a partir de qualquer classe (excepto fora do assembly onde foi criada).

5- Protected Internal: O atributo só pode ser acedido a partir da classe onde foi declarada e nas classes derivadas do mesmo assembly.

Introdução à POO em C# - Métodos

Os métodos são acções que os objectos podem realizar. Os métodos também são chamados de funções membros da classe.

Para criar um método usamos a seguinte sintaxe:

```
[acesso] tipo_de_retorno nome_do_método( [parametros] )
{
    return valor;
}
```

Introdução à POO em C# - Métodos Construtores

Sempre que uma classe é criada um método especial é criado: È o Método Construtor. Todas as classes C# possuem um método construtor. Se você não declarar um construtor, o compilador C# criará automaticamente um construtor padrão para você.

Mas, para que serve o método construtor?

Introdução à POO em C# - Métodos Construtores

Método Construtores: São os método que permitem inicializar um objecto no momento da sua criação.

Características dos Métodos Construtores:

1- Possuem o mesmo nome da classe.

2- Toda classe gera um método construtor automático.

3- Pode receber parâmetros, mas não retorna valores.

Introdução à POO em C# - Exercícios

1- Crie um programa que usa POO, e que que permita calcular a àrea de uma circunferência. Use uma classe que permita inicializar já (ou não) com o valor do raio ao invés de pedir ao usuário.

2- Crie uma Classe Verificações, que contenha vários métodos para verficar, se um número é: negativo, positivo, par ou ímpar, perfeito, se dois números são amigos, se um número é primo, se um número é maior do que outro... Implemente outras Funcionalidades que achar apropriadas, dentro do âmbito de Verificação

Introdução à POO em C# - Classes Abstractas

Uma classe abstrata é uma classe que não pode ser instanciada.

Você não pode criar um objeto a partir de uma classe abstrata

Uma Classe Abstrata pode ser herdada, e geralmente serve como classe Base para outras classes.

Elas podem conter métodos abstratos e métodos comuns. Podem também possuir métodos construtores, propriedades, eventos.

Introdução à POO em C# - Classes Abstractas

- Uma classe abstrata não pode ser estática (static)
- Uma classe abstrata não pode ser selada (sealed)
- Uma classe abstrata pode herdar de outra classe abstrata.
- Serve para atribuir funcionalidades iguais á Objectos diferentes

Introdução à POO em C# - Classes Abstractas

Para criar uma Classe Abstracta usamos a seguinte Sintaxe:

```
abstract Class Nome da Classe
{
                    Corpo da Classe;
}
```

Introdução à POO em C# - Herança

Como a classe-filha consegue herdar todos os atributos e métodos da classe-mãe, a herança apresenta a vantagem de evitar a duplicidade de código. Por outro lado, há também a possibilidade de definir novos atributos e adicionar outras funcionalidades.

Sintaxe para implementar Herança em C#:

```
Class Classe-filha : Classe-mãe

{

  //Aqui deverão constar as diversas instruções que definem a classe-filha

}
```

Introdução à POO em C# - Classes Seladas

São classes que não podem ser herdadas.

Para criar uma classe selada usamos a seguinte sintaxe:

```
 [acesso] Sealed Class nomedaClasse
{
   Corpo da Classe;
}
```

Introdução à POO em C# - Polimorfismo

Usando polimorfismo podemos :

- Invocar métodos da classe derivada através da classe base em tempo de execução;
- Permitir que classes forneçam diferentes implementações de métodos que são chamados com o mesmo nome;

Existem dois tipos básicos de polimorfismo:

1-Polimorfismo em tempo de compilação (Overloading/Sobrecarga);

2-Polimorfismo em tempo de execução (Overriding/Sobrescrita);

Introdução à POO em C# - Polimorfismo

Novamente é bom recordar: Polimorfismo significa muitas formas , na orientação a objetos você pode enviar uma mesma mensagem para diferentes objetos e fazê-los responder da maneira correta. Você pode enviar a mensagem mover para cada objeto semelhante a um veiculo e cada um vai se comportar de maneira diferente para atender a sua solicitação.

Quando uma mesma mensagem pode ser processada de diferentes formas temos um exemplo de polimorfismo.

Uma definição mais formal diria:

"Polimorfismo é o princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma identificação (assinatura) mas comportamentos distintos, especializados para cada classe derivada, usando para tanto uma referência a um objeto do tipo da superclasse"