Nome: Willian Ferreira Teixeira

PROGRAMAÇÃO ORIENTADA A OBJETO

A Programação Orientada a Objetos (OOP) é um paradigma de programação que revolucionou a forma como desenvolvemos software, oferecendo uma abordagem mais modular e eficiente para resolver problemas complexos. Ao organizar o código em objetos que interagem entre si, a OOP permite uma modelagem mais precisa do mundo real, facilitando a compreensão.

Conceitos de OPP:

Encapsulamento: Este conceito permite ocultar os detalhes internos de um objeto e expor apenas uma interface pública. Isso promove a segurança e a modularidade do código, permitindo que as implementações internas sejam alteradas sem afetar o restante do sistema.

Herança: A herança permite que uma classe herde atributos e métodos de outra classe. Isso promove a reutilização de código e facilita a criação de hierarquias de classes, onde classes mais específicas podem estender ou modificar o comportamento de classes mais genéricas.

Polimorfismo: Esse conceito permite que objetos de diferentes classes sejam tratados de maneira uniforme, proporcionando flexibilidade e extensibilidade ao código. O polimorfismo permite que métodos em classes base sejam substituídos por implementações específicas em classes derivadas, permitindo que o mesmo método se comporte de maneira diferente com base no tipo do objeto.

Abstração: é sobre o que pode ser combinado com um objeto, exemplo, uma pessoa não faz parte de um objeto carro, mas ela poderia virar o motorista ou passageiro vindo a se sentar, dois objetos separados são agregados para se tornar um componente.

Abstração trata de capturar a ideia central de um objeto e ignorar os detalhes ou especificidades, o uso da palavra-chave "abstract" formaliza esse conceito. Se uma classe não é explicitamente declarada como abstrata, então pode ser descrita como concreta. Normalmente, as classes base ou superclasses são abstratas

CRIANDO A CLASS LIBRARY

1. Use sua ferramenta de codificação preferida para criar um novo projeto, conforme definido na lista a seguir:

   • Modelo do projeto: Biblioteca de Classes (Class Library/classlib)
   • Arquivo e pasta do projeto: PacktLibraryNetStandard2
   • Arquivo e pasta da solução/workspace: Chapter05

2. Abra o arquivo PacktLibraryNetStandard2.csproj e observe que, por padrão, as bibliotecas de classes criadas pelo SDK .NET 7 têm como alvo o .NET 7 e, portanto, só podem ser referenciadas por outros assemblies compatíveis com o .NET 7, conforme mostrado no seguinte trecho de código

```
<Project Sdk=”Microsoft.NET.Sdk”>
        <PropertyGroup>
                <TargetFramework>net7.0</TargetFramework>
                <Nullable>>enable<Nullable>
                <ImplicitUsings>enable</ImplicitUsing>
        </PropertyGroup>
```

3. Modifique o framework para ter como alvo o .NET Standard 2.0, adicione uma entrada para usar explicitamente o compilador C# 11 e importe estaticamente a classe System.Console para todos os arquivos C#, conforme destacado no seguinte trecho de código:

```
<Project Sdk="Microsoft.NET.Sdk">
        <PropertyGroup>
                <TargetFramework>netstandard2.0
                <LangVersion>11
                <Nullable>enable
                <ImplicitUsings>enable>
        </PropertyGroup>
                <ItemGroup>
                <Using Include="System.Console" Static="true" />
                </ItemGroup>
        </Project>
```

Definindo uma classe com namespace

Para usar uma classe definida em um namespace em outro arquivo C# ou em outro projeto, você precisará adicionar uma referência usando a diretiva using no topo do código.

Compreendendo os membros

Os membros de uma classe podem ser campos, métodos ou versões especializadas de ambos. Aqui está uma descrição de cada tipo de membro:

Campo: São usados para armazenar dados e podem ser de três categorias especializadas:

- Constante: Os dados nunca mudam e são copiados literalmente pelo compilador sempre que são lidos.
- Somente leitura: Os dados não podem ser alterados após a instância da classe, mas podem ser calculados ou carregados de uma fonte externa no momento da instância.
- Evento: Os dados referenciam um ou mais métodos que devem ser executados quando algo específico acontece, como clicar em um botão.

Métodos: São utilizados para executar instruções e também podem ser de diferentes categorias:

- Construtor: As instruções são executadas quando a palavra-chave "new" é utilizada para alocar memória e instanciar uma classe.
- Propriedade: As instruções são executadas quando dados são obtidos ou definidos. Os dados geralmente são armazenados em um campo, mas podem ser armazenados externamente ou calculados durante a execução. Propriedades são preferíveis para encapsular campos, a menos que seja necessário expor o endereço de memória do campo.
- Indexador: As instruções são executadas quando dados são obtidos ou definidos usando a sintaxe de "array" [].
- Operador: As instruções são executadas quando operadores, como + e /, são utilizados em operandos do tipo da classe.

Dividindo classes usando parciais

Parciais são utilizados para dividir uma classe em várias partes físicas, armazenadas em arquivos diferentes. Isso é útil para separar a implementação de uma classe em partes distintas, facilitando a manutenção do código. A palavra-chave "partial" é usada para indicar que uma classe é parcial e pode ser estendida em diferentes arquivos.

Por exemplo, ao adicionar declarações à classe Person que são automaticamente geradas por uma ferramenta como um mapeador objeto-relacional, podemos dividir a classe em um arquivo de código gerado automaticamente e outro arquivo de código editado manualmente. Ao definir a classe como parcial, isso permite essa separação e facilita a organização do código.

Armazenando um valor usando um tipo enum

Às vezes, um valor precisa ser uma de um conjunto limitado de opções. Por exemplo, existem seis marcas de carro, e uma pessoa pode ter uma favorita. Em outros momentos, um valor precisa ser uma combinação de um conjunto limitado de opções. Por exemplo, uma pessoa pode ter uma lista de desejos de carros. Podemos armazenar esses dados definindo um tipo enum. Um tipo enum é uma maneira muito eficiente de armazenar uma ou mais opções porque, internamente, ele usa valores inteiros em combinação com uma tabela de pesquisa de descrições de string.

Controlando o acesso com propriedades e indexadores

Uma propriedade em C# é essencialmente um método (ou par de métodos) que age e se parece como um campo quando você quer obter ou definir um valor, simplificando assim a sintaxe. Os indexadores permitem acessar elementos de uma coleção usando uma sintaxe semelhante à de um array.

Para criar uma propriedade com capacidade de atribuição, você deve fornecer um par de métodos - uma parte get e uma parte set:

- Métodos Estáticos: Métodos estáticos são métodos que pertencem à classe em vez de instâncias individuais da classe.
- Métodos de Extensão: Métodos de extensão permitem adicionar novos métodos a tipos existentes sem modificar a definição do tipo original.
- Métodos Assíncronos: Métodos assíncronos permitem que você escreva código que pode ser executado de forma assíncrona, o que é útil para operações de E/S intensivas ou operações de longa duração que não bloqueiam a thread principal.

Correspondência de padrões com objetos

Correspondência de padrões com objetos é uma técnica em C# que permite realizar operações com base nas características de um objeto. Isso é útil em situações onde você precisa executar diferentes ações dependendo do estado ou dos valores dos atributos de um objeto.

Trabalhando com registros

introduzidos no C# 9, são úteis para modelar dados imutáveis de forma concisa. Eles geram automaticamente construtores, propriedades e suporte para comparação estrutural. Além disso, registros suportam padrões de desconstrução e clonagem imutável, sendo eficazes em várias situações.

Funcionalidades usando Funções locais

Funções locais são uma característica introduzida no C# 7 que permitem definir funções dentro do escopo de outra função. Isso é útil quando você precisa de uma função auxiliar que só será usada dentro de outra função e não precisa ser acessada externamente. Para implementar funcionalidades usando funções locais em C#, você pode declarar uma função dentro do corpo de outra função

Noção básica sobre registros

Registros em C# são uma novidade introduzida no C# 9, oferecendo uma forma concisa de definir tipos de dados imutáveis. Eles são úteis para modelar dados que não mudam após a criação e geram automaticamente construtores, propriedades e suporte para comparação de igualdade estrutural. Registros suportam recursos como padrões de desconstrução e clonagem imutável, proporcionando uma maneira eficiente de trabalhar com dados imutáveis em C#.