Microsoft .NET

Residência em Software

POO - Interfaces

Prof. Hélder Almeida

INSTITUIÇÃO EXECUTORA
cepedi UESC

COORDENADORA
MCTI FUTURO Softex

APOIO
MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO
GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

Residência em Software

Classes Abstratas e Interfaces

Em C#, interfaces e classes abstratas são ambos mecanismos utilizados para estabelecer contratos e promover a abstração em orientação a objetos. Aqui estão algumas semelhanças entre interfaces e classes abstratas:

- Abstração:
  - Ambas são conceitos fundamentais para abstração, que é a capacidade de representar as características essenciais de um objeto sem se preocupar com os detalhes específicos de implementação.
- Contratos:
  - Ambas são usadas para definir contratos que as classes derivadas devem seguir.
  - Em uma interface, o contrato é totalmente abstrato, enquanto em uma classe abstrata, pode haver métodos abstratos e métodos concretos.

Residência em Software

Classes Abstratas e Interfaces

Em C#, interfaces e classes abstratas são ambos mecanismos utilizados para estabelecer contratos e promover a abstração em orientação a objetos. Aqui estão algumas semelhanças entre interfaces e classes abstratas:

- Herança:
  - Ambas suportam herança, permitindo que as classes derivadas (ou implementadoras) herdem ou sigam o contrato estabelecido pela interface ou classe abstrata.
- Polimorfismo:
  - Ambas possibilitam o polimorfismo, permitindo que objetos sejam tratados como instâncias da interface ou classe abstrata, proporcionando flexibilidade e reutilização de código.

Residência em Software

Interfaces, o que são?

Uma interface, em C#, é um contrato que define um conjunto de métodos, propriedades e eventos.

As interfaces são usadas para definir um contrato comum que várias classes devem implementar.

Por convenção iniciamos seu nome com ‘I’

```
public interface IAnimal{
    void EmitirSom();
    void Mover();
}
```

```
public interface IPrintable{
        void Print();
}
```

```
public interface ITributavel{
    public double Tributo { get; set; }
    double CalcularTributo();
}
```

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/interface

Implementação de Interface

Uma classe pode implementar uma ou várias interfaces.

A implementação é feita usando o operador ':' seguido pelo(s) nome(s) da interface(s) separados por vírgula(s).

```
public class Gato : IAnimal{
    public void EmitirSom(){
        Console.WriteLine("Miau!");
    }
    public void Mover(){
        Console.WriteLine("O gato
se move graciosamente.");
    }
}
```

```
public abstract class Conta : IPrintable{
    public void Print(){
        Console.WriteLine($"{ToString()}");
    }
}
```

```
public class ContaPoupanca : Conta, ITributavel{
    public double Tributo { get; set; } = 0.05;
    public double CalcularTributo(){
        return Saldo * Tributo;
    }
}
```

Residência em Software

Herança vs Implementação de Interfaces

Uma classe pode herdar de apenas uma classe base, mas pode implementar várias interfaces.

As interfaces permitem que as classes tenham comportamentos específicos sem herdar toda a implementação.

Ambos os conceitos são poderosos e são escolhidos com base nas necessidades específicas de design e estruturação do código. Interfaces são frequentemente usadas para contratos mais flexíveis e múltipla herança, enquanto classes abstratas podem fornecer uma base comum para implementações mais ricas.

Residência em Software

Praticando

Vamos criar um sistema de pagamentos que lida com diferentes métodos de pagamento, como cartão de crédito, transferência bancária e pagamento em dinheiro. Utilize conceitos de interfaces e herança para modelar esse sistema.

Interface de Pagamento:
Crie uma interface chamada IPagamento com um método RealizarPagamento que recebe o valor a ser pago.

Métodos de Pagamento:
Implemente classes para diferentes métodos de pagamento, como CartaoCredito, TransferenciaBancaria e PagamentoDinheiro.
Cada classe deve implementar a interface IPagamento e fornecer uma implementação específica para o método RealizarPagamento. Por exemplo, o método RealizarPagamento em CartaoCredito pode exibir uma mensagem indicando que o pagamento foi feito com sucesso.

Cliente:
Crie uma classe Cliente com propriedades como Nome e Email.
A classe Cliente deve ter um método que aceite um objeto que implementa IPagamento e use esse objeto para realizar um pagamento fictício.

Programa Principal:
No programa principal, crie instâncias de diferentes métodos de pagamento (cartão de crédito, transferência bancária, etc.).
Crie um cliente e solicite que ele faça um pagamento utilizando cada método de pagamento criado.