

6 – Interfaces em POO com C#

Interfaces são como as classes abstratas, mas nelas não podemos implementar nenhum método, apenas declarar suas assinaturas, como nos métodos abstratos da abstração.

Uma classe ao herdar uma interface deverá escrever todos os seus, ao contrário da classe abstrata que só iremos escrever os métodos abstratos.

Com o tempo você aprenderá a ver qual é mais útil dependendo da situação é mais vantajoso usar uma classe abstrata ou uma interface. Não existe uma regra para definir isso, mas sim a necessidade de uma ou de outra. Por boas práticas, todas as interfaces começam com I antes de seu nome.

Veja os exemplos:

```
interface IVoador
{
    float AltitudeMaxima { get; set; }
    void Voa();
}

interface INadador
{
    void Nada();
}
```

Em vez de `class`, usa-se `interface`. Uma interface pode ter métodos e propriedades, todos sem implementação, necessariamente. Essa última característica proíbe a instanciación de interfaces, como nas classes abstratas. Por exemplo:

```
IVoador passaro = new IVoador(); //Não compila. Não se pode instanciar interfaces
```

Você pode se perguntar: "Qual a utilidade de se definir um tipo de classe que não faz nada?". O conceito mostra-se poderoso no uso de polimorfismo. Lembra as classes abstratas? Pense numa classe abstrata com todos os métodos abstratos. Eis o que a interface é. Você pode fazer classes herdarem de uma ou mais interfaces.

```
class Pato : INadador, IVoador
{
    public float AltitudeMaxima { get; set; }
    public void Voa()
    {
        Console.WriteLine("Pato voando");
    }
    public void Nada()
    {
        Console.WriteLine("Pato nadando");
    }
}
```

Você pode ainda argumentar: "Não bastava simplesmente criar as classes totalmente abstratas `IVoador` e `INadador`?". A resposta é: uma classe não pode herdar de mais de uma classe, mas pode herdar de mais de uma interface.

Agora vamos pensar no sentido. Qual o significado da palavra interface? É um ambiente de contato entre duas coisas distintas, no qual ambas conseguem se comunicar. Em C#, uma interface garante que uma classe tenha a implementação de certos métodos. Assim outras classes podem ter certeza de que o objeto que implementa interface de interesse tem os métodos adequados para fazer a comunicação.

Por exemplo, pense num professor:

Além de dar aula, também pode saber cozinhar, dirigir, fazer prova, etc. Ele pode possuir várias interfaces.

Como cada pessoa faz cada uma dessas coisas de um jeito, não precisamos chamá-las de classes e implementar cada um dos seus métodos, mas apenas saber que estes existem. Por isso interfaces não levam implementação.

Professor é uma pessoa. Logo, isso se encaixa bem num caso de herança. Geralmente podemos dizer uma única vez que "Uma coisa é outra coisa". Para evitar conflitos de herança, qualquer classe só pode ter uma classe base.

Um aluno também é uma pessoa, também dirige, também cozinha e também faz prova. Em alguns casos, estas ações são idênticas, ou semelhantes. Em outras, são completamente diferentes. É o caso de fazer prova. Enquanto o professor prepara a prova para o aluno fazê-la, o aluno resolve a prova feita pelo professor.

Da análise do professor e aluno, teríamos as seguintes classes e interfaces:

```
public class Pessoa
{
    public String nome {get; set;}
}

interface IDirige
{
    void Dirigir();
}

interface ICozinha
{
    void Cozinhar();
}

interface IFazProva
{
    void FazerProva();
}
```

```

    }

    public class Professor : Pessoa, IDirige, ICozinha, IFazProva
    {
        public void Dirigir()
        {
            Console.WriteLine("Professor dirigindo");
        }

        public void Cozinhar()
        {
            Console.WriteLine("Professor cozinhando");
        }

        public void FazerProva()
        {
            Console.WriteLine("Professor preparando prova");
        }
    }

    public class Aluno : Pessoa, IDirige, ICozinha, IFazProva
    {
        public void Dirigir()
        {
            Console.WriteLine("Aluno dirigindo");
        }

        public void Cozinhar()
        {
            Console.WriteLine("Aluno cozinhando");
        }

        public void FazerProva()
        {
            Console.WriteLine("Aluno resolvendo a prova");
        }
    }

    static void Main()
    {
        Professor sergio = new Professor();
        Aluno joao = new Aluno();
        sergio.FazerProva();
        joao.FazerProva();
        Console.ReadKey();
    }

```

Observações:

O que acontece quando métodos com o mesmo nome aparecem em duas interfaces que são herdadas por outra classe?

Existem 3 casos distintos:

1 – Caso os métodos tenham a mesma assinatura (nome e/ou parâmetros) mas retornam tipos diferentes, a classe não poderá comportar as duas interfaces. É impossível implementar.

2 – Caso os métodos tenham a mesma assinatura e sejam idênticos, implemente apenas uma vez o método.

3 – Caso tenham a mesma assinatura, mas recebam parâmetros diferentes, deverá ser feita uma sobrecarga de métodos (~~usando virtual em ambas as definições~~).

No Visual Studio 2022 não foi necessário colocar virtual em ambos os métodos. Apenas faça a sobrecarga dos métodos na classe herdada.

Exercícios:

1 – Crie uma interface chamada Conta. Conta terá os métodos depositar(double valor), sacar (double valor), ambos void por não retornarem nenhum dado e receberem valor do tipo double para realizar suas operações, e um método double getSaldo(), que irá retornar o valor do saldo da conta.

2 – Crie uma classe chamada ContaCorrente, que herda a interface Conta. Esta classe possui dois atributos/propriedades chamados saldo e tarifa, do tipo double. Realize a implementação dos métodos desta interface. Faça com que, a cada saque, seja descontado além do valor, o valor colocado no atributo chamado tarifa. Por exemplo. Se o saldo for 100, a tarifa for 1.50 e o valor solicitado para saque for 50, o novo saldo será $100 - 50 - 1.50$ (saldo – valor – tarifa) = 48,50. Não permita sacar se o saldo for insuficiente (não pode deixar a conta com saldo menor do que zero).

3 – Crie uma classe chamada ContaPoupanca, que herda a interface conta. Esta classe possui um atributos/propriedade chamado saldo, como a classe anterior (do tipo double). Realize a implementação dos métodos desta interface. No saque, faça o mesmo procedimento como colocado no exercício 2, porém em poupança não teremos o desconto da tarifa. Por exemplo, se o saldo é 100 e o valor do saque for 50, o novo saldo será $100 - 50$ (saldo – valor) = 50. Não permita sacar se o saldo for insuficiente (não pode deixar a conta com saldo menor do que zero).

4 – Crie uma outra classe chamada GeradordeExtratos, que possui um método que recebe a interface Conta como parâmetro, chamado GerarExtrato (Conta c), que mostre o saldo de qualquer uma das duas contas. (para isso, deve chamar c.getSaldo()).

5 – Escreva o Main() de seu programa de forma a testar os métodos criados.