E-Book

.NET Developer

Programação orientada a objetos

mensagem de recepção

Boas-vindas à nossa série de ebooks! Preparamos estes materiais pensando em como transmitir mais conhecimento diversificado para você explorar diferentes meios de desenvolver suas habilidades.

Aproveite este material e bons estudos!

SUMÁRIO

Introdução

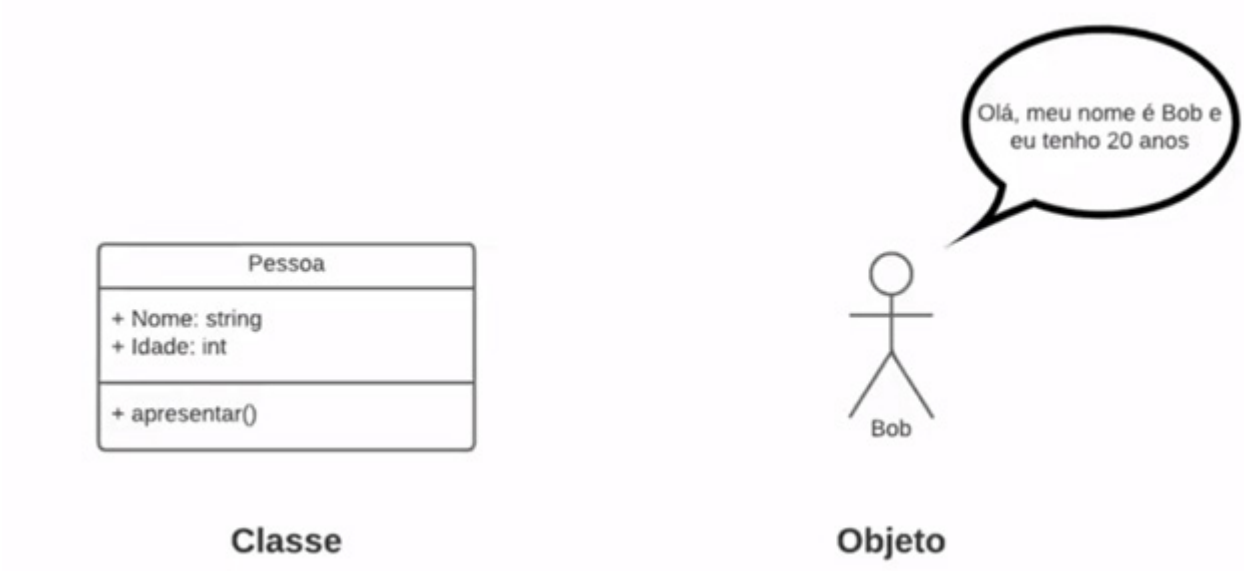
A POO é um paradigma de programação, ou seja, corresponde a uma técnica de programação para um fim específico.

Dentro desta técnica, existem quatro pilares

- Abstração
- Encapsulamento
- Herança
- Polimorfismo

O principal conceito da POO são classes e objetos

- Classe
  - O “Molde”
  - Representação de algo do mundo real
- Objeto
  - A Classe concretizada
  - Tem uma classe como tipo
  - Segue estritamente a Classe que representa



Tipos de paradigmas

Um paradigma nada mais é do que um modelo de técnicas, estruturas e formas de solucionar um problema.Paradigma de programação é diferente de linguagem de programação.Uma linguagem de programação implementa um ou mais paradigmas.

Alguns dos paradigmas de programação :

- Programação orientada a objetos
- Programação estruturada
- Programação imperativa
- Programação procedural
- Programação orientada a eventos
- Programação lógica

Abstração

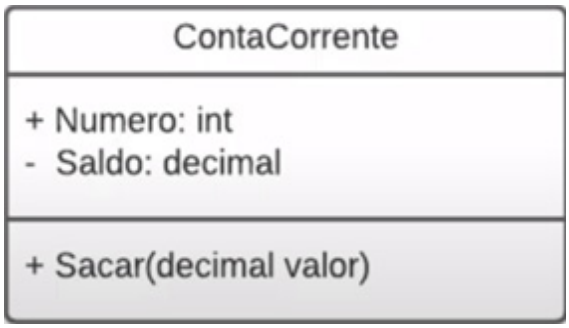
Abstrair um objeto do mundo real para um contexto específico, considerando apenas os atributos importantes.

Abstraindo uma pessoa do mundo real para um sistema onde será utilizado o nome e a idade :

```
8   public class Pessoa
9   {
10      public string Nome { get; set; }
11      public int Idade { get; set; }
12
13      public void Apresentar()
14      {
15          Console.WriteLine($"Olá, meu nome é {Nome} e tenho {Idade} anos!");
16      }
17  }
```

Encapsulamento

O Encapsulamento serve para proteger uma classe e definir limites para alteração de suas propriedades. Serve para ocultar seu comportamento e expor somente o necessário.



O sinal de + significa que a propriedade é pública. Já o sinal de- quer dizer que é privada.

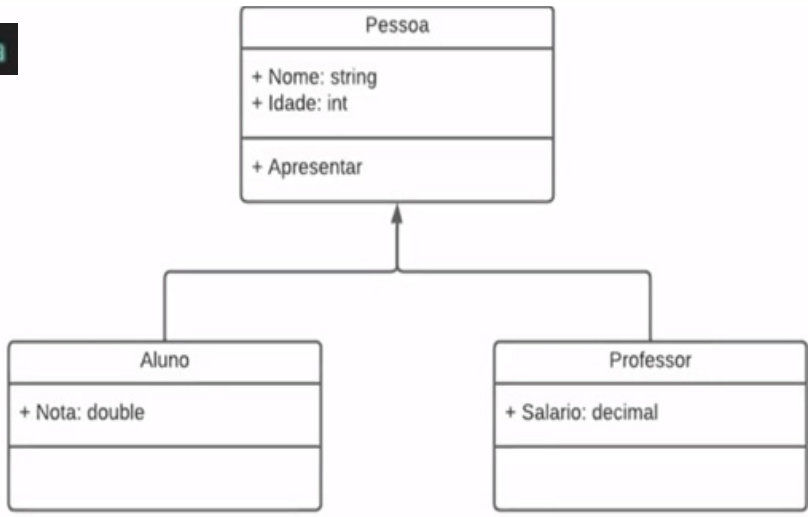
Uma propriedade privada só pode ser alterada pela própria Classe.

Herança

A herança nos permite reutilizar atributos, métodos e comportamentos de uma classe em outras classes. Serve para agrupar objetos que são do mesmo tipo, porém com características diferentes.

```
public class Aluno : Pessoa
```

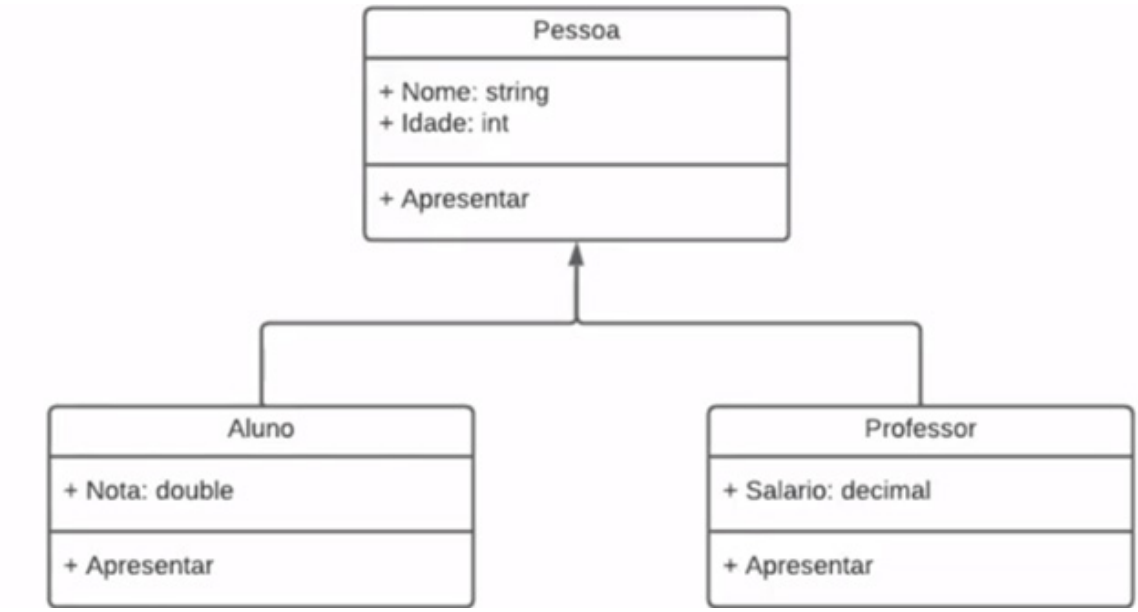
- Aluno
  - · Classe filha
- :
  - ·sinaliza herança no C#
- Pessoa
  - ·Classe pai



O conceito de herança vai ajudar a não repetir linhas de código. No C#, uma Classe filha só pode ter uma classe pai. Não é recomendado utilizar herança em cascata, ou seja, uma Classe filha herdar de outra Classe filha, pois isso aumentará a complexidade do código.

Polimorfismo

O polimorfismo vem do grego e significa “muitas formas”. Com o polimorfismo, podemos sobrescrever métodos das classes filhas para que se comportem de maneira diferente e ter sua própria implementação



Para adaptar o método apresentar que foi herdado da classe pai e modificá-lo para dois cenários diferentes, será utilizado o conceito de polimorfismo.

Apresentar

Olá, meu nome é X. Sou um aluno de nota Y.

Apresentar

Olá, meu nome é X. Sou um professor, ganho um salário de Y.

No C#, para sinalizar que um determinado método pode ser sobrescrito é utilizada a palavra reservada `virtual` na Classe pai.

```
public virtual void Apresentar()
```

Já na Classe filha utilizamos a palavra reservada `override`.

```
public override void Apresentar()
```

Classe abstrata

Uma classe abstrata tem como objetivo ser exclusivamente um modelo para ser herdado, portanto não pode ser instanciada. Você pode implementar métodos ou deixá-los a cargo de quem herdar.

class abstract

- `public abstract class Conta`
  - Classe abstrata

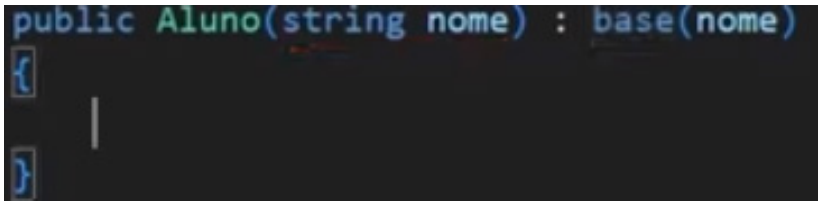
- `protected decimal saldo;`
  - Modificador de acesso protected
  - Protege a propriedade de modificações externas, com exceção das classes filhas

- `public abstract void Creditar(decimal valor);`
  - Método abstrato
  - Não tem um procedimento padrão
  - Todo método abstrato precisa ser implementado nas Classes filhas

Construtor por herança

Definir um construtor padrão na Classe pai para ser utilizado pelas classes filhas. A partir do momento que a Classe pai exige uma propriedade no construtor, deve-se implementar também um construtor nas classes filhas e referenciar nele a Classe pai.

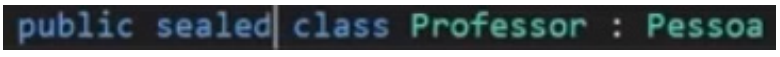
- ```
  public Pessoa(string nome)
  {
      Nome = nome;
  }
  ```
  - Construtor da Classe pai
  ```

- 

  · Construtor da Classe filha
  · “base” é a palavra reservada para referenciar a propriedade da Classe pai

Classe selada

Uma classe selada tem como objetivo impedir que outras classes façam uma herança dela, ou seja, nenhuma classe pode ser sua derivada. Também existem métodos e propriedades seladas.Métodos selados não podem ser sobrescritos pelas classes filhas.

- 

  · “sealed” é a palavra reservada para determinar uma classe selada

- 

  · Método selado

Classe object

A classe System.Object é a mãe de todas as classes na hierarquia do dotNET. Todas as classes derivam, direta ou indiretamente da classe Object, e ela tem como objetivo prover serviços de baixo nível para suas classes filhas.

Interfaces

Uma interface é um contrato que pode ser implementado por uma classe. É como se fosse uma classe abstrata, podendo definir métodos abstratos para serem implementados. Assim como uma classe abstrata, uma interface não pode ser instanciada.

- Ao invés da Classe herdar uma interface, o correto é dizer implementar uma interface
- A Classe que implementar uma interface é obrigada a implementar também todos os métodos sem corpo da interface
- Os métodos padrão de uma interface, ou seja, que possuem um corpo, tornam-se opcionais serem implementados ou não
- Uma Classe pode implementar uma ou mais interfaces ao mesmo tempo
- `public interface ICalculadora`
  - Por convenção, se utiliza a letra "i" maiúscula no início do nome da interface

- `int Somar(int num1, int num2);`
  - Método de uma interface
  - Não é necessário colocar um modificador de acesso pois a interface por padrão deixa tudo como public

- `public class Calculadora : ICalculadora`
  - A maneira de implementar uma interface é a mesma de herdar

- `ICalculadora calc = new Calculadora();`
  - Uma interface não pode ser instanciada
  - Uma classe que implementa uma interface pode ser instanciada normalmente

Obrigado pela leitura!

Você não chegou aqui pulando páginas, né? 😏
Brincadeiras à parte, realmente nós da DIO esperamos que esteja curtindo sua jornada de aprendizado aqui conosco e desejamos seu sucesso sempre!
Vamos em frente!