| Curso de Orientação a Objetos em C# | Aula02 |
|---|---|
| **Professora: Luana Fernandes** | **Turma: Terça – 21:00h** |

**Conteúdo desta aula:**

- Métodos (Parte 2)
- Construtor
- Encapsulamento

---

## MÉTODOS

### Métodos sem retorno

Esse tipo de método executa apenas o código que tem dentro dele, não retornando nenhum resultado, sendo identificados com a palavra-chave **void**.

```
public void Escrever( ) {
   Console.WriteLine("Método sem Retorno");
}
```

### Métodos com retorno

Esses métodos que não possuem a palavra-chave void incorporada na declaração, mas sim o **tipo de dados** que será retornado. Eles também apresentam em seu corpo a palavra reservada **return**, que informa que o método terá que retornar o mesmo tipo de dados que foi declarado.

```
public string RetornaNomeCompleto( ) {
    string nome = "Maria";
    string sobrenome = "Rodrigues"
    return nome + " " + sobrenome;
}

public int RetornaSoma( ){
   return 5 + 4;
}
```

**Métodos com parâmetros**

Métodos também podem receber parâmetros (argumentos), ou seja, valores necessários para executar a função que se encontra no corpo do método.

```
public int RetornaSomaDeDoisNumeros( int n1,  int n2 ) {
    return n1 + n2;
}
```

**Sobrecarga de método**

Em uma mesma classe pode haver vários métodos com o mesmo nome, contanto que possuam argumentos distintos. Os argumentos podem variar em número, tipo e ordem de declaração.

```
public int Soma (int valor1, int valor2) {
    return valor1 + valor2;
}

public double Soma (double valor1, double valor2) {
    return valor1 + valor2;
}

public string Soma (string valor1, string valor2) {
    return valor1 + valor2
}
```

---

## CONSTRUTOR

O construtor é um dos métodos mais importantes de uma classe, pois é ele que constrói as suas instâncias, os objetos. Todo classe tem um construtor interno chamado de construtor default. Ele normalmente não fica visível quando criarmos a classe, mas ele está lá e nós podemos escrevê-los se quisermos.

```
public class Aluno {

   public string nome;
   public int idade;
   public string matricula;


   public Aluno ( ) { // → Construtor Default

   }

}
```

Como os construtores também são métodos, eles são capazes de receber sobrecargas. Porém é muito importante lembrar que se uma sobrecarga for feita o construtor default deixa de existir, exceto que ele for expressamente escrito na classe.

```
public class Aluno {

   public string nome;
   public int idade;
   public string matricula;


  //Como só existe a sobrecarga, não é possível criar um aluno de outra forma.
   public Aluno ( string nome, int idade, string matricula) { // → Sobrecarga

       this.nome = nome;

       this.idade = idade;

       this.matricula = matricula;

   }

}
```

**Importante:** A palavra reservada **this** representa a própria classe, ou seja “this.nome” é o atributo da classe Aluno, diferente de “nome” (sem this) que é o nome do parâmetro recebido no método.

# ENCAPSULAMENTO

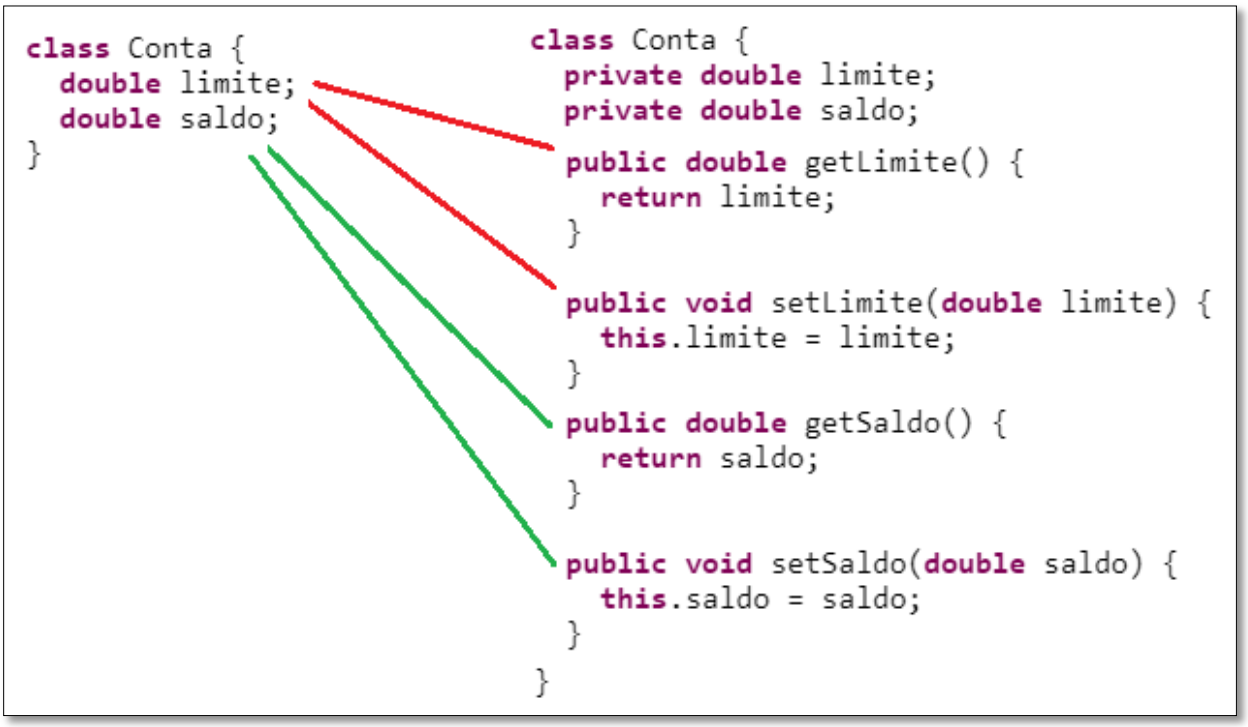
Encapsulamento é a técnica que faz com que detalhes internos do funcionamento dos métodos de uma classe permaneçam ocultos para os objetos. Com isso, o conhecimento a respeito da implementação interna da classe é desnecessário do ponto de vista do objeto, uma vez que isso passa a ser responsabilidade dos métodos internos da classe.

Tendo em mente que os métodos e os atributos de uma classe podem ser definidos como públicos ou privados, temos a seguinte situação:

- Tudo o que o usuário externo precisa conhecer a respeito de uma classe encontra-se em métodos declarados como públicos (public).
- Somente os métodos da classe são capazes de acessar seus atributos privados. Isso garante que não ocorrerão ações inadequadas, mas exige que a interface pública seja bem planejada para que o funcionamento interno da classe não seja muito exposto.

**Getters e Settters - Métodos Acessores**

Devemos sempre fornecer campos privados e métodos de acesso (getters) públicos, e caso a classe seja mutável devemos também fornecer os métodos modificadores (setters). Veja o exemplo da classe Conta não encapsulada, em seguida usando Encapsulamento, ou seja, tornando seus atributos privados e criando os setters e getters de acesso.



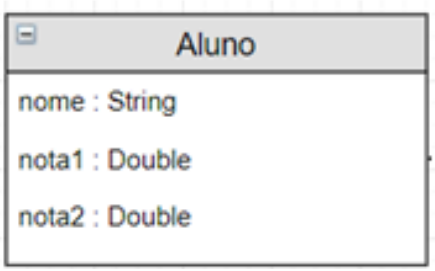
Desse modo, passaremos a usar os getters e setters dos atributos e não teremos mais acesso direto aos atributos, pois mudamos os modificadores para privados. Sendo assim, qualquer regra de negócio que venha a surgir no projeto pode ser implementada nos métodos. Por exemplo, dentro do setSaldo(double saldo) podemos fazer uma validação para verificar se o valor de saldo é maior que zero.

---

## EXERCÍCIOS

**Questão 1**

Após a explicação apresenta, crie um projeto chamado **escola** com a classe **Aluno** Siga o diagrama a seguir para incluir os atributos e encapsule-os. Não se esqueça de alterar a visibilidade dos atributos para **private**.




*OBS: Algumas bibliográficas criticam o uso genérico de getters e setters que nunca serão usados. Não adotaremos essa prática aqui, mas sugiro que leia sobre o assunto.*

**Questão 2**

Execute as seguintes validações dentro dos métodos criados:

a) Dentro de **SetNota1** e **SetNota2**, valide se o parâmetro recebido está entre 0 e 10.

b) Dentro de **SetNome**, verifique se o nome contém até 30 caracteres. Se houver mais, mostre na tela a mensagem: "**O nome deve conter até 30 caracteres**."

*OBS: Pesquise pelo comando length do Java.*

**Questão 3**

a) Crie uma classe chamada **ControleAluno** e inclua o método Main().

b) Use os getteres e setters criados para preencher os dados de um Aluno.

Veja um exemplo:

```
Aluno aluno1 = new Aluno();
aluno1.setNome("Maria");
aluno1.setNota1(8.3);
aluno1.setNota2(7.2);
```

c) Teste se as validações dos métodos estão funcionando passando como parâmetro os seguintes exemplos de valores:

- Um nome com mais de 30 caracteres;
- Uma nota1 maior que 10;
- Uma nota2 com valor negativo.