UNIDADE 4 – ATRIBUTOS E MÉTODOS DE CLASSE

Nessa unidade veremos reforçaremos as funcionalidades dos atributos e métodos de uma classe.

SUMÁRIO

1 ATRIBUTOS E MÉTODOS DE CLASSE

1.1 ATRIBUTOS ESTÁTICOS

Num sistema bancário, provavelmente, criaríamos uma classe para especificar os objetos que representariam os funcionários do banco.

```
1  private class Funcionario
2  {
3      public string nome;
4      public double salario;
5
6      0 references
7      public void AumentaSalario(double aumento)
8      {
9          this.salario += aumento;
10     }
11 }
```

Tabela 1: Funcionario.cs

Suponha que o banco paga aos seus funcionários um valor padrão de vale refeição por dia trabalhado. O sistema do banco precisa guardar esse valor. Poderíamos definir um atributo na classe Funcionário para tal propósito.

```
1  private class Funcionario
2  {
3      public string nome;
4      public double salario;
5      public double valeRefeicaoDiario;
6
7      0 references
8      public void AumentaSalario(double aumento)
9      {
10         this.salario += aumento;
11     }
12 }
```

Tabela 2: Funcionario.cs

O atributo valeRefeicaoDiario é de instância, ou seja, cada objeto criado a partir da classe Funcionário teria o seu próprio atributo valeRefeicaoDiario. Porém, não faz sentido ter esse valor repetido em todos os objetos, já que ele é único para todos os funcionários.

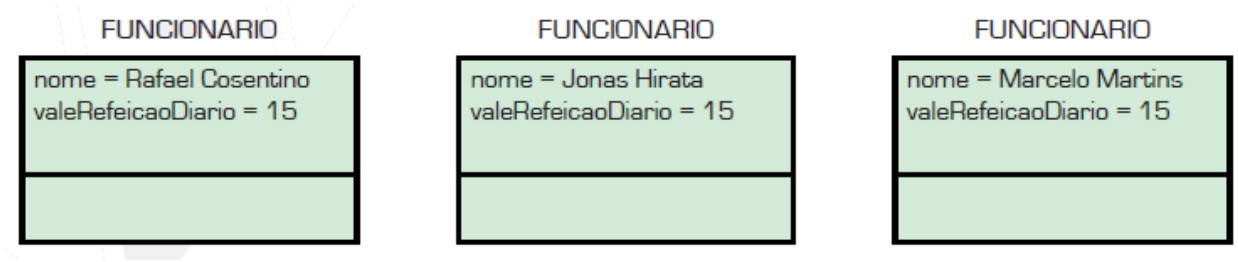


Figura 1: Atributos de instância

Para que o atributo valeRefeicaoDiario não se repita em cada objeto da classe Funcionario, devemos torná-lo um atributo de classe ao invés de um atributo de instância. Para isso, devemos aplicar o modificador **static** na declaração do atributo.

```
1  private class Funcionario
2  {
3      public string nome;
4      public double salario;
5      public static double valeRefeicaoDiario;
6
7      0 references
8      public void AumentaSalario(double aumento)
9      {
10         this.salario += aumento;
11     }
   }
```

Tabela 3: Funcionario.cs

Um atributo de classe deve ser acessado através do nome da classe na qual ele foi definido.

```
1  Funcionario.valeRefeicaoDiario = 15;
```

Tabela 4: Acessando um atributo de classe

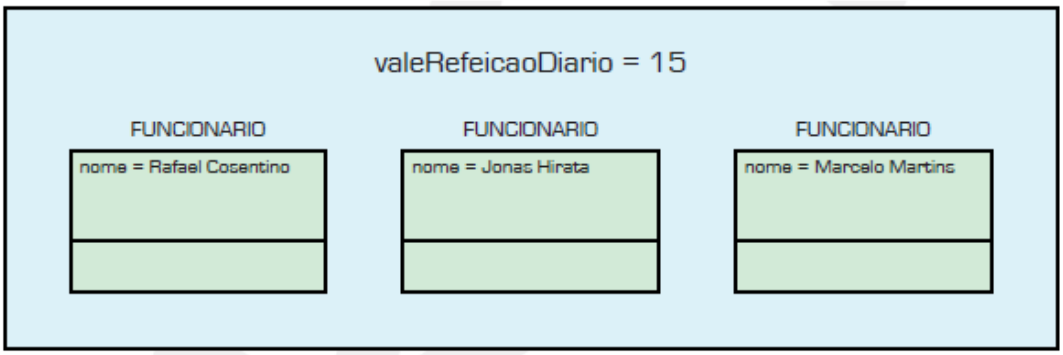


Figura 2: Atributos de classe

1.2 MÉTODOS ESTÁTICOS

Definimos métodos para implementar as lógicas que manipulam os valores dos atributos de instância. Podemos fazer o mesmo para os atributos de classe.

Suponha que o banco tenha um procedimento para reajustar o valor do vale refeição baseado em uma taxa. Poderíamos definir um método na classe Funcionario para implementar esse reajuste.

```
1  public void ReajustaValeRefeicaoDiario(double taxa)
2  {
3      Funcionario.valeRefeicaoDiario += Funcionario.valeRefeicaoDiario * taxa;
4  }
```

Tabela 5: Método que reajusta o valor do vale refeição

O método ReajustaValeRefeicaoDiario() é de instância. Consequentemente, ele deve ser chamado a partir da referência de um objeto da classe Funcionario.

Contudo, como o reajuste do valor do vale refeição não depende dos dados de um funcionário em particular, não faz sentido precisar de uma referência de um objeto da classe Funcionario para poder fazer esse reajuste.

Neste caso, poderíamos definir o ReajustaValeRefeicaoDiario() como método de classe ao invés de método de instância. Aplicando o modificador static nesse método, ele se tornará um método de classe. Dessa forma, o reajuste poderia ser executado independentemente da existência de objetos da classe Funcionário.

```
1  public void static ReajustaValeRefeicaoDiario(double taxa)
2  {
3      Funcionario.valeRefeicaoDiario += Funcionario.valeRefeicaoDiario * taxa
4  }
```

Tabela 6: Método que reajusta o valor do vale refeição

Um método de classe deve ser chamado através do nome da classe na qual ele foi definido.

```
1  Funcionario.ReajustaValeRefeicaoDiario(0.1);
```

Tabela 7: Chamando um método de classe

1.3 EXERCÍCIOS DE FIXAÇÃO

1) Crie um projeto novo chamado Unidade 4.

2) Crie uma classe chamada **Conta**. Defina um atributo de classe para contabilizar o número de objetos instanciados a partir da classe Conta. Esse atributo deve ser incrementado toda vez que um objeto é criado. Você pode utilizar construtores para fazer o incremento.

```
1  class Conta
2  {
3      // ATRIBUTO DE CLASSE
4      public static int contador;
5
6      // CONSTRUTOR
7      0 references
8      public Conta ()
9      {
10         Conta.contador++;
11     }
   }
```

Tabela 8: Conta.cs

3) Faça um teste criando dois objetos da classe Conta. Imprima o valor do contador de contas antes e depois da criação de cada objeto.

```
1  class Teste
2  {
3      0 references
4      private static void Main(string[] args)
5      {
6          Console.WriteLine(" Contador : " + Conta.contador);
7          new Conta();
8          Console.WriteLine(" Contador : " + Conta.contador);
9          new Conta();
10         Console.WriteLine(" Contador : " + Conta.contador);
11     }
   }
```

Tabela 9: Teste.cs

4) O contador de contas pode ser utilizado para gerar um número único para cada conta. Acrescente na classe Conta um atributo de instância para guardar o número das contas. Implemente no construtor a lógica para gerar esses números de forma única através do contador de contas.

```
1  public class Conta
2  {
3      // ATRIBUTO DE CLASSE
4      public static int contador;
5
6      // ATRIBUTO DE INSTÂNCIA
7      public int numero;
8
9      // CONSTRUTOR
10     2 references
11     public Conta()
12     {
13         Conta.contador++;
14         this.numero = Conta.contador;
15     }
16 }
```

Tabela 10: Conta.cs

5) Altere o teste para imprimir o número de cada conta criada.

```
1  class Teste
2  {
3      0 references
4      private static void Main(string[] args)
5      {
6
7          Console.WriteLine(" Contador : " + Conta.contador);
8
9          Conta c1 = new Conta();
10         Console.WriteLine(" Numero da primeira conta : " + c1.numero);
11
12         Console.WriteLine(" Contador : " + Conta.contador);
13
14         Conta c2 = new Conta();
15         Console.WriteLine(" Numero da segunda conta : " + c2.numero);
16
17         Console.WriteLine(" Contador : " + Conta.contador);
18     }
19 }
```

Tabela 11: Teste.cs

6) Adicione um método de classe na classe Conta para zerar o contador e imprimir o total de contas anterior.

```
1  public static void ZeraContador()
2  {
3      Console.WriteLine(" Contador : " + Conta.contador);
4      Console.WriteLine(" Zerando o contador de contas ... ");
5      Conta.contador = 0;
6  }
```

Tabela 12: Método estáticos

7) Altere o teste para utilizar o método ZeraContador().

```
1  class Teste
2  {
3      0 references
4      private static void Main(string[] args)
5      {
6          Console.WriteLine(" Contador : " + Conta.contador);
7          Conta c1 = new Conta();
8          Console.WriteLine(" Numero da primeira conta : " + c1.numero);
9
10         Console.WriteLine(" Contador : " + Conta.contador);
11
12         Conta c2 = new Conta();
13         Console.WriteLine(" Numero da segunda conta : " + c2.numero);
14
15         Console.WriteLine(" Contador : " + Conta.contador);
16
17         Conta.ZeraContador();
18     }
19 }
```

Tabela 13: Teste.cs

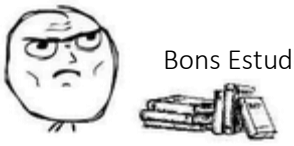
1.4 Exercícios Complementares

1) Crie uma classe para modelar os funcionários do banco. Defina nessa classe um atributo para armazenar o valor do vale refeição diário pago aos funcionários.

2) Faça um teste para verificar o funcionamento do vale refeição.

3) Defina um método para reajustar o vale refeição diário a partir de uma taxa.

4) Faça um teste para verificar o funcionamento do reajuste do vale refeição.



Bons Estudos!