Lógica e Pensamento Computacional II com C#

Prof. Waldeck Lindoso Jr.

Modificadores de acesso

	própria classe	subclasses do assembly	classes do assembly	subclasses fora do assembly	classes fora do assembly
public	x	x	x	x	x
protected internal	x	x	x	x	
internal	x	x	x		
protected	x	x		x	
private protected	x	x			
private	x				

Encapsulamento e Métodos de acesso(**get e set**)

É uma das principais técnicas que define a programação orientação a objetos.

Esse pilar, visa a proteção de variáveis importantes dentro de uma classe que não pode ser manipulada diretamente.

Encapsulamento e Métodos de acesso(**get e set**)

Sem encapsulamento

```
class Mensagem
{
    public string TextoMensagem;

    2 references
    public void ExibirMensagem()
    {
        Console.WriteLine(this.TextoMensagem);
    }
}
```

```
static void Main(string[] args)
{
    Mensagem msg1, msg2;

    msg1 = new Mensagem();
    msg1.TextoMensagem = "Primeiro objeto";
    msg1.ExibirMensagem();

    msg2 = new Mensagem();
    msg2.TextoMensagem = "Segundo objeto";
    msg2.ExibirMensagem();

}
```

Encapsulamento e Métodos de acesso(**get e set**)

Com encapsulamento (modo antigo)

```
class Mensagem
{
    private string TextoMensagem;
    0 references
    public string getTextoMensagem()
    {
        return this.TextoMensagem;
    }
    0 references
    public void setTextoMensagem(string txt)
    {
        this.TextoMensagem = txt;
    }
    2 references
    public void ExibirMensagem()
    {
        Console.WriteLine(this.TextoMensagem);
    }
}
```

```
static void Main(string[] args)
{
    Mensagem msg1, msg2;

    msg1 = new Mensagem();
    msg1.TextoMensagem = "Primeiro objeto";
    msg1.ExibirMensagem();

    msg2 = new Mensagem();
    msg2.TextoMensagem = "Segundo objeto";
    msg2.ExibirMensagem();
}
```

Encapsulamento e Métodos de acesso(**get e set**)

Com encapsulamento (modo antigo)

```
class Mensagem
{
    private string TextoMensagem;
    0 references
    public string getTextoMensagem()
    {
        return this.TextoMensagem;
    }
    0 references
    public void setTextoMensagem(string txt)
    {
        this.TextoMensagem = txt;
    }
    2 references
    public void ExibirMensagem()
    {
        Console.WriteLine(this.TextoMensagem);
    }
}
```

```
static void Main(string[] args)
{
    Mensagem msg1, msg2;

    msg1 = new Mensagem();
    msg1.setTextoMensagem("Primeiro objeto");
    msg1.ExibirMensagem();

    msg2 = new Mensagem();
    msg2.setTextoMensagem("Segundo objeto");
    msg2.ExibirMensagem();
}
```

Propriedades no C#

Encapsulamento através de Propriedades

```
class Mensagem
{
    private string textoMensagem;
    2 references
    public string TextoMensagem
    {
        get
        {
            return this.textoMensagem;
        }
        set
        {
            this.textoMensagem = value.ToUpper();
        }
    }
    2 references
    public void ExibirMensagem()
    {
        Console.WriteLine(this.textoMensagem);
    }
}
```

```
static void Main(string[] args)
{
    Mensagem msg1, msg2;

    msg1 = new Mensagem();
    msg1.TextoMensagem = "Primeiro objeto";
    msg1.ExibirMensagem();

    msg2 = new Mensagem();
    msg2.TextoMensagem = "Segundo objeto";
    msg2.ExibirMensagem();
}
```

Propriedades no C#

Outra forma:

```
class Mensagem
{
    private string textoMensagem;
    2 references
    public string TextoMensagem
    {
        get{return this.textoMensagem;}
        set{this.textoMensagem = value.ToUpper();}
    }
    2 references
    public void ExibirMensagem()
    {
        Console.WriteLine(this.textoMensagem);
    }
}
```

```
static void Main(string[] args)
{
    Mensagem msg1, msg2;

    msg1 = new Mensagem();
    msg1.TextoMensagem = "Primeiro objeto";
    msg1.ExibirMensagem();

    msg2 = new Mensagem();
    msg2.TextoMensagem = "Segundo objeto";
    msg2.ExibirMensagem();
}
```

Propriedades no C#

Porém desde a versão 3.0 do C#

```
class Mensagem
{
    3 references
    public string TextoMensagem { get; set; }
    2 references
    public void ExibirMensagem()
    {
        Console.WriteLine(this.TextoMensagem);
    }
}
```

```
static void Main(string[] args)
{
    Mensagem msg1, msg2;

    msg1 = new Mensagem();
    msg1.TextoMensagem = "Primeiro objeto";
    msg1.ExibirMensagem();

    msg2 = new Mensagem();
    msg2.TextoMensagem = "Segundo objeto";
    msg2.ExibirMensagem();
}
```

Sobrecarga de Métodos

```
class Calculadora
{
    1 reference
    public int somar(int num1, int num2)
    {
        return num1 + num2;
    }
    1 reference
    public double somar(double num1, double num2)
    {
        return num1 + num2;
    }
    1 reference
    public int somar(int num1, int num2, int num3)
    {
        return num1 + num2 + num3;
    }
}
```

```
static void Main(string[] args)
{
    Calculadora calc = new Calculadora();
    Console.WriteLine("Somar(int, int): " + calc.somar(10, 5));
    Console.WriteLine("Somar(double, double): " + calc.somar(10.5, 5.5));
    Console.WriteLine("Somar(int, int, int): " + calc.somar(10, 5, 6));
}
```

Herança

Classe base - Classe pai, que está sendo herdada por uma ou mais classes

```
class Veiculo // Classe base (pai)
{
    1 reference
    public string Marca { get; set; }

    1 reference
    public void buzinar()
    {
        Console.WriteLine("Tuut, tuut!!");
    }
}
```

Herança

Classe derivada - Classe filha, que está herdando de outra classe(pai)

```
class Carro : Veiculo // Classe derivada(filha)
{
    1 reference
    public string Modelo { get; set; }
}
```

Herança

Testando...

```
static void Main(string[] args)
{
    Carro meuCarro = new Carro();
    meuCarro.buzinar();
    Console.WriteLine($"{meuCarro.Marca} {meuCarro.Modelo}");
}
```

Polimorfismo

Polimorfismo significa "muitas formas" e ocorre quando temos muitas classes que estão relacionadas entre si por herança. A herança nos permite herdar campos e métodos de outra classe. O polimorfismo usa esses métodos para realizar diferentes tarefas. Isso nos permite realizar uma única ação de maneiras diferentes.

Polimorfismo

```
class Cachorro : Animal
{
    0 references
    public void emitirSom()
    {
        Console.WriteLine("O cachorrinho emitiu um som");
    }
}
```

```
class Porco : Animal
{
    0 references
    public void emitirSom()
    {
        Console.WriteLine("O porquinho emitiu um som");
    }
}
```

```
class Animal
{
    3 references
    public void emitirSom()
    {
        Console.WriteLine("O animal emitiu um som.");
    }
}
```

```
static void Main(string[] args)
{
    Animal meuAnimal = new Animal();
    Animal meuPorco = new Porco();
    Animal meuCachorro = new Cachorro();

    meuAnimal.emitirSom();
    meuPorco.emitirSom();
    meuCachorro.emitirSom();
}
```

Polimorfismo (**virtual e override**)

```
class Cachorro : Animal
{
    4 references
    public override void emitirSom()
    {
        Console.WriteLine("O cachorrinho emitiu um som");
    }
}
```

```
class Porco : Animal
{
    4 references
    public override void emitirSom()
    {
        Console.WriteLine("O porquinho emitiu um som");
    }
}
```

```
class Animal
{
    5 references
    public virtual void emitirSom()
    {
        Console.WriteLine("O animal emitiu um som.");
    }
}
```

```
static void Main(string[] args)
{
    Animal meuAnimal = new Animal();
    Animal meuPorco = new Porco();
    Animal meuCachorro = new Cachorro();

    meuAnimal.emitirSom();
    meuPorco.emitirSom();
    meuCachorro.emitirSom();
}
```

Perguntas ??