Disciplina: POO - Programação Orientada a Objetos

Encapsulamento

Profª. Drª. Giovana Angélica Ros Miola
giovana.miola@fatec.sp.gov.br

Problema?

- Já pensou se um membro da sua equipe realizasse isto?

```
//em um arquivo
conta.saldo -= 200;
//em outro arquivo
conta.saldo -= 1000;
//em outro arquivo
conta.saldo -= 500;
```

- Não existe nada no software que impediu esses acessos aos atributos diretamente

Encapsulamento

- Processo de ocultar ou esconder os membros (atributos) de uma classe do acesso exterior usando modificadores de acesso
- Fornece uma maneira de **preservar** a integridade do estado dos dados
- A classe bem encapsulada, deve ocultar seus dados e os detalhes de implementação do mundo exterior, processo denominado programação *caixa preta*
- A implementação do método pode ser alterada pelo autor da classe sem quebrar qualquer código existente
- A implementação se torna mais robusta, protegendo os atributos
- Separar o programa em partes, o mais isoladas possível

Por que Controlar o Acesso?

- Proteger informação privada.
- Esclarecer como outros programadores devem usar sua classe.
- Manter a implementação separada da interface.
- **Exemplo hipotético**: Suponha que você tenha uma classe que é responsável por transações de cartões de credito (geralmente este tipo de classe pertence a companhia de cartões de credito) e foi desenvolvida por um outro programador e esta dentro de uma DLL ou EXE, acontece que o desenvolvedor terceirizado cometeu um deslize deixando visível a propriedade numero do cartão. Um desenvolvedor malicioso pode facilmente capturar o número do cartão referenciando o objeto e realizar operações inapropriadas.

Modificador de acesso/visibilidade

- Todo o acesso deve ser feito por meio dos métodos públicos, não permitindo aos outros **COMO** a classe faz o trabalho dela, mas mostrando apenas **O QUE** ela faz
- Define o escopo e a visibilidade de um atributo (membro) da classe
- C# suporta:
  - **public**
  - **private**
  - **internal**
  - **protected**
  - **protected internal**

Modificador de acesso: public

- Permite que uma classe **exponha** seus atributos (variáveis de membros) e funções a outras funções e objetos.
- Qualquer atributo público pode ser acessado de fora da classe, mas a OO preza pelo encapsulamento.

```
class Funcionario
{
    public int codigo;
    public string nome;
    public double salario;
    public void mostraDados()
    {
       Console.WriteLine("Código: "+codigo+
                         "\tNome: " + nome +
                   "\tSalário: " + salario);
    }
}
```

```
class Program
{
   static void Main(string[] args)
   {
      Funcionario f = new Funcionario();
      f.codigo = 54;
      f.nome = "Ana";
      f.salario = 1000;
      f.mostraDados();
   }
}
```

Modificador de acesso: private

- Permite apenas o acesso local aos atributos, ou seja, os atributos ou métodos **private** apenas podem ser acessados pela própria classe
- O método Main(), somente consegue acessar os atributos pelo método MostraAtributos()
- Adequando...

```
 9   class Funcionario
10   {
11       private int codigo;
12       private string nome;
13       private double salario;
14       private void MostraDados()
15       {
16           Console.WriteLine("Código: " + codigo +
17               "\tNome: " + nome + "\tSalário: " + salario);
18       }
19   }
```

```
 9   class Program
10   {
11       static void Main(string[] args)
12       {
13           Funcionario f = new Funcionario();
14           f.codigo = 9;
15           f.nome = "Ana";
16           f.salario = 1000;
17           f.Mostra
18           Console.
19       }
20   }
```

struct System.Double

Represents a double-precision floating-point number.

"Funcionario.salario" é inacessível devido ao seu nível de proteção

Modificador de acesso: private

```
9   class Funcionario
10  {
11      private int codigo;
12
13      private string nome;
14      private double salario;
15      public double Salario
16      {
17          set
18          {
19              this.salario = value;
20          }
21          get
22          {
23              return this.salario;
24          }
25      }
```

```
9   class Program
10  {
11      static void Main(string[] args)
12      {
13          Funcionario f = new Funcionario();
14          f.codigo = 54;
15          f.nome = "Ana";
16          f.Salario = 1000;
17          f.mostraDados();
18      }
```

A declaração de uma **propriedade** é parecida com a declaração de um atributo, porém é necessário informar o que deve ser feito na alteração (**set**) e na obtenção (**get**) da propriedade

```
//método para modificar o salário
public void setSalario(double salario)
{
    this.salario = salario;
}
//método para pegar o salário
public double getSalario()
{
    return salario;
}
```

Exemplo em Java

Estes métodos apenas ilustram o que as *properties* C# fazem

A palavra ***this*** refere-se sempre ao contexto (objeto) atual. Na grande maioria dos casos, a palavra ***this*** pode ser omitida, mas quando existem coincidências entre nomes de variáveis ou parâmetros com campos ou propriedades de uma classe, é conveniente usar ***this***

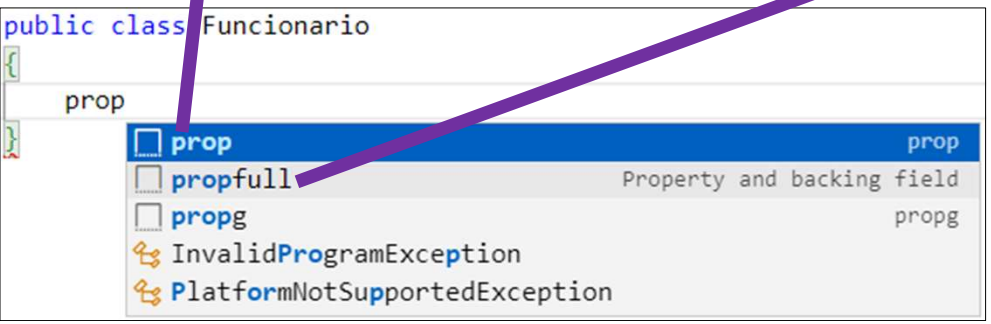
This

- O **this** é uma palavra reservada que é usada para a autorreferência
- Esta ocorre quando quer referenciar métodos e atributos da classe e objeto
- É usual utilizar this com membros de instância (os atributos)
- Cada objeto guarda seu próprio estado nos atributos, e o uso do **this** pode ajudar a diferenciar, por exemplo, parâmetros que possam vir a ter o mesmo nome dos atributos

Encapsulamento, sem validação para os conteúdos dos atributos

```
public class Funcionario
    {
        public int Codigo { get; set; }
        public string Nome { get; set; }
        public double Salario{ get; set; }
    }
```

Dica:




```
public class Funcionario
{
    2 references
    private int myVar;
    0 references
    public int MyProperty
    {
        get { return myVar; }
        set { myVar = value; }
    }

}
```

Encapsulamento de todos os atributos

```
public class Funcionario
{
    // atributos encapsulados
    2 references
    private int codigo;
    2 references
    private string nome;
    2 references
    private double salario;
    //Propriedades - get/set
    //Métodos de encapsulamento
    0 references
    public int Codigo
    {
        get { return codigo; }
        set { codigo = value; }
    }
    0 references
    public string Nome
    {
        get { return nome; }
        set { nome = value; }
    }
    0 references
    public double Salario
    {
        get { return salario; }
        set { salario = value; }
    }
}
```

https://sharplab.io

Exemplo de como a propriedade Numero (get e set), trabalha implicitamente, com o atributo numero encapsulado com o modificador de acesso private

```
using System;
public class Conta {
   public int Numero {get; set;}

   public void MostraAtributo() {
      Console.WriteLine("Número da conta:" + Numero);
   }
}
```

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Security;
using System.Security.Permissions;

[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: Debuggable(DebuggableAttribute.DebuggingModes.Default |
DebuggableAttribute.DebuggingModes.IgnoreSymbolStoreSequencePoints |
DebuggableAttribute.DebuggingModes.EnableEditAndContinue |
DebuggableAttribute.DebuggingModes.DisableOptimizations)]
[assembly: SecurityPermission(SecurityAction.RequestMinimum, SkipVerification = true)]
[assembly: AssemblyVersion("0.0.0.0")]
[module: UnverifiableCode]
public class Conta
{
   [CompilerGenerated]
   [DebuggerBrowsable(DebuggerBrowsableState.Never)]
   private int <Numero>k__BackingField;

   public int Numero
   {
      [CompilerGenerated]
      get
      {
         return <Numero>k__BackingField;
      }
      [CompilerGenerated]
      set
      {
         <Numero>k__BackingField = value;
      }
   }

   public void MostraAtributo()
   {
      Console.WriteLine(string.Concat("Número da conta:", Numero.ToString()));
   }
}
```

Encapsulamento, com validação para conteúdo de alguns atributos

```
public class Funcionario
{
    // atributos encapsulados
    3 references
    private int codigo;
    3 references
    private string nome;
    2 references
    private double salario;
    //Propriedades - get/set
    //Métodos de encapsulamento
    0 references
    public int Codigo
    {
        get => codigo;
        set {
            if (codigo > 0)
                codigo = value;
            else
                Console.WriteLine("Código inválido!");
        }
    }
```

Body Expressions

... Continuação ...

Propriedade

Validação antes do armazenamento ao atributo de um objeto

```
public string Nome
{
    get => nome;
    set {
        if (nome != "")
            nome = value;
        else
            Console.WriteLine("O nome não pode estar vazio!");
    }
}
0 references
public double Salario
{
    get => salario;
    set => salario = value;
}
```

Padrão C# para o encapsulamento

Por padrão a classe só pode ser vista dentro do próprio projeto (visível apenas no *assembly* que a declarou), esse é um nível de visibilidade conhecido como **internal**. Quando for necessário, trabalhar com bibliotecas externas ao projeto, as classes precisam ser declaradas como **public**.

Modificador de acesso Internal

Body Expressions

```
class Funcionario
{
    private int codigo;
    private double salario;
    private string nome;

    //public int Codigo { get => codigo; set => codigo = value; }... ou
    public int Codigo { get; set; }

    public string Nome { get => nome; set => nome = value; }
    public double Salario { get => salario; set => salario = value; }

    private void MostraDados()
    {
        Console.WriteLine("Código: " + Codigo +
            "\tNome: " + Nome + "\tSalário: " + Salario);
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Funcionario f = new Funcionario();
        f.Codigo = 54; // Sem erro C
        f.nome = "Ana";    maiúsculo
        f.Salario = 1000;
        f.MostraDados();
    }
}
```

//Com erro, n letra minúscula

Geração do arquivo Assembly

Modificador de acesso: protected

- Atributos e métodos com o modificador de acesso **protected** são visíveis a própria classe e para as classes filhas

- O **protected internal** tem o mesmo comportamento do **protected** com uma restrição a mais, não pode ser visto por outro assembly a não ser que a classe seja herdada.

- Este modificador será utilizado no assunto Herança