



PROGRAMAÇÃO O.O. (C#)



Encapsulamento
Professor: João Luiz Lagôas



Classes são compartilhadas

- Aplicações e sistemas reais são construídas através da associação de diversas classes diferentes.
- Cada programador pode criar suas próprias classes assim como pode utilizar classes que estão prontas e disponibilizadas por outros programadores.
- No caso de criar classes, é importante que o programador ofereça um código íntegro, de modo a reduzir a chance de outros desenvolvedores a criarem bugs ao reutilizar seu código.

É necessário portanto entender os três pilares da programação O.O.



Pilares da programação O.O.



Encapsulamento



- Recurso que provê proteção de acesso aos membros internos de um objeto.
- Alguns atributos necessitam de tratamento exclusivo pelos próprios métodos da classe para não serem modificados através de valores inconsistentes.

Atenção: Lembre-se que uma classe possui responsabilidade sobre os atributos que contém! Sendo assim, **os atributos quase nunca devem ser acessados diretamente** de fora do escopo de uma classe!



Por que me preocupar com encapsulamento?



```
class Relogio
{
    public int Horas;
    public int Minutos;
    public int Segundos;
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Relogio rel = new Relogio(12,30,15);

        rel.Horas = 792;
        rel.Minutos = 3728;
        rel.Segundos = -8372;

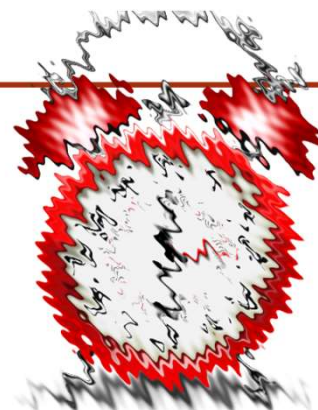
        rel.MostrarHorario();

        Console.ReadLine();
    }
}
```

classe
(Relogio)



new



objeto
(rel)

Como criar encapsulamento?



- O conceito de encapsulamento é obtido através do uso de modificadores de acesso.
- Apesar de C# ser um pouco mais sofisticado, basicamente existem dois tipos de modificadores de acesso: **público** (*public*) e **privado** (*private*).
- Um membro público pode ser livremente acessado por código definido fora da sua classe. Esse é o tipo de modificador de acesso que nós utilizamos até este ponto.
- Um membro privado pode ser acessado apenas por métodos definidos dentro da sua classe. É através do uso desse modificador que controlamos o acesso aos membros internos da classe.

Restringindo o acesso a membros da classe é parte fundamental da programação orientada a objetos. Dessa forma, o encapsulamento previne o uso inadequado de um objeto.

Public



- Quando definimos um atributo ou método public, estamos indicando que esses elementos são visíveis **dentro e fora da classe.**

Qualquer classe do Sistema pode ter acesso aos atributos e métodos com visibilidade public.



Exemplo



```
class Relogio
{
    public int Horas;
    public int Minutos;
    public int Segundos;
}
```

Os atributos
são públicos

```
class Program
{
    static void Main(string[] args)
    {
        Relogio rel = new Relogio(12,30,15);

        rel.Horas = 792;
        rel.Minutos = 3728;
        rel.Segundos = -8372;

        rel.MostrarHorario();

        Console.ReadLine();
    }
}
```

De fora da classe Relogio,
conseguimos acessar e
modificar os atributos de uma
instância de Relogio (rel).

Private



- Quando definimos um atributo ou método private, estamos indicando que esses elementos são visíveis **apenas dentro da classe.**

Somente dentro da classe os atributos e métodos podem ser acessados com visibilidade private.



Exemplo



```
class Relogio
{
    private int Horas;
    private int Minutos;
    private int Segundos;
}
```

Os atributos
são privados

```
class Program
{
    static void Main(string[] args)
    {
        Relogio rel = new Relogio(12,30,15);

        rel.Horas = 792;
        rel.Minutos = 3728;
        rel.Segundos = -8372;

        rel.MostrarHorario();

        Console.ReadLine();
    }
}
```

De fora da classe Relogio, **NÃO** conseguimos acessar e modificar os atributos de uma instância de Relogio (rel).

Error CS0122 Relogio.Horas' is inaccessible due to its protection level 0



Setters e Getters



Se não podemos alterar o valor dos atributos dessa forma, como faremos???

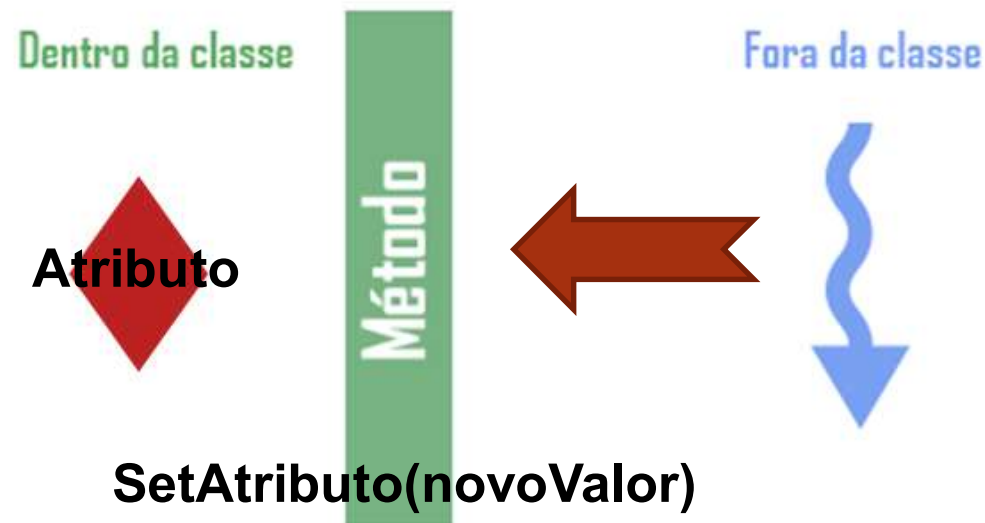
- Simples! Criaremos métodos públicos na classe para manipular os atributo privados de modo adequado.
- Dessa forma, garantimos a consistência dos atributos internos de uma classe através do que chamamos de métodos acessores (*getters*) e modificadores (*setters*).



Setters



- Métodos *Setters* ou Métodos Modificadores são utilizados para garantir que a alteração em **atributos internos privados** de uma classe seja realizada de modo íntegro.



Exemplo Prático

Método Set



```
class Relogio
{
    private int Horas;
    private int Minutos;
    private int Segundos;

    public Relogio(int h, int m, int s)
    {
        Horas = h;
        Minutos = m;
        Segundos = s;
    }

    public void MostrarHorario()
    {
        Console.WriteLine("{0}:{1}:{2}", Horas, Minutos, Segundos);
    }

    public void SetHoras(int h)
    {
        if(h >= 0 && h <= 23)
            Horas = h;
    }
}
```

Método
Setter

Exemplo 1



```
class Program
{
    static void Main(string[] args)
    {
        Relogio rel = new Relogio(12,30,15);

        rel.SetHoras(792);

        rel.MostrarHorario();

        Console.ReadLine();
    }
}
```

O que será mostrado no console??

Exemplo 1



```
class Program
{
    static void Main(string[] args)
    {
        Relogio rel = new Relogio(12,30,15);

        rel.SetHoras(792);

        rel.MostrarHorario();

        Console.ReadLine();
    }
}
```

O que será mostrado no console??

12:30:15

Exemplo 2



```
class Program
{
    static void Main(string[] args)
    {
        Relogio rel = new Relogio(12,30,15);

        rel.SetHoras(6);

        rel.MostrarHorario();

        Console.ReadLine();
    }
}
```

O que será mostrado no console??

Exemplo 2

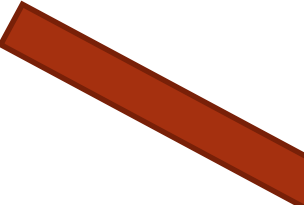


```
class Program
{
    static void Main(string[] args)
    {
        Relogio rel = new Relogio(12,30,15);

        rel.SetHoras(6);

        rel.MostrarHorario();

        Console.ReadLine();
    }
}
```



O que será mostrado no console??

6:30:15

Setters



- Para cada atributo privado de uma classe, é comum (mas não obrigatório) definir um método Setter.

```
public void SetHoras(int h)
{
    if(h >= 0 && h <= 23)
        Horas = h;
}

public void SetMinutos(int m)
{
    if(m >= 0 && h <= 59)
        Minutos = m;
}

public void SetSegundos(int s)
{
    if(h >= 0 && h <= 59)
        Segundos = s;
}
```



Repare que os métodos Setters agem como filtros evitando que os atributos da classe recebam valores inconsistentes.

Setters



Boa prática:

Apesar dos métodos Setters por definição não retornarem valores, é comum definir um valor de retorno booleano para sinalizar ao chamador do método se a modificação foi realizada ou não.

```
public bool SetHoras(int h)
{
    if(h >= 0 && h <= 23)
    {
        Horas = h;
        return true;
    }
    else
        return false;
}
```

Exemplo



```
class Program
{
    static void Main(string[] args)
    {
        Relogio rel = new Relogio(12,30,15);

        if(rel.SetHoras(792))
            Console.WriteLine("Alteração nas horas feita com sucesso!");
        else
            Console.WriteLine("Não foi possível ajustar as horas, valor inválido");

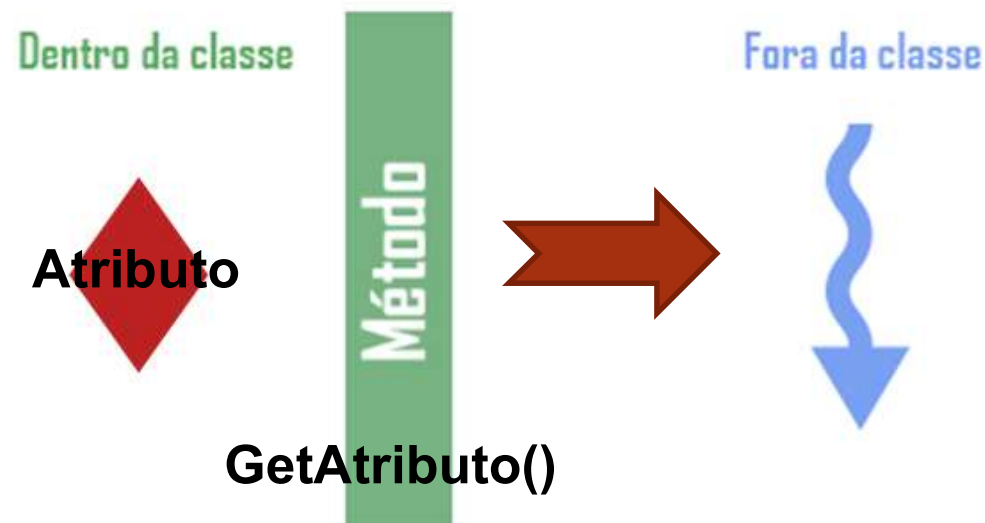
        rel.MostrarHorario();

        Console.ReadLine();
    }
}
```

Getters



- Métodos *Getters* ou Métodos Acessores são utilizados para garantir que a recuperação de atributos internos de uma classe seja realizada de modo íntegro. Isso é feito geralmente através do retorno de uma cópia do atributo!



Exemplo Prático

Método Get



```
class Relogio
{
    Atributos {
        private int Horas;
        private int Minutos;
        private int Segundos;
    }

    Outros Métodos {
        .
        .
        .
        .
    }

    Método Getter {
        public int GetHoras()
        {
            return Horas;
        }
    }
}
```



Exemplo



```
class Program
{
    static void Main(string[] args)
    {
        Relogio rel = new Relogio(12,30,15);

        Console.WriteLine("O hora atual do relógio vale {0}", rel.GetHoras());

        Console.ReadLine();
    }
}
```

Se for necessário recuperar o valor do atributo Horas do objeto rel, basta chamar seu método GetHoras(). Esse método irá retornar uma cópia do inteiro armazenado na variável de instância Horas.



Getters



- Para cada atributo privado de uma classe, é comum (mas não obrigatório) definir um método *Getter*.

```
public int GetHoras()
{
    return Horas;
}

public int GetMinutos()
{
    return Minutos;
}

public int GetSegundos()
{
    return Segundos;
}
```



Repare que os métodos *Getters* agem como mediadores. Eles repassam uma cópia do valor armazenado nas variáveis de instância.

Aplicando acesso Público e Privado



- O uso apropriado de acesso público e privado é um componente chave de se programar orientado a objeto com sucesso.
- Apesar de não existir uma regra rígida e estabelecida, existem alguns princípios que são úteis para se tomar a decisão:
 1. Membros de uma classe que são usados apenas dentro da classe devem ser privados.
 2. Dados que precisam ficar dentro de um intervalo de valores devem ser privados e estarem associados a métodos públicos que os tratem adequadamente.
 3. Métodos modificadores e acessores (setters e getters) devem ser públicos e seus valores relacionados privados.
 4. Atributos (variáveis de instância) podem ser públicas quando não há razão para se manter qualquer tipo de controle em seu valor.