



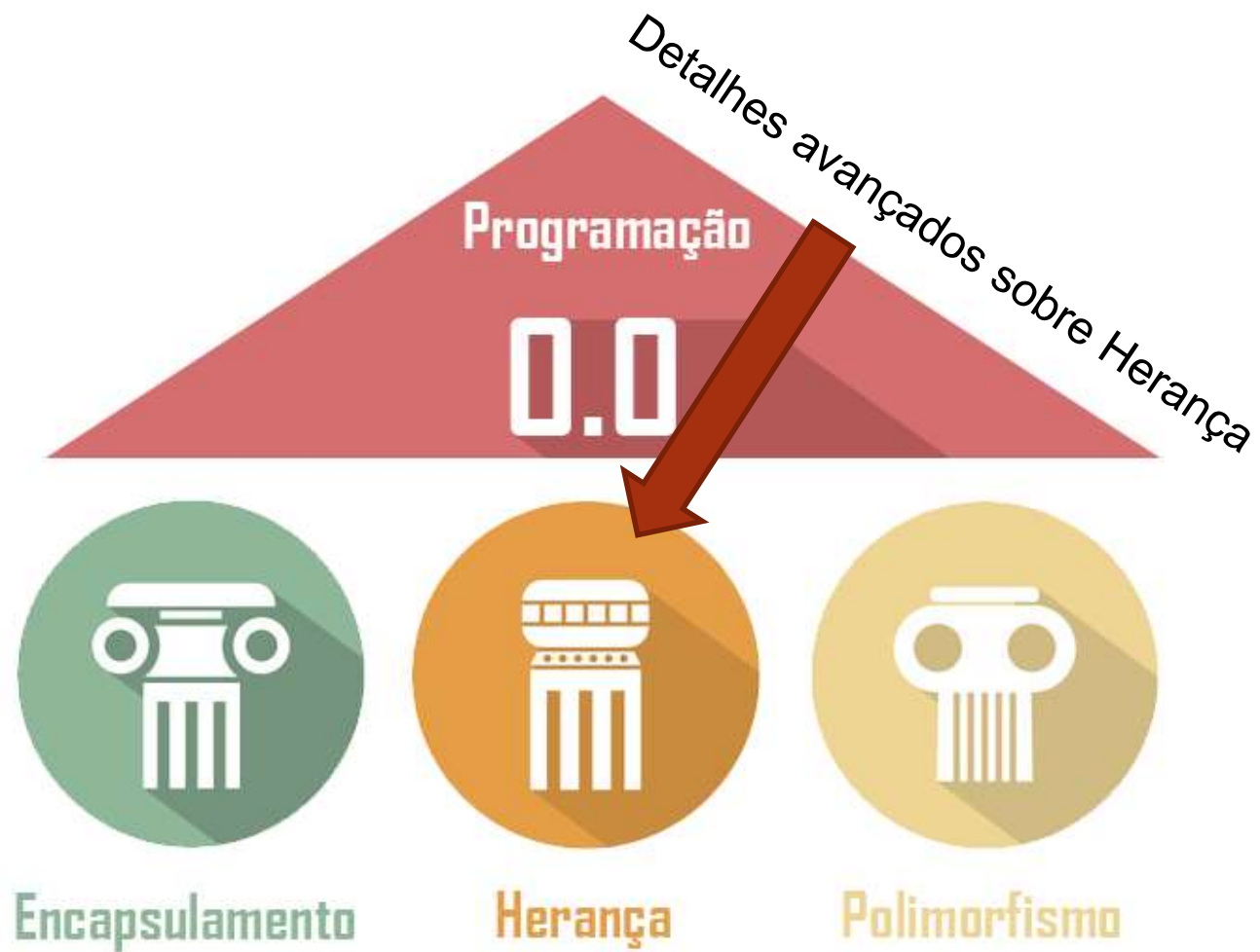
PROGRAMAÇÃO O.O. (C#)



Conceitos avançados de herança
Professor: João Luiz Lagôas



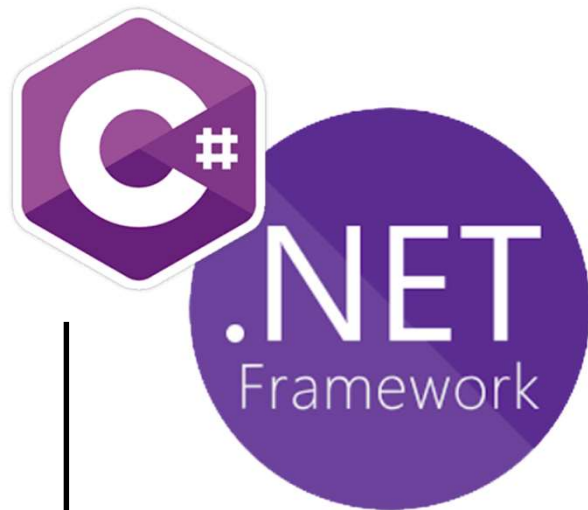
Roteiro



FCL e a Herança



- Várias das classes que nós utilizamos até agora implementam herança de alguma forma.



Framework Class Library

- Uma máquina virtual **CLR**;
- Um conjunto de bibliotecas robustas **FCL**;
- Um ambiente de desenvolvimento integrado completo **VS**;

FCL e a Herança



Framework Class Library



- Form
- TextBox
- Label
- ComboBox
- ListBox
- Button
- RadioButton
- CheckButton

Já sabemos que muitas dessas classes compartilham atributos e métodos semelhantes, como por exemplo o atributo **Text** que existe em quase todas elas.

Isso acontece porque a própria FCL implementa herança em várias classes para reusar o código e tornar sua biblioteca mais robusta e organizada.

Várias classes associadas à construção de formulários dentro do namespace Forms



FCL e a Herança



Documentação da classe ListControl no site da Microsoft

← → ↻ Seguro | [https://msdn.microsoft.com/pt-br/library/system.windows.forms.listcontrol\(v=vs.110\).aspx](https://msdn.microsoft.com/pt-br/library/system.windows.forms.listcontrol(v=vs.110).aspx) 🔍 ☆ 🇧🇷

- ▶ ListBox.SelectedIndexCollection Classe
- ▶ ListBox.SelectedObjectCollection Classe
- ▼ ListControl Classe
 - ▶ ListControl Métodos
 - ▶ ListControl Propriedades
 - ▶ ListControl Eventos
 - ListControl Construtor
- ▶ ListControlConvertEventArgs Classe
- ListControlConvertEventHandler Delegado
- ▶ ListView Classe
- ▶ ListView.CheckedIndexCollection Classe
- ▶ ListView.CheckedListViewItemCollection Classe
- ▶ ListView.ColumnHeaderCollection

Classe ListControl

.NET Framework (current version) | [Outras versões](#) ▼

📄 Observação

The .NET API Reference documentation has a new home. Visit the [.NET API Browser](#) on docs.microsoft.com to see the new experience.

Fornece uma implementação comum de membros para as classes [ListBox](#) e [ComboBox](#).

Namespace: [System.Windows.Forms](#)
Assembly: [System.Windows.Forms](#) (em [System.Windows.Forms.dll](#))

Hierarquia de Herança

```

System.Object
  System.MarshalByRefObject
    System.ComponentModel.Component
      System.Windows.Forms.Control
        System.Windows.Forms.ListControl
          System.Windows.Forms.ComboBox
          System.Windows.Forms.ListBox
  
```

🖨️ Imprimir

↶ Compartilhar

NESTE ARTIGO

- [Hierarquia de Herança](#)
- Sintaxe
- Construtores
- Propriedades
- Métodos
- Eventos
- Implementações Explícitas da Interface
- Comentários
- Exemplos
- Informações de Versão
- Acesso thread-safe
- Confira Também

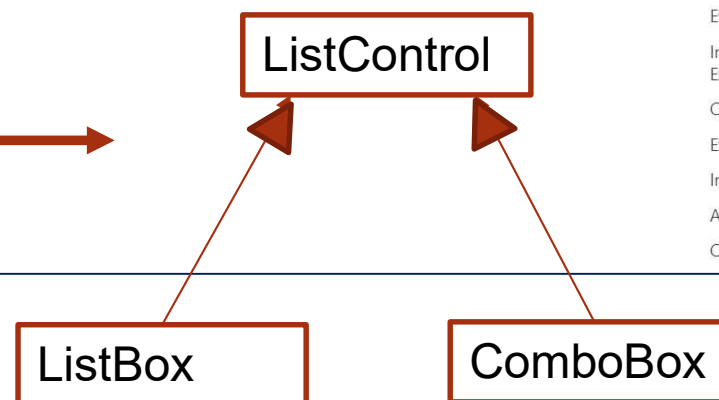


Diagrama de Classe da UML

Retornando ao exercício do Juros

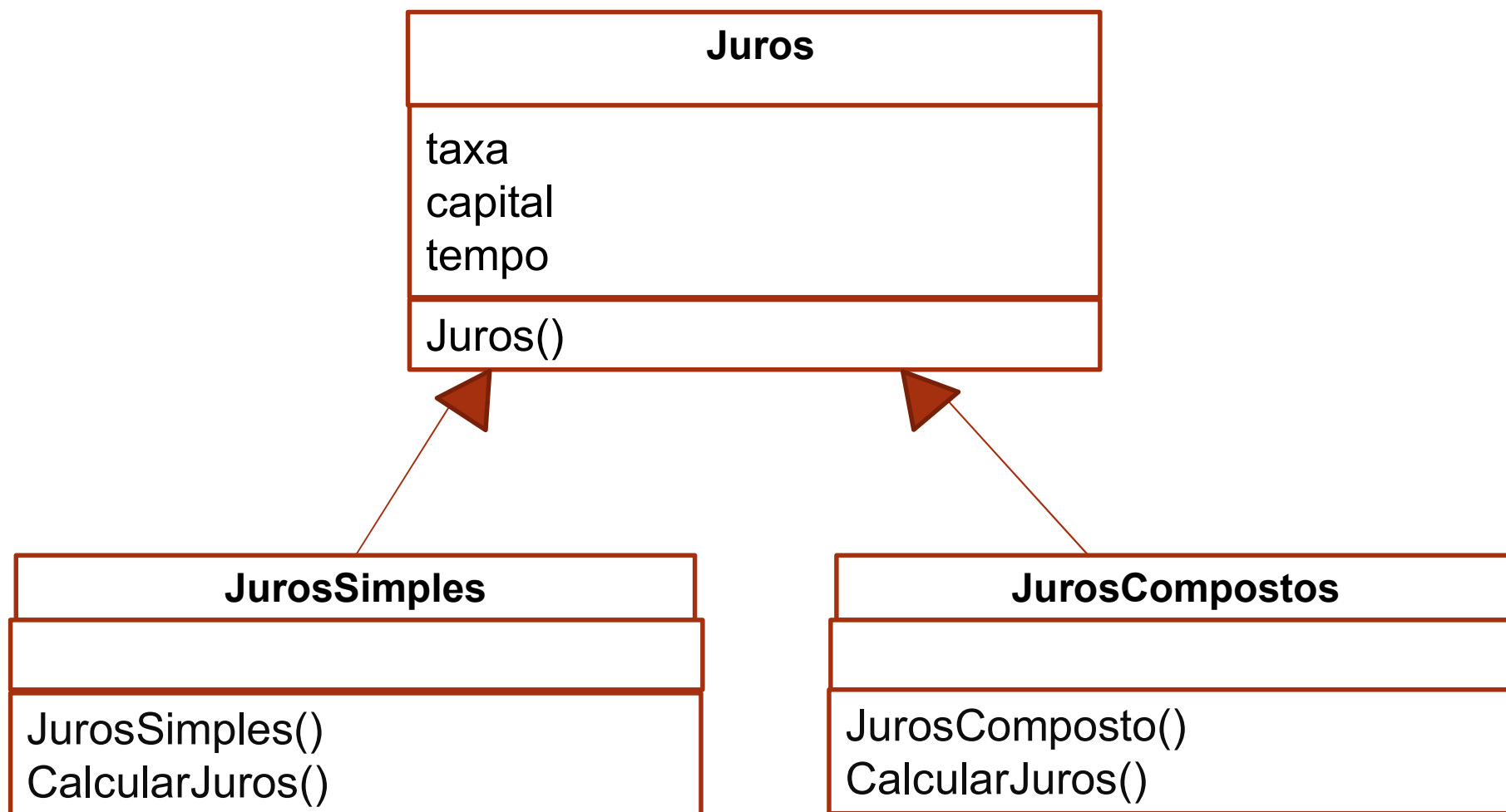


Proposta: iremos retomar o exercício dos Juros e pensar em várias alternativas para **melhorá-lo** aplicando conceitos mais avançados de Herança.



Exercício Juros

Diagrama de classe (UML)



Exercício Juros

Código



```
class Juros
```

```
{
```

```
    private int tempo;
```

```
    private double taxa;
```

```
    private double capitalInicial;
```

```
    public Juros(int tp, double tx, double cap)
```

```
{
```

```
        SetTempo(tp);
```

```
        SetTaxa(tx);
```

```
        SetCapitalInicial(cap);
```

```
}
```

• Métodos setters e getters

Classe correspondente
ao diagrama

Juros

taxa
capital
tempo

Juros()

Exercício Juros

Código



```
class JurosSimples : Juros
{
    public JurosSimples(int tp, double tx, double cap) :
        base(tp, tx, cap)
    {
        // Não preciso me preocupar por aqui já que
        // no construtor da minha classe pai a inicialização
        // dos meus atributos irá acontecer.
    }

    public double CalcularJuros()
    {
        return GetCapitalInicial() * GetTaxa() * GetTempo();
    }
}
```

Classe correspondente
ao diagrama

JurosSimples

JurosSimples()
CalcularJuros()

Exercício Juros

Código



```
class JurosCompostos : Juros
{
    public JurosCompostos(int tp, double tx, double cap) :
        base(tp, tx, cap)
    {
        // Não preciso me preocupar por aqui já que
        // no construtor da minha classe pai a inicialização dos
        // meus atributos irá acontecer.
    }

    public double CalcularJuros()
    {
        return GetCapitalInicial() * Math.Pow((1 + GetTaxa()),
            GetTempo()) - GetCapitalInicial();
    }
}
```

Classe correspondente
ao diagrama

JurosCompostos

JurosComposto()
CalcularJuros()

Exercício Juros

Código



```
class Program
{
    static void Main(string[] args)
    {
        int tempo = Convert.ToInt32(Console.ReadLine());
        double taxa = Convert.ToDouble(Console.ReadLine());
        double capitalInicial = Convert.ToDouble(Console.ReadLine());

        JurosCompostos jurosC = new JurosCompostos(tempo, taxa, capitalInicial);

        Console.WriteLine("O valor do juros composto é :" + jurosC.CalcularJuros());

        JurosSimples jurosS = new JurosSimples(tempo, taxa, capitalInicial);

        Console.WriteLine("O valor do juros simples é :" + jurosS.CalcularJuros());

        Juros juros = new Juros(tempo, taxa, capitalInicial);

        Console.ReadLine();
    }
}
```

Exercício Juros

Código



```
class Program
{
    static void Main(string[] args)
    {
        int tempo = Convert.ToInt32(Console.ReadLine());
        double taxa = Convert.ToDouble(Console.ReadLine());
        double capitalInicial = Convert.ToDouble(Console.ReadLine());

        JurosCompostos jurosC = new JurosCompostos(tempo, taxa, capitalInicial);

        Console.WriteLine("O valor do juros composto é :" + jurosC.CalcularJuros());

        JurosSimples jurosS = new JurosSimples(tempo, taxa, capitalInicial);

        Console.WriteLine("O valor do juros simples é :" + jurosS.CalcularJuros());

        Juros juros = new Juros();

        Console.ReadLine();
    }
}
```

Para que eu vou querer instanciar um objeto de Juros?

Faz algum sentido para o nosso problema instanciar um objeto de Juros?

Classes abstratas



- Há algumas classes que não precisam ser instanciadas... E, de fato, não faz nenhum sentido que elas sejam.
- Quando estamos lidando com este tipo de situação, é comum utilizarmos o que é chamado de **classe abstrata**.

Uma classe abstrata é como uma classe normal. Você define seus atributos/propriedades e métodos, define a visibilidade dos membros, define se ela vai ser uma classe derivada ou super classe, etc. Tudo exatamente igual como andamos fazendo.



Classes abstratas



- A maior diferença entre uma classe comum (concreta) e uma abstrata é que você não pode usar o operador **new** para criar uma instância dela.
- Se você fizer isso, o compilador mostrará a você um erro e impedirá a compilação.

Cannot create an instance of
the abstract class ClassName

Simplificando... Uma classe
abstrata não pode ser instanciada!

Classes abstratas



- Para tornar uma classe **abstrata**, basta utilizar a palavra reservada `abstract` antes da declaração da classe.

```
abstract class Juros
{
    private int tempo;
    private double taxa;
    private double capitalInicial;

    public Juros(int tp, double tx, double cap)
    {
        SetTempo(tp);
        SetTaxa(tx);
        SetCapitalInicial(cap);
    }

    • Métodos setters e getters
    •
    •
```

Classes abstratas

- Para tornar uma classe **abstrata**, basta utilizar a palavra reservada `abstract` antes da declaração da classe.

```
abstract class Juros
{
    private int tempo;
    private double taxa;
    private double capitalInicial;
```

```
    public Juros(int tp, double tx, double cap)
    {
        SetTempo(tp);
        SetTaxa(tx);
        SetCapitalInicial(cap);
    }
```

• Métodos setters e getters

De acordo com a **UML**, uma classe abstrata é denotada pelo seu nome em itálico.

Juros

taxa
capital
tempo

Juros()

Classes abstratas



```
class Program
{
    static void Main(string[] args)
    {
        int tempo = Convert.ToInt32(Console.ReadLine());
        double taxa = Convert.ToDouble(Console.ReadLine());
        double capitalInicial = Convert.ToDouble(Console.ReadLine());

        JurosCompostos jurosC = new JurosCompostos(tempo, taxa, capitalInicial);

        Console.WriteLine("O valor do juros composto é :" + jurosC.CalcularJuros());

        JurosSimples jurosS = new JurosSimples(tempo, taxa, capitalInicial);

        Console.WriteLine("O valor do juros simples é :" + jurosS.CalcularJuros());

        Juros juros = new Juros(tempo, taxa, capitalInicial);

        Console.ReadLine();
    }
}
```

Classes abstratas



```
class Program
{
    static void Main(string[] args)
    {
        int tempo = Convert.ToInt32(Console.ReadLine());
        double taxa = Convert.ToDouble(Console.ReadLine());
        double capitalInicial = Convert.ToDouble(Console.ReadLine());

        JurosCompostos jurosC = new JurosCompostos(tempo, taxa, capitalInicial);

        Console.WriteLine("O valor do juros composto é :" + jurosC.CalcularJuros());

        JurosSimples jurosS = new JurosSimples(tempo, taxa, capitalInicial);

        Console.WriteLine("O valor do juros simples é :" + jurosS.CalcularJuros());

        Juros juros = new Juros(tempo, taxa, capitalInicial);

        Console.ReadLine();
    }
}
```

Cannot create an instance of
the abstract class Juros

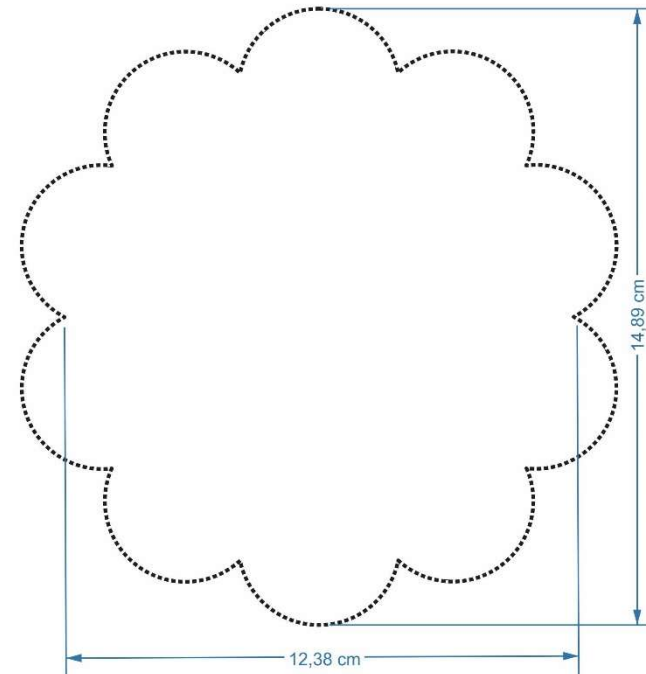
Erro de compilação

Uma classe que eu não posso instanciar?

- Porque você quer deixar algum código já disponível, mas ainda quer que as subclasses implementem o restante do código de modo a “especializar” o que já foi codificado.

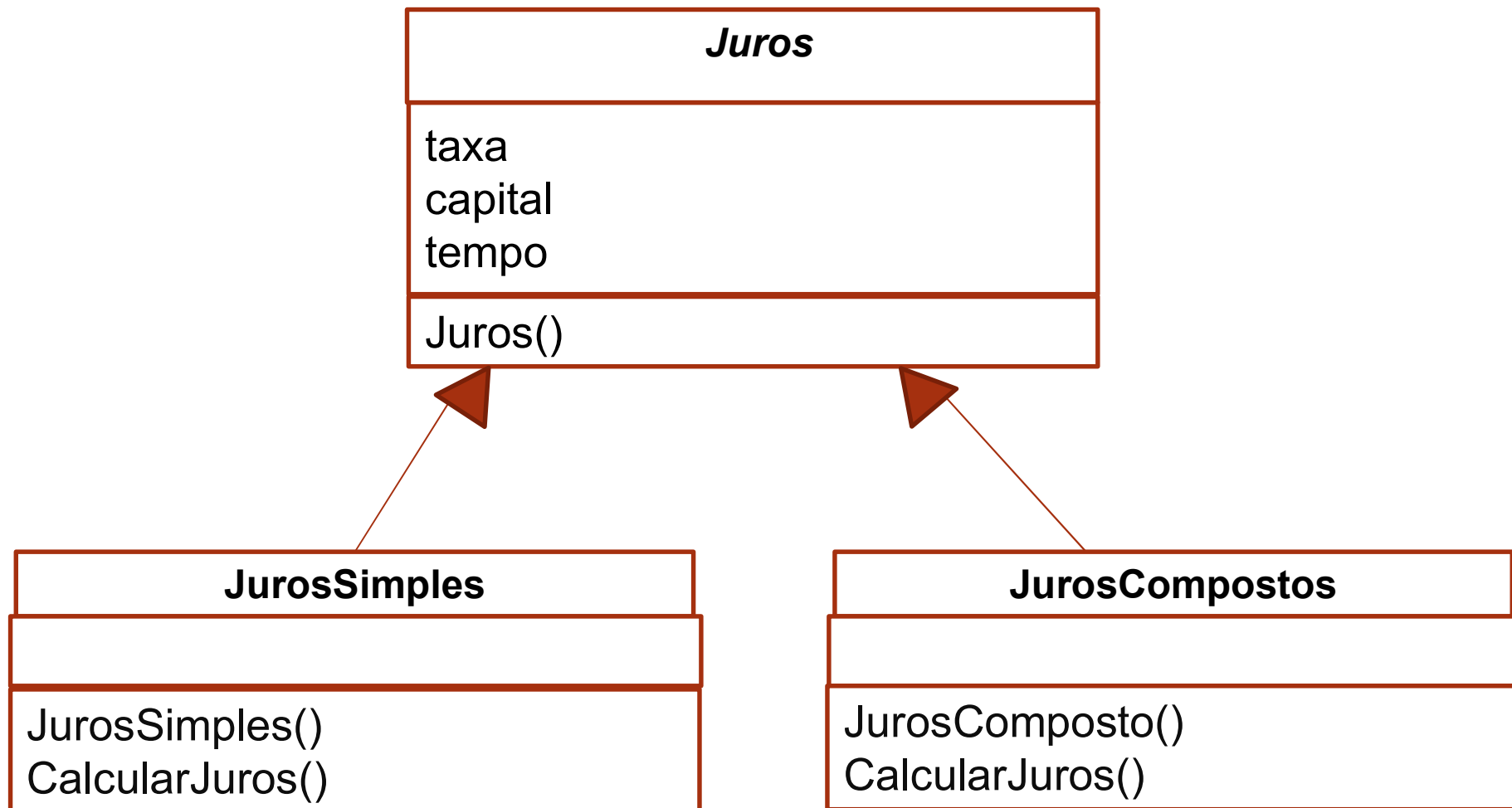
Classes abstratas servem como um **modelo** para as classes filhas!!

Isso fica mais claro de entender quando aprendemos a definição de **métodos abstratos**.



Exercício Juros

Diagrama de classe (UML)

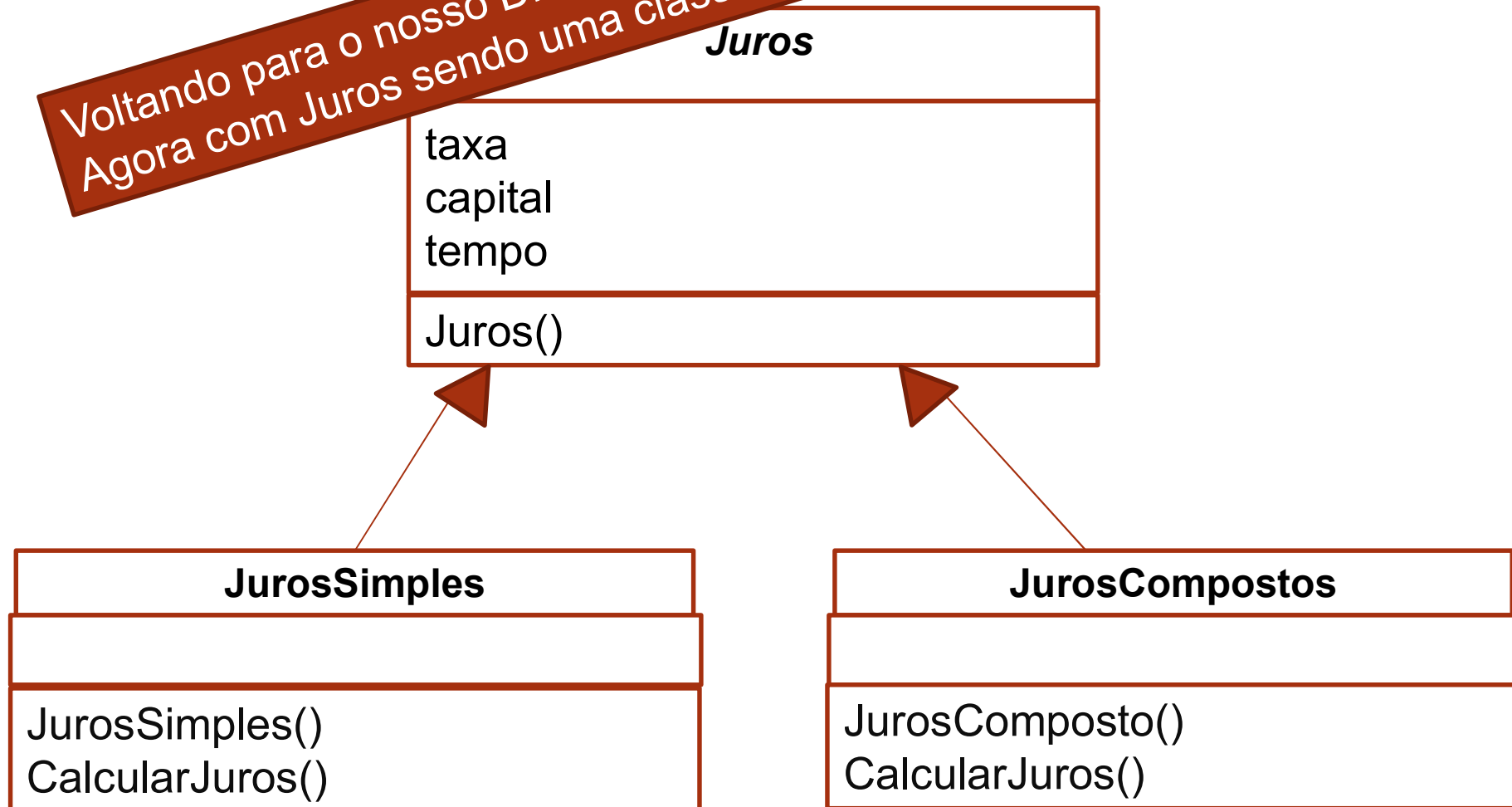


Exercício Juros

Diagrama de classe (UML)

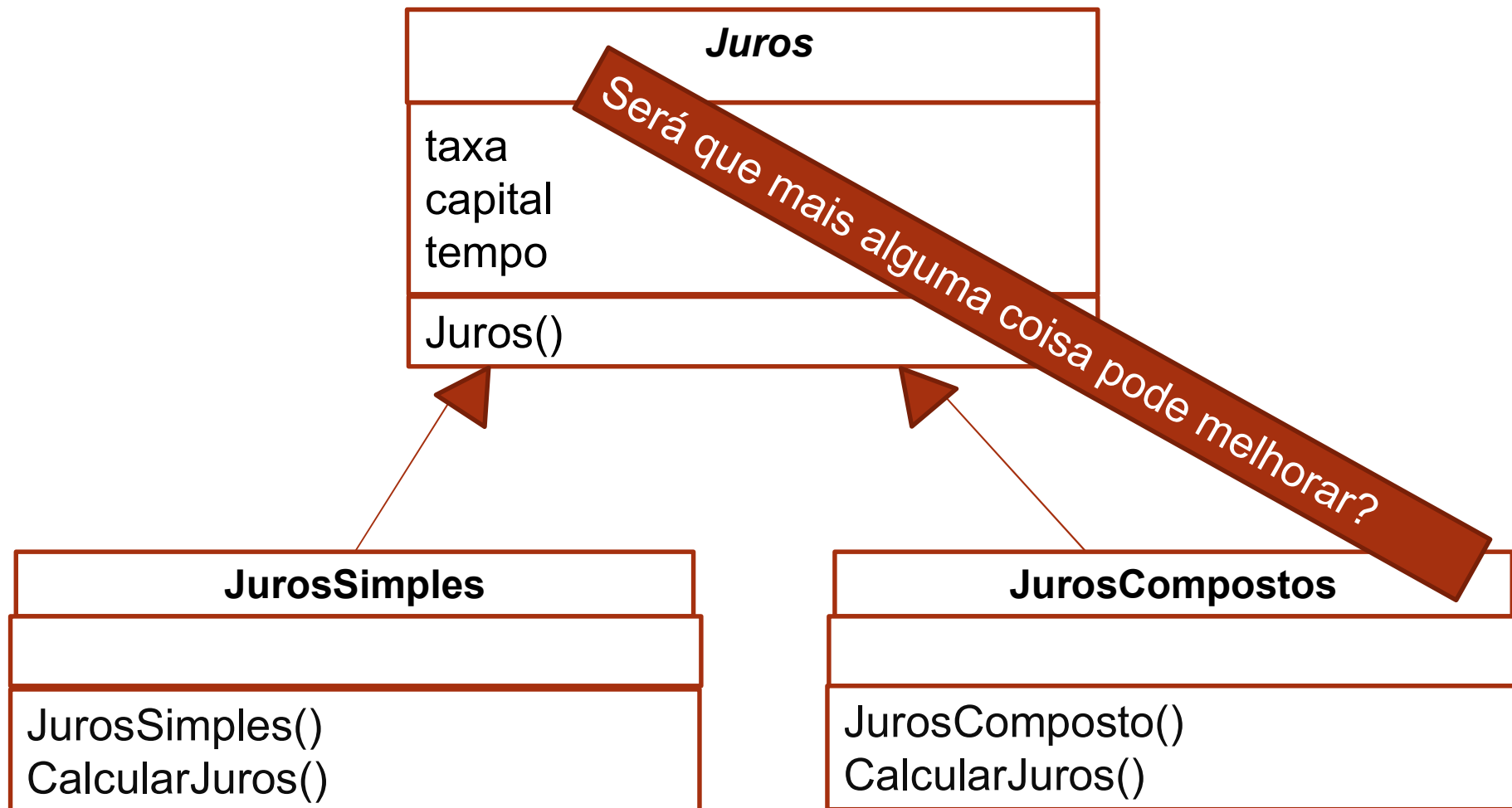


Voltando para o nosso Diagrama de Classe...
Agora com Juros sendo uma classe abstrata.



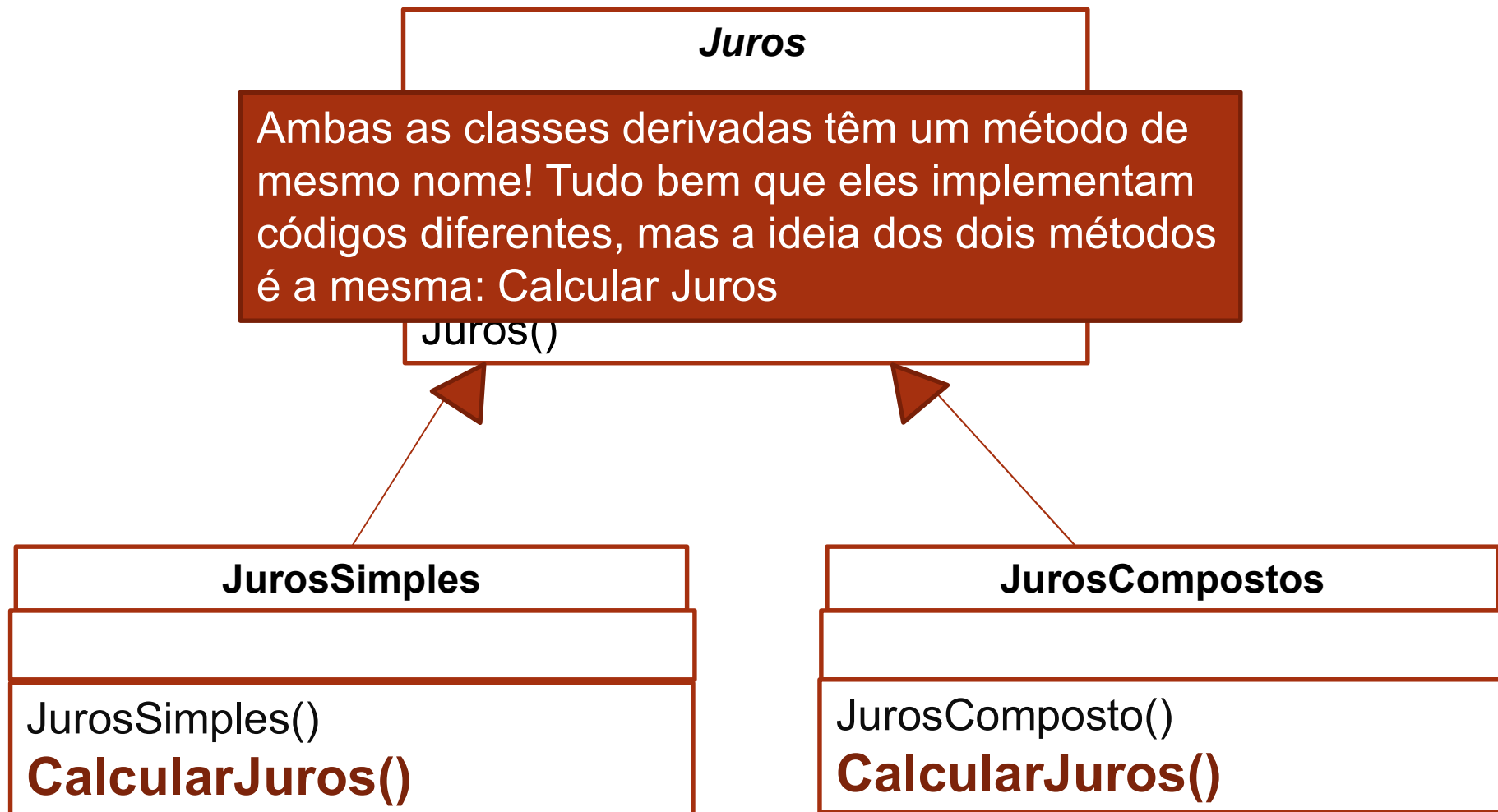
Exercício Juros

Diagrama de classe (UML)



Exercício Juros

Diagrama de classe (UML)



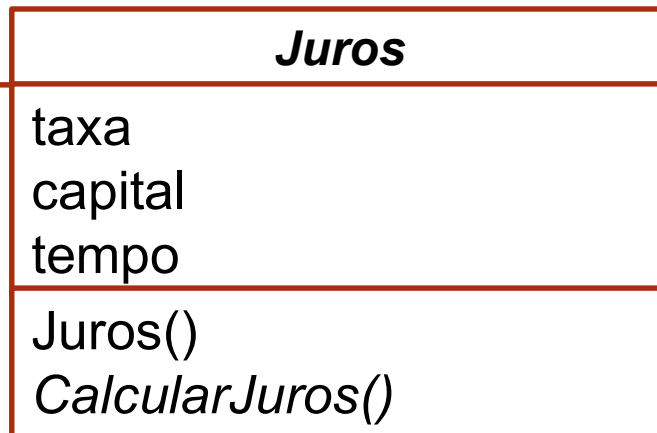
Método abstrato

- Métodos abstratos são métodos sem implementação. Eles não têm escopo.
- Esses métodos são utilizados apenas para sinalizar que PRECISAM ser implementados numa classe filha!

```
abstract class Juros
{
    private int tempo;
    private double taxa;
    private double capitalInicial;

    public Juros(int tp, double tx, double cap)
    {
        SetTempo(tp);
        SetTaxa(tx);
        SetCapitalInicial(cap);
    }

    public abstract double CalcularJuros();
    :
}
```



Método abstrato



- Quando você marca um método como abstrato, o C# te obriga a declarar a classe também como abstrata.
- Sendo assim, se uma classe tem um método abstrato ela necessariamente será abstrata.

O que faz total sentido! Imagine instanciar um objeto de uma classe que tem um método abstrato (um método sem implementação). Certamente daria problema se esse método fosse chamado, não é?

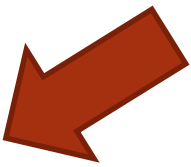
Método abstrato

- Declarando um método abstrato, estamos indicando que se uma classe for estender/derivar a classe Juros, então essa classe derivada, se quiser ser concreta, precisará implementar o método CalcularJuros() de alguma maneira.

```
abstract class Juros
{
    private int tempo;
    private double taxa;
    private double capitalInicial;

    public Juros(int tp, double tx, double cap)
    {
        SetTempo(tp);
        SetTaxa(tx);
        SetCapitalInicial(cap);
    }

    public abstract double CalcularJuros();
    :
}
```



Exercício Juros

Código



```
class JurosCompostos : Juros
{
    public JurosCompostos(int tp, double tx, double cap) :
        base(tp, tx, cap)
    {
        // Não preciso me preocupar por aqui já que
        // no construtor da minha classe pai a inicialização dos
        // meus atributos irá acontecer.
    }

    public override double CalcularJuros()
    {
        return Math.Pow(GetCapitalInicial() * (1 + GetTaxa()),
            GetTempo()) - GetCapitalInicial();
    }
}
```

Classe correspondente
ao diagrama

JurosCompostos

JurosComposto()
CalcularJuros()

Exercício Juros

Código



```
class JurosSimples : Juros
{
    public JurosSimples(int tp, double tx, double cap) :
        base(tp, tx, cap)
    {
        // Não preciso me preocupar por aqui já que
        // no construtor da minha classe pai a inicialização
        // dos meus atributos irá acontecer.
    }

    public override double CalcularJuros()
    {
        return GetCapitalInicial() * GetTaxa() * GetTempo();
    }
}
```

Classe correspondente
ao diagrama

JurosSimples

JurosSimples()
CalcularJuros()

Método abstrato

- Para indicar ao compilador que você irá sobrescrever/sobrepôr/implementar o método abstrato da classe pai, acrescente um método idêntico na subclasse e use a palavra reservada **override**.

```
class JurosSimples : Juros
{
    public JurosSimples(int tp, double tx, double cap) : base(tp, tx, cap)
    {
        // Não preciso me preocupar por aqui já que
        // no construtor da minha classe pai a inicialização // dos meus atributos irá
        acontecer.
    }

    public override double CalcularJuros()
    {
        return GetCapitalInicial() * GetTaxa() * GetTempo();
    }
}
```

Estamos dizendo pro compilador que iremos implementar o método abstrato, ou melhor, **sobrescrever** o método abstrato.

Outra solução para o problema dos Juros

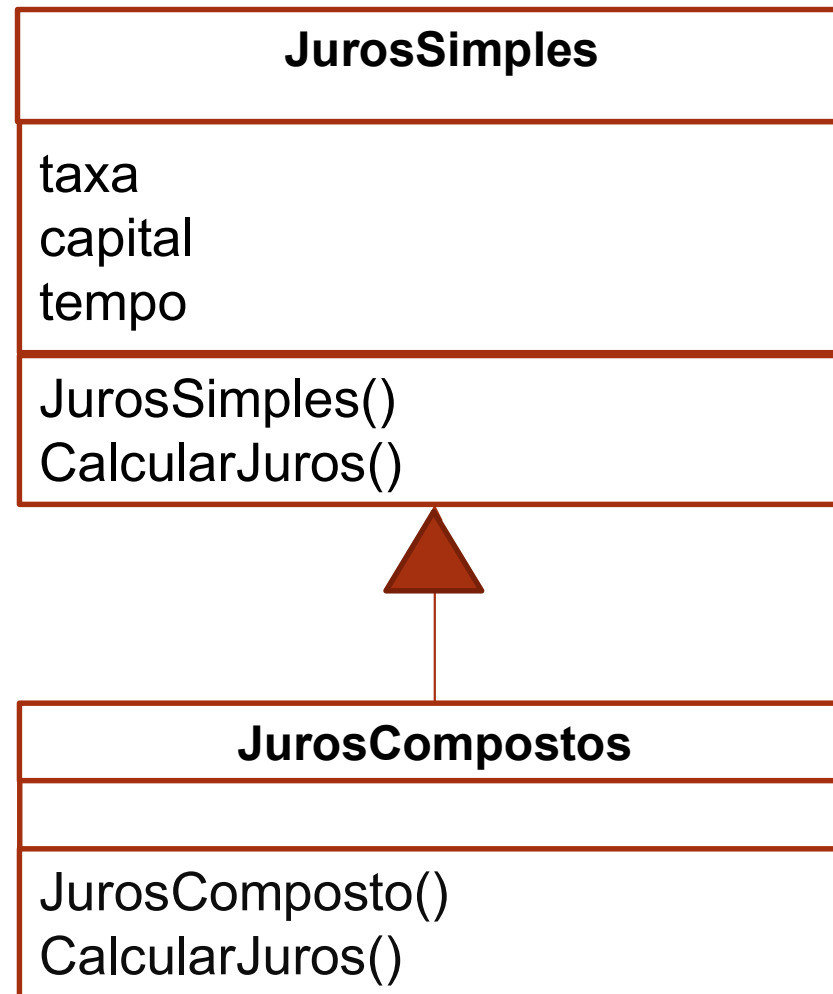


- Alguns alunos resolveram o exercício implementando o seguinte diagrama de classes.



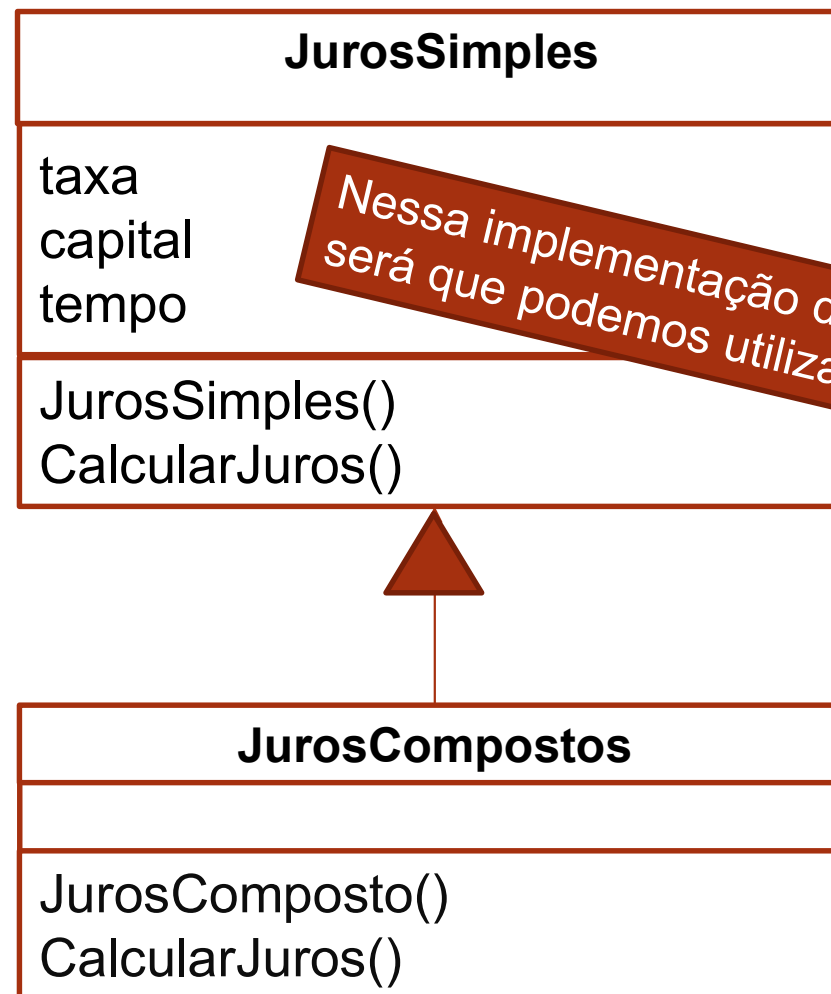
Exercício Juros

Diagrama de classe (UML)



Exercício Juros

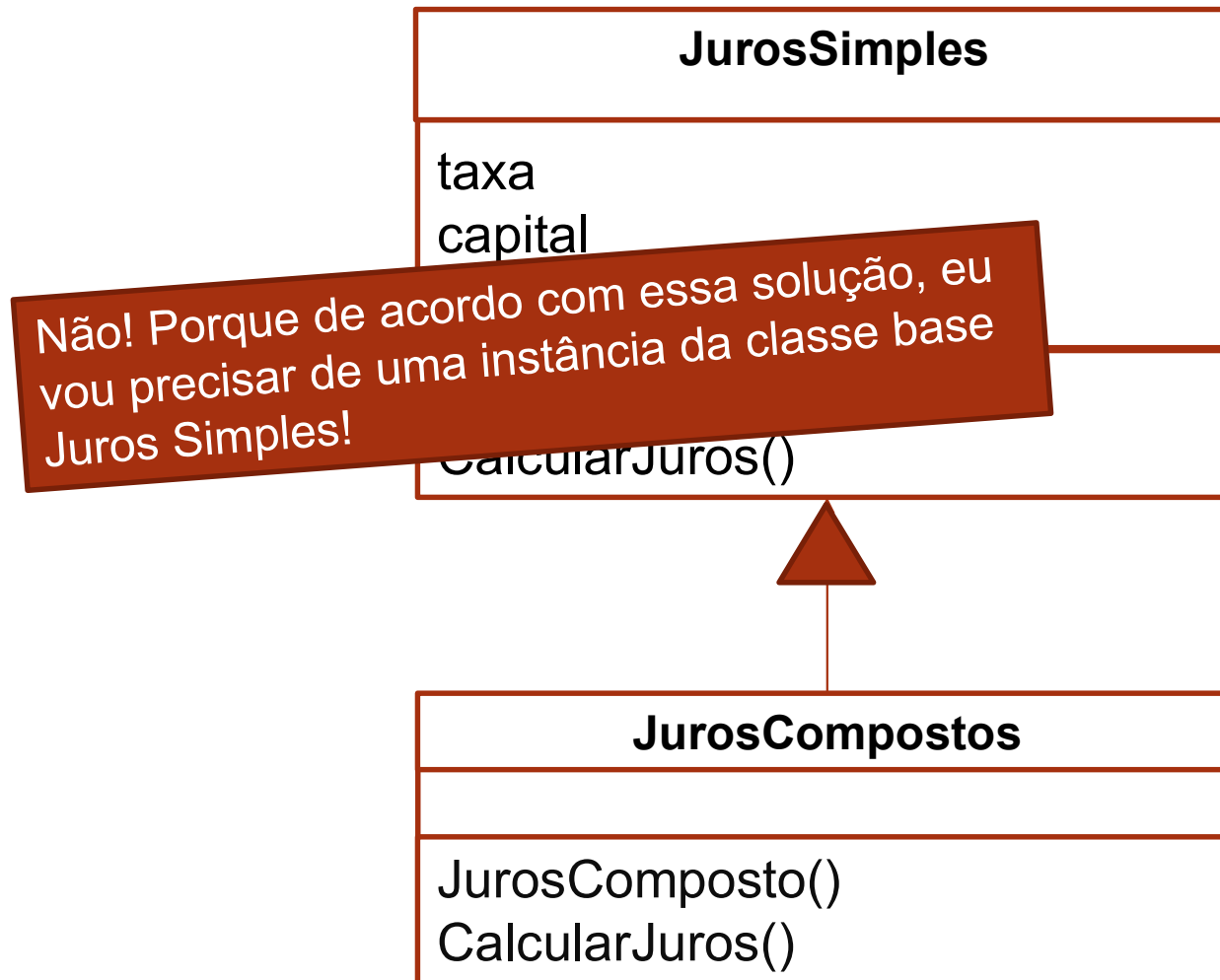
Diagrama de classe (UML)



Nessa implementação do exercício proposta, será que podemos utilizar classes abstratas?

Exercício Juros

Diagrama de classe (UML)



Exercício Juros

Código

```
class JurosSimples
{
    private int tempo;
    private double taxa;
    private double capitalInicial;

    public JurosSimples(int tp, double tx, double cap)
    {
        SetTempo(tp);
        SetTaxa(tx);
        SetCapitalInicial(cap);
    }

    public double CalcularJuros()
    {
        return tempo * taxa * capitalInicial;
    }
}
```

JurosSimples

taxa
capital
tempo

JurosSimples()
CalcularJuros()



Exercício Juros

Código



```
class JurosCompostos : JurosSimples
{
    public JurosCompostos(int tp, double tx, double cap) : base(tp, tx, cap)
    {

    }

    public double CalcularJuros()
    {
        return GetCapitalInicial() * Math.Pow( (1 + GetTaxa()), GetTempo()) -
            GetCapitalInicial();
    }
}
```

JurosCompostos

JurosComposto()
CalcularJuros()

Exercício Juros

Código



```
class Program
{
    static void Main(string[] args)
    {
        int tempo = Convert.ToInt32(Console.ReadLine());
        double taxa = Convert.ToDouble(Console.ReadLine());
        double capitalInicial = Convert.ToDouble(Console.ReadLine());

        JurosCompostos jurosC = new JurosCompostos(tempo, taxa, capitalInicial);

        Console.WriteLine("O valor do juros composto é :" + jurosC.CalcularJuros());

        JurosSimples jurosS = new JurosSimples(tempo, taxa, capitalInicial);

        Console.WriteLine("O valor do juros simples é :" + jurosS.CalcularJuros());

        Console.ReadLine();
    }
}
```

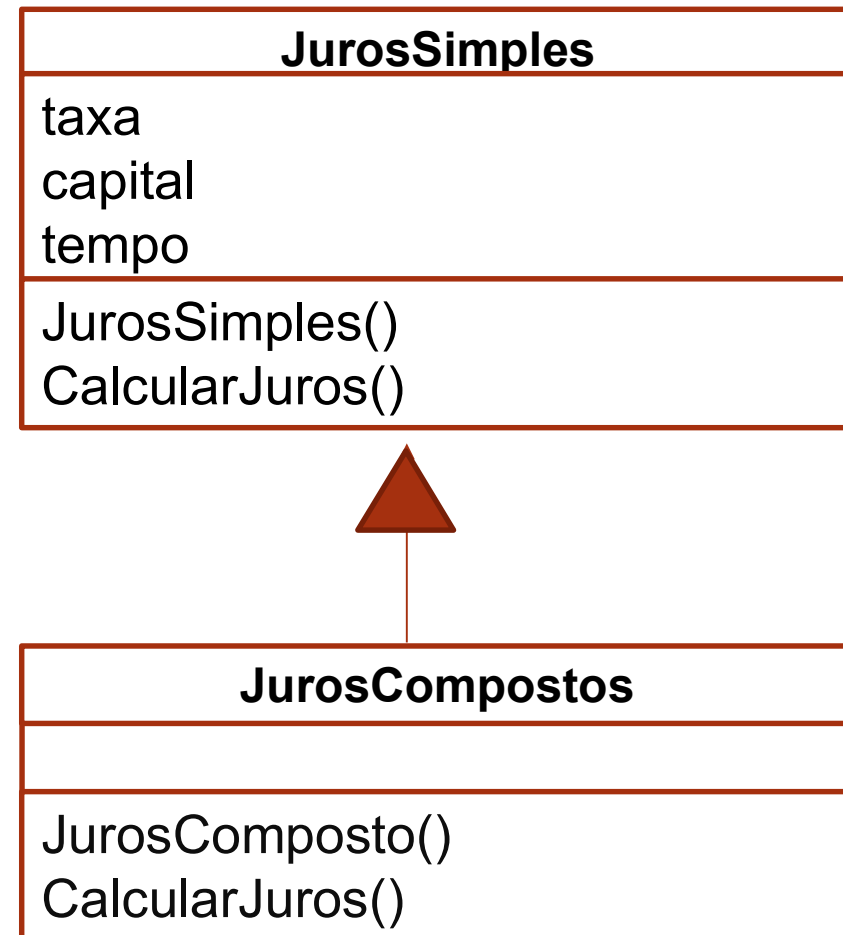
Tudo vai funcionar perfeitamente bem! Mas...

Quando há dois métodos com o mesmo nome...

Detalhe técnico



- A classe JurosCompostos irá ter dois métodos com o mesmo nome... Afinal de contas, ela herdou CalcularJuros() do pai e implementou outro método de mesmo nome.
- De qualquer maneira, o programa vai funcionar corretamente, mas é importante atentar para o que está acontecendo.

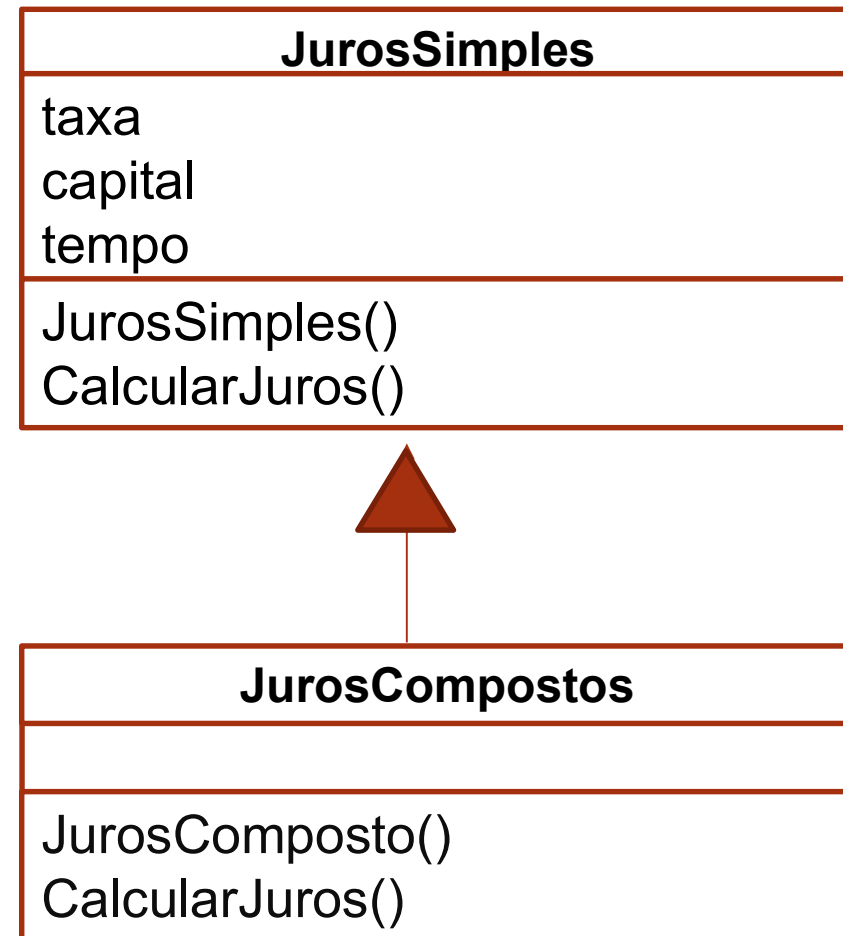


Quando há dois métodos com o mesmo nome...

Detalhe técnico



- O C# sempre chama o método mais específico. Observe o exemplo ao lado.
- Se você pedir a um objeto JurosCompostos para CalcularJuros, primeiro o C# olha dentro da classe JurosCompostos em busca desse método. Se JuroCompostos não possuir esse método, o C# irá procurar no seu pai, depois no seu “avô” e assim sucessivamente.

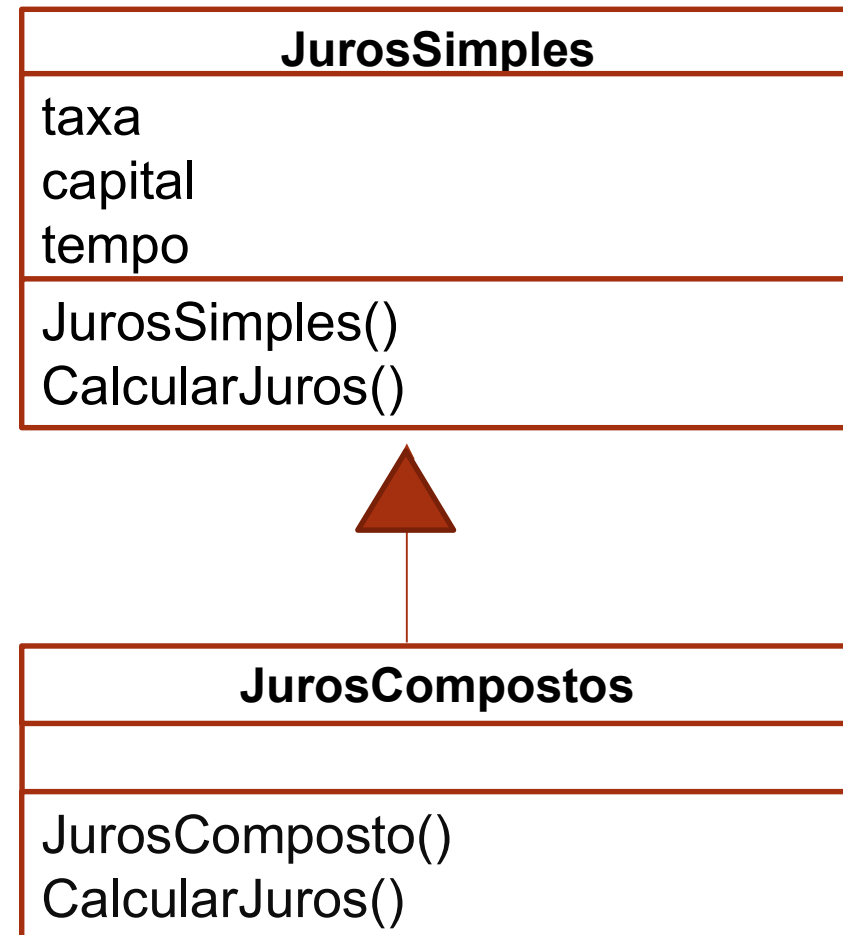


Quando há dois métodos com o mesmo nome...

Detalhe técnico



- É por conta deste motivo que tudo funciona.
- No entanto, temos DOIS métodos com o mesmo nome em JurosCompostos.
- Esse tipo de situação gera um **Warning** pois não é uma boa prática ter em uma classe com dois métodos idênticos.



Métodos Virtuais



- Quando você quer deixar claro que um método de uma classe pode ser SOBREPOSTO por um outro método de mesmo nome numa classe filha, marque esse método como virtual.

1

Adicione a palavra reservada “virtual” ao método na classe base

Uma subclasse só pode substituir um método se for marcada com a palavra reservada virtual. Isso diz ao C# que os métodos filhos desta classe podem sobrescrever o método virtual.

```
public virtual double CalcularJuros()  
{  
    return tempo * taxa * capitalInicial;  
}
```

Indica pro C# que alguma subclasse pode sobrescrever esse método.

Métodos Virtuais



2

Adicione um método com o mesmo nome na classe derivada com a palavra “override”

Você precisará ter exatamente a mesma assinatura – ou seja, o mesmo valor de retorno e parâmetros – e precisará usar a palavra reservada `override` na declaração

```
public override double CalcularJuros()  
{  
    return GetCapitalInicial() * Math.Pow( (1 + GetTaxa()), GetTempo()) -  
           GetCapitalInicial();  
}
```

Indica pro C# que alguma subclasse está sobrescrevendo esse método.

Exercício Juros

Código

```
class JurosSimples
{
    private int tempo;
    private double taxa;
    private double capitalInicial;

    public JurosSimples(int tp, double tx, double cap)
    {
        SetTempo(tp);
        SetTaxa(tx);
        SetCapitalInicial(cap);
    }

    public virtual double CalcularJuros()
    {
        return tempo * taxa * capitalInicial;
    }
}
```



JurosSimples

taxa
capital
tempo

JurosSimples()
CalcularJuros()

Exercício Juros

Código



```
class JurosCompostos : JurosSimples
{
    public JurosCompostos(int tp, double tx, double cap) : base(tp, tx, cap)
    {

    }

    public override double CalcularJuros()
    {
        return GetCapitalInicial() * Math.Pow( (1 + GetTaxa()), GetTempo()) -
            GetCapitalInicial();
    }
}
```

JurosCompostos

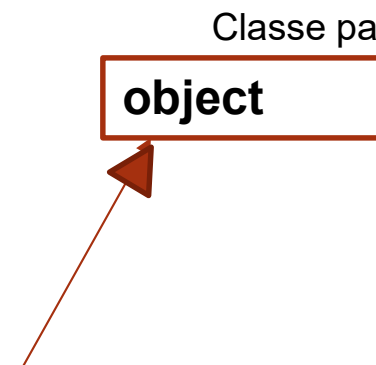
JurosComposto()
CalcularJuros()

Classe *object*

e seus métodos virtuais...



- Como discutimos em sala, qualquer classe criada é necessariamente subclasse/filha/classe derivada da classe *object*.
- Portanto, todos os métodos de *object* são herdados por qualquer classe que você venha a criar.



Toda e qualquer classe que é criada no C#

▪
▪
▪

Classe *object*

e seus métodos virtuais...

A classe *object* dá suporte a todas as classes na hierarquia de classe do .NET Framework e fornece serviços de nível baixo para classes derivadas. Esta é a classe base definitiva de todas as classes do .NET Framework. É a raiz da hierarquia de tipo.



Métodos presentes na classe *object*.

- Equals-Dá suporte a comparações entre objetos.
- Finalize-Executa operações de limpeza antes de um objeto é recuperado automaticamente.
- GetHashCode-Gera um número que corresponde ao valor do objeto para suporte ao uso de uma tabela de hash.
- ToString-Fabrica uma cadeia de caracteres de texto legível que descreve uma instância da classe.

Classe *object*

e seus métodos virtuais...

Como todas as classes do .NET Framework são derivadas da *object*, cada método definido no *object* classe está disponível em todos os objetos no sistema. Derivada pode classes e substituir alguns desses métodos, incluindo:



Esses métodos são marcados como **virtuais** e portanto podem ser **sobrescritos** na sua própria classe!

- Equals-Dá suporte a comparações entre objetos.
- Finalize-Executa operações de limpeza antes de um objeto é recuperado automaticamente.
- GetHashCode-Gera um número que corresponde ao valor do objeto para suporte ao uso de uma tabela de hash.
- ToString-Fabrica uma cadeia de caracteres de texto legível que descreve uma instância da classe.

Classe *object*

Exemplo de sobrescrita



```
class Bomberman
{
    public int vida = 10;
    public int velocidade = 5;
    public string cor = "branco";

    public Bomberman(int v, int vel, string c)
    {
        vida = v;
        velocidade = vel;
        cor = c;

        Console.WriteLine("Construindo bomberman...");
    }

    public virtual void ColocarBomba()
    {
        Console.WriteLine("Colocando Bomba!");
    }

    public override string ToString()
    {
        return "Bomberman de velocidade = " + velocidade + "; vida = " + vida + "; cor = " + cor;
    }
}
```

Sobrescrevemos o método
ToString() de object



Classe *object*

Exemplo de sobrescrita



```
class Program
{
    static void Main(string[] args)
    {
        Bomberman bomber = new Bomberman(10,5,
                                           "branco");

        Console.WriteLine(bomber.ToString());

        Console.ReadLine();
    }
}
```



Usando o método sobrescrito