



# PROGRAMAÇÃO O.O.

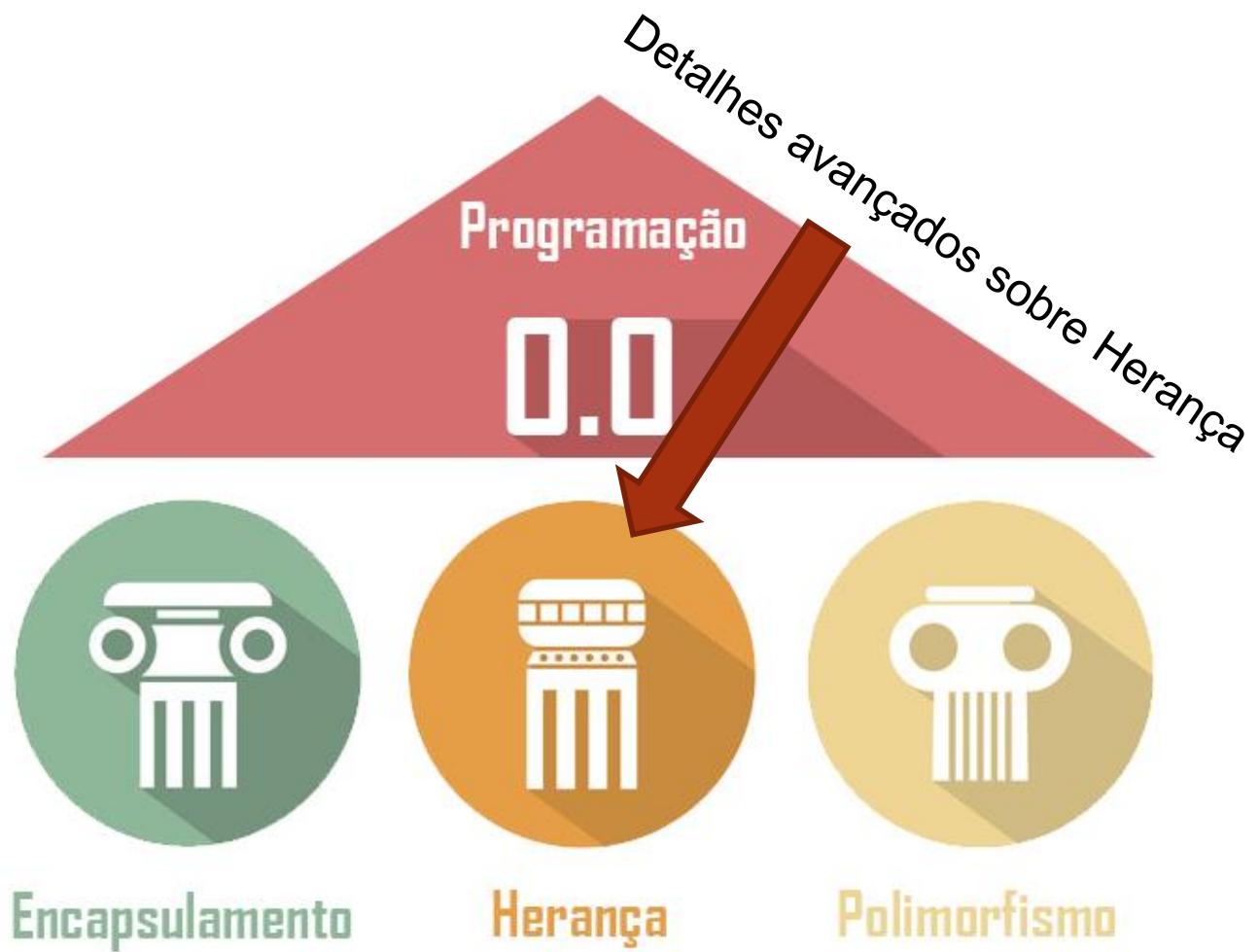
(C#)

---

{ Encapsulamento, Herança e Polimorfismo  
**Professor:** João Luiz Lagôas

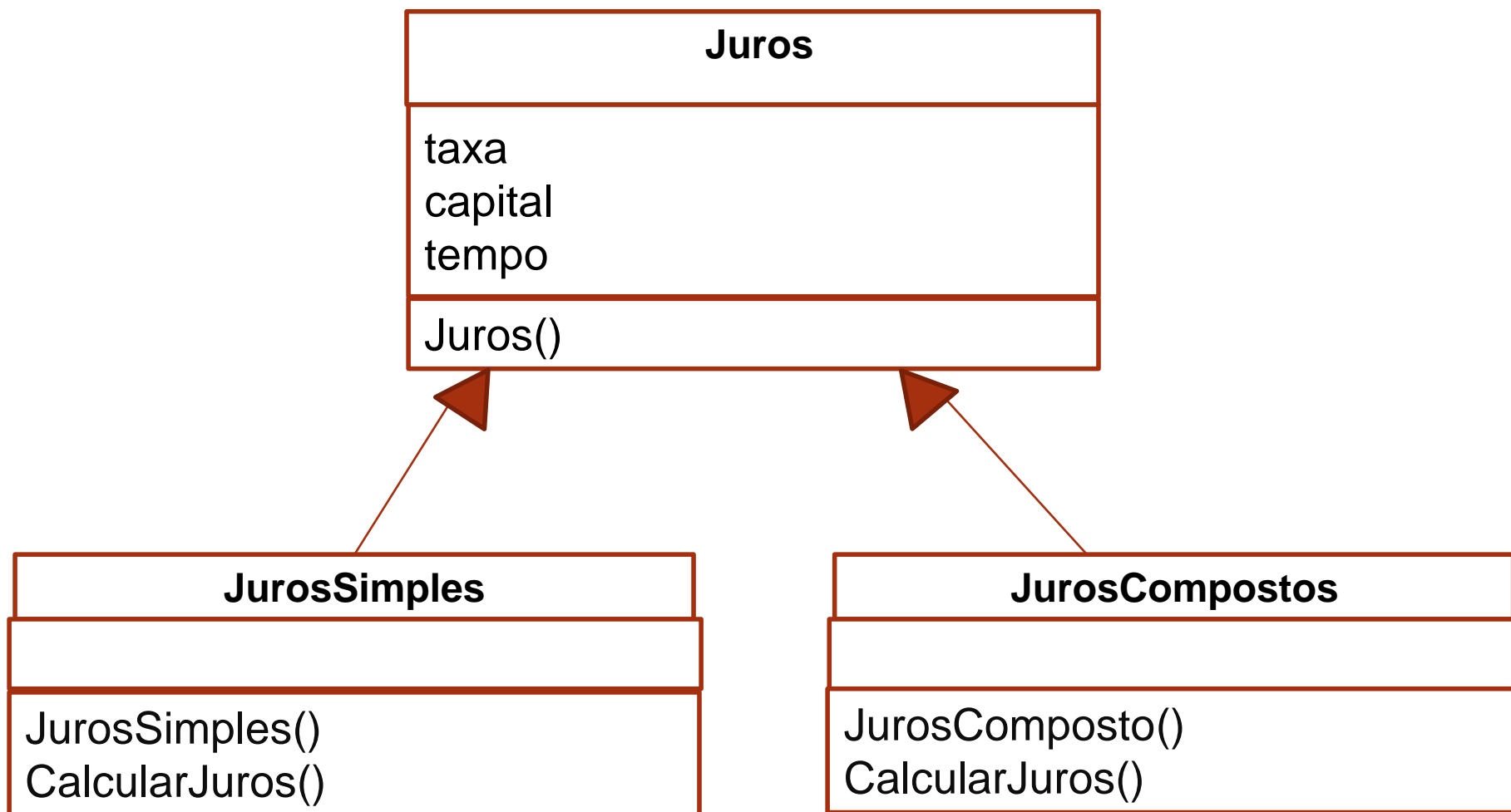
}

# Roteiro



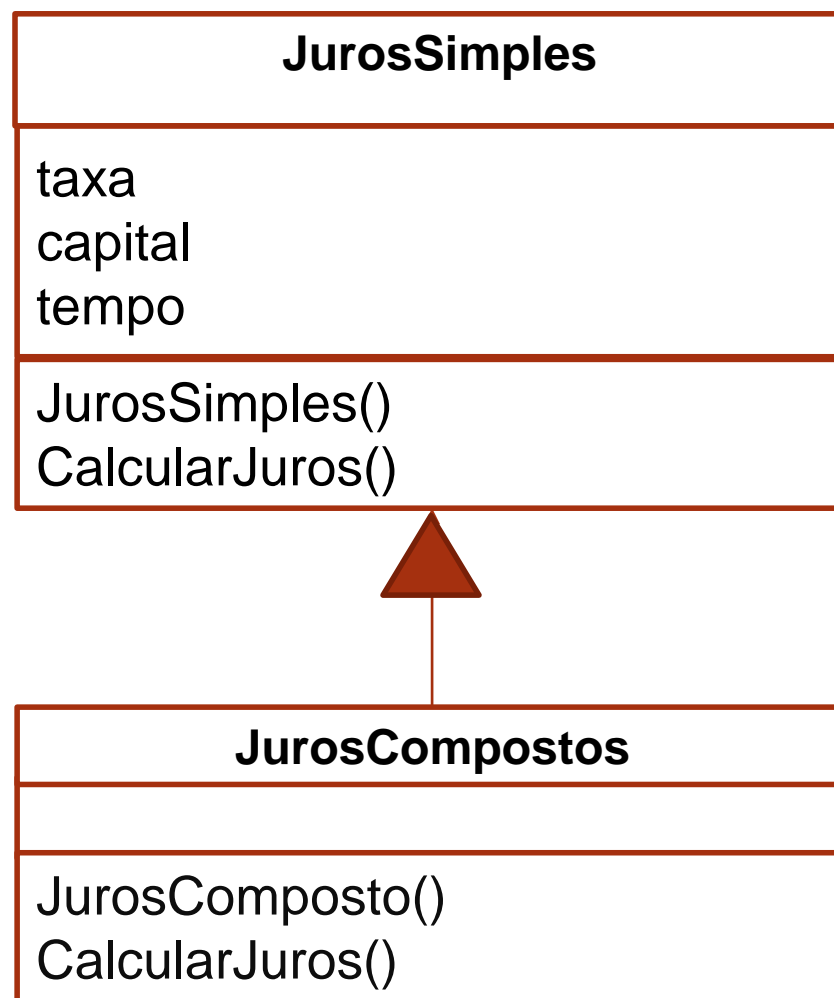
# Exercício Juros

## Diagrama de classe (UML)



# Exercício Juros

## Diagrama de classe (UML)



# 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.

Classe pai

**object**

▪

▪

▪

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*.

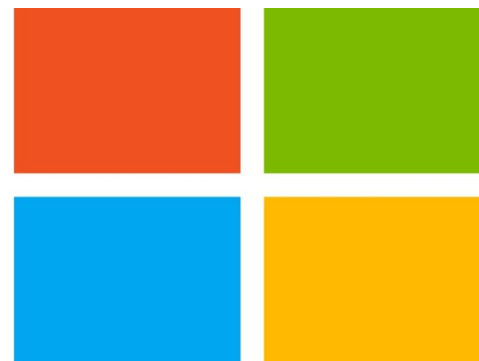
- Equals-Dá suporte a comparações entre objetos.
- Finalize-Executa operações de limpeza antes de um objeto é recuperado automaticamente.
- GetHashCode-Gera um número que corresponde ao valor do objeto para suporte ao uso de uma tabela de hash.
- ToString-Fabrica uma cadeia de caracteres de texto legível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



Colégio  
**Pedro II**

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

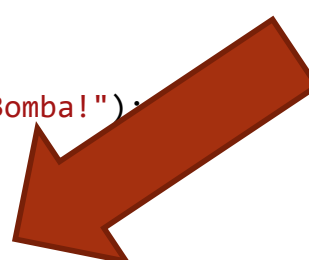
    public Bomberman(int v, int vel, string c)
    {
        vida = v;
        velocidade = vel;
        cor = c;

        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*

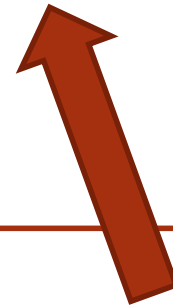


Colégio  
**Pedro II**

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

        Console.WriteLine(bomber.ToString());

        Console.ReadLine();
    }
}
```



Usando o método sobrescrito

# Modificadores de Acesso

Outros modificadores...



- Você já sabe como a palavra `private` é importante, como usá-la e como ela difere de `public`.
- Já sabemos que esses modificadores de acesso modificam a forma como outras classes podem acessar os atributos e métodos de outra.
- Existem no entanto outro modificador de acesso que você pode usar.

# Modificadores de Acesso

public



- Quando você marca uma classe ou um membro com **public** está dizendo ao C# que qualquer membro de outra classe pode acessá-lo.
- É o menos restritivo dos modificadores de acesso.

Qualquer um pode acessar o  
atributo ou método marcado como  
**público**



# Modificadores de Acesso

private



- Quando você marca um membro de uma classe como **private** ele só poderá ser acessado por outros membros dentro daquela classe.
- É o mais restritivo dos modificadores de acesso

Apenas dentro da classe os atributos ou métodos marcados como **privados** são acessíveis



# Modificadores de Acesso

protected



- Você já viu como uma subclasse está impossibilitada de acessar os membros privados de uma classe pai.
- Não seria conveniente se as subclasses pudessem acessar membros privados de sua classe pai? É para isso que serve o modificador de acesso **protected**.

# Modificadores de Acesso

protected



- Qualquer atributo ou método de classe marcado como **protected** pode ser acessado por qualquer outro membro da mesma classe e por qualquer membro de uma subclasse.
- É uma espécie de restrição intermediária entre o modificador público e privado.

Atributos e métodos protegidos são visíveis dentro da sua classe E dentro da classe de suas classes filhas.



# Exercício Juros

## Código



### JurosSimples

taxa  
capital  
tempo

JurosSimples()  
CalcularJuros()

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

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

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

# Exercício Juros

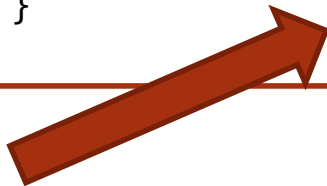
## Código



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

    }

    public override double CalcularJuros()
    {
        return capitalInicial * Math.Pow((1 + taxa), tempo) - capitalInicial;
    }
}
```



Há acesso a esses atributos na classe filha!

### JurosCompostos

JurosComposto()  
CalcularJuros()



# Exercício Juros

## Código



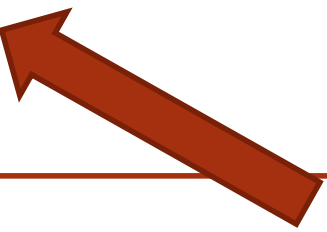
```
class Program
{
    static void Main(string[] args)
    {
        int tempo = Convert.ToInt32(Console.ReadLine());
        double taxa = Convert.ToDouble(Console.ReadLine());
        double capitalInicial = Convert.ToDouble(Console.ReadLine());

        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.taxa = 0.5;

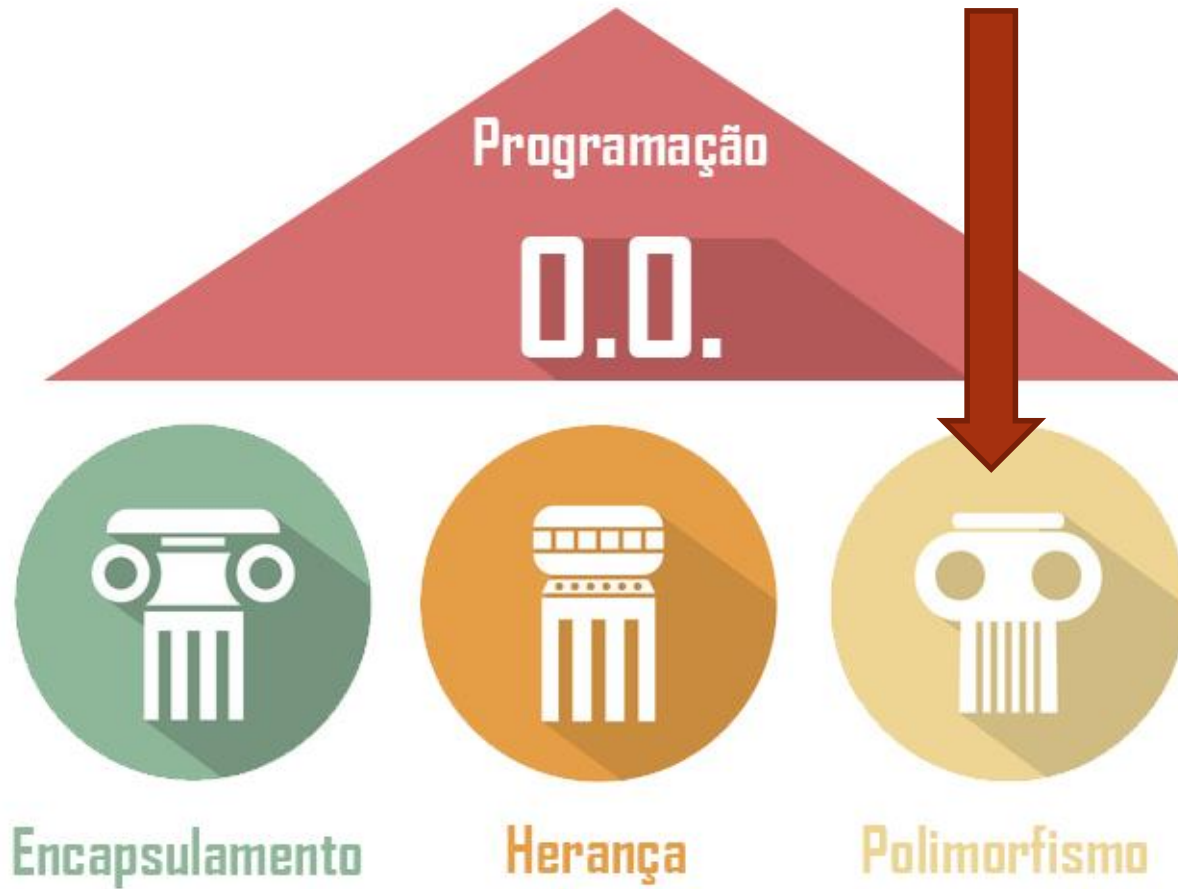
        Console.ReadLine();
    }
}
```



Mas fora da classe não há acesso! Como se fosse privado.

# Roteiro

O terceiro e último pilar da P.O.O.



# Tipos de polimorfismo



- Polimorfismo de Sobrecarga
  - Polimorfismo estático
  - Overloading
  - Binding estático (early binding)
  - Ocorre em tempo de compilação



- Polimorfismo de Sobrescrita
  - Polimorfismo dinâmico
  - Overriding
  - Binding dinâmico (late binding)
  - Ocorre em tempo de execução

# Tipos de polimorfismo



- Polimorfismo de Sobrecarga
  - Polimorfismo estático
  - Overloading
  - Binding estático (early binding)
  - Ocorre em tempo de compilação



**“o verdadeiro polimorfismo”**

- Polimorfismo de Sobrescrita
  - Polimorfismo dinâmico
  - Overriding
  - Binding dinâmico (late binding)
  - Ocorre em tempo de execução



# Polimorfismo de Sobrecarga



- Acontece quando para um método de mesmo nome, criamos comportamentos diferentes dependendo da sua lista de parâmetros.
- Exemplo: vários métodos com parâmetros diferentes configuram polimorfismo estático.

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

    public JurosSimples()
    {
    }

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

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

# Polimorfismo de Sobrecarga




```
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());

        JurosSimples j1 = new JurosSimples();

        JurosSimples j2 = new JurosSimples(tempo, taxa, capitalInicial);

        Console.ReadLine();
    }
}
```



As duas chamadas ao método  
são válidas já que há duas  
implementações na classe.

# Polimorfismo de Sobrecarga

## Regras



- Os métodos sobrecarregados DEVEM alterar a lista de argumentos;
- Os métodos sobrecarregados PODEM alterar o tipo de retorno.
- Os métodos sobrecarregados PODEM alterar os modificadores de acesso;

# Polimorfismo de Sobrescrita



Colégio  
Pedro II

- Além do reuso de código, outro ponto positivo da herança é a capacidade de usar uma subclasse no lugar da classe base da qual ela herda.
- Em qualquer lugar onde se pode usar uma classe base, se pode usar uma de suas subclasses.

Uma variável do tipo classe base PODE armazenar um objeto do tipo subclasse.

- É através desse mecanismo que se implementa polimorfismo de sobrescrita.



# Polimorfismo de Sobreescrita



Colégio  
Pedro II

```
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!");
    }
}
```



# Polimorfismo de Sobreescrita



Colégio  
**Pedro II**

```
class FireBomber : Bomberman
{
    public FireBomber(int v, int vel, string c):base(v, vel, c)
    {
        Console.WriteLine("Construindo Firebomber...");
    }

    public override void ColocarBomba()
    {
        Console.WriteLine("Colocando bomba de Fogo!");
    }

    public void Explodir()
    {
        Console.WriteLine("Explodindo!");
    }
}
```



# Polimorfismo de Sobreescrita



Colégio  
Pedro II

```
class AquaBomber : Bomberman
{
    public AquaBomber(int v, int vel, string c):base(v, vel, c)
    {
        Console.WriteLine("Construindo AquaBomber...");
    }

    public override void ColocarBomba()
    {
        Console.WriteLine("Colocando bomba de água!");
    }

    public void Nadar()
    {
        Console.WriteLine("Nadando!");
    }
}
```



# Polimorfismo de Sobreescrita



Colégio  
Pedro II

Observe a atribuição de uma variável do tipo FireBomber e AquaBomber para outra do tipo Bomberman.

```
class Program
{
    static void Main(string[] args)
    {
        Bomberman bomber;

        FireBomber fireBomber = new FireBomber(20, 2, "vermelho");

        AquaBomber aquaBomber = new AquaBomber(10, 6, "azul");

        bomber = fireBomber;

        bomber.ColocarBomba();

        bomber = aquaBomber;

        bomber.ColocarBomba();

        Console.ReadLine();
    }
}
```

# Polimorfismo de Sobreescrita



Colégio  
Pedro II

Observe a atribuição de uma variável do tipo FireBomber e AquaBomber para outra do tipo Bomberman.

```
class Program
{
    static void Main(string[] args)
    {
        Bomberman bomber;

        FireBomber fireBomber = new FireBomber(20, 2, "vermelho");

        AquaBomber aquaBomber = new AquaBomber(10, 6, "azul");

        bomber = fireBomber;

        bomber.ColocarBomba();

        bomber = aquaBomber;

        bomber.ColocarBomba();

        Console.ReadLine();
    }
}
```

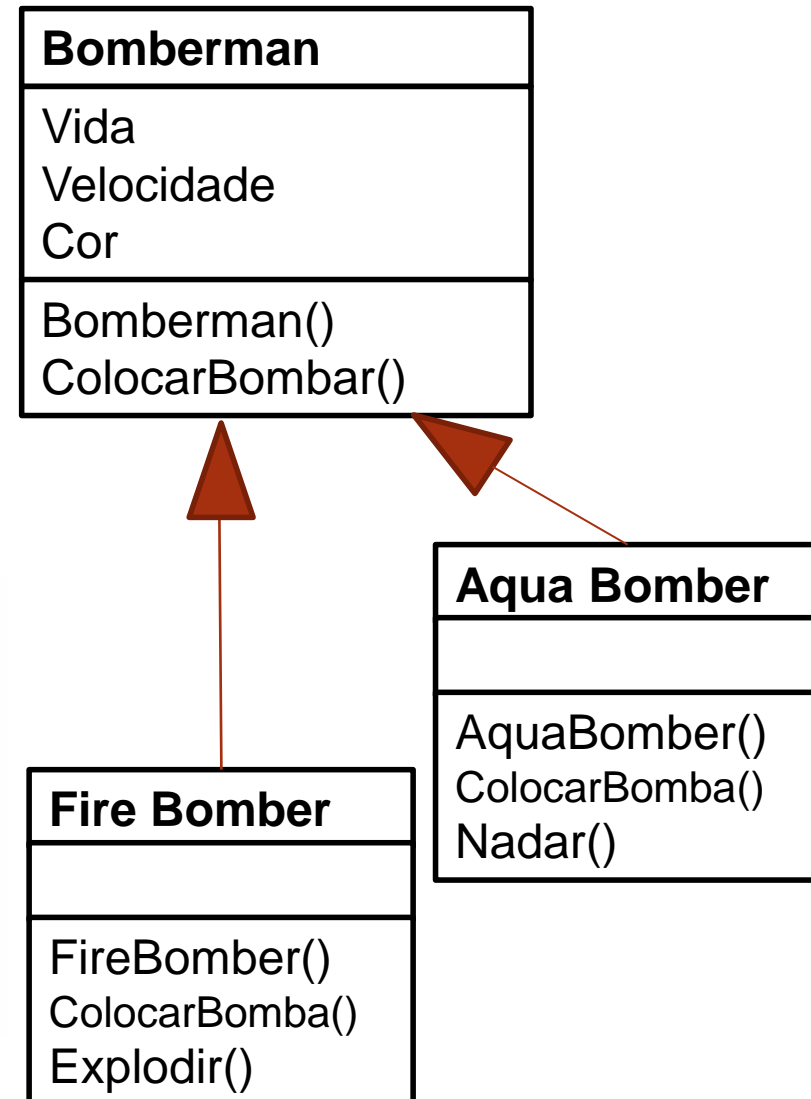
Essa atribuição é válida apesar da variável bomber ter tipo diferente das variáveis fireBomber e aquaBomber!

# Polimorfismo de Sobreescrita



Colégio  
**Pedro II**

- A herança nos diz que todo FireBomber ou AquaBomber **É UM** Bomberman, por isso é possível que uma variável Bomberman armazene uma referência para um objeto FireBomber.
- No entanto o contrário não é válido. Será que todo Bomberman **É UM** FireBomber ou AquaBomber? Definitivamente não.



# Polimorfismo de Sobreescrita



Colégio  
Pedro II

```
class Program
{
    static void Main(string[] args)
    {
        Bomberman bomber;

        FireBomber fireBomber = new FireBomber(20, 2, "vermelho");

        AquaBomber aquaBomber = new AquaBomber(10, 6, "azul");

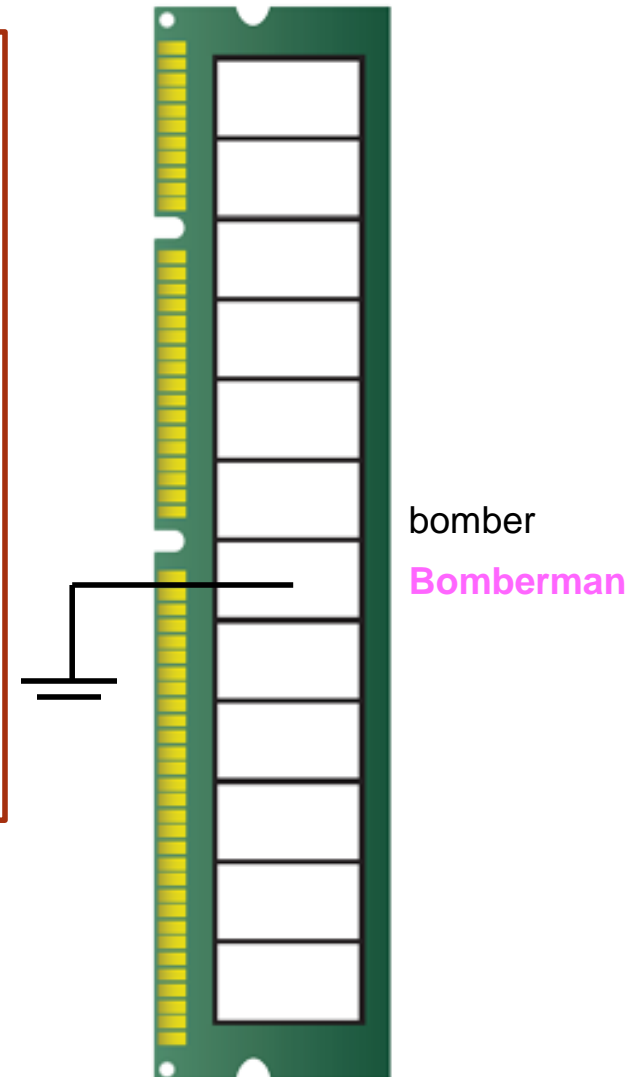
        bomber = fireBomber;

        bomber.ColocarBomba();

        bomber = aquaBomber;

        bomber.ColocarBomba();

        Console.ReadLine();
    }
}
```



# Polimorfismo de Sobreescrita



Colégio  
Pedro II

```
class Program
{
    static void Main(string[] args)
    {
        Bomberman bomber;

        FireBomber fireBomber = new FireBomber(20, 2, "vermelho");
        AquaBomber aquaBomber = new AquaBomber(10, 6, "azul");

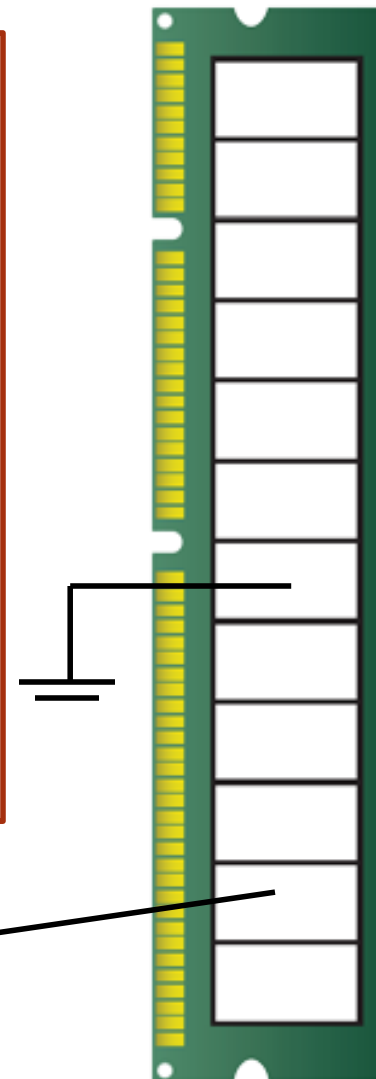
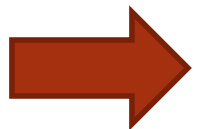
        bomber = fireBomber;

        bomber.ColocarBomba();

        bomber = aquaBomber;

        bomber.ColocarBomba();

        Console.ReadLine();
    }
}
```



bomber  
**Bomberman**

fireBomber  
**FireBomber**



# Polimorfismo de Sobreescrita



Colégio  
Pedro II

```
class Program
{
    static void Main(string[] args)
    {
        Bomberman bomber;

        FireBomber fireBomber = new FireBomber(20, 2, "vermelho");
        AquaBomber aquaBomber = new AquaBomber(10, 6, "azul");

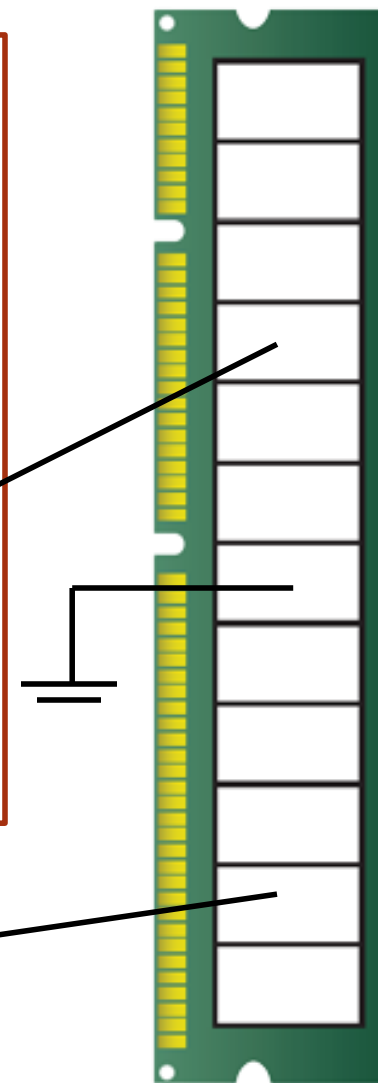
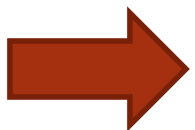
        bomber = fireBomber;

        bomber.ColocarBomba();

        bomber = aquaBomber;

        bomber.ColocarBomba();

        Console.ReadLine();
    }
}
```



aquaBomber  
AquaBomber

bomber  
Bomberman

fireBomber  
FireBomber

# Polimorfismo de Sobreescrita



Colégio  
Pedro II

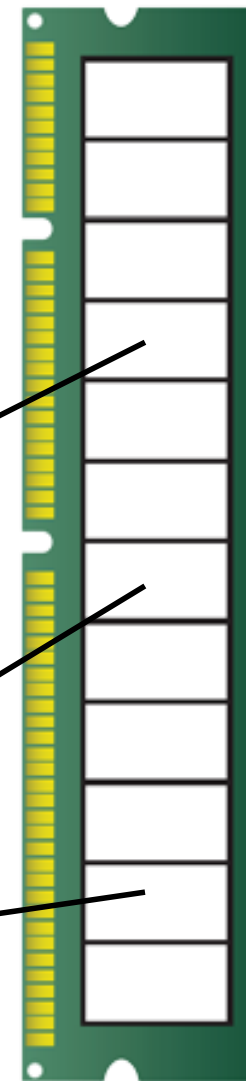
```
class Program
{
    static void Main(string[] args)
    {
        Bomberman bomber;

        FireBomber fireBomber = new FireBomber(20, 2, "vermelho");
        AquaBomber aquaBomber = new AquaBomber(10, 6, "azul");

        bomber = fireBomber;
        bomber.ColocarBomba();

        bomber = aquaBomber;
        bomber.ColocarBomba();

        Console.ReadLine();
    }
}
```



aquaBomber  
AquaBomber

bomber  
Bomberman

fireBomber  
FireBomber

# Polimorfismo de Sobreescrita



Colégio  
**Pedro II**

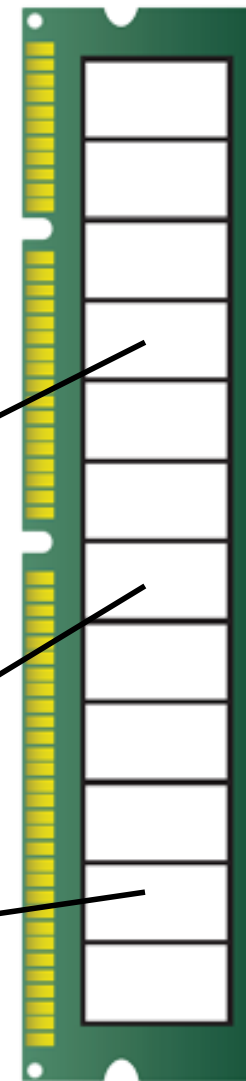
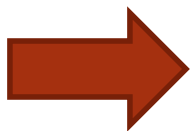
Console

## Colocando Bomba de Fogo...

```
class
{
    static void Main(string[] args)
    {
        Bomberman bomber;

        FireBomber fireBomber = new FireBomber(20, 2, "vermelho");
        AquaBomber aquaBomber = new AquaBomber(10, 6, "azul");

        bomber = fireBomber;
        bomber.ColocarBomba();
        bomber = aquaBomber;
        bomber.ColocarBomba();
        Console.ReadLine();
    }
}
```



aquaBomber  
**AquaBomber**

bomber  
**Bomberman**

fireBomber  
**FireBomber**

# Polimorfismo de Sobreescrita



Colégio  
Pedro II

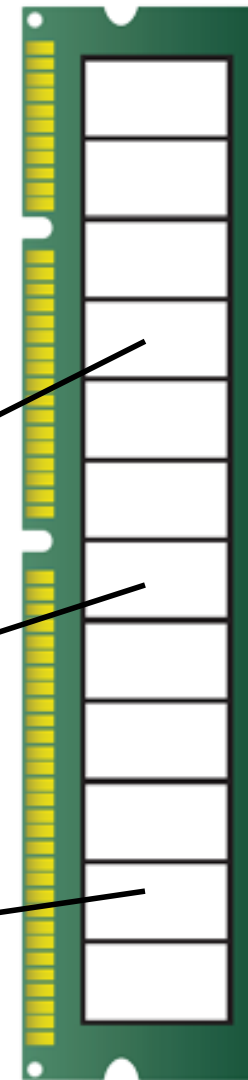
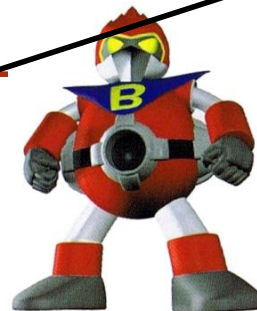
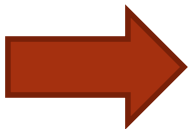
```
class Program
{
    static void Main(string[] args)
    {
        Bomberman bomber;

        FireBomber fireBomber = new FireBomber(20, 2, "vermelho");
        AquaBomber aquaBomber = new AquaBomber(10, 6, "azul");

        bomber = fireBomber;
        bomber.ColocarBomba();

        bomber = aquaBomber;
        bomber.ColocarBomba();

        Console.ReadLine();
    }
}
```



aquaBomber  
**AquaBomber**

bomber  
**Bomberman**

fireBomber  
**FireBomber**

# Polimorfismo de Sobreescrita



Colégio  
Pedro II

Console

## Colocando Bomba de Água...

```
class
{
    static void Main(string[] args)
    {
        Bomberman bomber;

        FireBomber fireBomber = new FireBomber(20, 2, "vermelho");
        AquaBomber aquaBomber = new AquaBomber(10, 6, "azul");

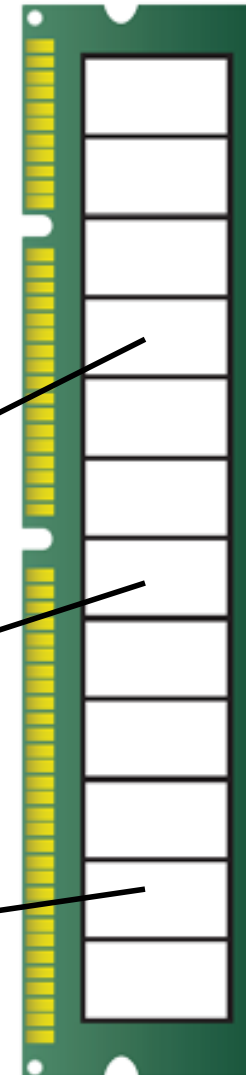
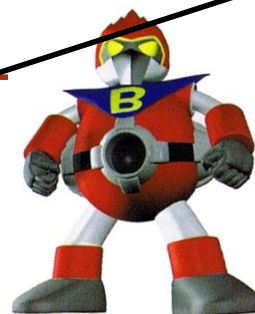
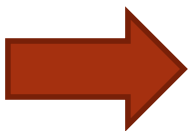
        bomber = fireBomber;

        bomber.ColocarBomba();

        bomber = aquaBomber;

        bomber.ColocarBomba();

        Console.ReadLine();
    }
}
```



aquaBomber  
AquaBomber

bomber  
Bomberman

fireBomber  
FireBomber

# Polimorfismo de Sobrescrita



O polimorfismo dinâmico (o verdadeiro polimorfismo) acontece quando uma mesma instrução assume “formas” diferentes.

- Isso é obtido através da mudança da referência de uma classe base para uma classe filha em conjunto com a chamada de um método sobrescrito.

```
bomber = fireBomber;  
  
bomber.ColocarBomba();  
  
bomber = aquaBomber;  
  
bomber.ColocarBomba();
```

Repare que as instruções **bomber.ColocarBomba()** fazem ações diferentes na linha 2 e 4.

# Regras da Sobrescrita



- Os métodos sobrescritos NÃO PODEM alterar a lista de argumentos
- Os métodos sobrescritos NÃO PODEM alterar o retorno do método (exceto quando trata-se de um subtipo).
- Os métodos sobrecarregados PODEM alterar o nível de acesso APENAS se for mais restritivo.