# INTRODUÇÃO AO FRAMEWORK .NET

Cairu

Fundação Visconde de Cairu - Desde 1905

# AULA 5 – ORIENTAÇÃO A OBJETOS

- Herança
- Polimorfismo
- Sobrecarga
- Métodos e Atributos Estáticos
- Tratamento de Exceção

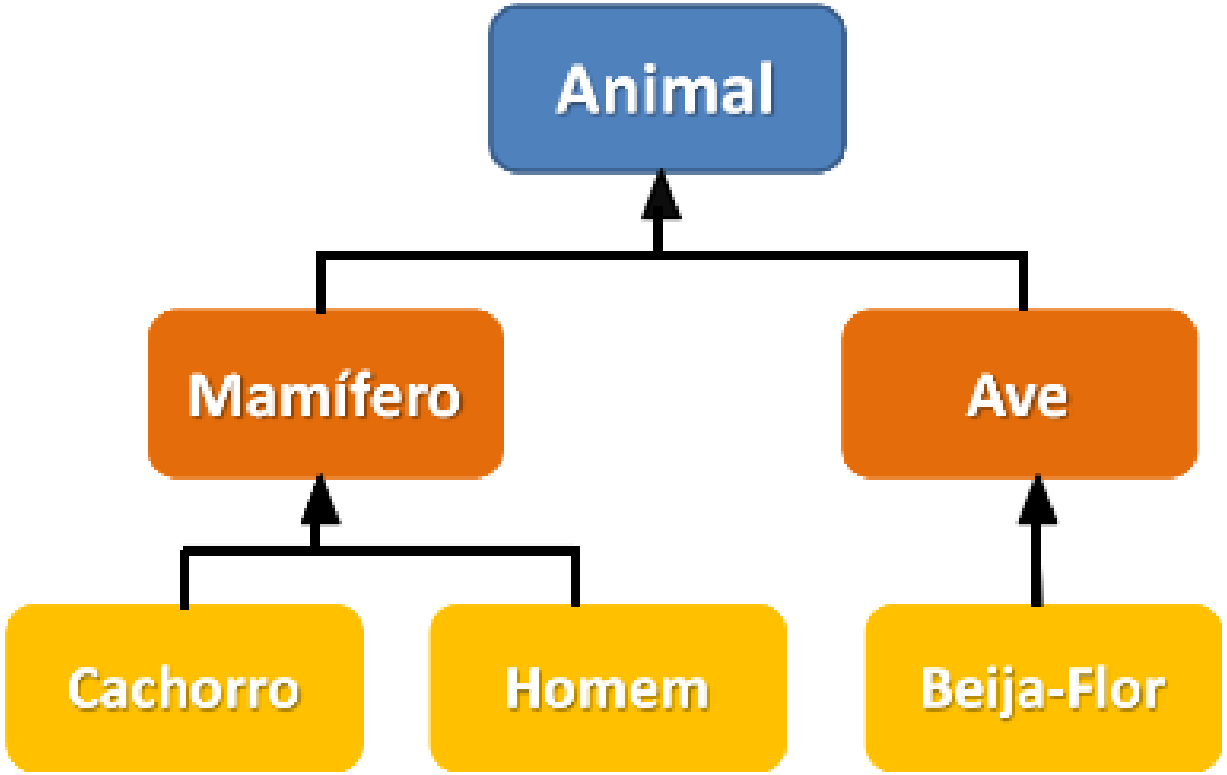
# APRESENTAÇÃO DIA 1 – 11/07 – PROF. ARISTÓTELES

| Início | Fim | Atividade |
|---|---|---|
| 18:30 | 19:20 | Apresentação |
| 19:20 | 20:00 | Introdução ao Framework .Net e Apresentação do Visual Studio 2013 (github) |
| 20:00 | 20:10 | Intervalo |
| 20:10 | 20:50 | Introdução ao C#: tipos de dados, if, else, case, for, while, e/s console |
| 20:50 | 21:30 | Prática 1 – Console Application |

# POO – HERANÇA

- O reuso de código é uma das grandes vantagens da programação orientada a objetos. Muito disso se dá por uma questão que é conhecida como *herança*.
- Herança pode ser definida como a capacidade de uma classe herdar atributos e comportamento de uma outra classe.

# POO – HERANÇA – EXEMPLO



# POO – HERANÇA – EXEMPLO

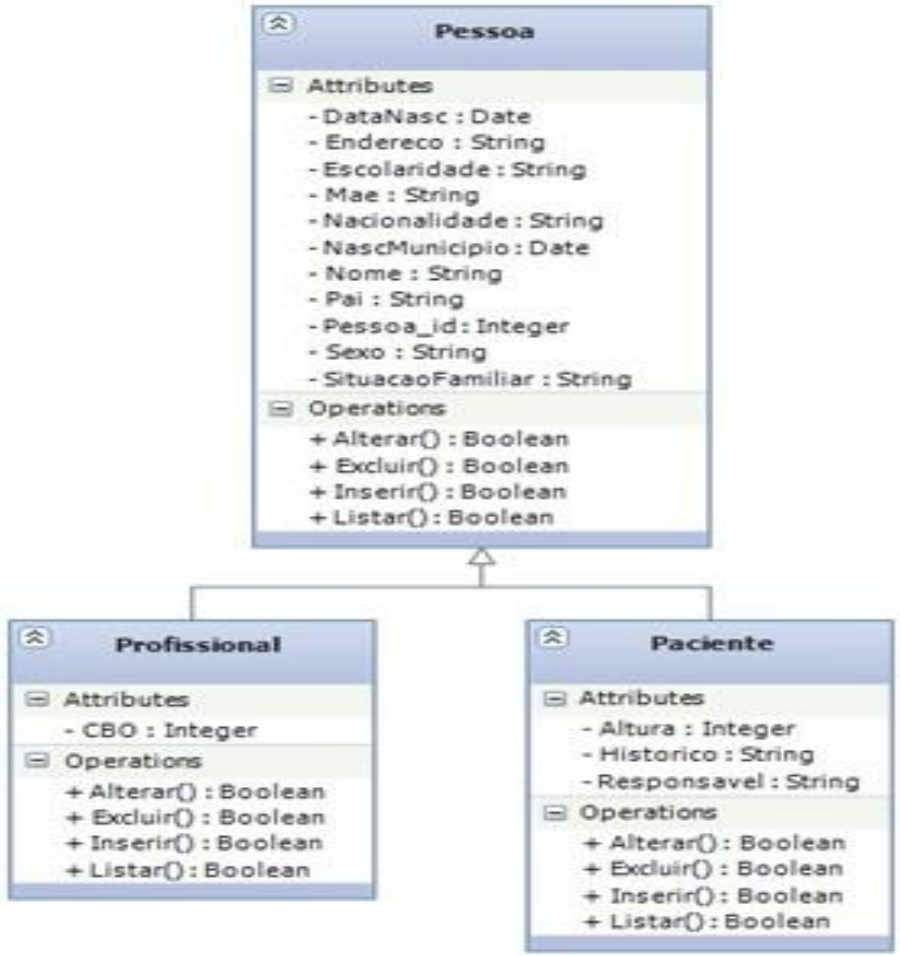
Um ser humano é um animal. Ele tem todas as características (atributos) e pode realizar todas as ações (métodos) de um animal. Mas além disso, ele tem algumas características e ações que só ele pode realizar.

Em momentos como este, é utilizado a herança. Uma classe pode estender todas as características de outra e adicionar algumas coisas a mais. Desta forma, a classe **SerHumano** será uma especialização (ou **subclasse**) da classe **Animal**. A classe Animal seria a **classe pai** da serHumano, e logicamente, a classe SerHumano seria a **classe filha** da Animal.

# POO – HERANÇA – EXEMPLO

- Classe Pessoa: Nome e Idade
- Classe Pessoa Física: Nome, Idade e CPF
- Classe Pessoa Jurídica: Nome, Idade e CNPJ

# POO – HERANÇA – EXEMPLO



# POO – HERANÇA

- A utilização da herança é mais que uma simples economia de código, significa mais integridade. Quando um comportamento é alterado, todas as classes que descende dela terá acesso aos métodos atualizados sem necessidade de reprogramação.

# POO – HERANÇA EM C#.NET

- Para especificar que uma classe herda de uma classe base

```
class DerivedClass : BaseClass
{
...
}
```

# POO – HERANÇA EM C#.NET

- Para especificar que uma classe não pode ser usada como classe base

```
sealed class SampleClass
{
...
}
```

# POO – HERANÇA EM C#.NET (CLASSE ABSTRATA)

- Para especificar que uma classe pode ser usada apenas como uma classe base e não pode ser instanciada

```
abstract class BaseClass
{
...
}
```

# POO – POLIMORFISMO

- Um dos conceitos mais complicados de se entender, e também um dos mais importantes, é o **Polimorfismo**. O termo polimorfismo é originário do grego e significa "muitas formas".
- O polimorfismo está diretamente ligado à hereditariedade das classes, este trabalha com a redeclararão de métodos herdados, ou seja, os métodos têm a mesma assinatura (têm o mesmo nome), mas a forma de implementação utilizada diferem o da superclasse
- Segundo Sintes (2002, p. 122), “de sua própria maneira, o polimorfismo é o distúrbio das múltiplas personalidades do mundo do software, pois um único nome pode expressar muitos comportamentos diferentes”.

# POO – POLIMORFISMO

| Modificador C# | Definição |
| --- | --- |
| Virtual | Permite que um membro da classe seja substituído em uma classe derivada. |
| Override | Substitui um membro virtual (substituível) definido na classe base. |
| Abstract | Exige que um membro da classe seja substituído na classe derivada |

# POO – POLIMORFISMO EM C#.NET

Virtual

```
public virtual void Desenhar(Graphics g)
{

}
```

Override

```
public override void Desenhar(Graphics g)
{
  base.Desenhar ();
  g.DrawEllipse(Pens.Red, 5, 5, 100, 100);
}
```

# POO – POLIMORFISMO

- Capacidade de objetos diferentes possuírem métodos de mesmo nome e mesma lista de parâmetros que quando chamados executam tarefas de maneiras diferentes
- A diferença entre o uso dos modificadores Overridable/Virtual com Overrides/Overrride e Abstract é que este na implementação da classe é obrigatório a implementação do método

# POO – SOBRECARGA

- A sobrecarga de métodos, também conhecida como *overloading*, ocorre quando criamos dois ou mais métodos com o mesmo nome mas com uma lista de argumentos diferentes
- As sobrecarga pode ser aplicada ao construtor ou métodos da classe

# POO – SOBRECARGA – CONSTRUTOR

```
class ClasseSimples
{
        public ClasseSimples()
        {
                // Add code here.
        }
        public ClasseSimples(string s)
        {
                // Add code here.
        }
}
```

# POO – SOBRECARGA – MÉTODO

```
class ClasseSimples
{
        public void SimplesFunc(string SimplesParam)
        {
                // Add code here
        }
        public void SimplesFunc(int IntParam)
        {
                // Add code here
        }
}
```

# POO – NAMESPACE (PACOTE)

- Um Namespace é um esquema lógico de nomes para tipos no qual um nome de tipo simples, como MeuTipo, aparece precedido por um nome hierárquico separado por ponto

| | |
|---|---|
| **Data:** | •System.Data<br>•System.XML |
| **Component Model:** | •System.CodeDOM<br>•System.ComponentModel<br>•System.Core |
| **Configuration:** | •System.Configuration |

# POO – ATRIBUTOS E MÉTODOS ESTÁTICOS

- As propriedades e métodos de uma classe podem ser membros de instância da classe ou membros compartilhados (shared) ou estáticos.
- Membros de instância são associados com instâncias de um tipo , enquanto que membros estáticos (shared) são associados com a classe e não com uma instância em particular.
- Métodos são métodos de instância a menos que você explicitamente os defina como métodos estáticos (shared) usando a palavra-chave Shared.
- A grande maioria dos métodos são métodos de instância , isto significa que você deve acessá-los a partir de um objeto especifico. Quando você precisar de um método de classe , ou seja , um método que seja acessado diretamente da classe você deve defini-lo como estático ou Shared.
- Você pode acessar um membro shared através do nome da classe na qual ele foi declarado.

# POO – ATRIBUTOS E MÉTODOS ESTÁTICOS

```
class Cachorro
{
        private static int instancias = 0;
        private int peso;
        private string nome;
        public Cachorro(string name, int
peso)
        {
                instancias += 1;
                this.nome = nome;
                this.peso = peso;
        }
```

```
public static void quantosCachorros()
{
    Console.WriteLine("{0} cachorros
instanciados", instancias);
}
public void informaPeso()
{
   Console.WriteLine("{0} is {1} Kg", nome, peso);
}

}
```

# POO – TRATAMENTO DE ERRO

- A estrutura de Try/Catch é utilizada para evitar que o sistema deixe seu funcionamento por causa de um erro

```
try {
        Numero *= 888888;
        Console.WriteLine("Numero alterado.");
} catch (Exception Erro) {
        Interaction.MsgBox("Ocorreu um erro na aplicação. " + Erro.Message);
} finally {
        Interaction.MsgBox(Numero.ToString());
}
```

# PRÁTICA 5

# AULA 6 – ORIENTAÇÃO A OBJETOS

- Exercício de Fixação

# PRÁTICA 6