# Apostila de ASP.NET

# 1. Herança

É um conceito usado na programação orientada a objetos para descrever a relação entre as classes. Através da herança uma classe herda todas as propriedades, métodos e atributos de uma outra classe, podendo assim estender a sua funcionalidade.

A classe que cede os membros a outra classe é chamada de **superclasse, classe pai ou classe derivada**. Este processo permite a reutilização de código e especifica um relacionamento de especialização/generalização.

Para indicarmos que uma classe será derivada de outra devemos utilizar o símbolo **:** (dois pontos), como mostra o exemplo:

**public class ClasseB : ClasseA**

## 1.1. Criando a classe Animal

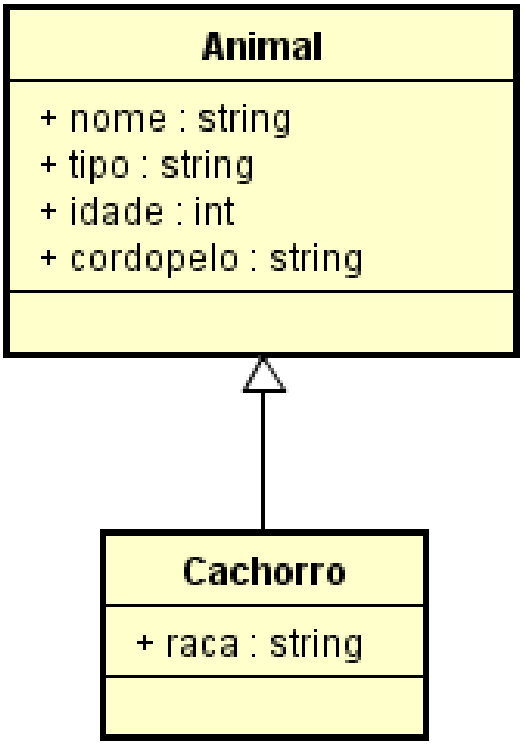


***Figura 1 - Diagrama da classe Animal***

Vamos criar o projeto do tipo **Class Library**, com o nome **Animal**, e devemos definir os atributos como mostra a figura 2.

```
public class Animal
{
    public string nome;
    public string tipo;
    public int idade;
    public string cordopelo;
}
```

***Figura 2 - Programação da classe Animal***

Para aplicar a herança a classe criada, devemos criar outra classe, dentro do mesmo projeto, clicando com o botão direito sobre a aplicação, como mostra a figura 3.

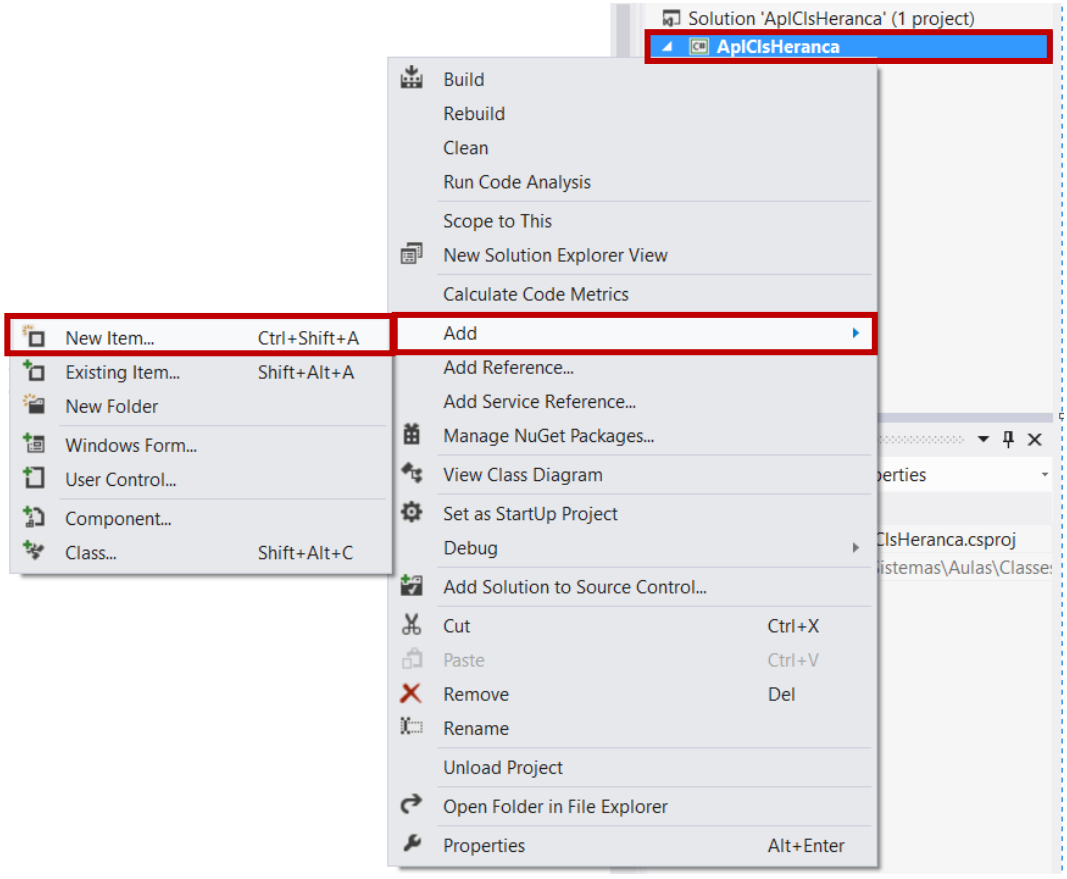


***Figura 3 - Criando outra classe dentro do mesmo projeto***

Em seguida devemos selecionar a template **Class**, como mostra a figura 4.

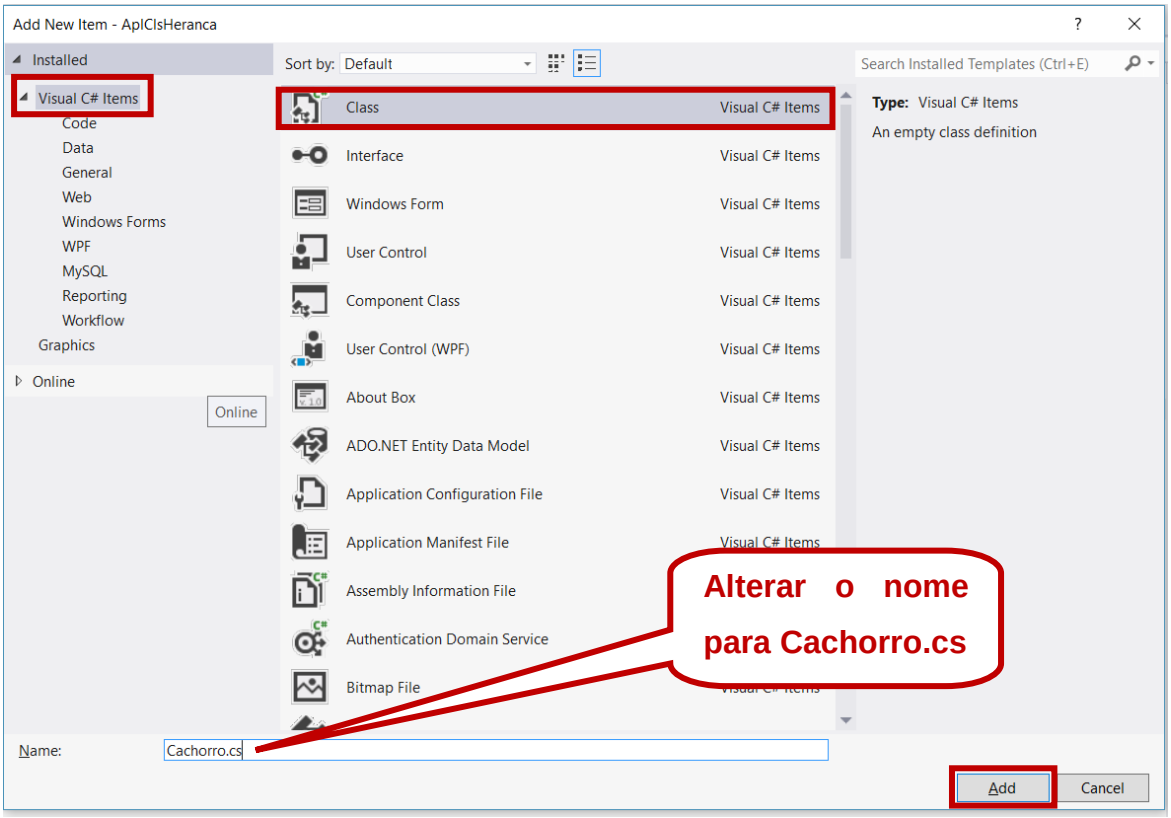


***Figura 4 - Seleção da template Class***

Após inserir a nova classe, devemos criar o relacionamento **(herança)** entre as classes através do símbolo **: (dois pontos)**, como mostra a figura 5.

```
public class Cachorro : Animal
{
    public string raca;
}
```

***Figura 5 - Classe Cachorro herdando da classe Animal***

Não podemos esquecer de adicionar ao início da classe a instrução **public**, com isso, podemos deixar a classe visível quando instanciamos.

A partir de agora devemos gerar a biblioteca, ou seja, o arquivo **.dll**. Para isso, devemos clicar com o botão direito sobre a aplicação (**AplClsHeranca**) e selecionar a opção **Build**, como mostra a figura 6.

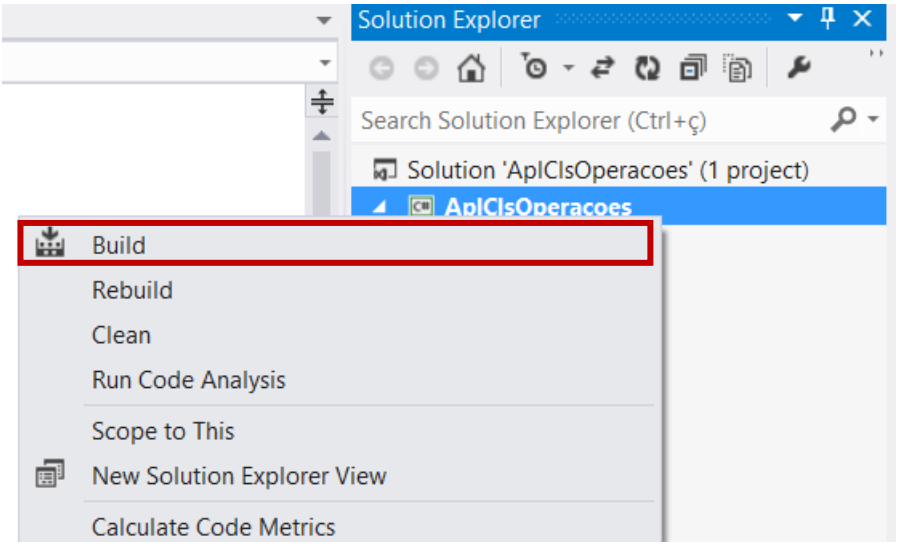


***Figura 6 - Build da classe Animal***

Após a seleção da opção, no canto inferior do ambiente de programação, aparecerá o início e fim da construção da sua biblioteca, como mostra a figura 7.

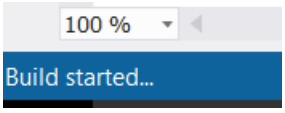


***Figura 7 – Build da classe inicializado***

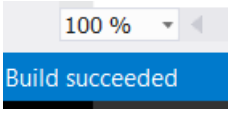


***Figura 8 - Build da classe finalizado***

Não se esqueça que a biblioteca foi gerada na pasta Debug do projeto.

## 1.2. Instanciando a classe

Antes de utilizarmos a classe devemos inseri-la ao nosso projeto através da **References**, localizada na aba **Solution Explorer** (vide figura 13 da aula 2).

Após a inserção da biblioteca devemos importa-la ao ambiente de

programação como mostra a figura 9.

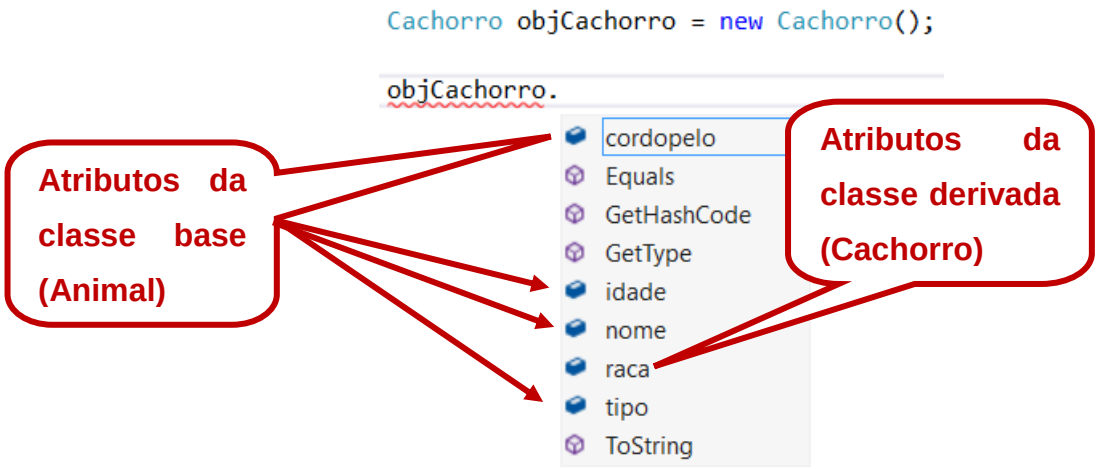


***Figura 9 - Instanciado a classe Cachorro***

## 1.3. Operador Base

Este operador permite ter acesso aos atributos da classe base. Com isso, qualquer referência feita usando o operador **base**, dentro de uma classe derivada, o mesmo será apontado a classe base.

Por exemplo, se temos uma classe base chamada de **Pai**, e outra classe derivada chamada de **Filha**, quando referenciamos na classe **Filha** qualquer atributo com o operador **base**, estamos apontando ao atributo da classe **Pai** e não da **Filha**, como mostraremos a seguir.

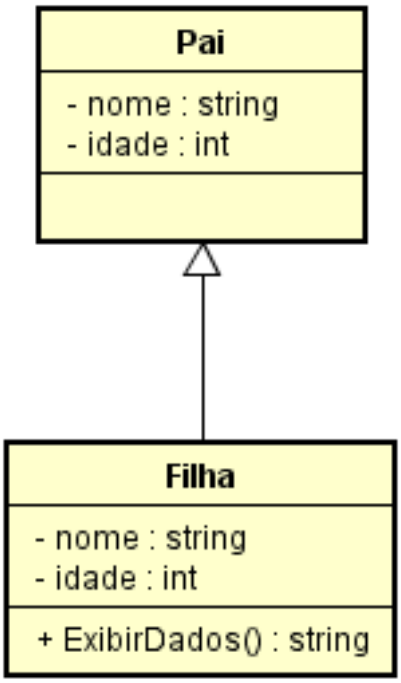


*Figura 10 - Diagrama da classe Pai e Filha*

Como mostra a figura 11, iremos programar as classes **Pai** e **Filha**.

```
public class Pai
{
    public string nome;
    public int idade;
}
```

```
public class Filha : Pai
{
    public new string nome;
    public new int idade;

    public string ExibirDados()
    {

    }
}
```

*Figura 11 - Diagrama das classe Pai e Filha*

Observem que as classes possuem os mesmos atributos, ou seja, com a mesma identificação. Apesar de não ser recomendado, podemos manter atributos de nomes iguais nas classes **Pai** e **Filha**. Este procedimento é chamado de **hiding**.

Se vamos manter os atributos iguais e para que não haja conflito de dados, devemos adicionar a instrução **new** na frente do atributo da classe derivada, caso contrário o compilador irá enviar um alerta sobre o uso de **hiding**, como mostra a figura 12.

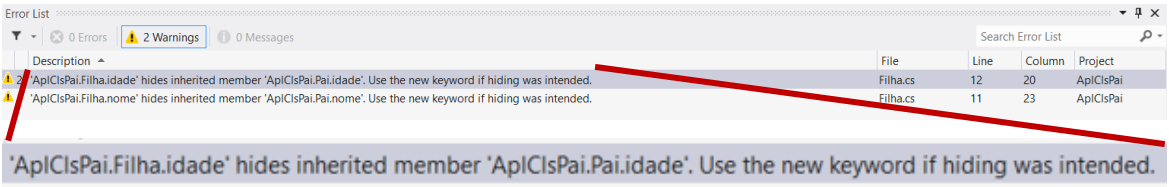


*Figura 12 - Erro de hiding na classe*

Agora vamos criar o método **ExibirDados()**, dentro da classe Filha, como mostra a figura 13.

```
public class Filha : Pai
{
    public new string nome;
    public new int idade;

    public Filha(string nomepai, int idadepai)
    {
        base.nome = nomepai;
        base.idade = idadepai;
    }

    public string ExibirDados()
    {
        string strTexto = "Nome do Pai: " + base.nome + "<br>";
        strTexto += "Idade do Pai: " + base.idade + "<br>";
        strTexto += "Nome da Filha: " + this.nome + "<br>";
        strTexto += "Idade da Filha: " + this.idade;

        return strTexto;
    }
}
```

*Figura 13 - Programação da classe Filha*

## 1.4. Polifmorfismo

É um dos pilares da programação orientada a objeto mais utilizada. Se definirmos a palavra, **Poli** significa **muitos** e **Morfo** significa **formas**, então literalmente quer dizer **“muitas formas”**.

Há três tipos de polimorfismo:

a) Por herança, quando se herda de uma classe normal e pode se converter nela mesma;

b) Por abstração, quando se herda de uma classe abstrata e pode se converter nela mesma;

c) Por interface, é a possibilidade que temos de implementar uma

interface e pode se converter nela mesmo.

Neste caso iremos descrever somente o polimorfismo por herança.

### 1.4.1. Polimorfismo por herança

Este tipo de polimorfismo é o mais comum que existe e tem a facilidade de herdar de uma classe pai e reescreve-la. Iremos utilizar o diagrama de classe conforme a figura 14.

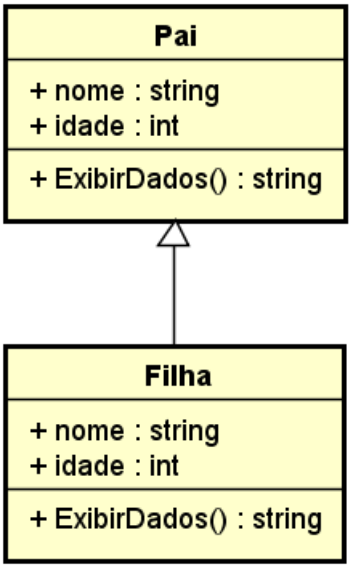


*Figura 14 - Diagrama da classe Pai e Filha (Polimorfismo)*

Para podermos reescreve-la na classe derivada devemos utilizar a instrução **override** e na classe pai devemos utilizar a instrução **virtual** no método que será sobrescrito, como mostra a figura 15.

Primeiramente iremos definir o método **ExibirDados** da classe Pai com a instrução **virtual**, conforme a figura 16.

```
public class Pai
{
    public string nome;
    public int idade;

    public virtual string ExibirDados()
    {
        return "O pai " + this.nome + " tem " + this.idade.ToString() + " anos.";
    }
}
```

*Figura 15 - Programação da classe Pai (Polimorfismo)*

Em seguida vamos definir o método **ExibirDados** da classe Filha com a instrução **override**, conforme a figura 17.

```
public class Filha : Pai
{
    public new string nome;
    public new int idade;

    public Filha(string nomepai, int idadepai)
    {
        base.nome = nomepai;
        base.idade = idadepai;
    }

    public override string ExibirDados()
    {
        return "A filha " + this.nome + " tem " + this.idade.ToString() + " anos.";
    }
}
```