5 – Polimorfismo

Iremos entrar agora num assuno muito interessante que é polimorfismo. Se você seguiu toda a trajetória das aulas sobre POO deve ter visto várias formas de polimorfismo (isso não foi um trocadilho) durante a leitura.

Vimos em:
Herança (POO) Abstração e Interface (POO)

Leia esta aula, depois volte a eles e tente identificar o uso do polimorfismo. Aqui iremos abordar o polimorfismo de uma forma mais ampla e detalhada. Então vamos lá.

Polimorfismo
Etimologicamente, polimorfismo quer dizer:
1. Que se apresenta ou ocorre sob formas diversas.
2. Que assume ou passa por várias formas, fases etc.

Em orientação a objetos podemos dizer também que o objeto pode ter "vários comportamentos".
Quando você chama um método ele pode se referir a qualquer objeto, por exemplo o método "Abrir()", ele pode abrir uma tampa de garrafa, uma janela, uma porta, pode abrir alguma coisa.

Em polimorfismo é necessário que os métodos tenham a mesma assinatura, o mecanismo utilizado é o de sobrescrita de métodos (Override), não confundir com sobrecarga de métodos (Overload).

Em polimorfismo podemos:
1. Instanciar objetos diferentes para um mesmo tipo.
2. Substituir condicionais com o polimorfismo.
3. Usar parâmetros que aceitam diferentes tipos de objetos desde que tenham a mesma herança de classes.

Vamos a alguns exemplos para reforçarmos as ideias.

Vamos criar nossas classes que servirão de base para todo o exemplo.

Iremos criar uma calculadora bem simples, que implemente as quatro operações básicas.

Classe Base

```
public abstract class CalculoMatematico
 {
     public double Valor1 { get; set; }
     public double Valor2 { get; set; }

     public abstract double Resultado();

 }
```

Agora iremos criar as classes que realizarão as operações específicas, que herdarão a classe abstrata CalculoMatematico e irão reescrever o método abstrato Resultado de acordo com a operação desejada.

Multiplicação

```
public class Multiplicacao : CalculoMatematico
 {
     public override double Resultado()
     {
         return Valor1 * Valor2;
     }
 }
```

Divisão

```
public class Divisao : CalculoMatematico
 {
     public override double Resultado()
     {
         return Valor1 / Valor2;
     }
 }
```

Adição

```
 public class Adicao : CalculoMatematico
 {
     public override double Resultado()
     {
         return Valor1 + Valor2;
     }
 }
```

Subtração

```
 public class Subtracao : CalculoMatematico
 {
     public override double Resultado()
     {
         return Valor1 - Valor2;
     }
 }
```

Vamos agora ao primeiro exemplo. No Main() de seu programa, vamos escrever o seguinte:
Partindo de um mesmo tipo de objeto, podemos instanciar os outros tipos. Mas para isso todos as classes devem implementar a mesma classe Pai.
Sabemos que não podemos instanciar uma classe abstrata, mas nada impede de criar um objeto (sem instanciá-lo) deste tipo de classe. Isso assim:

```
CalculoMatematico calc;
```

Agora podemos instanciar este objeto, de forma que ele seja derivado da classe que representa o tipo de operação que desejamos executar. Por exemplo, para fazer uma adição faremos assim:

```
Calc = new Adicao();
```

Agora, basta colocar os valores nos atributos/propriedades e chamar o método resultado para que ele retorne a soma.

```
    Console.Write("Digite um valor inteiro para o valor 1: ");
    Calc.Valor1 = Convert.ToInt32(Console.ReadLine());
    Console.Write("Digite um valor inteiro para o valor 2: ");
    Calc.Valor2 = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("A soma de {0} + {1} é {2}.", Calc.Valor1, Calc.Valor2, Calc.Resultado());
```

Então, como agora queremos realizar outro tipo de operação, basta na sequência instanciar Calc com a próxima operação. Por exemplo, se quisermos agora realizar uma divisão, basta colocar logo após esta última linha, uma troca de instância para Calc, desta forma:

```
Calc = new Divisao();
```

Porém, quando fazemos isso, seus atributos são novamente reiniciados, portanto devemos fazer novamente a leitura deles. Poderíamos criar duas variáveis inteiras no momento da leitura e só repassar aos atributos antes de realizar a conta, caso os números sejam os mesmos.

```
Console.Write("Digite um valor inteiro para o valor 1: ");
Calc.Valor1 = Convert.ToInt32(Console.ReadLine());
Console.Write("Digite um valor inteiro para o valor 2: ");
Calc.Valor2 = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("A divisão de {0} / {1} é {2}.", Calc.Valor1, Calc.Valor2, Calc.Resultado());
```

Faremos o mesmo para subtração e multiplicação:

```
Calc = new Multiplicacao();
Console.Write("Digite um valor inteiro para o valor 1: ");
Calc.Valor1 = Convert.ToInt32(Console.ReadLine());
Console.Write("Digite um valor inteiro para o valor 2: ");
Calc.Valor2 = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("A multiplicação de {0} * {1} é {2}.", Calc.Valor1, Calc.Valor2, Calc.Resultado());

Calc = new Subtracao();
Console.Write("Digite um valor inteiro para o valor 1: ");
Calc.Valor1 = Convert.ToInt32(Console.ReadLine());
Console.Write("Digite um valor inteiro para o valor 2: ");
Calc.Valor2 = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("A subtração de {0} - {1} é {2}.", Calc.Valor1, Calc.Valor2, Calc.Resultado());
```

O nosso segundo item diz que podemos substituir condicionais com o uso de polimorfismo. Será? Vamos ver os exemplos abaixo, reparem bem na diferença entre os dois métodos da classe Calculadora.

Para isso iremos utilizar o conceito de enumeração (enum em C#).
Um tipo de enumeração (também chamado uma enumeração ou enum) fornece uma maneira eficiente para definir um conjunto de constantes nomeadas inteiro que podem ser atribuídas a uma variável. Por exemplo, suponha que você precisa definir uma variável cujo valor representa uma operação matemática. Há, no nosso exemplo, apenas quatro valores significativos que essa variável sempre armazenará. Para definir esses valores, você pode usar um tipo de enumeração, que é declarada usando a palavra-chave de enum, desta forma:

```
public enum TipoCalculo { Adicao, Subtracao, Multiplicacao, Divisao }
```

Desta forma, o TipoCalculo funcionará convertendo Adição em 0, Subtração em 1, Multiplicação em 2 e Divisão em 3, como se fosse índice de um vetor (array).

Então, antes do Main(), vamos inserir esse nosso enum e ainda criar uma classe estática* com a seguinte especificação:
** Só para lembrar, o que é classe estática? Classe não precisa ser instanciada. Isso quer dizer que não é preciso criar um "novo" objeto, podemos simplesmente chamar seus métodos e propriedades.*

```
public static class Calculadora
 {
    public static double Calcular(TipoCalculo tipo, double valor1, double valor2)
    {
        double resultado = 0; //variável que guardará o resultado da conta solicitada

        switch (tipo) //aqui vamos fazer a condicional do tipo de calculo
        {
            case TipoCalculo.Adicao :
                resultado = valor1 + valor2;
                break;
            case TipoCalculo.Subtracao :
                   resultado = valor1 - valor2;
                break;
            case TipoCalculo.Multiplicacao :
                   resultado = valor1 * valor2;
                break;
            case TipoCalculo.Divisao :
                   resultado = valor1 / valor2;
                break;
            default :
                Console.WriteLine("nunca teremos como chegar aqui!");
                break;
        }
        return resultado;
    }
 }
```

Então podemos agora implementar em nosso Main() as chamadas para os métodos da classe estática:

```
//Exemplo utilizando a condicional switch
Console.WriteLine(Calculadora.Calcular(TipoCalculo.Adicao, 23, 34));

//podemos pedir os valores também para processarmos a operação:
int num1, num2;
Console.Write("Digite um valor inteiro para o valor 1: ");
num1 = Convert.ToInt32(Console.ReadLine());
Console.Write("Digite um valor inteiro para o valor 2: ");
```

```
num2 = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("Adicao:");
Console.WriteLine(Calculadora.Calcular(TipoCalculo.Adicao, num1, num2));
Console.WriteLine("Subtracao:");
Console.WriteLine(Calculadora.Calcular(TipoCalculo.Subtracao, num1, num2));
Console.WriteLine("Multiplicacao:");
Console.WriteLine(Calculadora.Calcular(TipoCalculo.Multiplicacao, num1, num2));
Console.WriteLine("Divisao:");
Console.WriteLine(Calculadora.Calcular(TipoCalculo.Divisao, num1, num2));
```

Para exemplo do item 3, podemos criar na classe estática calculadora, um método que aceitará vários tipos de objetos como parâmetros. Vamos adicionar então este método na classe Calculadora

```
public static double Calcular(CalculoMatematico calc)
{
    return calc.Resultado();
}
```

Desta forma, todos os objetos que são instanciados que herdaram as caracteríscicas da classe abstrata CalculoMatematico pode ser utilizados como parâmetro.
Acrescente no Main(), logo depois da leitura das variáveis num1 e num2 as seguintes instruções:

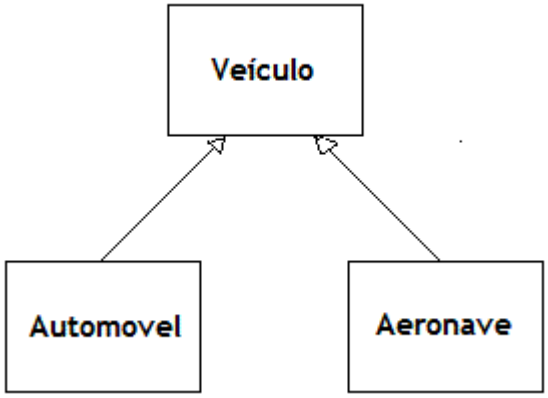
```
//chamando o método com parâmetros de diferentes tipos de objetos
CalculoMatematico Calcgenerico;
Calcgenerico = new Adicao(); //criado como objeto adição
Calcgenerico.Valor1 = num1;
Calcgenerico.Valor2 = num2;
Console.WriteLine(Calculadora.Calcular(Calcgenerico));

Calcgenerico = new Subtracao(); //criado como objeto subtração
Calcgenerico.Valor1 = num1;
Calcgenerico.Valor2 = num2;
Console.WriteLine(Calculadora.Calcular(Calcgenerico));

Calcgenerico = new Multiplicacao(); //criado como objeto multiplicação
Calcgenerico.Valor1 = num1;
Calcgenerico.Valor2 = num2;
Console.WriteLine(Calculadora.Calcular(Calcgenerico));

Calcgenerico = new Divisao(); //criado como objeto divisão
Calcgenerico.Valor1 = num1;
Calcgenerico.Valor2 = num2;
Console.WriteLine(Calculadora.Calcular(Calcgenerico));
```

Assim terminamos a exemplificação das formas de polimorfismo de POO em C#

O código fonte completo de nossa aplicação será colocado no final deste texto.

Exercício:

Crie uma classe abstrata chamada Veículo, que terá duas classes filhas chamadas Automóvel e Aeronave, seguindo a seguinte estrutura:



A classe abstrata Veículo terá o parâmetro Nomedoveículo, do tipo string, e dois métodos abstratos do tipo void chamados Mover() e Parar().
Crie as duas classes que herdam veículo, reescrevendo seus métodos de acordo com o tipo de ação que deve ser feito pelo objeto para mover ou parar o veículo (se for automóvel, acelerar e frear o automóvel, se for aeronave, decolar ou pousar a aeronave).
Em seguida, escreva no Main() um objeto do tipo Veiculo que, quando instanciado para automóvel, mostre os métodos Mover() e Parar(); depois instancie ele com aeronave e repita os procedimentos.

```
//Código fonte completo do exemplo desenvolvido para a aula

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;


    class Program
    {
        public abstract class CalculoMatematico
        {
            public double Valor1 { get; set; }
            public double Valor2 { get; set; }

            public abstract double Resultado();

        }

        public class Multiplicacao : CalculoMatematico
        {
            public override double Resultado()
            {
                return Valor1 * Valor2;
            }
        }

        public class Divisao : CalculoMatematico
        {
            public override double Resultado()
            {
                return Valor1 / Valor2;
            }
        }

        public class Adicao : CalculoMatematico
        {
            public override double Resultado()
            {
                return Valor1 + Valor2;
            }
        }

        public class Subtracao : CalculoMatematico
        {
            public override double Resultado()
            {
                return Valor1 - Valor2;
            }
        }

        public enum TipoCalculo { Adicao, Subtracao, Multiplicacao, Divisao }

        public static class Calculadora
        {
           public static double Calcular(TipoCalculo tipo, double valor1, double valor2)
           {
               double resultado = 0; //variável que guardará o resultado da conta solicitada

               switch (tipo) //aqui vamos fazer a condicional do tipo de calculo
               {
                   case TipoCalculo.Adicao :
```

```
                resultado = valor1 + valor2;
                break;
            case TipoCalculo.Subtracao :
                   resultado = valor1 - valor2;
                break;
            case TipoCalculo.Multiplicacao :
                   resultado = valor1 * valor2;
                break;
            case TipoCalculo.Divisao :
                   resultado = valor1 / valor2;
                break;
            default :
                Console.WriteLine("nunca teremos como chegar aqui!");
                break;
        }
        return resultado;
    }

    public static double Calcular(CalculoMatematico calc)
    {
        return calc.Resultado();
    }
}

public static void Main()
{
    //Exemplo utilizando a condicional switch
    Console.WriteLine(Calculadora.Calcular(TipoCalculo.Adicao, 23, 34));

    //podemos pedir os valores também para processarmos a operação:
    int num1, num2;
    Console.Write("Digite um valor inteiro para o valor 1: ");
    num1 = Convert.ToInt32(Console.ReadLine());
    Console.Write("Digite um valor inteiro para o valor 2: ");
    num2 = Convert.ToInt32(Console.ReadLine());

    //chamando o método com parâmetros de diferentes tipos de objetos
    CalculoMatematico Calcgenerico;
    Calcgenerico = new Adicao(); //criado como objeto adição
    Calcgenerico.Valor1 = num1;
    Calcgenerico.Valor2 = num2;
    Console.WriteLine(Calculadora.Calcular(Calcgenerico));

    Calcgenerico = new Subtracao(); //criado como objeto subtração
    Calcgenerico.Valor1 = num1;
    Calcgenerico.Valor2 = num2;
    Console.WriteLine(Calculadora.Calcular(Calcgenerico));

    Calcgenerico = new Multiplicacao(); //criado como objeto multiplicação
    Calcgenerico.Valor1 = num1;
    Calcgenerico.Valor2 = num2;
    Console.WriteLine(Calculadora.Calcular(Calcgenerico));

    Calcgenerico = new Divisao(); //criado como objeto divisão
    Calcgenerico.Valor1 = num1;
    Calcgenerico.Valor2 = num2;
    Console.WriteLine(Calculadora.Calcular(Calcgenerico));

    Console.WriteLine("Adicao:");
    Console.WriteLine(Calculadora.Calcular(TipoCalculo.Adicao, num1, num2));
    Console.WriteLine("Subtracao:");
    Console.WriteLine(Calculadora.Calcular(TipoCalculo.Subtracao, num1, num2));
```

```
            Console.WriteLine("Multiplicacao:");
            Console.WriteLine(Calculadora.Calcular(TipoCalculo.Multiplicacao, num1, num2));
            Console.WriteLine("Divisao:");
            Console.WriteLine(Calculadora.Calcular(TipoCalculo.Divisao, num1, num2));


            CalculoMatematico Calc;
            Calc = new Adicao();
            Console.Write("Digite um valor inteiro para o valor 1: ");
            Calc.Valor1 = Convert.ToInt32(Console.ReadLine());
            Console.Write("Digite um valor inteiro para o valor 2: ");
            Calc.Valor2 = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("A soma de {0} + {1} é {2}.", Calc.Valor1, Calc.Valor2,
Calc.Resultado());

            Calc = new Divisao();
            Console.Write("Digite um valor inteiro para o valor 1: ");
            Calc.Valor1 = Convert.ToInt32(Console.ReadLine());
            Console.Write("Digite um valor inteiro para o valor 2: ");
            Calc.Valor2 = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("A divisão de {0} / {1} é {2}.", Calc.Valor1, Calc.Valor2,
Calc.Resultado());


            Calc = new Multiplicacao();
            Console.Write("Digite um valor inteiro para o valor 1: ");
            Calc.Valor1 = Convert.ToInt32(Console.ReadLine());
            Console.Write("Digite um valor inteiro para o valor 2: ");
            Calc.Valor2 = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("A multiplicação de {0} * {1} é {2}.", Calc.Valor1, Calc.Valor2,
Calc.Resultado());

            Calc = new Subtracao();
            Console.Write("Digite um valor inteiro para o valor 1: ");
            Calc.Valor1 = Convert.ToInt32(Console.ReadLine());
            Console.Write("Digite um valor inteiro para o valor 2: ");
            Calc.Valor2 = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("A subtração de {0} - {1} é {2}.", Calc.Valor1, Calc.Valor2,
Calc.Resultado());

            Console.ReadKey();

        }
    }
```