Abstrações e Funções de Alta Ordem - *Delegates*

Kleber Jacques F. de Souza

Delegates

- *Delegates* são classes especiais que criamos para servir como **ponteiros para métodos**.
- É como se armazenássemos métodos dentro de variáveis, da mesma maneira que armazenamos um conjunto de caracteres dentro de uma variável *string*, por exemplo.

Delegates

- *Delegates* são semelhantes aos ponteiros de função em C++; entretanto, *delegates* são **fortemente tipados** e **seguros**.

<visibilidade> delegate <retorno> <nome>[(<parâmetros>)]

Delegates

```
public delegate int ExecCalculo(int a, int b);


        public int Somar(int a, int b){
                    return a + b;
        }

        public int Subtrair(int a, int b){
                    return a - b;
        }
```

Delegates

```
 static void Main(string[] args){
   ExecCalculo somarDelegate = new ExecCalculo(Somar);
   ExecCalculo subtrairDelegate = new ExecCalculo(Subtrair);
   Console.WriteLine(somarDelegate(2, 1));
   Console.WriteLine(subtrairDelegate(2, 1));
}
```

Delegates

- Os *delegates* armazenam as seguintes informações sobre um método:
  - Endereço de memória do método.
  - Quantidade, ordem e tipo dos parâmetros.
  - Tipo do retorno do método, mesmo que seja **void.**

Delegates Multicast

- Quando criamos um *delegate*, na verdade estamos criando uma classe que tem a capacidade de apontar para um método.
- Qualquer *delegate* que você crie é um ***delegate*** do tipo ***multicast***.

Delegates Multicast

- ***Delegates Multicast*** são *delegates* que podem **apontar para vários métodos ao mesmo tempo**, executando-os na sequência em que são atribuídos ao *delegate* em questão.

Delegates Multicast

```
public delegate void EscreverMsg();

    public void Mensagem1() {
            Console.WriteLine("Mensagem 1");
    }
    public void Mensagem2(){
            Console.WriteLine("Mensagem 2");
    }
    public void Mensagem3(){
            Console.WriteLine("Mensagem 3");
    }
```

Delegates Multicast

```
static void Main(string[] args){
    EscreverMsg mensagemDelegate = new EscreverMsg(Mensagem1);
    mensagemDelegate += Mensagem2;
    mensagemDelegate += Mensagem3;
    mensagemDelegate();
}
```

Mensagem 1
Mensagem 2
Mensagem 3

Delegates Multicast

```
static void Main(string[] args){
    EscreverMsg mensagemDelegate = new EscreverMsg(Mensagem2);
    mensagemDelegate += Mensagem3;
    mensagemDelegate += Mensagem1;
    mensagemDelegate -= Mensagem2;
    mensagemDelegate();
}
```

Mensagem 3
Mensagem 1

Delegates Multicast

- E se utilizarmos o *delegate* do tipo multicast no nosso primeiro exemplo? Qual seria o resultado?

```
ExecCalculo meuDelegate = new ExecCalculo(Somar);
meuDelegate += Subtrair;
Console.WriteLine(meuDelegate(2, 1));
```

Delegates como parâmetros de métodos

- Você **não precisa** passar a **instância** explicitamente, pode passar o método diretamente!

```
private void EscreverResultado(int a, int b, ExecCalculo del){
    int resultado = del(a, b);
    Console.WriteLine(string.Format("Resultado= {0}", resultado));
}

void Main(string[] args){
    EscreverResultado(3, 3, Somar);
}
```

Delegates de Tipos Genéricos

- Da mesma maneira que podemos aplicar o conceito de ***Generics*** em classes e métodos, também podemos criar ***delegates*** **parametrizados** com Generics a fim de evitar as operações de *boxing* e *unboxing*

Delegates de Tipos Genéricos

```
private delegate T ExecCalculo<T>(T a, T b);

private static int Somar(int a, int b)
{
      return a + b;
}

private static decimal Somar(decimal a, decimal b)
{
      return a + b;
}
```

Delegates de Tipos Genéricos

```
ExecCalculo<int> del1 = new ExecutarCalculo<int>(Somar);
// del1 está parametrizado com o tipo int, portanto, irá apontar para
o método Somar que trabalha com int
Console.WriteLine(string.Format("int: {0}", del1(2, 2)));
// Irá escrever "Delegate com int: 4"

ExecCalculo<decimal> del2 = new ExecutarCalculo<decimal>(Somar);
// del2 está parametrizado com o tipo decimal, portanto, irá apontar
para o método Somar que trabalha com decimal
Console.WriteLine(string.Format("decimal: {0}", del2(2.2M, 2.4M)));
// Irá escrever "Delegate com decimal: 4.6"
```

Delegates dos tipos Func e Action

- O .NET Framework disponibiliza dois tipos de delegates bem genéricos:
  - os delegates **Func** e **Action**, ambos do namespace System.

Delegates dos tipos Func

- *Delegates* do tipo Func podem apontar para métodos que **retornam algum tipo de informação**.
- O último parâmetro sempre indica o tipo de retorno do método.

Delegates dos tipos Func

```
private static int Somar(int a, int b)
{
      return a + b;
}

Func<int, int, int> del = Somar;
Console.WriteLine(del(1, 1));
//Irá imprimir "2", resultado de "1 + 1"
```

Delegates dos tipos Action

- *Delegates* do tipo **Action** podem ser utilizados para métodos que definem "ações", ou seja:
    - métodos que retornam **void**.

Delegates dos tipos Action

```
public void EscreverMensagem(string mensagem)
{
      Console.WriteLine(mensagem);
}

   Action<string> del = EscreverMensagem;
   del("Teste");
   //Irá escrever "Teste"
```

Referências Bibliográficas

Microsoft 2017. **Delegates**. Disponível em: https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/delegates/

Microsoft 2017. ***Delegates Multicast***. Disponível em: https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/delegates/how-to-combine-delegates-multicast-delegates