IFRN

PROGRAMAÇÃO ORIENTADA A OBJETOS EM C#

Tipos Anônimos e Delegados

Prof. Gilbert Azevedo

Objetivos

- Estudar os tópicos relacionados com a Linguagem de Consulta LINQ
  - Tipagem Implícita
  - Tipos Anônimos
  - Inicialização de Objetos e Coleções
  - Delegados e Expressão Lambda
  - Delegados do Framework

Tipagem Implícita

- Definição de objetos sem a declaração explícita do tipo.
- A classe do objeto é inferida no momento de sua instanciação

```
var i = 10;
var s = "TADS";
var v = new[] { 10, 10, 2000 };
```

- i – System.Int32
- s – System.String
- v – System.Int32[]

Tipos Anônimos

- Instanciação de objetos de uma classe não codificada explicitamente

```
var v1 = new
  { modelo="Gol", marca="VW", ano=2010, preco=30000.0 };
var v2 = new
  { modelo="Corsa", marca="GM", ano=2010, preco=35000.0 };
var c1 = new
  { nome="a", email="a@email.com", fone="1234-5678" };
```

- v1, v2 : <>__AnonType0`4
- c1 : <>__AnonType1`3

Inicializadores de Objetos

- Inicialização de objetos sem definição de um construtor

```
class Contato {
  public string Nome { get; set; }
  public string Email { get; set; }
  public string Fone { get; set; }
  public DateTime Nascimento { get; set; }
}

Contato c1 = new Contato
  { Nome = "Nome1", Email = "nome1@email.com" };
Contato c2 = new Contato
  { Nome = "Nome2", Fone = "1234-5678",
    Nascimento = DateTime.Parse("10/10/2000") };
```

Inicializadores de Coleções

- Inicialização de coleções de objetos

```
List<string> l1 = new List<string> {
  "Tecnologia",
  "Análise e Desenvolvimento",
  "Sistemas“
};
```

Coleções de Tipos Anônimos

- Inicialização de vetor de objetos de classes anônimas

```
var l2 = new[] {
  new { modelo = "Gol", marca = "VW", ano = 2010,
    preco = 30000.0 },
  new { modelo = "Corsa", marca = "GM", ano = 2010,
    preco = 35000.0 },
  new { modelo = "Punto", marca = "Fiat", ano = 2010,
    preco = 42000.0 }
};
```

Delegados

- Delegados (delegates) são tipos que representam referências de métodos, especificando uma *lista de parâmetros* e um *tipo de retorno*
  - Permite associar a uma instância do delegado um método compatível
  - Permite invocar o método com a instância do delegado
  - São usados para passar métodos como argumentos para outros métodos

```
delegate double Calculo (double d);
public static void Main (string[] args) {
  Calculo m = Math.Sqrt;    // public static double Sqrt (double d);
  double v = m(16);         // double v = Math.Sqrt(16);
  Console.WriteLine(v);
}
```

Delegados

- Delegados podem referenciar métodos de instância e de classe (estáticos)

```
delegate double Calculo (double d);
class Funcoes {
  public double Quadrado(double d) { return d * d; }
}
class MainClass {
  public static void Main (string[] args) {
    Calculo m = (new Funcoes()).Quadrado;
    double v = m(16);
    Console.WriteLine(v);
  }
}
```

Delegados e Genéricos

- Delegados podem utilizar genéricos na definição dos tipos dos parâmetros e do retorno do método

```
delegate void Dobro<T> (T x);
public static void DobroNumero(double d) {
  Console.WriteLine(2 * d);
}
public static void DobroTexto(string s) {
  Console.WriteLine(s + s);
}
public static void Main (string[] args) {
  Dobro<double> m1 = DobroNumero;
  Dobro<string> m2 = DobroTexto;
  m1(16); m2("Delegates em C#.... ");
}
```

Expressões Lambda

- Expressões Lambdas permitem a definição de métodos anônimos que podem ser invocados com um delegado
  - Expressão Lambda: (parâmetros) => expressão
  - Instrução Lambda: (parâmetros) => { sequência de instruções }

```
delegate double Calculo (double d);
public static void Main (string[] args) {
  Calculo m1 = x => x * x * x;
  Calculo m2 = x => { double t = 0;
    for (int i = 1; i <= x; i++) t += i;
    return t;
  };
  double v1 = m1(10); double v2 = m2(10);
}
```

Delegados do Framework

- Tipos delegados do framework são muito utilizados
  - Action: recebe nenhum ou vários argumentos e não retorna um valor
  - Func: recebe nenhum ou vários argumentos e retorna um valor
  - Predicate: recebe um argumento e retorna um bool

```
public delegate void Action();
public delegate void Action<in T>(T obj);
…
public delegate TResult Func<out TResult>();
public delegate TResult Func<in T, out TResult>(T arg);
…
public delegate bool Predicate<in T>(T obj);
```

Delegados do Framework

- Exemplo de Action, Func e Predicate

```
public static void DobroNumero(double d) {
  Console.WriteLine(2 * d);
}
public static void Main (string[] args) {
  Action<double> m1 = DobroNumero;
  Func<double, double> m2 = Math.Sqrt;
  Predicate<double> m3 = x => x < 40000;
  m1(100);
  double v1 = m2(100);
  bool v2 = m3(100);
}
```

Métodos como Argumentos

- O método ForEach de Array recebe um método como argumento

```
public static void ForEach<T> (T[] array, Action<T> action);
```

```
public static void DobroNumero(double d) {
  Console.WriteLine(2 * d);
}
public static void Main (string[] args) {
  double[] v = { 1, 2, 3, 4, 5 };
  Array.ForEach<double>(v, DobroNumero);
  Array.ForEach<double>(v, x => Console.WriteLine(x*x*x));
}
```

Referências

- Microsoft Visual C# 2010 – Passo a passo – John Sharp, Bookman, 2008
- .Net Magazine – Ano 5, Edição 48 – C#3.0 – Rodrigo Sendin
- Tipos Anônimos
  - https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/anonymous-types
- Delegates
  - https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/delegates/
- Expressões Lambda
  - https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/operators/lambda-expressions