C#

Tiago Tagliari Martinez

Sistemas de Objetos Distribuídos

25/06/2002

Resumo da Apresentação

- “Hello World”
- Básicos do ambiente .NET
- Objetivos da C#
- Recursos da Linguagem

Hello World!

```
using System;

class Hello
{
  static void Main() {
    Console.WriteLine("Hello world!");
  }
}
```

Básicos do Ambiente .NET

- Common Language Runtime – CLR
  - Independência de linguagem
  - Classes comuns entre todas as linguagens
  - Tipos comuns entre todas as linguagens
- WebServices
  - XML / SOAP

Objetivos da C#

- Linguagem Orientada a Componentes
- Tudo é um objeto
- Ambiente de Execução Controlado
- Facilidade de Aprendizado
- Interoperabilidade

Linguagem Orientada a Componentes

- Conceitos de Componentes Fazem Parte da Linguagem
  - Propriedades, Métodos e Eventos
  - Atributos de desenvolvimento e de execução
  - Documentação integrada usando XML
- Redução de Dependências
  - Não são necessários arquivos externos (IDL, cabeçalhos, etc)

Tudo é um Objeto

- C++, Java: tipos primitivos “mágicos” fora da hierarquia de classes
- Smalltalk, Lisp: tipos primitivos são objetos a um custo de performance
- C# unifica tipos primitivos e objetos, prometendo nenhuma perda de performance
- Novos tipos primitivos: SQL, Decimal
- Coleções funcionam para todos os tipos de maneira uniforme

Facilidade de Aprendizado

- Herança de C++
  - namespace, enum, unsigned, ponteiros (em blocos de código "*unsafe*")
- Adições
  - foreach, switch com strings
  - Tipo decimal para aplicações financeiras
  - Parâmetros *ref* e *out*

Ambiente de Execução Controlado

- Coleta de Lixo
- Tratamento de Excessões
- Elimina tipos não inicializados e conversões de tipo inseguras
- Mecanismos para controle de versões embutido no ambiente de execução e na linguagem

Interoperabilidade

- Com outras linguagens da .NET
- WebServices usando XML/SOAP
- COM / Automação OLE
- DLLs locais e sistema operacional (código marcado como inseguro)

Recursos da Linguagem

- Estrutura do Programa
- Hierarquia de Tipos
- Classes e Estruturas
- Interfaces e Enumerações
- *Delegates* (Delegação?)
- Propriedades e Indexadores
- Eventos
- Sobrecarga de operadores
- Controle de Versão
- Código Inseguro

Estrutura do Programa

- Namespaces
  - Contém tipos e outros namespaces
- Declaração de Tipos
  - Classes, estruturas, interfaces, enumerações e *delegates*
- Membros
  - Constantes, campos, métodos, propriedades, indexadores, eventos, operadores, construtores e destrutores
- Indepêndencia da ordem de declaração
  - Implementação junto com declaração (Java)

Estrutura do Programa

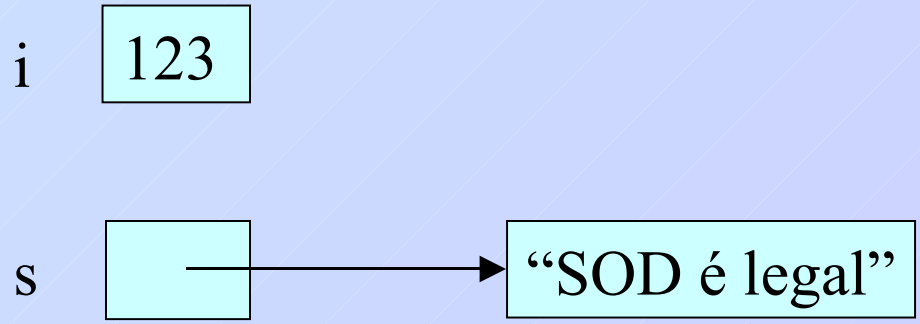
```
using System;
Namespace System.Collections
{
  public class Stack
   {
    Entry top;
    public void Push(object data) {
      top = new Entry(top, data);
    }
    public object Pop() {
      if (top == null) throw new InvalidOperationException();
      object result = top.data;
      top = top.next;
      return result;
    }
  }
}
```

Tipos

- *Value types*
  - Contém diretamente o valor
  - Não podem ser nulos
- *Reference types*
  - Contém uma referência para um objeto
  - Podem ser nulos

Tipos

```
int i = 123;
string s = "SOD é legal";
```



Tipos

- Tipos Valor
  - Primitivos — int i;
  - Enumerações — enum Cor {Azul, Verde}
  - Estruturas — struct Point {int x, y;}
- Tipos Referência
  - Classes — class A : B, IC {...}
  - Interfaces — interface IA : IB {...}
  - Vetores — string[] v = new string[10];
  - Delegações — delegate void Algo();

Tipos Predefinidos

- Tipos predefinidos do C#:
  - Referenciados — object, string
  - Com sinal — sbyte, short, int, long
  - Sem sinal — byte, ushort, uint, ulong
  - Caractere — char
  - Ponto flutuante — float, double, decimal
  - Lógico — bool
- Tipos são apelidos para tipos do .NET
  - int = System.Int32

Classes

- Herança simples
- Implementação de múltiplas interfaces
- Membros de uma classe:
  - Constantes, campos, métodos, propriedades, indexadores, eventos, operadores, construtores e destrutores
  - Membros estáticos e da instância
  - Sub-tipos
- Acesso a membros
  - Public, protected, internal e private

Estruturas

- Como classes, exceto que:
  - Armazenados na pilha
  - Atribuição copia os dados e não a referência
  - Não permitem herança!
- Ideal para objetos menores
  - Complex, Point, Rectangle, Color
  - int, float, double, etc... são estruturas
- Vantagens
  - Sem alocações na heap, menos trabalho para o C. Lixo
  - Uso mais eficiente da memória

Interfaces

- Herança múltipla
- Pode conter métodos, propriedades, indexadores e eventos
- Implementações privadas

```
interface IDesenhavel
{
  void Desenha(Superficie destino);
}

class Triangulo : IDesenhavel
{
  void Desenha(Superficie destino);
}
```

Enumerações

- Fortemente Tipadas
  - Sem conversão implícita de/para int
  - Operadores: +, -, ++, --, &, |, ^, ~
- Pode especificar o tipo interno
  - byte, short, long

```
enum Cor : byte
{
  Red           = 1,
  Green         = 2,
  Blue          = 4,
  Black         = 0,
  White         = Red | Green | Blue;
}
```

Delegações

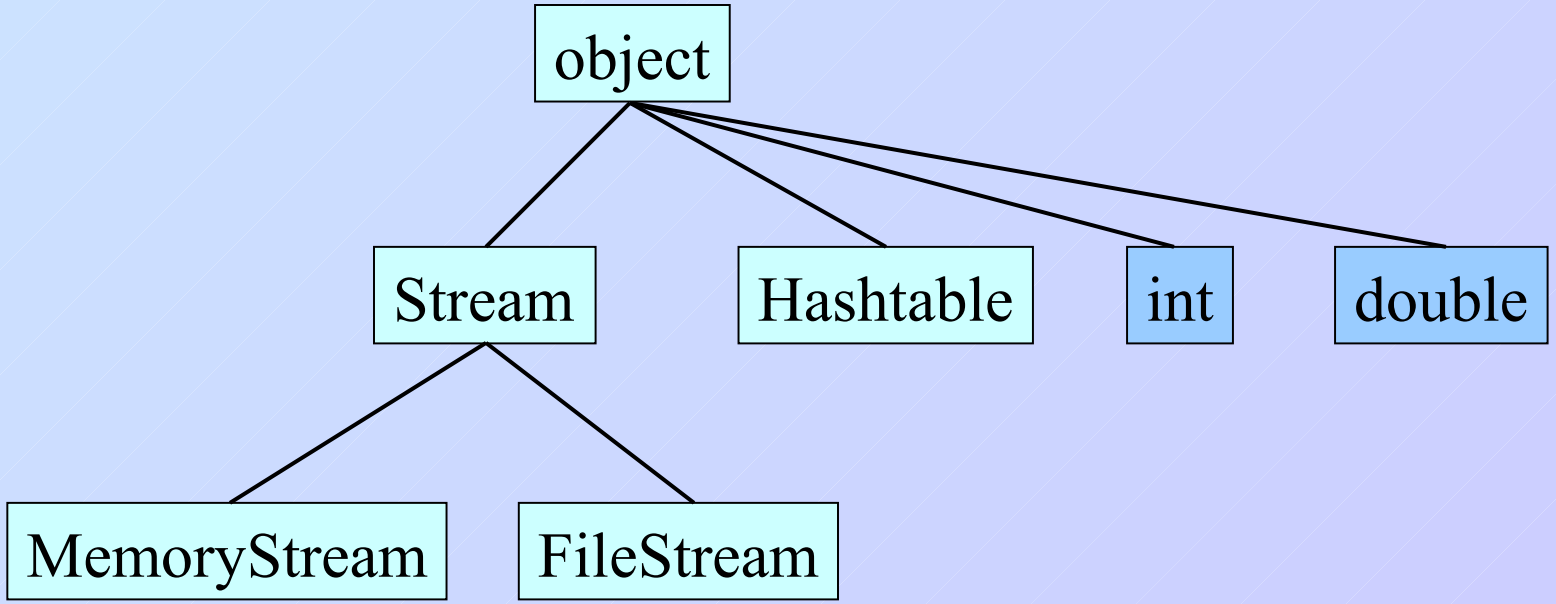
- Ponteiros para funções orientados a objetos
- Múltiplos recipientes
  - Cada delegação tem uma lista de invocação
  - Operações + e – seguras entre threads
- Base para o sistema de eventos da .NET

```
delegate void MouseEvent(int x, int y);
delegate double Func(double x);

Func func = new Func(Math.Sin);
double x = func(1.0);
```

Hierarquia de Tipos

- Tudo deriva da classe *object*
- Boxing / Unboxing



Desenvolvimento de Componentes

- O que define um componente?
  - Propriedades, métodos e eventos
  - Documentação integrada
  - Informações para desenvolvimento
- Suporte interno para estes elementos:
  - Sem nomenclatura padrão, adaptadores, etc...
  - Sem necessidade de arquivos externos

Propriedades

- Sintaxe simples, controle de acesso

```
Public class Button: Control
{
  private string caption;

  public string Caption {
    get {
      return caption;
    }
    set {
      caption = value;
      Repaint();
    }
  }
}
```

Indexadores

```
Public class ListBox: Control
{
  private string[] items;
  public string this[int index] {
    get {
      return items[index];
    }
    set {
      items[index] = value;
      Repaint();
    }
  }
}
```

Criando e Disparando Eventos

```
public delegate void EventHandler(object sender, EventArgs e);

public class Button
{
  public EventHandler Click;

  protected void OnClick(EventArgs e) {
    if (Click != null) Click(this, e);
  }
}
```

Tratando o Evento

```
public class MyForm: Form
{
  Button okButton;

  public MyForm() {
    okButton = new Button(...);
    okButton.Caption = "OK";
    okButton.Click += new EventHandler(OkButtonClick);
  }

  void OkButtonClick(object sender, EventArgs e) {
    ShowMessage("Botão Ok!");
  }
}
```

Atributos

- Associar informações a tipos e membros
  - URL de documentação para uma classe
  - Contexto de transação para um método
  - XML de mapeamento de persistência

Atributos

```
public class TrataPedidos
{
   [WebMethod]
   public void FazPedido(OrdemPedido pedido) {...}
}

[XmlRoot("Pedido", Namespace="urn:acme.b2b-schema.v1")]
public class OrdemPedido
{
   [XmlElement("endereco")] public string Endereco;
   [XmlElement("cobrar")]   public string Cobranca;
   [XmlElement("items")]    public Items[] Items;
}
```

Construções e Expressões

- if, while, do requer condição booleana
- switch sem continuar para o próximo
- foreach
- Expressões checked e unchecked
- Expressões devem executar alguma operação

Código Inseguro

- Blocos de código ou métodos podem ser marcados como *unsafe*
- Aritmética de ponteiros
- Chamada a outros métodos *unsafe*

Conclusão

- Nenhuma revolução ou grande inovação
- Coleção de diversas pequenas idéias de diversos outros lugares
- Voltado ao mercado Windows e Internet
- Influência da Microsoft não é despresível

Referências

- Programming C#, Jesse Liberty, O'Reilly & Associates
- C# Language Specification, Microsoft Corp
- C# Station, http://www.csharp-station.com/
- C# Corner, http://www.c-sharpcorner.com/
- C# Today, http://www.csharptoday.com/