Aspectos Adicionais do C# e do .Net

- Propriedades
- Excepções
- *Delegates* e Eventos
- *Threads* e Sincronização
- Atributos
- *Assemblies* e *Intermediate Language*

2ª Aula Prática de ARGE, 2003/2004

Propriedades

- Maneira simples de controlar o acesso a campos privados de classes, *structs* e interfaces
- Não podem ser passadas como variáveis

```
abstract class X {
    private string NomeX;
    private int IdadeX;
    public string Nome {
        get { return NomeX; }
        set { NomeX = value; }
    }

    public abstract int Idade {
        get;
    }
```

Excepções

- Causadas por uma situação excepcional
- São lançadas pelo sistema ou usando throw
- Usa-se colocando o código dentro de um bloco:

```
try {
   //código susceptível de gerar excepção
} catch (<TipoDaExcepção> e) {
   // tratamento da excepção e
} finally {
// é sempre executado
// limpeza de recursos alocados no try
}
```

- Podem-se criar novas excepções derivando de System.ApplicationException

Lançamento de uma Excepção

```
class MinhaExcepcao: ApplicationException {
    // métodos e atributos arbitrário
}

...

if (<condição de erro>) {
  throw new MinhaExcepcao();
}
```

Intercepção de Excepções

- É procurado, pela pilha acima, um bloco try
- Procura-se um catch para a excepção
- Se não houver, executa-se o finally
- e sobe-se pelo pilha acima, executando os finally que houver, até a excepção ser apanhada ou o programa ser terminado ☹

Excepções Típicas

- System.ArithmeticException
- System.ArrayTypeMismatchException
- System.DivideByZeroException
- System.IndexOutOfRangeException.
- System.InvalidCastException
- System.MulticastNotSupportedException
- System.NullReferenceException
- System.OutOfMemoryException
- System.OverflowException
- System.StackOverflowException
- System.TypeInitializationException

Sem Delegates: Problema

```
public class DriverBombaElectrica
{
        public void IniciarArranqueBombaElectrica( )
{...}
}

public class DriverBombaPneumatica
{
        public void Ligar( ){...}
}
```

Sem Delegates: Problema (cont.)

```
public class MonitorTempNucleo {
  private ArrayList bombas = new ArrayList();
  public void Adicionar(object bomba) { bombas.Add(bomba); }
  public void LigarTodasAsBombas(){
  foreach (object bomba in bombas) {
    if (bomba is DriverBombaElectrica) {
         ((DriverBombaElectrica)bomba).IniciarArranqueBombaElectrica();}
       if (bomba is DriverBombaPneumatica) {
         ((DriverBombaPneumatica)bomba).Ligar();}
  }
}

public class ExemploDeUso {
  public static void Main() {
    MonitorTempNucleo mtn = new MonitorTempNucleo();
       DriverBombaElectrica de1 = new DriverBombaElectrica();
       mtn.Adicionar(de1);
       DriverBombaPneumatica dp1 = new DriverBombaPneumatica ();
       mtn.Adicionar(dp1);
    mtn.LigarTodasAsBombas();
  }
}
```

Delegates

- Semelhante a apontadores para funções:

  bool (*minhaFunc) (int) /* em C */
- Apontadores para métodos de objectos ou de classes:

  delegate bool meuDelegate(int x);

  meuDelegate md =

    new meuDelegate(NomeMetAssinIgual);
- Um *delegate* vazio é igual a null.
- *Delegates* guardam uma “lista” de métodos.
- Podem ser manipulados com operações aritméticas: Combine (+), Remove (-)

Delegates

```
delegate void MyDelegate(string s);

class MyClass {
public static void Hello(string s) {
  Console.WriteLine(" Hello, {0}!", s);
}

public static void Goodbye(string s) {
  Console.WriteLine(" Goodbye, {0}!", s);
}
```

```
public static void Main() {
  MyDelegate a, b;
  a = new              MyDelegate(Hello);
  b = new
MyDelegate(Goodbye);
  a("A");
  b("B");
}
}
```

```
Hello, A!
GoodBye, B!
```

Alternativa? *Delegates*

```
public delegate void IniciarBomba( );

public class MonitorTempNucleo2 {
  public IniciarBomba bombas;
  public void Adicionar(IniciarBomba bomba) {
    bombas = bombas + bomba;
  }
}

class ExemploDeUso {
  public static void Main( ) {
    MonitorTempNucleo2 mtn = new MonitorTempNucleo2( );
    DriverBombaElectrica de1 = new DriverBombaElectrica( );
    mtn.Adicionar(new IniciarBomba(de1.ArranqueBombaElectrica));
    DriverBombaPneumatica dp1 = new DriverBombaPneumatica( );
    mtn.Adicionar(new IniciarBomba(dp1.Ligar));
    mtn.bombas( );
  }
}
```

Eventos

- Linguagem Publicação – Subscrição:
  - Classe ***Editora*** : gera um evento para avisar os objectos interessados;
  - Classe ***Subscritora*** : fornece um método que é chamado quando é gerado um evento
- A rotina invocada por um evento é um delegado:

```
public delegate void IniciarBomba( );
public event IniciarBomba Sobreaquecimento;
```

- Delegados usados para subscrever eventos têm de ser declarados como devolvendo **void**
- Só podem ser manipulados com += e -=
- Não podem ser igualados a null

Argumentos

- Convenção, um delegado subscritor aceita sempre dois argumentos:
  - 1º: o objecto que gerou o evento
  - 2º: um objecto que herde da classe EventArgs

```
public class SobreaquecimentoEventArgs: EventArgs {
   private readonly int temperatura;
   public SobreaquecimentoEventArgs(int temperatura) {
       this.temperatura = temperatura;
   }
   public int GetTemperature( ) {return temperatura;}
}
```

Subscrever um Evento

```
DriverBombaElectrica ed1 = new DriverBombaElectrica( );

DriverBombaPneumatica pd1 = new DriverBombaPneumatica( );

...

mtn.Sobreaquecimento += new
                IniciarBomba(ed1.IniciarBombaElectrica);

mtn.Sobreaquecimento += new IniciarBomba(pd1.Ligar);
```

Notificar Subscritores

```
public void LigarBombasTodas( ) {
   if (Sobreaquecimento != null) {
Sobreaquecimento( ); }
}
```

Nota: *É verificado se existe pelo menos um delegado subscritor do evento. Se for gerado um evento para o qual não haja subscritores é gerada uma excepção.*

Threads

- Quando se usam threads:
  - Várias tarefas simultâneas com partilha de dados
- Construção:

```
//ThreadStart é um public delegate void ThreadStart();
ThreadStart ts = new ThreadStart(y.xpto);
Thread t = new Thread(ts);
t.Start(); // inicia execução
t.Join(); //  espera terminação
```

- Outros métodos:Abort, Sleep, Suspend, Resume, Join

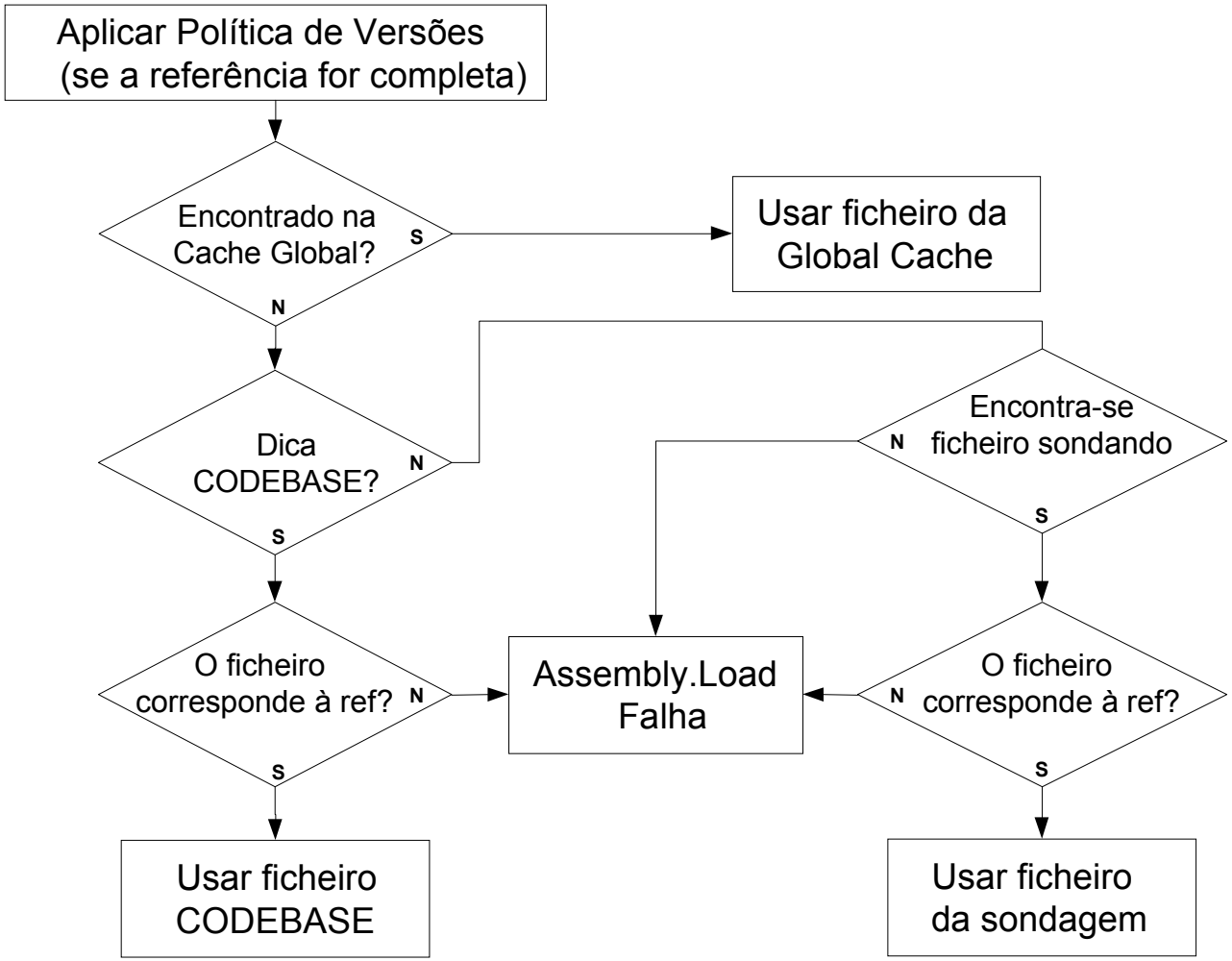
Sincronização

- Concorrência (*threads*) implicam sincronização.
- lock permite sincronizar um troço de código usando um objecto
- lock(this) nos métodos de uma classe: monitor
- lock(typeof(this)): sincroniza uma classe

Assemblies

- Unidade de partilha e instalação
- Uma colecção de tipos e recursos
- Um exe ou uma dll
- Tem código e um ponto de entrada (DllMain, Main)
- Manifesto que descreve:
  - Identificação
  - Tipos exportados
  - *Assemblies* importadas
- Inspeccionável usando o ILDASM

Encontrar *Assemblies*



Exercício 1

- Abra o VS .Net
- Procure o tutorial de delegates no help
- Estude e execute o exemplo2
- O que acontece se igualar um delegate a null antes do invocar?
- Programe a intercepção da excepção gerada
- Crie um bloco finally e observe se é ou não executado mesmo que não apanhe a excepção

Exercício 2

- Implemente numa classe um buffer circular com métodos para inserir/retirar elementos e exibir o estado do buffer.
- Implemente um programa com diversas threads (idealmente um número configurável de threads) que produzam e consumam elementos do buffer.
- Crie um interface que permita adicionar e retirar threads dinâmicamente.

Exercício 3

- Implemente o problema das bombas de arrefecimento da central nuclear, implementando as bombas numa dll em J# e a aplicação (que detecta o sobreaquecimento e liga as bombas) em C#
- Use o ILDASM para observar o conteúdo do executável e da DLL.

Eventos – Exemplo (1)

```
class NovoEmailEventArgs : EventArgs {
  string tema;
  string mensagem;

  public NovoEmailEventArgs(string tema,
                            string mensagem) {
    this.tema = tema;
    this.mensagem = mensagem;
  }

  public string Tema {
    get { return(tema); }
  }

  public string Mensagem {
    get{ return(mensagem); }
  }
}
```

```
class ReceptorDeEmail {
  public delegate void TrataNovoEmail
    (object emissor, NovoEmailEventArgs e);

  public event TrataNovoEmail HouveNovoEmail;

  public void RecebeMensagem(string tema,
                             string mensagem) {
    NovoEmailEventArgs e = new
NovoEmailEventArgs(tema, mensagem);
    if (HouveNovoEmail != null)
      HouveNovoEmail(this, e);
  }
}
```

Eventos – Exemplo (2)

```
class MostraMensagens {
  public MostraMensagens(ReceptorDeEmail
                         receptor) {
    receptor.HouveNovoEmail += new
  receptorDeEmail.TrataNovoEmail(RecebiEmail);
  }

  void RecebiEmail(object emissor,
                   NovoEmailEventArgs e){
    Console.WriteLine("Nova Mensagem:
                       {0}\n{1}", e.Tema,
                       e.Mensagem);
  }
}
```

```
class Teste {
  public static void Main(){
    ReceptorDeEmail receptor = new
                          ReceptorDeEmail();
    MostraMensagens mostrador = new
          MostraMensagens(receptor);
    receptor.RecebeMensagem("Hello!",
                  "Welcome to Events!!!");
  }
}
```

Exemplo 2 do tutorial de *delegates*

```
// compose.cs
using System;

delegate void MyDelegate(string s);

class MyClass
{
   public static void Hello(string s)
   {
      Console.WriteLine("  Hello, {0}!", s);
   }

   public static void Goodbye(string s)
   {
      Console.WriteLine("  Goodbye, {0}!", s);
   }

   public static void Main()
   {
      MyDelegate a, b, c, d;

      // Create the delegate object a that references
      // the method Hello:
      a = new MyDelegate(Hello);
      // Create the delegate object b that references
      // the method Goodbye:
      b = new MyDelegate(Goodbye);
      // The two delegates, a and b, are composed to
      form c:
      c = a + b;
      // Remove a from the composed delegate, leaving
      d,
      // which calls only the method Goodbye:
      d = c - a;

      Console.WriteLine("Invoking delegate a:");
      a("A");
      Console.WriteLine("Invoking delegate b:");
      b("B");
      Console.WriteLine("Invoking delegate c:");
      c("C");
      Console.WriteLine("Invoking delegate d:");
      d("D");
   }
}
```