



# THREADS EM C#

Prof. Alexandre Krohn

# THREADS

"Uma Thread é Um fluxo de controle sequencial isolado dentro de um programa"

- Como um programa sequencial qualquer, um thread tem um começo, um fim, e uma seqüência de comandos. Entretanto, um thread não é um programa, não roda sozinho, roda dentro de um programa.

# THREADS

- Threads permitem que um programa simples possa executar várias tarefas diferentes ao mesmo tempo, independentemente umas das outras.
- Programas multi-threaded são programas que contém vários threads, executando tarefas distintas, ao mesmo tempo.

# THREADS

Thread é um processo leve

Cada thread possui sua área de memória própria

Uma thread pode compartilhar memória com outras threads

Executada diretamente pelo sistema operacional

# THREADS

Threads são executadas concorrentemente ou pseudo-concorrentemente

- ▢ Trocas rápidas de threads no processador

Threads são suportadas em C# através do namespace `System.Threading`

- ▢ Contém as classes para a criação e sincronização de threads concorrentes

# SYSTEM.THREADING

## Classe Thread

Métodos	Descrição
<code>Abort()</code>	Aumenta <code>ThreadAbortException</code> na thread em que ele é chamado, para iniciar o processo de finalizar a thread. Chamar este método normalmente encerra a thread.
<code>Join()</code>	Bloquear a thread de chamada até que uma thread termina.
<code>Start()</code>	Faz com que o sistema operacional modifique o estado da instância atual a <code>ThreadState.Running</code> .
<code>Sleep()</code>	Suspende a thread atual por um período especificado.

# SYSTEM.THREADING

Os estados de uma Thread podem ser obtidos usando a enumeration ThreadState, que é uma propriedade da classe Thread

Estado	Descrição
Aborted	Thread já abortada
AbortRequested	Quando uma thread chama o método Abort()
Running	Rodando depois que outra thread chama o método Start()
Stopped	Depois de terminar o método Run() ou Abort()
Suspended	Thread suspensa depois de chamar o método Suspend()
Background	Rodando em background
Unstarted	Thread criada mas não iniciada
WaitSleepJoin	Quando uma thread chama Sleep() ou Join() ou quando uma outra thread chama Join()

# SYSTEM.THREADING

## Classe Thread

- Cria e controla uma thread, define a sua prioridade, e obtém seu status

Propriedade	Descrição
CurrentThread	Obtém o thread em execução no momento.
IsAlive	Obtém um valor indicando o status de execução da thread atual.
IsBackground	Obtém ou define um valor indicando se uma thread é uma thread background.
Name	Obtém ou define o nome da thread.
Priority	Obtém ou define um valor que indica a prioridade de programação de uma thread (Highest, AboveNormal, Normal, BelowNormal, Lowest)
ThreadState	Obtém um valor que contém os estados da thread atual (Unstarted, Running, Suspended, Stopped, WaitSleepJoin, ..)



# THREADS: CRIAÇÃO

## Criação de uma thread

```
Thread minhaThread = new Thread(meuMetodoThread);
```

- meuMetodoThread pode ser

- ▮ Método de classe (static) ou de instância da classe que criou a thread
- ▮ Método de classe (static) ou de instância de alguma outra classe

- meuMetodoThread deve ter a seguinte assinatura

```
void meuMetodoThread();
```

# THREADS: EXECUÇÃO

Para executar uma thread, deve-se invocar o método Start()

```
minhaThread.Start();
```

É possível enviar somente 1 parâmetro para a thread na execução da mesma

▮ O método Start() recebe o parâmetro a ser enviado

```
thread.Start(25);
```

O método a ser executado pela thread deve possuir um parâmetro da classe object para receber o valor (casting deve ser realizado)

```
void meuMetodoThreadParametro(object dado) {}
```

# THREADS: EXECUÇÃO

Após invocar o método `Start()`, o sistema inicia a execução da thread, mas não tão assíncrona para o segmento principal.

Isso significa que o programa continua a executar o código imediatamente após o método `Start()` enquanto que a thread é submetida simultaneamente a inicialização.

Para garantir que o programa não tente terminar a thread de trabalho antes que ela tem a oportunidade de executar, o programa deve aguardar até que a propriedade `IsAlive` do objeto é definida como `true`

```
while (!minhaThread.IsAlive);
```

# THREADS: EXEMPLO

```
using System;
```

```
using System.Threading;
```

```
namespace UsandoThreads_1
```

```
{
```

```
    class Program
```

```
    {
```

```
        static void Main(string[] args)
```

```
        {
```

```
            // dispara uma nova thread para executar
```

```
            Thread t = new Thread(NovaThread);
```

```
            t.Start();
```

```
            // Simultaneamente, executa uma tarefa na thread principal
```

```
            for (int i = 0; i < 10000; i++) Console.Write("1");
```

```
        }
```

```
static void NovaThread()
```

```
{
```

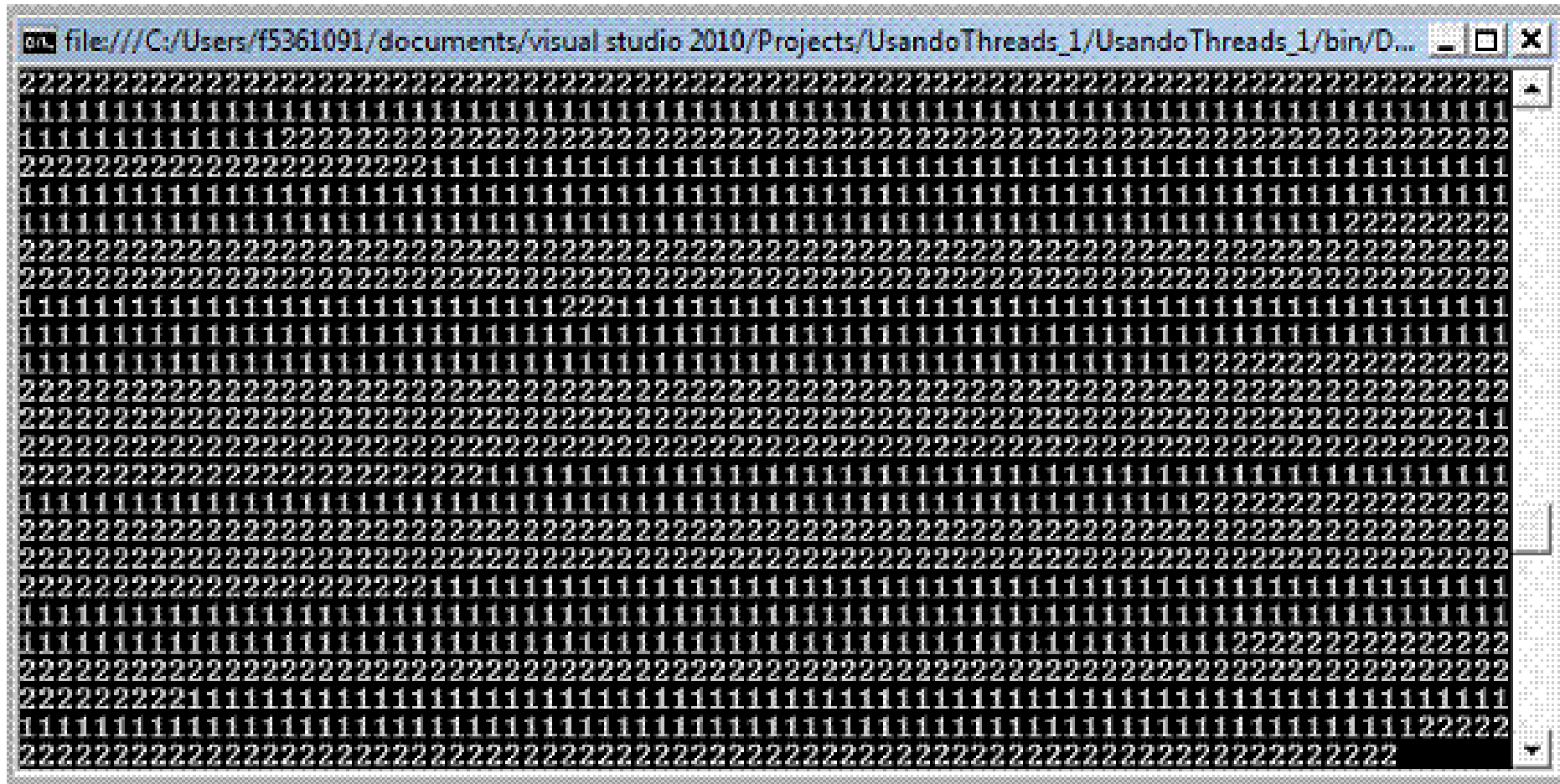
```
    for (int i = 0; i < 10000; i++) Console.Write("2");
```

```
}
```

```
}
```

```
}
```

# THREADS: EXEMPLO



The screenshot shows a Windows command prompt window with the title bar "file:///C:/Users/f5361091/documents/visual studio 2010/Projects/UsandoThreads\_1/UsandoThreads\_1/bin/D...". The window contains a large grid of characters, primarily '1's and '2's, arranged in a pattern that suggests a sequence of operations or data being processed. The grid is approximately 30 lines high and 100 columns wide. The characters are white on a black background. The pattern of '1's and '2's is complex and non-uniform, indicating a specific sequence of events or data being generated by the program.

# THREADS

Uma thread termina quando o *delegate* (o *método*) passado para o construtor da thread encerra a execução

Uma thread terminada não pode ser reiniciada

Não esqueça que controle de interface rodam na thread do programa principal e por isso devem ser manipulados através de *delegates* (ver slides sobre Timers)

# THREAD - SINCRONIZAÇÃO

**Thread safety** é um conceito existente no contexto de programas multi-threads onde um trecho de código é **thread-safe** se: *ele funcionar corretamente durante execução simultânea por vários threads.*

- ▮ Se o código não for **thread-safe** e estiver rodando em um ambiente multi-threads os resultados obtidos podem ser inesperados ou incorretos pois uma thread pode desfazer o que a outra thread já realizou.
- ▮ Esse é o problema clássico da atualização do contador em uma aplicação multi-thread.

# THREAD - SINCRONIZAÇÃO

## Solução:

- ▢ Serializar o acesso ao recurso comum de várias threads
  - ▢ Este processo de serialização chama-se de sincronização)

## Método

- ▢ Utilizar a instrução **lock()** que vai bloquear os recursos até que a thread que o está usando acabe de processá-lo.



# THREAD - SINCRONIZAÇÃO

```
class Programa
{
    static bool ok;
    static readonly object bloqueador = new object();

    ...
    void meuMetodoThread()
    {
        lock(bloqueador)
        {
            if (!ok)
            {
                // Acessando unicamente o campo ok
                ok = true;
            }
        }
    }
}
```

Quando duas threads simultâneas contêm um lock() ou bloqueio, uma thread espera, até que o bloqueio seja liberado. Neste caso, somente uma thread terá acesso ao recurso por vez e dessa forma torná-lo thread-safe.

# THREAD - SINCRONIZAÇÃO

Enquanto estiver bloqueada uma thread não consome recursos do sistema

Para aguardar o término de uma thread, pode-se utilizar o método **Join()**

```
Thread thread =  
new Thread(meuMetodoThread);  
thread.Start();  
...  
thread.Join();
```

É possível definir um tempo em milissegundos que deve ser aguardado, passando o tempo como parâmetro.