C# Avançado
Aula 08

- Revisão
- Resolver exercício
- Delegates
- Lambda
- Processo assíncrono
- Exercícios

C# Avançado - DELEGATES

- Delegate https://learn.microsoft.com/pt-br/dotnet/standard/delegates-lambdas
  - Um delegado define um tipo que representa referências aos métodos que têm uma lista de parâmetros e um tipo de retorno específicos. Um método (estático ou instância) cuja correspondência de lista de parâmetros e tipo de retorno podem ser atribuídos a uma variável desse tipo, então chamado diretamente (com os argumentos adequados) ou passado como um argumento para outro método e, então, chamado. O exemplo a seguir demonstra o uso de um delegado.

```
public class Program{
   public delegate string Reverse(string s);
   static string ReverseString(string s)    {
      return new string(s.Reverse().ToArray());
   }
   static void Main(string[] args)    {
      Reverse rev = ReverseString;
      Console.WriteLine(rev("a string"));
   }
}
```

C# Avançado - DELEGATES

- Lambda https://learn.microsoft.com/pt-br/dotnet/standard/delegates-lambdas
  - As expressões lambda ou apenas "lambdas" para abreviar, foram introduzidas no C# 3.0 como um dos principais elementos de construção da LINQ (Consulta integrada à linguagem). Elas são apenas uma sintaxe mais conveniente para usar delegados. Elas declaram uma lista de parâmetros e um corpo do método, mas não têm uma identidade formal própria, a menos que sejam atribuídas a um delegado. Ao contrário dos representantes, elas podem ser atribuídas diretamente como o lado direito do registro de eventos ou em várias cláusulas e métodos LINQ.

    Como uma expressão lambda é apenas outra maneira de especificar um delegado, nós podemos reescrever o exemplo acima para usar uma expressão lambda em vez de um delegado anônimo.

```
public class Program
{
   public delegate string Reverse(string s);
   static void Main(string[] args)
   {
      Reverse rev =  s => new string(s.Reverse().ToArray());
      Console.WriteLine(rev("a string"));
   }
}
```

C# Avançado - DELEGATES

- Delegate - Exercício

Suponha que você esteja desenvolvendo um sistema de gerenciamento de funcionários, e você deseja calcular o valor do inss com base no salário de cada funcionário. Crie um programa que utilize um delegate para calcular o inss para diferentes tipos de faixa.

As alíquotas inss são de 7,5% para aqueles que ganham até R$ 1.320,00; de 9% para quem ganha entre R$ 1.320,01 até R$ 2.571,29; de 12% para os que ganham entre R$ 2.571,30 até R$ 3.856,94; e de 14% para quem ganha de R$ 3.856,95 até R$ 7.507,29.

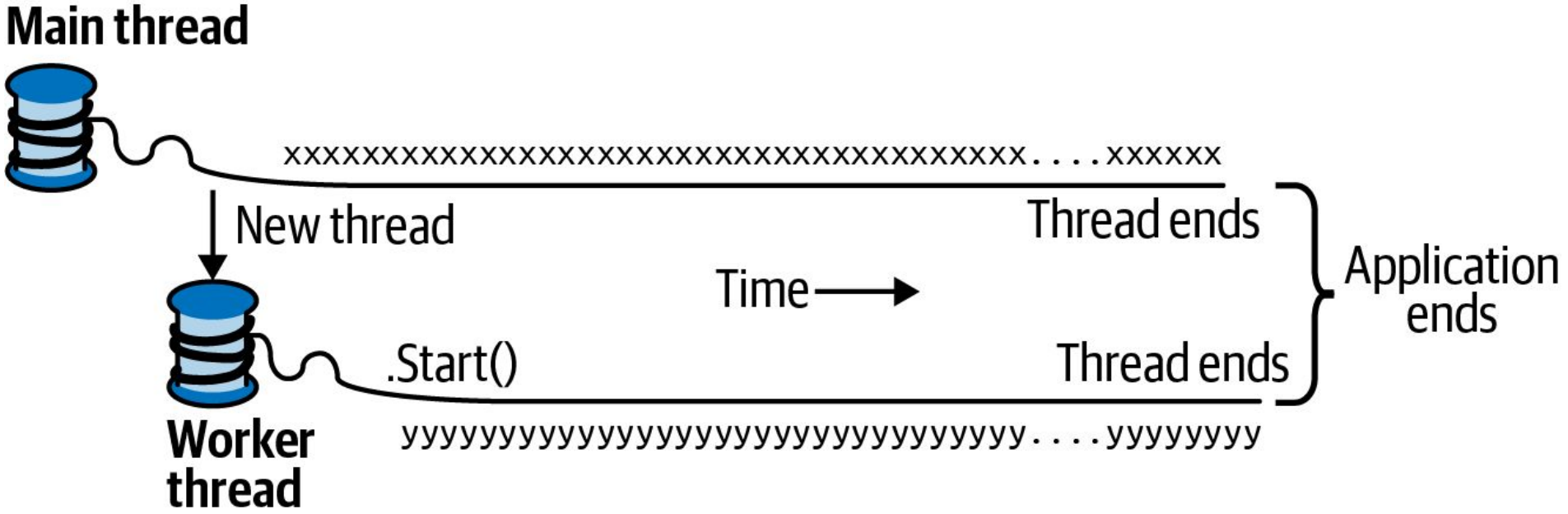
01 - Crie uma implementação usando um método para o delegate

02 - Resolve com o uso do lambda

Considere a seguinte assinatura para o delegate:

```
public delegate double CalcularInss(double salario);
```

C# Avançado Assíncrono



C# Avançado - Assíncrono

- https://learn.microsoft.com/pt-br/dotnet/csharp/asynchronous-programming/
- Thread
  - Join
  - Sleep
  - Thread safe (Lock, Semaphore)
- Task https://learn.microsoft.com/pt-br/dotnet/api/system.threading.tasks.task
  - Task.Run
  - Task.Wait
  - Task.Delay
  - Task.WhenAny
  - Task.WhenAll
  - Task.Factory
- CancelationTokenSource https://learn.microsoft.com/pt-br/dotnet/api/system.threading.cancellationtokensource
- Awaiting
- Parallelism

C# Avançado - Assíncrono

- Exercício: FizzBuzz com Threads
- O jogo FizzBuzz é um problema clássico onde os participantes contam de 1 até um determinado número, substituindo múltiplos de 3 por "Fizz", múltiplos de 5 por "Buzz" e múltiplos de ambos por "FizzBuzz". Neste exercício, você implementar uma versão concorrente do FizzBuzz usando threads.
- Crie uma aplicação em C# que utilize threads para contar de 1 até 100.
- Utilize três threads separadas para tratar os múltiplos de 3, 5 e ambos (3 e 5). Cada thread deve imprimir "Fizz", "Buzz" ou "FizzBuzz", respectivamente.

Próximo slide contém código

Implemente usando THREAD e TASK

C# Avançado - FizzBuzz com Threads

```
using System;
using System.Threading;
class Program
{
    static object lockObject = new object();
    static int currentNumber = 1;
    static int maxNumber = 100;
    static void Main()
    {
        Thread fizzThread = new Thread(Fizz);
        Thread buzzThread = new Thread(Buzz);
        Thread fizzBuzzThread = new Thread(FizzBuzz);

        fizzThread.Start();
        buzzThread.Start();
        fizzBuzzThread.Start();

        fizzThread.Join();
        buzzThread.Join();
        fizzBuzzThread.Join();
    }

    static void Fizz()
    {
        while (true)
        {
            lock (lockObject)
            {
                if (currentNumber > maxNumber)
                    break;
                if (currentNumber % 3 == 0 && currentNumber % 5 != 0)
                {
                    Console.WriteLine("Fizz");
                    currentNumber++;
                }
            }
            Thread.Sleep(10);
        }
    }
    static void Buzz()
    {
        // Implementação semelhante para múltiplos de 5 (Buzz)
    }
```

C# Avançado - Assíncrono

- Exercício
- Vamos modificar o nosso projeto da corrida de cachorros, para os cachorros correm todos ao mesmo tempo.
- Também é necessário alterar a maneira de verificar os ganhadores, para isso agora vamos usar o DateTime e TimeSpan, para poder verificar o tempo que cada cachorro levou para percorrer os 100 metros
- Deve também agora mostrar o progresso da corrida, para isso para cada movimento de cachorro fazer esperar 100 milissegundos para poder mover novamente.

SENAI

Serviço Nacional de Aprendizagem Industrial

PELO FUTURO DO TRABALHO

0800 048 1212 **sc.senai.br**

Rodovia Admar Gonzaga, 2765 - Itacorubi - 88034-001 - Florianópolis, SC