



SENAI

Enum & Struct

Fundamentos de Programação Orientada a Objeto

Enum & Struct

Estruturas em C#

- ✓ Vamos tratar de dois assuntos importantes:
 - ✓ Enum e;
 - ✓ Struct.
- ✓ Primeiro vou explicar o Enum – Enumeradores e depois Struct. Ok!



Enum

Definição

- ✓ Enum ou enumeradores são constantes fortemente tipadas;
- ✓ Conjunto de constantes;
- ✓ São estáticas, ou seja, não necessitam de instância – new;
- ✓ Declarar na namespace para que todas as classes possam acessar;
- ✓ Não podem ser declarados dentro de métodos;
- ✓ Aceita os seguintes modificadores de acesso:
 - ✓ `protected`, `private` e `public`.

Enum

Vantagens

- ✓ É útil quando precisamos criar estruturas que serão **pouco** alteradas no projeto;
- ✓ Oferece ao desenvolvedor mais facilidade de manutenção do código.



Enum

Sintaxe

```
public enum <nome>
```

```
{
```

```
    item1,
```

```
    item2,
```

```
    item3
```

```
};
```

“;” é opcional

```
enum <nome> { item1, item2, item3 };
```

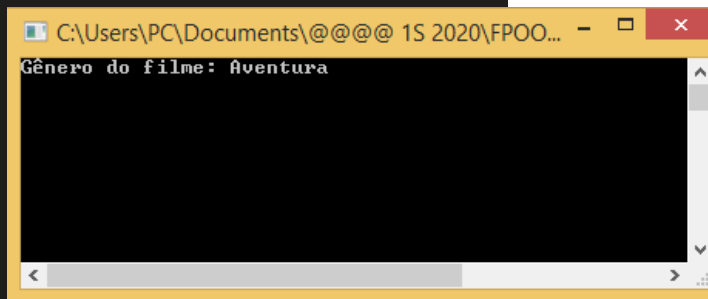
Neste exemplo criamos um enum com o nome **Generos** com alguns valores definidos.

```
1  using System;
2
3  namespace CA_Enum
4  {
5      0 referências
6      public enum Generos
7      {
8          Aventura,
9          Policial,
10         Ficção,
11         Comédia
12     }
13     0 referências
14     class Program
15     {
16         0 referências
17         static void Main(string[] args)
18         {
19             Console.WriteLine("Hello World!");
20         }
21     }
22 }
```

```
1  using System;
2
3  namespace CA_Enum
4  {
5      0 referências
6      public enum Generos { Aventura, Policial, Ficção, Comédia }
7      0 referências
8      class Program
9      {
10         0 referências
11         static void Main(string[] args)
12         {
13             Console.WriteLine("Hello World!");
14         }
15     }
16 }
```

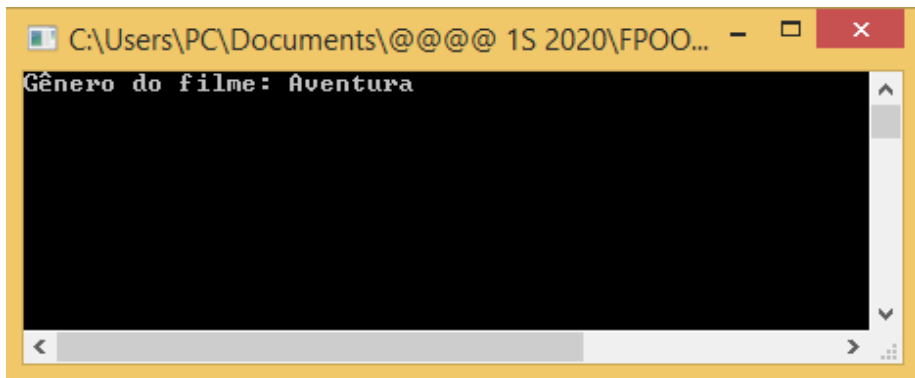


```
1  using System;
2
3  namespace CA_Enum
4  {
5      2 referências
6      public enum Generos {
7          Aventura,
8          Policial,
9          Ficção,
10         Comédia
11     }
12
13     0 referências
14     class Program
15     {
16         0 referências
17         static void Main(string[] args)
18         {
19             //variavel que recebe um valor do enum - setada com Aventura
20             Generos valor = Generos.Aventura;
21
22             Console.WriteLine("Gênero do filme: " + valor);
23             Console.ReadKey();
24         }
25     }
26 }
```



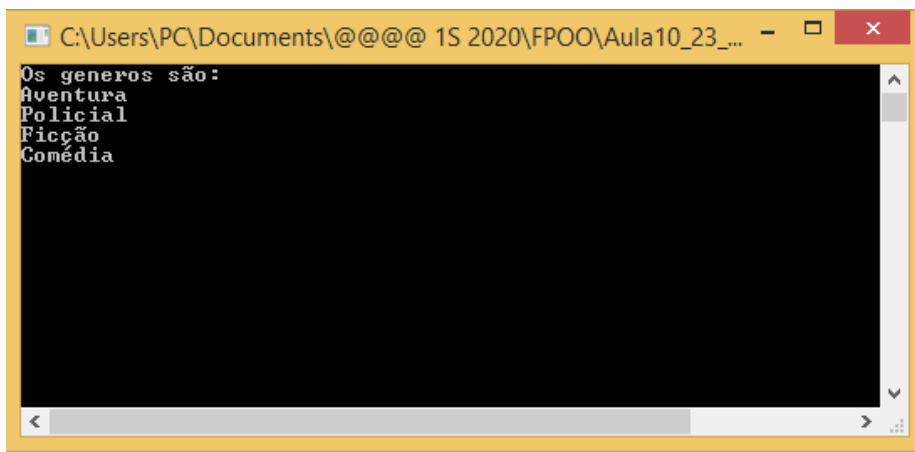
Linha 16: Declara uma variável para receber o nosso enum com o nome **valor** e logo depois essa variável é **setada** com o valor “**Aventura**”.

Observem as duas saídas ao lado seguir:



```
C:\Users\PC\Documents\@@@ 1S 2020\FPOO...  
Gênero do filme: Aventura
```

Nesta primeira exibimos somente **UM** dos elementos do enum



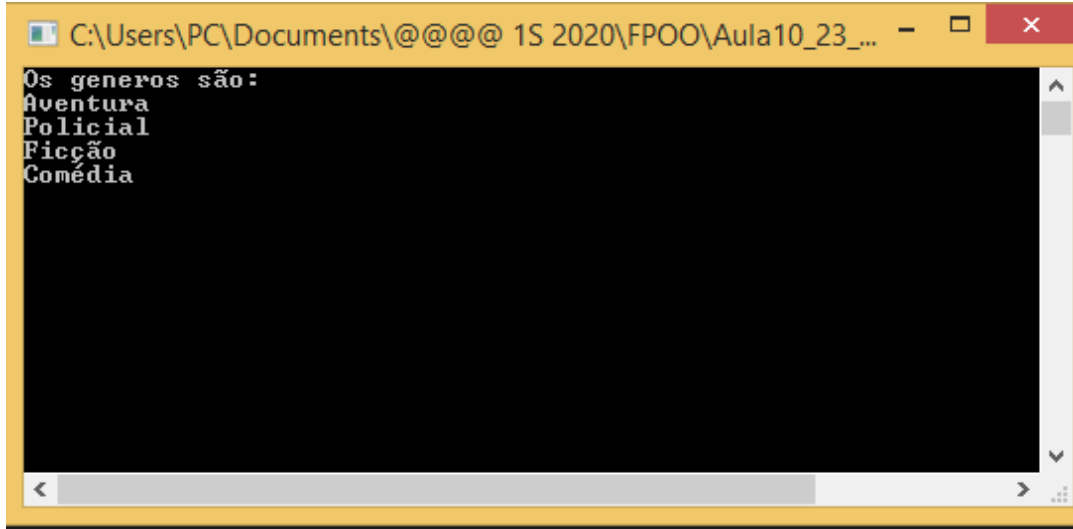
```
C:\Users\PC\Documents\@@@ 1S 2020\FPOO\Aula10_23_...  
Os generos são:  
Aventura  
Policial  
Ficção  
Comédia
```

Nesta segunda exibimos **TODOS** os elementos do enum

Entenderam a diferença?

Vocês conseguem imaginar a solução para a segunda saída?

Bem, vamos lá...



```
C:\Users\PC\Documents\@@@ 1S 2020\FPOO\Aula10_23_... - [X]  
Os generos são:  
Aventura  
Policial  
Ficção  
Comédia
```

- ✓ Para exibir todos os elementos do enum vamos usar um **loop**, Ok!
- ✓ Nesse caso vamos usar a estrutura de repetição chamada **foreach(){}**
- ✓ Lembrando que já vimos algumas estruturas de repetição tais como:
 - ✓ `for() {}`
 - ✓ `while(){}`
 - ✓ `do ... while().`

```
1  using System;
2
3  namespace CA_Enum
4  {
5      1 referência
      public enum Generos { Aventura, Policial, Ficção, Comédia }
6      0 referências
      class Program
7      {
8          0 referências
          static void Main(string[] args)
9          {
10             //Iterando através de uma enumeração
11             Console.WriteLine("Os generos são: ");
12
13             foreach (var gen in Enum.GetNames(typeof(Generos)))
14             {
15                 Console.WriteLine(gen);
16             }
17
18             Console.ReadKey();
19         }
20     }
21 }
```

A classe **System.Enum** fornece a classe base para enumerações e o método **Enum.GetNames()**.

GetNames() - recupera uma matriz dos **nomes** das constantes em uma enumeração especificada e retorna um **array de strings de nomes**.

Este método requer que seja passado o tipo enum como parâmetro usando a palavra chave **typeof()**

Na linha 13 temos uma estrutura de repetição (**foreach**) que vai iterar todos os elementos do **enum - Generos** e guardar em **gen** para serem exibidos enquanto a iteração ocorrer.

Get**Names**() - recupera uma matriz dos **nomes** das constantes em uma enumeração especificada e retorna um **array de strings de nomes**.

E Get**Values**() - recupera o que?

Qual a diferença entre os dois métodos?

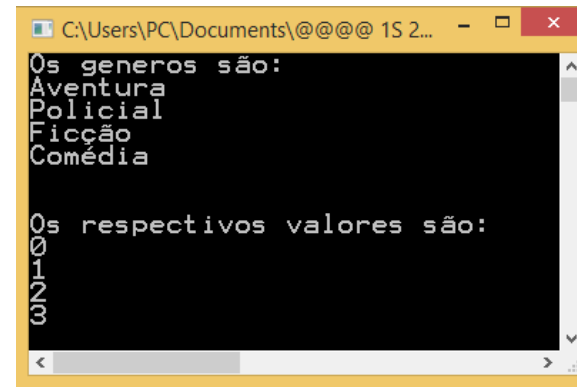
```
5 public enum Generos {  
6     Aventura,  
7     Policial,  
8     Ficção,  
9     Comédia  
10 }
```

O método Enum.Get**Values**() - retorna um array de **valores inteiros** para cada item da enum.

Quando definimos uma enumeração, todos os elementos do enumerador terão valores atribuídos a partir do zero (0).

Sendo assim, **Aventura** = **0**, **Policial** = **1**, **Ficção** = **2** **Comédia** = **3**

```
9 static void Main(string[] args)
10 {
11     //Iterando através de uma enumeração
12     Console.WriteLine("Os generos são: ");
13
14     foreach (var gen in Enum.GetNames(typeof(Generos)))
15     {
16         Console.WriteLine(gen);
17     }
18
19     //Iterando através de uma enumeração
20     Console.WriteLine("\n\nOs respectivos valores são: ");
21
22     foreach (var valor in Enum.GetValues(typeof(Generos)))
23     {
24         Console.WriteLine(Convert.ToInt32(valor));
25     }
26
27     Console.ReadKey();
28 }
```



The screenshot shows a console window with the following output:

```
Os generos são:
Aventura
Policial
Ficção
Comédia

Os respectivos valores são:
0
1
2
3
```



O método `Enum.GetValues()` – retorna um array de **valores inteiros** para cada item da enum.

Sendo assim, Aventura = 0, Policial = 1, Ficção = 2 Comédia = 3

Pensem nessa enumeração como pares de valores **Nome** e **Valor**

Nome = “Aventura” e **Valor = 0**

```
C:\Users\PC\Documents\@@@ 1S 2020\FPOO\Aula10_23_Jun_2020\CA_En...
Os generos são:
Aventura
Policial
Ficção
Comédia

Os respectivos valores são:
0
1
2
3

Os respectivos nomes e valores são:
Aventura = 0
Policial = 1
Ficção = 2
Comédia = 3
```

nomes

valores

Observem a saída ao lado com nomes e valores devidamente formatados...

```
Console.WriteLine("\n\nOs respectivos nomes e valores são: ");
foreach (string dados in Enum.GetNames(typeof(Generos)))
{
    Console.WriteLine("{0, -10} = {1}", dados,
        Enum.Format(nome_valor,
            Enum.Parse(nome_valor, dados), "d"));
}
```

Alinhamento de controle

```
Console.WriteLine("\n\nOs respectivos nomes e valores são: ");  
foreach (string dados in Enum.GetNames(typeof(Generos)))  
{  
    Console.WriteLine("{0, -10} = {1}", dados,  
        Enum.Format(nome_valor,  
        Enum.Parse(nome_valor, dados), "d"));  
}
```

Por padrão, as cadeias de caracteres são alinhadas à direita no campo se você especificar uma largura de campo.

Para alinhar as cadeias de caracteres de um campo à esquerda, você precede a largura do campo com um sinal negativo, como {0, -10} para definir um campo alinhado à esquerda de 10 caracteres.

<https://docs.microsoft.com/pt-br/dotnet/api/system.string.format?view=netcore-3.1>

Exercícios de fixação

1. Vamos criar um projeto console CA_EFEnums e criar 3 enumeradores para:

Cores;
Estações do ano;
Meses do ano;

e exibir seus nomes e valores.

2. Desenvolver um aplicativo Desktop – Windows Forms para exibir os dados de um enumerador referente ao Tipo Pessoa (física, jurídica ou ambas) conforme a tela a seguir.

The screenshot shows a Windows Forms application window titled "Enum - Pessoa". Inside the window, there is a button labeled "Ler tipos pessoa". Below the button is a list box containing three items: "Física", "Jurídica", and "Ambas". The item "Ambas" is currently selected and highlighted in blue. To the right of the list box, there are two text boxes. The first text box is labeled "Descrição do Enum Selecionado:" and contains the text "Ambas". The second text box is labeled "Valor selecionado:" and contains the number "3".

Struct

Definição

- ✓ Muito semelhante a uma classe;
- ✓ Classe é o tipo de referência, enquanto struct é o tipo de valor
- ✓ Os Structs são tipos de dados definidos pelo usuário e que consistem em um conjunto de variáveis simples;
- ✓ Possuem aplicabilidades bem específicas;
- ✓ Não aceita construtor vazio - obrigado a ter parâmetros;
- ✓ Usando estrutura não podemos inicializar campos de instância em seu ponto de declaração, o que é possível nas classes;
- ✓ Podem conter construtores, constantes, campos, métodos, propriedades, etc.

Struct

Quando usar?

- ✓ Mas, quando seria recomendável utilizar struct ao invés de classe?
 - ✓ em situações em que se necessita apenas atribuição de valor;
 - ✓ no lugar de classes que executam operações muito simples:
 - ✓ Conversões de temperaturas;
 - ✓ Coordenadas;
 - ✓ Etc.

Struct

Anatomia

```
<Modificadores> struct <struct_name>
{
    // membros Estrutura
}
```

```
public struct Pessoa
{
    public string Nome;
    public int Idade;
}
```

✓ Usamos estruturas a maior parte do tempo

✓ Os tipos numéricos primitivos int, long, short e float são aliases - apelidos, referências - para as estruturas:

- short - System.Int16
- int - System.Int32
- long - System.Int64

Exemplo 1



```

1  using System;
2
3  namespace CA_Estruturas
4  {
5      0 referências
6      class Program
7      {
8          1 referência
9          public struct Data
10         {
11             1 referência
12             public Data(int objDia, int objMes, int objAno)
13             {
14                 this.dia = objDia;
15                 this.mes = objMes;
16                 this.ano = objAno;
17             }
18         }
19     }
20 }

```

C:\Users\PC\Documents\@@@ 1S 202

```

Data de Hoje: 30/1/2011
Data de Ontem: 29/1/2011
Data de Amanhã: 31/1/2011

```

```

18
19 0 referências
20 static void Main(string[] args)
21 {
22     //Instancio meu Struct, passando os parâmetros referentes ao dia, mês e ano e exibo ao usuário
23     Data objDataHoje = new Data(30, 01, 2011);
24
25     Console.WriteLine("\nData de Hoje: " + objDataHoje.dia + "/" + objDataHoje.mes + "/" + objDataHoje.ano + "\n");
26
27     //Atribuo a variável objDataHoje à variável objDataOntem, decremento o valor em 1 e exibo ao usuário
28     Data objDataOntem = objDataHoje;
29
30     objDataOntem.dia--;
31
32     Console.WriteLine("Data de Ontem: " + objDataOntem.dia + "/" + objDataOntem.mes + "/" + objDataOntem.ano + "\n");
33
34     //Atribuo a variável objDataHoje à variável objDataAmanha, incremento o valor em 1 e exibo ao usuário
35     Data objDataAmanha = objDataHoje;
36
37     objDataAmanha.dia++;
38
39     Console.WriteLine("Data de Amanhã: " + objDataAmanha.dia + "/" + objDataAmanha.mes + "/" + objDataAmanha.ano + "\n");
40
41     Console.ReadKey();
42 }
43
44 }
45

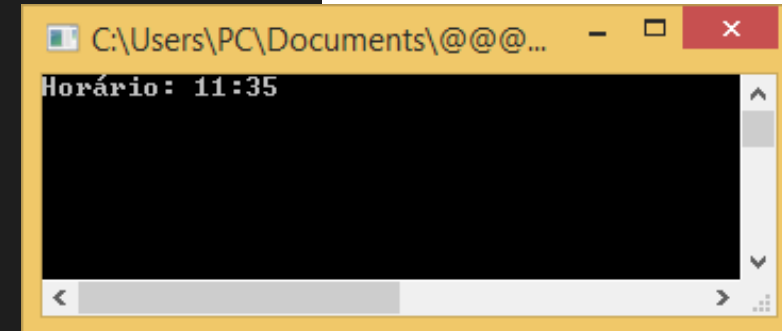
```

Exemplo 2



JANEIRO 2020 — SENAI-SP

```
1  using System;
2
3  namespace CA_Estruturas
4  {
5      0 referências
6      class Program
7      {
8          1 referência
9          public struct Horario
10         {
11             public int hora, minuto, segundo;
12         }
13
14         0 referências
15         static void Main(string[] args)
16         {
17             Horario agora;
18             agora.hora = DateTime.Now.Hour;
19             agora.minuto = DateTime.Now.Minute;
20             Console.WriteLine("Horário: {0}:{1}", agora.hora, agora.minuto);
21
22             Console.ReadKey();
23         }
24     }
25 }
```



Struct requer menos memória que uma classe, instancia mais rapidamente.



É considerada uma boa prática de programação para objetos menores, mas somente para objetos menores!

Exemplo 3



JANEIRO 2020 — SENAI-SP

```
7 public struct Horario
8 {
9     private int hora, minuto, segundo;
10
11     1referência
12     public Horario(int h, int m, int s)
13     {
14         this.hora = h % 24;
15         this.minuto = m % 60;
16         this.segundo = s % 60;
17     }
18
19     1referência
20     public int Hora()
21     {
22         return this.hora;
23     }
24
25     1referência
26     public int Minuto()
27     {
28         return this.minuto;
29     }
30
31     1referência
32     public int Segundo()
33     {
34         return this.segundo;
35     }
36 }
```

Linha 07: Criamos uma struct Horario com o modificador de acesso public;

Linha 09: Declarei três variáveis do tipo int e com o modificador de acesso private - **encapsulamento**;

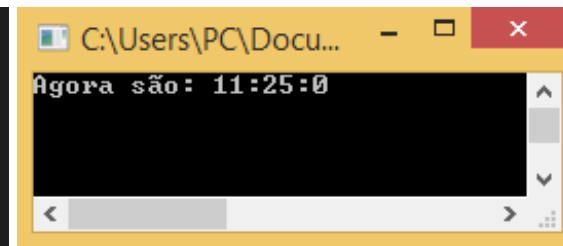
Linhas 11 – 16: Construtor recebendo três parâmetros de fora da estrutura – struct – e “setando” os três atributos;

Linhas 18 a 30: Implementamos o método GET;

Linha 35: Criamos a variável de instância agora – struct;

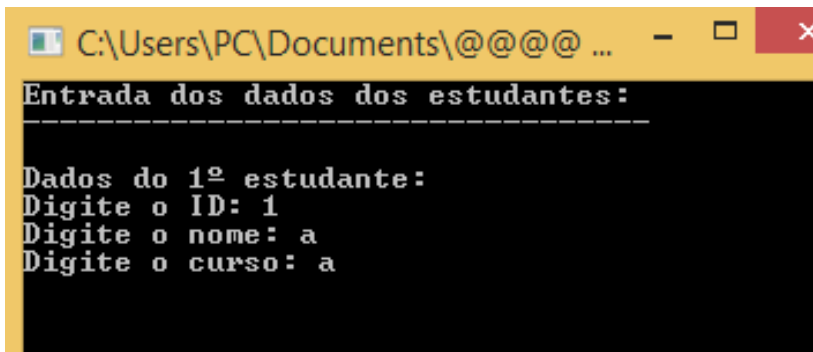
Linha 36: Exibimos a hora, o minuto e o segundo.

```
33 static void Main(string[] args)
34 {
35     Horario agora = new Horario(11, 25, 00);
36     Console.WriteLine("Agora são: {0}:{1}:{2}", agora.Hora(), agora.Minuto(), agora.Segundo());
37
38     Console.ReadKey();
39 }
```



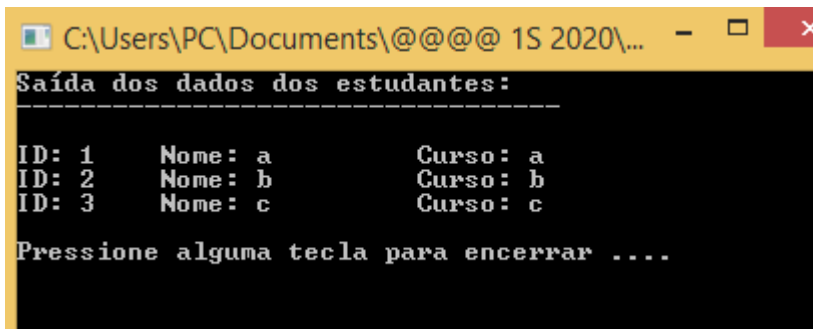
Exercício – 30 min

Desenvolver uma aplicação console que leia o ID, nome e curso de três estudantes conforme tela a seguir. A cada entrada, limpar a tela para solicitar a entrada do próximo aluno.



```
C:\Users\PC\Documents\@@@@@ ...  
Entrada dos dados dos estudantes:  
-----  
Dados do 1º estudante:  
Digite o ID: 1  
Digite o nome: a  
Digite o curso: a
```

e em seguida exiba os dados na tela. Implementar o código com struct e Array



```
C:\Users\PC\Documents\@@@@@ 1S 2020\...  
Saída dos dados dos estudantes:  
-----  
ID: 1      Nome: a      Curso: a  
ID: 2      Nome: b      Curso: b  
ID: 3      Nome: c      Curso: c  
  
Pressione alguma tecla para encerrar ....
```



Atila Andreatti Olivi

atila.olivi@senaisp.edu.br

(11) 5642-3400

Escola SENAI Suíço Brasileira “Paulo E. Tolle”

R. Bento Branco de Andrade Filho, 379