**Aula 02 : Iniciando com .NET**

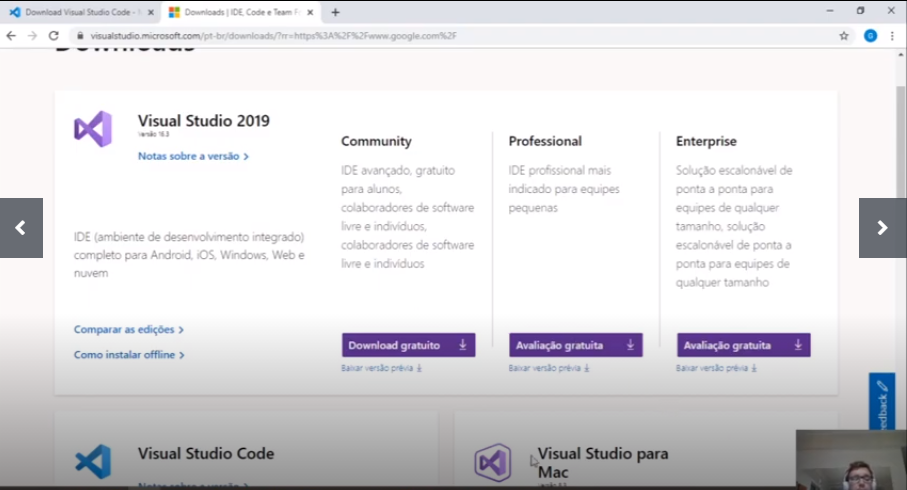
**Preparando o Ambiente**

Objetivos da aula:

1. Preparar o ambiente;
2. Conhecer o CLI do .NET;
3. Criar uma aplicação console.

1 – A versão 3.1 do DotNet é ***LTS*** (Long Term Suport). Ou seja, ela tem suporte ou será mantida por 3 anos.

O VS Code é um editor de texto não é uma IDE. Já o Visual Studio, mostrado abaixo, é uma IDE. Só que diferente do VS Code, que roda em qualquer sistema operacional, o Visual Studio para Windows roda apenas em Windows e o Visual Studio para Mac, roda apenas no Mac.

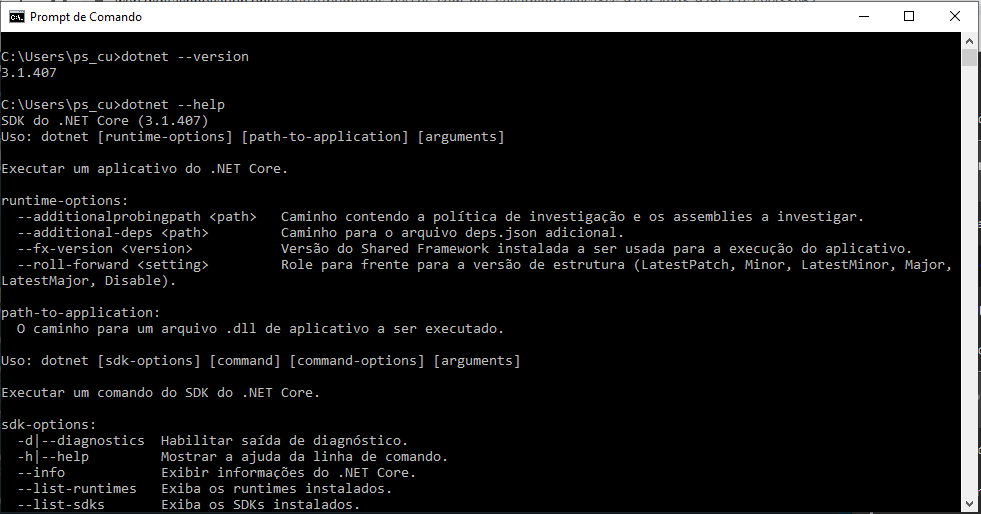


**Conhecendo a CLI do .NET**

Alguns comandos úteis que podem ser digitados no Prompt de Comando do Windows ou no Git Bash a fim de explorarmos melhor a CLI do .NET são:

dotnet - -version: Exibe a versão do .NET que está instalada em sua máquina;

dotnet - -help: Exibe todos os comandos e demais informações úteis para se trabalhar com o .NET.



A descrição completa do comando “dotnet - -help” do .NET executada no propt de comando do Windows está descrita abaixo:

SDK do .NET Core (3.1.407)

Uso: dotnet [runtime-options] [path-to-application] [arguments]

Executar um aplicativo do .NET Core.

runtime-options:

--additionalprobingpath <path> Caminho contendo a política de investigação e os assemblies a investigar.

--additional-deps <path> Caminho para o arquivo deps.json adicional.

--fx-version <version> Versão do Shared Framework instalada a ser usada para a execução do aplicativo.

--roll-forward <setting> Role para frente para a versão de estrutura (LatestPatch, Minor, LatestMinor, Major, LatestMajor, Disable).

path-to-application:

O caminho para um arquivo .dll de aplicativo a ser executado.

Uso: dotnet [sdk-options] [command] [command-options] [arguments]

Executar um comando do SDK do .NET Core.

sdk-options:

-d|--diagnostics Habilitar saída de diagnóstico.

-h|--help Mostrar a ajuda da linha de comando.

--info Exibir informações do .NET Core.

--list-runtimes Exiba os runtimes instalados.

--list-sdks Exiba os SDKs instalados.

--version Exibir todas as versões do SDK do .NET Core instaladas na máquina inclusive a versão corrente.

Comandos do SDK:

add Adicionar um pacote ou uma referência a um projeto do .NET.

build Criar um projeto do .NET.

build-server Interagir com servidores iniciados por um build.

clean Limpar as saídas do build de um projeto do .NET.

help Mostrar a ajuda da linha de comando.

list Listar as referências de um projeto do .NET.

msbuild Executar comandos do MSBuild (Microsoft Build Engine).

new Criar um novo projeto ou arquivo do .NET.

nuget Fornece comandos adicionais do NuGet (Gerenciador de pacotes para .NET)

pack Criar um pacote do NuGet.

publish Publicar um projeto do .NET para implantação.

remove Remover um pacote ou uma referência de um projeto do .NET.

restore Restaurar as dependências especificadas em um projeto do .NET.

run Criar e executar uma saída de projeto do .NET.

sln Modificar os arquivos da solução do Visual Studio.

store Armazenar os assemblies especificados no repositório de pacotes do runtime.

test Executar testes de unidade usando o executor de testes especificado em um projeto do .NET.

tool Instalar ou gerenciar ferramentas que ampliam a experiência do .NET.

vstest Executar comandos do VSTest (Microsoft Test Engine).

Comandos adicionais de ferramentas em pacote:

dev-certs Crie e gerencie certificados de desenvolvimento.

fsi Iniciar F# Interativo / executar scripts do F#.

sql-cache Ferramentas de linha de comando do cache do SQL Server.

user-secrets Gerencie segredos do usuário de desenvolvimento.

watch Inicie um observador de arquivo que executa um comando quando os arquivos são alterados.

Execute 'dotnet [command] --help' para obter mais informações sobre um comando.

**Criação de uma aplicação console com C#**

Uma aplicação console nada mais é do que uma aplicação que vai ser executada em algum terminal tal como o CMD (Propt Comando) ou Git Bash.

Comando para a criação de uma aplicação console no DotNet: “***dotnet new -n + Nome da Pasta ou do projeto”***.

Abri o Visual Code na pasta em que se encontra o projeto: ***code + [Espaço] + (.) ponto***. Basta digitar essa informação no Terminal do Windows (Prompt de comando) ou no Terminal do GitBash.

Segue a sintaxe apresentada pela linguagem C# quando iniciamos um console ou projeto:

***“using System;***

***namespace Revisão***

***{***

***class Program***

***{***

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

***{***

***Console.WriteLine("Hello World!");***

***}***

***}***

***}”***

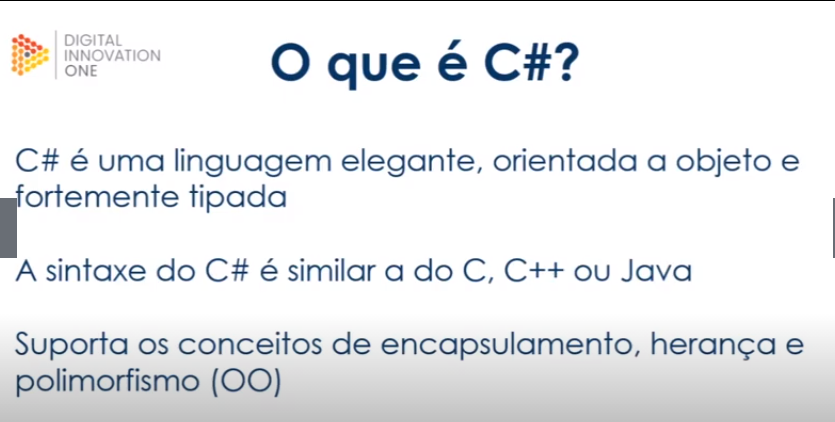
A palavra “***Revisão***”, mostrada acima, é o nome dado ao novo console/Projeto/Pasta que acabou de ser criado.

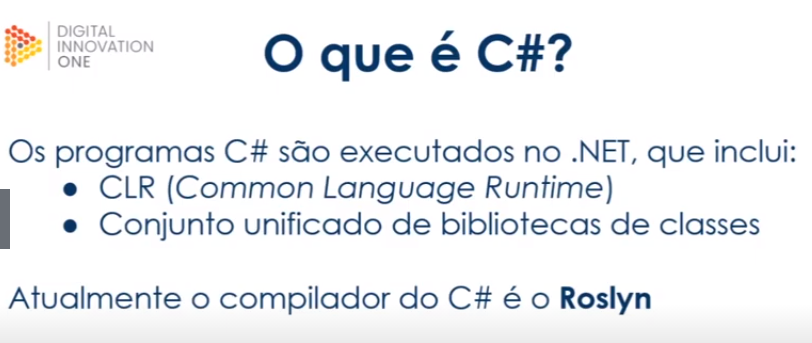
O comando “***Console.WriteLine***” “Imprime” ou exibe a frase que digitarmos logo após ele e já faz a quebra de linha. No caso acima, ele exibirá a palavra “*Hello World!*”. A expressão que deverá ser exibida por esse comando deverá sempre estar entre aspas duplos.

**Aula 03 : Conhecendo o C#**

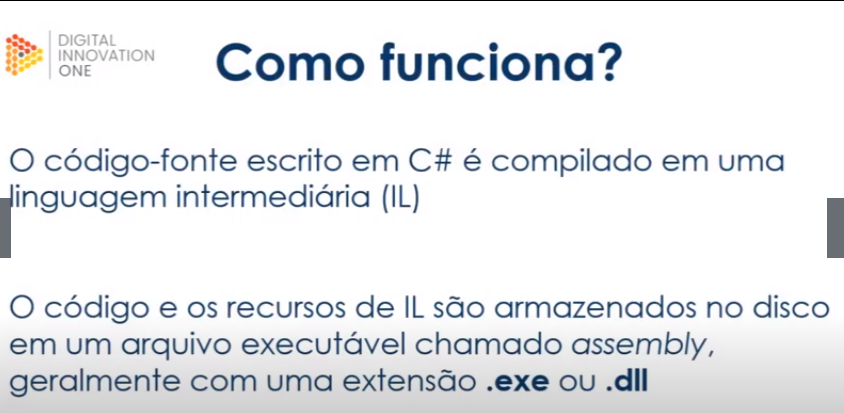
**1. O que é o C#**

A linguagem C# foi criada no final da década de 90 por Anders Rysberg. É a linguagem oficial do DotNet. Ao longo dos anos, vem sendo desenvolvida junto com o DotNet.

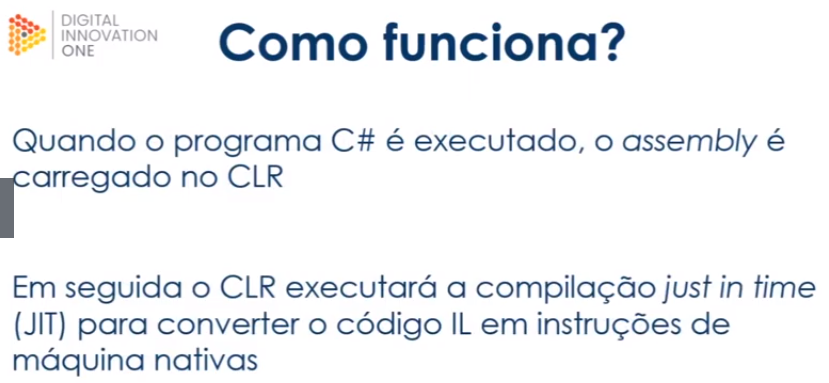




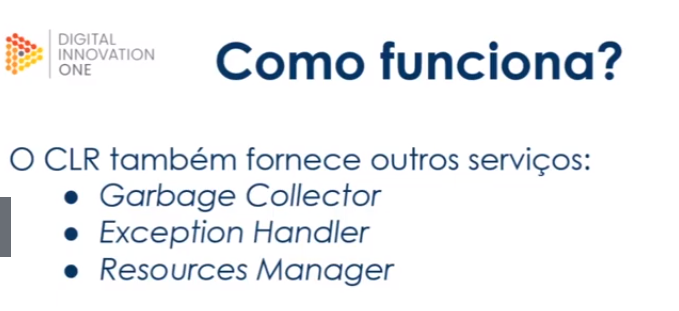
**OBS.:** O Roslyn, compilador da linguagem C#, é escrito em C#.



O comando “***dotnet build***”, quando executado, transforma o código (compila), escrito na linguagem C#, considerada uma linguagem de alto nível, em uma linguagem intermediária. A linguagem intermediária, nesse coso é a DLL. Ela contém o código escrito em C#, linguagem de alto nível, na forma de linguagem Intermediária. O próximo passo é passar esse código da linguagem intermediária para a linguagem de máquina ou de baixo nível.



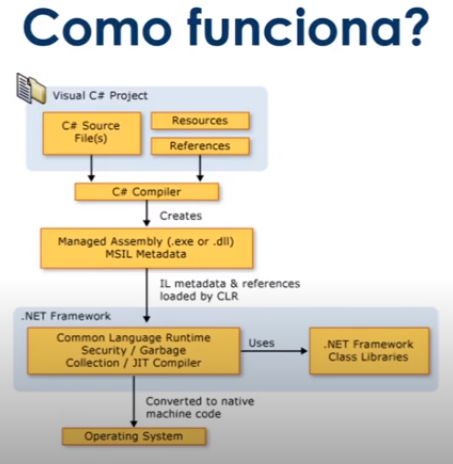
**CLR:** Common Languange Runtime.

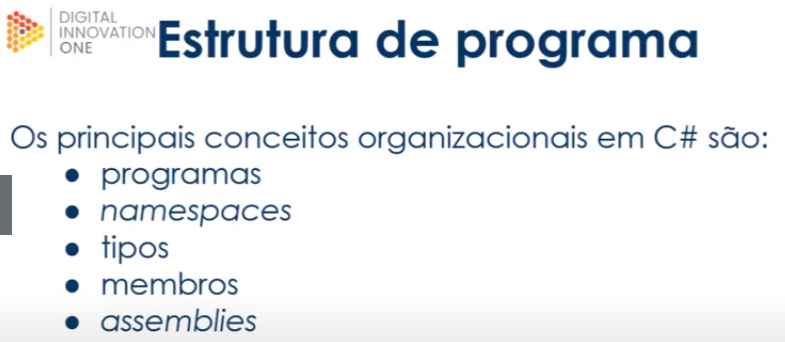


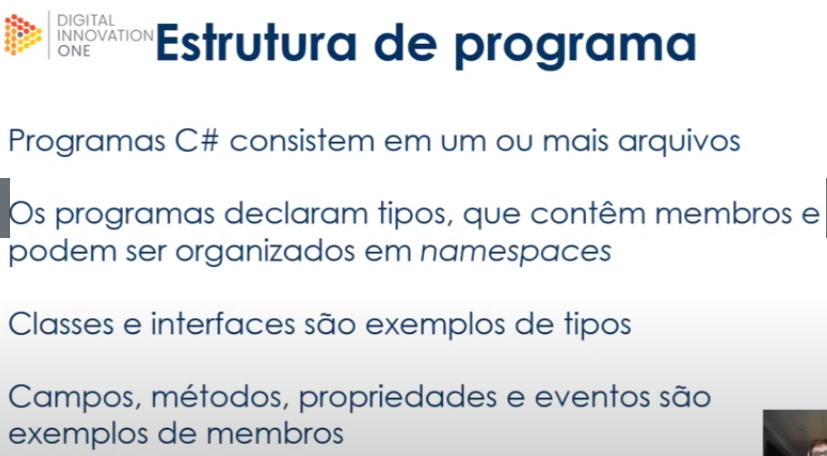
**Garbage Collector:** Literalmente significa “coletor de lixo”. Ele valida ou verifica objetos em memória que não estão sendo mais utilizados e desaloca esses objetos da memória a fim de liberar espaço.

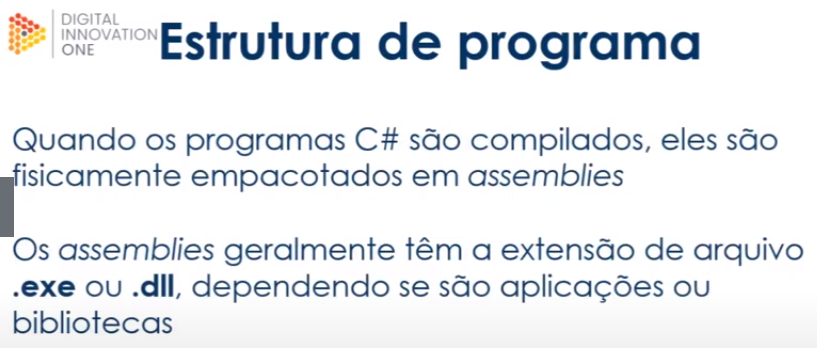
**Exception Handler:** Manipulador de Exceções. Recebe e controla as exceções que ocorrem de erros no código durante a execução.

**Resources Manager:** Gerente de Recursos.







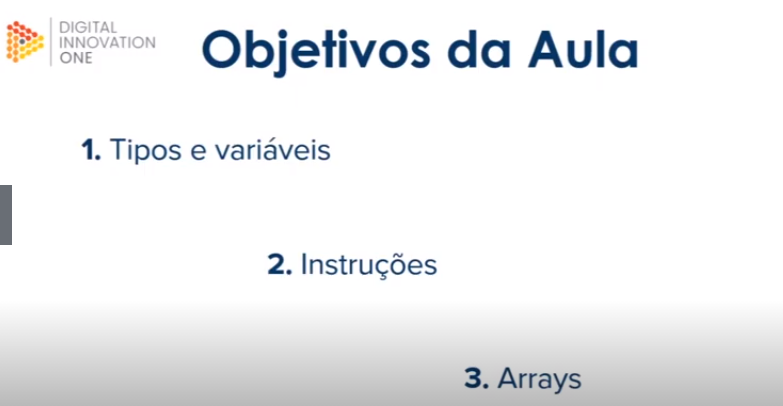


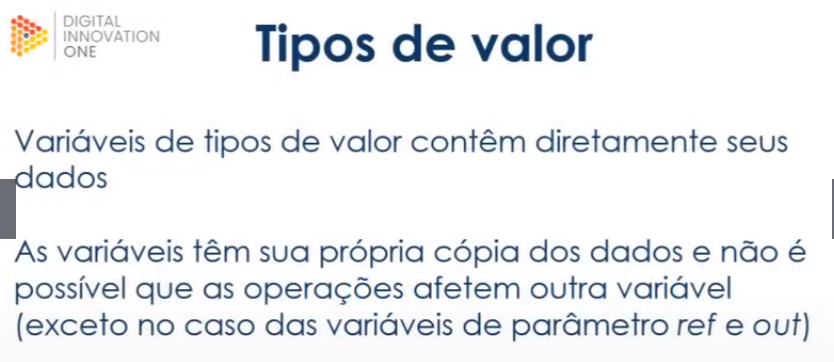
**OBS.:** As Assemblies são as DLL’s ou os EXE’s dependendo do projeto que se está desenvolvendo.

**Namespace** é o nome dado ao projeto.

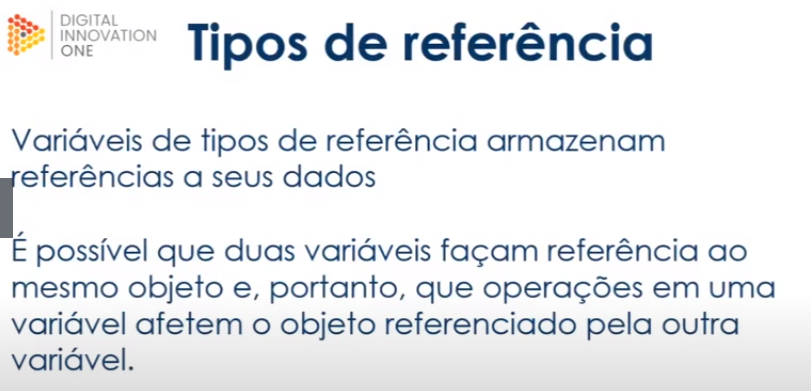
O conceito de ***Pilha*** é o de uma estrutura de dados em que o último que entra é o primeiro que sai.

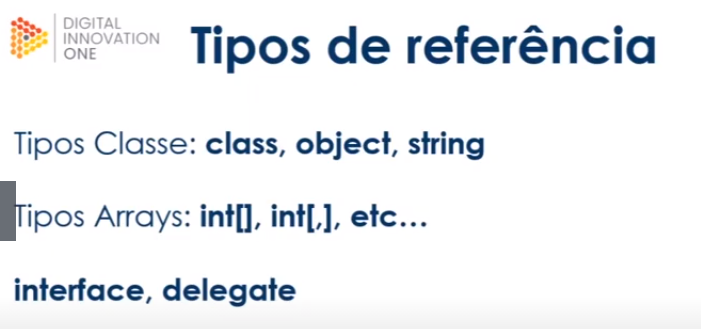
**Aula 04 : Variáveis e Instruções no C#**

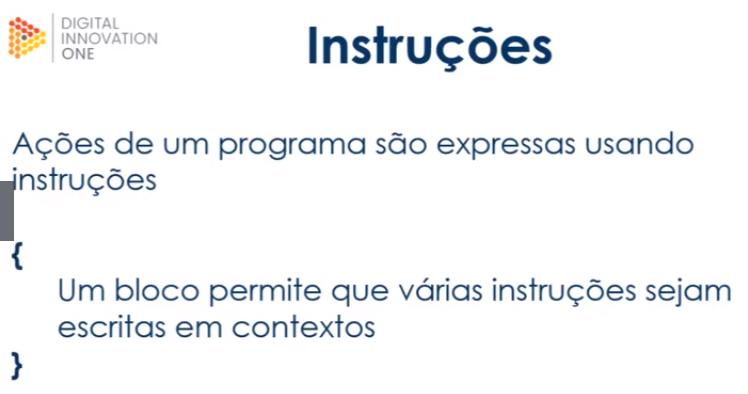


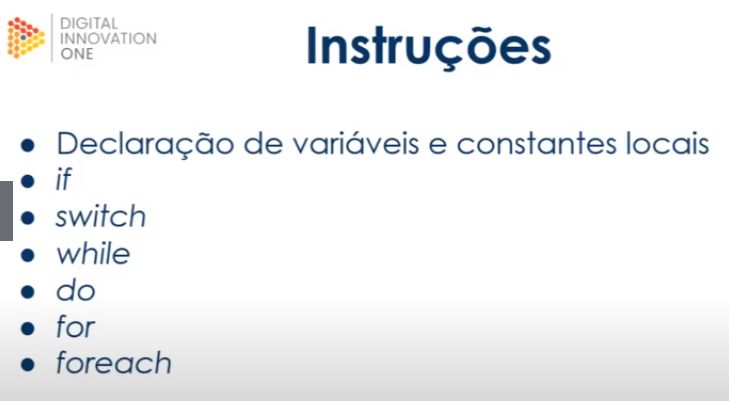












As instruções apresentadas acima, podem ser divididas da seguinte forma:

* ***Instruções de Condição:*** if, switch;
* ***Instruções de Repetição:*** while, do, for e foreach.

**Instruções Auxiliares**



**OBS.:** break, continue e return são instruções que nos auxiliam a trabalhar em conjunto com as demais instruções que vimos.

***Instruções para trativas de exceptions (exceções):*** throw, try..catch..finally.

“***using***” é utilizada para a impostação de pacotes adicionais e namespaces durante a execução do programa.

Podemos declarar variáveis e podemos declarar constantes ao se trabalhar com um código em C#.

Valores de constantes em C# não podem ser alterados.

**Arrays:** O mesmo que listas;

**Strings:** Sequência de caracteres no formato de texto. Deve ser colocada entre aspas duplas;

**$:** Utilizado na frente de uma String Interpolation (Interpolação de String) para concatenar ou alterar o texto de uma string;

**Realizar comentários no Visual Code:** Selecione o trecho que deseja comentar e tecle “CTRL + K + C” ou “CTRL + ;” (ponto e vírgula);

**OBS.:** Os comentários em C# iniciam com “**//**”;

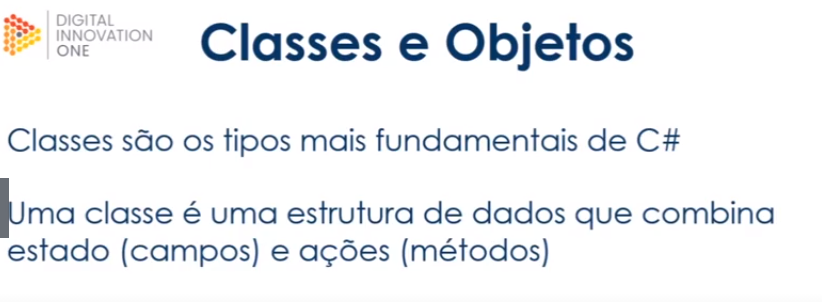
Para trocar de linha utilizamos F11. O F11 executa a função de desempilhar no Console WriteLine;

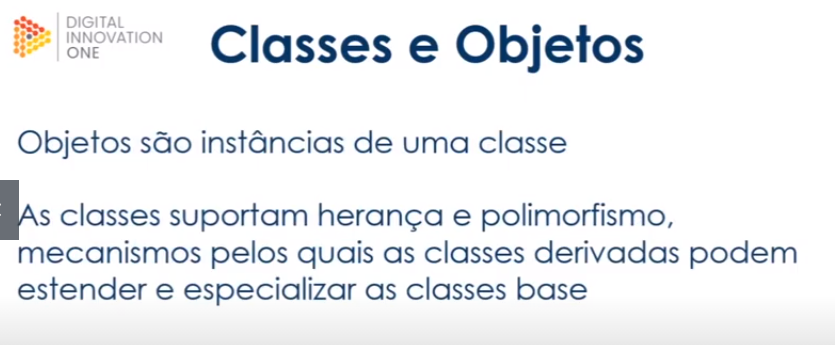
**Aula 05 : Classes e objetos essenciais em C#**

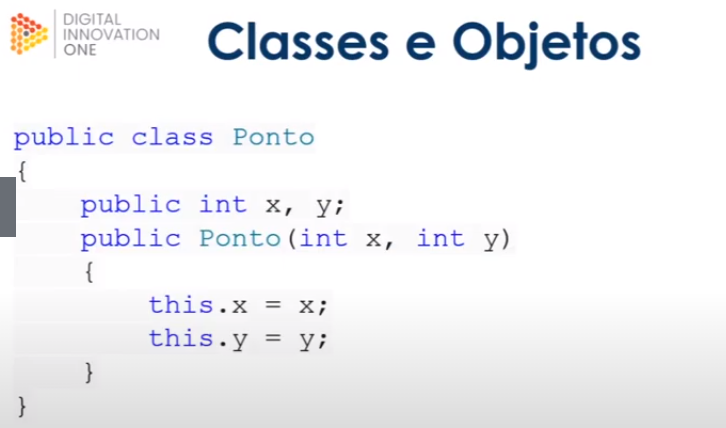
**Preparando o Ambiente**

Objetivos da aula:

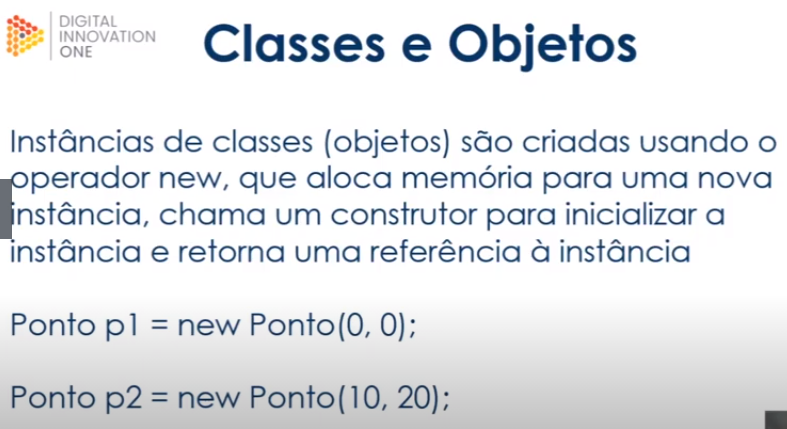
1 – Conhecer Classes e Objetos.





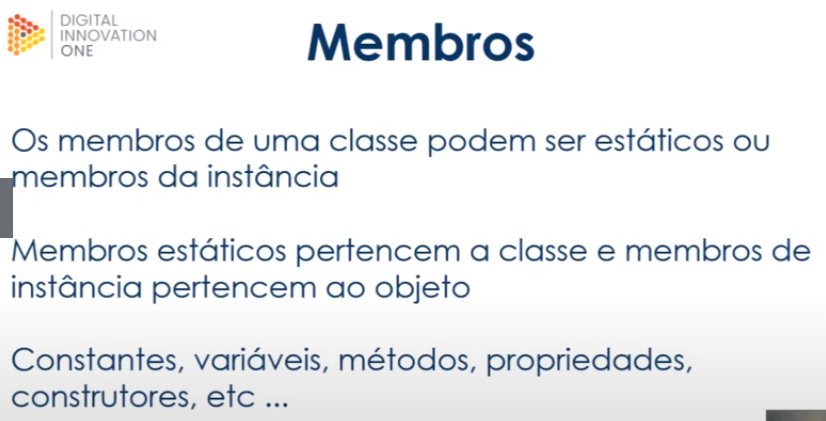


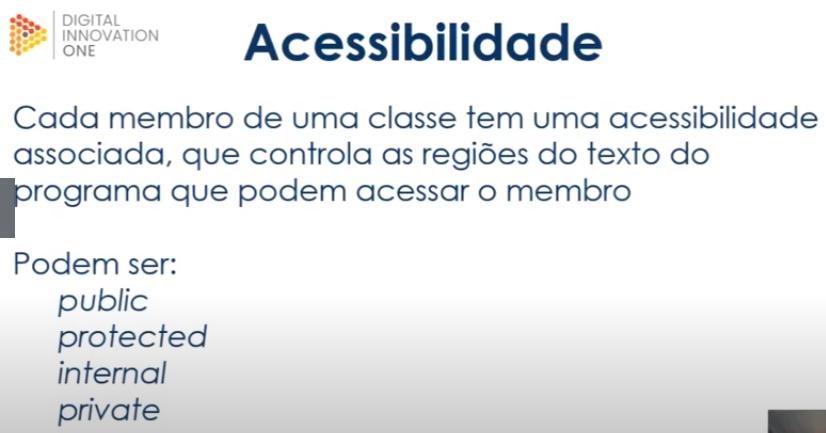
“public Ponto” mostrado acima é chamado de “Construtor” da Classe.

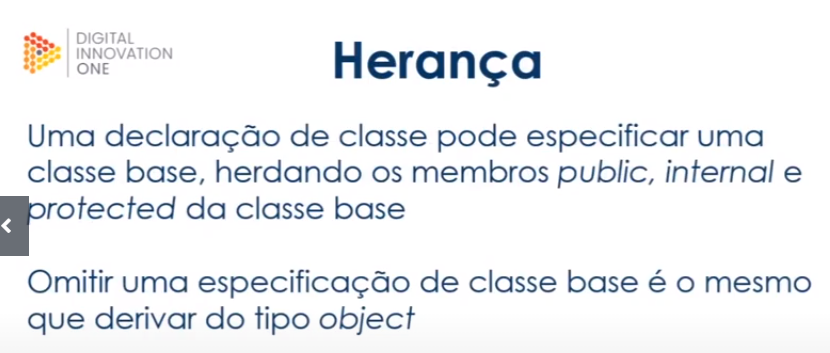


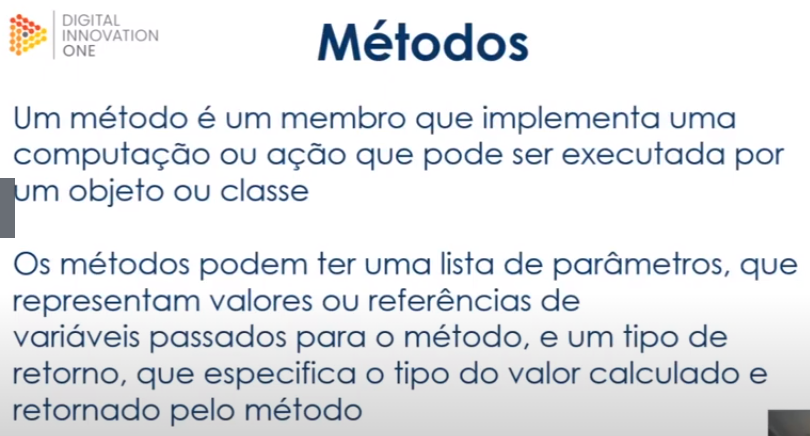


**OBS.:** O responsável por realizar essa ‘limpeza’ da memória quando um objeto ou item não está mais sendo utilizado é o Garbage Colector (Coletor de Lixo) como já vimos em uma aula anterior.





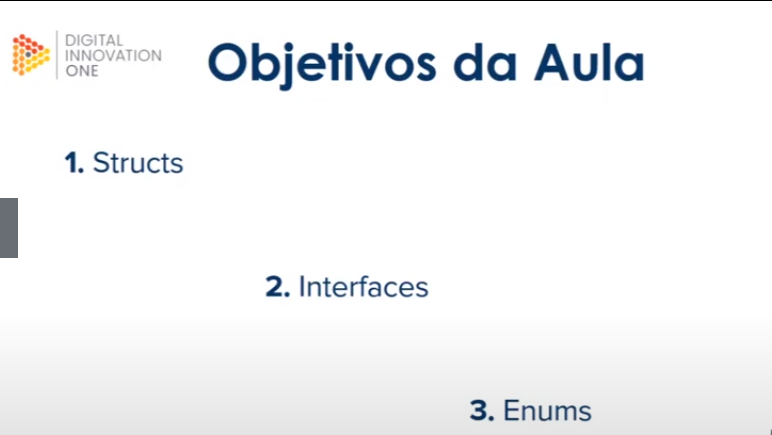


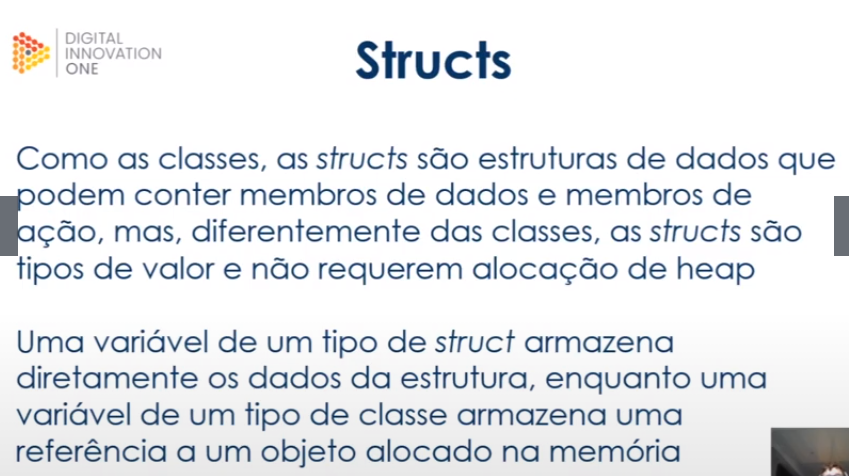


Ao darmos um nome para um método é importante que esse nome seja um verbo pois indica uma ação. Já para as ‘propriedades’ desse método é importante que utilizemos substantivos pois indicarão as propriedades dos estados, dos objetos, etc.

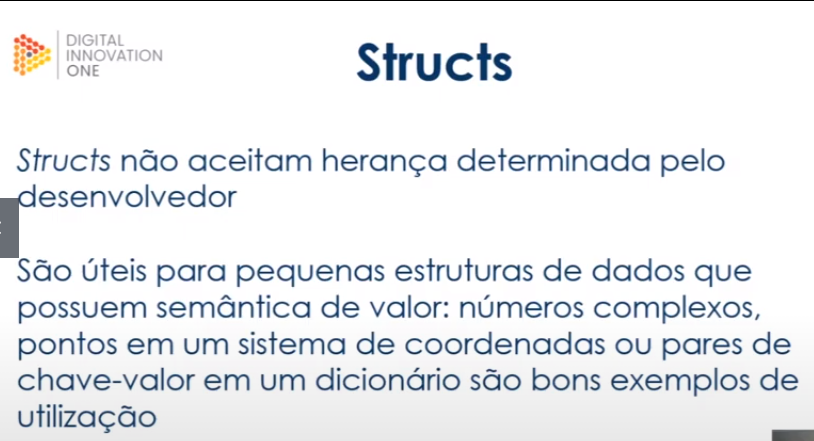
**ALGUMAS CURIOSIDADES SOBRE A LINGUAGEM**

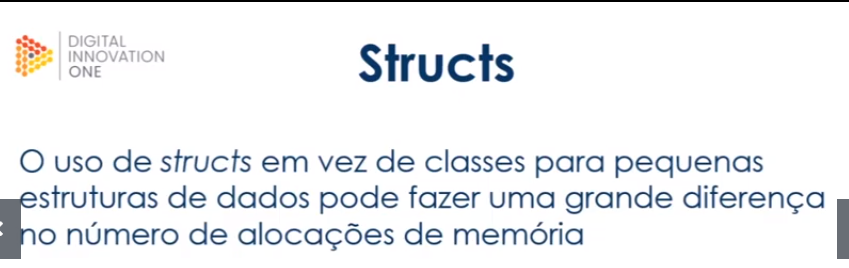
“this” : Utilizado para fazer referência a uma determinada variável. Pode-se também utilizar o (\_) ‘underline’. O underline também é utilizado para indicar que determinada classe é ‘private’.

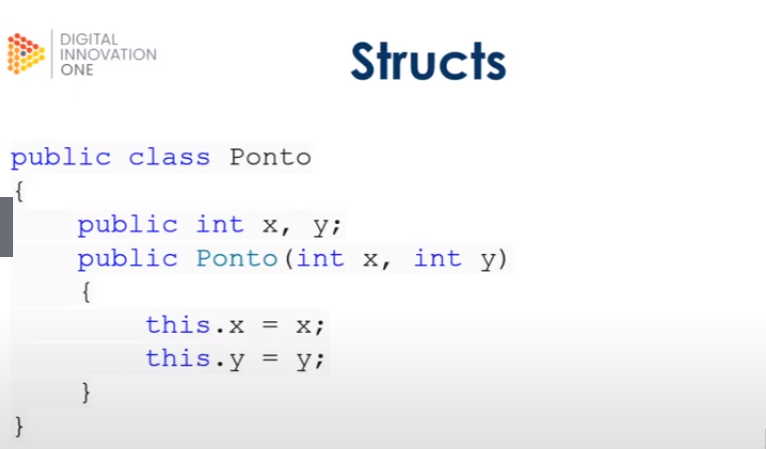


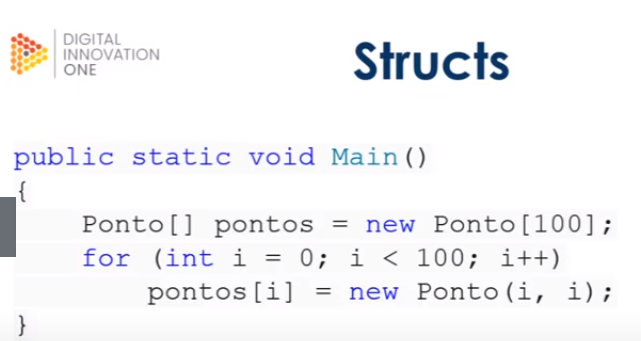


**Heap** – É o local onde fica alocado o valor de um objeto quando utilizamos variáveis de referência. Isso porque as variáveis de referência não guardam o valor do objeto em si. Como o próprio nome já diz, elas fazem apenas referência a um valor armazenado em outro local. Esse outro local é justamente o heap.

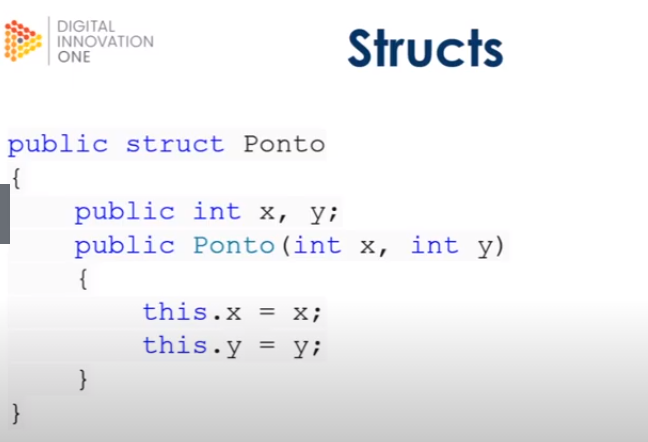




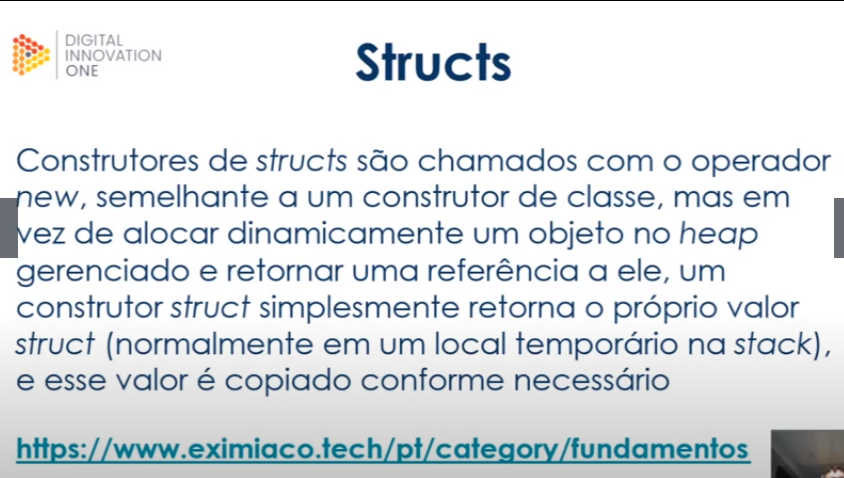




Veja abaixo um exemplo de design de uma Struct

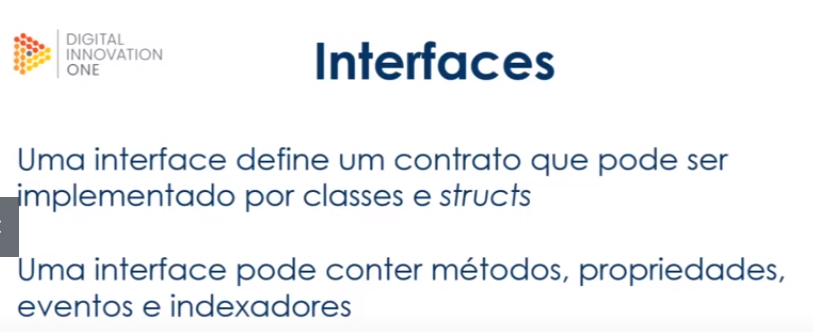


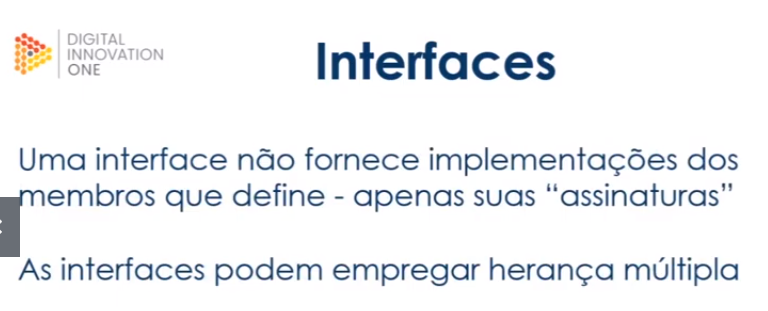
Note que a única diferença de um construtor para Struct de um construtor para a Classe é a palavra “struct” logo após a palavra ‘public’.

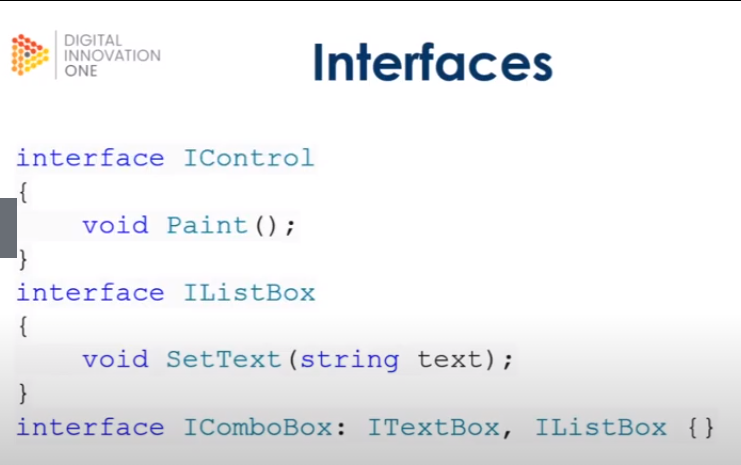


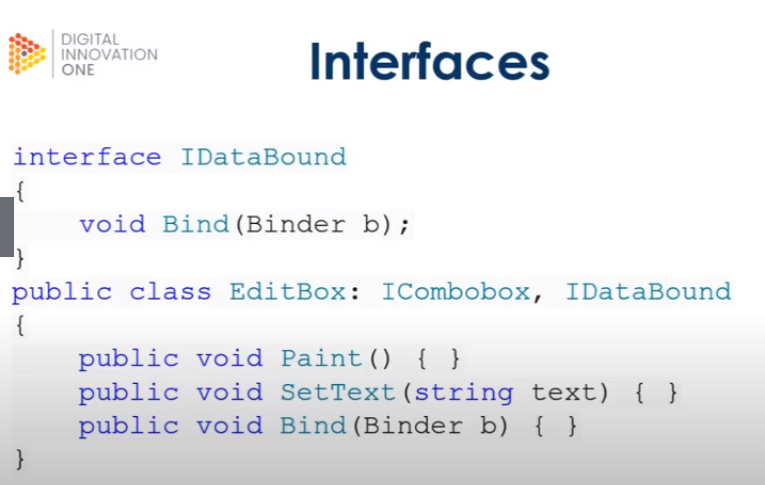
**Stack** – Local da memória em que são guardados os valores de um objeto sem a necessidade de se fazer referência. É diferente da *Heap* que aloca na memória valores referenciados.

**Interfaces e Enums**

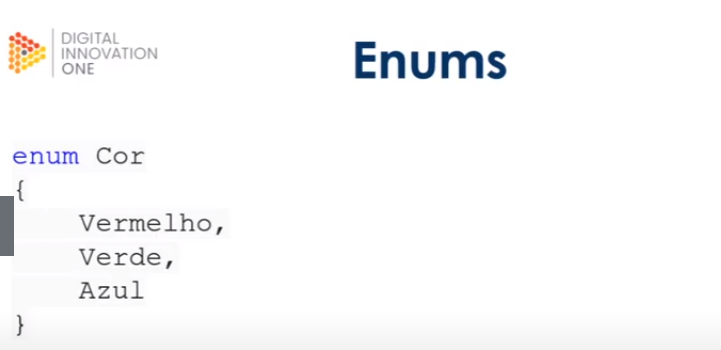


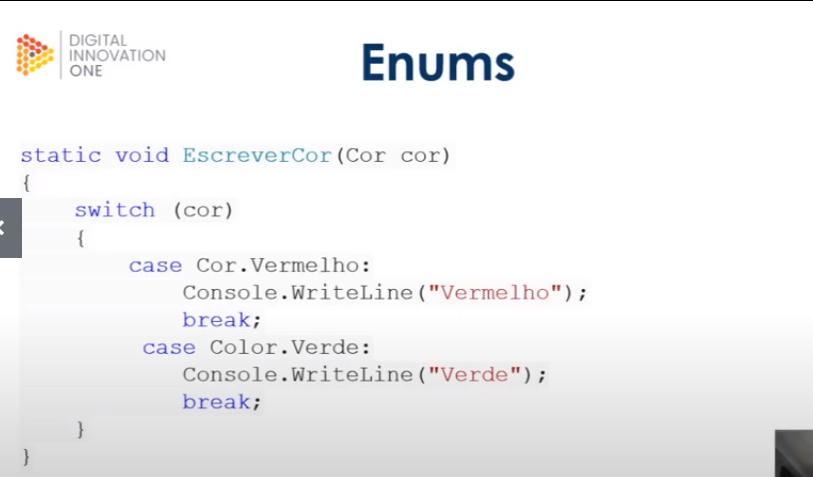


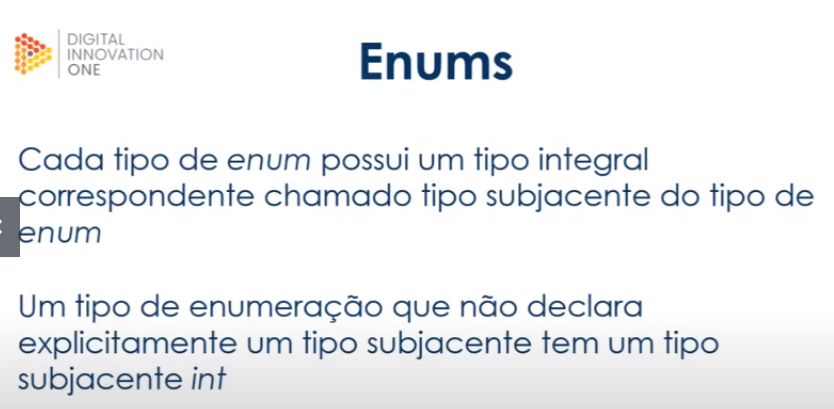


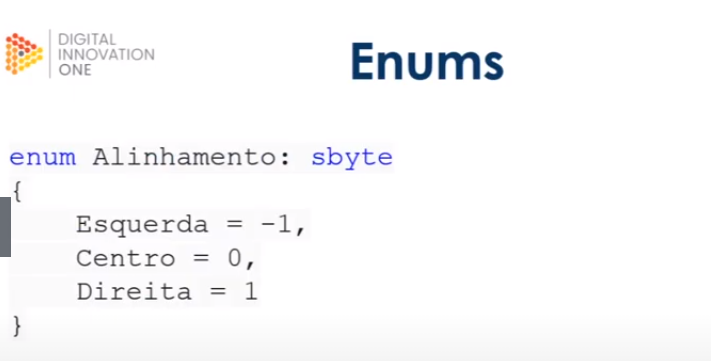












**Aula 07: Síntese do que é .NET**

**Restrospectiva**

