Microsoft iniciou nos anos de 70, com basic.

Anos 80 surge o DOS, utilizado como OS padrão para computadores IBM

1997 consolidar as ferramentas de desenvolvimento IDEs e Runtimes com VS97

Visual basic 5, Visual FoxPro 5, C++ 5, J++

1998 Visual Studio 6

Visual Basic 6, Visual FoxPro 6, c++6. J++6

1999 – Scott Guthrie criou a ferramneta web com java chamada ASP+ (depois chamou ASP next e depois ASPX) Jason Zander ajudou na criação de um common runtime para VB e C++ (CLR)

Sun Microsystem fez um acordo para Microsoft para que não mexessem mais com o Java

Anders Hejlsberg começou a trabalhar no C#

2000 Microsoft lança o novo ambiente de desenvolvimento .NET 1.0

2001 Miguel de Lcaza começa a trabalhar no projeto Mono, uma reimplementação black box do .NET, open source e multiplataforma

2002 lança o visual studio .Net com C# 1.0, conhecido como 22 linguagens 1 plataforma:

C#, C++, VB, J#, entre outras .net

2003 Lançamento do .net 1.1 com o VS 2003, trabalham em melhoria na CLR para lançar a CLR 2

2005 Lançamento do .NET 2.0 com C# 2.0 no VS 2005

2007-2008 .NET 3.5 com C# 3.0 no VS 2008 com Silverlight, WPF e WCF. Time de experiente open source e começam a atuar na criação do ASP.Net MVC

2010 .NET 4.0 com C# 4.0 no VS 2010 com F#, lançamento Win Azure, Anders criou o Type Script e C#

2011 Miguel de Icaza inicia Xamarin, aplicativos criados em C# para Android e iOS

2012 .NET 4.5 com C# 5 no VS 2012 e Lançamento do TypeScript

2013 .NET 4.5.1 no VS 2013. Início do Roslyn, um novo complicador para C# e VB.NET

2014 Saya Nadella se torna CEO da Microsoft e direciona o foco da empresa para Cloud. Criação do .NET foundation para gestão de projetos Open souce. Windowns Azure -> Microsoft Azure. Nova versão do ASP.Net vNext posteriormente chamado de ASP.NET core

2015 lançamento do .NET 4.6 com C# 6.0 no VS 2015. Lançamento do Visual Studio Code

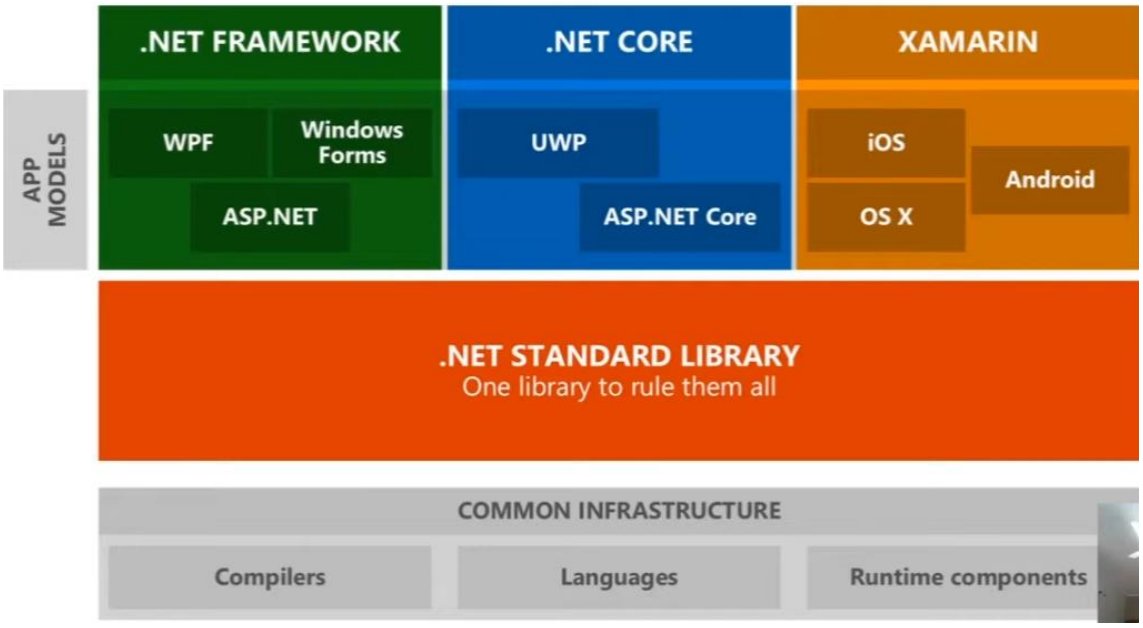
2016 Microsoft adquire a Xamarin e adiciona produto como parte de sua Stack .NET e projetos open source. Lançamento do VS para Mac

2017 lançamento do .NET Framework 4.7 com C# 7.0 no VS2017. Lançamento do .NET Core 2.0 com C# 7 em todos os outros

2019 .NET Framework 4.8 com C# 7.3 e .Net Core 3.0 com C# 8

Infraestrutura para desenvolvimento de Software criada pela Microsoft. Uma aplicação .NET é desenvolvida para e roda em uma das seguintes implementações do .NET:

.NET core, Framework, Mono e Universal Windowns Plataform (UWP)



Cada implementação inclui um ou mais .NET runtimes(Ambiente de execução)

.Net core – CoreCLR e Cor

.Mono – Mono RuntimeUMP – .NET Native

Linguagem suportada – C# F# e VB

Criar projeto

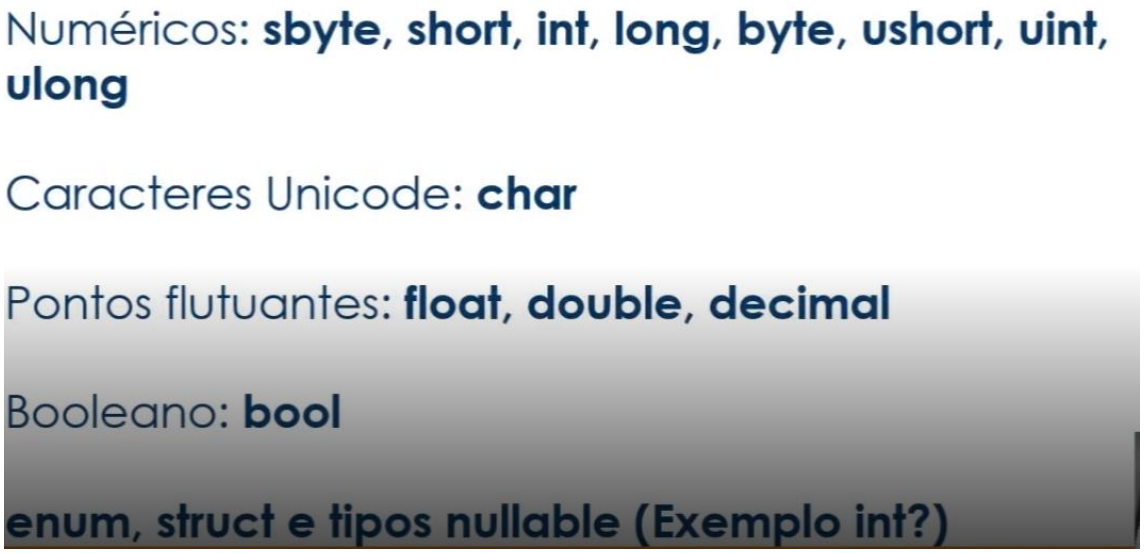
dotnet new + -h[help] (Puxar o que o comando pode executar)

dotnet new console -n [nomear] nome_da_pasta [criar uma pasta para o projeto com o nome dado]

dotnet restore

dotnet build [criação dos arquivos na pasta bin = .dll]

dotnet run [rodar aplicação]



- Declaração de variáveis e constantes locais
- *if*
- *switch*
- *while*
- *do*
- *for*
- *foreach*

- *break*
- *continue*
- *return*
- *throw*
- *try .. catch .. finally*
- *using*

```
int[] a = new int[10];
for (int i = 0; i < a.Length; i++)
{
   a[i] = i * i;
}
for (int i = 0; i < a.Length; i++)
{
   Console.WriteLine($"a[{i}] = {a[i]}");
}
```

Array multidimensional

```
int[,] a2 = new int[10, 5];

int[,,] a3 = new int[10, 5, 2];
```

Podem ser:

public
protected
internal
private

Structs

Structs não aceitam herança determinada pelo desenvolvedor

São úteis para pequenas estruturas de dados que possuem semântica de valor: números complexos, pontos em um sistema de coordenadas ou pares de chave-valor em um dicionário são bons exemplos de utilização

Interfaces

Uma interface define um contrato que pode ser implementado por classes e *structs*

Uma interface pode conter métodos, propriedades, eventos e indexadores

Enums

Um *enum* é um tipo de valor distinto com um conjunto de constantes nomeadas

Você define enumerações quando precisa definir um tipo que pode ter um conjunto de valores discretos. Eles usam um dos tipos de valor integral como armazenamento subjacente. Eles fornecem significado semântico aos valores discretos

Orientação a objeto (P.O.O [Programação Orientada a Objetos])

Paradigma de programação, uma maneira de fazer com quatro pilares

Abstração, encapsulamento, Herança, Polimorfismo

Paradigma de programação é diferente de linguagem de programação.

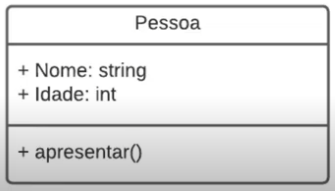
Uma linguagem de programação implementa um ou mais paradigmas.

Um paradigma nada mais é do que um modelo de técnicas, estruturas e formas de solucionar um problema.

- Programação orientada a objetos (é o que estamos estudando!)
- Programação estruturada
- Programação imperativa
- Programação procedural
- Programação orientada a eventos
- Programação lógica

Abstração

Abstrair um objeto do mundo real para um contexto específico, considerando apenas os atributos importantes.



Classe

Encapsulamento

O encapsulamento serve para proteger uma classe e definir limites para alteração de suas propriedades.

Serve para ocultar seu comportamento e expor somente o necessário.

Tabela de resumo

Local do chamador	`public`	`protected internal`	`protected`	`internal`	`private protected`	`private`
Dentro da classe	✔	✔	✔	✔	✔	✔
Classe derivada (mesmo assembly)	✔	✔	✔	✔	✔	✘
Classe não derivada (mesmo assembly)	✔	✔	✘	✔	✘	✘
Classe derivada (assembly diferente)	✔	✔	✔	✘	✘	✘
Classe não derivada (assembly diferente)	✔	✘	✘	✘	✘	✘

Herança

A herança nos permite reutilizar atributos, métodos e comportamentos de uma classe em outras classes.

Serve para agrupar objetos que são do mesmo tipo, porém com características diferentes.

Polimorfismo

O polimorfismo vem do grego e significa “muitas formas”.

Com o polimorfismo, podemos sobrescrever métodos das classes filhas para que se comportem de maneira diferente e ter sua própria implementação.

Classe Abstrata

Uma classe abstrata tem como objetivo ser exclusivamente um modelo para ser herdado, portanto não pode ser instanciada.

Você pode implementar métodos ou deixa-los a cargo de quem herdar.

Classe abstrata selada

Uma classe selada tem como objetivo de impedir que outras classes façam uma herança dela, ou seja, nenhuma classe pode ser sua derivada.

Também existem métodos e propriedades seladas.

Classe abstrata object

A classe System.Object é a mãe de todas as classes na hierarquia do .NET

Todas as classes derivam, direta ou indiretamente da classe Object, e ela tem como objetivo prover serviços de baixo nível para suas classes filhas.

Métodos

Equals(Object)	Determina se o objeto especificado é igual ao objeto atual.
Equals(Object, Object)	Determina se as instâncias de objeto especificadas são consideradas iguais.
Finalize()	Permite que um objeto tente liberar recursos e executar outras operações de limpeza antes de ser recuperado pela coleta de lixo.
GetHashCode()	Serve como a função de hash padrão.
GetType()	Obtém o Type da instância atual.
MemberwiseClone()	Cria uma cópia superficial do Object atual.
ReferenceEquals(Object, Object)	Determina se as instâncias de Object especificadas são a mesma instância.
ToString()	Retorna uma cadeia de caracteres que representa o objeto atual.

Interface

Uma interface é um contrato que pode ser implementado por uma classe.

É como se fosse uma classe abstrata, podendo definir métodos abstratos para serem implementados.

Assim como uma classe abstrata, uma interface não pode ser instanciada.

Arquivo

O C# nos apresenta algumas classes estáticas que facilitam o trabalho com arquivos, dentre elas:

- File
- Directory
- Path

```
Z:\Gabriel\Decola-Tech\Orientação a Objeto>dotnet new sln  --name ExemploPOO
O modelo "Arquivo de Solução" foi criado com êxito.

Z:\Gabriel\Decola-Tech\Orientação a Objeto>mkdir ExemploPOO

Z:\Gabriel\Decola-Tech\Orientação a Objeto>cd ExemploPOO

Z:\Gabriel\Decola-Tech\Orientação a Objeto\ExemploPOO>dotnet new console
O modelo "Aplicativo do Console" foi criado com êxito.

Processando ações pós-criação...
Executando 'dotnet restore' em Z:\Gabriel\Decola-Tech\Orientação a Objeto\ExemploPOO\ExemploPOO.csproj...
  Determinando os projetos a serem restaurados...
  Z:\Gabriel\Decola-Tech\Orientação a Objeto\ExemploPOO\ExemploPOO.csproj restaurado (em 113 ms).
A restauração foi bem-sucedida.


Z:\Gabriel\Decola-Tech\Orientação a Objeto\ExemploPOO>cd ..

Z:\Gabriel\Decola-Tech\Orientação a Objeto>dotnet sln add ExemploPOO/ExemploPOO.csproj
O projeto 'ExemploPOO\ExemploPOO.csproj' foi adicionado à solução.

Z:\Gabriel\Decola-Tech\Orientação a Objeto>
```

Construtores

Um construtor é um método especial, que contém o mesmo nome do seu tipo classe, e tem o objetivo de definir valores padrão, limitar uma instância e facilitar a instanciação de um objeto.

- Um construtor não possui um retorno.
- Um construtor padrão é sempre definido quando não declaramos nenhum para sua classe.
- Uma classe pode ter mais de um construtor.

Get e Set

O Get e Set serve para que você possa atribuir um valor em uma variável de maneira controlada e com validações.

Modificadores

Readonly

O modificador readonly (somente leitura) bloqueia um campo contra alterações que não sejam em sua inicialização ou pelo próprio construtor.

Constante

Uma constante representa um valor que somente pode ser atribuído no momento de sua inicialização, e não pode ser modificado posteriormente.

Delegates

Um delegate é uma maneira de passar um método como referência, podendo ser usado como um callback, aceitando qualquer método que contenha a mesma assinatura.

Eventos

Eventos é um mecanismo de comunicação entre objetos, onde existe um publisher, que realiza o evento e o subscriber, que se inscreve em um evento.

Arrays

O array é uma estrutura de dados que armazena valores do mesmo tipo, com um tamanho fixo.

A classe Array é uma classe do C# que nos oferece diversos métodos que nos auxiliam a trabalhar com arrays.

Para ordenar um array, existem diversos algoritmos de ordenação, diferentes técnicas e casos a serem considerados.

Coleções genericas

No C#, existem classes de coleções que agrupam valores, e essas classes são padronizadas para as operações mais comuns, como:

- Ordenação
- Obter valor por índice
- Obter valor com expressões
- Trabalhar com tamanhos dinâmicos

Coleções especificas

As coleções específicas implementam regras para sua ordem de acesso e manipulação de seus elementos, são elas:

- Queue (Fila): Obedece a ordem FIFO (First In First Out)
- Stack (Pilha): Obedece a ordem LIFO (Last In First Out)

Dicionario

- Um dicionário é uma coleção de chave e valor, permitindo que você recupere rapidamente seus itens baseado em sua chave.

- O dicionário armazena a sua chave em hash.

Criação de um dicionário:

```
Dictionary<string, string> estados = new Dictionary<string,string>()
```

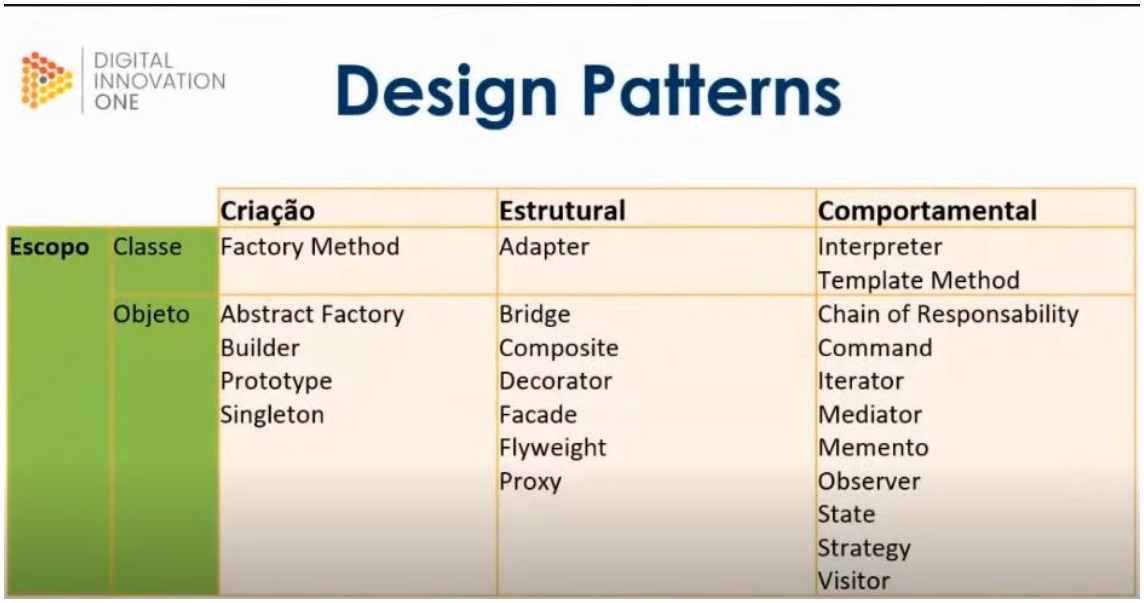
LINQ

O Language-Integrated Query (LINQ) é uma maneira de você utilizar uma sintaxe de consulta padronizada para coleções de objetos.

```
int[] numbers = { 5, 10, 8, 3, 6, 12 };

//Query syntax:
IEnumerable<int> numQuery1 =
    from num in numbers
    where num % 2 == 0
    orderby num
    select num;

//Method syntax:
IEnumerable<int> numQuery2 = numbers.Where(num => num % 2 == 0).OrderBy(n => n);
```



Design Patterns – Algumas dicas

Você não precisa saber todos. É bom conhecer bem os mais usados. (Adapter, Factory, Observer, Strategy, Builder, State, Singleton).
Os menos usados, basta saber a proposta dele e conseguir discutir sobre. Se precisar implementar, basta revisar a literatura.