dotnet core - csharp

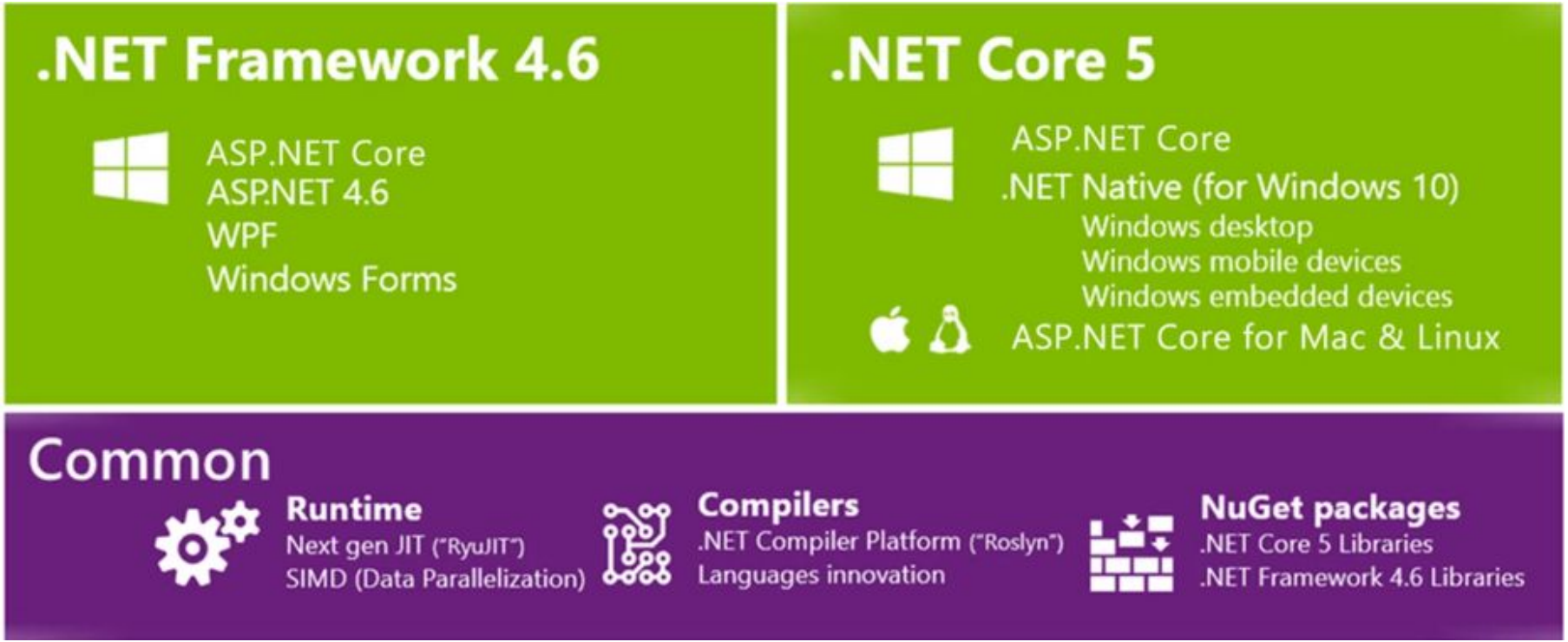
Linguagem de Programação Comercial

@cassiocosta_

Documentação oficial

- http://docs.microsoft.com
- https://docs.microsoft.com/pt-br/dotnet/csharp/
- https://docs.microsoft.com/pt-br/dotnet/core/tools/ (dotnet cli)
- https://github.com/dotnet/core/

DotNetCore 2.x



.Net Core 2.x

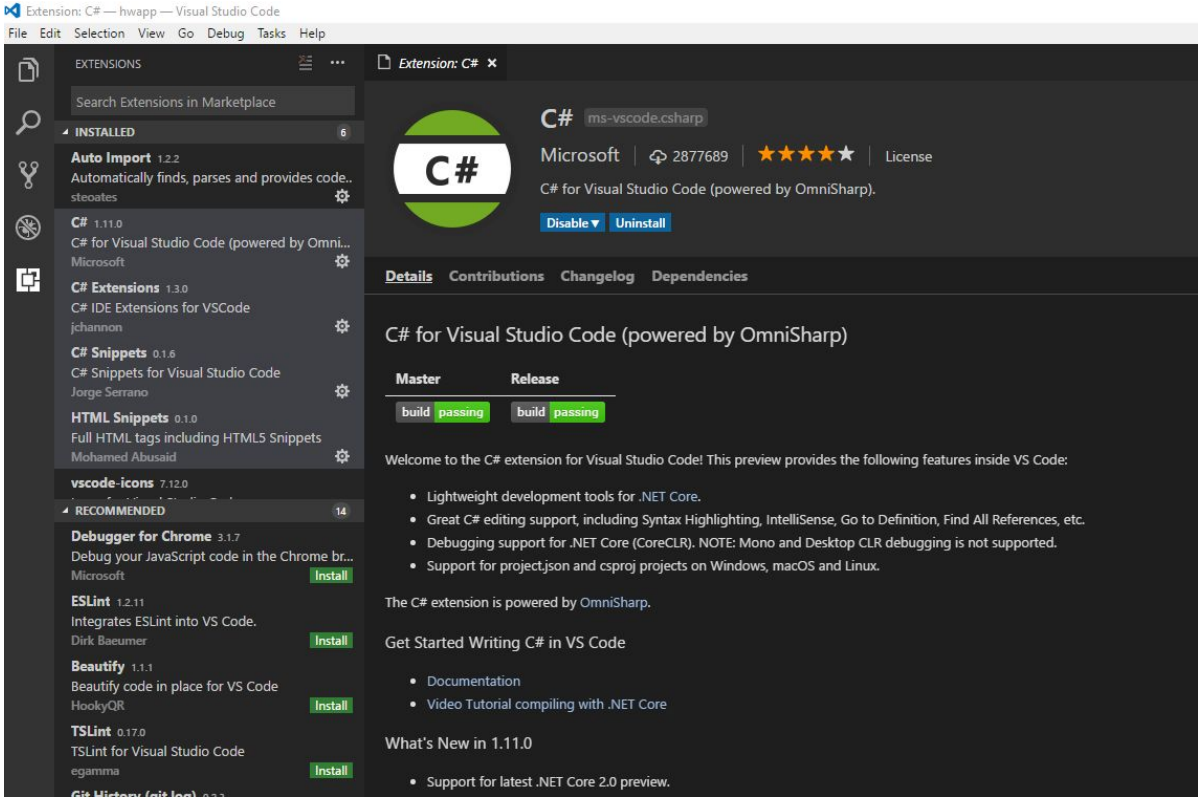
- .Net Core é completamente modularizado. Cada componente está distribuído via NugGet.
- Apps podem ser atualizados independente e não dependem do framework instalado no servidor.
- Pacotes de implantação incluem todo o necessário para rodar.
- Croos-plataform.
- Open source.
- Você escolhe seu editor e ferramenta.

SDK e editor

https://github.com/dotnet/core/blob/master/release-notes/2.1/2.1.2.md

VSCODE

- c# e c# extensions
- C# Snippets
- Vscode-icons
- Html-snippets



Linguagem de programação - C# versão atual 7.x

- multiparadigma, de tipagem forte, parte da plataforma .NET.
- Orientada a Objetos
- baseada no C++, influências do Object Pascal e Java.
- Compilado para Common Intermediate Language (CIL)
- interpretado pela máquina virtual (CLR).
- Projetada para funcionar no .net

C# - características

- Simplicidade: curva de aprendizado é baixa;
- OO: qualquer variável tem de fazer parte de uma classe; Tudo é um objeto;
- Linguagem gerenciada: os programas desenvolvidos em C# executam num ambiente gerenciado,
- o que significa que todo o gerenciamento de memória é feito pelo runtime via o GC (Garbage Collector)

C# - Variáveis e tipos primitivos

- As variáveis guardam informações de um tipo específico.
  - Ex. guardar um número inteiro representando o número de uma conta, um texto para representar o nome do correntista ou número real para representar o saldo atual da conta.
  - uma variável deve ser declarada no programa.
  - sempre dizer o tipo (inteiro, texto ou real por exemplo) e,
  - qual o nome para referenciá-la no programa.

```
int numeroDaConta;
```

C# - Variáveis e tipos primitivos (Cont)

Além do tipo int, temos também os tipos double, float (números reais), strings (para textos) entre outros.

```
double saldo = 100.0;
```

```
numeroDaConta = 1;
```

C# - Operações com Variáveis

Agora que já sabemos como guardar informações no programa, estamos interessados em executar operações nesses valores. Pode ser interessante um correntista saber qual será o saldo de sua conta após um saque de 10 reais. Para realizar essa operação, devemos subtrair 10 do saldo da conta.

```
double saldo = 100.0;
double valorAposOSaque = saldo - 10.0;
```

Podemos ainda guardar o valor do saque em uma variável.

```
double saldo = 100.0;
double valorDoSaque = 10.0;
double valorAposSaque = saldo - valorDoSaque;
```

C# - Estruturas de Controle

No C#, podemos executar código condicional utilizando a construção `if` :

```
if (condicao)
{
    // Esse código será será executado somente se a condição for verdadeira
}
```

```
double saldo = 100.0;
double valorSaque = 10.0;
if (saldo >= valorSaque)
{
    // código do saque.
}
```

```
if (saldo >= valorSaque)
{
    // código do saque
}
else
{
    MessageBox.Show("Saldo Insuficiente");
}
```

C# - Estruturas de Controle (Cont.)

Repare na expressão que passamos para o `if` : `saldo >= valorSaque` . Nele utilizamos o operador "maior ou igual". Além dele, há outros operadores de comparação que podemos utilizar: maior (>), menor (<), menor ou igual (<=), igual (==) e diferente (!=). E ainda existem mais opções que podemos passar como condição para o `if` .

```
bool podeSacar = (saldo >= valorSaque);
```

Também podemos realizar algumas operações com valores do tipo `bool` . Podemos, por exemplo, verificar se duas condições são verdadeiras ao mesmo tempo usando o operador `&&` (AND):

```
bool realmentePodeSacar = (saldo >= valorSaque) && (valorSaque > 0);
```

```
if (realmentePodeSacar)
{
    // código do saque
}
else
{
    MessageBox.Show("Saldo Insuficiente");
}
```

C# - estruturas de repetição (for)

```
for (inicialização; condição; atualização)
{
    // Esse código será executado enquanto a condição for verdadeira
}
```

```
double valorInvestido = 1000.0;
for (int i = 1; i <= 12; i += 1)
{
    valorInvestido = valorInvestido * 1.01;
}
```

C# - estruturas de repetição (while)

```
double valorInvestido = 1000.0;
int i = 1;
while (i <= 12)
{
    valorInvestido = valorInvestido * 1.01;
    i += 1;
}
```

C# - Classes e objetos

Uma Conta bancária é geralmente composta por número, nome do titular e saldo. Podemos guardar essas informações em váriáveis.

```
int numeroDaConta1 = 1;
string titularDaConta1 = "Joaquim José";
double saldoDaConta1 = 1500.0;
```

Para representar outros correntistas, vamos usar novas variáveis.

```
int numeroDaConta2 = 2;
string titularDaConta2 = "Silva Xavier";
double saldoDaConta2 = 2500.0;
```

Então por quê não criar um modelo para isso?

C# - Classe

Ou seja, dizer ao programa criar uma conta e ele se encarrega de criar essas 3 variáveis automaticamente.

Para fazer isso vamos criar uma CLASSE.

Uma Classe representa uma entidade, um conceito de domínio da nossa aplicação. Para representar uma conta (em arquivo Conta.cs). Ela tem número, nome do titular e saldo:

```
class Conta
{
    public int numero;
    public string titular;
    public double saldo;
}
```

C# - Objetos - uma instância de uma classe

Podemos agora criar muitas contas, todas elas terão os atributos definidos na classe.

Para criar uma conta basta usar a palavra NEW e atribuir para uma variável.

```
Conta umaConta = new Conta();
```

Com essa instância na mão, podemos modificar os valores.

```
Conta umaConta = new Conta();
umaConta.numero = 1;
umaConta.titular = "Joaquim José";
umaConta.saldo = 1500.0;
```

```
Conta outraConta = new Conta();
outraConta.numero = 2;
outraConta.titular = "Silva Xavier";
outraConta.saldo = 2500.0;
```

c# - OO: Comportamentos - Métodos

Se eu quiser pegar o dinheiro de uma conta e passar pra outra?

```
1. // criando duas contas
2. Conta mauricio = new Conta();
3. Conta guilherme = new Conta();
4. mauricio.saldo = 1000.0;
5. guilherme.saldo = 2000.0;
6.
7. // transferindo dinheiro entre elas
8. double valorASerTransferido = 100.0;
9. mauricio.saldo -= valorASerTransferido;
10. guilherme.saldo += valorASerTransferido;
```

c# - OO: Métodos

Implementação de uma transferência é bem simples! Mas qual o problema?

- Os dados são dinâmicos. Os valores vão mudar.

- pontos diferentes do programa terá que fazer isso. Será que saberemos todos esses pontos?

- **Um bom design orientado a objetos é aquele que facilita as alterações, a sua manutenção.**

- SOLUÇÃO?
  - colocar esse comportamento em algum lugar que seja fácil de ser encontrado e de ser reutilizado. Um bom lugar para ele é dentro da própria classe conta . Uma classe pode conter, além de atributos, métodos que manipulam esses atributos.

Encapsulamento

C# - Métodos

```
public class Conta
{
    public int numero;
    public string titular;
    public double saldo;

    public void Sacar(double valorSerSacado)
    {
        this.saldo -= valorSerSacado;
    }
}
```

Note o uso do This. Diz que o saldo é um atributo da classe. Observe o uso do Public, dará acesso a este método a outros trechos de código. Para fazer uso do novo comportamento (método):

```
1. Conta mauricio = new Conta();
2. mauricio.saldo = 1000.0;
3. mauricio.Saca(150.0);
```

```
1. Conta guilherme = new Conta();
2. guilherme.saldo = 500.0;
3. guilherme.Saca(100.0);
```

Faça o método Depositar e transferir;

c# - OO: Composição de Classes

E se precisar de mais dados não somente da conta mas do correntista/Cliente?

Devemos criar mais atributos, mas em qual classe? pois essas opções não pertencem a conta e sim ao cliente do banco.

```
class Cliente
{
    public string nome;
    public string cpf;
    public string rg;
    public string endereco;
}
```

Sabe-se que toda a conta está associada a um cliente, ou seja, uma conta possui um cliente.

```
class Conta
{
    // outros atributos da Conta

    public Cliente cliente;

    // comportamentos da conta
}
```

C# - Composição de Classes

Logo, para criar uma conta deve-se informar o cliente.

```
Cliente victor = new Cliente();
victor.nome = "victor";

Conta umaConta = new Conta();
umaConta.cliente = victor;
```

a variável “umaConta” guarda uma referência(cliente) para uma instância de Cliente(objeto na memória). Então a atribuição umaConta.cliente = victor está copiando uma referência da variável victor para o atributo cliente.

```
Cliente victor = new Cliente();
victor.nome = "victor";

Conta umaConta = new Conta();
umaConta.cliente = victor;

umaConta.cliente.rg = "12345678-9";
```

Exercícios práticos