APONTAMENTOS

C#

Uma aplicação é composta por classes.

As classes podem ser agrupadas por Namespace.

NameSpace e um agrupamento lógico de classes relacionadas.

Assemby (DLL ou EXE) é uma agrupamento físico de classes relacionadas (build)

Aplicação é um agrupamento de assemblyes relacionados.

* Indentação automática: CTRL + K + D
* Rodar o projeto; CTRL + F5
* Fonte: CTRL + Mouse scroll
* CW tab tab = Console.WriteLine()
* F9 – marcar/desmarcar breakpoint
* F5 – iniciar/continuar o debug
* F10 – executar um passo (pula função)
* F11 – executar um passo (entra na função)
* Shift+f11 – sair do método em execução
* Shift+F5 – parar debug
* Warch (expressões personalizadas)
* Autos (expressões “interessants” detectadas pelo Visual Studio
* Lcals (variávies locais)

Carateres Unicode (ex.: char letra = ‘\u0041’) representa a letra A

Podemos consultar uma tabela de caracteres Unicode

**3.1. Variáveis**

* long – colocar L no final da variável
* float – colocar f no final da variável
* string – é imutável (segurança, simplicidade, thread safe)

**3.2. Restrições para nomes de variáveis**

* Não pode começar com dígito: use uma letra ou \_
* Não usar acentos ou til
* Não pode ter espaço em branco
* Use nomes que tenham significado

**3.3. Convenções**

* Camel Case: lastName (parâmetros de métodos, variáveis dentro de métodos)
* Pascal Case: LastName (namespace, classe, properties e métodos)
* Padrão \_lastName (atributos “internos” da classe)

**3.4. Casas decimais**

Console.WriteLine(variável.ToString(“Fx”))

x – corresponde ao número de casas decimais depois da virgula

**using System.Globalizacion**

Para usar (.) em vez de (;) é necessário utilizar o using System.Globalizacion;

Console.WriteLine(variável.ToString(“Fx”), CultureInfo.InvariantCulture))

x- corresponde ao número de casas decimais depois do ponto

**3.5. Placeholders**

Console.Writeline(“{0} têm {1} anos e têm saldo igual a {2:F2} reais”, nome, idade, saldo)

**3.6. Interpolação**

Console.WriteLine($”{nome} tem {idade} anos e tem saldo igual a {saldo:F2} reais”)

**3.7. Concatenação**

Console.WriteLine(nome + “ têm “ + idade + “ anos e têm saldo igual a “ + saldo.toString(“F2”) + “ reais”)

**3.8. Conversões implícitas e explicítas (CAST)**

Consultar link para esclarecimentos adicionais e para consultar a tabela de conversões

<https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/builtin-types/numeric-conversions>

é possível forçar a conversão colocando antes da variável entre parêntesis o tipo de variável final

double a;

float b;

a = 5.1;

b = (float)a; // força a conversão, o developer assume o risco da conversão

Console.WriteLine(b);

**3.9. operadores aritméticos**

**3.10. Entrada de dados**

Console.ReadLine();

* Lê da entrada padrão até a quebra de linha;
* Retorna os dados lidos na forma de string;

SPLIT

String[] – representação de um vetor

String[] vet = s.split(‘ ‘);

String p1 = vet[0];

Conversão da string do Console.ReadLine()

Int n = int.parse(Console.readLine())

**3.11. Operadores Lógicos**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Operador | Significado |  | C1 | C2 | C1 && C2 |  | C1 | C2 | C1 || C2 |
| && | E |  | F | F | F |  | F | F | F |
| || | OU |  | F | V | F |  | F | V | V |
| ! | Não |  | V | F | F |  | V | F | V |
|  |  |  | V | V | V |  | V | V | V |

Precedências ! > && > ||

**3.12 Estrutra Condicional**

|  |  |  |
| --- | --- | --- |
| Simples | Composta | Encadeamentos |
| If ( condição ) { Comando } | If ( condição ) { Comando }  Else { Comando } | If ( condição ) { Comando }  Else If { Comando }  Else { Comando } |

**3.13 Funções**

Tipo de variável variável = função (parametro1, parametro2, parâmetro n)

Double resultado = Maior(n1, n2,n3);

**3.14 Estrutura “enquanto”**

While (condição ) { Comando }

**3.15 Estrutura “para”**

For (ínicio; condição; incremento) { comando}

Ex: for(int i = 1; i <= n; i++)

**4.1 Classe**

Criamos uma classe Triangulo que vai conter atributos/variaveis (ex: x, y) com o prefixo Public para poderem ser consultadas em qualquer lugar do código

Ex

Class Triangulo

{

public Int x;

public Int y;

}

Para instânciar cada variável é necessário invocar a classe

Ex

Triangulo x = new Triangulo();

Triangulo y = new Triangulo();

**4.2 Classe/Método**

Método é uma função dentro da classe

Ex

Class Triangulo

{

public Int x;

public Int y;

public double Area()

{

double p = ( A + B + C ) / 2.0;

return Math.Sqrt ( p \* ( p – A ) \* ( p – B ) \* ( p – C ));

}

}

Para instânciar cada variável é necessário invocar a classe

Ex

Triangulo x = new Triangulo();

Triangulo y = new Triangulo();

double xArea = x.Area();

double yArea = y.area();

**4.3. ToString**

ToStrin converte o objeto para string

Ex:

publiv override string ToString()

{

Return x;

}

**5.1 Construtores**

O construtor é construído na classe e tem o mesmo nome da classe e pode ou não ter parâmetros de entrada. É obrigatório instância a classe com valores nos parâmetros.

Ex:

public Produto (string nome, double preco, int quantidade)

{

None = nome;

Preco = preco;

Quantidade = quantidade;

}

UTILIZANDO O GIT HUB

* Iniciar um projeto em visual studio;
* Criar ficheiro .gitignore:
  + No diretório do projeto carregar no lado esquerdo do rato e aceder a “new/texto document”:
  + Alterar o nome do documento para “.gitignore”;
    - Em alternativa criar um ficheiro git.txt:
    - Carregar em Shift e no lado esquerdo do rato e selecionar “open powershell ou prompt window”;
    - Renomear o ficheiro criado anteriormente para “ren git.txt .gitignore”;
  + Pesquisar na web “gitignore visual studio 2019” ex: <https://github.com/github/gitignore/blob/master/VisualStudio.gitignore#L2>;
  + Copiar todo o texto disponível e colar no ficheiro .gitignore e gravar;
* Aceder ao Gitbash no diretório a onde criamos o ficheiro .gitignore:
  + Carregar no lado esquerdo do rato para aceder ao Gitbash;
  + Digitar : **git init**
* Configurar a identificação do utilizador no GIT:
  + Digitar: **git config – global user.name “Tiago Nogueira”;**
  + Digitar: **git config – global user.email “**[**tiago.nogueira@sapo.pt**](mailto:tiago.nogueira@sapo.pt)**”**;
* Salvar uma versão do projeto – “commit”;
  + Digitar: **git status**;
    - Vermelho: arquivos não rastreados;
    - Verde: arquivo prontos para serem “commitado” (stage);
  + Digitar: **git add .** ;
    - O (.) quer dizer que serão adicionados todos os arquivos à área de stage;
    - No git status todos os arquivos deverão estar a verde;
  + Digitar: **git commit -m “nome descitivo do projeto e alteração afetudas”**;
    - No git status a msg deverá ser nothing to commit;
* Verificar as versões carregadas do projeto:
  + Digitar: **git log**;
    - Dará informação do autor da data e titulo dos commits efetuados;
  + Digitar: **git log –oneline**;
    - Versão simplificada do log que mostra a informação numa linha;
* Para sair do editor VIM:
  + Digitar: **<esc>** depois **:q!** para sair do editor;
* Descartar as modificações atuais voltando ao estado commita anterior:
  + Digitar: **git clean -df**
  + Digitar: **git checkout -- .**
    - Irá surgir uma mensagem no visual studio que deverá confirmar. O código será atualizado para a ultima versºaoi commitada;
* Remover o último commit mantendo as alterações nos arquivos:
  + Digitar: **git reset –soft HEAD~1**
* Remover o ultimo commit inclusive as alterações nos arquivos (PERIGO!):
  + Digitar: **git reset –hard HEAD~1** (irá aparecer uma mensagem no visual studio para confirmar);
* Navegar entre commits, alterando os arquivos temporariamente:
  + Digitar: **git checkout <código do commit>** (não pode existir nada para commitar)
* Voltar para o último commit:
  + Digitar: **git checkout <nome do branch>** (o nome do branch neste caso é master)
* Associar o nosso repositório local ao repositório remoto, dando o no de “origin”:
  + Digitar: **git remote add origin <URI do repositório remoto>** (o URI do repositório remoto pode ser encontrado assim que se cria a repositório do github)
* Associar o repositório local a um outro repositório remoto, podem mantendo o mesmo nome:
  + Digitar: **git remote set-url origin <URI do repositório remoto>**
* Como enviar o repositório local para o GitHub:
  + Digitar: **git push -u origin master** (se utilizou a opção -u, nas oróximas vees que for subir o branch master, basta fazer: **git push**)
* Copiar um repositório remoto para o seu computador:
  + Digitar: **git cone <URI do repositório remoto>** (copia o repositório inclusive o histórico de commits, simplesmente copiar os arquivos não traz a história de commits)
* Como atualizar o repositório loval remoto:
  + Digitar: **git pull origin master** (o git só deixa continuar um trabalho e deposis subi-lo para o repostório remoto, se mantiver a sequência coerente de commits;