XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
MUITO IMPORTANTE
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
1. Aprender a programar é essencialmente pôr o computador a fazer coisas e, no noso caso, ver essas coisas acontecerem na consola. Para isso é imprescindivel VER o que de facto SAI na consola e pensar um bocadinho sobre o que terá acontecido e como terá acontecido para a saída na consola ser aquela ... experimentar é fazer e ver se o que se fez coincide com o que nós achávamos que iria acontecer.

2. Caros alunos, não podem continuar a desligar durante as teóricas. Precisam desesperadamente de compreender o que eu digo e para isso precisam de compreender as novas palavras (e os conceitos que lhes estão subjacentes). Precisam de levantar a mão e dizer: pare que eu não estou a compreender ;-)

3. Nas aulas de PV:
Doravante NomeApelido é o seu nome e apelido por extenso e _NA são as iniciais do seu Primeiro Nome e Último Apelido.

IMPORTANTE: em programação cada caracter, cada espaço, cada virgula CONTA!! Se inadvertidamente apagar um (ou) ou { ou } ou ; ou , ou etc etc passará a ter um erro sintático e terá de estudar atentamente o código para descobrir o que é que apagou e onde é que está o erro. Muitas vezes para quem começa os erros são provocados porque escrevem código em sitios onde não o podem escrever (fora da classe, fora do método etc)

Este tutorial destina-se a permitir-lhe aprender a parte prática e a praticar repetindo as mesmas ações até que se tornem mecânicas e automáticas e as possa realizar sem perder tempo a pensar no que está a fazer (digamos que num jogo de cartas a matéria aqui abordada corresponde a conhecer o valor de cada carta nesse jogo. Sem conhecer de cor o valor das cartas nunca conseguirá aprender a jogar esse jogo.)

****
TUTORIAL RECUPERAÇÃO MINITESTE 1
****

****
Criar a pasta para guardar os materiais de PV
onde estejam rápida e facilmente acessíveis
****
0.A Crie uma pasta ProgVis2122 NA RAIZ DO SEU DISCO RÍGIDO (não é em repos, nem em documents, nem em Desktop nem em alguma recondita pasta do seu HD!!!) - é na RAIZ do DISCO!
0.B Dentro da pasta ProgVis2122 crie uma pasta Semana03

****
Criar um projeto e configurar o VS por forma
a conseguir ler bem o que escreve
****
1.A. Abra o VisualStudio
1.B. Crie um projeto denominado MT1_REC_NomeApelido (NOTE: o seu NOME E APELIDO POR EXTENSO, Não é _NA, nem _NT nem _NomeApelido, é o seu Nome e o seu Apelido por extenso tal como está escrito no separador do teste do moodle)

1.C Guarde o projeto na pasta Semana03 que criou em 1:A (não é em repos, nem onde lhe apetecer, é na pasta Semana03, se se enganou feche o projeto que criou e crie outro na pasta Semana03 que está dentro da pasta PV2122 que está na raiz do disco (está, não está?)
1.D vá ao menu Tools -> General -> Theme -> Light !!!! --> L I G H T <--
1.E Fonts & Colors size 14 !!!
1.F OK !!
O Sr. Está a aprender a programar e por isso tem dever BEM o que escreve, com caracteres 12 sobre sobre um fundo negro nem sequer consegue ler decentemente o que está a escrever.

****
DEFINIR CLASSES
2. NOTE QUE CADA CLASSE TEM DE ESTAR NUM FICHEIRO SEPARADO
3. NOTE QUE OS NOMES DAS CLASSES COMEÇAM COM MAIÚSCULA -> NomeDaCLasse
4. Na janela da direita (Solution Explorer) tem uma árvore com MT1_ MT1_REC_NomeApelido (tem?Está a vê-la?)
5. Criar um ficheiro para a classe Pessoa :
Prima o botão direito do rato (BDR) sobre o nome do projeto (MT1_REC_NomeApelido) add -> Class -> nome: Pessoa_NA -> next

****
Atributos e propriedades
6.A NOTE QUE OS ATRIBUTOS DE UMA CLASSE SÃO SEMPRE private e
6.B NOTE QUE OS IDENTIFICADORES DOS ATRIBUTOS COMEÇAM POR mINÚSCULA
6.C NOTE QUE AS PROPRIEDADES SÃO (por norma) public e
6.D NOTE QUE AS PROPRIEDADES COMEÇAM POR Maiúscula
7.A Crie três propriedades com atributos explicitos para String nome, String número, short ano, para tal:
7.B escreva propfull e prima duas vezes tab
7.C substitua int por String no atributo (o identificador que começa por minuscula)
7.D substitua myVar no atributo por nome_NA
7.E substitua MyProperty por Nome_NA
7.F deve obter isto:

```
private String nome;
public String MyProperty
{
   get { return nome; }
   set { nome = value; }
}
```

7.G repita as alineas 7.B a F para número_NA e ano_NA
(ATENÇÃO às maiusculas, como em String e minúsculas como em short)

****
Construtor com todos os parâmetros
8.A Criar um construtor com parâmetros para inicializar todos os atributos:
8.B Logo abaixo da última propriedade definida escreva ctor e prima duas vezes tab.
8.C Defina a lista de parametros com 3 parametros (String nome_NA, String numero_NA, byte ano_NA)
8.D Atribua cada um dos parâmetros à respetiva propriedade.
8.E deve obter isto:

```
public Pessoa(String nome_NA, String numero_NA, short ano_NA)
```

```
   {
      Nome = nome_NA; // ou this.Nome = nome_NA etc
      Numero = numero_NA;
      Ano = ano_NA;
   }
```

****
Construtor sem parametros
NOTE que as BPP impoem que só codifiquemos o construtor com mais parametros pelo que todos os restantes devem invocá-lo com os parametros recebidos e valores para os parametros que não seja recebidos. No caso do construtor sem parametros não é recebido nenhum parametro pelo que invocaremos o codificado com valores por omissão.
9.A Escreva ctor + tab tab e logo à direita do cabeçalho do construtor escreva:
: this ("Sem Nome", "Sem Numero, (short) 0)
9.B deve obter isto:

```
   public Pessoa() : this ("Sem Nome", "Sem Numero", (short) 0)
   {
   }
```

****
Outros Construtores
No caso de construtores que recebam parametros para inicializar apenas alguns dos atributos... vamos ver para um que receba só o nome:
9.C Escreva ctor + tab tab
9.D Defina na lista de parametros 1 parametro para o nome (String nome_NA)
9.E Logo à direita do cabeçalho do construtor escreva
: this (nome_NA, "Sem Numero, (short) 0)
9.F deve obter isto:

```
        public Pessoa(String nome_NA): this (nome_NA, "Sem Numero", 0)
        {
        }
```

****
CRIAR E USAR OBJETOS
10. Na janela da direita (solution explorer) clique duas vezes sobre o ficheiro da classe Program.cs que tem o método main: MT1_etc ... para o abrir
11.A Insira o cursor no método Main (antes da chaveta que fecha o metodo main)
11.B Escreva System.out.println("\nXXXXXXXXXXXXXX CRIAR E USAR OBJETOS");
12.A Crie um objeto do tipo Pessoa fornecendo valores para todos os atributos:

```
Pessoa p1 = new Pessoa ("123456789", "Diogo Fernandes", (short) 1999);
```

13. Imprima os estado da pessoa p1 usando as propriedades. (pode usar cw tab tab para inserir o ConsoleWriteLine ():

```
Console.WriteLine(p1.Nome );
Console.WriteLine(p1.Numero);
Console.WriteLine(p1.Ano);
```

TESTE: corra o programa premindo o botão verde a seguir ao martelo da barra de botões ... e veja na consola se o que saiu é o que estava à espra ou não. Se foi, excelente, caso contrário estude atentamente o código até perceber onde é que se enganou ;-)
14.A Crie um objeto do tipo Pessoa com valores por omissão:

```
Pessoa p2 = new Pessoa();
```
14.B Vamos criar um método que imprime os atributos para não repetirmos código ...

14.C Logo abaixo do Main (depois da chaveta que fecha o Main e antes da que fecha a classe defina o método imprime:

```
public static void Imprime(String nome_NA, String numero_NA, short ano_NA)
        {
            Console.WriteLine(nome_NA);
            Console.WriteLine(numero_NA);
            Console.WriteLine(ano_NA);
        }
```
14.D Use o método imprime para mostrar na consola o estado dos objetos p1 e p2:

```
imprime(p1.Nome, p1.Numero, p1.Ano);
imprime(p2.Nome, p2.Numero, p2.Ano);
```
14.E Crie um objeto do tipo Pessoa com um nome e os restantes valores por omissão:

```
Pessoa p3 = new Pessoa("José Saramago");
```
14.D Use o método imprime para mostrar na consola o estado do objeto p3:

```
imprime(p3.Nome, p3.Numero, p3.Ano);
```

****

PRATICAR definições de classes

(use intensivamente os snippets que aprendeu: propfull ctor cw etc ...)

15.A Repita os passos 5 a 9 para as classes com os atributos abaixo:

15.B Individuo_NA

com atributos para nome_NA (String), mês_NA de nascimento (byte) e ano_NA de nascimento (short).

15.B Aluno_NA

com atributos para nome_NA do aluno (String), codigo_NA de aluno (String) e se está inscrito_NA (bool).

15.C Trabalhador_NA

com atributos para nome_NA do trabalhador (String), numero_NA de segurança social (long) e qtdAnos_NA de trabalho na empresa (byte

****

PRATICAR a criação de objetos e o uso de métodos dos objetos

16. Repita os passo 10 a 14 para as classes que criou na alinea 15.

TESTE: corra o programa premindo o botão verde a seguir ao martelo da barra de botões.

****

Definir e usar um ToString para imprimir mais facilmente o estado do objeto

17.A Insira o cursor no final da classe Pessoa (antes da chaveta que fecha a classe) e escreva o código:

```
override public String ToString()
{
        return Nome + "\t" + Numero + "\t" + Ano;
}
```
17.B Repita o passo 17 para as classes que criou na alinea 15.

****

SINTAXE DA HERANÇA DE CLASSES

****

Vamos criar uma classe Docente como subclasse de Pessoa:
18.A Para criar a classe:
BDR em cima do nome do projeto -> Add -> Class -> nome: Docente_NA
18.B Para especificar que Docente_NA herda de Pessoa_NA (É uma pessoa) à direita do cabeçalho da classe escreva:
`: Pessoa_NA`
18.C Crie propriedades com atributos implicitos (sem o atributo explicito!) para Categoria_NA (String) e Grau_NA (String)
escreva prop + 2xTab => substitua MyVar por Categoria_NA e int por String. Faça o mesmo para Grau_NA.

****
Construtores em herança

****
Construtor com todos os parâmetros

NOTE que Docente herda de Pessoa (é uma Pessoa) pelo que para definir/construir um docente terei de fornecer 5 parametros (3 para os atributos de Docente enquanto Pessoa e 2 para os atributos de Docente)
19.A Criar o construtor => ctor+ 2 X tab
19.B Definir a lista de parametros (eu prefiro começar pelos da super classe ...):

```
public Docente_NA(String nome_NA,
                  String numero_NA,
                  short ano_NA,
                  String categoria_NA,
                  String grau_NA)
{

}
```

19.C Vamos invocar o construtor com todos os parametros da superclasse (Pessoa) com os parametros necessários logo à direita do cabeçalho da classe =>

```
public Professor_NA(String nome_NA,
                  String numero_NA,
                  short ano_NA,
                  String categoria_NA,
                  String grau_NA) : base ( nome_NA,    // <<-----
                                           numero_NA,  // <<-----
                                           ano_NA)     // <<-----
{

}
```

19.D e finalmente vamos inicializar as Propriedades de Docente_NA =>

```
public Docente_NA (String nome_NA,
                           String numero_NA,
                           short ano_NA,
                           String categoria_NA,
                           String grau_NA) : base ( nome_NA,
                                                    numero_NA,
                                                    ano_NA)
        {
            Categoria_NA = categoria_NA; // <<-----
            Grau_NA = grau_NA;           // <<-----
```

```
        }

**********************************
Construtor sem parametros
NOTE que as BPP impoem que só codifiquemos o construtor com mais parametros pelo
que todos os restantes devem invocá-lo com os parametros recebidos e valores
para os parametros que não seja recebidos. EXATAMENTE COMO FEZ NA ALINEA 9:
20.A    Escreva ctor + tab tab e logo à direita do cabeçalho do construtor
escreva =>
: this ("Sem Nome", "Sem Numero, (short) 0, "Sem Categoria", "Sem Grau")
20.B deve obter isto:
public Professor_NA() : this( "Sem Nome",
                              "Sem Numero",
                               (short)0,
                               "Sem Categoria",
                               "Sem Grau")
{

}

**********************************
Outros Construtores
No caso de construtores que recebam parametros para inicializar apenas alguns
dos atributos... vamos ver para um que receba só o nome e a categoria:
21.A    Escreva ctor + tab tab
21.B    Defina na lista de parametros 2 parametros para o nome e a categoria
(String nome_NA, String categoria_NA)
21.C    Logo à direita do cabeçalho do construtor escreva
: this (nome_NA, "Sem Numero, (short) 0, categoria_NA, "Sem Grau)
21.D    deve obter isto:
public Docente_NA(String nome_NA, String categoria_NA) : this(nome_NA,
                                                          "Sem Numero",
                                                          (short)0,
                                                          categoria_NA,
                                                          "Sem Grau")
        {

        }

**************************************
PRATICAR Herança:
22. Repita as alineas 18 a 21 para a classe Funcionário com propriedade com
atributo implicito para Secção (String).
23. Altere a classe Aluno para passar a herdar de Pessoa. Note que o nomeDoALuno
desaparece da classe Aluno uma vez que o nome já é herdado de Pessoa (Apague o
atributo e a prop na classe Aluno) e reescreva os construtores usando o que
aprendeu nas alineas 18 a 21.

****************************************
Reutilização dos métodos da super classe
24. Vamos definir o método ToString de Docente invocando o ToString da sua
classe base (superclasse) Pessoa:
25. Na classe Docente (no final antes da chaveta que termina a classe) defina um
método ToString =>
```

```
        override
        public String ToString()
        {
            return base.ToString() +
                " - " + Categoria_NA +
                " - " + Grau_NA;
        }
```

26. Repita a alinea 25 para as classes Aluno e Funcionário

```
*****************************************
*** NICE WORK! CONGRATULATIONS!         ***
*** YOU ARE NOW ABBLE TO START          ***
*** A LONG SAGA WITH C#PROGRAMMING!     ***
*****************************************
```

que é como quem diz:

```
*********************************************
*** EXCELENTE TRABALHO! PARABÉNS!           ***
*** O SR. ESTÁ AGORA CAPAZ DE COMEÇAR       ***
*** A LONGA EPOPEIA DE PROGRAMAÇÃO EM C#    ***
*********************************************
```