cencal centro de formação profissional para a indústria de cerâmica

Programação em C#

António Ramos | Outubro 2022

Co-Financiado por:

IEFP APICER associação portuguesa das indústrias de cerâmica e cristalaria ARO Associação Empresarial da Região Oeste REPÚBLICA PORTUGUESA POCH PORTUGAL 2020 UNIÃO EUROPEIA Fundo Social Europeu

cencal centro de formação profissional para a indústria de cerâmica

Programação orientada a objectos

- Divisão do programa em vários objectos
- Interação entre classes
- Criação através de "templates"

Criação de classes

- Keyword "class"
- Convenção PascalCasing
- Conteúdo (campos, propriedades, métodos) são criados dentro da classe

```
class Pessoa {
    // Campos, Propriedades e Métodos
}
```

Campos

- Semelhante às variáveis + o "access modifier"
  - public
  - private
  - protected
  - internal

```
class Pessoa {
    private string nomeDaPessoa;
    private int numeroFiscal;
}
```

Propriedades

- Utilizadas para dar acesso a campos
- Controlo sobre o direito à informação aos campos “private”.

Declaramos as propriedades, indicando o access modifier e o data type da propriedade.

```
public int NumeroFiscal
{
    // get
    // set
}
```

Propriedades

Têm dois métodos especiais denominados como "accessors":

- Getter
  - Retorna o valor do campo
- Setter
  - Define o valor do campo

```
get {
    return numeroFiscal;
}

set {
    numeroFiscal = value;
}
```

Nota: Podemos declarar os acecessors como privados.

Propriedades auto-implementadas

Versão "short-hand" utiliza-se quando não é necessário utilizar lógica adicional ao getter e setter.

Não é necessário definir os campos “private”, pois o compilador cria-os automaticamente.

```
public int NumeroFiscal { get; set; }
```

Métodos

- Blocos de código que efetuam determinada tarefa
- Criados dentro da classe com o nível de acesso e depois o tipo de retorno

```
public void BoasVindas()
{
    Console.WriteLine("Bem-vindo!! :)");
}
```

Métodos

Podem receber parâmetros e retornar dados:

```
public double CalculaCubo(double valor)
{
    return Math.Pow(valor, 3);
}
```

Overloading

Utilização de métodos com o mesmo nome, mas com assinaturas diferentes.

```
public void BoasVindas(string nome)
{
    Console.WriteLine("Bem-vindo {0}!! :)", nome);
}
```

Construtores

- Métodos especificos para a criação de objetos do tipo da classe
- 1º método a ser chamado para a criação de objetos e usado para inicializar os métodos e as variáveis.
- Deve ter o nome da Classe e não retornar nenhum valor.

```
public Pessoa(int nif)
{
    numeroFiscal= nif;
    Console.WriteLine($"NIF = {numeroFiscal}");
}
```

Instanciar objectos

Devemos instanciar as classes no nosso programa para a utilização dos objetos

```
NomeClasse nomeObjectoX = new NomeClasse(argumentos);
```

Keyword "static"

Usada para a criação de objectos sem a criação de objectos.

Nota: Em classes estáticas todos os membros da classe devem ser estáticos

Parâmetros (Tipos de Valor VS Tipos de Referência)

Tipo de valor:

Na passagem de uma variável para o método, qualquer alteração dessa variável será feita nesse contexto.

Quando o programa sai do método, a alteração deixa de ser válida.

Parâmetros (Tipos de Valor VS Tipos de Referência)

Tipos de Referência:

Se passamos a referência do variável, qualquer alteração é válida, mesmo depois do término do método.

Cont.

```
public void PassagemPorValor(int a)
{
  a = 10;
  Console.WriteLine("valor de a = {0}", a);
}

public void PassagemPorReferência(int[] b)
{
  b[0] = 5;
  Console.WriteLine("valor de b[0] = {0}", b[0]);
}
```

Parâmetro "ref"

Para passar as variáveis do tipo de valor como referência

```
public void PassagemPorValor(ref int a)
{
  a = 10;
  Console.WriteLine("valor de a = {0}", a);
}
```

Herança

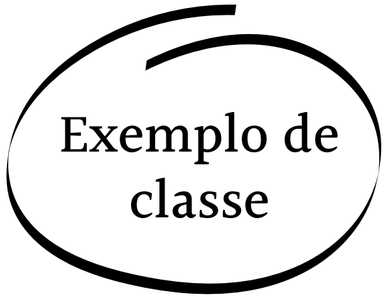
Permite a criação de uma classe, a partir de uma classe já existente.

Permite:

- Extensão de funcionalidades
- Reutilização de código
- Modificação do comportamento definido em outras classes.

Herança

A classe cujos membros são herdados é definido como classe “base” e a classe que herda esses membros é classe “derivada”.




```
class Membro
{
  protected int feeAnual;
  private string nome;
  private int membroID;
  private int membroDesde;
}
```

Classes derivadas (Subclasses)

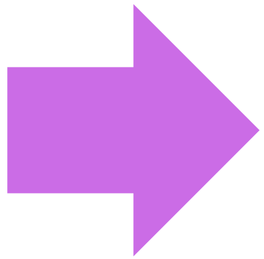
Herdam todas os membros públicos e “protected” da classe Mãe (“base”).

Para indicarmos que a classe é derivada de outra, usamos os dois pontos (:) a seguir ao nome da classe derivada.

Classes derivadas (Subclasses)

```
class Membro
{
  protected int feeAnual;
  private string nome;
  private int membroID;
  private int membroDesde;
}
```

Classe base



```
class MembroVIP : Membro
{

 }
```

Classe derivada