

SEMANAS:

- INICIO/APRESENTAÇÃO
- INTRODUÇÃO AOS ALGORITMOS E ESTRUTURAS DE DADOS
- ESTRUTURAS DE CONTROLO DE FLUXO EM C#
- CONCEITO DE CLASSE E OBJETO
- SEPARAÇÃO ENTRE INTERFACE PÚBLICA E IMPLEMENTAÇÃO PRIVADA
- ESTRUTURAS DE DADOS – INTRODUÇÃO
- ANÁLISE DE ALGORITMOS E COMPLEXIDADE
- ALGORITMOS DE ORDENAÇÃO
- ALGORITMOS DE ORDENAÇÃO (PARTE 2)
- PROVA DE AVALIAÇÃO
- LISTAS E FILAS
- TABELAS DE HASH
- TRATAMENTO DE COLISÕES
- ÁRVORES BINÁRIAS
- TIPOS GENÉRICOS E TRATAMENTO DE EXCEPÇÕES
- PROVA DE AVALIAÇÃO

Sumário:

- Conceito de classe e objeto
- Definição de uma classe e criação de instâncias em C#.

# Algoritmos e Estruturas de Dados

**CTeSP - Tecnologias e Programação de Sistemas de Informação 2020/21**



**Docente: [Ricardo Henriques](#).**

## Noção de classe e objeto

Programar um computador para dele tirar partido significa instruí-lo a compreender-nos.

Todas as formas de comunicação entre dois seres, duas pessoas, entre duas máquinas, ou entre uma máquina e uma pessoa assentam na noção de linguagem - é necessário que ambas as partes partilhem uma linguagem que serve de base de entendimento. A linguagem tem três aspetos que a caracterizam:

- O vocabulário;
- A sintaxe;
- A semântica;

No caso do C# estamos perante uma linguagem que permite instruir o computador a "compreender" como atuar sobre objetos (representações, ou seja abstrações de objetos concretos).

Qualquer que seja a linguagem que segue o PPO permite registar, numa entidade designada por **classe**, as características que do ponto de vista informático são relevantes para que o computador comprehenda e lide com alguma componente da realidade.

## Características de uma linguagem OO

A programação por objetos é antes de mais um paradigma atual de programação, o qual visa auxiliar o engenheiro de software a:

- Dominar a complexidade do problema.
- Escrever boas soluções (programas).

Apresenta características que a distinguem de qualquer outro paradigma convencional:

- Noções de classe, objeto, herança e polimorfismo.
- Abstração, modularidade, reutilização e extensibilidade.

- Permite o desenho e programação incremental (*bottom-up*).
- Proximidade entre o modelo e o objeto real modelado.

Um programa nesta abordagem é um modelo que simula (representa) a estrutura e o comportamento de parte do mundo real ou criativo. Um objeto agrupa dois eixos de raciocínio:

1. Dados - estrutura.
2. Operações - semântica funcional ou comportamento.

## Tudo são objetos?

As entidades do mundo real são, neste contexto, consideradas capazes de serem modeladas em objetos desde que sobre elas se possua informação sobre:

- Identidade (e autonomia).
- Estrutura - elementos caracterizadores
- Comportamento - propriedades dinâmicas: ações ou métodos.
- Interação - relação com outras entidades.

Vamos supor que se pretende representar num computador uma conta bancária (ou uma sua simplificação - ou seja algumas operações triviais). Um dos primeiros conceitos que associamos a uma conta bancária é a possibilidade de a abrir e nela depositar e levantar dinheiro.

- Três operações: abrir, depositar e levantar;
- Uma propriedade ou atributo da conta, o saldo, o dinheiro atualmente presente.

Rapidamente observamos que nos falta talvez mais qualquer coisa, nomeadamente, a quem pertence a conta: é um atributo, ou **propriedade** associada à conta, o(s) titular(es) da conta - da mesma forma que o saldo.

Mas também interessante é refletirmos um pouco mais profundamente na operação de abrir a conta... Iremos ver que este tipo de operação de iniciação é, o que no mundo o PPO se designa por um **construtor**.

Por seu turno as operações que mudam o estado do objeto, têm a designação em PPO de **métodos**.

## Definição de uma classe e criação de instâncias em C#

### Classes

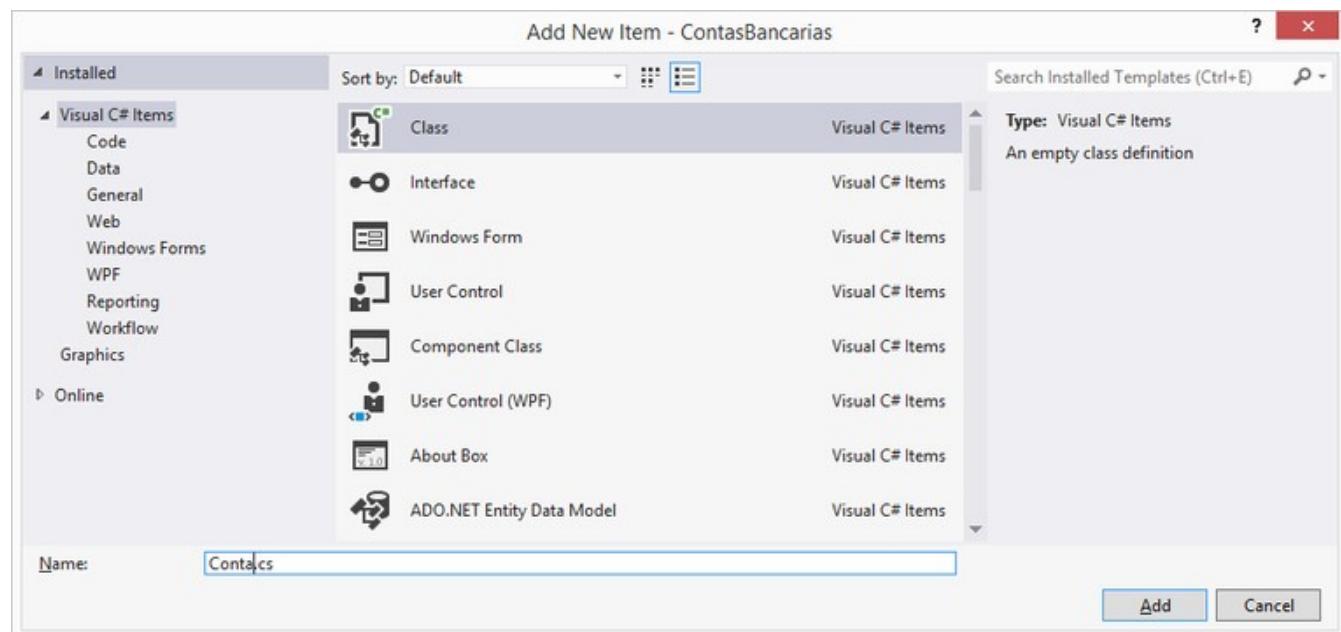
Define-se **classe** como um tipo criado pelo utilizador composto por **field data** (também designado por **member variables**) e por **members** os quais operam sobre estes dados (tal como construtores, métodos, eventos, etc.). Coletivamente estes dados representam o estado da instância da classe, também designada por **object** (objeto).

Uma classe é como um molde do qual se podem construir objetos concretos e diferentes entre si, as instâncias. Cada instância tem as suas características próprias (o seu estado representado em propriedades). Tal como os objetos do mundo real, estes interagem cor

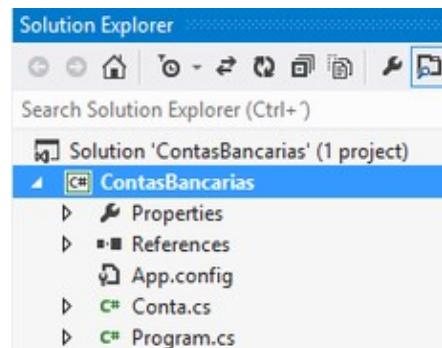
outros objetos através de "mensagens/eventos". Estas são uma sub-parte da funcionalidade que possuem e representam o que pode fazer com um objeto em si. A implementação desta funcionalidade é conseguida mediante a definição de métodos.

Vamos agora ver como o Visual Studio auxilia a que estes conceitos sejam definidos em C#.

Comece por criar um projecto C# em modo consola. Para adicionar a classe `Conta.cs` no âmbito do seu projecto faça, Project > Add Class..., o qual, dependendo da sua versão e dos componentes que tiver instalados abrirá uma janela semelhante à que se exibe abaixo:



Repare que no *Solution Explorer* surge agora a nova classe no âmbito do seu projeto:



A definição da mesma fica, para já, completa com:



```

namespace ContasBancarias {
    class Conta {
        private string nomeTitular;
        private double saldo;

        public Conta() {}

        public void Deposita(double quantia) {
            if (quantia > 0)
                saldo += quantia;
        }

        public double Levanta(double quantia) {
            if (saldo >= quantia) {
                saldo -= quantia;
                return quantia;
            } else
                return 0;
        }
    }
}

```

## Como criar um objeto?

A classe define o comportamento e as propriedades que caracterizam o estado dos objetos do tipo que ela define. Mas como se cria um objeto?

Tal como no C++, Java e noutras linguagens OO a criação de um objeto ou instância da classe é feita por recurso à primitiva `new`. Esta instrução efetua a alocação do objeto em memória e, a partir desse instante ele passa a existir e a interagir. No `main` desta aplicação, que abaixo se completa, pode ver não só a criação do objeto `c1`, o qual é uma instância da classe `Conta`, bem como a manipulação do objeto através do seu interface.

```

namespace ContasBancarias {
    class Program {
        static void Main(string[] args) {
            Conta c1 = new Conta();
            c1.Deposita(100);
        }
    }
}

```

O interface de um objeto é composto pelos métodos cuja assinatura é pública podendo, portanto, servir para consultar os valores do estado do objeto bem como para alterar o estado do mesmo.

Boa prática: embora seja possível declarar elementos *field data* de uma classe como *public* é boa prática no PPO que não sejam manipulados/consultados diretamente, mas apenas pelos métodos da própria classe, ie. tenham acesso *private*.

Nota: os modificadores de visibilidade serão estudados aquando da apresentação do estudo da separação entre interface pública e implementação privada.

## Construtores de objetos uma classe

Dado que os objetos tem estado, representado pelos valores das suas *member variables*, o utilizador irá, na generalidade dos casos, atribuir valores a essas propriedades. No caso da classe account faz todo o sentido ao criar a conta definir o titular e o valor de abertura ↑

com o qual foi aberta a conta.

Para num só passo criar e inicializar os valores de uma instância de classe, existe um tipo particular de métodos designado de construtores que é invocado indiretamente quando é criado o objeto com a primitiva `new`.

Toda a classe possui por omissão um construtor o qual pode ser reescrito. Por definição este construtor não tem parâmetros. Além de alocar em memória o objeto assegura que as propriedades são inicializadas com um valor por omissão do respetivo tipo de dados associado. Normalmente são valores zero (para os tipos numéricos) e *string* vazia no caso de se tratar de uma. Verifique...

Se o programador não definir um construtor o compilador C# garante um por omissão (a herança auxilia a compreender como isto surge), permitindo o utilizador instânciar a classe, ou seja efetuar a alocação em memória. Contudo se o programador definir um construtor o compilador assume que o programador tomou o assunto em mãos e silenciosamente remove a definição por omissão.

## Construtores personalizados

Pode ser redefinida a inicialização do construtor por omissão. Por exemplo poderemos acrescentar na definição da classe:

Este construtor, o qual reescreve o construtor por omissão define como valores iniciais um valor de abertura de 100 euros.

```
namespace ContasBancarias {
    class Conta {
        private string nomeTitular;
        private double saldo;

        public Conta() {
            saldo = 100;
        }
    }
}
```

Mas uma classe pode ter múltiplos construtores sendo diferentes, aos olhos do compilador do C# por terem número e tipos diferentes de parâmetros. Assim, o exemplo seguinte mostra dois construtores que vão ser adicionados à definição da classe `Conta`. Num deles define-se apenas o titular, ao passo que no outro define-se logo o titular e um valor inicial de depósito.

```
public Conta(string nome) {
    nomeTitular = nome;
}

public Conta(string nome, double valorAbertura) {
    nomeTitular = nome;
    saldo = valorAbertura;
}
```

## A utilização do `this`

O C# tal como todas as linguagens baseadas em C (e Java) facultam a primitiva `this` como forma de aceder à instância atual da classe. Um possível uso é para resolver um problema típico de ambiguidade que surge do exemplo seguinte:

```
class Pessoa {
```



```

String nome;
int idade;

    public void setNome(String nome) {
        nome = nome;
    }
}

```

No caso anterior nome em ambos os lados da atribuição é o argumento, e como tal não foi atribuído valor de estado à propriedade nome. Veja agora a solução:

```

class Pessoa {
    String nome;
    int idade;

    public void setNome(String nome) {
        this.nome = nome;
    }
}

```

Iremos na próxima matéria ver esta primitiva com outra utilização.

## Leitura e escrita de ficheiros

Vamos agora abrir um parêntesis para tocar num pequeno capítulo sobre a persistência de dados, no caso concreto, através da leitura e escrita de ficheiros em modo de texto, em C#.

### Persistência

*designa-se pela capacidade de conseguir salvar (guardar) informação ou objetos de forma perseverante. A escrita em disco ou num qualquer suporte que permita que mesmo desligado de qualquer fonte de energia a informação possa posteriormente ser lida e recuperada.*

Nota: neste exemplo não se teve o esforço de separar a componente de apresentação com a leitura e escrita em ficheiros.

```

static void Main(string[] args) {
    string[] aEscrever = {"1;Serafim Saudade; 1975",
                          "2;Zulmira Pereira; 1908",
                          "3;Genaro Silva;1992" };

    EscreverFicheiro("resultado.txt", aEscrever);
    LerFicheiro("entrada.txt");
}
(método Main)

```

Pode observar a iniciação de um *array* (matéria a aprofundar em breve) com três *strings*. A invocação ao método `EscreverFicheiro` recebe o nome do ficheiro a criar no persistente (disco) e o conjunto de *strings* a escrever.



```

private static void EscreverFicheiro(string nomeFicheiro, string[] aEscrever) {
    using (StreamWriter sw = File.CreateText(nomeFicheiro)) {
        foreach (string s in aEscrever) {
            sw.WriteLine(s);
        }
        sw.Flush();
    }
}
(método EscreverFicheiro)

```

A escrita no ficheiro passou por constituir um ficheiro de texto, através da primitiva `CreateText`; ao mesmo tempo colocou-se a referência desse ficheiro atribuída a um objeto `StreamWriter`.

## Serialização

*capacidade de escrever (um objeto) numa stream (seja num ficheiro ou numa conexão numa rede) e posteriormente ler e reconstruir o objeto.*

### Stream vs. pipe

*Nos antigos terminais Unix o I/O era implementado por recurso a uma fila. Nas versões mais antigas tais abstrações designavam-se de clists e posteriormente de streams. Em qualquer dos casos visam implementar uma parte da ligação entre processos (por exemplo um descritor de um ficheiro) e o controlador (driver) do device (por exemplo uma porta série, ou um terminal de texto, ou um teclado). O termo pipe é também uma forma de fila, normalmente associada à abstração de comunicar/ligar processos, como são disso exemplos os sockets.*

```

private static void LerFicheiro(string nomeFicheiro) {
    using (StreamReader sd = File.OpenText(nomeFicheiro)) {
        string linha;
        Console.WriteLine("===== inicio =====");
        while ((linha = sd.ReadLine()) != null) {
            Console.WriteLine(linha);
        }
        Console.WriteLine("===== fim =====");
    }
}
(método LerFicheiro)

```

A leitura a partir do ficheiro é um processo semelhante: abre-se o descritor do ficheiro e a sua referência é passada ao objecto `StreamReader`. Através dele o texto flui pela instrução `ReadLine`; é feita a retenção, linha a linha, do valor lido numa variável auxiliar e enviado o resultado para a consola.

Nota: a *serialização* é processo importante e quando se refere ao caso de um objeto que é escrito em disco, tal designa-se de *object persistence*.

## Exercício 5:

**Resolva em grupos de 3 ou 5 elementos. Entrega no moodle até às 23h59, do dia 09 Abril 2021, da solução (pasta completa da solução).**

Para efectuar o cálculo do salário líquido deve considerar:

- Valor do salário base;
- A situação matrimonial;
- Número de titulares;



- O número de dependentes;
- Deficiências.

O caso exemplificativo está disponível no Diário da República com as tabelas de [retenção](#) sobre rendimentos do trabalho dependente.

**Aplique apenas (para simplificar) o cálculo para rendimentos do trabalho, dois titulares; sem considerar casos em que o titular ou algum dos dependentes é deficiente, nem casos de pensão.**

Utilize a noção de classe para caracterizar a pessoa, e a composição de um agregado familiar.

- Apenas deve usar a noção de classes, objetos e *array* (uma ou mais dimensões);
- A entrega por grupo deve ser feita no *moodle* (tópico a consultar no *moodle* da disciplina);
- A entrega deve ser feita apenas por um elemento do grupo, e identificando o grupo (ver listagem de grupos);
- Faça um .ZIP com a pasta de toda a solução.

#### **Escalonamento das entregas no moodle e apresentações**

- Entrega: até às 18h00 de 22 Mar 2021 - composição do grupo; basta um elemento enviar um .pdf com os nomes, e números, seu e dos colegas de grupo; (5%)
- Entrega: até 18h00 de 25 Mar 2021 - esboço dos *CRC cards*(5%); documento .pdf.
- Entrega: até 18h00 de 31 Mar 2021 - versão definitiva *CRC cards*(5%); documento .pdf.
- Entrega até às 23h59 de 09 Abr 2021 - submissão da solução (pasta completa da solução), em .zip(55%).
- Apresentação na aula de 13 Abr 2021 (30%).

*MRH, 2020/21*

