Ficha de trabalho #4

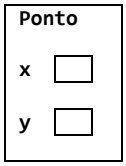
Classe e objeto
Componentes de uma classe
Visibilidade e encapsulamento
JavaDoc

Uma **classe** é uma estrutura especial que serve de molde (podemos pensar numa classe como se fosse um formulário) para a criação de **objetos** (podemos pensar que cada objeto corresponde a um formulário preenchido) com estrutura e comportamentos comuns. Ao definir uma classe, estamos a criar um tipo de dados. Os objetos criados a partir deste novo tipo (classe) são **instâncias** da classe, ou seja: são criados através da instanciação da classe. Os componentes de uma classe são atributos, construtores e métodos.

Os **atributos** são os elementos que definem a estrutura da classe. De forma a obedecer ao princípio do encapsulamento, os atributos são antecedidos pela palavra reservada **`private`**, de forma a não serem visíveis a partir de fora da classe, para que não sejam acedidos/modificados. De forma a ser possível aceder e modificar os valores dos atributos, necessitamos de aceder à interface pública da classe (ver secção sobre "métodos"). Os atributos podem ser de diferentes tipos:

- **Atributos de instância**: podem ser diferentes em todas as instâncias da classe. No exemplo apresentado abaixo, na classe `Ponto`, temos dois atributos de instância: `x` e `y`.
- **Atributos de classe**: têm o mesmo valor para todas as instâncias da classe. São identificados pela palavra **`static`**. Na classe `Ponto` temos três atributos de classe:
  - `nrPontos`, cujo valor inicial é 0 – utilizado para manter o número de pontos criados.
  - Constantes, identificadas pela palavra **`final`**, `ABCISSA_OMISSAO` e `ORDENADA_OMISSAO`, ambas com o valor 0.0 (ver secção sobre "construtores")

Utilizando a analogia do formulário na classe Ponto, teríamos o seguinte:

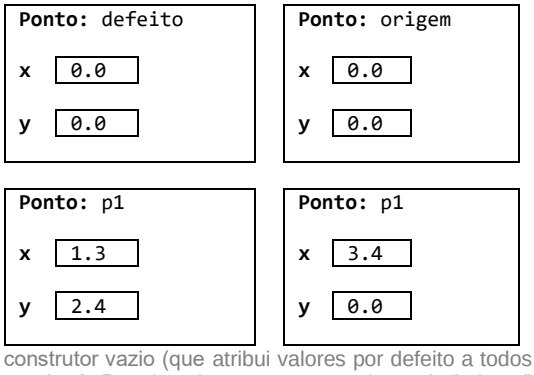


Os **construtores** são conjuntos de instruções que permitem construir um objeto da classe (instanciar um objeto, criar uma instância da classe). É usual uma classe ter sempre o construtor **vazio** e o construtor **completo**. Na classe `Ponto` temos os seguintes construtores:

- `Ponto(int x, int y)`: cria um ponto com base nas coordenadas `x` e `y` passadas por parâmetro. Incrementa (aumenta em 1 unidade) o valor do atributo `nrPontos`. É o construtor **completo**, porque preenche os valores de todos os atributos.
- `Ponto()`: cria um ponto com base em valores definidos por defeito para as coordenadas `x` e `y` (`ABCISSA_OMISSAO` e `ORDENADA_OMISSAO`). Incrementa o valor do atributo `nrPontos`. É o construtor **vazio**, ou por defeito, porque preenche os valores de todos os atributos com valores por defeito.
- `Ponto(int x)`: construtor que cria um ponto com base na coordenada `x` passada por parâmetro, definindo a coordenada `y` como um valor por defeito. Incrementa o valor do atributo `nrPontos`.

Após implementarmos os construtores, podemos já instanciar objetos de uma classe, numa classe de teste que irá conter a função `main(String[] args)`. Ao fazê-lo, estamos a criar objetos do tipo da classe que estamos a implementar, ou seja, objeto do novo tipo que criámos. Continuando com o exemplo da classe `Ponto`, podemos criar uma classe `TestePonto`, para instanciar objetos da classe `Ponto` e testar as suas funcionalidades. Para já, vamos apenas criar algumas instâncias da classe `Ponto`:

```
public class TestePonto {
    public static void main(String[] args) {
        Ponto defeito = new Ponto();
        Ponto origem = new Ponto(0, 0);
        Ponto p1 = new Ponto(1.3, 2.4);
        Ponto p2 = new Ponto(3.4);
    }
}
```



Começamos por criar um ponto chamado "defeito" utilizando o construtor vazio (que atribui valores por defeito a todos os atributos), pelo que as coordenadas x e y ficam ambas com o valor 0. Depois, criamos um ponto chamado "origem" em que utilizamos o construtor completo, definindo explicitamente, que as coordenadas x e y são ambas 0. Na instrução seguinte voltamos a usar o construtor completo, desta vez com os valores 1.3 e 2.4 para os valores das coordenadas x e y, respetivamente. Por fim, utilizamos o construtor parcial para criar um ponto chamado "p2" com o valor 3.4 na coordenada x e o valor por defeito (0) na coordenada y.

Os **métodos** são funções associadas às classes e aos objetos, que constituem a interface pública de uma classe, ou seja: a forma como se pode aceder à estrutura e comportamentos da classe a partir de fora. À semelhança do que acontece com os atributos, os métodos podem ser:

- **Métodos de instância**: métodos que acedem aos atributos de instância (leitura e/ou escrita).
- **Métodos de classe**: métodos que não acedem aos atributos de instância através de `this`, métodos que apenas acedem aos seus parâmetros, ou então métodos que acedem aos atributos de classe. São identificados pela palavra reservada **`static`** e são chamados associados à classe em vez de serem chamados associados a um objeto da classe.

Na classe `Ponto` temos alguns métodos de instância, como por exemplo:

```
public class TestePonto {
    public static void main(String[] args) {
        Ponto p1 = new Ponto(1.3, 2.4);
        System.out.println(p1.getX());
    }
}
```

Chamamos o método `getX` associado ao objeto p1 para obter o valor do atributo x desse objeto. O resultado do print será 1.3

```
public class TestePonto {
    public static void main(String[] args) {
        Ponto origem = new Ponto(0, 0);
        Ponto p1 = new Ponto(1.3, 2.4);
        System.out.println(p1.distanciaEuclideana(origem));
    }
}
```

Chamamos o método `distanciaEuclideana` associado ao ponto p1 para obter a distância euclidiana desse ponto ao ponto passado por parâmetro. O resultado do print é 2.7294688127912363. Quando o método estiver a ser executado, dentro da classe, o this diz respeito ao ponto a partir do qual o método foi chamado (neste caso, p1). O ponto que é passado por parâmetro é o ponto origem.

```
public class TestePonto {
    public static void main(String[] args) {
        Ponto p1 = new Ponto(1.3, 2.4);
        System.out.println(p1.distanciaOrigem());
    }
}
```

Chamamos o método `distanciaOrigem` associado ao ponto p1 para obter a distância desse ponto à origem do referencial. O resultado do print é o mesmo que no exemplo anterior, porque em ambos os casos estamos a calcular a distância de p1 a um ponto com coordenadas (0,0).

Esta classe tem também alguns métodos de classe:

```java
public class TestePonto {
    public static void main(String[] args) {
        System.out.println(Ponto.getNrPontos()); // 0
        Ponto defeito = new Ponto();
        System.out.println(Ponto.getNrPontos()); // 1
        Ponto origem = new Ponto(0, 0);
        System.out.println(Ponto.getNrPontos()); // 2
        Ponto p1 = new Ponto(1.3, 2.4);
        System.out.println(Ponto.getNrPontos()); // 3
        Ponto p2 = new Ponto(3.4);
        System.out.println(Ponto.getNrPontos()); // 4
    }
}
```

A primeira instrução consiste em mostrar o número de pontos, utilizando para isso o método de classe `getNrPontos`, que é chamado associado à classe Ponto, em vez de ser chamado a partir de um objeto (como acontece para os métodos de instância).
Como ainda não foi criado nenhum ponto, o resultado desse print é 0. À medida que vamos criando pontos, o atributo de classe `nrPontos` vai sendo incrementado para guardar o número de pontos criados. No final, após serem criados todos os pontos, o resultado do print é 4, porque foi esse o número de pontos que foram criados. Note-se que, neste caso, não existe método `setNrPontos`, porque este não faz sentido, uma vez que será atualizado aquando da criação de pontos.

```java
public class TestePonto {
    public static void main(String[] args) {
        System.out.println(Ponto.distanciaEuclideana(3.5,7.4,2.3,3.1));
    }
}
```

Chamamos o método de classe `distanciaEuclideana`, associada à classe `Ponto`, para calcular a distância entre dois pontos definidos pelas suas coordenadas passadas por parâmetro. Neste caso, estamos a obter a distância entre os pontos (3.5, 7.4) e (2.3, 3.1) e o resultado será 3.981205847478877. Note-se que neste caso não são usados os atributos da classe `Ponto` (`this`).

```java
public class TestePonto {
    public static void main(String[] args) {
        Ponto p1 = new Ponto(1.3, 2.4);
        Ponto p2 = new Ponto(3.4);
        System.out.println(Ponto.distanciaEuclideana(p1,p2));
    }
}
```

Chamamos o método de classe `distanciaEuclideana`, associada à classe `Ponto`, para calcular a distância entre dois pontos passados por parâmetro. Neste caso, estamos a obter a distância entre os pontos p1 e p2 e o resultado será 3.1890437438203945. Note-se que neste caso também não são usados os atributos da classe `Ponto` (`this`).

É usual as classes conterem os seguintes métodos:

- **Getters**: métodos que permitem ler o valor dos atributos (ex: `getX()` para obter o valor do atributo x)
- **Setters**: métodos: métodos que permitem modificar o valor dos atributos (ex: `setX(int x)` para redefinir o valor do atributo x para o valor passado por parâmetro)
- **`equals(Object outro)`**: método que permite verificar se o objeto a partir do qual é chamado (`this`) é igual ao objeto passado por parâmetro (`outro`). Retorna `true` caso sejam iguais e `false` caso contrário.
- **`toString()`**: método utilizado para retornar a descrição textual legível de um objeto
- **Outros métodos**

Regras:

- Cada ficheiro apenas pode conter uma classe
- O nome da classe tem de coincidir com o nome do ficheiro (ex: a classe Ponto encontra-se no ficheiro Ponto.java) e com o nome do construtor (ex: os construtores da classe Ponto são Ponto(), Ponto(int x, int y) e Ponto(int x))
- Os atributos são sempre private
- Os métodos podem ser public

Convenções de nomenclatura:

- O nome de uma classe inicia-se por letra maiúscula
- O nome de uma instância / um objeto inicia-se por letra minúscula
- O nome de uma constante é composto por letras maiúsculas

Classes e objetos podem ser representados através de **UML**, utilizando **diagramas de classes** e **diagramas de objetos**, respetivamente. Abaixo encontra-se um exemplo para a classe `Ponto` e para os objetos `Ponto` criados no exemplo acima.

Diagrama de classes

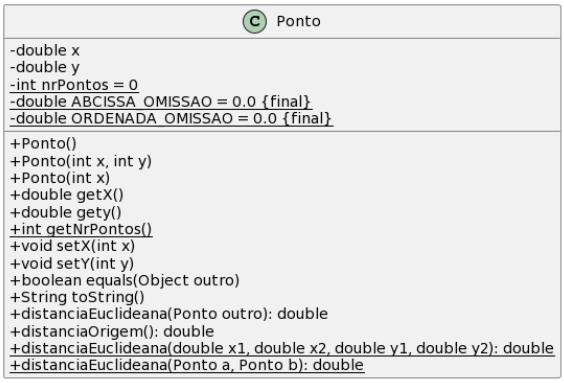
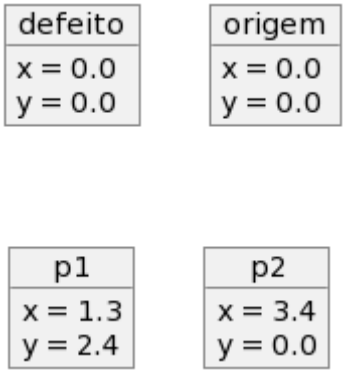


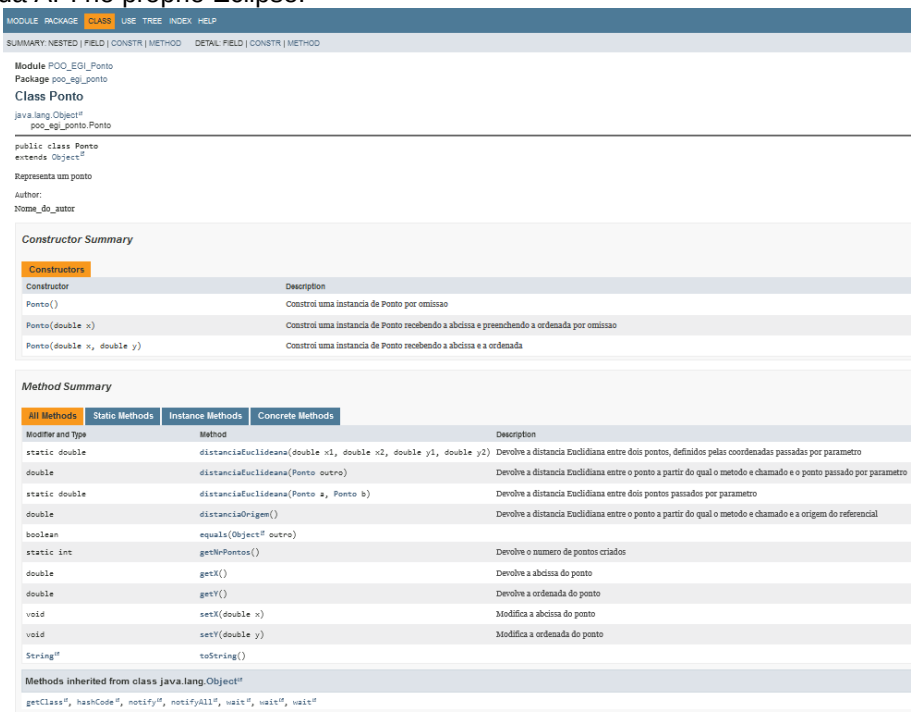
Diagrama de objetos



A documentação de código é uma tarefa muito importante. Podemos recorrer ao **JavaDoc** para documentar a API (*Application Programming Interface*) do nosso programa. Para isso, introduzimos código especial, parecido com os comentários, que permitem identificar as funcionalidades das classes e dos seus componentes.
Os comentários JavaDoc iniciam-se com /** e terminam com */. Há algumas palavras reservadas a ter em conta:

- `@param:` identifica parâmetros passados aos métodos
- `@return:` identifica qual é o retorno de um método

Após introdução do código para documentação JavaDoc, é possível gerar a documentação no formato HTML (formato de uma página Web), com links clicáveis (No Eclipse: Project > Generate JavaDoc). Em anexo a este documento encontra-se a classe Ponto completa, incluindo documentação JavaDoc. De seguida apresenta-se o output da geração da documentação JavaDoc e seguem-se alguns exemplos do código JavaDoc, do output na documentação, e da consulta da API no próprio Eclipse.



MODULE PACKAGE CLASS USE TREE INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Module POO_EGI_Ponto
Package poo_egi_ponto

Class Ponto

java.lang.Object
poo_egi_ponto.Ponto

```
public class Ponto
extends Object
```

Representa um ponto

Author:
Nome_do_autor

Constructor Summary

Constructors

Constructor	Description
Ponto()	Constroi uma instancia de Ponto por omissao
Ponto(double x)	Constroi uma instancia de Ponto recebendo a abcissa e preenchendo a ordenada por omissao
Ponto(double x, double y)	Constroi uma instancia de Ponto recebendo a abcissa e a ordenada

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

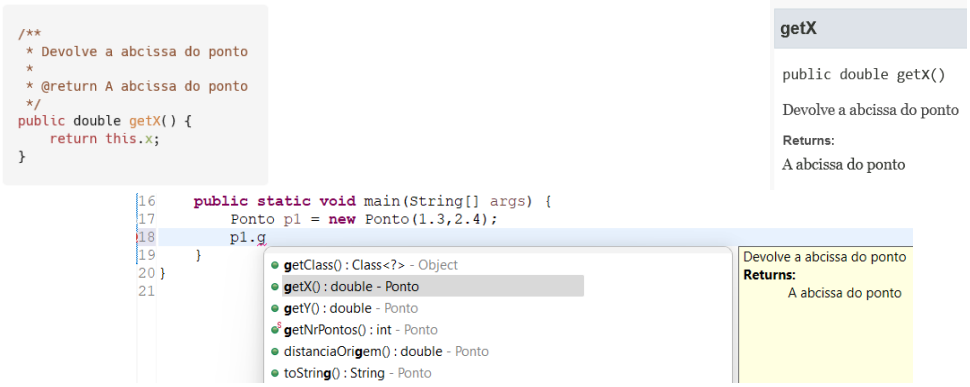
Modifier and Type	Method	Description
static double	distanciaEuclideana(double x1, double x2, double y1, double y2)	Devolve a distancia Euclidiana entre dois pontos, definidos pelas coordenadas passadas por parametro
double	distanciaEuclideana(Ponto outro)	Devolve a distancia Euclidiana entre o ponto a partir do qual o metodo e chamado e o ponto passado por parametro
static double	distanciaEuclideana(Ponto a, Ponto b)	Devolve a distancia Euclidiana entre dois pontos passados por parametro
double	distanciaOrigem()	Devolve a distancia Euclidiana entre o ponto a partir do qual o metodo e chamado e a origem do referencial
boolean	equals(Object outro)	
static int	getNrPontos()	Devolve o numero de pontos criados
double	getX()	Devolve a abcissa do ponto
double	getY()	Devolve a ordenada do ponto
void	setX(double x)	Modifica a abcissa do ponto
void	setY(double y)	Modifica a ordenada do ponto
String	toString()	

Methods inherited from class java.lang.Object

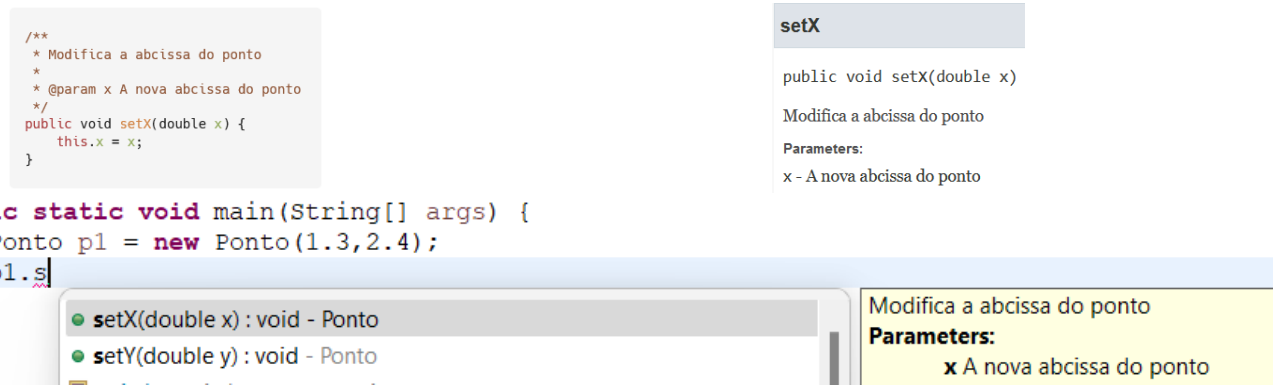
getClass, hashCode, notify, notifyAll, wait, wait, wait

JavaDoc para documentação de um dos construtores, em que se define a descrição do construtor o que representam cada um dos dois parâmetros recebidos.

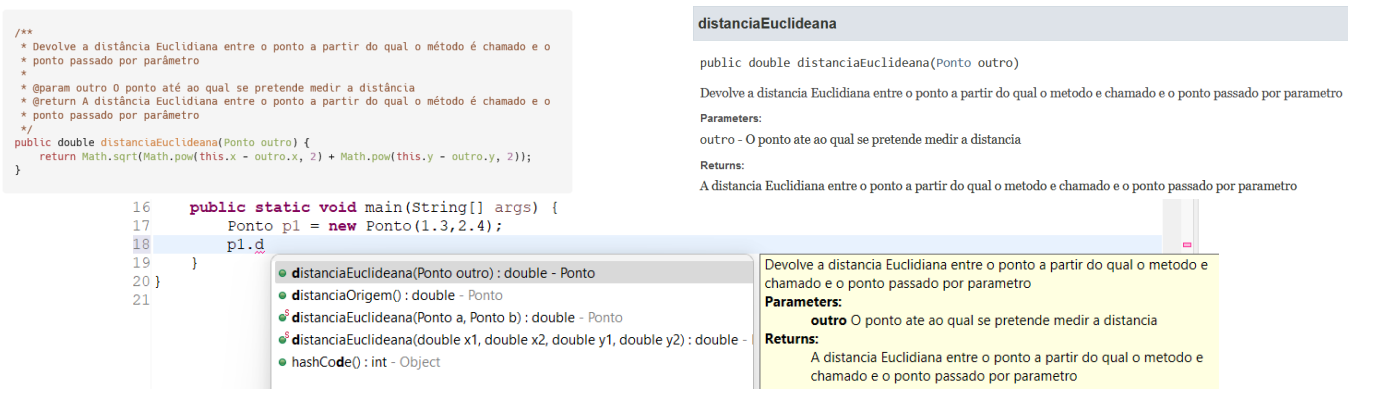
JavaDoc para documentação de um método, que define a descrição do método e o que representa o resultado retornado



JavaDoc para documentação de um método, que define a descrição do método e o que representa o parâmetro recebido



JavaDoc para documentação de um método, que define a descrição do método, o que representa o parâmetro recebido, e ainda o que representa o resultado retornado



1. Pretende-se criar um programa em Java, seguindo as regras da POO e com documentação recorrendo a JavaDoc, para representar pessoas.

   1.1. Escrever um programa (criar um projeto chamado pessoas e a função main na classe TestePessoa) que pergunte o nome e mostre o nome introduzido. Ex:

Input	Output
`Qual é o nome? Ana`	`O nome é Ana`

   1.2. Alterar o programa para utilização de POO:

      1.2.1. Criar uma classe Pessoa, com:
      - Atributo de instância: nome, do tipo String
      - Construtor: completo
      - Getter e setter de nome

      1.2.2. Na classe TestePessoa, que contém a função main, testar as funcionalidades implementadas. Na função main:
      - Criar uma pessoa p1, a partir do nome introduzido pelo utilizador
      - Tentar imprimir p1

      1.2.3. Criar o método toString(), que retorna a descrição textual legível de Pessoa (ex: `O nome é Ana`) e voltar a tentar imprimir p1

   1.3. Adaptar o programa de forma a ser registada também a idade, adicionando/alterando:
   - Atributo de instância: idade, do tipo int
   - Construtor: adaptar o construtor para receber também a idade
   - Getter e setter de idade
   - Método toString(): passa a mostrar o nome e a idade (ex: `Ana tem 30 anos`)
   - Método idadeFutura(int x) que retorna qual será a idade da pessoa daqui a x anos
   - Método idadePassada(int x) que retorna qual era a idade da pessoa há x anos

   1.4. Adaptar a classe TestePessoa de forma a testar as novas funcionalidades implementadas, escrevendo o código necessário para:
   - Criar três pessoas: pessoa1, pessoa2 e pessoa3 (sem ler input do utilizador)
   - Mostrar a informação das 3 pessoas
   - Alterar a idade de pessoa2 para a idade de pessoa1 e mostrar a pessoa2
   - Mostrar quantos anos terá a pessoa3 daqui a 5 anos
   - Mostrar quantos anos tinha a pessoa 2 há 10 anos

2. Pretende-se criar um programa em Java, seguindo as regras da POO e com documentação recorrendo a JavaDoc, para representar livros.

   2.1. Para cada livro pretendemos guardar o título (String), autor (String), ano de publicação (int), editor (String), e preço (double). Deve ser implementado o construtor, os getters e setters e o método toString() para retornar a descrição textual de um livro. Deve ainda ser implementado o seguinte método:

   - `precoComDesconto(double pcDesconto)` que retorna o preço ao aplicar uma percentagem de desconto

   2.2. Implementar uma classe de teste para testar as funcionalidades implementadas, escrevendo o código necessário para:

   2.2.1. Criar dois livros:

	Título	Autor	Ano	Editor	Preço
livro1	POO em Java	Mário Martins	2017	FCA	37,75 €
livro2	Programação em Java	Pedro Coelho	2012	FCA	40,99 €

   2.2.2. Mostrar a descrição textual dos livros

   2.2.3. Alterar o ano de publicação do livro2 para 2016

   2.2.4. Mostrar a descrição textual do livro2

   2.2.5. Mostrar o preço do livro1 com desconto de 20%

3. Pretende-se criar um programa em Java, seguindo as regras da POO e com documentação recorrendo a JavaDoc, para representar atletas inscritos num ginásio.

   3.1. Para cada atleta pretendemos guardar nome (String), idade (int), género (char), peso (em kg, double) e altura (em m, double). Deve ser implementado o construtor, os getters e setters e o método toString() para retornar a descrição textual de um atleta. Devem ainda ser implementados os seguintes métodos:

   - `calcularIMC()`, que calcula o IMC do atleta de acordo com o seu peso e a sua altura e a fórmula apresentada abaixo
   - `determinarObesidade()`, que determina o grau de obesidade do atleta de acordo com o seu IMC e com a tabela apresentada abaixo
   - `isSaudavel()`, que determina se um atleta é saudável (tem peso normal) com base nos métodos anteriores

   3.2. Implementar uma classe de teste para testar as funcionalidades implementadas, escrevendo o código necessário para:

   3.2.1. Criar dois atletas: atleta1 e atleta2

   3.2.2. Mostrar os dois atletas

   3.2.3. Mostrar o IMC do atleta1

   3.2.4. Mostrar o grau de obesidade do atleta2

   3.2.5. Mostrar se os atletas são saudáveis

Cálculo do IMC

$$IMC = \frac{peso}{altura^2}$$

Grau de obesidade

IMC	Classificação
< 18.4	Abaixo do peso
[18.5, 25.0[	Peso normal
[25.0, 30.0[	Sobrepeso