2º Teste de Introdução à Programação

Duração: 2 horas

Departamento de Informática, FCT NOVA
18 de Janeiro de 2021

(O teste tem **16** páginas e **5** grupos)

Número: _____ **Nome:** _____

Regras do Teste

- Responda a cada pergunta na caixa atribuída para o efeito. Se faltar espaço, coloque a continuação numa zona livre do enunciado, **numa folha do mesmo grupo da pergunta**.
- Identifique todas as folhas do enunciado no local apropriado.
- As respostas podem ser escritas a lápis.
- Em cima da mesa, só pode ter o documento de identificação e material de escrita (caneta, lápis, borracha).
- O teste é sem consulta. Não pode consultar quaisquer elementos para além deste enunciado.
- Não pode usar dispositivos electrónicos (como calculadoras, telemóveis, *tablets*, *smartwatches* e portáteis).
- Não pode ter folhas de rascunho. Use os espaços para rascunhos no enunciado.
- Não pode desagrafar o enunciado.
- Antes de começar a resolver cada grupo, leia o enunciado das perguntas do grupo com atenção, do princípio até ao fim.
- Não há esclarecimento de dúvidas. Se suspeitar que o enunciado tem algum erro, deve avisar o docente.
- Só pode sair da sala quando o teste terminar.
- Se pretender que o seu teste não seja avaliado, escreva "Desisto" na zona de identificação desta página.

No Final do Teste

- Verifique que todas as folhas estão identificadas com o seu número e o seu nome.

(Esta zona é para rascunho)

Enunciado comum aos Grupos 1 a 4

Pretende-se implementar uma aplicação para gerir artistas e as suas obras. As entidades, respectivos atributos e funcionalidades relevantes são descritos abaixo. Note que o que se pede concretamente para resolver está bem identificado (por caixas) no enunciado. **Pode preencher qualquer caixa mesmo que não tenha preenchido outras.**

As obras serão instâncias da classe **`Oeuvre`**, têm autor, nome, ano de concepção, preço e indicação se já foram vendidas. A classe permite obter o autor, o nome, o ano de concepção e o preço da obra, permite saber se a obra está ou não vendida, permite passar o estado da obra de não vendida para vendida, permite saber se a obra tem o mesmo autor e o mesmo ano que outra e permite comparar a obra com outra, dizendo se é maior ou não.

Uma obra é maior que outra se o seu autor for alfabeticamente maior que o da outra; caso sejam do mesmo autor, se o seu ano for maior que o da outra; finalmente, se ambas também forem do mesmo ano, se o seu nome for alfabeticamente maior que o da outra.

Os artistas serão instâncias da classe **`Artist`**, têm nome e guardam uma colecção de obras da sua autoria, que começa vazia e pode aumentar (não tendo limite). A classe permite obter o valor total ganho com a venda das obras do artista e o seu ano mais produtivo (quando fez mais obras), desde que tenha obras. Se houver vários anos em que a produtividade foi a maior, o ano mais produtivo é o mais recente desses.

Pretende-se também listar todas as obras de um dado ano de um artista. Para isso, usa-se a classe **`IteratorOeuvresArtistYear`** que, dados um vector de obras (possivelmente de artistas diferentes), o nome de um artista e um ano, constrói um iterador que permite aceder a todas as obras criadas nesse ano pelo artista.

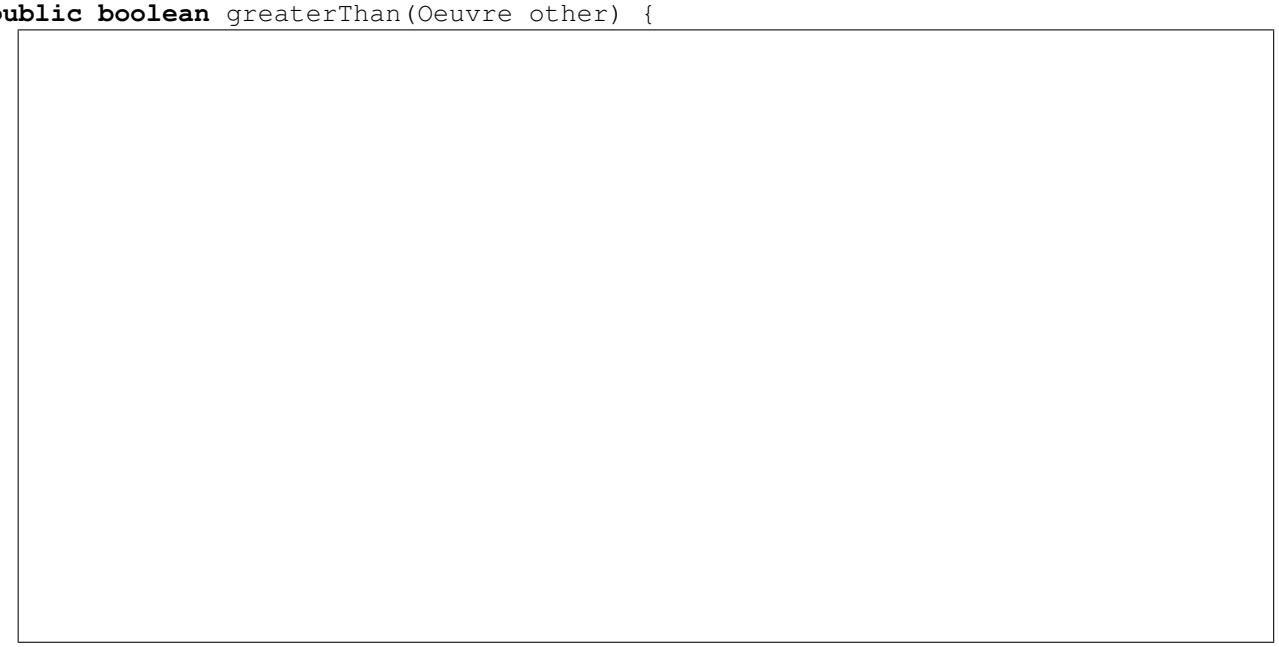
A aplicação também permite carregar um vector de objectos **`Oeuvre`**, com obras de um único artista, a partir dum ficheiro e gravar num ficheiro a informação relevante das obras de um artista. O formato do ficheiro é descrito no Grupo 4.

Grupo 1 [1.5 valores]

Complete o corpo do método `greaterThan` da classe `Oeuvre`:

```
public class Oeuvre {
  private String author;
  private String name;
  private int year;
  private double price;
  private boolean sold;

  // Cria uma obra com autor, nome, ano e preco,
  // e define o seu estado como por vender.
  // @pre author != null && name != null && price > 0
  public Oeuvre(String author, String name, int year, double price) {
    this.author = author;
    this.name = name;
    this.year = year;
    this.price = price;
    sold = false;
  }

  public String getAuthor() {
    return author;
  }

  public String getName() {
    return name;
  }

  public int getYear() {
    return year;
  }

  public double getPrice() {
    return price;
  }

  public boolean isSold() {
    return sold;
  }

  // Coloca o estado da obra como vendida.
  public void sell() {
    sold = true;
  }

  // @return true, se o autor, o nome e o ano forem iguais, respectivamente,
  //         ao autor, ao nome e ao ano de other;
  //         false, no caso contrario.
  public boolean equals(Oeuvre other) { ... }
```

```
  /* Compara obras.
   *
   * @return true, se o autor for alfabeticamente maior que o autor de other;
   *         caso sejam do mesmo autor, se o ano for maior que o ano de other;
   *         caso sejam do mesmo autor e do mesmo ano, se o nome for
   *         alfabeticamente maior que o nome de other;
   * @return false, nos restantes casos.
   *
   * @pre other != null
   */
  public boolean greaterThan(Oeuvre other) {
```



```
  }

}
```

(Esta zona é para rascunho)

Grupo 2 [8 valores]

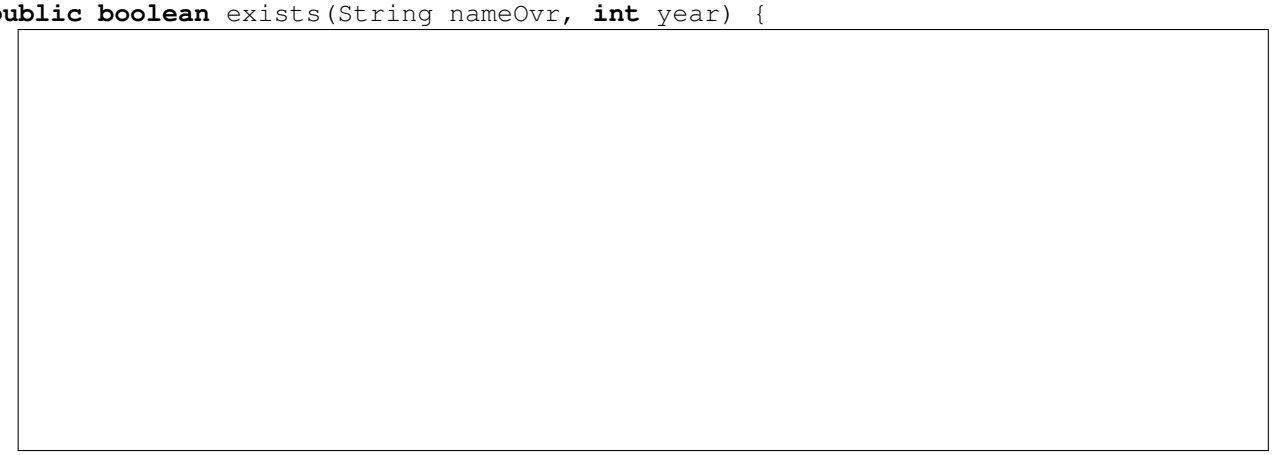
Defina as constantes da classe `Artist` e o corpo dos métodos `addOeuvre`, `bestYear`, `exists`, `sell` e `valueSales`. Note que o número de obras dum artista não é limitado. Pode acrescentar os métodos auxiliares que achar convenientes.

```
public class Artist {
```

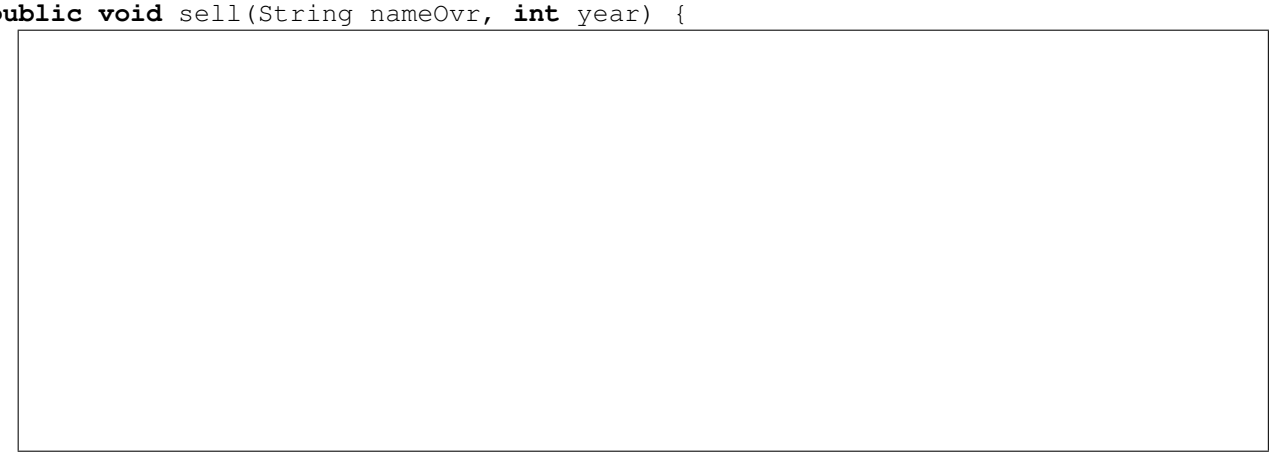
```
  private String name;
  private Oeuvre[] oeuvres;
  private int count;

  /* Cria um artista com o nome dado e uma coleccao vazia de obras da sua autoria,
   * e coloca o contador de obras a zero.
   *
   * @pre name != null
   */
  public Artist(String name) { ... }

  public String getName() { ... }

  // @return true, se o artista ja' produziu alguma obra;
  //         false, no caso contrario.
  public boolean hasOeuvres() { ... }

  /* Insere uma nova obra (no estado por vender) na coleccao de obras do artista,
   * se nao existir uma obra com o nome e o ano especificados.
   *
   * A operacao deve manter a coleccao de obras do artista por ordem crescente
   * de ano; em caso de empate no ano, por ordem alfabetica de nome da obra.
   *
   * @return OEUVRE_EXISTS, se ja' existir uma obra com o nome e o ano dados;
   *         SUCCESS, no caso contrario.
   *
   * @pre nameOvr != null && price > 0
   */
  public int addOeuvre(String nameOvr, int year, double price) {
```

```
  }
```

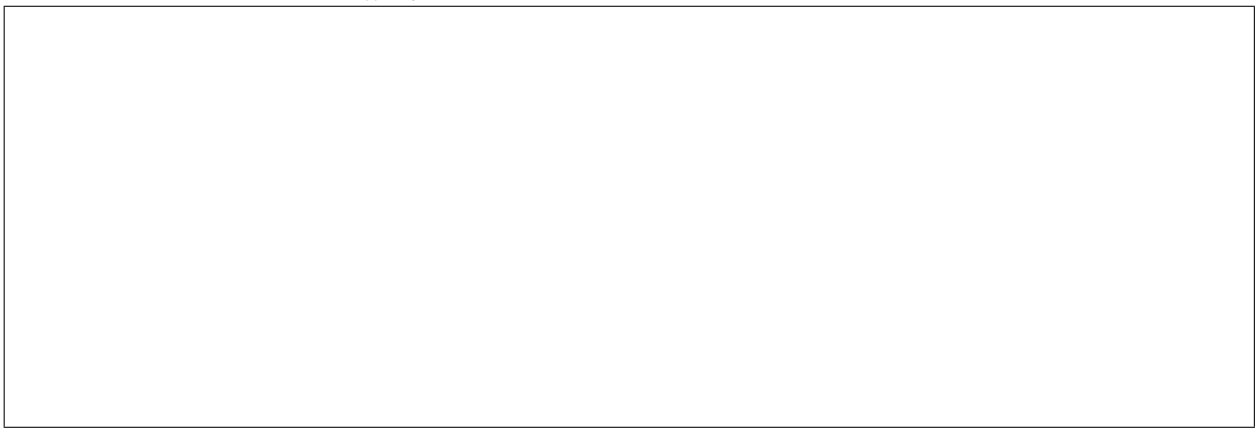
```
// @return true, se existir na coleccao uma obra com o nome e o ano dados;
//         false, no caso contrario.
// @pre nameOvr != null
public boolean exists(String nameOvr, int year) {
```



```
}

/* Coloca o estado de uma obra, identificada pelo seu nome e ano, como vendida,
 * assumindo-se que o artista tem uma obra com o nome e o ano dados.
 *
 * @pre nameOvr != null && exists(nameOvr, year)
 */
public void sell(String nameOvr, int year) {
```



```
}

// @return a soma dos precos de todas as obras vendidas.
public double valueSales() {
```



```
}
```

```
/* @return o ano em que o autor produziu mais obras;
 *         caso haja varios anos com o numero maximo de obras produzidas,
 *         retorna o mais recente desses anos.
 *
 * @pre hasOeuvres()
 */
public int bestYear() {
```

```
}
```

```
// Metodos auxiliares que entenda conveniente acrescentar:
```

```
}
```

(Esta página é para rascunho)

Grupo 3 [4 valores]

Implemente o corpo do construtor da classe `IteratorOeuvresArtistYear`, bem como os métodos `hasNext` e `next`. Pretende-se dar acesso a todas as obras de um dado artista num ano específico. Pode acrescentar os métodos auxiliares que achar convenientes.

```
public class IteratorOeuvresArtistYear {

  private String name;
  private int year, currentOeuvre;
  private Oeuvre[] works;

  /* Dados um vector de objectos do tipo Oeuvre (totalmente ocupado, possivelmente
   * desordenado e possivelmente com obras de varios artistas), o nome dum artista
   * e um ano, constroi um iterador que da' acesso 'as obras do artista com o nome
   * dado que tenham sido concebidas no ano referido (se existir alguma).
   *
   * @pre works != null && name != null
   */
  public IteratorOeuvresArtistYear(Oeuvre[] works, String name, int year) {
```

```
  }

  // @return true, se ainda ha' alguma obra a devolver; false, no caso contrario.
  public boolean hasNext() {
```

```
  }

  // @pre hasNext()
  public Oeuvre next() {
```

```
  }
```

```
// Metodos auxiliares que entenda conveniente acrescentar:
```

```
}
```

(Esta zona é para rascunho)

Grupo 4 [3 valores]

Implemente os métodos `loadVec` e `saveVec`, respectivamente, para carregar um vector de objectos `Oeuvre`, com obras de um único artista, a partir dum ficheiro e para gravar num ficheiro a informação relevante das obras de um artista. A informação no ficheiro deve conter: na primeira linha, o nome do artista; na segunda linha, o número de obras; seguem-se as obras, cada uma em 4 linhas, com o respectivo nome, ano, preço (com duas casas decimais) e indicação se está vendida.

Um exemplo do formato e do conteúdo pretendido para o ficheiro é:

```
Joana Vasconcelos
2
Dorothy
2007
99999.99
true
Coeur de Paris
2018
500000.00
true
```

```
// Cria um vector de objectos do tipo Oeuvre e preenche-o completamente com a
// informacao no ficheiro de texto com o nome dado.
// Para simplificar, assuma que ha' pelo menos uma obra do artista no ficheiro.
// @pre fileName != null
public Oeuvre[] loadVec(String fileName) throws FileNotFoundException {
```

```
}
```

```
// Escreve no ficheiro de texto com o nome dado o conteudo do vector de objectos do
// tipo Oeuvre, que esta' completamente preenchido com obras de um unico artista.
// @pre works != null && works.length > 0 && fileName != null
public void saveVec(Oeuvre[] works, String fileName) throws FileNotFoundException {
```

```
}
```

```
// Metodos auxiliares que entenda conveniente acrescentar:
```

```
}
```

(Esta zona é para rascunho)

Grupo 5 [3.5 valores]

Questão 1. Considere a seguinte definição recursiva da função de Ackermann, $\mathcal{A}(m,n)$, onde m e n são números inteiros não negativos:

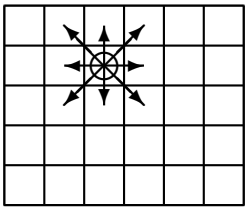
$$\mathcal{A}(m,n) = \begin{cases} n+1 & \text{se } m = 0 \\ \mathcal{A}(m-1,1) & \text{se } m \geq 1 \text{ e } n = 0 \\ \mathcal{A}(m-1,\mathcal{A}(m,n-1)) & \text{se } m \geq 1 \text{ e } n \geq 1 \end{cases}$$

Implemente em Java um método recursivo, chamado `ackermann`, que recebe dois inteiros não negativos, m e n, e retorna o valor de $\mathcal{A}(m,n)$ (que é sempre um número inteiro).

```
/**
 * Calcula a funcao de Ackermann
 * @return o valor da funcao de Ackermann de (m,n)
 * @pre m >= 0 && n >= 0
 */
public static int ackermann(int m, int n) {









}
```

Questão 2. Pretende-se determinar se numa matriz de inteiros positivos alguma célula do interior tem mais do que um vizinho par.

A *fronteira* da matriz é constituída pela primeira linha, pela última linha, pela primeira coluna e pela última coluna. O *interior* da matriz é formado pelas células que não se encontram na fronteira. Repare que qualquer célula do interior tem 8 células vizinhas, como se ilustra na figura da esquerda.



1	3	1	3	1	**6**
1	9	3	**4**	(5)	5
7	1	5	7	9	**4**
6	1	1	9	3	5
3	7	5	3	2	8

Na figura da direita, a célula assinalada com uma circunferência tem 3 vizinhos pares (escritos a *bold*). Portanto, na matriz do exemplo, há alguma célula do interior que tem mais do que um vizinho par.

Implemente o método `existsEvenNeighbours`, que determina se, numa matriz de inteiros positivos, alguma célula do interior da matriz tem mais do que um vizinho par. O método não tem argumentos e pressupõe que existe na classe uma variável de instância chamada `matrix` com o tipo `int[][]`. Pode acrescentar os métodos auxiliares que considerar convenientes.

```
/**
 * @return true, se alguma celula do interior da matriz matrix tem pelo
 *         menos dois vizinhos pares; false, no caso contrario.
 * @pre matrix != null && matrix.length > 0 && matrix[0].length > 0
 */
private boolean existsEvenNeighbours() {
    int rows = matrix.length;
    int cols = matrix[0].length;
```

Coloque na caixa abaixo os métodos auxiliares que entenda conveniente acrescentar:

(Esta zona é para rascunho)

(Esta página é para rascunho)