

2º Teste
Introdução à Programação
13/12/2017
Duração 2H

Instruções importantes:

- Responda a cada pergunta no espaço atribuído para o efeito.
- Verifique que **todas** as folhas estão identificadas com o seu nome e número.
- Antes de começar a resolver, leia o enunciado do **princípio até ao fim**.
- Não é permitido consultar quaisquer elementos para além deste enunciado.
- Qualquer tentativa de fraude comprovada acarretará a reprovação na disciplina.

Nome:

Número de aluno:

Número de páginas entregues:

Considere o seguinte enunciado do problema, o qual será usado nos grupos I, II e III:

Objectivo:

Gerir os apartamentos para venda na empresa X.

Descrição:

A empresa X tem vários vendedores (no máximo 40) e apartamentos para venda.

Cada vendedor tem um nome único que o identifica, um local de residência (nome de uma localidade) e gere até 30 apartamentos da empresa na localidade.

Cada apartamento para venda é identificado por um número único e caracterizado pelo local (nome de uma localidade), tipologia ("T1", etc...). É sempre possível adicionar novos apartamentos à empresa. Caso não exista nenhum vendedor na área, ou os que existem já tenham 30 apartamentos atribuídos, o apartamento fica "sem vendedor". Uma vez vendido, o apartamento deixa de existir na empresa.

Para além disso, é sempre possível consultar os vendedores da empresa, assim como os apartamentos que gere um dado vendedor, ou que existam numa dada localidade.

Interacção com o utilizador:

Interpretador de comandos, onde é permitido adicionar um vendedor ("AV"), adicionar um apartamento("AA"), registar a venda dum apartamento ("RVA"), listar todos os vendedores ("LV"), listar todos os apartamentos geridos por um dado vendedor ("LAV"), listar todos os apartamentos num dado local ("LAL") e sair da aplicação ("S").

Para a resolução do problema, foram definidas 6 classes: Apartamento, Vendedor, ColApartamentos, ColVendedores, Empresa e Main.

As classes Apartamento, ColVendedores e Empresa estão definidas neste enunciado, não sendo necessário implementá-las (assuma que já estão implementadas). As classes Vendedor e ColApartamentos serão implementadas neste teste, enquanto que a classe Main já está parcialmente implementada, tendo apenas que a complementar.

-----Classe Apartamento-----

```
public class Apartamento {
/* Representa um apartamento para venda */
...

/* Verifica se é um tipologia válida
 * @param tipo tipo de apartamento
 * @pre tipo != null
 */
public static boolean eTipologia(String tipo) { ... }

/* Cria e inicializa um apartamento
 * @param id número de identificação do apartamento (único)
 * @param cidade cidade onde se localiza
 * @param tipo tipo de apartamento
 * @pre cidade != null && tipo !=null && Apartamento.eTipologia(tipo))
 */
public Apartamento(int id, String cidade, String tipo) { ... }
```

```

/** Consulta da cidade do apartamento
 * @return – cidade
 */
public String cidade() { ... }

/** Consulta o tipo do apartamento
 * @return – tipo
 */
public String tipo() { ... }

/** Consulta o identificador do apartamento
 * @return – identificador
 */
public int identificador() { ... }

/** Consulta se o apartamento tem vendedor atribuído
 * @return – true se tiver vendedor atribuído, false caso contrário
 */
public boolean temVendedor() { ... }

/** Atribui o vendedor dado ao apartamento
 * @param vende - vendedor
 * @pre !temVendedor() && vende != null
 */
public void atribuiVendedor(Vendedor vende) { ... }
}

```

-----Classe ColVendedores -----

```

public class ColVendedores {
/** Representa uma colecção de vendedores */
...
/** Cria uma colecção de zero vendedores, onde a lotação prevista é a indicada.
 * @param previstaLotacao número previsto de vendedores, à partida
 * @pre lotacaoPrevista >0
 */
public ColVendedores(int lotacaoPrevista) {...}

/** Consulta do número de vendedores na colecção
 * @return – número de vendedores
 */
public int numeroVendedores() {...}

/** Indica se existe um vendedor com o nome dado
 * @param – nome - nome
 * @pre nome != null
 * @return – true, se existe e false caso contrário
 */
public boolean existe(String nome) { ...}

/** Adiciona o vendedor dado à colecção, caso não exista um vendedor com o mesmo nome na colecção. Se
necessário, a colecção deve aumentar a sua capacidade.

```

```

* @param – elem vendedor a adicionar
* @pre elem!=null
* @return – true, se adicionar, false caso contrário.
*/
public boolean adiciona(Vendedor elem) {...}

/** Devolve o vendedor com o nome dado, caso exista. Se não existir devolve null
* @param – nome - nome
* @pre nome!=null
* @return – o vendedor, se existir; null caso contrário
*/
public Vendedor vendedor(String nome) {...}

/** Elimina o vendedor com o nome indicado da colecção.
* @param – nome - nome
* @pre nome!=null && existe(nome)
*/
public void remove(String nome) {... }
}

```

-----Classe Empresa -----

```

public class Empresa{
/* Representa a empresa de venda de apartamentos */
...
/* Cria uma empresa, onde a lotação prevista de vendedores é a indicada.
* @param numPrevistoVendedores número previsto de vendedores, à partida
* @pre numPrevistoVendedores >0
*/
public Empresa(int numPrevistoVendedores){...}

/** Indica se existe um vendedor com o nome dado
* @param – nome - nome
* @pre nome != null
* @return – true, se existe e false caso contrário
*/
public boolean eVendedor(String nome) {...}

/** Adiciona o vendedor dado à empresa.
* @param –nome nome do vendedor a adicionar
* @param – localResidencia nome do local de residência do vendedor a adicionar
* @pre nome!=null && localResidencia != null && !eVendedor(nome)
*/
public void adicVendedor(String nome, String localResidencia){...}

/** Remove o vendedor com o nome dado da empresa.
* @param –nome nome do vendedor a remover
* @pre nome!=null && eVendedor(nome)
*/
public void remVendedor(String nome){...}
}

```

```

/** Indica se existe um apartamento com o identificador dado
 * @param – id - identificador do apartamento
 * @return – true, se existe e false caso contrário
 */
public boolean existeApartamento (int id){...}

/** Adiciona o apartamento. Gera um identificador único para o apartamento. O apartamento fica para um
 * vendedor com o mesmo local de residência, caso seja possível ou fica como “sem vendedor”
 * @param - local nome do local do apartamento
 * @param –tipo tipo do apartamento
 * @pre local!=null &&tipo != null
 * @return retorna o identificador do apartamento
 */
public int adicApartamento(String local, String tipo){...}

/** Regista a venda do apartamento, removendo o apartamento com o identificador dado da empresa.
 * @param –id identificador do apartamento a vender
 * @pre existeApartamento(id)
 */
public void vendaApartamento(int id){...}

/** Inicializa o iterador dos vendedores da empresa.
 */
public void iteraVendedores(){...}

/** Indica se há mais vendedores para iterar.
 * @return true, se existir; false, caso contrário
 */
public boolean temSeguinteVendedor(){...}

/**Consulta do próximo vendedor na iteração.
 * @pre temSeguinteVendedor()
 * @return o vendedor
 */
public Vendedor seguinteVendedor(){...}

/** Inicializa o iterador dos apartamentos da empresa.
 */
public void iteraApartamentos(){...}

/** Indica se há mais apartamentos para iterar.
 * @return true, se existir; false, caso contrário
 */
public boolean temSeguinteApartamento(){...}

/**Consulta do próximo apartamento na iteração.
 * @pre temSeguinteApartamento()
 * @return o apartamento
 */
public Apartamento seguinteApartamento(){...}
}

```

Nome:

Número:

Grupo I

Este exercício visa a construção de uma classe *ColApartamentos* que representa uma colecção - à partida não limitada - de apartamentos para venda.

Implemente a classe **(7,5 valores)**:

```
public class ColApartamentos {  
    /* Representa uma colecção de apartamentos */  
    /* Constantes e Variáveis de instância */
```

```
    /* Cria uma colecção de zero apartamentos, onde a lotação prevista é a indicada.
```

```
    * @param previstaLotacao número previsto de apartamentos, à partida
```

```
    * @pre lotacaoPrevista >0
```

```
    */
```

```
    public ColApartamentos(int lotacaoPrevista) {
```

```
    }
```

```
    /** Consulta do número de apartamentos na colecção
```

```
    * @return – número de apartamentos
```

```
    */
```

```
    public int numeroApartamentos() {
```

```
    }
```

```
/** Indica se existe um apartamento com o identificador dado
 * @param – id - identificador
 * @return – true, se existe e false caso contrário
 */
```

```
public boolean existe(int id)
```

```
}
```

```
/** Adiciona o apartamento dado à colecção, caso não exista um apartamento com o mesmo identificador na
colecção. Se necessário, a colecção deve aumentar a sua capacidade.
```

```
 * @param – elem apartamento a adicionar
 * @pre elem!=null
 * @return – true, se adicionar, false caso contrário.
 */
```

```
public boolean adiciona(Apartamento elem) {
```

```
}
```

```
/** Devolve o apartamento com o identificador dado, caso exista. Se não existir devolve null
```

```
 * @param – id - identificador
 * @return – o apartamento, se existir; null caso contrário
 */
```

```
public Apartamento apartamento(int id) {
```

```
}
```

```
/** Elimina o apartamento com o identificador indicado da colecção.
```

```
* @param – id identificador do apartamento
```

```
* @pre existe(id)
```

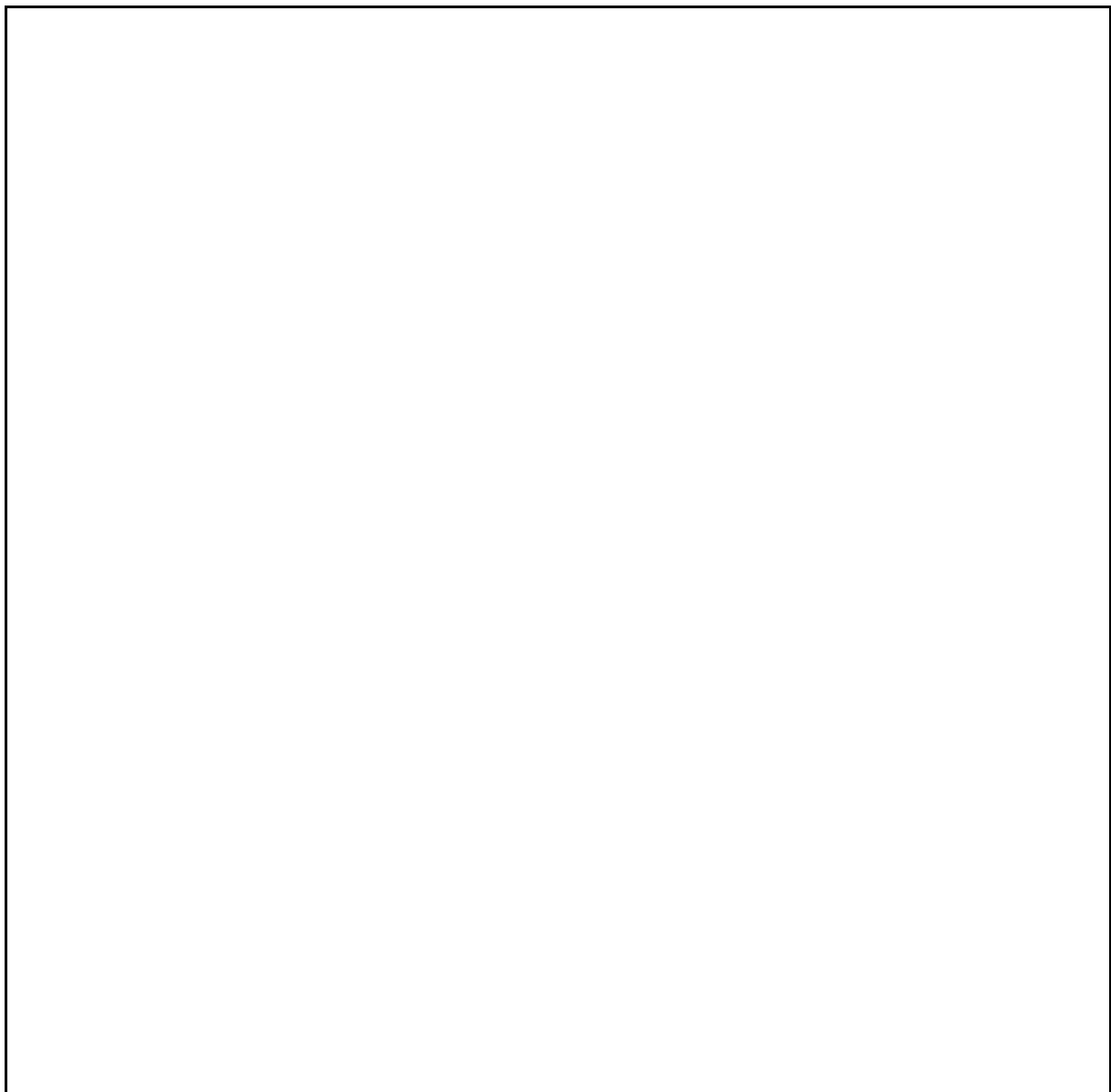
```
*/
```

```
public void remove(int id) {
```



```
}
```

```
/* Métodos privados */
```



```
/* Iterador de identificadores de Apartamentos: */  
/**Inicializa o iterador de identificadores de apartamentos.  
*/
```

```
public void inicIterador() {
```

```
}
```

```
/** Indica se há mais apartamentos para iterar
```

```
* @return – true, se houver, false caso contrário
```

```
*/
```

```
public boolean temSeguinte() {
```

```
}
```

```
/**Consulta o identificador do seguinte apartamento
```

```
* @pre –temSeguinte()
```

```
* @return – id identificador do seguinte apartamento
```

```
*/
```

```
public int seguinte() {
```

```
}
```

```
}
```

Nome:

Número:

Grupo II

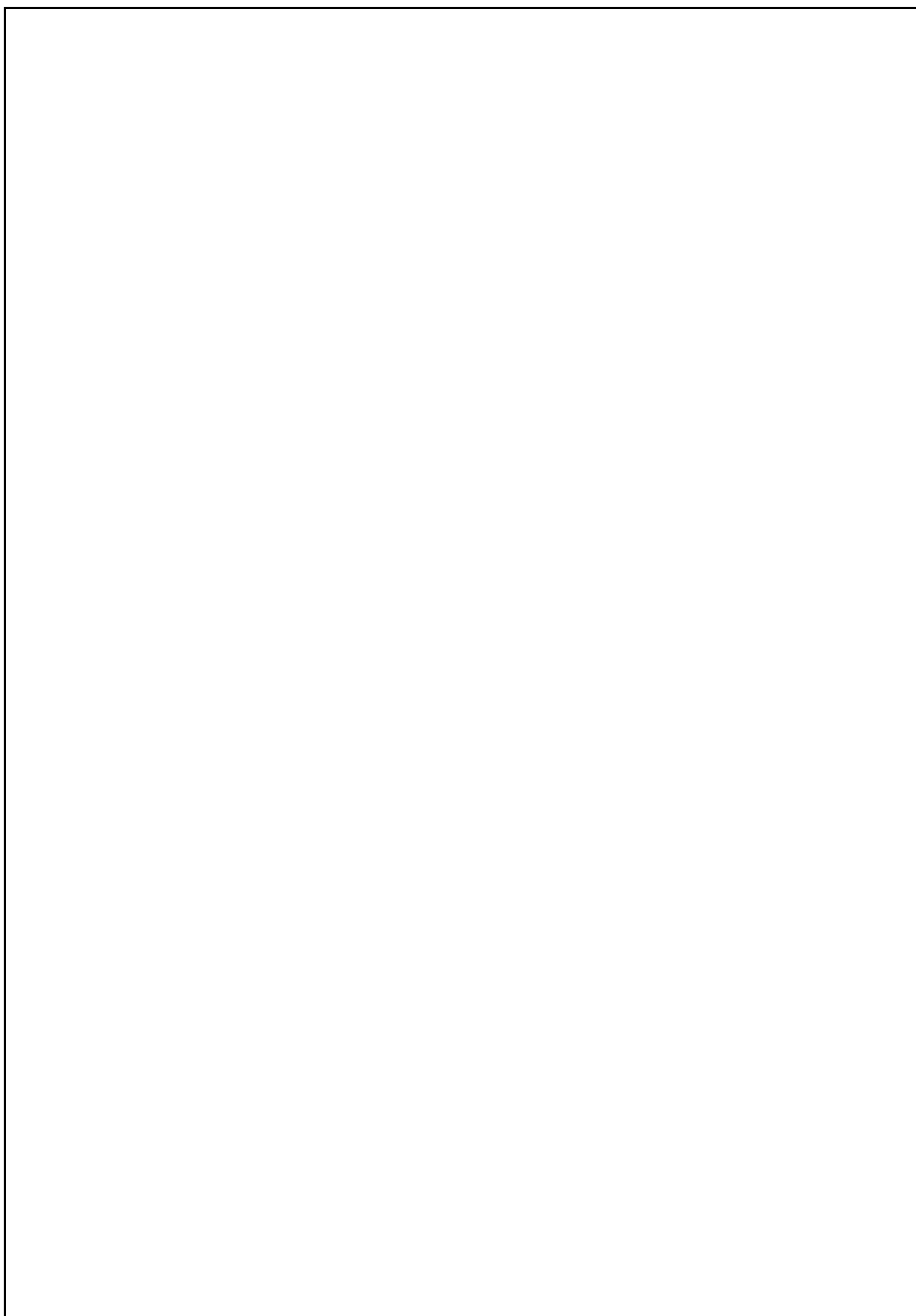
Este exercício visa a construção de uma classe *Vendedor* que representa um vendedor e os apartamentos a seu cargo para venda (no máximo 30). Considerando que esta classe tem como constantes e variáveis de instância:

```
private static final int CAPACIDADE = 30;  
private String nome;  
private String localidade;  
private ColApartamentos apartamentos;
```

e as seguintes funcionalidades:

1. Consulta o nome do vendedor;
2. Consulta a localidade do vendedor;
3. Consulta o número de apartamentos a cargo do vendedor;
4. Acrescenta um apartamento dado à colecção gerida pelo vendedor, se o apartamento está no local de residência do vendedor e se este ainda tem capacidade para gerir mais apartamentos (recorde que apenas pode gerir até 30 apartamentos);
5. Vende um apartamento (que tem de existir), dado o seu identificador, removendo-o da colecção a seu cargo;
6. Consulta o apartamento com o identificador dado;
7. Permite a listagem de todos os identificadores de apartamentos;
8. Quando cria um vendedor é indicado o nome e localidade. A colecção fica vazia, ainda que a capacidade máxima será de 30.

Implemente apenas as funcionalidades 2, 3, 4, 5 e 8 da classe *Vendedor*. Não esqueça colocar o comentário, protótipo e corpo dos métodos públicos pedidos (6,5 valores).



Nome:

Número:

Grupo III.

Implemente o seguinte método auxiliar que existe na classe Main, onde é implementado o interpretador de comandos, o qual usa a classe *Empresa* (**3 valores**):

```
public class Main {
```

```
/* Escreve num ficheiro de texto, os nomes dos vendedores da empresa dada,
```

```
 * que tem apartamentos para venda na localidade dada.
```

```
 * Em cada linha é escrito o nome de um vendedor.
```

```
 * @param emp - empresa criada no método main
```

```
 * @param localidade - nome do local/cidade a procurar
```

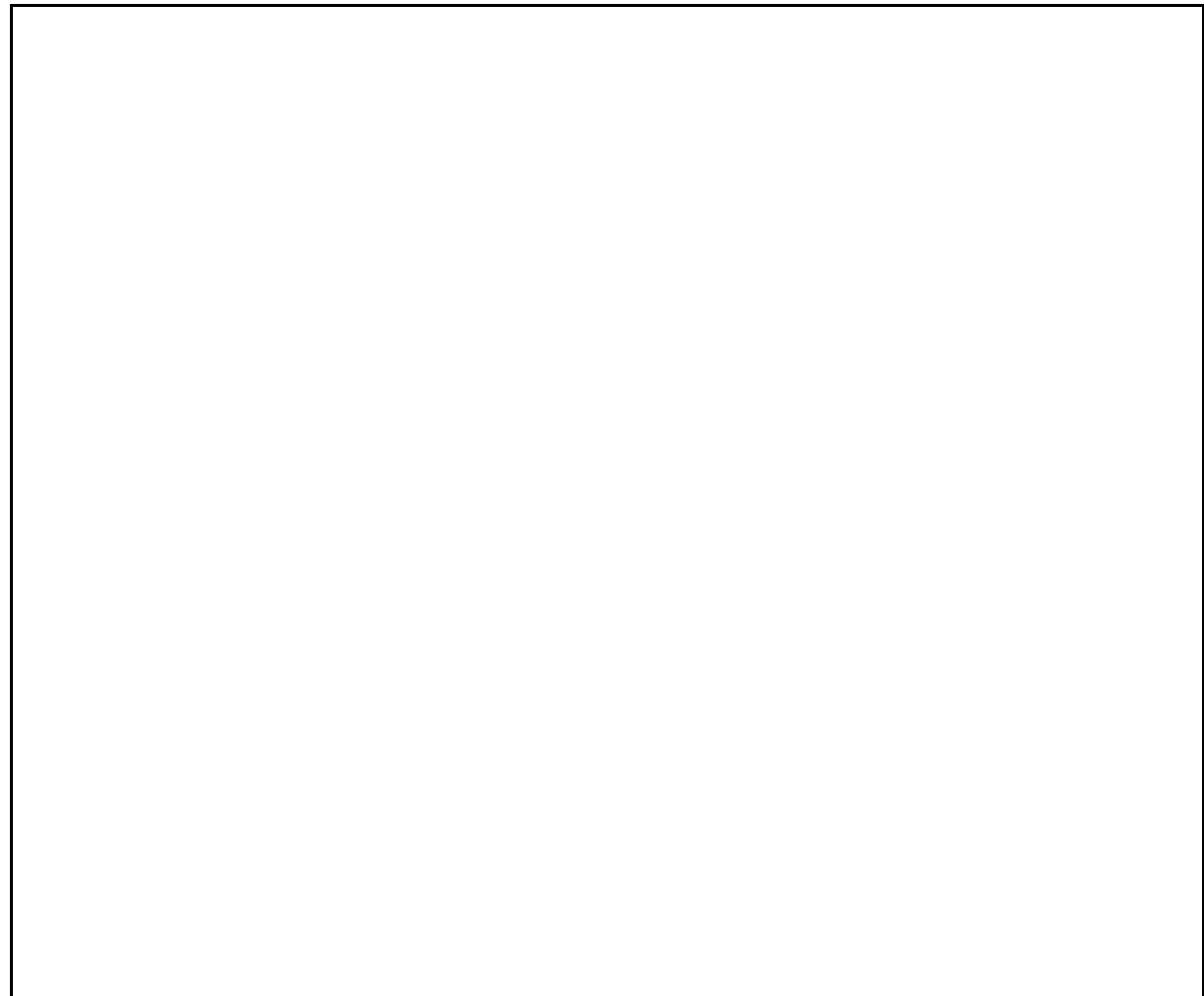
```
 * @param nomeFich - nome do ficheiro onde se pretende escrever a informação pedida
```

```
 * @pre: emp!=null && localidade != null && nomeFich != null
```

```
 */
```

```
private static void vendedoresNaLocalidade (Empresa emp, String localidade, String nomeFich)
```

```
    throws FileNotFoundException{
```



```
}
```

Nome:

Número:

Grupo IV.

Uma sequência de inteiros de tamanho pelo menos 2 diz-se estritamente crescente se cada elemento é maior que o anterior.

Considere uma classe com uma variável de instância `vec` que contém um vector de inteiros de comprimento pelo menos 2, totalmente preenchido. O conteúdo do vector pode ser visto como uma sequência, podendo ter várias subsequências (de)crescentes.

Implemente o método `compSeqCresc` que devolve o comprimento da maior sequência crescente no vector.

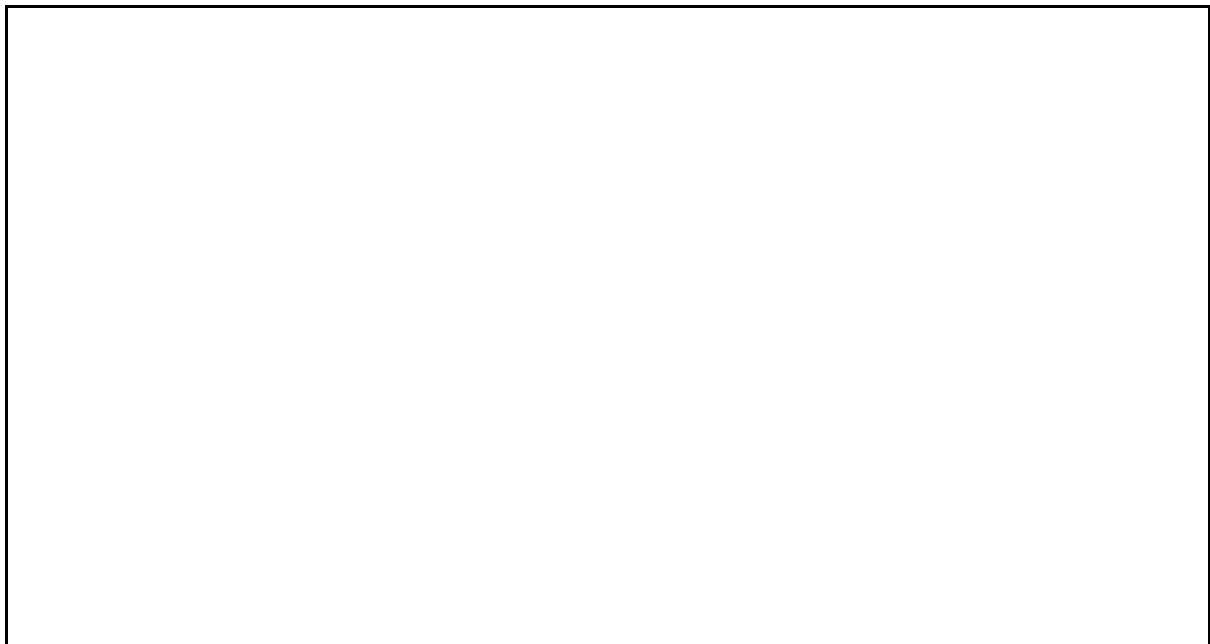
Exemplos:

- Se `vec = [4,6,7,3,4,1,8]` então `compSeqCresc() = 3`
- Se `vec = [4,6,7,3,1,4,8]` então `compSeqCresc() = 3`
- Se `vec = [5,3,-4,-3,-3,-8,2,1]` então `compSeqCresc() = 2`
- Se `vec = [4,3,2,1]` então `compSeqCresc() = 0`

Note que este método nunca devolve o valor 1.

Implemente o método (**3 valores**):

```
public int compSeqCresc () {
```



```
}
```