Exame da época normal
Introdução à Programação
12/01/2018

Duração 2H30M

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.
- Tem de entregar **todas** as folhas, incluíndo respostas em branco.
- 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:

Considere o seguinte enunciado do problema, o qual será usado nos grupos I, II e III:
Objectivo: Implementar uma plataforma de partilha de viaturas para viagens.
Descrição: A plataforma visa permitir que os seus utilizadores registem viagens para as quais estejam disponíveis para oferecer como boleia, e para pesquisar ofertas de boleias que estejam interessados em aproveitar. De cada utilizador apenas temos um *nickname*. Cada viagem é caracterizada pelos seguintes atributos: um número único (int), um local de partida (String), um local de destino (String), uma data e hora de partida (DataHora), um tempo estimado para a viagem em minutos (int), um veículo, o nome do condutor e os nomes dos restantes passageiros. Cada veículo está identificado pela sua matricula, e tem como característica relevante para este problema a capacidade de passageiros (incluindo o condutor). O sistema tem um interpretador de comandos, nomeadamente para registar novas viagens, associar um novo passageiro às viagens já existentes (desde que a viagem ainda possua capacidade) e pesquisar viagens para aproveitar boleia.

A solução para este problema envolve 8 classes: GestorDeBoleias, Viagem, Viatura, ColViagens, ColViaturas, ColPassageiros, DataHora e Main. As classes GestorDeBoleias, Viatura, ColViagens, ColViaturas e DataHora já estão definidas neste enunciado (total ou parcialmente), não sendo necessário implementá-las. As classes **ColPassageiros**, **Viagem** e **Main** serão implementadas total ou parcialmente nesta prova.

----------Classe DataHora ----------

```
/**
 * Representa uma dia e hora, com precisão até aos minutos.
 */
public class DataHora {
   /**
    * Indica se os parametros dados podem formar uma data válida
    */
   public static boolean eValidaData(int ano, int mes, int dia) {...}
   /**
    * Indica se os parametros dados podem formar uma Hora válida
    */
   public static boolean eValidaHora(int hora, int minuto) {...}

/* @pre DataHora.eValidaData(ano,mes,dia) && DataHora.eValidaHora(hora,minuto) */

   public DataHora(int ano, int mes, int dia, int hora, int minuto) {...}
   public int ano() {...}
   public int mes() {...}
   public int dia() {...}
   public int hora() {...}
   public int minuto() {...}
   /**
    * @return true, se 'outra' representar o mesmo dia, mês e ano
    * (ainda que a hora seja diferente);
    * false caso contrário.
    */
   public boolean igualData(DataHora outra) {...}
}
```

----------Classe Viatura ----------

```
/**
 * Representa uma viatura.
 */
public class Viatura {
	/** @pre matricula != null && capacidade > 1 */
	public Viatura(String matricula, int capacidade) {...}
	...
	public String matricula() {...}
	...
	public int capacidade(){...}
}
```

----------Classe ColViaturas ----------

```
/**
 * Representa uma coleção de viaturas. Inserção, Remoção e Pesquisa
 */
public class ColViaturas {
	… // Não vai ser usada neste enunciado
}
```

----------Classe ColViagens ----------

```
/**
 * Representa uma coleção de viagens. Inserção, Remoção e Pesquisa
 */
public class ColViagens {
	… // Não vai ser usada neste enunciado
}
```

----------Classe GestorDeBoleias ----------

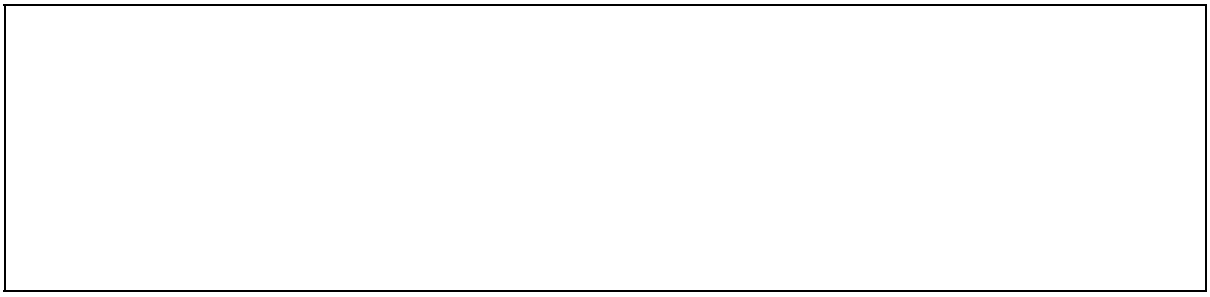
```
/**
 * Representa o gestor de boleias. Inclui uma coleção de viagens e de viaturas
 */
public class GestorDeBoleias {
	…

	public int numeroViagens() {...}
	…

	/* Iterador de viagens registadas */

	public void iteradorViagens() {...}

	public boolean temSeguinteViagem() {...}

	public Viagem seguinteViagem() {...}

}
```

Nome:
Número:

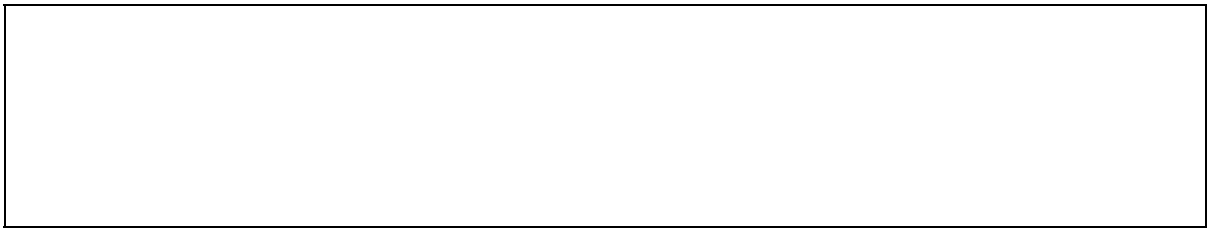
Grupo I (6 valores)

Este exercício visa a construção de uma classe *ColPassageiros* que representa uma colecção de *nicknames* de passageiros *(String)* com capacidade limitada e iterador. Esta classe é usada na classe *Viagem*, como se vê no Grupo II.

```
/**
 * Colecção de Passageiros com capacidade limitada, especificada no construtor.
 */
public class ColPassageiros {
/* Variáveis de instância */
```

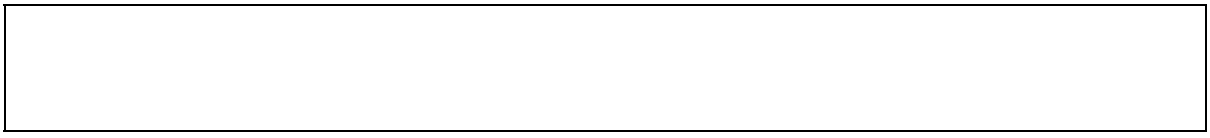


```
/**
 * Inicializa o objecto de modo a representar uma colecção com limite definido
 * para o número de elementos, especificado pelo argumento capacidade
 * @pre capacidade > 0
 */
public ColPassageiros(int capacidade)
```



```
/**
 * @return capacidade da colecção, i.e., número de elementos que pode conter
 */
public int capacidade() {
```

```
}
/**
 * @return número de elementos na colecção
 */
public int numeroElementos() {
```

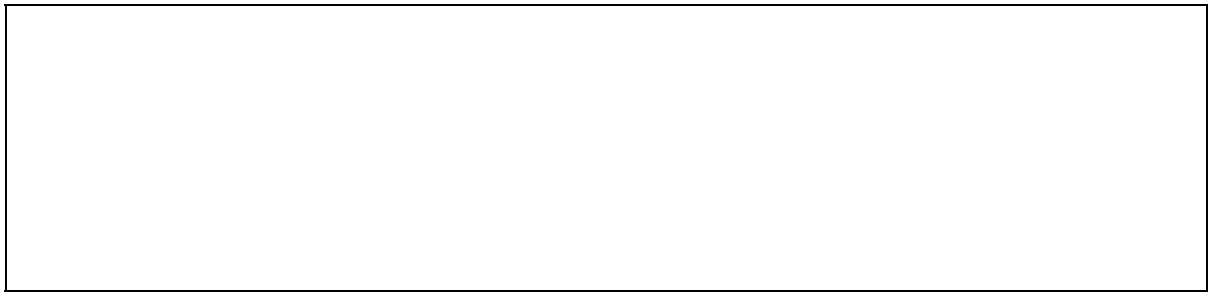


```
}
```

```
/**
 * @return true se colecção está cheia; false caso contrário
 */
public boolean capacidadeEsgotada()
```

```
}
/**
 * @param nickname cuja inclusão na colecção está a ser consultada
 * @return true caso str já exista na colecção; false caso contrário
 * @pre nickname != null
 */
public boolean existe(String nickname) {
```

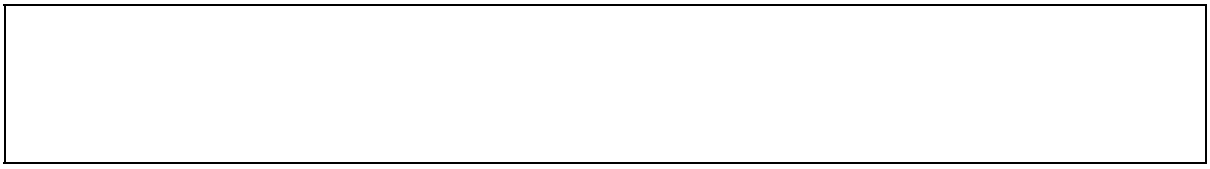
```
}
/**
 * Acrescenta um novo passageiro, caso haja capacidade
 * @param nickname a inserir
 * @return true, caso a inserção tenha sucesso; false caso contrário
 * @pre
 *
 */
public boolean acrescenta(String nickname) {
```



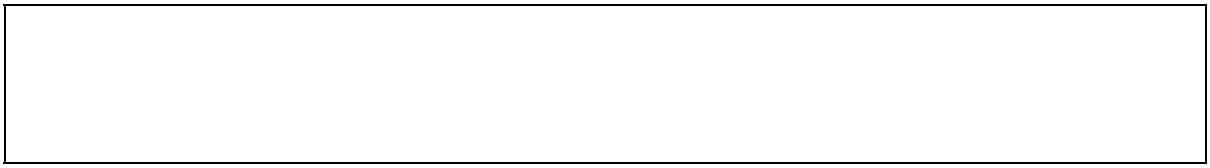
```
}
```

```
 /**
  * Remove o elemento com o nickname especificado, preservando a ordem dos
  * restantes elementos
  * @pre
  *
  * @param nickname do passageiro a remover
  */
public void remove(String nickname) {

}

/* Métodos privados */
```

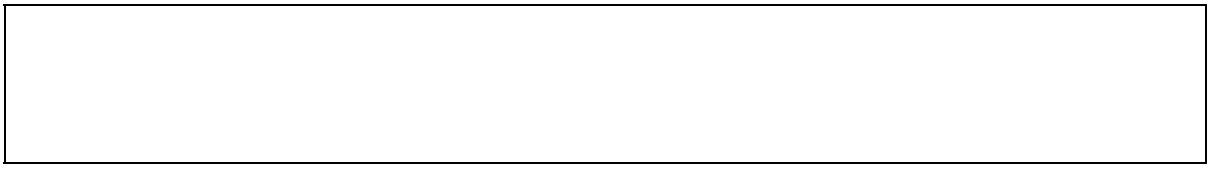
```
/**
 * Inicia a parte iterador deste objecto colecção
 */
public void inicIterador() {
```



```
}

/**
* @return true caso haja mais elementos a iterar; false caso contrário
*/
public boolean temSeguinte() {
```



```
}

/**
 * @return próximo elemento (passageiro) a iterar
 * @pre temSeguinte()
 */
public String seguinte() {
```



```
}
```

Nome:
Número:

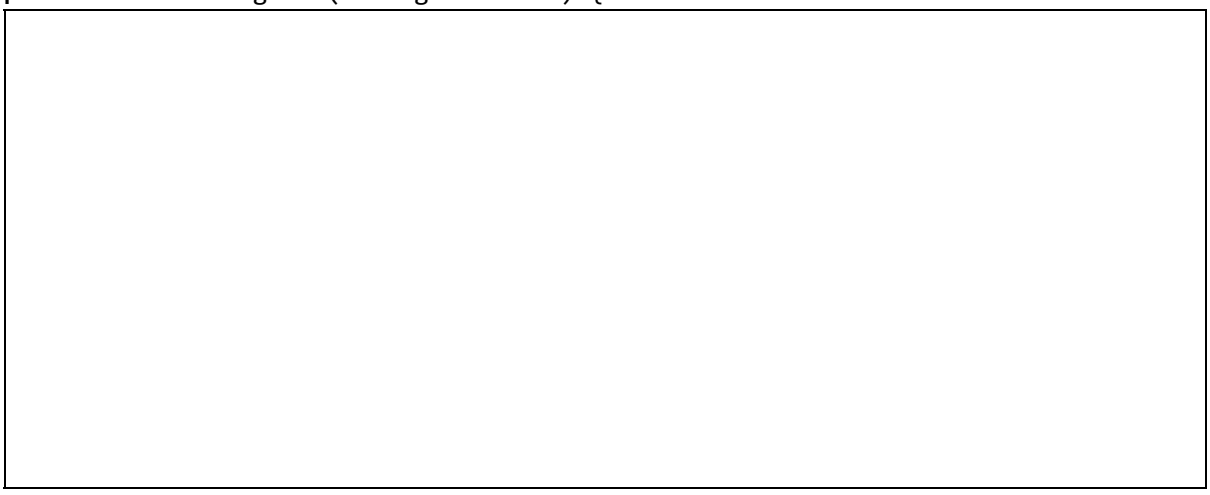
Grupo II (6 valores)

Nesta secção, pede-se parte da implementação da classe Viagem. Implemente unicamente os métodos que tenham caixas para o efeito.

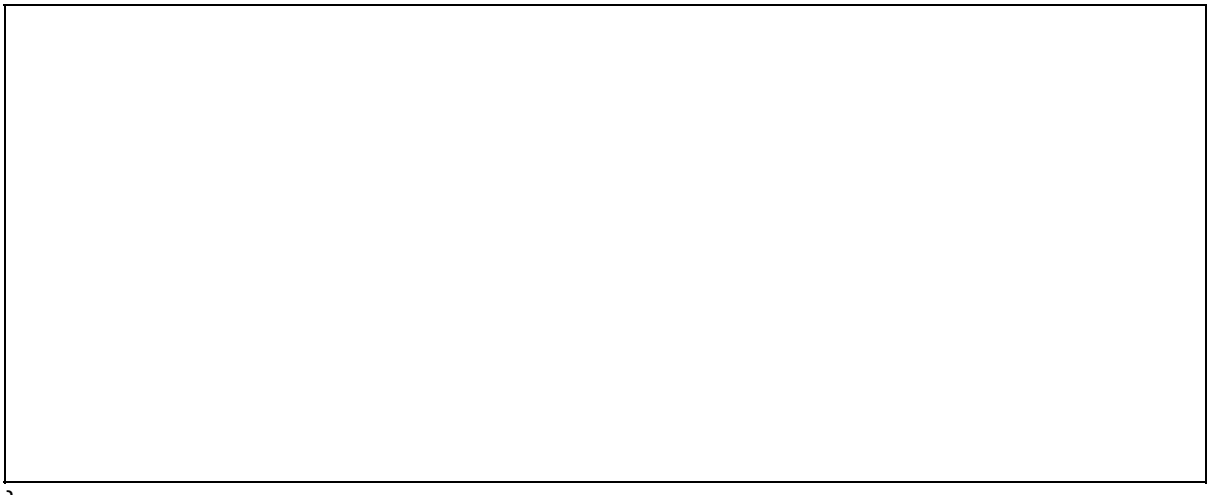
```
/**
 * Representação de uma Viagem, identificada por um valor inteiro
 */
public class Viagem {
/* Variáveis de instância */

private int identificador;
private String localOrigem;
private String localDestino;
private int duracaoPrevistaMinutos;
private Viatura carro;
private DataHora dataViagem;
private String condutor;
private ColPassageiros passageiros; // Não inclui condutor

/**
 * Inicializa a classe
 * @pre identificador >= 0 && condutor != null && partida != null
 *      && destino != null && data != null && carro!=null
 */
public Viagem(int identificador, String condutor, String partida, String destino,
              DataHora data, int duracao, Viatura carro) {...}

/**
 * @return identificador desta viagem
 */
public int identificador() {...}

/** @return nickname do condutor desta viagem  */
public String condutor() {...}

/** @return local de partida da viagem (e.g., Almada, Lisboa, Loures)  */
public String localPartida() {...}

/** @return local destino da viagem (e.g., Mangualde, Viseu, Penafiel) */
public String localDestino() {...}

/** @return data de partida da viagem */
public DataHora horaPartida() {...}

/** @return duração estimada da viagem */
public int duracaoPrevista() {...}
```

```
/**
 * @return capacidade para passageiros, sem incluir o condutor
 */
public int capacidade() {
```

```
}

/**
 * Regista um novo passageiro com o nickname dado, caso ainda haja capacidade
 * para tal e esse passageiro não esteja já registado ou seja o condutor
 * @return true se esta operação teve sucesso; false caso contrário
 * @pre nickname != null
 */
public boolean regista(String nickname) {
```

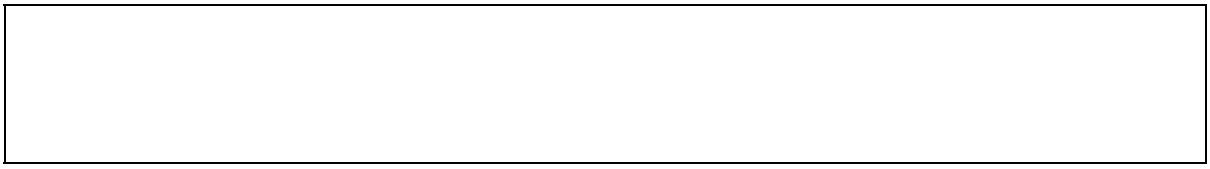


```
}
/**
 * Remove o passageiro com o nickname dado, caso seja um passageiro registado
 * @return true se esta operação teve sucesso; false caso contrário
 * @pre nickname != null
 */
public boolean remove(String nickname) {
```



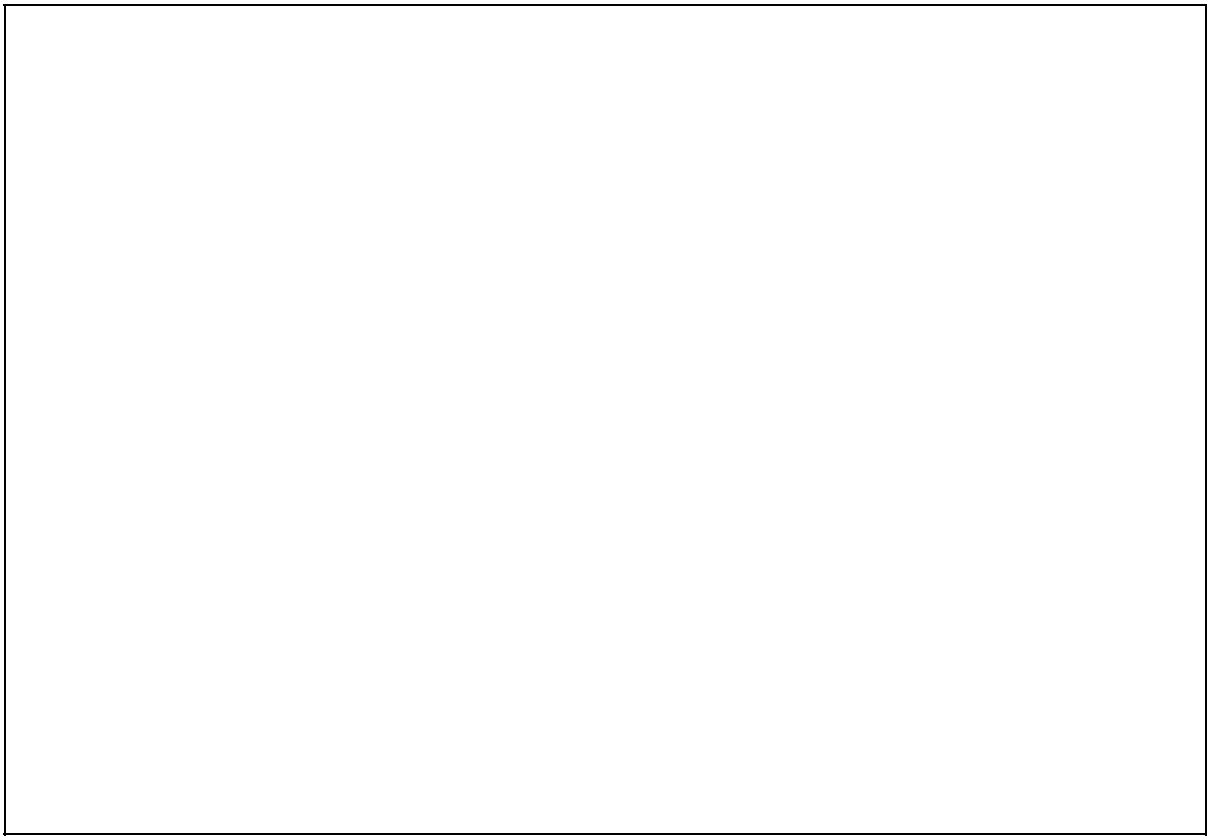
```
}
```

```
/**
 * @return número de passageiros já registados, incluindo o condutor
 */
public int numeroPassageiros() {
```



```
}

/**
 * @return true se a capacidade de passageiros está esgotada;
 * false caso contrário
 */
public boolean capacidadeEsgotada() {...}

/**
 * Indica se a viagem tem os mesmos locais de partida e destino dados, se é na
 * mesma data dada, e se a duração prevista é menor ou igual à duração dada
```

```
 * @pre
 *
 */

public boolean coincides(String origem, String destino, int ano, int mes, int dia,
                         int duracao) {
```



```
}
} // fecha a classe Viagem
```

Nome:
Número:

Grupo III (4 valores)

Um dos comandos da classe Main destina-se a escrever num ficheiro dado, todas as viagens que tem como origem e destino as localidades dadas. Em cada linha do ficheiro deve escrever a matrícula, os nicknames do condutor e passageiros e a data e hora da viagem. Assume-se na pré-condição que o gestor tem viagens para listar.

```
/** @pre gestor != null && origem != null && destino != null && nomeFich != null
*  &&
 */
private static void listaViagens(GestorDeBoleias gestor, String origem,
                   String destino, String nomeFich) throws FileNotFoundException {
```

```
}
```

Nome:
Número:

Grupo IV (4 valores)

Uma sequência de inteiros de tamanho pelo menos 2 diz-se estritamente decrescente/crescente se cada elemento é menor/maior que o anterior. Uma sequência (de)crescente também se diz monótona. Chama-se *ponto de inflexão* a um valor da sequência que tem imediatamente antes um valor menor (respectivamente maior) e imediatamente depois um maior (respectivamente menor). A altura de um ponto de inflexão é o máximo entre os comprimentos da sequência monótona à sua esquerda e da sequência monótona à sua direita (excluindo-se o ponto de inflexão). Note que um ponto de inflexão pertence sempre a duas sub-sequências, isto é, é sempre o último ponto da sequência monótona à sua esquerda e o primeiro ponto da sequência monótona à sua direita.

Considere uma classe com uma variável de instância `vec` que contém um vector de inteiros de comprimento pelo menos 3, totalmente preenchido. O conteúdo do vector pode ser visto como uma sequência, podendo ter várias subsequências (de)crescentes.

Exemplos:

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

Implemente o método `maiorAltura` que devolve a maior altura de todos os pontos de inflexão do vector `vec`.

```
public int maiorAltura(){
```

```

```

```
}
```