Nome: _____

Nº de aluno: __ __ __ __ __

ISCTE IUL
Instituto Universitário de Lisboa

Exame de Programação Orientada para Objetos
2019 - 2020, 1º semestre, DCTI, 1ª Época

Notas:

- Duração da prova: 30min Parte 1, 2h Parte 2
- Serão concedidos dois **curtos** períodos de perguntas e respostas, o primeiro cerca de 30m depois do início da Parte 2 e o segundo cerca de 1h após o início da prova. As perguntas deverão ser feitas em voz alta, (obviamente, sem dar indicações sobre a resposta a qualquer questão), e as clarificações serão dadas para todos os presentes. **Não serão respondidas quaisquer questões fora desses períodos.** Durante a Parte 1 não haverá períodos de perguntas e respostas.
- Tenha por favor um documento de identificação em cima da mesa desde o início da prova.
- Não se esqueça de indicar o seu nome e **número de aluno** nesta folha, e na folha da Parte 1.
- Deve parar de escrever quando o docente indicar que terminou o tempo, entregar a prova e sair em silêncio.
- O telemóvel deve estar desligado e guardado.
- O exame é individual e sem consulta, qualquer tentativa de quebrar esta regra será penalizada com anulação imediata da prova e processo disciplinar.
- Não destaque folhas nem use folhas de rascunho.
- Responda no enunciado claramente e tenha em atenção o espaço reservado à resposta. Se alguma resposta exceder o espaço reservado continue nas duas folhas do fim indicando claramente a que pergunta se refere o código.
- Não é permitido sair da sala até entregar a prova e o enunciado.
- Só pode sair da sala após 1 hora de prova.
- Pode fazer a prova a lápis, exceto a Parte 1. É da sua responsabilidade garantir que a sua resposta é legível.
- As cotações das questões encontram-se junto das mesmas.
- Ao longo do enunciado estão mencionadas sucintamente algumas classes/operações das bibliotecas do Java que são relevantes para a resolução das questões. Por vezes são apresentadas elementos alternativos que podem ser utilizadas na resposta, o que não significa que todos os elementos tenham que o ser.
- **Ao escrever código nas respostas, <u>não é necessário</u>:**
    - **escrever declarações de importação (*imports*);**
    - **fazer validações de parâmetros, a menos que solicitado explicitamente.**

Parte 2. Problema – Agenda de Contactos

Considere uma aplicação para gestão de uma agenda de contactos. A agenda contém diversos contactos, onde cada qual tem um nome, e um número de telefone opcional. O nome estará sempre definido, e não deverá ser permitida a criação de contactos com nome indefinido, nem com menos do que dois nomes (palavras).

A (3.5)

Escreva a classe Contact, definindo:

- Dois construtores:
  - Apenas fornecendo o nome, validando se a String passada contém pelo menos dois nomes;
  - Fornecendo nome e número.
- Funções para obter:
  - Primeiro nome;
  - Último nome;
  - Saber se algum dos nomes do contacto contém uma dada String.

java.lang.String split(String) : String[] (p.e. String[] parts= s.split(“ “)) contains(String) : boolean	**java.lang.IllegalArgumentException** new IllegalArgumentException(String)

```
public class Contact {
```

B (1.5)

Desenvolva um procedimento estático para ordenar uma lista de Contact, por ordem alfabética do último nome dos contactos. Deverá utilizar a interface Comparator do Java, definindo uma classe compatível ou através de uma expressão lambda.

java.util.List<E>
sort(Comparator<E>) : void
	public interface Comparator<T> {
	  int compare(T a, T b);
	}
		Para obter o resultado da comparação alfabética de duas String *a* e *b*, basta invocar *a.compareTo(b)*.

C (2.0)

a) Defina uma interface para representar um seletor de objetos Contact. Dado um contacto, um seletor decide se o mesmo é aceite ou não. Um possível propósito destes seletores seria servirem como critérios de filtragem para uma lista de contactos.

```
public interface ContactSelector {
```

b) Defina uma classe que implementa a interface definida em (a), sendo o critério de aceitação baseado no facto do contacto ter algum nome que contenha uma dada String, parameterizada no construtor.

D (5.0)

Desenvolva a classe AddressBook, para representar agendas de contactos:

a) Defina os atributos, tendo em conta que os objetos são compostos por uma lista de Contact.

```
public class AddressBook {
```

b) Defina um construtor que recebe um ficheiro (de texto) e carrega objetos Contact do mesmo. O ficheiro contém pares de linhas em sequência, sendo que a primeira contém o nome e a segunda o número (caso não esteja definido, a segunda linha estará vazia). Assuma que o ficheiro está bem formado.

java.util.Scanner	**java.util.ArrayList<E>**	**java.lang.String**
new Scanner(File)	add(E) : boolean	isEmpty() : Boolean
hasNextLine() : boolean	get(int) : E	
nextLine() : String	size() : int	**java.lang.Integer**
nextInt() : int		parseInt(String) : int (estático)
close() : void		

```
public AddressBook(File f) throws FileNotFoundException {
```

c) Defina uma função para saber o número total de contactos da agenda, e um procedimento para adicionar um contacto.

d) Defina duas funções para realizar uma procura de contactos, devolvendo o resultado numa lista de Contact. Considere e utilize os elementos das questões B e C. Uma das funções recebe um objeto compatível com a interface ContactSelector, e devolve todos os contactos aceites. A outra função recebe uma String, e devolve todos os contactos que tenham um nome que contenha essa String. Ambas as funções deverão devolver a lista de contactos ordenada por ordem alfabética do último nome.

E (3.0)

Desenvolva uma classe derivada de AddressBook, estendendo a classe base com a possibilidade de ter grupos de contactos, onde cada grupo é identificado por uma String.

- Deverá existir um procedimento para adicionar um contacto, fornecendo o nome do grupo (caso não exista, será criado). O contacto será adicionado normalmente (como na classe base), e para além disso será associado ao grupo.
- Deverá também existir uma função para obter todos os contactos associados a um grupo, dada uma String (nome do grupo). Caso não exista um grupo com o nome dado, deverá ser devolvida uma lista de contactos vazia.

java.util.HashMap<K,V>
put(K,V) : void
get(K) : V

Espaço para rascunho ou continuação de respostas