Exame de Programação Orientada para Objetos, 2º semestre, 2013 - 2014, DCTI, ISCTE-IUL

(prova-modelo)

Notas:

- Duração da prova: 2h
- Os primeiros 10m, são única e exclusivamente para tirar dúvidas (não contam para o tempo de prova). Nesses 10m leia atentamente toda a prova e ponha as questões que entender em voz alta (sem dar indicações sobre a resposta a qualquer questão). Não serão respondidas quaisquer questões após esse tempo.
- Tenha um documento de identificação em cima da mesa
- Deve parar de escrever quando o docente indicar que terminou o tempo
- Silencie o telemóvel
- Não é permitido sair da sala até entregar a prova
- Não destaque folhas nem use folhas de rascunho
- Só pode sair da sala após 1 hora
- As notas são publicadas no e-learning (ou na página da UC) até dia …
- A revisão de provas terá lugar no gabinete ..., no dia ..., pelas ... horas.

Exercício 1 [cotação 2, tempo estimado de resolução: 5m]

Objetivo: Usar as estruturas de dados fundamentais de uma biblioteca.

Dada a classe Car

```
public class Car {
	private String plate;
	private int yearOfConstruction;

	public Car(String plate, int yearOfConstruction) {
		this.plate = plate;
		this.yearOfConstruction = yearOfConstruction;
	}

	public String getPlate() {
		return plate;
	}

	public int getYearOfConstruction() {
		return yearOfConstruction;
```

```
      }
 // ...
}
```

e um comparador

```
public class ComparatorOfCarByYear implements Comparator<Car> {
 // …
}
```

como utilizaria o método da classe `Collections`

```
public static <T> void sort(List<T> list, Comparator<T> c) { … }
```

para ordenar uma lista de objetos do tipo `Car`?

Exercício 2 [cotação 5, tempo estimado de resolução: 10m]

Objetivo: Usar as estruturas de dados fundamentais de uma biblioteca.

Defina completamente o comparador `Comparator<Car>` que ordene por ordem crescente de ano instâncias da classe `Car`. Em caso de empate deverá usar ordem alfabética de matricula para desempate.

Exercício 3 [cotação 5, tempo estimado de resolução: 20m]

Objetivo: Usar os conceitos de abstração, encapsulamento, herança e polimorfismo.

Dada a seguinte estrutra para a classe base (abstrata) Account e para a classe StatementLine

```
public abstract class ContaBancaria {

      private String id;
      private String nomeCliente;
      private List<Movimento> linhasDeMovimentos = new
LinkedList<Movimento>();
 //  ...

public class Movimento {

      private Date data;
      private Date dataValor;
      private String descricao;
      private double quantiaMovimentada;
```

```
        private double saldoContabilistico;
        private double saldoCorrente;
        private Categoria categoria;
// …
```

(e assumindo definidos todos os inspectores para os atributos de ambas as classes) defina como método abstrato na classe ContaBancaria e faça duas implementações do método saldoMédioEstimado para duas classes derivadas de ContaBancaria (ContaX e ContaY). Na primeira sobreposição o resultado do método deve ser igual a metade do saldo atual da conta (o saldo atual é o saldo do último movimento, assumem-se ordenados por ordem crescente de data). Na segunda implementação o resultado deve ser uma média dos saldos pesada pelo número de dias que cada um esteve depositado (para todos os movimentos registados). Está disponível na classe Data o método:

```
public static int differencaEmDiasEntre(Data data1, Data data2);
```

que calcula a diferença em dias entre as duas datas dadas como argumentos.

Exercício 4 [cotação 6, tempo estimado de resolução: 40m]

Objetivo: Usar mecanismos de controlo de erros. Leitura/Escrita de ficheiros simples

Defina a classe Categoria e um método de classe que permita a leitura de uma lista de categorias de um ficheiro. Cada categoria é constituída por um nome (uma String), uma lista de descrições (lista de String) e uma lista das suas subcategorias. Cada categoria pode ter sub-categorias. O formato do ficheiro deve ser:

```
nome da categoria; categoria base; descrição1; descrição2; … ;
```

Por exemplo:

```
Casa;;EDP;Televisão;Internet;rendas;limpeza
Telecomunicações e Serviços;Casa;EDP;Televisão;Internet
Rendas;Casa;rendas
Outras;Casa;limpeza
Educação;;Iscte;Livros;Material
Material;Educação;Livros;Material
Propinas;Educação;Iscte
```

Constructor and Description

Scanner(**File** source)

Constructs a new Scanner that produces values scanned from the specified file.

Modifier and Type	Method and Description
void	**close**() Closes this scanner.
boolean	**hasNext**() Returns true if this scanner has another token in its input.
boolean	**hasNext**(**Pattern** pattern) Returns true if the next complete token matches the specified pattern.
boolean	**hasNext**(**String** pattern) Returns true if the next token matches the pattern constructed from the specified string.
boolean	**hasNextDouble**() Returns true if the next token in this scanner's input can be interpreted as a double value using the **nextDouble()** method.
boolean	**hasNextFloat**() Returns true if the next token in this scanner's input can be interpreted as a float value using the **nextFloat()** method.
boolean	**hasNextInt**() Returns true if the next token in this scanner's input can be interpreted as an int value in the default radix using the **nextInt()** method.
boolean	**hasNextInt**(int radix) Returns true if the next token in this scanner's input can be interpreted as an int value in the specified radix using the **nextInt()** method.
boolean	**hasNextLine**() Returns true if there is another line in the input of this scanner.

boolean	**hasNextLong**() Returns true if the next token in this scanner's input can be interpreted as a long value in the default radix using the **nextLong()** method.
boolean	**hasNextLong**(int radix) Returns true if the next token in this scanner's input can be interpreted as a long value in the specified radix using the **nextLong()** method.
boolean	**hasNextShort**() Returns true if the next token in this scanner's input can be interpreted as a short value in the default radix using the **nextShort()** method.
boolean	**hasNextShort**(int radix) Returns true if the next token in this scanner's input can be interpreted as a short value in the specified radix using the **nextShort()** method.
String	**next**() Finds and returns the next complete token from this scanner.
String	**next**(**Pattern** pattern) Returns the next token if it matches the specified pattern.
String	**next**(**String** pattern) Returns the next token if it matches the pattern constructed from the specified string.
double	**nextDouble**() Scans the next token of the input as a double.
float	**nextFloat**() Scans the next token of the input as a float.
int	**nextInt**() Scans the next token of the input as an int.
int	**nextInt**(int radix) Scans the next token of the input as an int.

String	**nextLine**() Advances this scanner past the current line and returns the input that was skipped.
long	**nextLong**() Scans the next token of the input as a long.
long	**nextLong**(int radix) Scans the next token of the input as a long.
Scanner	**useDelimiter**(**Pattern** pattern) Sets this scanner's delimiting pattern to the specified pattern.
Scanner	**useDelimiter**(**String** pattern) Sets this scanner's delimiting pattern to a pattern constructed from the specified String.

Exercicio 5 [cotação 2, tempo estimado de resolução: 10m]

Objetivo: Explicar a utilidade da utilização de padrões de desenho de software e demonstrar a sua utilização de padrões simples.

Indique um padrão de desenho de software que tenha encontrado no trabalho final e descreva-o.