Exame de POO de 2015/2016

I

1 - Um produto tem um código, um nome, um preço e a quantidade existente. O código do produto é um inteiro que deve ser atribuído de forma automática cada vez que é criado um novo produto.

a) Para a classe produto defina o cabeçalho, os atributos e um construtor que receba como parâmetro o nome do produto.

```
public class Produto{

    private int codigo;
    private static int ultimo;
    private String nome;
    private double preco;
    private int quantidade;

    public Produto(String nome){

        this.nome = nome;
        ultimo++;
        codigo = ultimo;
        preco = 0.0;
        quantidade = 1;
    }
}
```

b) Construa os getters e setters para os atributos da classe produto.

```
// não vale a pena implementar get() e set() para o último, já que é
alterado automaticamente
public int getCodigo() {
    return codigo;
}

public void setCodigo(int codigo) {
    this.codigo = codigo;
}

public String getNome() {
    return nome;
}

public void setNome(String nome) {
    this.nome = nome;
```

```
}

public double getPreco() {
    return preco;
}

public void setPreco(double preco) {
    this.preco = preco;
}

public int getQuantidade() {
    return quantidade;
}

public void setQuantidade(int quantidade){
    this.quantidade = quantidade;
}
```

c) Para a classe Produto construa o método toString.

```
public String toString(){

    return("código: " + codigo + " nome: " + nome + " preço:" + preco + "
quandidade: " + quantidade);
}
```

d) Para a classe Produto construa um método chamado subirProduto, que permita subir o preço do produto de uma dada percentagem dada como parâmetro. O valor do parâmetro deverá ser um valor entre 0 e 100. Se o valor do parâmetro estiver fora desse intervalo o método deverá gerar um exceção do tipo ValorInvalido. Quando é gerada a exceção deve ser-lhe associada a mensagem de erro: "O valor introduzido, xxx, não está entre 0 e 100". Onde xxx representa o valor dado para o parâmetro do método. A exceção deverá ser capturada posteriormente na função que invocar o método.

```
public void subirProduto(int percentagem) throws ValorInvalido{

        if(percentagem<0 || percentagem>100)
            throw new ValorInvalido("O valor introduzido, " + percentagem
+ ", não está entre 0 e 100");

        preco += preco*(percentagem/100.0);
    }
```

e) Construa a classe de exceção ValorInvalido.

```
public class ValorInvalido extends Exception {

    public ValorInvalido(String msg) {
        super(msg);
    }
}
```

2 - Construa uma classe de teste para a classe produto. Nessa classe deverá:

i) Construir um Produto com o nome “Portátil Asus” e preço 600€ e outro produto com o nome “Portátil HP” e preço 800€. e ii) Aumentar o preço do Portátil Asus para 660€ usando o método subirProduto.

```
public class Teste {

    public static void main(String[] args) {

        Produto p1 = new Produto("Portátil Asus");
        Produto p2 = new Produto("Portátil HP");

        p1.setPreco(600.0);
        p2.setPreco(800.0);

        try{

            p1.subirProduto(10);
        }
        catch(ValorInvalido v){

            System.out.println(v.getMessage());
        }

        System.out.println(p1.getPreco());
    }

}
```

iii) Represente as variáveis que existem no programa que criou e qual o seu valor.

```
// aquela pergunta da treta
```

3 - Construa um método public e static que receba como parâmetro um objeto do tipo java.util.ArrayList. O método deverá devolver o nome do produto mais caro que exista na ArrayList.

```
public static String produtoMaisCaro(ArrayList<Produto> lista){

        double mais_caro = 0.0;
        String nome = "";

        for(int i=0; i<lista.size(); i++){

            if(lista.get(i).getPreco()>mais_caro){

                mais_caro = lista.get(i).getPreco();
                nome = lista.get(i).getNome();
            }
        }

        return(nome);
    }
```

4 - Construa um método public e static que receba como parâmetro um objeto do tipo java.util. ArrayList e uma String com o nome de um produto. O método deverá devolver o número de produtos que existem na lista de produtos com nome igual ao nome dado como parâmetro.

```
public static int produtosIguais(ArrayList<Produto> lista, String nome){

        int iguais = 0;

        for(int i=0; i<lista.size(); i++){

            if(lista.get(i).getNome().equals(nome))
                iguais++;
        }

        return(iguais);
    }
```

5 - Um produto alimentar é um Produto que tem um prazo de validade, isto é, uma data até à qual deverá ser consumido. A data deverá ser um objeto do tipo LocalDate.

a) Construa a classe ProdutoAlimentar definindo o cabeçalho, os atributos e um construtor que receba como parâmetros um objeto do tipo Produto e a sua data de validade.

```
public class ProdutoAlimentar extends Produto{

    private LocalDate data_validade;
```

```
    public ProdutoAlimentar(Produto produto, LocalDate data_validade){

        super(produto.getNome());
        super.setPreco(produto.getPreco());
        super.setQuantidade(produto.getQuantidade());
        super.setCodigo(produto.getCodigo()); // o código deste
ProdutoAlimentar deve ser o mesmo do Produto dado como parâmetro
        this.data_validade = data_validade;
    }

}
```

b) Construa os métodos get e set para o atributo dataValidade.

```
public LocalDate getDataValidade() {
    return data_validade;
}

public void setDataValidade(LocalDate data_validade) {
    this.data_validade = data_validade;
}
```

c) Construa o método equals para a classe ProdutoAlimentar. Note que para a classe Produto não foi construído o método equals e portanto não poderá usá-lo para responder à pergunta.

```
public boolean equals(ProdutoAlimentar p){

    if(p!=null && p.getClass()==this.getClass()){

        if(this.data_validade.compareTo(p.getDataValidade())!=0)
            return(false);

        if(!this.getNome().equals(p.getNome()) ||
this.getPreco()!=p.getPreco()
                || this.getQuantidade()!=p.getQuantidade() ||
this.getCodigo()!=p.getCodigo())
            return(false);

        return(true);
    }

    return(false); // se for impossível comparar "p" ao objeto atual,
devolvemos false
}
```

6 – Considere a classe Inventário listada abaixo:

```
public class Inventario {

    private ArrayList<Produto> produtos;

    public Inventario (){
        produtos = new ArrayList<> ();
    }
}
```

- Para a classe Inventário, construa um método que calcule e devolva o valor total das existências de produtos alimentares fora de prazo. Um produto fora de prazo é um produto cujo prazo de validade é inferior à data em que o programa é executado. Nota: (Para comparar datas, use o método compareTo da classe GregorianCalendar cuja descrição tem em anexo)

```
public double foraDePrazo(ArrayList<ProdutoAlimentar> lista){

    double total = 0.0;

    for(int i=0; i<lista.size(); i++){

        if(lista.get(i).getDataValidade().compareTo(LocalDate.now())<0)
            total += lista.get(i).getPreco();
    }

    return(total);
}
```

II

1 – Explique a diferença entre uma variável de instância e uma variável de classe? Dê um exemplo de cada.

Uma variável de classe é declarada com a **palavra reservada static**, sendo que o valor que elas contêm é o mesmo para todos os membros (i.e. objetos) dessas classe. Por outro lado, uma variável de instância tem um **valor próprio para cada objeto** (i.e. não há partilha de valor, cada um diz respeito apenas ao seu objeto)

2 - Explique a diferença entre sobrecarga (overloading) e sobreposição (overriding) de métodos. Dê um exemplo de cada.

Overloading é quando numa classe temos 2, ou mais, **métodos com o mesmo nome mas parâmetros diferentes** (e.g. int add(int a, int b) e int add(int a, int b, int c) - o segundo método é uma versão mais potente do primeiro) Overriding é quando numa subclasse extendemos o método da classe pai. Ou seja, na classe pai exite

um método e na subclasse **adaptamos/extendemos esse método** para se adequar às necessidade da subclasse.

3 – Explique a diferença entre uma classe abstrata e uma interface. Explique para que servem.

Uma classe abstrara **tem pelo menos 1 método sem implementação** (i.e. só tem o protótipo). Pode depois ter métodos implementados normalmente e os que não estiverem implementados, têm obrigatoriamente de ser implementados pelas subclasses. Uma interface **não tem nenhuma implementação** e todos os métodos têm de ser implementados nas subclasses.

4 – Explique o que é uma exceção? Qual a diferença entre uma exceção verificável e uma exceção não verificável.

Uma exceção é um **erro do qual o programa pode recuperar**. As exceção são tratadas num bloco try{...} catch{...} finally{...} (o finally é opcional) onde a exceção potencialmente ocorre no try, é apanhada no catch e são tomadas medidas dentro desse bloco. Uma **exceção verificável é previsível** e o compilador pode verificar se o código a trata. Por outro lado, uma **exceção não verificável ocorre em tempo de execução** e é difícil prevê-las a priori.