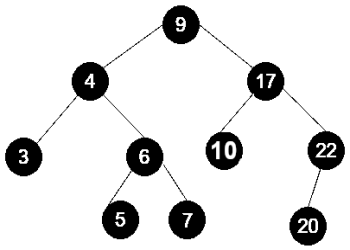
IPS Instituto Politécnico de Setúbal Escola Superior de Tecnologia de Setúbal	**Exame de Programação Avançada 2021/22** Exame de Época de Recurso 26 de fevereiro de 2022 | 10H Duração: 2h

Nome:_____ **Número:**_____

Q1(1,0)	**Q2**(1,0)	**Q3**(1,5)	**Q4**(1,0)	**Q5**(1,5)	**Q6**(2,0)
Q7(1,5)	**Q8**(1,5)	**Q9**(3,0)	**Q10**(2,0)	**Q11**(4,0)	**TOTAL**

Q1 - Travessia de Árvores (1 val)

(1 val) Considere a seguinte árvore binária de pesquisa. Apresente o resultado da sua travessia em **pós-ordem**.



R: Ordem visitados: _____

Q2 - Padrão Simple Factory (1 val)

(1 val) Considere as sub-classes de `Game`: `TetrisGame`, `SnakeGame` e `PongGame`; todas possuem construtores sem argumentos. Complete e implemente classe `GameFactory` que funciona como uma *Simple Factory* para os jogos disponíveis:

```
public class GameFactory {

    public static ____________ create( ________________________________ ) {









    }
}
```

Q3 - Modelação com grafos (1,5 val)

(1,5 val) Dê um exemplo sucinto de um problema que representaria através de um **dígrafo** (grafo-orientado). Indique a assinatura e atributos dos tipos de dados (classes) que armazenaria nos vértices e nas arestas.

Q4 - Algoritmo Dijkstra (1 val)

(1 val) Considere o seguinte resultado do algoritmo *Dijsktra* sobre um **grafo orientado (dígrafo)** e valorado.

Vértice	Custo	Predecessor
A	1	D
B	2	D
C	5	F
D	0	
E	1	D
F	4	A
G	(infinito)	

Responda às seguintes questões:

a) Qual o vértice de origem na execução do algoritmo? _____

b) Qual o custo do menor caminho até ao vértice C? _____

c) Qual o caminho do vértice de origem até ao vértice C? _____

d) O vértice G é garantidamente um vértice isolado do dígrafo? (sim/não)? _____

Q5 - Algoritmia com árvores (1,5 val)

(1,5 val) Considere a classe `BST` que representa uma árvore binária de pesquisa e que armazena inteiros.

- Forneça o código do método `countNodesWithBothSubtrees` que retorna o número de nós que possuem ambas as *sub-árvores*. Recomenda-se uma abordagem recursiva.

```
public class BST {
    private BSTNode root;
    //...
    private class BSTNode {
        int elem;
        BSTNode leftTree;
        BSTNode rightTree;
    }

    public int countNodesWithBothSubtrees() {









}
```

Q6 - Algoritmia com grafos (2 val)

(2 val) Implemente o algoritmo *Depth-First Search* (ver anexo) no método abaixo.

- Deve utilizar os métodos da interface `Graph` (ver anexo).
- No passo `process(v)` deve simplesmente *imprimir o vértice* `v`.

```
public static void dfs(Graph<V, E> graph, Vertex<V> vertice_root) {

}
```

Q7 - Padrão Strategy (1,5 val)

Considere o padrão *Strategy*.

a) (0,5 val) Em que categoria se insere (Criacional, Comportamental ou Estrutural)?

b) (0,5 val) Qual o **problema** que esse padrão propõe resolver?

c) (0,5 val) Faça uma breve comparação entre esse padrão e o padrão *Template Method* (lecionado durante o *refactoring*) no que se refere ao problema que propõem resolver.

Q8 - Padrão Iterador (1,5 val)

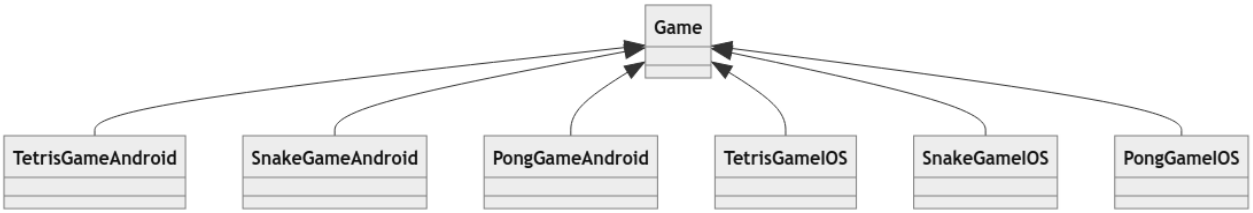
(1,5 val) Considere o seguinte código parcial de uma implementação do ADT List, que utiliza como estrutura de dados uma lista simplesmente ligada.

- Complete o código em falta por forma a que o iterador implementado devolva os elementos armazenados em posições pares {0, 2, 4, ...}.

```
public class MyList<E> {
    private Node first; /* reference of element at index 0*/
    private int size;

    public MyList() {
        first = null;
    }

    private class Node {
        private E elem;
        private Node next;
    }

    private class MyIterator implements Iterator<E> {
        private Node cursor;
        public MyIterator() {









        }

        public boolean hasNext() {









        }

        public E next() {









        }

    }
}
```

Q9 - Padrão Factory Method (3 val)

Recorde a questão anterior sobre o padrão *Simple Factory*. No problema apresentado, surgiu a possibilidade de suportar diferentes plataformas, i.e., *Android* e *iOS*, sendo que a hierarquia de classes passou a ser:



a) (1,5 val) Aplique o padrão **Factory Method**, indicando o código da *interface* `GameFactory` e das fábricas `AndroidGameFactory` e `iOSGameFactory`.

b) (0,75 val) Identifique os **participantes** deste padrão.

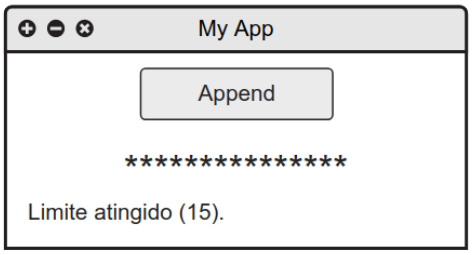
Product	
Factory	
Concrete Product	
Concrete Factory	

c) (0,75 val) Exemplifique a **utilização destas fábricas**, instanciando um jogo de cada plataforma.

```
public static void main(String[] args) {

}
```

Q10 - Padrão MVC (2 val)

(2 val) Considere o código relativo à implementação do padrão MVC de uma aplicação que tem como concatenar uma *string* um asterisco (*) cada vez que se aciona o botão Append, sabendo que não é possível concatenar mais asteriscos quando o limite pré-configurado for atingido (ver figura abaixo); quando o limite é atingido deve ser mostrado um erro ao utilizador `"Limite atingido."`. **Preencha o código em falta nas zonas TODO.**




```
public class Main extends Application {
    //...
    @Override
    public void start(Stage stage) throws Exception {  // TODO




        Scene scene = new Scene(panel);
        stage.setTitle("MVC App");
        stage.setScene(scene);
        stage.show();
    }
}

public class Asterisks extends Subject {
    private String value;
    private int max;

    public Asterisks(int max) {
        this.max = max;
        this.value = "";
    }

    public String getValue() {
        return value;
    }

    public void append() throws AsteriskLimitException { //TODO




    }

}
```

```
public class AsterisksController {
    public Asterisks asterisks;
    public AsterisksPanel asterisksPanel;

    public AsterisksController(Asterisks asterisks, AsterisksPanel
asterisksPanel){        //TODO




    }

    public void doNext() { //TODO









    }
}

public class AsterisksPanel extends BorderPane implements Observer {
    private Asterisks asterisks;
    private Button btn1;
    private Label lblAsterisks;
    private Label lblError;

    public AsterisksPanel(Asterisks asterisks) {
        this.asterisks = asterisks;
        createLayout();
    }

    private void createLayout() {
        btn1= new Button("Append");
        StackPane btnPane= new StackPane();
        btnPane.getChildren().add(btn1);
        lblAsterisks= new Label();
        lblError= new Label();
        setTop(btnPane);
        setCenter(lblAsterisks);
        setBottom(lblError);
        update(null);
    }

    public void setTriggers(AsterisksController ctrl){
        btn1.setOnAction((ActionEvent event) -> { //TODO




        });
    }
```

```
    @Override
    public void update(Object obj) {  // TODO




    }
}
```

Q11 – Refactoring (4 val)

Considere o código do *Anexo - Refactoring*.

a) (1 val) Identifique os seguintes *bad smells*, indicado a(s) linha(s) onde ocorrem.

Bad Smell	**Linha(s)**
Magic Number	
Primitive Obsession	
Inappropriate Intimacy	
Data Class	
Refused Bequest	

b) (1 val) Para cada um dos *bad smells*, indique qual a técnica de refactoring que aplicará (se não souber o nome, descreva-a numa frase).

- **Magic Number:**

- **Primitive Obsession:**

- **Inappropriate Intimacy:**

- **Data Class:**

- **Refused Bequest:**

c) (2 val) Apresente o código final após a aplicação das técnicas de *refactoring* identificadas em b).

(fim enunciado)

[página livre para uso em alguma questão que necessite]

ANEXOS

PODE DESTACAR ESTA FOLHA DO EXAME.

Interface (parcial) de `Graph`

```
public interface Graph<V, E> {
    //...

    public Collection<Vertex<V>> vertices();

    public Collection<Edge<E, V>> edges();

    public Collection<Edge<E, V>> incidentEdges(Vertex<V> v)
        throws InvalidVertexException;

    public Vertex<V> opposite(Vertex<V> v, Edge<E, V> e)
        throws InvalidVertexException, InvalidEdgeException;

    public boolean areAdjacent(Vertex<V> u, Vertex<V> v)
        throws InvalidVertexException;
}
```

As exceções são *não-verificadas*, i.e., extendem de `RuntimeException`.

Algoritmo *Depth-First Search*

```
Algorithm: DFS(graph, vertice_root)
BEGIN
    setVisited(vertice_root)
    push(s, vertice_root)
    WHILE s is not EMPTY DO
        v <- pop(s)
        process(v)
        FOR EACH w adjacentsVerticesOf(v)
            IF w is not visited THEN
                setVisited(w)
                push(s, w)
            END IF
        END FOR
    END WHILE
END
```

Refactoring

```java
public class Product {
    private String reference;
    private int unitPrice; /* Preço unitário */
    private int quantity;
    private double vat; /* imposto, i.e., IVA em Portugal, e.g., 0,23*/

    //construtor, getters e setters
}

public class ShoppingCart extends HashMap<String, Product> {
    private String userName;
    private String userEmail;
    private double total;

    public ShoppingCart(String userName, String userEmail) {
        if(!validateEmail(userEmail)) throw IllegalArgumentException("Invalid email.");

        this.userName = userName;
        this.userEmail = userEmail;
    }

    private boolean validateEmail(String email) {
        return Pattern.compile("^(.+)@(\\S+)$")
            .matcher(email)
            .matches();
    }

    public void addProduct(Product p) throws LimitException {
        if(size() > 100) throw new LimitException(100);

        String reference = p.getReference();
        if(contains(reference) {
            Product existing = get(reference);
            existing.setQuantity( existing.getQuantity()
                                  + p.getQuantity());
        } else {
            put(reference, p);
        }
    }

    public double computeCheckout() {
        total = 0;
        for(Product p : values()) {
            total += (p.getUnitPrice() * (1 + p.getVat())) * p.getQuantity();
        }
        return total;
    }
}
```