Exame- Época Normal de Programação Avançada 2022/23

2 de fevereiro de 2022 (Duração: 2h + 15 minutos tolerância)

Solução

IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

Nome:_____ **Número:**_____

GRUPO1	1|	2|	3|	4|	5|	6|	7|	8|
Preencher Aluno	C	A	C	A	D	C	C	B

GRUPO II

Q1– GRAFOS

a)

```
public class GraphEdgeList<V,E> implements Graph<V,E> {
    private List<MyVertex> vertices;
    private List<MyEdge> edges;

    public GraphEdgeList() {
        vertices = new ArrayList<>();
        edges = new ArrayList<>();
    }

    private class MyVertex implements Vertex<V> {
        private V element;
        public MyVertex(V element) { this.element = element; }
    }
    private class MyEdge implements Edge<E, V> {
        private E element;
        private MyVertex v1;
        private MyVertex v2;
        public MyEdge(MyVertex v1, MyVertex v2, E element) {
            this.element = element;
            this.v1 = v1;
            this.v2 = v2;
        }
    }
}
```

b)

```
Edge<E, V> insertEdge(Vertex<V> v, Vertex<V> w, E edgeElement) {
    /* Ignore validação de parâmetros */
    MyEdge e = new MyEdge(v, w, edgeElement);
    edges.add(e);
    return e;
}

boolean areAdjacent(Vertex<V> v, Vertex<V> w) {
    /* Ignore validação de parâmetros */
    for(MyEdge e : edges) {
        if(e.v1 == v && e.v2 == w || e.v1 == w && e.v2 == v)
            return true
    }
    return false;
}
```

c)

```
public class GraphEdgeListTest {
    @Test
    public void create_and_check_graph() {
        /* All code must go here */
        Grapg<String, Integer> g = new GraphEdgeList<>();
        Vertex<String> a = g.insertVertex("A");
        Vertex<String> b = g.insertVertex("B");
        Vertex<String> c = g.insertVertex("C");
        Vertex<String> d = g.insertVertex("D");
        Edge<Integer, String> e1 = g.insertEdge(a,b,1);
        Edge<Integer, String> e2 = g.insertEdge(d,b,2);
        Edge<Integer, String> e3 = g.insertEdge(c,d,3);
        Edge<Integer, String> e4 = g.insertEdge(a,d,4);
        assertTrue( g.areAdjacent(a, d) );
        assertFalse( g.areAdjacent(a, c) );
    }
}
```

Q2– Pattern Factory

a)

```
public class CitationFactory {
    public static Citation create(String type){
        switch(type){
            case "book": return new BookCitation();
            case "book": return new ChapterCitation();
            case "book": return new JournalCitation();
            default: throw new UnsupportedOperationException("this "+ type + " is
not supported"
        }
    }
}
```

b)

```
public interface CitationFactory {
    public static Citation create(String type);
}



public class APACitationFactory implements CitationFactory{
        public Citation create(String type){
            switch(type){
                case "book": return new APABookCitation();
                case "book": return new APAChapterCitation();
                case "book": return new APAJournalCitation();
                default: throw new UnsupportedOperationException("this "+ type + "
is not supported"
            }
    }
}
public class IEEECitationFactory implements CitationFactory{
        public Citation create(String type){
            switch(type){
                case "book": return new IEEEBookCitation();
                case "book": return new IEEEChapterCitation();
                case "book": return new IEEEJournalCitation();
                default: throw new UnsupportedOperationException("this "+ type + "
is not supported"
            }
        }
}
```

c)

```
public class Main {

    public static void main(String[] args){
        APACitationFactory f1= new APACitationFactory();
        IEEECitationFactory f2= new IEECitationFactory();

        Citation citation1= f1.create("book");
        Citation citation2= f2.create("book");

    }
}
```

Q3 – Pattern Strategy

a)

Participante	Classe/Interface
Context	X
Client	Main
Strategy	StrategyFormat
ConcreteStrategy	StA, StB, StC

b)

```
public interface StrategyFormat {

  public String format(String txt, String inc);
}


public class X {
   private String txt;
   private StrategyFormat st;


    public X(String txt) {
        this.txt = txt;

    }

    public void setSt(Strategy st) {
        this.st = st;
    }

    public String format(String inc) {
       if(st==null)  throw new UnsupportedOperationException(" not supported");
       return st.format(txt,inc);
    }
}
```

c)

```
public class Main {

    public static void main(String[] args) {
      X x= new pt.qs.solution.X("Bom dia");
      x.setSt(new StA1());
      System.out.println(x.format("IPS"));
      x.setSt(new StA2());
      System.out.println(x.format("IPS"));
    }
}


}
```

Q4 – Refatoring

a)

Code Smell	Linha(s) Código	Técnica de Refactoring
Primitive Obsession	3, 12-13	Extract Class / Replace Data/Array with Object
Data Clump	5, 20	Extract Class
Duplicate Code	21-27, 34-40	Extract Method
Inappropriate Intimacy	23, 36, 45	Hide Delegate / Move Method
(Outros) Quais? Magic Number; Data Class; Switch Statements	16,21; 1-9; 12-13; 48-59	Replace Magic Number with Symbolic Constant; Move Method; Nada a fazer

b)

```
class MyDate {
    int day, month, year;
    /* Construtor + getters + setters */
}

public class Person {
    public final String name;
    public final MyDate dateOfBirth;

    public Person(String name, MyDate dateOfBirth) {
        this.name = name;
        this.dateOfBirth = dateOfBirth;
    }

    public String getZodiacSign() {
        int day = dateOfBirth.getDay();
        int month = dateOfBirth.getMonth();
        String sign = "";
        if (month == 1) {
         if (day < 20)
            sign = "Capricorn";
         else
            sign = "Aquarius";
        }
        else if (month == 2) {
            if (day < 19)
                sign = "Aquarius";
            else
                sign = "Pisces";
        }
        /* ... other else ifs ommited, but present */
       return sign;
    }
}

public class Astrology {
    private List<Person> people;
    private static final int MAXIMUM = 100;
    public Astrology() {
        people = new ArrayList<>();
    }

    private int findPerson(String name) {
        for(int i=0; i<people.size(); i++) {
            if(people.get(i).getName().equalsIgnoreCase(name))
                return i;
        }
        return -1;
    }
```

```
    public addPerson(String name, MyDate dateOfBirth) {
        if(people.size() == MAXIMUM) return;

        int index = findPerson(name);
        if(index == -1) return;

        people.add( new Person(name, dateOfBirth) );
    }

    public String getZodiacSignOf(String name) {
        int index = findPerson(name);

        if(index == -1) return "Person not found";

        String sign = people.get(index).getZodiacSign();

        return "Sign of " + name + " is " + sign;
    }
}
```