TESTE 17 -18

PARTE 1

1

(A)

```
public printFamiliaNumerosa (String nomeFich){
    try{
        FileWriter file = new FileWriter (nomeFich);
        BufferedWriter writer = new BufferedWriter (file);
        contribuintes.values()
                .stream()
                .filter(c -> c instanceof Bonificado)
                .forEach(c->{
                    writer.write(c.getNome()+" "
                            +c.getNif()+ " "
                            +((Bonificado)c).reducaoImporto()+"\n");
                });
    } catch (IOException ioe){
        //...
    }
}
```

(B)

```
public int compareTo (Fatura f){
    return data.compareTo (f.getData());
}
```

(C)

```
public Map<Atividade, List<Fatura>> porAtividade(){
    return contribuintes.values().stream()
            .map(Contribuinte::getFaturas)
            .flatMap(List::stream)
            .map(Fatura::clone)
            .collect(
                groupingBy(Fatura::getAtividade));
}
```

PARTE 2

2

```
public class CasaInteligente {
    private Map<String, Lampada> lampadas;
```

(A)

```
    public CasaInteligente(Collection<Lampada> nl) {
        lampadas = new HashMap<>();
        for (Lampada l : nl)
            lampadas.put(l.getId(), l.clone());
    }
```

(B)

```
    public int getEmEco() {
        return lampadas.values()
                .stream()
                .filter(l -> l.getEstado() == 2)
                .count();
    }
```

(C)

```
    public removeLampada(String id) throws NaoExisteEx {
        if(lampadas.remove(id) == null)
            throw new NaoExisteEx("...");
    }
```

(D)

```
    public double consumoTotal() {
        return lampadas.values()
                .stream()
                .mapToDouble(Lampada::totalConsumo)
                .sum();
    }
}
```

PARTE 3

3 (A)

```
public Set<String> topConsumo(int x) {
    return lampadas.values()
            .stream()
            .sorted((l1,l2) -> Integer.compare(l2.totalConsumo(), l1.totalConsumo()))
            .limit(x)
            .map(Lampada::getId)
            .collect(Collectors.toSet());
}
```

(B)

```
public class LampadaLED extends Lampada {
    priv. double per;

    public LampadaLED(String id, double cl, double cet
                      , double per) {
        super(id, cl, ce);
        this.per = per;
    }

    public double totalConsumo() {
        return per * super.totalConsumo();
    }
}
```

(C)

```
public void gravaBinario (String nomeFich) throws LampadaWriteFail {
    try {
        FileOutputStream foo = new FileOutputStream(nomeFich);
        ObjectOutputStream oos = new ObjectOutputStream(foo);
        oos.writeObject(this);
        oos.flush();
        oos.close();
    } catch (Exception exc) {
        throw new LampadaWriteFail(exc.getMessage());
    }
}
```

PARTE 4

4

(a)

```
public abstract Poligono {
    private List<Ponto> pontos;
    public Poligono (Collection<Ponto> c) {
        pontos = c.stream().map(Ponto::clone)
                  .collect(toList());
    }
```

```
public void addPonto (Ponto p) {
        pontos.add (p.clone);
}

public boolean eFechada() {
    return (pontos.size() > 1 && pontos.get(0).equals(
                                    pontos.get(pontos.size()
                                        -1));
}

public double perimetro() {
        Ponto tmp = null; double r = 0.0;
        for (Ponto p : pontos) {
            if (tmp != null) {
            r += tmp.getDistancia(p);
            }
            tmp = p;
        }
        return r;
    }
    public abstract area();
}
```

(B)

// Losangos