UNIVERSITÉ PARIS DESCARTES

IUT

DÉPARTEMENT INFORMATIQUE

DISCIPLINE : BIO

Date de l'épreuve : 06/06/18

Année : 1 Groupe : 112

Écrire très lisiblement (en capitales)

NOM : CAPOSTOLET

Prénom : Arsène

NOTE DE 0 À 20	APPRÉCIATIONS
18	

Ne rien écrire dans cette marge

Mise en bouche

1) public class Constantes {
private static HashMap<String, Double> constantes = new HashMap<String, Double>();

public static void ajouter (String nom, double valeur) throws RuntimeException {
if (constantes.containsKey(nom)) {
throw new RuntimeException ("nom déjà occupé");
}
else {
constantes.add(nom, valeur);
}
}

4/10

```
public static void obtenir(String
nom) throws RuntimeException {
    if(!constantes.containsKey(nom)){
        throw new RuntimeException
        ("Nom inconnu");
    }...
    else {
        constantes.getValue(nom);
    }
}
```

Entrée

2)

```
public class AnnoncesConclues{
    private static LinkedList<Annonce>
    annonces = new LinkedList<Annonce>();
    private static LinkedList<Date> dates
    = new LinkedList<Date>();

    public static void ajouter(Annonce a){
        annonces.add(a);
        dates.push(new Date());
    }

    public static double volumeGlobal(){
        double s = 0;
        for(Annonce a : annonces){
            s += a.montant();
        }
        return s;
    }
```

faites une classe pour encapsuler une Annonce et une Date

Scanned by CamScanner

public static double montantTotal(Date d1, Date d2) {
double s = 0;
for (Date d : dates) {
if (d.after(d1) && d.before(d2)) {
s += annonces(dates.indexOf(d));
}
}
return s;
}

public static void purge() {
annonces.clear();
dates.clear();
}

3) public abstract class AnnonceVelo implements Annonce {
private double montant;

public AnnonceVelo(double montant)
throws IllegalArgumentException {
if (montant < (constante.obtenir("prix plancher"))) {
throw new IllegalArgumentException("prix invalide");
} else {
this.montant = montant;
}
}

3/10

@Override
public double montant() {
    return this.montant;
}

public void setMontant(double montant)
throws IllegalArgumentException {
    if (montant < Contantes.getInstance
("prix plancher")) {
        throw new IllegalArgument
        Exception("montant invalide");
    }
    else {
        this.montant = montant;
    }
}
@Override
public abstract double dime();
}

Etat Principal

4) public class AnnonceNegociée extends
AnnonceValuée {
    private double dime;

    public AnnonceNegociée(double
montant, double dime) throws
IllegalArgumentException {
    if (!(dime < montant)) {
        throw new IllegalArgument
        Exception("dime invalide");
    }

11/10

Ne rien écrire dans cette marge

```
    dire {
        try {
            super(montant);
        } catch (IllegalArgumentException e) {
            throw new IllegalArgumentException(e);
        }
        this.dime = dime;
    }
    @Override
    public double dime() {
        return this.dime;
    }
}

public class AnnonceAuPourcentage
extends AnnonceValuee {
    public AnnonceAuPourcentage(double
    montant) {
        try {
            super(montant);
        } catch (IllegalArgumentException e) {
            throw new IllegalArgumentException(e);
        }
    }
```

1ère instr. du constructeur

inutile

e !

5/6

@Override
public double dime() {
    return montant() * Constantes.
    obtenir("pourcentage");
}

5) public class AnnonceGroupée implements Annonce {
    private ArrayList<Annonce> annonces;

    public AnnonceGroupée() {
        annonces = new ArrayList
        <Annonce>();
    }

    public void addAnnonce(Annonce
    a) {
        annonces.add(a);
    }

    public void removeAnnonce
    (Annonce a) {
        annonces.remove(a);
    }

    @Override
    public double montant() {
        double s = 0;
        for(Annonce a: annonces) {
            s += a.montant - Constantes.
            obtenir("réduction");
        }

6/10

return s;
}
public double Sum() {
double s = 0;
for (Annonce a : annonces) {
s += a.montant();
}
return s;
}
}

Desert

6)
public List<Annonce> select(List<Annonce> annonces, Critere critere) {
for (Annonce a : annonces) {
if (!critere.estSatisfaitPar(a)) {
annonces.remove(a);
}
return annonces;
}

public
7) public class CritereEntreDeuxValeurs implements Critere {
private double valeur1;
private double valeur2;

public CritereEntreDeuxValeurs(double valeur1, double valeur2) {
this.valeur1 = valeur1;
this.valeur2 = valeur2;
}

↳ pas idéal de propager l'effet de bord sur le paramètre

commande

```
public boolean estSatisfait(Commande c){
    return c.montant > valeur1
    && c.montant < valeur2);
}
}
```

8) Java ne propose pas la surcharge des opérateurs.
C'est pourquoi je propose de faire une classe qui implémente critère et utilise une énumération pour l'opérateur. Elle stockerait également les critères qu'elle combine.

```
// OpérateurLogique.java

public enum OpérateurLogique {
NEGATION, CONJONCTION, DISJONCTION}

// CritèreCombinatoire.java

import OpérateurLogique;

public class CritèreCombinatoire implements Critère {
    private OpérateurLogique op;
    private Critère c1;
    private Critère c2;
```

2/10

NOTE DE 0 À 20

APPRÉCIATIONS

Ne rien écrire dans cette marge

```
public CritereCombinatoire(Critere c1,
Critere c2, OperateurLogique op){
    this.c1 = c1;
    this.c2 = c2;
    this.op = op;
}

@Override
public boolean estSatisfaitPar(Annonce a){
    switch(op){
    case NEGATION:
        return !(c1.estSatisfaitPar(a));
        break;
    case CONJONCTION:
        return (c1.estSatisfaitPar(a)
        && c2.estSatisfaitPar(a));
        break;
    case DISJONCTION:
        return (c1.estSatisfaitPar(a)
        || c2.estSatisfaitPar(a));
        break;
    }
}
```

Scanned by CamScanner

Exemple d'utilisation :

public class Appli {
public static void main(String[] args){
AnnonceNégociée a = new AnnonceNégociée(10., 5.);

CritèreEntreDeuxValeurs c1 = new CritèreEntreDeuxValeurs(12., 15.);

CritèreEntreDeuxValeurs c2 = new CritèreEntreDeuxValeurs(12., 15.)

CritèreCombinatoire c3 = new CritèreCombinatoire(c1, null, OpérateurLogique.NEGATION);

c3.estSatisfaitPar(a);

// ça renvoie true

CritèreCombinatoire c4 = new CritèreCombinatoire(c1, c2, OpérateurLogique.CONJONCTION);

c4.estSatisfaitPar(a);

// ça renvoie false

CritèreCombinatoire c5 = new CritèreCombinatoire(c1, c2, OpérateurLogique.DISJONCTION)

c5.estSatisfaitPar(a);
// ça renvoie true