

Nom : MOUSS A

Prénom : Pierre

DS POO 2018-2019

## 1ère session

Tous documents autorisés

durée : 2h

L'objectif du sujet est de réaliser un programme permettant de simuler le fonctionnement d'un distributeur automatique de produits (boisson, barre chocolatées...).

Le sujet comporte plus de points que nécessaire pour obtenir la note maximale. La moyenne se situera autour de 40 points.

Des bonus sont attribués lorsque vous traitez correctement une partie complète.

Le code partiel de ce programme est donné en fin de sujet, il vous appartient de le compléter en fonction des questions. Les questions sont indépendantes mais il est préférable de les traiter dans l'ordre.

Les étoiles (\*, \*\*, \*\*\*) indiquent le niveau de difficulté de la question (facile, moyen, difficile).

➤ ***Les questions sont en gras italique comme ceci précédées d'une flèche.***

**Le barème est sur 100 ; il est indicatif**

### 1 Cahier des charges :

Un distributeur automatique de produits alimentaires est composé d'un monnayeur et d'un système de stockage de produits sous forme de plusieurs files, toutes numérotées.

Le numéro d'une file permet de choisir le produit que l'utilisateur veut acheter.

Les files d'un distributeur ont toutes la même capacité de stockage.

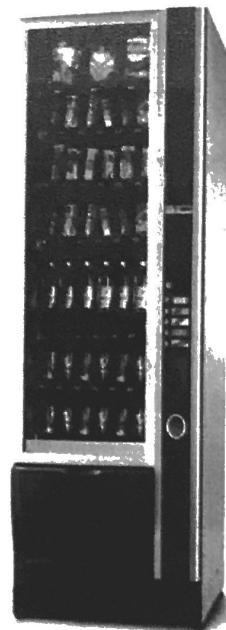
Il existe des monnayeurs qui rendent la monnaie et d'autres qui ne rendent pas la monnaie.

#### 1.1 Code

Le code est organisé en 3 packages : model, ihm et test.

Le package test contient 3 classes de test : une pour les Monnayeur, une pour le Distributeur et une pour l'interface graphique.

Le code est volontairement peu commenté.



## 2 Le modèle

### 2.1 Les produits (8 points) \*/\*\*

Afin de simplifier le sujet, on se contentera d'un seul type concret de produit : le chocolat.

➤ Complétez le code suivant (4 points) (\*):

```

1 package ds4eti2019_1S.model.produits;
2 public interface Produit {
3     /** renvoie le prix du produit */
4     public double getPrix();
5     /** renvoie le nombre de jours restants avant la date limite de consommation */
6     public double getJoursRestants();
7 }
8

9 package ds4eti2019_1S.model.produits;
10 public abstract class AbstractProduit implements Produit {
11     private int nbJoursRestants;
12
13     public AbstractProduit(int nbJoursRestants) {
14         this.nbJoursRestants = nbJoursRestants;
15     }
16
17     @Override
18     public double getJoursRestants() {
19         return nbJoursRestants;
20     }
21 }
22 }

24 package ds4eti2019_1S.model.produits;
25 public class Chocolat extends AbstractProduit {
26     private String nom;
27
28     public Chocolat(int nbJoursRestants, String nom) { //constructeur
29         this.nbJoursRestants = nbJoursRestants;
30         this.nom = nom;
31
32         // J'aime bien mettre le this qd on parle de l'objet de la classe
33     }
34
35     public double getPrix() {
36         return 1.5;
37     }
38 }
```

On souhaite pouvoir trier les produits en fonction du nombre de jours restants avant la date limite de consommation. La collection retenue pour cette fonctionnalité est un TreeSet.

➤ Quelles sont les modifications à apporter aux classes existantes ?(2 points)

- Initialiser un TreeSet      trié produit;
- Implémenter Comparable, modifier la fonction "compareTo()" pour trier par jours restants (sur abstractProduct)
- Ajouter chaque objets du distributeur dans le TreeSet.

➤ Écrivez un petit programme de test permettant de créer quelques Produits, et un TreeSet (pas d'affichage demandé). Expliquez comment se fait le tri (2 pts) \*\*.

```
39 public static void main(String[] args) {  
    • TreeSet<? extends AbstractProduct> = new TreeSet<?>();  
    • // Normalement à ce stade, triéProd.compareTo() a été modifié.  
    • Iterator<? extends AbcProduct> iter = triéProd.iterator();  
    • If (!iter.hasNext()) // On vérifie qu'il est vide (on sait jamais)  
        => triéProd.add(Objets 1)           // Objets initialisés  
          triéProd.add(Objets 2...)         grâce à  
                                         Chocolat Bounty = new Chocolat;  
40 }
```

## 2.2 Les pièces de monnaie (8 points) /\*\*

Les pièces de monnaie ont toutes un PieceType et un poids.

Il existe des pièces étrangères qui ressemblent aux pièces européennes : on leur associe donc le PieceType de la pièce européenne la plus ressemblante.

Ainsi, une pièce de 100 francs CFA sera modélisée comme une pièce étrangère (**PieceHorsEuro**) dont le type est CinquanteCentimes car elle ressemble à la pièce européenne de 50 centimes.

➤ A quoi sert la classe Configuration ? Pourquoi ses méthodes et attributs sont-ils statiques ?(2 point) (\*)

- Permet d'initialiser les configurations de notre distributeurs.
  - méthodes static: on peut utiliser ses méthodes sans initialiser sa classe
  - attributs static: Ils sont les mêmes (en terme de ref) pour tout les objets qui hériteraient de cette classe:
    - Concrètement, on définit des règlages communs.

Static Map< > ...;

même objet / attributs  
référence

<sup>3</sup>  
sous-classe 1

sous-classe 2 ...

➤ Complétez les trous laissés dans le code (5 points) (\*/\*)

➤ Pourquoi ligne de code 92 currentId est-il statique ? (1 point) (\*)

Comme dit avant : si chaque sous-classe référence le même . Si j'ajoute +1 à l'id est ne l'attribut, chaque sous-classe aura un id différent et croissant

```

41 package ds4eti2019_1S.model.monnaie;
42 public enum PieceType {
43     DeuxEuro, UnEuro, CinquanteCentimes, VingtCentimes, DixCentimes;
44 }

45 package ds4eti2019_1S.model;
46 import java.util.HashMap;
47 import java.util.Map;
48
49 import ds4eti2019_1S.model.monnaie.PieceType;
50
51 public class Configuration {
52     /* Attributs et méthodes de classe */
53     private static Map<PieceType,Double> valeurs = initMapValeur();
54     private static Map<PieceType,Double> poids = initMapPoids();
55
56     private static Map<PieceType,Double> initMapValeur(){
57         Map<PieceType,Double> ret = new HashMap<PieceType,Double>();
58         ret.put(PieceType.DeuxEuro, 2.0);
59         ret.put(PieceType.UnEuro, 1.0);
60         ret.put(PieceType.CinquanteCentimes, 0.50);
61         ret.put(PieceType.VingtCentimes, 0.20);
62         ret.put(PieceType.DixCentimes, 0.10);
63         return ret;
64     }
65
66     private static Map<PieceType,Double> initMapPoids(){
67         Map<PieceType,Double> ret = new HashMap<PieceType,Double>();
68         ret.put(PieceType.DeuxEuro, 8.5);
69         ret.put(PieceType.UnEuro, 7.5);
70         ret.put(PieceType.CinquanteCentimes, 7.8);
71         ret.put(PieceType.VingtCentimes, 5.74);
72         ret.put(PieceType.DixCentimes, 4.10);
73         return ret;
74     }
75
76     public static double getPoids(PieceType type) {
77         if (poids.contains(type)) // si le type de pièce existe
78             | return poids.(clé du "type") (j'ai oublié la
79         }                                | syntaxe,
80         public static double getValeur(PieceType type) {          on aurait pu faire
81             if (valeurs.contains(type))                               avec iterator)
82                 | return valeurs.(clé de "type")
83         }
84     }

```

```

81 package ds4eti2019_1S.model.monnaie;
82 public interface Piece {
83     public PieceType getType(); /
84     public double getValeur(); /
85     public double getPoids(); /
86 }

87 package ds4eti2019_1S.model.monnaie;
88 import ds4eti2019_1S.model.Configuration;
89 public abstract class AbstractPiece implements Piece
90 {
91     /*attributs de classe*/
92     private static int currentId = 1;
93     /*Attributs et méthodes d'instance*/
94     private PieceType type;
95     private double masseEnGramme;
96     private int id;
97
98     public AbstractPiece(PieceType type, double masse) { //Const.
99         this.type=type;
100        this.masseEnGramme = masse;
101        this.id = currentId;
102        currentId++;
103    }
104
105    @Override
106    public PieceType getType() {return type;}
107
108    @Override
109    public double getValeur() {return Configuration.getValeur(getType());}
110
111    @Override
112    public double getPoids() {return masseEnGramme;}
113
114    public String toString() {return getValeur()+"€ ("+id+");"}
115
116    public int hashCode() {return id;}
117
118    public boolean equals(Object o) {
119        return o instanceof AbstractPiece && ((AbstractPiece)o).id==id;
120    }
121 }

122 package ds4eti2019_1S.model.monnaie;
123 import ds4eti2019_1S.model.Configuration;
124 public class PieceEuro extends AbstractPiece {
125
126     public PieceEuro(PieceType type, double masse) {
127         this(type); //On renvoie au constructeur au dessous.
128         this.masseEnGramme = masse;
129     } → super(type, masse) //On l'a au dessus :(
130     public PieceEuro(PieceType type) {
131         this(type, Configuration.getPoids(type));
132     }
133 }

```

```

134 package ds4eti2019_1S.model.monnaie;
135 public class PieceHorsEuro extends AbstractPiece {
136     public PieceHorsEuro(PieceType type, double masse) {
137         super(type, masse);
138     }
139     public String toString() {
140         return "X";
141     }
142 }

```

### 2.3 Les Monnayeurs (22 points) (\*/\*\*/\*\*\*)

Un monnayeur collecte les pièces de l'utilisateur.

Certains Monnayeurs rendent la monnaie, d'autre non.

Il dispose de plusieurs files, une pour chaque type de pièce qu'il peut recevoir.

Lorsque l'utilisateur insère une pièce, il teste si la pièce est valide (elle a le bon poids pour son type).

Si c'est le cas, il accepte la pièce et l'insère dans la file adéquate, sinon il la rejète.

Si une pièce étrangère a le poids de la pièce européenne qui lui ressemble, le monnayeur se trompe et accepte la pièce ;

Si une pièce européenne n'a pas le poids qu'elle est censée avoir (cas des fausses pièces par exemple), il la refuse.

Le déroulement du programme de test du Monnayeur est donné page suivante.  
Le code des Monnayeurs est également donné ensuite.

➤ Quels sont les types abstraits de collections utilisés dans la classe Monnayeur ? (2 points) (\*)

→ AbstractCollection (il englobe presque tout aussi)  
 → AbstractSet → AbstractMap → Abstract  
 → AbstractQueue → AbstractEnum? → Abstract

➤ Quels sont les types concrets de collections utilisés dans la classe Monnayeur ? (2 points) (\*)

→ Deque → Queue → LinkedBlockingQueue  
 → Set → EnumMap

➤ Justifiez pour chaque collection concrète de la classe Monnayeur si le choix est pertinent ou non. Si le choix n'est pas pertinent, proposez une autre collection plus adaptée et dites pourquoi<sup>1</sup> (8 points) (\*\*)

Tout dépend de comment on range le monnayeur:  
 → [ ] → on introduit 3 pièces,  
 la première arrive → [ ] → on introduit 3 pièces,  
 la dernière arrive sort

<sup>6</sup> 1 pensez à regarder la documentation fournie en fin de sujet

Il est plus logique de faire du LIFO :

→ Pièce 1 : OK

→ Pièce 2 : OK

Pièce 3 : X → on sort la dernière, avec FIFO ; On est obligé d'en sortir 3 ....

On utilise une "pile" (stack) :

Deck<Piece> stack = new ArrayDeque<Piece>;

On ajoute un objet avec : stack.push(obj)

Vérifie si le type est OK avec :  
if ( stack.peek().getFoids ==  
contigne.getFoid... )

On enlève l'objet avec stack.pop(obj)

```
143 package ds4eti2019_1S.tests;
144 import java.util.Set; (+Simple qu've pas)
145
146 import ds4eti2019_1S.model.monnaie.Monnayeur;
147 import ds4eti2019_1S.model.monnaie.MonnayeurRembourseur;
148 import ds4eti2019_1S.model.monnaie.Piece;
149 import ds4eti2019_1S.model.monnaie.PieceEuro;
150 import ds4eti2019_1S.model.monnaie.PieceHorsEuro;
151 import ds4eti2019_1S.model.monnaie.PieceType;
152
153 ➤ public class TestMonnayeur {
154     public static void main(String args[]) {
155         System.out.println("/**Test du Monnayeur simple");
156         test(new Monnayeur(10));
157         System.out.println("/** Test du Monnayeur rembourseur");
158         test(new MonnayeurRembourseur(15));
159     }
160
161     public static void test(Monnayeur m) {
162         m.add(new PieceEuro(PieceType.DeuxEuro)); //OK
163         m.add(new PieceEuro(PieceType.DeuxEuro, 8.2)); //KO
164         m.add(new PieceHorsEuro(PieceType.DeuxEuro, 7.2)); //KO
165         m.add(new PieceHorsEuro(PieceType.DeuxEuro, 8.5)); //OK
166
167         for(int i = 0; i<11;i++) {
168             m.add(new PieceEuro(PieceType.UnEuro));
169         }
170         for(int i = 0; i<5;i++) {
171             m.add(new PieceEuro(PieceType.DixCentimes));
172         }
173         System.out.println(m);
174         System.out.println("**Test du rendu de monnaie");
175         System.out.println("*Rendre 4€");
176         Set<Piece>monnaie = m.getMonnaie(4);
177         System.out.println(m);
178         System.out.println("Monnaie: "+monnaie);
179
180         System.out.println("\n*Rendre 1.55€");
181         monnaie = m.getMonnaie(1.50);
182         System.out.println(m);
183         System.out.println("Monnaie: "+monnaie);
184     }
185 }
```

```
186 ***Test du Monnayeur simple
187 Plus de place pour les pièces de UnEuro
188 Etat du monnayeur:
189 [2.0€ (1), X]
190 [1.0€ (5), 1.0€ (6), 1.0€ (7), 1.0€ (8), 1.0€ (9), 1.0€ (10), 1.0€ (11), 1.0€ (12), 1.0€ (13),
191 1.0€ (14)]
192 []
193 []
194 [0.1€ (16), 0.1€ (17), 0.1€ (18), 0.1€ (19), 0.1€ (20)]
195
196 **Test du rendu de monnaie
197 *Rendre 4€
198 Je ne rends pas la monnaie!
199 Etat du monnayeur:
200 [2.0€ (1), X]
201 [1.0€ (5), 1.0€ (6), 1.0€ (7), 1.0€ (8), 1.0€ (9), 1.0€ (10), 1.0€ (11), 1.0€ (12), 1.0€ (13),
202 1.0€ (14)]
203 []
204 []
205 [0.1€ (16), 0.1€ (17), 0.1€ (18), 0.1€ (19), 0.1€ (20)]
206
207 Monnaie: []
208
209 *Rendre 1.55€
210 Je ne rends pas la monnaie!
211 Etat du monnayeur:
212 [2.0€ (1), X]
213 [1.0€ (5), 1.0€ (6), 1.0€ (7), 1.0€ (8), 1.0€ (9), 1.0€ (10), 1.0€ (11), 1.0€ (12), 1.0€ (13),
214 1.0€ (14)]
215 []
216 []
217 [0.1€ (16), 0.1€ (17), 0.1€ (18), 0.1€ (19), 0.1€ (20)]
218
219 Monnaie: []
220 *** Test du Monnayeur remboursleur
221 Etat du monnayeur:
222 [2.0€ (21), X]
223 [1.0€ (25), 1.0€ (26), 1.0€ (27), 1.0€ (28), 1.0€ (29), 1.0€ (30), 1.0€ (31), 1.0€ (32), 1.0€
224 (33), 1.0€ (34), 1.0€ (35)]
225 []
226 []
227 [0.1€ (36), 0.1€ (37), 0.1€ (38), 0.1€ (39), 0.1€ (40)].
228
229 **Test du rendu de monnaie
230 *Rendre 4€
231 Etat du monnayeur:
232 []
233 [1.0€ (25), 1.0€ (26), 1.0€ (27), 1.0€ (28), 1.0€ (29), 1.0€ (30), 1.0€ (31), 1.0€ (32), 1.0€
234 (33), 1.0€ (34), 1.0€ (35)]
235 []
236 []
237 [0.1€ (36), 0.1€ (37), 0.1€ (38), 0.1€ (39), 0.1€ (40)]
238
239 Monnaie: [2.0€ (21), X]
240
241 *Rendre 1.55€
242 Etat du monnayeur:
243 []
244 [1.0€ (26), 1.0€ (27), 1.0€ (28), 1.0€ (29), 1.0€ (30), 1.0€ (31), 1.0€ (32), 1.0€ (33), 1.0€
245 (34), 1.0€ (35)]
246 []
247 []
248 []
249
250 Monnaie: [0.1€ (36), 0.1€ (37), 0.1€ (38), 0.1€ (39), 0.1€ (40), 1.0€ (25)]
```

```

251 package ds4eti2019_1S.model.monnaie;
252 import java.util.EnumMap;
253 import java.util.HashSet;
254 import java.util.Map;
255 import java.util.Queue;
256 import java.util.Set;
257 import java.util.concurrent.LinkedBlockingQueue;
258 import ds4eti2019_1S.model.Configuration;
259
260 public class Monnayeur {
261     private Map<PieceType, Queue<Piece>> pieces = new EnumMap<PieceType>(),
262     Queue<Piece>>(PieceType.class);
263     private int capacite;
264
265     public Monnayeur(int capacite) {
266         setCapacite(capacite);
267         //construit la collection de pieces
268         for(PieceType type: PieceType.values()) {
269             pieces.put(type, new LinkedBlockingQueue<Piece>(this.capacite));
270         }
271     }
272
273     protected final void setCapacite(int capacite) {
274         if(capacite>0) {
275             this.capacite = capacite;
276         }
277     }
278
279     public String toString() {
280         String ret ="Etat du monnayeur:\n";
281         for(PieceType type: pieces.keySet()) {
282             Queue<Piece> filePieces = pieces.get(type);
283             ret += filePieces + "\n";
284         }
285         return ret;
286     }
287
288     public Set<Piece> getMonnaie(double somme){
289         System.out.println("Je ne rends pas la monnaie!");
290         return new HashSet<Piece>();
291     }
292
293     public boolean add(Piece p) {
294         boolean ret = false;
295         //la piece a un poids correspondant à son type
296         //l'ajout dans la bonne file est possible (capacite ok)
297         try {
298             if(p.getPoids()==Configuration.getPoids(p.getType()) &&
299                 pieces.get(p.getType()).add(p)){
300                 ret=true;
301             }
302         }
303         catch(IllegalStateException ise) {
304             System.err.println("Plus de place pour les pièces de "+p.getType());
305         }
306         return ret;
307     }
308
309     public Piece remove(PieceType type) {
310         return pieces.get(type).poll();
311     }
312 }

```

```

313 package ds4eti2019_1S.model.monnaie;
314 import java.util.HashSet;
315 import java.util.Set;
316 import ds4eti2019_1S.model.Configuration;
317
318 public class MonnayeurRembourseur extends Monnayeur {
319     public MonnayeurRembourseur(int capacite) {
320         super(capacite);
321     }
322
323     public Set<Piece> getMonnaie(double somme) {
324         Set<Piece> ret = new HashSet<Piece>();
325         for(PieceType type : PieceType.values()) {
326             boolean fini=false;
327             while(!fini && somme >= Configuration.getValeur(type)) {

```

2.4 L  
33

Iterator <Piece> iter = ~~set.~~piece.iterator();

while (iter.hasNext()) → 0,10 en mai

↓

so if (somme → 0)

{ somme -= iter.next().getValeurs();  
ret.add (iter.next());  
}

} // A Améliorer // TODO : check si la diff ne tombe pas < 0

On parcours piece (du monnayeur). Si somme > 0,10 (plus petite pièce)  
on soustrait.

```

328     }
329     return ret;
330 }
331 }
332 }
```

⚠ Si somme = 0.20  
et on a pièce = 1€  
↳ erreur

J'ai pas le temps de faire mieux, mais on aurait put check si la diff est < 0  
→ Complétez la ligne 310 (1 point) (\*\*)

On est dans un file queue,  
on remove la tête (FIFO)

➤ Terminez de coder la méthode getMonnaie de la classe MonnayeurRembourseur (6 points) (\*\*)

➤ Comment pourrait-on rendre plus générique la modélisation des Monnayeurs et mieux respecter la philosophie objet ? Décrivez votre proposition (code facultatif) (3 point) (\*\*)

1: Une classe Monnayeur Abstraite avec des fonctions utiles en abstraite: getMonnaie ... .

2: Une interface Rend La Monnaie avec getMonnaie dedans histoire que si plusieurs monnayeur ont la fonction c'est simple de s'en rendre compte etc :

public interface Rend Monnaie {

public .Constructeur /

3

→ public abstract getMonnaie (double somme);

on doit rendre  
la somme en pièce  
(faire l'accord)

## 2.4 Le distributeur (22 points)(\*\*/\*\*\*)

Prenez connaissance du code suivant ; les questions et indications viennent après.

```

333 package ds4eti2019_1S.model;
334 //imports non donnés dans le sujet par soucis de gain de place
335 public class Distributeur {
336
337     private Monnayeur monnayeur;
338     private List<Stack<Produit>> produits; →
339     private int capacite;
340     private int nbFiles;
341     private double sommeCourante=0;
342
343     public Distributeur(int capacite, int nbFiles, Monnayeur m) {
344         this.capacite = capacite;
345         this.nbFiles = nbFiles;
346         this.monnayeur = m;
347         produits = new ArrayList<Stack<Produit>>();
348         for(int i=0;i<nbFiles;i++) {
349             produits.add(new Stack<Produit>());
350         }
351     }
352
353     public boolean add(Produit p) {
354         boolean ret = false;
355         boolean fini = false;
356         int i=0;
357         while(i<nbFiles && !fini) { // i = 0 i+1
358             if(produits.get(i).size() < this.capacite) { // en gros
359                 produits.get(i).push(p);                                on est pas
360                 fini = false;                                         plein
361             } else {                                                 fini=true;
362                 }
363             }
364         }
365         return ret;
366     }
367
368     public final Retour getRetour(int num, Set<Piece> pieces) {
369         Produit produit = null;
370         Set<Piece> monnaie = new HashSet<Piece>();
371
372         Set<Piece> refusees = ajoutePieces(pieces);
373         produit = getProduit(num, sommeCourante);           produit: article à bonne
374                                                 position
375         System.out.println(sommeCourante); refusee.
376         //ajout des piece refusées à monnaie
377         Iterator<Piece> iter2 = refusees.iterator(); // on parcours les
378         Iterator<Piece> iter = monnaie.iterator();          pièces
379         while (iter.hasNext()) {                            on ajoute au
380             if (refusees.contains(iter.next())) {            refusées
381                 monnaie.add(iter.next());                  }
382             }
383         }
384         //restitution des pièces par le monnayeur
385         if(produit == null) { //produit invalide : remboursement de ce qui a été versé
386             Iterator<Pieces> iter2 = monnaie.iterator();
387         }

```

22

```

int somme
while (iter.hasNext())
    somme += iter.next().getValeur() // on incrémente la
                                    somme
                                    à rendre
} m. getMonnaie(somme) → On rend tout
else {//Produit non null : restitution des sommes trop perçues
    dem = produit.getValeur(); → some produit
    while (iter.hasNext())
        { while (dem < somme)
            m. getMonnaie(somme - dem)
}
    }
382
383     sommeCourante = [0 ?]; // ?
384     return new Retour(monnaie, produit);
385 }
386
387 private Produit getProduit(int num, double somme) {
388     Produit p = null;
389     if (num < produits.size() &&
390         !produits.get(num).isEmpty()) {
391         Produit candidat = produits.get(num).peek();
392         if (somme >= candidat.getPrix()) {
393             p = produits.get(num).pop();
394         }
395     }
396     return p;
397 }
398
399 private Set<Piece> ajoutePieces(Set<Piece> pieces) {
400     Set<Piece> refusees = new HashSet<Piece>();
401     //ajout des pieces valide au monnayeur et maj de sommecourante ;
402     for (Piece p: pieces) {
403         if (monnayeur.add(p)) {
    sommeCourante += p.getValeur(),
}
404     }
405     else {//piece non valide à ajouter aux pièces refusées
        refusee.add (p)
}
406     }
407     return refusees;
408 }
409
410
411 public String toString() {
412     String ret = "***Distributeur:\n";
413     int i=1;
414     for (Stack<Produit> file: produits) {
415         ret += "\t* file " + i + ":" + file + "\n";
416         i++;
417     }
418     ret += monnayeur;
419     return ret;
420 }

```

```

421     public List<ArrayList<String>> getInfos() {
422         List<ArrayList<String>> infos = new ArrayList<ArrayList<String>>();
423         for(Stack<Produit> s:produits) {
424             ArrayList<String> al = new ArrayList<String>();
425             if(!s.isEmpty())al.add(s.peek().toString());
426         }
427         return infos;
428     }
429 }
430
431 public class Retour {
432     private Set<Piece> pieces;
433     private Produit produit;
434
435     public Retour(Set<Piece> pieces,Produit produit) {
436         this.pieces = pieces;
437         this.produit = produit;
438     }
439
440     public Set<Piece> getPieces() {return pieces;}
441     public Produit getProduit() {return produit;}
442 }
443
444 package ds4eti2019_1S.tests;
445 //import non donnés dans le sujet par soucis de gain de place
446 public class TestDistributeur {
447     public static void main(String[] args) {
448         Monnayeur m = new MonnayeurRembourseur(5);
449         Distributeur d = new Distributeur(4,3,m); //trois files de 4 places
450         ajouteProduits(d);
451         System.out.println(d);
452         Set<Piece> monnaie = new HashSet<Piece>();
453         createMonnaie(monnaie);
454         d.getRetour(5, monnaie); //KO
455         System.out.println(d);
456         d.getRetour(2, monnaie); //OK
457         System.out.println(d);
458     }
459
460     private static void createMonnaie(Set<Piece> monnaie) {
461         monnaie.add(new PieceEuro(PieceType.UnEuro)); //OK
462         monnaie.add(new PieceEuro(PieceType.DixCentimes)); //OK
463         monnaie.add(new PieceEuro(PieceType.VingtCentimes,8.2)); //KO
464         monnaie.add(new PieceHorsEuro(PieceType.DeuxEuro,7.2)); //KO
465         monnaie.add(new PieceHorsEuro(PieceType.DeuxEuro,8.5)); //OK
466     }
467
468     private static void ajouteProduits(Distributeur d) {
469         d.add(new Chocolat(10,"Croustitruc"));
470         d.add(new Chocolat(5,"Croustitruc"));
471         d.add(new Chocolat(3,"CroustiMachin"));
472         d.add(new Chocolat(6,"Noir"));
473         d.add(new Chocolat(6,"Blanc"));
474         d.add(new Chocolat(6,"Lait"));
475         d.add(new Chocolat(6,"Noisette"));
476         d.add(new Chocolat(10,"Croustitruc"));
477         d.add(new Chocolat(5,"Croustitruc"));
478         d.add(new Chocolat(3,"CroustiMachin"));
479         d.add(new Chocolat(6,"Noir"));
480         d.add(new Chocolat(6,"Blanc"));
481         d.add(new Chocolat(6,"Lait"));
482     }
483 }

```

| nb Jours Restants

**Sortie console :**

```

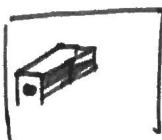
484 ***Distributeur:
485     * file 1:[Croustitruc, Croustitruc, CroustiMachin, Noir]
486     * file 2:[Blanc, Lait, Noisette, Croustitruc]
487     * file 3:[Croustitruc, CroustiMachin, Noir, Blanc]
488 Etat du monayeur:
489 []
490 []
491 []
492 []
493 []
494
495 3.1
496 ***Distributeur:
497     * file 1:[Croustitruc, Croustitruc, CroustiMachin, Noir]
498     * file 2:[Blanc, Lait, Noisette, Croustitruc]
499     * file 3:[Croustitruc, CroustiMachin, Noir, Blanc]
500 Etat du monayeur:
501 []
502 []
503 []
504 []
505 []
506
507 3.1
508 ***Distributeur:
509     * file 1:[Croustitruc, Croustitruc, CroustiMachin, Noir]
510     * file 2:[Blanc, Lait, Noisette]
511     * file 3:[Croustitruc, CroustiMachin, Noir, Blanc]
512 Etat du monayeur:
513 [X]
514 []
515 []
516 []
517 []

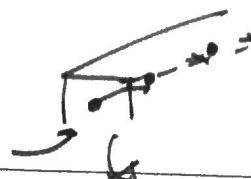
```

➤ La sortie console donnée est-elle bien conforme au programme de test ? Justifiez (2 points)

- On a 4 places par piles. On crée 3 piles, on ajoute 13 éléments, ça choque personne
- on a un problème avec le get produit  
Après l'étape 3 : on enlève le 2:  
On a un ~~est~~ System.out de get retour qui apparaît par  $\oplus$  Somme comme ?
- Expliquez le choix de la collection `List<Stack<Produit>>` produits pour représenter l'organisation des produits dans le distributeur. (2 points) (\*)

C'est une stack car on a un distributeur:



 C'est du LIFO !

➤ complétez la méthode add : on cherche la première place disponible sur l'ensemble des piles. (4 points) (\*\*)

➤ quelle méthode doit-on appeler lorsqu'on souhaite utiliser le distributeur pour acheter un produit ? A quoi correspondent ses paramètres ? (2 points) (\*\*)

On utilise getRetour qui appelle getProduit  
→ num : file du produit (position)  
→ pieces : pièces qu'on met dans le monnayeur  
(issue de monnaie, issu de createMonnaie, issu de "add" de pièce)

➤ pourquoi la méthode getProduit est-elle privée ? (2 points)

Elle est utilisée seulement dans la classe et le doit : si elle est autre, une sous classe peut l'utiliser et modifier son comportement.  
↳ Arnaque de distributeur.

➤ pourquoi la méthode getRetour est-elle final ? Qu'est ce que cela implique ? (3 points)

On ne peut pas avoir un article autrement qu'avec cette méthode : les classes filles de celle-ci ne peuvent pas modifier getRetour()  
Cela signifie soit que la forme finale de Retour est cette classe (dernier fils), soit qu'on veut pas d'enfants modifiant Distributeur (On met les final en bout d'arbre souvent)

➤ A quoi sert la classe Retour ? Pourquoi a-t-elle été créée ? (2 points) (\*\*)

La sortie de getRetour est de type Retour.  
→ On définit ce à quoi doit ressembler un retour.  
→ On fait alors un clone du produit retourné.  
↳ C'est comme ça on retourne une copie de l'image du produit et on diminue qd même les indices de stock

➤ Complétez le code de la méthode getRetour ? (4 points) (\*\*/\*\*\*)

➤ Complétez le code de la méthode ajoutePieces ? (3 points) (\*)

➤ Soit on veut "garder" un historique des objets  
Soit on veut juste pouvoir faire différentes types de retours (ajouter des System.out ....)

## 2.5 Interface graphique (20 points) (\*/\*\*)

Prenez connaissance du code suivant

```
518 public class TestGUI {  
519  
520     public static void main(String[] args) {  
521         Monnayeur m = new MonnayeurRembourseur(30);  
522         Distributeur d = new Distributeur(25, 10, m);  
523         DistributeurIHM ihm = new DistributeurIHM(d);  
524     }  
525 }  
526  
527 public class Clavier extends JPanel implements ActionListener {  
528     private static final long serialVersionUID = 1L;  
529     private List<JButton> lesBoutons;  
530     private int numero = 0;  
531     private JLabel message;  
532  
533     public Clavier() {  
534         lesBoutons = new ArrayList<JButton>();  
535         buildClavier();  
536     }  
537  
538     private void buildClavier() {  
539         JPanel chiffres = new JPanel();  
540         chiffres.setLayout(new GridLayout(4,3));  
541         for(int i=0;i<9;i++) {  
542             JButton bt = new JButton(""+(i+1));  
543             bt.addActionListener(this);  
544             lesBoutons.add(bt);  
545             chiffres.add(bt);  
546         }  
547  
548         setLayout(new BorderLayout());  
549         message = new JLabel("Numéro:");  
550         add(message,BorderLayout.SOUTH);  
551         add(chiffres,BorderLayout.CENTER);  
552     }  
553  
554     @Override  
555     public void actionPerformed(ActionEvent e) {  
556         numero = numero *10 + lesBoutons.indexOf(e.getSource())+1;  
557         message.setText("Numéro: " + numero);  
558         repaint();  
559     }  
560  
561     public int getNumero() {  
562         return numero;  
563     }  
564  
565     public void efface() {  
566         }  
567     }
```

**ActionListener** {

Chiffres → Sud Center

Classe Anonyme ??

this.message = 0 ;

```

568 public class DistributeurIHM extends JFrame{
569
570     private static final long serialVersionUID = 1L;
571     private Clavier clavier;
572     private ProduitsIHM produitsIHM;
573     private Distributeur distributeur;
574
575     public DistributeurIHM(Distributeur distributeur) {
576         this.distributeur = distributeur;
577         clavier = new Clavier();
578         produitsIHM = new ProduitsIHM(distributeur);
579         buildFrame();
580         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
581         setSize(500,500);
582         setVisible(true);
583     }
584
585     private void buildFrame() {
586         JPanel fond = new JPanel();
587         fond.setLayout(new BorderLayout());
588         fond.setBackground(Color.green);
589         fond.add(clavier,BorderLayout.EAST);
590
591         fond.add(new JLabel("Mon super distributeur"),BorderLayout.NORTH);
592
593         Container boutons = new JPanel();
594         JButton btReset = new JButton("Effacer");
595         btReset.addActionListener(new Bouton Listener Rest)

```

→ class Bouton Listener Rest implements ActionListener()

{

    @Override  
    public void actionPerformed(ActionEvent e)

    {  
        fond.repaint();      // on enlève

        ↳ fond.setReset style

}

596 );

597 JButton btValider = new JButton("Valider");

598 class Bouton Listener Valide r implements ActionListener

599 {  
600 @Override  
601 public void actionPerformed(ActionEvent e)

602 {  
603 TestGuit test = new TestGuit();

604 }  
605 ↳ action de validation du code

→ on entre  
les chiffres

```

599 boutons.add(btReset);
600 boutons.add(btValider);
601 fond.add(boutons, BorderLayout.SOUTH);
602 fond.add(produitsIHM, BorderLayout.CENTER);
603 setContentPane(fond);
604 pack();
605 }
606 }
```

```

607 public class ProduitsIHM extends JPanel{
608     private Distributeur distributeur;
609
610     public ProduitsIHM(Distributeur d) {
611         distributeur = d;
612         setPreferredSize(new Dimension(400,400));
613     }
614     public void paintComponent(Graphics g) {
615         int x=0;
616         int y=0;
617         for(ArrayList<String> liste: distributeur.getInfos()) {

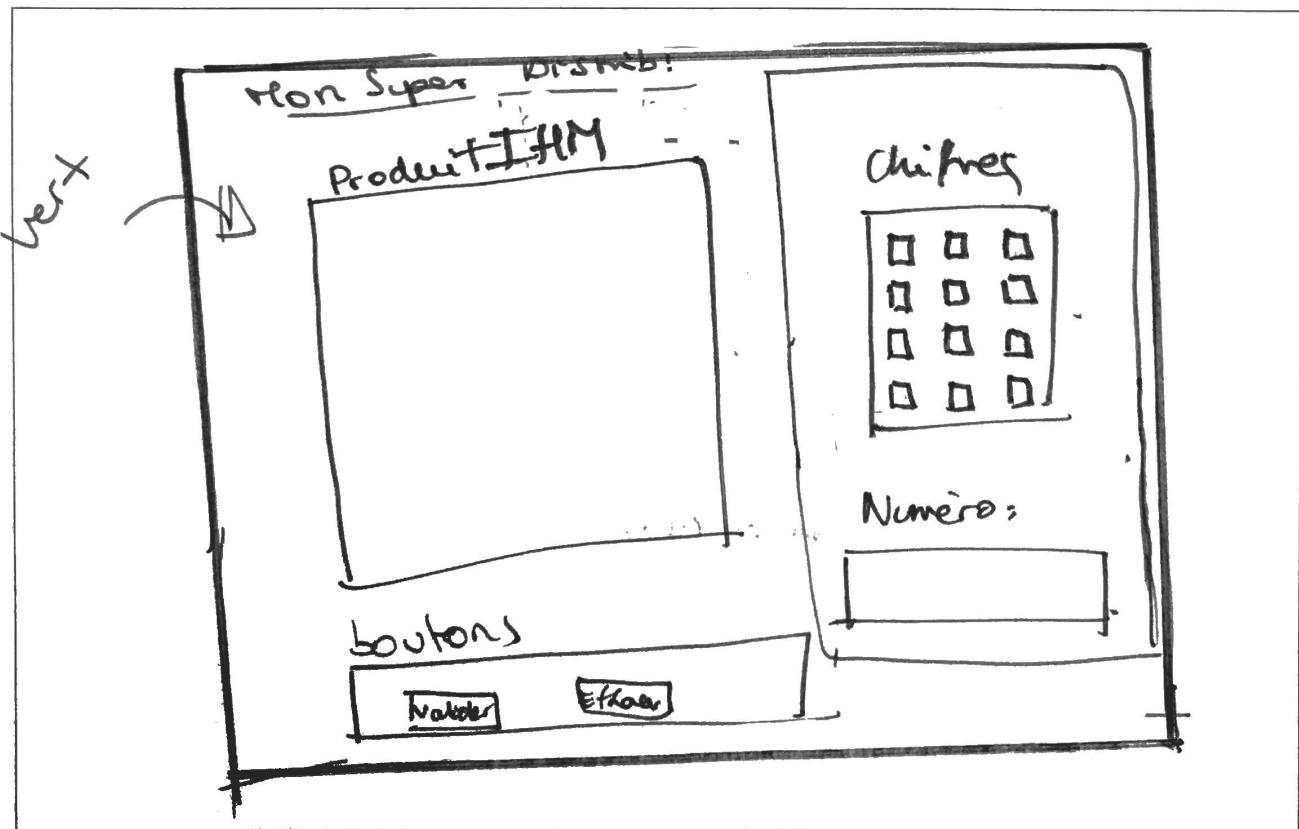
```

```

618     }
619 }

```

➤ Dessinez l'interface graphique correspondant au code donné ; vous indiquerez le nom des composants sur votre dessin (6 points) (\*)



- La méthode `efface de Clavier` réinitialise le numéro affiché dans le label « message ». Codez-là (2 points) (\*)
- Complétez la ligne 527, et donnez le code qui suit les lignes 595 et 598. (8 points) (\*\*/\*\*)
- Codez le dessin des produits du distributeur. (ligne 617) (6 points) (\*\*\*)

On se contentera de dessiner le produit qui se situe au premier plan de chaque file, en indiquant son nom. Si la file est vide, on ne dessine rien. Si il y a un produit, on dessinera un rond bleu.

Code pour dessiner un rond bleu et écrire une chaîne de caractères noire :

```
621     g.setColor(Color.blue);
622     g.fillOval(0, 0, 40, 40); //rond contenu dans le carré de coin supérieur gauche
623     (0,0) et de côté 40 px
624     g.setColor(Color.BLACK);
625     g.drawString("Hello", 0, 0); //chaine "Hello" dessinée aux coordonnées (0,0)
```

### 3 Question de synthèse (18 point)

Les classes Monnayeur et Distributeur proposent toutes les deux des comportements de gestion de collection (ajouter, enlever, etc.).

- Que faudrait-il faire pour que ces classes soient effectivement des collections au sens de l'objet et de Java ? Quelles sont les différentes solutions possibles ?
- Étudiez la question pour la classe Monnayeur. (9 points)
- Étudiez la question pour la classe Distributeur (9 points)

**Il n'est pas nécessaire de donner du code. Vous pouvez vous aider de schémas.**

## 4 Documentation

### 4.1 Class `LinkedBlockingQueue<E>` extends `AbstractQueue<E>` implements `BlockingQueue<E>`

- `java.lang.Object`
  - `java.util.AbstractCollection<E>`
    - `java.util.AbstractQueue<E>`
      - `java.util.concurrent.LinkedBlockingQueue<E>`

- Type Parameters:

`E` - the type of elements held in this collection

All Implemented Interfaces:

`Serializable`, `Iterable<E>`, `Collection<E>`, `BlockingQueue<E>`, `Queue<E>`

An optionally-bounded **blocking queue** based on linked nodes. This queue orders elements FIFO (first-in-first-out). The *head* of the queue is that element that has been on the queue the longest time. The *tail* of the queue is that element that has been on the queue the shortest time. New elements are inserted at the tail of the queue, and the queue retrieval operations obtain elements at the head of the queue. Linked queues typically have higher throughput than array-based queues but less predictable performance in most concurrent applications.

The optional capacity bound constructor argument serves as a way to prevent excessive queue expansion. The capacity, if unspecified, is equal to `Integer.MAX_VALUE`. Linked nodes are dynamically created upon each insertion unless this would bring the queue above capacity.

This class is a member of the [Java Collections Framework](#).

#### Constructor and Description

##### `LinkedBlockingQueue()`

Creates a `LinkedBlockingQueue` with a capacity of `Integer.MAX_VALUE`.

##### `LinkedBlockingQueue(Collection<? extends E> c)`

Creates a `LinkedBlockingQueue` with a capacity of `Integer.MAX_VALUE`, initially containing the elements of the given collection, added in traversal order of the collection's iterator.

##### `LinkedBlockingQueue(int capacity)`

Creates a `LinkedBlockingQueue` with the given (fixed) capacity.

Modifier and Type	Method and Description
<code>void</code>	<code>clear()</code> Atomically removes all of the elements from this queue.
<code>boolean</code>	<code>contains(Object o)</code> Returns true if this queue contains the specified element.
<code>Iterator&lt;E&gt;</code>	<code>iterator()</code> Returns an iterator over the elements in this queue in proper sequence.
<code>boolean</code>	<code>offer(E e)</code> Inserts the specified element at the tail of this queue if it is possible to do so immediately without exceeding the queue's capacity, returning <code>true</code> upon success and <code>false</code> if this queue is full.
<code>boolean</code>	<code>offer(E e, long timeout, TimeUnit unit)</code> Inserts the specified element at the tail of this queue, waiting if

necessary up to the specified wait time for space to become available.

**peek()**

E

Retrieves, but does not remove, the head of this queue, or returns `null` if this queue is empty.

**poll()**

E

Retrieves and removes the head of this queue, or returns `null` if this queue is empty.

**put(E e)**

void

Inserts the specified element at the tail of this queue, waiting if necessary for space to become available.

**remainingCapacity()**

int

Returns the number of additional elements that this queue can ideally (in the absence of memory or resource constraints) accept without blocking.

**remove(Object o)**

boolean

Removes a single instance of the specified element from this queue, if it is present.

**size()**

int

Returns the number of elements in this queue.

## 4.2 public class Stack<E> extends Vector<E>

- [java.lang.Object](#)
  - [java.util.AbstractCollection<E>](#)
    - [java.util.AbstractList<E>](#)
      - [java.util.Vector<E>](#)
        - [java.util.Stack<E>](#)
  - All Implemented Interfaces: [Serializable](#), [Cloneable](#), [Iterable<E>](#), [Collection<E>](#), [List<E>](#)

The `Stack` class represents a last-in-first-out (LIFO) stack of objects. It extends class `Vector` with five operations that allow a vector to be treated as a stack. The usual push and pop operations are provided, as well as a method to peek at the top item on the stack, a method to test for whether the stack is empty, and a method to search the stack for an item and discover how far it is from the top.

When a stack is first created, it contains no items.

A more complete and consistent set of LIFO stack operations is provided by the `Deque` interface and its implementations, which should be used in preference to this class. For example:

```
Deque<Integer> stack = new ArrayDeque<Integer>();
```

### Constructor and Description

**Stack()** Creates an empty Stack.

### Modifier and Type

### Method and Description

boolean

**empty()**

Tests if this stack is empty.

E

**peek()**

Looks at the object at the top of this stack without removing it from the stack.

- E **pop()**  
Removes the object at the top of this stack and returns that object as the value of this function.
- E **push(E item)**  
Pushes an item onto the top of this stack.

### 4.3 Méthodes de Collection

Modifier and Type	Method and Description
boolean	<b><u>add(E e)</u></b> Ensures that this collection contains the specified element (optional operation).
boolean	<b><u>addAll(Collection&lt;? extends E&gt; c)</u></b> Adds all of the elements in the specified collection to this collection (optional operation).
void	<b><u>clear()</u></b> Removes all of the elements from this collection (optional operation).
boolean	<b><u>contains(Object o)</u></b> Returns true if this collection contains the specified element.
boolean	<b><u>containsAll(Collection&lt;?&gt; c)</u></b> Returns true if this collection contains all of the elements in the specified collection.
boolean	<b><u>isEmpty()</u></b> Returns true if this collection contains no elements.
abstract <u>Iterator&lt;E&gt;</u>	<b><u>iterator()</u></b> Returns an iterator over the elements contained in this collection.
boolean	<b><u>remove(Object o)</u></b> Removes a single instance of the specified element from this collection, if it is present (optional operation).
boolean	<b><u>removeAll(Collection&lt;?&gt; c)</u></b> Removes all of this collection's elements that are also contained in the specified collection (optional operation).
boolean	<b><u>retainAll(Collection&lt;?&gt; c)</u></b> Retains only the elements in this collection that are contained in the specified collection (optional operation).
abstract int <u>Object[]</u>	<b><u>size()</u></b> Returns the number of elements in this collection.
<u>T[]</u>	<b><u>toArray()</u></b> Returns an array containing all of the elements in this collection.
<u>T[]</u>	<b><u>toArray(T[] a)</u></b> Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.
<u>String</u>	<b><u>toString()</u></b> Returns a string representation of this collection