Statistiques Module POO

4ETI – 2018-2019 1S

1 Pourcentage de validation

Pourcentage des étudiants ayant obtenu plus de 10

	2017-2018	2018-2019	différence
1S	24,48 %	54,47 %	+30
ds 1S	18,18 %	40,65 %	+22,47
tp 1S	39,16 %	100,00 %	+60,84

2 Moyennes

	2017-2018	2018-2019	différence
1S	7,9	11,0	+3,1
ds 1S	7,4	9,5	+2,1
tp 1S	10,0	13,2	+3,2

3 Assiduité en cours/ résultats

61,04 % des étudiants non présents au 5ème cours ne valident pas le module.

4 Résultats des étudiants de x-ème session

4.1 TP

1 validation sur **12** tentatives

4.2 DS

2 validations sur **13** tentatives

4.3 Module

2 validations sur **13** tentatives

10 étudiants n'ont pas participé à la session

23 étudiants n'étaient pas autorisés à s'inscrire à la session

Nom :

Prénom :

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

5 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.

5.1 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.

6 Le modèle

6.1 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
12     private int nbJoursRestants;
13
14     public AbstractProduit(int nbJoursRestants) {
15       this.nbJoursRestants = nbJoursRestants;
16     }
17
18     @Override
19     public double getJoursRestants() {
20       return nbJoursRestants;
21     }
22
23 }
```

```
24 package ds4eti2019_1S.model.produits;
25 public class Chocolat extends AbstractProduit {
26     private String nom;
27
28     public Chocolat(int nbJoursRestants, String nom) {
         super(nbJoursRestants);
         this.nom = nom;
29     }
30
31     public double getPrix() {
32       return 1.5;
33     }
34
35     public String toString() {
36       return nom;
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)***

Soit implémenter Comparable : Produit extends Comparable<Produit> et AbstractProduit définit compareTo(Produit p)

Soit utiliser un comparateur de Produit utilisant getJoursRestants(), definir compare(Produit a, Produit b) dans ce comparateur et le donner en paramètre au constructeur de 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) {
```

```
/* Exemple avec une implémentation de Comparable dans AbstractProduit et :
public int compareTo(Produit p) {
    return this.nbJoursRestants - (int)(p.getJoursRestants());
}
Regardez de votre côté l'autre solution avec Comparator
*/
    Set<Produit> set = new TreeSet<Produit>();
    set.add(new Chocolat(10,"Croustitruc"));
    Produit p = new Chocolat(12,"Noisette");
    set.add(p);
    set.add(new Chocolat(3,"CroustiMachin"));
    System.out.println(set);

/*Sortie attendue :
[CroustiMachin, Croustitruc, Noisette]
TreeSet utilise la méthode compareTo pour construire son arbre binaire
*/
```

```
40  }
```

6.2 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) (*)***

1) Définition des valeurs par défaut pour une exécution

2) classe utilitaire pour accès à ses valeurs

methodes statiques/attributs statique : revoir le cours et les TPs

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

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

cf. dans le TP les identifiants uniques d'Agents
cf. dans les TPs guidés le TP sur les Moutons

```
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) {
       return poids.get(type);
77     }
78     public static double getValeur(PieceType type) {
       return valeurs.get(type);
79     }
80 }
```

```java
package ds4eti2019_1S.model.monnaie;
public interface Piece {
    public PieceType getType();
    public double getValeur();
    public double getPoids();
}
```

```java
package ds4eti2019_1S.model.monnaie;
import ds4eti2019_1S.model.Configuration;
public Abstract class AbstractPiece implements Piece {
    /*attributs  de classe */
    private static int currentId = 1;
    /* Attributs et méthodes d'instance */
    private PieceType type;
    private double masseEnGramme;
    private int id;

    public AbstractPiece(PieceType type, double masse) {
      this.type=type;
      this.masseEnGramme = masse;
      id = currentId;
      currentId++;
    }

    @Override
    public PieceType getType() {return type;}

    @Override
    public double getValeur() {return Configuration.getValeur(getType());}

    @Override
    public double getPoids() {return masseEnGramme;}

    public String toString() {return getValeur()+"€ ("+id+")";}

    public int hashCode() {return id;}

    public boolean equals(Object o) {
      return o instanceof AbstractPiece && ((AbstractPiece)o).id==id;
    }
}
```

```java
package ds4eti2019_1S.model.monnaie;
import ds4eti2019_1S.model.Configuration;
public class PieceEuro extends AbstractPiece{

    public PieceEuro(PieceType type, double masse) {
      super(type, masse);

    }

    public PieceEuro(PieceType type) {
      this(type, Configuration.getPoids(type));
    }

}
package ds4eti2019_1S.model.monnaie;
```

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

6.3 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) (*)***

revoir le cours sur les collections
revoir ce qu'est un type abstrait (abstract ou interface)
Il y a 3 types abstraits à trouver

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

revoir le cours sur les collections
revoir ce qu'est un type concret
Il y a 3 types concrets à trouver

➢ ***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[1] (8 points) (**)***

se servir de l'arbre de décision que vous avez dû réaliser comme demandé en cours

1 pensez à regarder la documentation fournie en fin de sujet

critères à étudier dans notre cas :

fifo/lifo

doublons

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

```
***Test du Monnayeur simple
Plus de place pour les pièces de UnEuro
Etat du monayeur:
[2.0€ (1), X]
[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),
1.0€ (14)]
[]
[]
[0.1€ (16), 0.1€ (17), 0.1€ (18), 0.1€ (19), 0.1€ (20)]

**Test du rendu de monnaie
*Rendre 4€
Je ne rends pas la monnaie!
Etat du monayeur:
[2.0€ (1), X]
[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),
1.0€ (14)]
[]
[]
[0.1€ (16), 0.1€ (17), 0.1€ (18), 0.1€ (19), 0.1€ (20)]

Monnaie: []

*Rendre 1.55€
Je ne rends pas la monnaie!
Etat du monayeur:
[2.0€ (1), X]
[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),
1.0€ (14)]
[]
[]
[0.1€ (16), 0.1€ (17), 0.1€ (18), 0.1€ (19), 0.1€ (20)]

Monnaie: []
*** Test du Monnayeur rembourseur
Etat du monayeur:
[2.0€ (21), X]
[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€
(33), 1.0€ (34), 1.0€ (35)]
[]
[]
[0.1€ (36), 0.1€ (37), 0.1€ (38), 0.1€ (39), 0.1€ (40)]

**Test du rendu de monnaie
*Rendre 4€
Etat du monayeur:
[]
[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€
(33), 1.0€ (34), 1.0€ (35)]
[]
[]
[0.1€ (36), 0.1€ (37), 0.1€ (38), 0.1€ (39), 0.1€ (40)]

Monnaie: [2.0€ (21), X]

*Rendre 1.55€
Etat du monayeur:
[]
[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€
(34), 1.0€ (35)]
[]
[]
[]

Monnaie: [0.1€ (36), 0.1€ (37), 0.1€ (38), 0.1€ (39), 0.1€ (40), 1.0€ (25)]
```

```java
package ds4eti2019_1S.model.monnaie;
import java.util.EnumMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Queue;
import java.util.Set;
import java.util.concurrent.LinkedBlockingQueue;
import ds4eti2019_1S.model.Configuration;

public class Monnayeur {
    private Map<PieceType, Queue<Piece>> pieces = new EnumMap<PieceType,
Queue<Piece>>(PieceType.class);
    private int capacite;

    public Monnayeur(int capacite) {
      setCapacite(capacite);
      //construit la collection de pieces
      for(PieceType type:PieceType.values()) {
            pieces.put(type,new LinkedBlockingQueue<Piece>(this.capacite));
      }
    }

    protected final void setCapacite(int capacite) {
      if(capacite>0) {
            this.capacite = capacite;
      }
    }

    public String toString() {
      String ret ="Etat du monayeur:\n";
      for(PieceType type: pieces.keySet()) {
            Queue<Piece> filePieces = pieces.get(type);
            ret += filePieces + "\n";
      }
      return ret;
    }

    public Set<Piece> getMonnaie(double somme){
      System.out.println("Je ne rends pas la monnaie!");
      return new HashSet<Piece>();
    }

    public boolean add(Piece p) {
      boolean ret = false;
      //la piece a un poids correspondant à son type
      //l'ajout dans la bonne file est possible (capacite ok)
      try {
            if(p.getPoids()==Configuration.getPoids(p.getType()) &&
                  pieces.get(p.getType()).add(p)){
                  ret=true;
            }
      }
      catch(IllegalStateException ise) {
            System.err.println("Plus de place pour les pièces de "+p.getType());
      }
      return ret;
    }

    public Piece remove(PieceType type) {
      return pieces.get(type).poll();
    }
}
```

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

                    if(p!=null) {
                          ret.add(p);
                          somme = somme - p.getValeur();
                    }
                    else {
                          fini = true;
                    }
327             }
328       }
329       return ret;
330     }
331 }
```

- ***Complétez la ligne 310 (1 point) (**)***

- ***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) (**)***

Cf. mise en place d'abstractions : interface, classe Abstraites, comme ce qui est fait juste avant dans le DS pour les Pieces, les Produits, et ce qui est fait dans les TPs.

Possibilités à détailler et justifier.

6.4 Le distributeur (22 points)(**/***)

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

```
package ds4eti2019_1S.model;
//imports non donnés dans le sujet par soucis de gain de place
public class Distributeur {

    private Monnayeur monnayeur;
    private List<Stack<Produit>> produits;
    private int capacite;
    private int nbFiles;
    private double sommeCourante=0;

    public Distributeur(int capacite, int nbFiles, Monnayeur m) {
      this.capacite = capacite;
      this.nbFiles = nbFiles;
      this.monnayeur = m;
      produits = new ArrayList<Stack<Produit>>();
      for(int i=0;i<nbFiles;i++) {
            produits.add(new Stack<Produit>());
      }
    }

    public boolean add(Produit p) {
      boolean ret = false;
      boolean fini = false;
      int i=0;
      while(i<nbFiles && !fini) {
            if(produits.get(i).size()  < capacite ) {
                  produits.get(i).push (p);
                  fini =  true ;
            }
            else {
                  i++ ;
            }
      }
      return ret;
    }

    public final Retour getRetour(int num, Set<Piece> pieces) {
      Produit produit = null;
      Set<Piece> monnaie = new HashSet<Piece>();

      Set<Piece> refusees = ajoutePieces(pieces);
      produit = getProduit(num, sommeCourante);

      System.out.println(sommeCourante);
//ajout des piece refusées à monnaie
      monnaie.addAll(refusees);
    //restitution des pièces par le monnayeur
      if(produit == null) {//produit invalide : remboursement de ce qui a été versé
      monnaie.addAll(monnayeur.getMonnaie(sommeCourante));
```

```
        }
        else {//Produit non null : restitution des sommes trop perçues
          monnaie = monnayeur.getMonnaie(sommeCourante-produit.getPrix());
        }
        sommeCourante = 0;
        return new Retour(monnaie, produit);
      }

      private Produit getProduit(int num, double somme) {
        Produit p = null;
        if(num<produits.size() &&
                    !produits.get(num).isEmpty()) {
              Produit candidat = produits.get(num).peek();
              if(somme>=candidat.getPrix()) {
                    p = produits.get(num).pop();
              }
        }
        return p;
      }

      private Set<Piece> ajoutePieces(Set<Piece> pieces) {
        Set<Piece> refusees = new HashSet<Piece>();
      //ajout des pieces valide au monnayeur et maj de sommecourante ;
        for(Piece p:pieces) {
              if(monnayeur.add(p)) {
                sommeCourante= sommeCourante + p.getValeur();
              }
              else {//piece non valide à ajouter aux pièces refusées
                refusees.add(p);
              }
        }
        return refusees;
      }

      public String toString() {
        String ret = "***Distributeur:\n";
        int i=1;
        for(Stack<Produit>file:produits) {
              ret+="\t* file "+ i +":" +file+"\n";
              i++;
        }
        ret += monnayeur;
        return ret;
      }
      public List<ArrayList<String>> getInfos() {
```

```
        List<ArrayList<String>> infos = new ArrayList<ArrayList<String>>();
        for(Stack<Produit> s:produits) {
              ArrayList<String> al = new ArrayList<String>();
              if(!s.isEmpty())al.add(s.peek().toString());
        }
        return infos;
      }
}

public class Retour {
      private Set<Piece> pieces;
      private Produit produit;

      public Retour(Set<Piece> pieces,Produit produit) {
        this.pieces = pieces;
        this.produit = produit;
      }

      public Set<Piece> getPieces() {return pieces;}
      public Produit getProduit() {return produit;}
}

package ds4eti2019_1S.tests;
//import non donnés dans le sujet par soucis de gain de place
public class TestDistributeur {
      public static void main(String[] args) {
        Monnayeur m = new MonnayeurRembourseur(5);
        Distributeur d = new Distributeur(4,3,m);//trois files de 4 places
        ajouteProduits(d);
        System.out.println(d);
        Set<Piece> monnaie = new HashSet<Piece>();
        createMonnaie(monnaie);
        d.getRetour(5, monnaie);//KO
        System.out.println(d);
        d.getRetour(2, monnaie);//OK
        System.out.println(d);
      }

      private static void createMonnaie(Set<Piece> monnaie) {
        monnaie.add(new PieceEuro(PieceType.UnEuro));//OK
        monnaie.add(new PieceEuro(PieceType.DixCentimes));//OK
        monnaie.add(new PieceEuro(PieceType.VingtCentimes,8.2));//KO
        monnaie.add(new PieceHorsEuro(PieceType.DeuxEuro,7.2));//KO
        monnaie.add(new PieceHorsEuro(PieceType.DeuxEuro,8.5));//OK
      }

      private static void ajouteProduits(Distributeur d) {
        d.add(new Chocolat(10,"Croustitruc"));
        d.add(new Chocolat(5,"Croustitruc"));
        d.add(new Chocolat(3,"CroustiMachin"));
        d.add(new Chocolat(6,"Noir"));
        d.add(new Chocolat(6,"Blanc"));
        d.add(new Chocolat(6,"Lait"));
        d.add(new Chocolat(6,"Noisette"));
        d.add(new Chocolat(10,"Croustitruc"));
        d.add(new Chocolat(5,"Croustitruc"));
        d.add(new Chocolat(3,"CroustiMachin"));
        d.add(new Chocolat(6,"Noir"));
        d.add(new Chocolat(6,"Blanc"));
        d.add(new Chocolat(6,"Lait"));
      }
}
```

Sortie console :

```
***Distributeur:
    * file 1:[Croustitruc, Croustitruc, CroustiMachin, Noir]
    * file 2:[Blanc, Lait, Noisette, Croustitruc]
    * file 3:[Croustitruc, CroustiMachin, Noir, Blanc]
Etat du monayeur:
[]
[]
[]
[]
[]

3.1
***Distributeur:
    * file 1:[Croustitruc, Croustitruc, CroustiMachin, Noir]
    * file 2:[Blanc, Lait, Noisette, Croustitruc]
    * file 3:[Croustitruc, CroustiMachin, Noir, Blanc]
Etat du monayeur:
[]
[]
[]
[]
[]

3.1
***Distributeur:
    * file 1:[Croustitruc, Croustitruc, CroustiMachin, Noir]
    * file 2:[Blanc, Lait, Noisette]
    * file 3:[Croustitruc, CroustiMachin, Noir, Blanc]
Etat du monayeur:
[X]
[]
[]
[]
[]
```

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

oui

justifier l'état du distributeur et celui du monayeur, expliquer les affichages notamment la X dans le monnayeur

- ***expliquez le choix de la collection `List<Stack<Produit>> produits` pour représenter l'organisation des produits dans le distributeur. (2 points) (*)***

même remarques que précédemment : quels sont les critères, les deux collections répondent-elles aux besoins ?

- ***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) (**)***

getRetour

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

reprendre le cours sur l'encapsulation, les getters, les setters

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

revoir principe de template méthode du cours et du TP

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

indice : comment récupérerait-on la monnaie et le produit si cette classe n'existait pas ?

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

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

6.5 Interface graphique (20 points) (*/**)

Prenez connaissance du code suivant

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

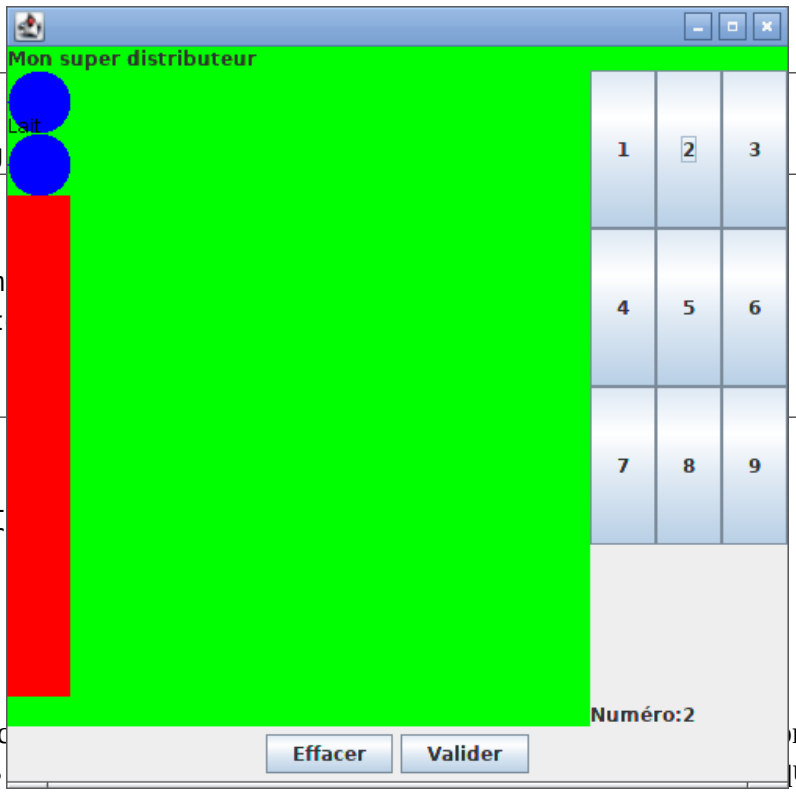
```
567 public class DistributeurIHM extends JFrame{
568
569     private static final long serialVersionUID = 1L;
570     private Clavier clavier;
571     private ProduitsIHM produitsIHM;
572     private Distributeur distributeur;
573
574     public DistributeurIHM(Distributeur distributeur) {
575  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

```
            @Override
            public void actionPerformed(ActionEvent e) {
                  clavier.efface();
            }
```

```
595 );
596
597       JButton btValider = new J
```

```
      @Override
      public void actionPerform
            distributeur.getRet
      }
```

```
598       boutons.add(btReset);
599       boutons.add(btValider);
600       fond.add(boutons, BorderL
601       fond.add(produitsIHM, Bor
602       setContentPane(fond);
603       pack();
604     }
605 }
```



---

18 2 Il fallait ici s'apercevoir que l'interfac révoir
19 une IHM permettant d'introduire des ue de
20 détecter le problème ; bonus si solution proposée.

22

```
606 public class ProduitsIHM extends JPanel{
607     private Distributeur distributeur;
608
609     public ProduitsIHM(Distributeur d) {
610       distributeur = d;
611       setPreferredSize(new Dimension(400,400));
612     }
613     public void paintComponent(Graphics g) {
614       int x=0;
615       int y=0;
616       for(ArrayList<String> liste: distributeur.getInfos()) {
            for(String s:liste) {
                  if(s.equals("")) {
                        g.setColor(Color.red);
                        g.fillRect(x, y, 40, 40);
                  }
                  else {
                        g.setColor(Color.blue);
                        g.fillOval(x, y, 40, 40);
                        g.setColor(Color.BLACK);
                        g.drawString(s, x, y);
                  }
                  x+=40;
            }
            x=0;
            y+=40;
617       }
618     }
619 }
```

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

Donner le nom des composants :
Jxxxxx, Clavier (extends JPanel),
etc.

- ***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 :

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

7 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.

revoir le cours sur les collections.

L'idée de base est de regarder s'il est possible que Monnayeur et Distributeur héritent d'AbstractCollection ou implémentent une interface de collection.

Il faut étudier les méthodes qui doivent être implémentées pour savoir si cela est possible.

Pour un des deux composants, ce sera envisageable.

Pour l'autre, il faudra bien choisir ce qui constitue la collection et cela risque d'être compliqué à mettre en œuvre.

A vous de trouver pourquoi.

8 Documentation

8.1 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
`void`	`clear()` Atomically removes all of the elements from this queue.
`boolean`	`contains(Object o)` Returns `true` if this queue contains the specified element.
`Iterator<E>`	`iterator()` Returns an iterator over the elements in this queue in proper sequence.
`boolean`	`offer(E e)` 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 `true` upon success and `false` if this queue is full.
`boolean`	`offer(E e, long timeout, TimeUnit unit)` Inserts the specified element at the tail of this queue, waiting if

	necessary up to the specified wait time for space to become available.
E	peek() Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
E	poll() Retrieves and removes the head of this queue, or returns null if this queue is empty.
void	put(E e) Inserts the specified element at the tail of this queue, waiting if necessary for space to become available.
int	remainingCapacity() Returns the number of additional elements that this queue can ideally (in the absence of memory or resource constraints) accept without blocking.
boolean	remove(Object o) Removes a single instance of the specified element from this queue, if it is present.
int	size() Returns the number of elements in this queue.

8.2 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.

8.3 Méthodes de Collection

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