Université de Carthage
École Supérieure de la Statistique et de l'Analyse de l'Information

Session Principale
Examen java 1
Mme Aïcha El Golli Jabbes

Année Universitaire **2021-2022**
$2^{ème}$ année
Durée : 1h30

Cet examen contient 5 pages. *Veuillez rendre une copie propre et claire*

Exercice 1 (4 Points)

Soient les classes suivantes. Qu'affiche le code suivant ?

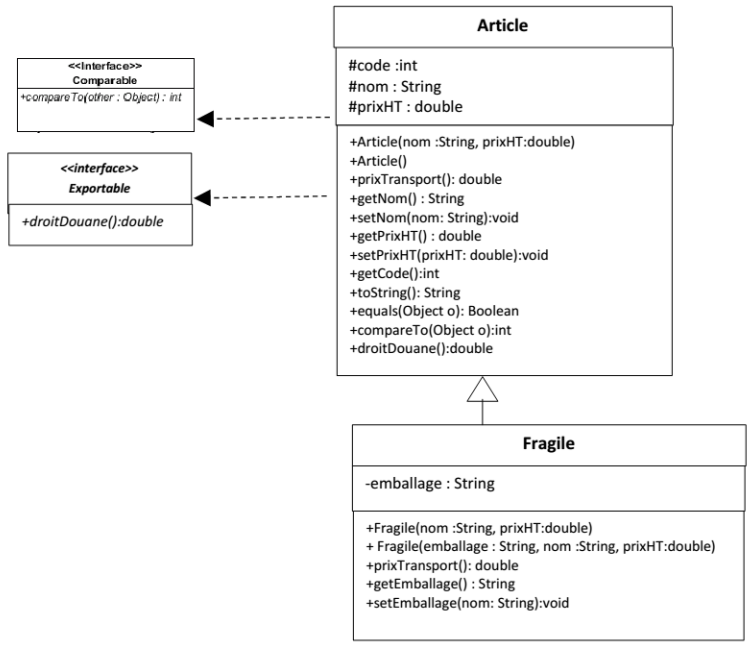
```
    class Mere {
 protected int x;
 Mere(int k) {x=k;
 System.out.println("constructeur mère");}
 void ajoute(int a) { x+=a; }
 public void moi() { System.out.println(this); }
 public String toString() {return "x ="+x;}
}
 class Enfant1 extends Mere {
 protected String y;
 Enfant1 (int k, String l) { super(k); y=l;
 System.out.println("constructeur enfant1");}
 void ajoute(int a) { x= x+2*a;}
 public String toString() {return super.toString()+" y= "+y;}
}
 class Enfant2 extends Enfant1 {
 protected int z ;
 Enfant2 (int k, String l, int m) { super(k, l); z= m;
 System.out.println("constructeur enfant2");}
 void ajoute(int a) { x= x+3*a;}
 public String toString() {return super.toString()+" z= "+z;}
}
public class Essai{
 public static void main (String args[]) {
 int a =2;
 Mere p = new Mere(2);
 p.moi();
 p.ajoute(a);
 System.out.println(p );
 Enfant1 e1 = new Enfant1(2, "Essai");
 e1.moi();
 e1.ajoute(a);
 System.out.println( e1 );
 e1 = new Enfant2(3, "POO", 5);
 e1.moi();
 e1.ajoute(a);
 System.out.println( e1 );
 } }
```

Solution: constructeur mère (0.25pt)
x =2 (0.25pt)
x =4 (0.25pt)
constructeur mère (0.5pt)
constructeur enfant1 (0.25pt)
x =2 y= Essai (0.25pt)
x =6 y= Essai (0.25pt)
constructeur mère (0.5pt)
constructeur enfant1 (0.5pt)
constructeur enfant2 (0.25pt)
x =3 y= POO z= 5 (0.25pt)

x =9 y= POO z= 5 (0.5pt)

Exercice 2 (10 Points)

Nous allons créer deux classes, l'une représentant les articles, nommée *Article* et l'autre représentant les articles fragiles nommée *Fragile*.
Soit le schémas UML suivant :



1. (7 points) Écrire la classe *Article* avec les trois attributs d'instances ***protégés*** suivants :
   nom : String ; le nom de l'article.
   code : int, le code de l'article, est un code unique et automatique.
   prixHT : double, le prix hors taxe de l'article.
   (a) La classe Article doit disposer d'un constructeur publique complet, où le code d'un article est unique.
   Article(String nom, double prixHT)
   (b) La classe *Article* doit contenir des accesseurs (les getters) publiques pour les trois attributs. La classe dispose que de deux mutateurs (les setter). Le mutateur du nom de l'article ne doit pas accepter les valeurs nulls et celui du prixHT ne doit accepter que des valeurs positives.
   (c) Écrire une méthode publique double prixTransport() qui permet de calculer le prix du transport, à savoir 5% du Prix Hors Taxes de l'article. Cette méthode retourne le prix calculé.
   (d) Redéfinir la méthode toString() donnant une représentation d'un objet de la classe *Article* comme suit :

   l'article <son code> <son nom>

   (e) Redéfinir la méthode equals, deux articles sont les mêmes si le nom et le prixHT sont les mêmes.
   (f) La classe article doit aussi implémenter l'interface Exportable et calculer le droit de douane. En effet, seuls les articles d'une valeurs supérieure ou égale à 150dt sont soumis aux droits de douane qui est fixé à 10% du prix de l'article. Cette méthode retourne le droit de douane (0 dans le cas du prixHT<150dt).

```
public interface Exportable
{
```

```
    double droitDouane();
}
```

(g) La classe article doit aussi implémenter l'interface Comparable suivante :

```
public interface Comparable
{
    int compareTo(Object other);
}
```

l'unique méthode compareTo, renvoie un nombre entier qui représente le résultat de la comparaison. La mise en oeuvre de cette méthode doit renvoyer les valeurs suivantes :
— 0 si cette instance et other sont égales
— > 0 si cette instance est supérieure à other
— < 0 si cette instance est inférieure à other
La méthode renvoie un nombre entier basé sur la comparaison des prix des articles. Étant donnée que le prix d'un article est un double **utilisez la méthode statique Double.compare (arg1, arg2)** qui renvoie une valeur négative si le premier argument est inférieur au deuxième argument, 0 si elles sont égales et une valeur positive sinon.

2. (3 points) Écrire la classe des articles fragiles *Fragile* **héritant** de *Article* et ayant un attribut privé : ***emballage***, de type String.
   (a) Écrire le constructeur complet de Fragile(String emballage, String nom, double prixHT) en utilisant un appel à super(...).
   (b) Écrire le constructeur de Fragile(String nom, double prixHT) qui appelle constructeur complet et qui initialise emballage à une chaîne vide.
   (c) Le prix de transport d'un article fragile est deux fois le prix de transport normal d'un Article. Redéfinir la méthode prixTransport() de la classe Fragile.
   (d) Écrire le getter ainsi que le setter de l'attribut emballage qui ne doit pas accepter les valeurs nulls

Solution:

```
//7pt
public class Article implements Exportable, Comparable {//0,5pt
    protected int code;//0,25pt
    protected String nom;//0,25pt
    protected double prixHT;//0,25pt

    private static int comp;//0,5pt

    public Article(String nom, double prixHT) {//1pt
        this.code = ++comp;//0,5
        setNom( nom);//0,25
        setPrixHT(prixHT); //0,25
                }

    public String getNom() {//0,25pt
        return nom; }
    public void setNom(String nom) {//0,5pt
        if(nom!=null) this.nom = nom;}

    public double getPrixHT() {//0,25pt
        return prixHT;  }

    public void setPrixHT(double prixHT) {//0,5pt
        if(prixHT>=0) this.prixHT = prixHT; }
```

```
    public int getCode() {//0,25pt
        return code;}
    public double prixTransport() {//0,5pt
        return prixHT * 0.05; // 5% du Prix Hors Taxes de l'article
    }
    @Override
    public String toString() {//0,5pt
        return "l'article "+this.code + " " + this.nom;
    }
    @Override
    public boolean equals(Object o) {//0,5pt
        if (this==o)return true;
    if (!(o instanceof Article ))  return false;
    Article p =(Article ) o;
       return ((nom.equals(p.nom)) && (prixHT==p.prixHT) );
    }
    @Override
    public int compareTo(Object other) {//0,5pt
        Article p = (Article)other;
        return Double.compare(prixHT, p.prixHT);
    }
    @Override
    public double droitDouane() {//0,5pt
        if(prixHT>=150)
            return prixHT/10;
        else
            return 0;
    }}
//3 points
class Fragile extends Article{ //0,5pt
    private String emballage;//0,25pt

    public Fragile(String emballage, String nom, double prixHT) {
        super(nom, prixHT);
        setEmballage ( emballage);
    }//0,5pt
    public Fragile(String nom, double prixHT) {
        this("",nom, prixHT);
    }//0,5pt
    public String getEmballage() {//0,25pt
        return emballage;}

    public void setEmballage(String emballage) {//0,5pt
        if(emballage!=null) this.emballage = emballage;
    }
    public double prixTransport() {//0,5pt
        return 2 * super.prixTransport();
}}
```

Questions de réflexion (6 Points)

1. (½ point) Que signifie le mot clef "static" associé à un attribut ?
   (a) Que la valeur de cet attribut est constante
   (b) Que cet attribut sera toujours passé par valeur
   (c) Que cet attribut a une valeur unique pour toutes les instances de la classe

2. (½ point) Pour le mot clef "abstract", quelles assertions sont fausses ?
   (a) Une classe abstraite ne peut être instanciée
   (b) Une méthode abstraite n'a pas d'implémentation
   (c) Une classe abstraite n'a pas forcément de classe fille
   (d) Une classe abstraite doit contenir au moins une méthode abstraite
3. (1½ points) Pour la classe ArrayList, quelles assertions sont justes ? :
   (a) la classe ArrayList implémente les tableaux dynamiques
   (b) la classe ArrayList implémente les listes chaînées
   (c) la classe ArrayList appartient au package util de java
   (d) la classe ArrayList a une unique méthode add
   (e) sa méthode public boolean add(E e), permet l'ajout d'une référence qui pointe le null
4. (3½ points) indiquer si chaque affirmation est correcte ou non :
   (a) (½ point) En Java, toutes les classes possèdent une méthode equals
   (b) (½ point) Une méthode marquée comme *private* est accessible depuis toutes les autres classes du même paquetage.
   (c) (½ point) Il est possible d'instancier une classe abstraite.
   (d) (½ point) Il est possible de définir un constructeur dans une classe abstraite.
   (e) (½ point) Une variable d'instance déclarée *protected* n'est accessible directement que par cette instance.
   (f) (½ point) Une classe ne peut implémenter qu'une seule interface.
   (g) (½ point) Dans un constructeur d'une classe, il est possible d'accéder à un autre constructeur de la même classe à l'aide de l'appel du *this(...)*

Solution:
1. (c) 0,5pt
2. (c) et (d) 0,25pt+0,25pt
3. (a), (c) et (e), chacune sur 0,5pt si erreur (en plus) -0,25pt
4.
   (a) vrai
   (b) faux
   (c) faux
   (d) vrai
   (e) faux
   (f) faux
   (g) vrai

Bon travail