Contrôle terminal info1 / Semestre 2 (période 3)

R2.01 : Développement orientés objets

Nom du responsable :	**BORNE Isabelle, FLEURQUIN Régis**
Date du contrôle :	**29/03/22**
Durée du contrôle :	**1h30**
Nombre total de pages :	**7 pages**
Impression :	**Recto**
Documents autorisés :	**Cours, TD, TP**
Calculatrice autorisée :	**Non**
Réponses :	**2 Copies**

Bien lire le sujet

La correction tiendra compte de la lisibilité du code, du respect des consignes de nommage en Java, du respect de l'énoncé et de la bonne indentation du code.

2 exercices à faire sur 2 copies séparées

Exercice 1 (Copie 1, 45 minutes, 10 points)

Remarques préliminaires :

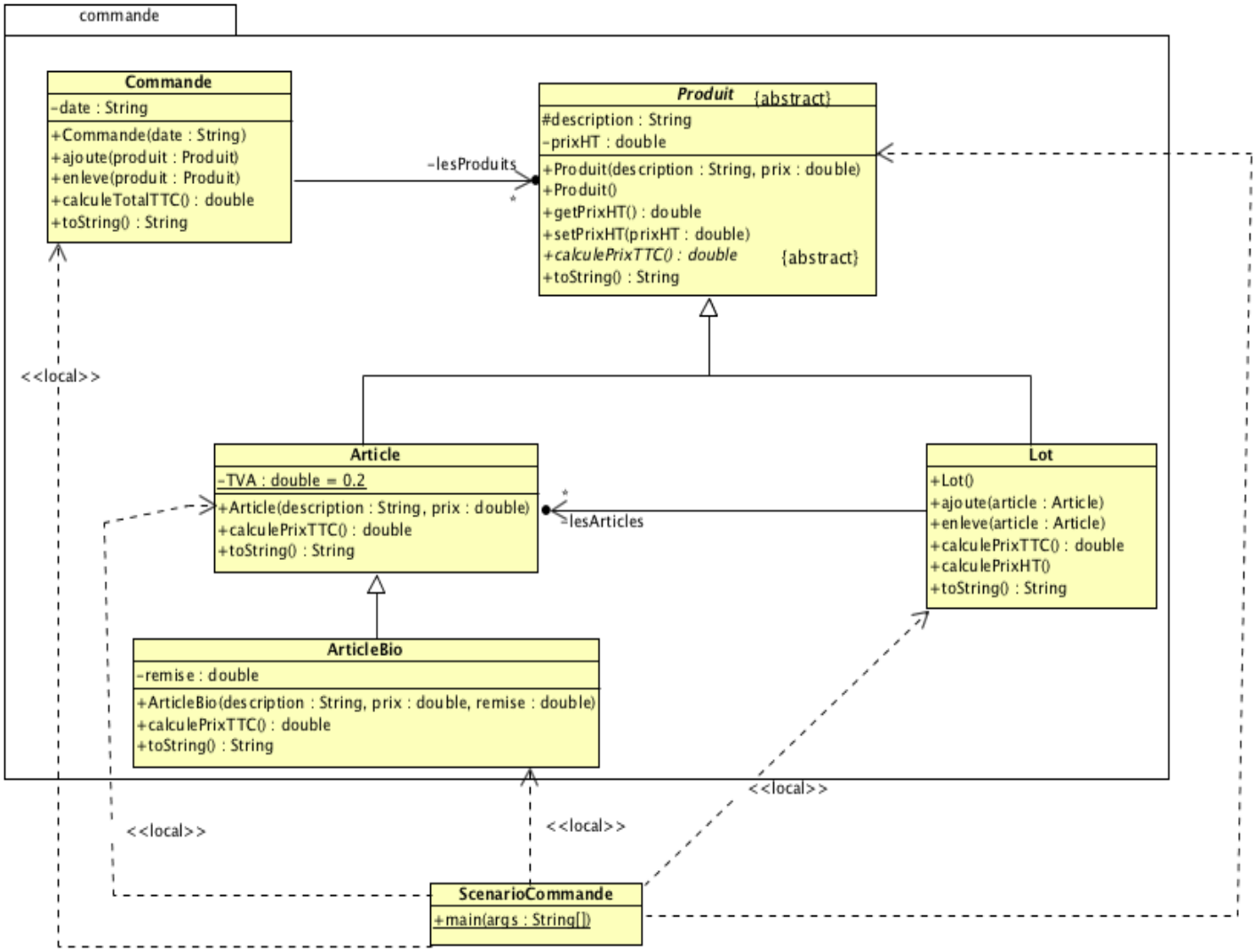
- Ne pas écrire la Javadoc.
- Le test des paramètres d'entrée des méthodes est exigé, ce test doit donner un message succinct mais explicite et indiquer le nom de la méthode.
- Toutes les classes Java sont de visibilité « public ».
- Les listes sont des ArrayList.
- Ne pas oublier de préciser le « this » partout où il est nécessaire (attribut, envoi de message).

Il s'agit de réaliser en Java la simulation d'une commande de produits où l'on calcule les prixTTC des produits.

Un objet Commande possède une liste de produits qui peuvent être des articles, des articles Bio et des lots.

Un lot modélise un ensemble d'articles.

Le diagramme de classes de l'application est le suivant :



On étudiera avec attention ce diagramme de classes. On remarquera qu'il y a un package.

Note : HT signifie « Hors Taxe » et TTC signifie « Toutes Taxes Comprises ». Le pourcentage de TVA est à calculer sur un prix HT et s'ajoute au prix HT pour obtenir le prix TTC.

Classe `Commande`

Le constructeur de Commande permet d'initialiser la date de la commande.

Les méthodes **`ajoute(Produit produit)`** et **`enleve(Produit produit)`** permettent respectivement d'ajouter ou d'enlever un produit de la liste.

La méthode **`double calculeTotalTTC()`** retourne le montant TTC de la commande, c'est-à-dire la somme des prix TTC des produits qui la composent.

La méthode **`String toString()`** retourne une chaine de caractère avec la date, les caractéristiques des différents produits et le montant total de la commande.

Classe `Produit`

L'attribut **`description`** est une chaîne de caractères contenant le nom d'un produit.

Le constructeur avec paramètre est chargé d'initialiser les attributs d'instance.

La méthode **`double calculePrixTTC()`** est abstraite.

La méthode **`String toString()`** retourne une chaine de caractère contenant la description et le prix HT du produit.

Classe `Article`

Cette sous-classe représente un simple Article qui possède une constante de classe qui définit le taux de la TVA (20%).

- La méthode **`double calculePrixTTC()`** retourne le prix TTC .
- La méthode **`String toString()`** retourne une version imprimable de l'article avec sa description, son prix HT et le taux de TVA.

Classe `ArticleBio`

Comme on souhaite favoriser l'achat de produits bio, une remise est accordée pour les articles bio.

Cette classe représente donc un article qui possède en plus une remise.

- La méthode **`double calculePrixTTC()`** retourne le prix (TTC) avec prise en compte de la remise.
- La méthode **`String toString()`** retourne une version imprimable de l'article avec tous ses attributs

Classe `Lot`

Cette classe modélise représente un ensemble de produits. La liste sera représentée par un ArrayList. A la création d'un lot sa liste est vide.

- Le constructeur sans paramètre crée la liste vide, initialise le prixHT à 0 et la description avec la chaîne "Lot : ".
- La méthode **`double calculePrixTTC()`** retourne le prix TTC du lot.
- La méthode **`double calculePrixHT()`** retourne le prix hors taxe du lot.
- La méthode **`String toString()`** retourne une version imprimable du lot contenant sa description et toutes les versions imprimables des éléments de sa liste de produits.

2- Questions

1. Ecrire le code complet de la classe **`Produit`**.
2. Ecrire les entêtes complets et les constructeurs des classes **`Article, ArticleBio`** et **`Lot`**.
3. Ecrire le code des méthodes **`public double calculePrixTTC()`** des classes **`Article, ArticleBio`** et **`Lot`**.
4. Ecrire la méthode **`public double calculeTotalTTC()`** de la classe **`Commande.`**
5. Ecrire la méthode **`public String toString()`** de la classe **`Commande`** telle que la chaîne renvoyée commence par « Commande du (la date) », puis contienne la liste des produits et se termine par le montant total TTC de la commande.

Exercice 2 (Copie 2, 45 minutes, 10 points)

On considère le code Java donné en annexe 1 qui compile sans erreurs.

Question 1 (4 points, 20 minutes)

Donnez une cartographie de ce code avec un diagramme de classes UML.

Question 2 (2 points, 5 minutes)

Qu'affiche à l'écran l'exécution du `main()` de la classe `Scolarite` ?

Question 3 (4 points, 20 minutes)

Donnez le diagramme de séquence complet correspondant à l'exécution du `main()` de la classe `Scolarite`. On ne représentera pas dans ce diagramme les instances de la classe `ArrayList` et `Iterator` (et donc aucun des messages allant vers des instances de ces deux classes).

Question 4 (bonus, pour les meilleurs et les plus rapides uniquement)

Ce code fait usage du patron de conception « `Factory` » qui offre certains avantages.

a) Que faut-il modifier à ce code pour introduire des étudiants titulaires du baccalauréat général avec le profil Math et PC comme spécialités de terminale 1 et 2 et NSI en spécialité de première ? On suppose que le codage de ce profil se fait avec la chaîne « General2 »

b) Même question si l'on souhaite introduire des bacheliers professionnels qui n'ont aucune spécialité en terminale et dont le code de profil serait « Pro1 ». Indiquez seulement ce qu'il faudrait modifier à l'application actuelle sans en donner le code.

c) En déduire l'intérêt de la classe `EtudiantFactory` et de la classe `Etudiant` et plus généralement du patron « `Factory` ».

Annexe 1

```
import inscription.Etudiant;
public class Scolarite {

	public static void main(String[] args){
		Secretaire sec;
		Etudiant e;
		int numEtud1, numEtud2;

		sec=new Secretaire();

		numEtud1=sec.inscrireEtudiant("Kent", "Clark", "STi2D1");
		numEtud2=sec.inscrireEtudiant("Curry", "Arthur", "general1");

		System.out.println(sec.nombreInscrits());

		e=sec.getEtudiant(numEtud2);
		e.getProfil();
	}
}
```

```
import inscription.Etudiant;
import inscription.EtudiantFactory;
import java.util.ArrayList;
import java.util.Iterator;
public class Secretaire {
      private ArrayList<Etudiant> promotion;
      private EtudiantFactory ef;

    public Secretaire(){
            this.promotion=new ArrayList<Etudiant>();
            this.ef=new EtudiantFactory();
    }

    public int inscrireEtudiant(String nom, String prenom, String profil){
            int ret=-1;
            Etudiant e;
            e=this.ef.getEtudiant(nom,prenom,profil);
            if (e!=null){
                  this.promotion.add(e);
                  ret=e.getNumero();
            }
            return ret;
    }

    public int nombreInscrits(){
            return this.promotion.size();
    }

    public Etudiant getEtudiant(int num){
            Etudiant ret=null;
            boolean found=false;
            Iterator<Etudiant> it = this.promotion.iterator();
            Etudiant etud;
            while((it.hasNext()) && (!found)) {
                  etud = it.next();
                  if(etud.getNumero()==num) {
                        found = true;
                        ret = etud;
                  }
            }
            return ret;
    }
}
```

```
package inscription;
public class EtudiantFactory {
      private static int indice=1;

    public Etudiant getEtudiant(String nom, String prenom, String profil){
            Etudiant ret=null;
            if((profil!=null)&&(nom!=null)&&(prenom!=null)){
                  if(profil.equalsIgnoreCase("STI2D1")){
                        ret=new Sti2d(nom, prenom, this.indice, "SIN");
                        this.indice=this.indice+1;
                  }
                  if(profil.equalsIgnoreCase("GENERAL1")){
                        ret=new General(nom,prenom,this.indice, "Math", "NSI", "PC");
                        this.indice=this.indice+1;
                  }
            }
            return ret;
    }
}
```

```
package inscription;
public abstract class Etudiant {
    private String nom;
    private String prenom;
    private int numero;
    private String bac;

    public Etudiant(String n, String p, int num, String b){
            this.nom=n;
            this.prenom=p;
            this.numero=num;
            this.bac=b;
    }
    public int getNumero(){
            return this.numero;
    }
    public void getProfil(){
            System.out.println("Nom : "+this.nom);
            System.out.println("Prénom : "+this.prenom);
            System.out.println("Numéro : "+this.numero);
            System.out.println("Bac : "+this.bac);
            this.getSpecialite();
    }

    protected abstract void getSpecialite();
}
```

```
package inscription;
class General extends Etudiant{
    private String specialiteTerm1;
    private String specialiteTerm2;
    private String specialitePrem;

    public General(String n, String p, int num, String s1, String s2, String s3){
      super(n,p,num, "Bac General");
      this.specialiteTerm1=s1;
      this.specialiteTerm2=s2;
      this.specialitePrem=s3;
    }

    protected void getSpecialite(){
      System.out.println("Spécialité de terminale 1 : "+this.specialiteTerm1);
      System.out.println("Spécialité de terminale 2 : "+this.specialiteTerm2);
      System.out.println("Spécialité de première    : "+this.specialitePrem);
    }
}
```

```
package inscription;
class Sti2d extends Etudiant{
    private String specialiteTerm;

    public Sti2d(String n, String p, int num, String s){
      super(n,p,num, "Bac STI2D");
      this.specialiteTerm=s;
    }

    protected void getSpecialite(){
      System.out.println("Spécialité de terminale : "+this.specialiteTerm);
    }
}
```