UE Programmation Orientée Objet

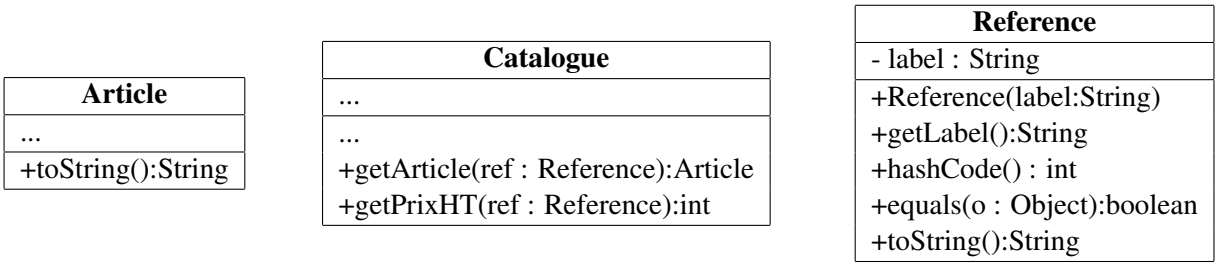
Examen mai 2008

3 heures
documents de cours, TD et TP autorisés
livres, portables et calculatrices interdits

▷ Les 3 exercices sont totalement indépendants et l'ordre dans lequel ils apparaissent ne préjuge pas de leur difficulté réelle ou supposée.

Exercice 1 : Commandes

Les classes de cet exercice appartiendront au paquetage `e-commerce`.
On s'intéresse aux commandes d'une boutique en ligne.
On suppose qu'existe une classe `Catalogue` qui permet de gérer l'ensemble des articles, de type `Article`, proposés par cette boutique et leur prix. Un article est identifié par sa référence, unique, du type `Reference`.
Les classes `Article`, `Catalogue` et `Reference` sont définies, au moins, ainsi :



Les méthodes `getArticle` et `getPrixHT` de `Commande` lèvent une exception `ReferenceInconnueException` si la référence n'existe pas dans le catalogue.

Une commande est définie par :

- le numéro du client qui a passé la commande (une chaîne de caractères pour simplifier),
- les articles commandés et leurs quantités. Chaque article est identifié par sa référence de type `Reference`.
- le statut de la commande, 3 statuts différents sont possibles : *en cours* de préparation, *expédiée* et *livrée*.

Q 1 . Donnez un code Java pour la classe `Reference`.

Q 2 . Faites une proposition pour gérer le statut des commandes et donnez le code Java associé à votre proposition.

Q 3 . Complétez le code ci-dessous (ne recopiez pas les commentaires sur votre copie !).

Les classes d'exception `CommandeExpedieeException` et `ReferenceInconnueException` disposent d'un contructeur prenant en paramètre un message qui est affiché dans la trace de l'exception si celle-ci se produit.

```
COMPLETER

public class Commande {
   COMPLETER
   /** crée une commande, initialement sans article
   * @param cata le catalogue dont dépend la commande
```

```
    * @param numClient le numéro du client qui passe la commande
    */
    public Commande(Catalogue cata, String numClient) {
       COMPLETER
    }
    /** modifie pour cette commande de qte la quantité d'articles dont
    * la référence est reference.
    * Cette opération n'est possible que quand la commande a
    *     pour statut "en cours".
    * Si l'article n'était pas encore présent dans la commande, il est ajouté.
    * Si la quantité dans la commande devient négative, l'article est
    *     supprimé de la commande.
    * @param reference la référence de l'article concerné
    * @param qte la quantité d'articles à prendre en compte
    *     (ajout si >0, retrait si <0)
    * @exception CommandeExpedieeException si la commande n'est pas "en cours".
    * @exception ReferenceInconnueException si la référence n'existe pas dans
    *     le catalogue.
    */
    COMPLETER méthode modifieQuantite
    /** fournit le prix hors-taxe total de la commande
    * @return le prix hors-taxe total de la commande
    */
    COMPLETER méthode totalHT
    /** La commande est expédiée au client.  Son statut passe à "expédiée",
    * on ne peut ensuite plus modifier les articles et quantités commandés.
    */
    COMPLETER méthode expediee
    /** la commande a été livrée au client .  Son statut passe à "livrée".
    * on ne peut plus modifier les articles et quantités commandés.
    */
    COMPLETER méthode livree
    /** affiche la commande :  le client, puis les articles à raison
    * d'un article et sa quantité commandée par ligne.
    * En fin de commande le prix total hors-taxe est affiché,
    * ainsi que le statut de la commande.
    */
    COMPLETER méthode affiche
}
```

Q 4 . Il manque à la proposition précédente un identifiant ("numéro", mais pas forcément un nombre) de commande, indispensable pour différencier les commandes. Evidemment cet identifiant doit être unique.

Faites une proposition pour générer un identifiant de commande et garantir son unicité (pour une exécution dans une JVM donnée). Vous donnerez le code correspondant à votre proposition, vous indiquerez précisément où il s'insère dans la classe `Commande` sans réécrire toute la classe.

Exercice 2 : Parc d'Attractions

Les classes de cet exercice appartiendront au paquetage `parc`.

Un parc d'attractions est caractérisé par une liste d'attractions proposées aux visiteurs. L'accès à ces attractions peut être contraint au respect de certaines conditions, comme l'âge ou la taille.

Chaque attraction a un coût fixe en euro. Pour participer à une attraction il faut s'acquitter de ce coût ou utiliser un *ticket* qui a pu être acheté précédemment (les tickets s'achètent par carnet). Le coût d'un ticket est constant et fixé par le parc dans la constante `PRIX_TICKET`.

Les moyens de paiement sont définis par le type énuméré suivant :

```
package parc;
public enum MoyenPaiement {
   euro, ticket;
}
```

Les visiteurs du parc sont modélisés par une classe `Personne` dont les instances sont caractérisées par un âge (un nombre entier d'années), une taille (un nombre entier de centimètres) et un nom (une chaîne de caractères).

Q 1 . Donnez un diagramme UML pour une classe `Personne`.

Les conditions d'accès à une activité sont définies via l'interface :

```
package parc;
public interface ConditionAcces {
   public boolean accesPossible(Personne p);
   public String getDescription();
}
```

La méthode `accesPossible` ne renvoie la valeur *true* que si la personne p respecte la condition (et aura donc accès à l'attraction) et la méthode `getDescription` fournit une information textuelle sur la condition d'accès.

Q 2 . Donnez le code d'une classe `ConditionAge` qui permet de définir une condition d'accès qui sera respectée si et seulement si une personne a au moins un âge fixé à la construction de la condition.

Par la suite on pourra supposer qu'il existe d'autres classes permettant de représenter d'autres conditions d'accès.

Un parc d'attractions est une instance de la classe `ParcAttraction` dont voici le diagramme UML :

ParcAttractions
+ PRIX_TICKET : float (final) - lesAttractions : List<Attraction> - nom : String
+Parc(nom : String) + ajouteAttraction(att : Attraction) +getRecette():float +affichage()

La recette du parc d'attractions correspond au cumul des recettes de toutes les attractions.
L'affichage correspond à afficher le nom du parc puis pour chaque attraction, son nom, sa condition d'accès et sa recette.

Les attractions sont modélisées par la classe `Attraction`. Une attraction est définie par un nom (une chaîne de caractères), un coût (en nombre d'euros) à acquitter si le mode de paiement est l'euro et une condition d'accès.
Les propriétaires du parc souhaitent également disposer d'informations sur l'utilisation de chaque attraction et savoir :

- combien de personnes utilisent l'attraction,
- quelle est la recette en euros de l'attraction. On tient bien sûr compte du mode de paiement, en euros ou en ticket, de chaque visiteur.

Q 3 . Donnez le code d'une classe `Attraction` correspondant à ce cahier des charges.

Votre classe devra au moins disposer d'une méthode :

```
public void utilise(Personne p, MoyenPaiement m) throws AccesInterditException
```

qui permet de prendre en compte que la personne `p` utilise l'attraction en payant selon le moyen `m`. L'exception est levée si la personne ne respecte pas la condition d'accès.

La classe devra également disposer d'une méthode permettant de fixer, et donc modifier, la condition d'accès.

Q 4 . Donnez le code d'une classe `ParcAttractions` correspondant au diagramme précédent.

Q 5 . Lorsqu'une attraction est en maintenance, il est bien évident qu'il n'est pas possible d'y accéder momentanément. Comment proposez vous que puisse être pris en compte une telle situation ? Soyez précis et clair dans votre réponse. Aucun code n'est demandé.

Exercice 3 : Décryptage

On s'intéresse au décryptage de messages (cryptés). On suppose qu'existe une classe `Message` pour représenter un message et que celle-ci dispose d'une méthode :

```
public boolean estCrypte()
```

qui permet de savoir si un message est sous forme cryptée (valeur de retour *true*) ou décryptée (valeur de retour *false*).

On définit un algorithme de décryptage comme un processus prenant en entrée un message m_{in} (de type `Message`) et le transforme pour fournir en sortie un autre message m_{out}. On considère que l'algorithme a réussi l'opération de décryptage si m_{out} est décrypté.

On suppose que l'on dispose de plusieurs algorithmes de décryptage que nous appellerons pour simplifier `Algo1`, `Algo2` et `Algo3`. Il peut en exister d'autres.

On appelle chaîne de décryptage un processus qui s'appuie sur plusieurs algorithmes de décryptage et les essaie successivement sur un message jusqu'à avoir réussi à décoder ce message. La liste des algorithmes de décryptage gérés par la chaîne doit pouvoir évoluer.

- Faites une proposition de conception pour gérer les algorithmes de décryptage ainsi que la chaîne de décryptage.

  Vous donnerez votre réponse sous la forme d'un diagramme UML faisant apparaître les différents types impliqués et détaillant leurs attributs, constructeurs et méthodes ainsi que leurs types, paramètres ou valeurs de retour.

- Donnez le code de la méthode qui permet de décrypter un message pour un objet "chaîne de décryptage".