ANNEXE

EXAMEN POO

Session : Juin 2008	**Groupe :** II3, IG3, RI3
Enseignants : Mlle Sghaier Imene, Mme Aloulou Narjes, Mr Rekik Rojdi	🕮 **:** Non autorisés
Durée : 1h30	**Nb. Pages:** 2

Enoncé :

Une société de vente par correspondance propose des articles dans différents rayon : habillement, électro-ménager, vidéo/hi-fi et culturel. Chaque article est caractérisé par une désignation (son nom), une référence (incrémenté de 1 pour un nouvel article), un descriptif, un prix HT. Il est possible d'obtenir le prix TTC d'un article et d'afficher les informations le concernant.

Les vêtements sont en plus définis par leur composition qui énumère les matières qui les composent et dans quelles proportions. Le taux TVA sur les vêtements est de 19.6%. Les frais de port sont 1 dinar par vêtement.

Les articles de l'électro-ménager et de la vidéo/hi-fi possèdent une consommation électrique, une durée de garantie, un poids (en Kg) et des dimensions (l x h x p) en cm. Le taux TVA sur les articles est de 19.6%. Les frais du port sont de 1 dinar tous les deux kilos majorés de 10% si le volume de l'article dépasse 0.1 m^3. La garantie est de 3 ans en électro-ménager et de 5 ans en vidéo/hi-fi. Les articles culturels sont répartis entre livres et disques. La TVA sur les livres est de 5.5 % alors qu'elle est de 19.6 % sur les disques. Les frais de port sont nuls pour les produits culturels.

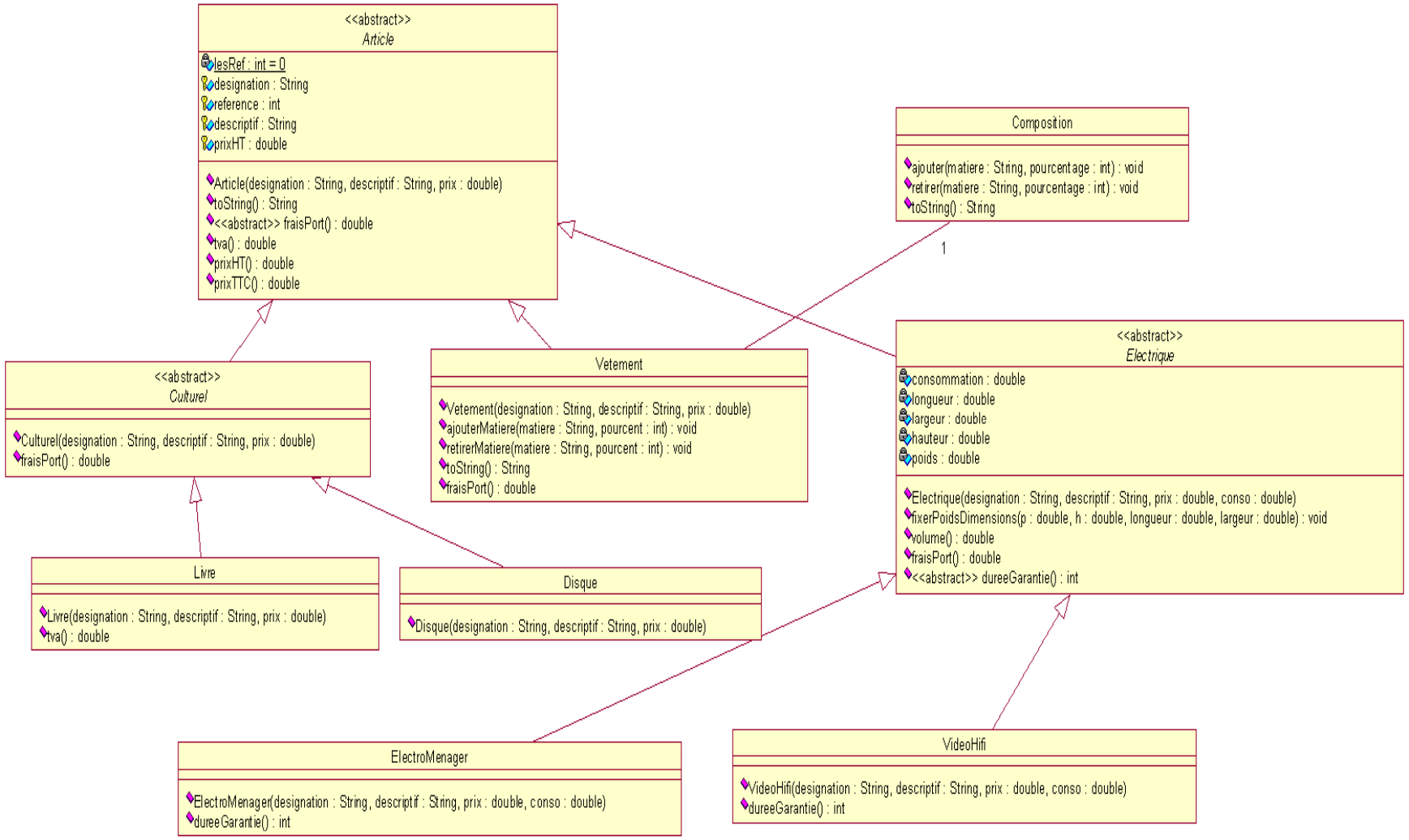
Questions :

- Développer toutes les classes en respectant le digramme de classes (Vente par correspondance) ;
- Pour la composition des vêtements (la classe Composition), on peut utiliser une Map qui pour chaque matière donne un pourcentage. N'oublier pas de vérifier que la somme des pourcentages vaut 100.

Remarque :

La relation entre la classe X et Y sera traduite en java par :


```
public  class X{
private Y  y ;
public X(parameters of X){
y= new Y(parameters of Y) ;
}}
```



Digramme de classe : Vente par correspondance

EXAMEN POO

Session : Janvier 2008	**Groupe :** II3, IG3, RI3
Enseignants : Mlle Sghaier Imene, Mme Aloulou Narjes, Mr Rekik Rojdi	🕮 **:** Non autorisés
Durée : 1h30	**Nb. Pages:** 3

Enoncé :

Une agence de location de voitures offre à ses clients la possibilité de choisir la voiture louée en fonction de différents critères.

Les voitures sont définies par une marque, un nom de modèle, une année de production et un prix de location à la journée. Pour simplifier les deux premiers paramètres seront des objets de la classe String et les deux derniers seront des int. Deux voitures sont considérées égales si tous leurs attributs sont égaux.

La classe Voiture pour laquelle on souhaite disposer des méthodes habituelles **equals** et **toString**. La méthode **toString** retournant une chaîne de caractères reprenant la marque, le nom du modèle et le prix de location. La méthode **equals** teste si une voiture est égale à une autre.

Il est possible de sélectionner parmi les voitures à louer toutes les voitures satisfaisant un critère donné.

On définit l'interface Critere ainsi :

```
public interface Critere {
//@param v la voiture dont on teste la confirmité
// @return true si et seulement si la voiture est conforme au
//critère (on dit que v satisfait le critère)
public boolean estSatisfaitPar(Voiture v);
}
```

La classe Agence gère des locations des voitures. Un client ne peut louer qu'un véhicule à la fois. On supposera pour simplifier que les clients sont identifiés par une chaîne de caractères (leur nom par exemple).

On décide de gérer ces locations par une table (java.util.Map) qui associe les clients représentés par leurs noms (clés) avec les voitures (valeurs) qu'ils ont louées. Un client n'est présent dans cette table que si il loue actuellement une voiture.

Travail demandé :

- Donner le code d'une classe ***CritereMarque*** qui est un critère satisfait par toutes les voitures d'une marque donnée. La marque est précisée à la construction du critère.
- Donner le code d'une classe ***CriterePrix*** qui est un critère satisfait par toutes les voitures dont le prix est inférieur à un prix fixé à la construction du critère.
- Donner le code d'une classe V***oiture.***

NB : Les classes ***CritereMarque*** et ***CriterePrix*** implémentent toutes les deux l'interface ***Critere***.

On donne ici une description des méthodes de la classe Agence :

- **public void loueVoiture(String client, Voiture v) throws VoitureException** : permet au client de louer la voiture v ; un client ne peut louer qu'une seule voiture, il ne peut louer une autre que s'il rend la première. L'exception est levée soit si la voiture n'existe pas dans l'agence soit si elle est déjà Louée.
- **public void afficheSelection(Critere c) :** permet d'afficher toutes les voitures qui satisfont un critère c donné qui peut être évidement un CriterePrix ou un CritereMarque.
- **public boolean estLoueur(String client) :** renvoie true si et seulement si le client est un client qui loue actuellement une voiture.
- **public boolean estLoue(Voiture v) :** renvoie true si et seulement si la voiture est actuellement louée.
- **public void rendVoiture(String client) :**le client rend la voiture qu'il a louée. Il ne se passe rien si il n'avait pas loué de voiture.
- **public Iterator lesVoituresLouees()** qui renvoie la collection des voitures de l'agence qui sont actuellement louées.

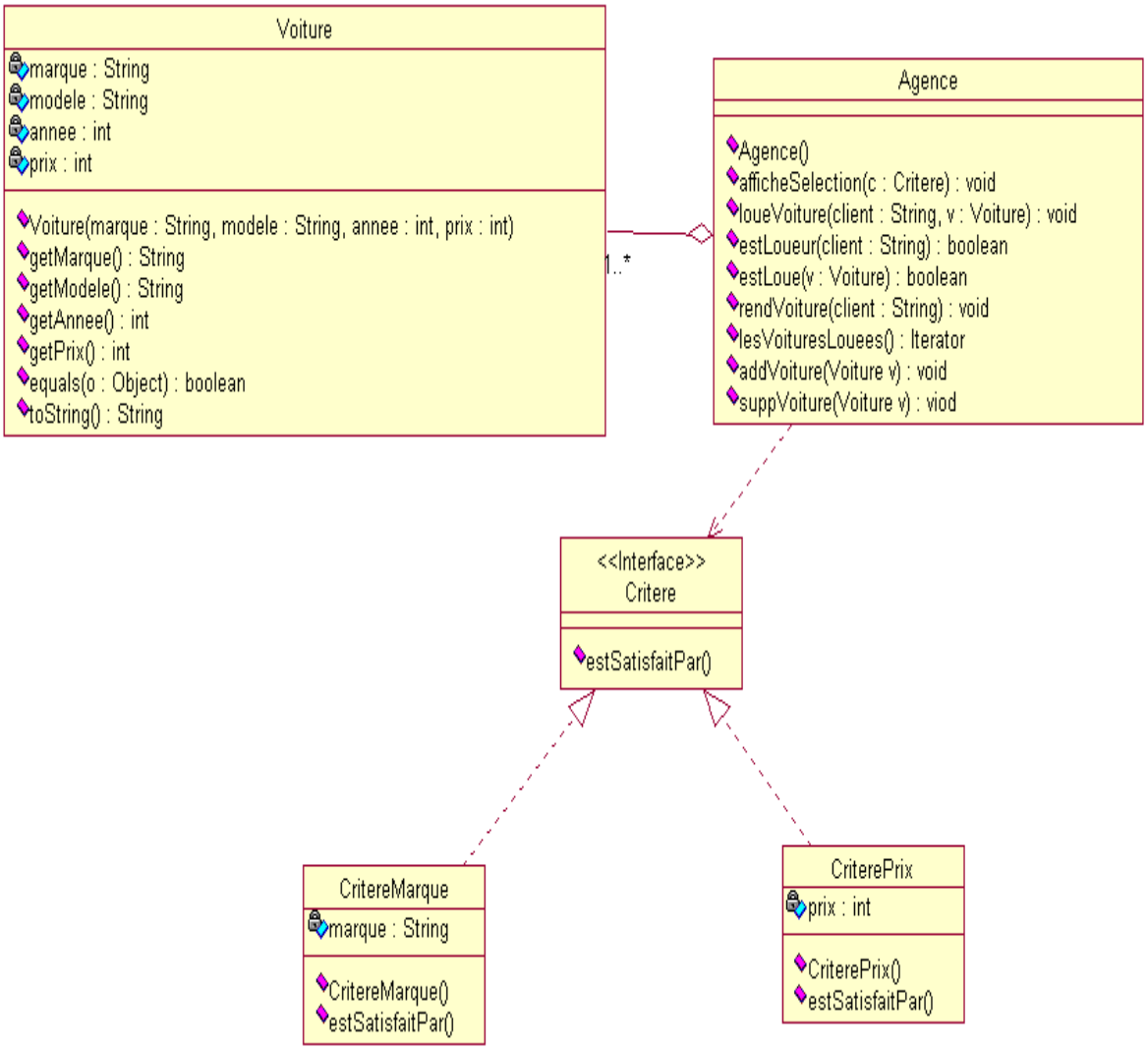


Diagramme de classe : Agence de Location de Voitures

DEVOIR SURVEILLE

Matière	**:** Programmation Orientée Objets	**Groupe**	**:** II3
Enseignante :	**Mlle I.Sghaier**	🕐	**:** 1H
🕮	**:** Non autorisés	**Nb. Pages:**	2

Exercice1: (10 pts: 5+5)

1 ère partie:

1. Créer une classe « employe» correspondante au schéma UML suivant. La méthode decrisToi() affiche à l'écran le nom et le prénom de l'employé

Class employe
- String nom - String prenom - Double Salaire - int NCI -
+ Employe() + Employe(int n) + Employe(String nom,String prenom) + Employe(String nom,String prenom,int n, double s) + decrisToi() + String getNom() + String getPrenom() + Double getSalaire() + Int getNCI()

2. Créer une classe equipe qui utilise la classe Employe. La fonction main de Equipe comprendra la création d'un tableau de quatre employés et l'affichage de leurs noms

3. Peut-on modifier la variable d'instance nom depuis l'extérieur de la classe ? et depuis l'intérieur ?

2ème Partie :

1. Créer une classe technicien qui hérite de la classe employe. Celle-ci comprendra une variable d'instance supplémentaire : specialite.
    a. Ecrire des constructeurs : sans arguments et à 1 seul argument « String spécialité ». Vous utiliserez les constructeurs de la classe mère.
    b. Redéfinisser la méthode decrisToi(), en faisant appel à la méthode de la classe mère. (décris toi va afficher à l'écran : nom, prénom, spécialité).

2. En changeant les modificateurs d'accès des données membres de la classe mère, de private vers protected peut-on accéder à ces variables de l'extérieur de la classe Employe? Quel danger cela présente t-il en termes d'encapsulation ?
3. Expliquer le mécanisme de construction d'une classe dérivée
4. Ecrire une classe, qui teste la classe technicien.

Exercice 2: (10 pts : 6+4)

1. Ecrire une classe appelée « FonctionMath » qui contient les méthodes **statiques** suivantes :
   - Factoriel : c'est une méthode qui permet de calculer et d'afficher le factoriel d'un entier passé lors de l'exécution.
   - RacineCarré : C'est une méthode qui permet de calculer et d'afficher la racine carrée d'un nombre n passé lors de l'exécution
   - BinaryCode : C'est une méthode qui permet de lire un entier sur la ligne de la commande et afficher sa correspondance en binaire
   - HexCode : C'est une méthode qui permet de lire un entier sur la ligne de la commande et afficher sa correspondance en Hexadécimal.

NB : Pour les trois dernières méthodes utiliser les méthodes suivantes de la classe **Integer** : *toBinaryString et toHexString*.

2. Ecrire une classe «TestFonctionMath» qui utilise ces fonctions mathématiques.

Devoir à la maison

Soit la classe **« Liste »** caractérisée par les attributs suivants :

- **tab** : représente un tableau d'entiers.
- **Taille** : représente le nombre d'éléments de la liste.

Et les méthodes suivantes :

- **Ajout_tête** : Ajoute un élément à la tête de la liste.
- **Ajout_queue** : Ajoute un élément à la fin de la liste.
- **Ajoutk** : Ajoute un élément à une position donnée.
- **Suppression_tête** : Supprime le premier élément de la liste.
- **Suppression_queue** : Supprime le dernier élément de la liste.
- **Suppk** : Supprime l'élément se trouvant à une position donnée.
- **Suppval** : Supprime un élément donné d'une liste.
- **Accesval** : Retourne la position de 1 ère occurrence d'un élément dans une liste.
- **Liste_vide** : Vérifie si la liste est vide ou non.
- **Liste_pleine** : Vérifie si la liste est saturée ou non.
- **Longueur** : retourne la taille d'une liste.
- **Occurrence** : Retourne le nombre d'occurrence d'un élément donné.

Travail demandé

1- Ecrire en Java la classe Liste

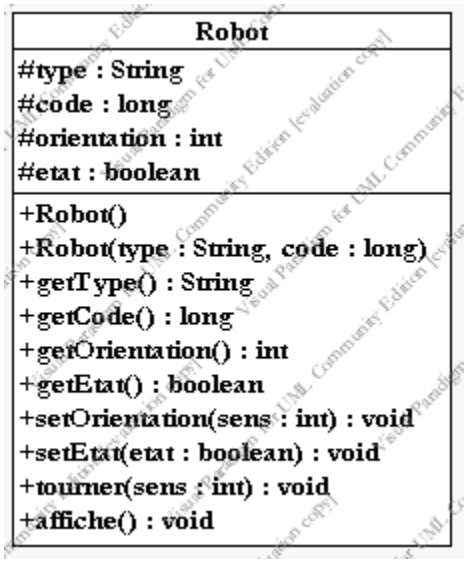
2- Ecrire une classe Test_Liste qui permet de créer un objet de type Liste et le manipuler à travers l'appel des méthodes de cette classe.

DEVOIR SURVEILLE

Matière : Programmation Orientée Objets	**Groupe** : II3
Enseignante : Mlle Imene Sghaier	🕐 : 1H
🕮 : Non autorisés	**Nb. Pages :** 2

Partie I

1. Ecrire en java la classe **Robot** obéissant au schéma UML suivant.



- ✈ orientation : est un attribut qui désigne l'orientation du Robot (**1 = Nord, 2 = Est, 3 = Sud, 4 = West**)
- ✈ La méthode **tourner ()** qui permet de tourner l'orientation du Robot.
- ✈ Les constructeurs qui permettent d'initialiser les attributs du Robot
- ✈ La méthode **affiche()** qui permet d'afficher l'état et l'orientation d'un Robot

2. Ecrire une classe **TestRobot** qui utilise cette classe ; sa fonction main comprendra la création d'un tableau de quatre Robot, initialiser leur attributs, les afficher, les mettre en marche (etat=true) ainsi que les orienter vers des orientations différentes chacun de l'autre.

Partie II

Soit une classe **RobotMobile** qui hérite de **Robot** et ayant en plus les attributs entiers privés abs et ord : ce sont les attributs qui définissent la positon de **RobotMobile** (abscisse et ordonné) ainsi qu'une méthode **void avancer(int x)** qui permet d'avancer le Robot selon son orientation :

- si on avance de x vers l'Est l'abscisse augmente de x,
- si on avance vers le West de x l'abscisse diminue de x,
- si on avance vers le nord de x l'ordonnée augmente de x,
- si on avance vers le Sud de x l'ordonnée diminue de x,

Et une méthode **void affichePosition()** qui affiche la position (coordonnées).

1. Ecrire un constructeur sans argument de la classe RobotMobile
2. Ecrire un constructeur à quatre arguments (type, code, abs et ord) de la classe RobotMobile
3. Redéfinissez la méthode affiche tout en utilisant celle de la classe mère et la méthode **affichePosition().**
4. Ecrire une classe testRobotMobile qui permet de créer un RobotMobile, lui afficher les attributs et lui appliquer la séquence d'actions suivante tout en affichant chaque fois la position du Robot ainsi que son orientation.
    a. Tourner vers l'Est
    b. Avancer de 4 vers le West
    c. Avancer de 6 vers le Nord
    d. Avancer de 14 vers l'Est
    e. Reculer de 8 vers le sud

Bon courage !