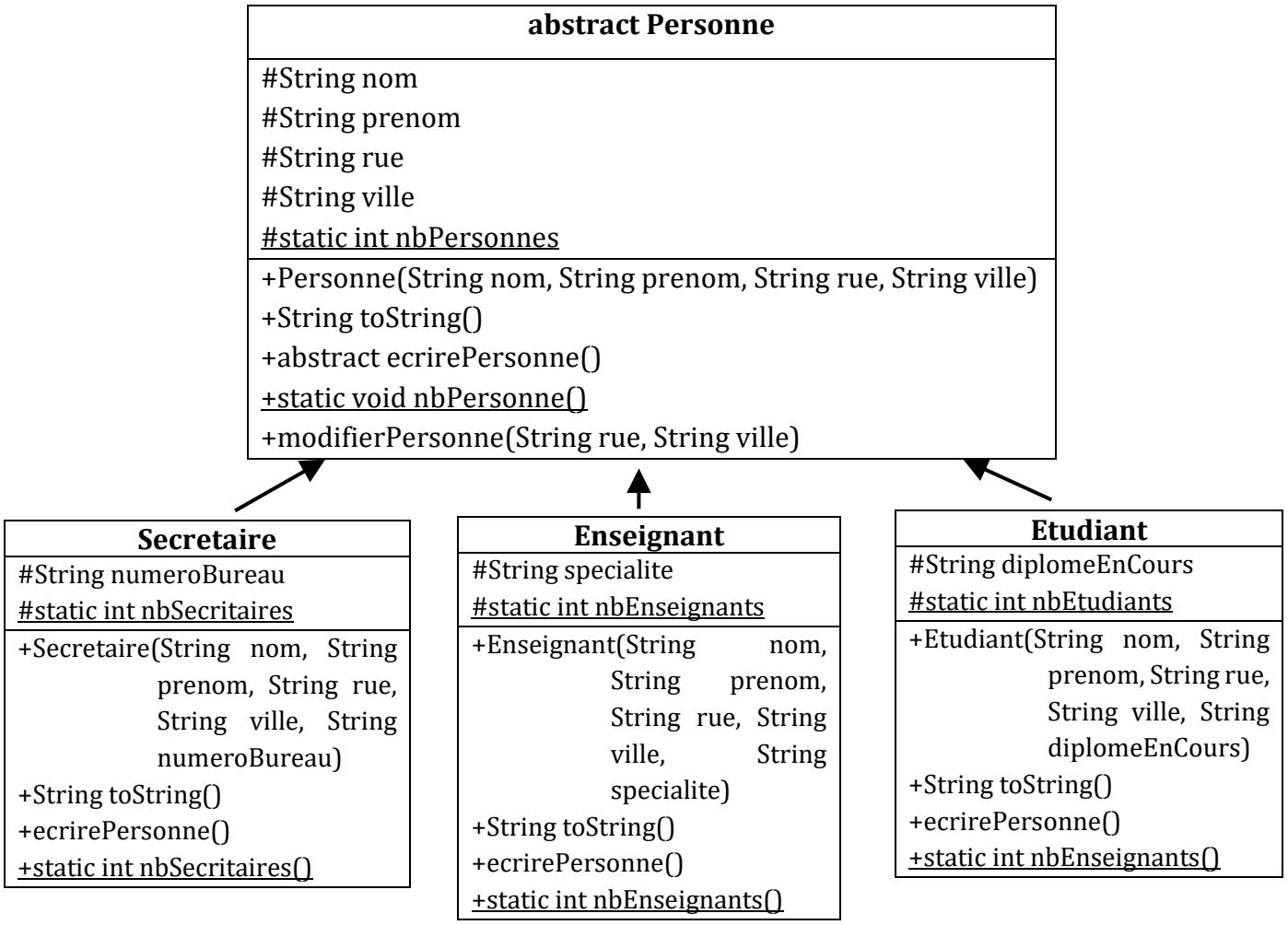
TD-TP no.5&6: Classes Abstraites et Interfaces

Partie 1 : (Classes Abstraites)

Exercice 1 :

Dans un établissement d'enseignement et de manière schématique, on trouve trois sortes de personnes : des administratifs (au sens large, incluant des techniciens) représentés par la catégorie secrétaire, des enseignants et des étudiants. Chaque personne est caractérisée par son nom et prénom, son adresse (rue et ville) qui sont des attributs privés et communs à toutes les personnes. On peut représenter une personne suivant un schéma UML comme indiqué sur la figure ci-dessous. Les variables d'instances (attributs) sont le nom, le prénom, la rue et la ville. nbPersonnes est une variable de classe (donc static) qui comptabilise le nombre de Personne dans l'établissement.



➢ Classe Personne

On trouve dans la classe Personne les méthodes public suivantes:

- ***le constructeur Personne*** (String nom, String prenom, String rue, String ville) : crée et initialise un objet de type Personne.
- ***String toString ()*** : fournit une chaîne de caractères correspondant aux caractéristiques (attributs) d'une personne.
- ***EcrirePersonne ()*** : pourrait écrire les caractéristiques d'une personne. Sur l'exemple ci-dessus, elle ne fait rien. Elle est déclarée abstraite.
- ***static nbPersonnes ()*** : écrit le nombre total de personnes et le nombre de personnes par catégorie. C'est une méthode de classe.
- ***modifierPersonne (String rue, String ville)*** : modifie l'adresse d'une personne et appelle ecrirePersonne () pour vérifier que la modification a bien été faite.

➢ Classe Secretaire

Secretaire est une Personne. Elle possède toutes les caractéristiques d'une Personne (nom, prenom, rue, ville) plus les caractéristiques spécifiques d'une secrétaire soit sur l'exemple, un numéro de bureau. Les méthodes suivantes sont définies pour un objet de la classe Secretaire:

- ***Secretaire (String nom, String prenom, String rue, String ville, String numeroBureau)*** : le constructeur d'un objet de la classe Secretaire doit fournir les caractéristiques pour construire une Personne, plus les spécificités de la classe Secretaire(numéro de bureau).
- ***String toString ()*** : fournit une chaîne contenant les caractéristiques d'une Secretaire.
- ***EcrirePersonne ()*** : écrit "Secrétaire : " suivi des caractéristiques d'une Secretaire.

➢ Classe Enseignant

Un Enseignant est une Personne enseignant ayant une spécialité (informatique, mathématiques, anglais, gestion, etc.). Les méthodes pour Enseignant sont similaires à celles de Secretaire.

➢ Classe Etudiant

Un Etudiant est une Personne préparant un diplôme (**diplomeEnCours**). Les méthodes pour Etudiant sont similaires à celles de Secretaire.

Une variable privée **static** dans chaque classe compte le nombre de personnes créées dans chaque catégorie. Une méthode **static** du même nom que la variable fournit la valeur de cette variable **static** (**nbSecretaires, nbEnseignants, nbEtudiants**).

Ecrire le code de :

- la classe Perssone,
- la classe Secretaire.
- la classe Enseignant.
- et la classe Etudiant.

Écrire un programme **TestPersonne** qui permet de tester les trois classes. Dans cette classe :

- Créer un objet de type **Secretaire**.
  S :["Said", "Abidi","Rue Elamal", "Casablanca", "A123"]; puis afficher les caractéristiques de l'objet **Secretaire** par la méthode **EcrirePersonne().**
- Créer un objet de type **Enseignant**.
  E :["Ahmed", "Sbihi", "Rue Bel Air", "Fès", "Informatique"]; puis afficher les caractéristiques de l'objet **Enseignant** par la méthode **EcrirePersonne().**
- Créer deux objets de type **Etudiant**.
  E1 :["Samir", "Merras", "rue saules ", "Oujda", "licence informatique"];
  E2 : ["Hamid", "Nissani", "rue d'olivier", "Taza", "DUT informatique"] puis afficher les caractéristiques de l'objet **Etudiant E2** par la méthode **EcrirePersonne().**
- Modifier l'adresse de l'objet Enseignant et l'objet Secretaire par :
  E : "rue du grenadier", "Rabat" et S : "rue Taounat", "Safi"
- Afficher le nombre total de personnes et le nombre de chaque type d'objet en utilisant la méthode **nbPerssonne().**
- Afficher les caractéristiques de l'objet Enseignant et Secretaire après modification d'adresse.
- Créer d'autres objets (de type Secretaire, Enseignant, Etudiant) et appliquer les différentes méthodes des classes précédentes.

Exercice 2

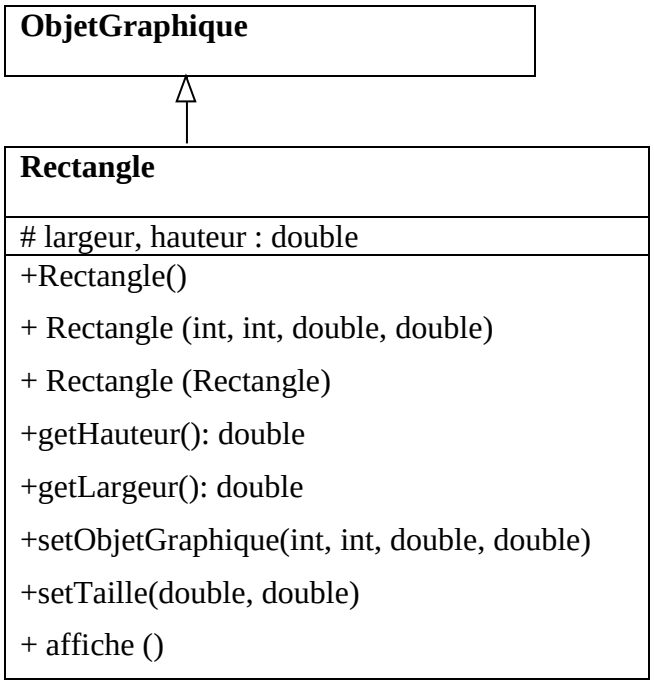
1. Ecrire la classe abstraite **ObjetGraphique** qui est définie par le diagramme de classe suivant :

ObjetGraphique
x,y : entier
+ObjetGraphique() +ObjetGraphique(int, int) +ObjetGraphique(ObjetGraphique) +getX(): int +getY(): int +setPosition(int, int) +deplacer(int, int) + affiche *()*

Remarque : affiche est une méthode abstraite qui sera définie dans les classes dérivées et permettant d'afficher la description de l'objet qui l'a fait appel.

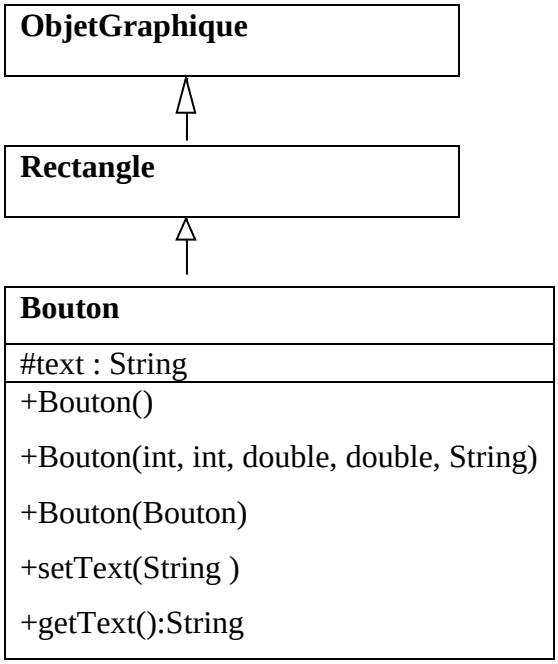
2. On veut créer une classe nommé **Rectangle** qui hérite de la classe **ObjetGraphique**.

Ecrire la classe **Rectangle**, son diagramme de classe suivant :



3. On veut créer une classe nommé **Bouton** qui hérite de la classe **Rectangle** et qui a une variable membre de type String, appelé **text**.

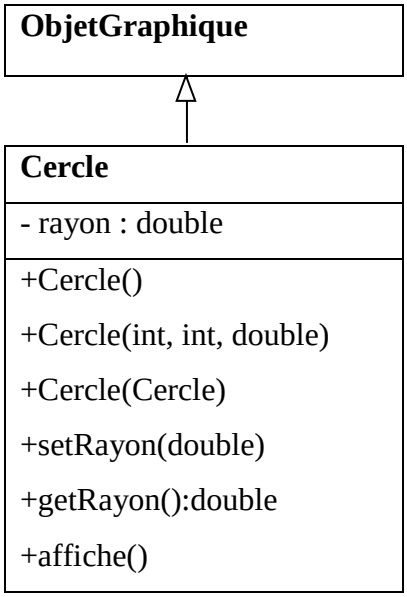
Ecrire la classe **Bouton**, son diagramme de classe suivant :



+ affiche ()

4. On veut créer une classe nommé **Cercle** qui hérite de la classe **ObjetGraphique**.

Ecrire la classe **Cercle**, son diagramme de classe suivant :



5. Ecrire une classe de test qui permet :

- Créer un tableau hétérogène qui contient plusieurs cercles, rectangles et boutons
- Afficher les informations de tous les objets du tableau

6. Soit la classe ListeObjetGraphique ayant la structure suivante:

```
public class ListeObjetGraphique {
private ObjetGraphique[] liste; // le tableau des objets
private int taille;// la taille du tableau liste
private int index;//le nombre des éléments dans le tableau
 public ListeObjetGraphique(int taille) { } //constructeur pour l'initialisation
 public void ajouter(ObjetGraphique f) {}//fonction permettant d'ajouter un objet au tableau
 public void afficheObjetsGraphique () { }//fct pour afficher les objets du tableau liste
        }
```

a) Implémenter la classe ListeObjetGraphique (et ses différentes méthodes).

b) Ecrire une classe de test qui permet :

- Créer un objet de type ListeObjetGraphique qui permet de mémoriser jusqu'à 10 objets graphiques

- Ajouter plusieurs cercles, rectangles et boutons.

7. Le package java.util dispose des deux classes ArrayList et ListIterator, permettant la gestion de liste :

ArrayList : est une classe de collection (du package java.util) qui représente un tableau d'objets de taille variable, fonctionnant comme une liste. Cette classe dispose des méthodes suivantes :

- **add(élément)** : permet d'ajouter un objet nommé élément à collection;
- **listIterator()** : permet de retourner un iterateur (de type ListIterateur) pour le parcours de la collection.

ListIterator : permet le parcours d'une liste (comme ArrayList) d'une manière séquentiel. Cette classe dispose des méthodes suivantes :

- **hasNext()** : permet de vérifier l'existence d'un élément dans la liste, parcouru par cet itérateur;
- **next()** : permet de retourner l'élément suivant dans la liste, parcouru par cet itérateur;

Refaire la question 6 en créant un objet de type ArrayList pour mémoriser les 10 objets graphiques.

Partie 2 : (Interfaces)

*Exercice 3 (*Gestion de stock d'un magasin)

On veut créer une application qui permet de gérer le stock d'un magasin en utilisant la notion d'interface. Dans cette application, vous devez d'abord comprendre la hiérarchie des classes et des interfaces. Cela est nécessaire pour voir clairement les paramètres de retour des méthodes ainsi que la liste de leurs paramètres formels. La spécification du cette application en termes de classes et des interfaces se présente comme suit:

1. On a trois interfaces

1.1 VendableKilo: l'interface pour les produits qui se vendent par kilogramme

Méthodes:

vendre: cette méthode reçoit la quantité vendue du produit, retourne le revenu du magasin et modifie le stock

1.2 VendablePiece: l'interface pour les produits qui se vendent par pièces

Méthodes:

vendre: cette méthode reçoit la quantité vendue du produit, retourne le revenu du magasin et modifie le stock

1.3 Solde: l'interface pour les produits succeptibles d'être vendus en solde

Méthode:

a. **lanceSolde**: cette méthode baisse le prix du produit par le pourcentage donné
b. **termineSolde**: cette méthode augmente le prix du produit par le pourcentage donné

2. On a une classe générale des Arcticles (Article).

Propriétés:

a. **prixAchat**: le prix pour lequel le supermarché achète le produit
b. **prixVente**: le prix pour lequel le supermarché vend le produit
c. **nom**: le nom du produit
d. **fournisseur**: le nom du fournisseur du produit

Méthodes (autre que le constructeur):

a. **rendement** : calculateur du taux du rendement
b. **toString** : description des caractéristiques du produit sur l'écran (les prix, le nom, le fournisseur; rendement)

Cette classe n'implémente aucune interface.

3. On a deux classes dérivées de la classe « Article »

Chaque classe dérivée de la classe article respecte la règle suivante: au moment de la construction de l'objet, le stock et vide.

3.1 La classe des artciles élécroménagers(ArticleElectromenager)

Propriétés suplémentaires:

stock : nombre de pièces en stock

Méthodes supplémentaires (autre que le constructeur):

a. **remplirStock** : remplir le stock
b. **toString**: description des caractéristiques du produit sur l'écran (les prix, le nom, le fournisseur; rendement; stock)

Il faut implémenter les interfaces correspondantes à cette classe.

3.2 La classe des primeurs(ArticlePrimeur)

Propriété supplémentaires:

stock : quantité en stock

Méthodes supplémentaires:

a. **remplirStock** : remplir le stock

b. **toString** : description des caractéristiques du produit sur l'écran (les prix, le nom, le fournisseur; rendement; stock)

Il faut implémenter les interfaces correspondantes à cette classe, sachant que les primeurs ne peuvent pas être vendues en solde.

4. On a une classe pour les magasins

Propriétés:

a. **depense** : le coût d'achat des produits
b. **revenu**: les revenus après la vente des produits
c. **ArticleElectromenager, ArticlePrimeur**: deux tableaux de deux articles (éléctroménagers et primeurs)

Méthodes (autre que le constructeur):

a. **toString** : description de l'état du magasin
b. **rendement** : calculateur du taux de rendement

Questions:

1. Créer les différentes interfaces et classes.
2. Pour tester le travail que vous avez réalisé, écrire un programme d'essai permettant de : définir les articles à vendre, effectuer le remplissage du stock , simuler les achats et utiliser les différentes autres méthodes des classes crées.