USTHB 2014-2015
Licence 2 Académique Semestre 4
TD POO I. Boussaïd & C. Ighilaza

Feuille d'exercices N°3

(Héritage et polymorphisme)

Exercice 1 *Gestion de la bibliothèque*

On modélise une application devant servir à l'inventaire d'une bibliothèque. Elle devra traiter des documents de nature diverse : des livres, des dictionnaires, et autres types de documents qu'on ne connaît pas encore précisément mais qu'il faudra certainement ajouter un jour (articles, bandes dessinées, ...). Tous les documents possèdent un *numéro d'enregistrement* et un *titre*. À chaque livre est associé, en plus, un *auteur* et un *nombre de pages*, les dictionnaires ont, eux, pour attributs supplémentaires une *langue* et un *nombre de tomes*. On veut manipuler tous les articles de la bibliothèque au travers de la même représentation : celle d'un document.

Q1. Ecrire les classes Java `Document`, `Livre` et `Dictionnaire`.

Q2. Dans chacune de ces classes définissez le constructeur qui prend autant d'arguments qu'il y a de variables d'instance et qui se limite à initialiser ces dernières avec les valeurs des arguments.

Q3. Comme les variables d'instance ont été déclarées `private` (c'est conseillé) définissez également des accesseurs publics permettant de consulter les valeurs de ces variables.

Q4. Définissez la classe `ListeDeDocuments` permettant de créer une liste vide de documents, puis y adjoindre une fonction permettant d'ajouter un document.
- Dans la classe `ListeDeDocuments` définissez une méthode `tousLesAuteurs()` qui affiche la liste des numéros des documents de la liste avec, pour chacun, l'éventuel auteur.
- Ajoutez dans la classe `ListeDeDocuments` une méthode `tousLesDocuments()` qui affiche consécutivement la description de tous les documents.

Q5. Redéfinissez la méthode `toString` des différentes classes produisant une description sous forme de chaîne de caractères des instances de la classe.

Q6. Définissez une classe `Bibliothèque` réduite à une méthode `main` permettant de tester les classes précédentes.

Exercice 2 *Calcul des impôts locaux*

Dans le cadre de l'informatisation d'une mairie, on veut automatiser le calcul des impôts locaux. On distingue deux catégories d'habitation : les habitations à usage professionnel et les maisons individuelles, l'impôt se calculant différemment selon le type d'habitation. Pour cela, on définit les classes `HabitationProfessionnelle` et `HabitationIndividuelle` et les caractéristiques communes à ces deux classes sont regroupées dans la classe `Habitation`. On a donc un schéma de classes où les classes `HabitationProfessionnelle` et `HabitationIndividuelle` héritent de la classe `Habitation`.
L'objet de cet exercice est d'implémenter ce schéma d'héritage et de mettre en œuvre le mécanisme de liaison dynamique.

Q1. **Définition de la classe Habitation**

Attributs La classe `Habitation` comprend les attributs :
- `proprietaire` du type chaîne de caractères et qui correspond au nom du propriétaire,
- `adresse` du type chaîne de caractères et qui correspond à l'adresse de l'habitation,
- `surface` du type `double` et qui correspond à la surface de l'habitation et qui permet de calculer le montant de l'impôt.

Constructeur La classe `Habitation` possède un constructeur à trois paramètres permettant d'initialiser une instance de la classe `Habitation` :

```
Habitation(String P, String A, double S);
```

Méthodes d'instances La classe `Habitation` possède les méthodes d'instance suivantes :
- `double Impot()` qui permet de calculer le montant de l'impôt que doit payer le propriétaire de l'habitation à raison de 2€ par m^2.
- `void Affiche()` qui permet d'afficher les trois attributs de la classe `Habitation`.

Question Définissez la classe `Habitation`

Q2. **Définition de la classe HabitationIndividuelle**

Le calcule de l'impôt d'une maison individuelle est différent de celui d'une habitation, il se calcule en fonction de la surface habitable, du nombre de pièces et de la présence ou non d'une piscine. On compte 15€/pièce et 80€ supplémentaire en cas de présence d'une piscine.

Question Définir la classe `HabitationIndividuelle` qui hérite de la classe `Habitation`. Ajouter les attributs `NbPieces` de type entier et `Piscine` de type booléen. Redéfinir les méthodes `Impot` et `Affiche`. La méthode `Affiche` doit afficher, les attributs `proprietaire`, `adresse` et `surface` de la classe `Habitation`, et les attributs `NbPieces` et `Piscine` propres à la classe `HabitationIndividuelle`.

Q3. **Définition de la classe HabitationProfessionnelle**

Le calcule de l'impôt d'une habitation à usage professionnel est également différent de celui d'une habitation. Il se calcule en fonction de la surface occupée par le bâtiment et du nombre d'employés travaillant dans l'entreprise. On compte 150€ supplémentaire par tranche de 10 employés.

Question Définir la classe `HabitationProfessionnelle` qui hérite de la classe `Habitation`. Ajouter l'attribut `NbEmployes` de type entier. Redéfinir les méthodes `Impot` et `Affiche`. La méthode `Affiche` doit afficher, en plus des attributs `proprietaire`, `adresse` et `surface`, l'attributs `NbEmployes`.

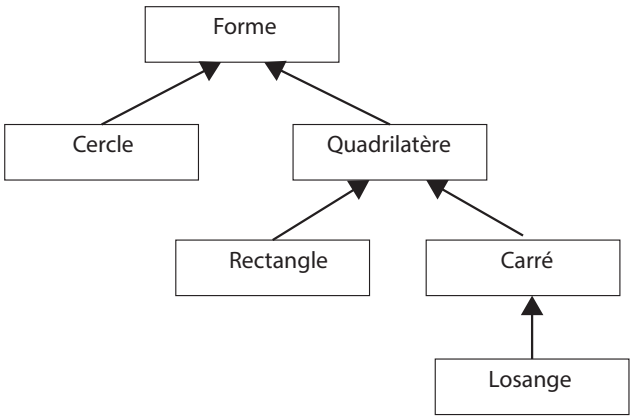
Q4. **Gestion des habitations d'une commune**

Objectif : Mettre en oeuvre le mécanisme de liaison dynamique. On désire à présent calculer l'impôt local des habitations (individuelles ou professionnelles) d'une commune. Pour cela, on vous demande d'utiliser une collection d'objets représentée par un tableau où chaque élément désigne une habitation individuelle ou professionnelle.

Exercice 3 *(Examen 2009-2010)*

Partie I :

1. Définir une classe `Point` pour représenter les coordonnées d'un points caractérisées par deux nombres flottants `x`, `y`, mémorisés dans deux variables d'instance privées (de type `double`). La classe `Point` comportera les méthodes d'instance suivantes :
   - un constructeur recevant en argument les coordonnées d'un point,
   - les accesseurs qui renvoient/modifient les coordonnées du point,
   - `String toString();` qui transforme un point en une chaîne de caractères de la forme `"(x,y)"`, en vue de son affichage,
   - `void affiche();` qui affiche un point.
2. Définir les classes qui respectent la hiérarchie suivante :



La classe Forme : Formes géométriques.

Attributs : Le point centre de la forme

Deux Constructeurs : L'un initialise le centre à 0, l'autre initialise le centre à (x,y)

Méthodes :

- `String toString()` : retourne le point (x,y) en vue de son affichage
- `void afficheForme()` : affiche une forme ainsi "Le centre de cette forme est : (x,y)"
- `double surface() { return 0 ; }`. On retourne 0 car on ne sait pas de quelle forme il s'agit
- `double périmètre() { return 0 ; }` . Idem

La classe Cercle : Sous classe de la classe Forme.

Attributs : Un cercle est identifié par son centre (Point) et son rayon

Un constructeur : Avec arguments

Accesseur : Ceux du rayon

Méthodes :

- `double surface()`
- `double périmètre()`

- `boolean estDansCercle(Point p)` : vérifie si `p` est ou non à l'intérieur du cercle, `(x,y)` est dans le cercle de rayon `R` ssi $x^2 + y^2 \leq R^2$
- `void afficheForme()` : affiche "Cercle : Le centre de cette forme est (x,y)"

La classe Quadrilatère : Sous classe de la classe Forme.

Attributs : Un quadrilatère est identifié par 4 points `a, b, c, d`.

Constructeur : Avec arguments

Méthodes :

- `static Point milieu(Point p1, Point p2)`
- `static double distance(Point p1, Point p2)`
- `boolean parallélogramme(double epsilon)` : vérifie si un quadrilatère est un parallélogramme
- `void afficheForme()` : affiche "Quadrilatère : Le centre de cette forme est (x,y)"

La classe Rectangle : sous classe de la classe Quadrilatère.

Attributs : Un rectangle est identifié par 4 points et 2 cotés, longueur et largeur.

Constructeur : Avec arguments

Accesseurs : Ceux de longueur et largeur

Méthodes :

- `double surface()`
- `double périmètre()`
- `void afficheForme()` : affiche "Rectangle : Le centre de cette forme est (x,y)"

La classe Carré : sous classe de la classe Quadrilatère.

Attributs : Un carré est identifié par 4 points et un coté.

Constructeur : Avec arguments

Accesseurs : Ceux du coté

Méthodes :

- `double surface()`
- `double périmètre()`
- `void afficheForme()` : affiche "Carré : Le centre de cette forme est (x,y)"

La classe Losange : Sous classe de la classe Carré.

Attributs : Un losange est identifié par 4 points et un coté.

Constructeur : Avec arguments

Méthodes :

- `double surface()`
- `double périmètre()`
- `void afficheForme()` affiche "Losange : Le centre de cette forme est (x,y)"

Programme principal . Ecrire le programme qui :

a) Étant donné un tableau `TF` de `N` Formes ($N \leq 100$), affiche pour chaque `Forme` s'il s'agit d'un `Cercle`, `Rectangle`, `Carre`, `Losange` puis affiche leur surface et leur périmètre.

b) À votre avis quelle surface et périmètre affichera votre programme si la forme est un `Quadrilatere` ?

c) Étant donné un tableau `TQ` de `M` Quadrilatères ($M \leq 100$), affiche pour chaque quadrilatère s'il s'agit d'un rectangle, carré, losange ou aucun des trois c'est-à-dire un quadrilatère quelconque en effectuant les tests nécessaires.

Partie II : Si on décide d'écrire les méthodes milieu et distance dans la classe `Point` (au lieu de la classe `Quadrilatere`) quelles seront les modifications à faire ? Réécrire les classes concernées.

Partie III : Si dans la classe `Forme` on ne met que les signatures des méthodes `surface()` et `perimetre()` quels sont les modifications à faire dans votre programme ? Réécrire les classes concernées.

Bon travail !