Programmation Orientée Objet (POO)

EX 4 : Héritage

Objectifs

Comprendre les mécanismes de l'héritage d'attributs et de méthodes
Connaître les notions de méthode polymorphe et de résolution d'appel de méthode
Savoir concevoir et programmer une hiérarchie de classes

I. Gestion d'une bibliothèque

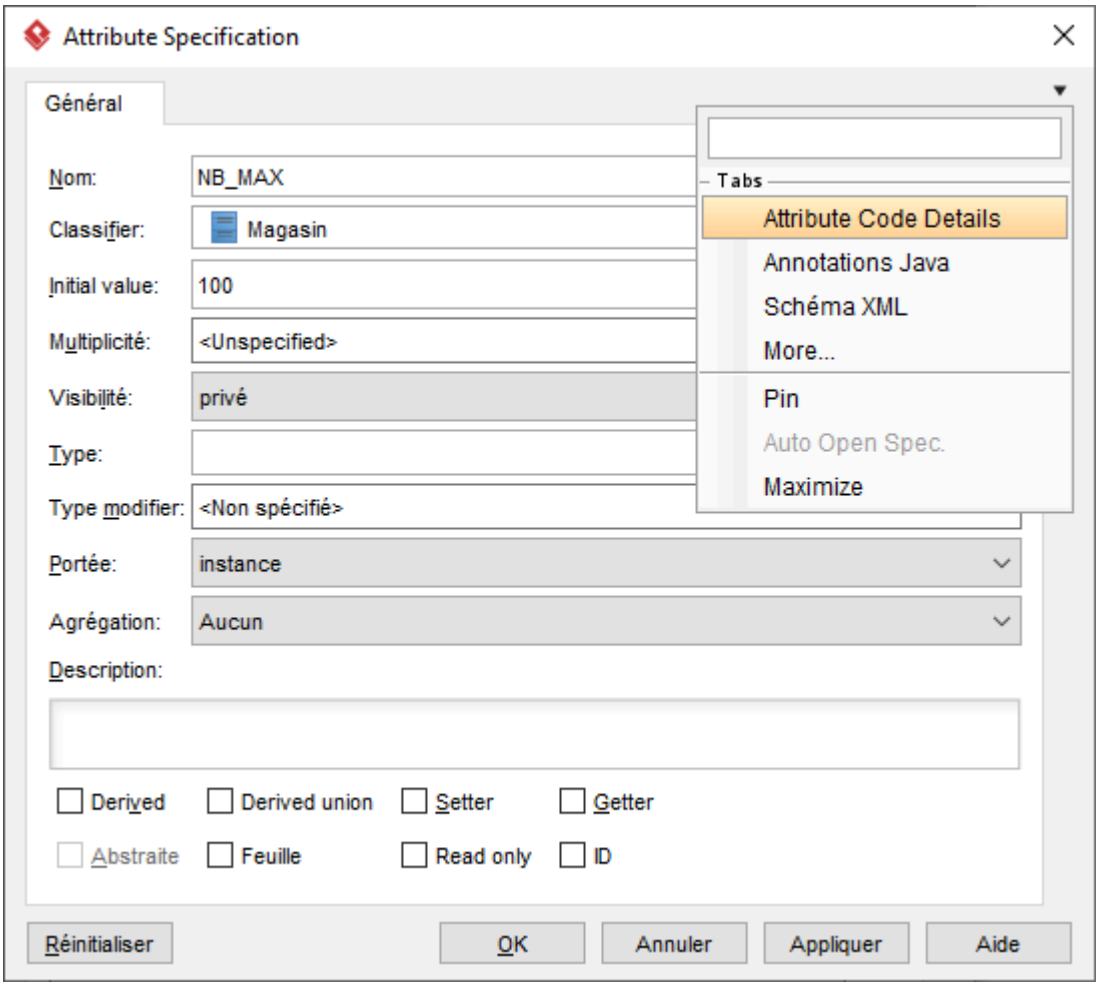
Cet exercice vise à vous faire manipuler une hiérarchie des classes et l'héritage d'attributs.
Pour la gestion d'une bibliothèque on vous demande d'écrire une application traitant des *documents* de natures diverses : des livres, des revues, des dictionnaires, etc.
Les *livres*, à leur tour, peuvent être des *romans* ou des *manuels*.
Tous les documents ont un numéro d'enregistrement (un entier) et un titre (une chaîne de caractères).
Les livres ont, en plus, un auteur (une chaîne) et un nombre de pages (un entier).

Les romans ont éventuellement un prix littéraire (un entier conventionnel).
Vous définirez les valeurs possibles pour cet attribut sous la forme de constantes dans la classe Roman : GONCOURT = 1, MEDICIS = 2, INTERALLIE = 3.

Les manuels ont un niveau scolaire (un entier).
Les revues ont un mois et une année (des entiers).
Les dictionnaires ont une langue (une chaîne de caractères convenue, comme "anglais", "allemand", "espagnol", etc.).
Tous les objets en question ici (livres, revues, dictionnaires, romans, etc.) doivent pouvoir être manipulés en tant que documents.
Représentez aussi le fait que les documents sont associés à une bibliothèque.

1) Faire le schéma des classes en UML
On pourra mettre en **protected** les attributs des classes qui pourraient être accédés directement par les sous-classes.
Vous prévoirez une association entre Bibliothèque et Document qui se traduire ensuite en Java par un attribut de Bibliotheque de type ArrayList<Document>.
ATTENTION : Il faut vérifier la logique de votre schéma notamment qu'il est possible de dire par exemple : "un Roman est-un Livre"

Pour pouvoir sélectionner final dans Visual Paradigm : cliquez sur la **petite flèche noire en haut à droite** dans la spécification d'un attribut et sélectionnez "Atribute Code Details" => cela ouvrira un onglet qui vous permettra de cocher "final":



2) Générer les classes en Java depuis Visual Paradigm
ATTENTION : ne pas redéclarer les attributs hérités dans les sous-classes !

II. Banque et héritage

Dupliquer votre paquetage banqueAL de EX 03 en un projet banqueHeritee (sélectionner le package banqueAL, menu droit Copy, menu droit Paste, nouveau nom : banqueHeritee).
Déplacer ce nouveau package dans votre projet EX-04.

Modifier les différentes classes de manière à gérer de manière plus générale une classe Compte et ses sous-classes CompteEpargne et CompteCourant.

On gérera maintenant la numérotation de tous les comptes (courants et épagne) dans la classe Compte.

Vérifier que c'est logique de dire :

Un …….. EST-UN ...

Un …….. EST-UN ...

Un …….. A-UN ...

Un …….. A-UN ...

Quels sont les attributs que vous devez déclarer dans Compte ?

Faire un schéma UML des classes comme vu en cours avec : noms des classes, noms des attributs, flèches « est-un ».

Modifier le programme banqueArrayList de EX-03 avec la nouvelle classe mère Compte.

On utilisera le modificateur d'accès protected pour les attributs de la classe Compte.

Programmer une méthode créditer et débiter dans chaque classe Compte.

Dans la méthode debiter des comptes courants, prendre en compte le fait qu'il y a un seuil de découvert autorisé.

Pour la méthode crediter des comptes epargnes, prendre en compte le fait qu'il y a un taux d'intérêt à appliquer au montant crédité.

III. Gestion d'un volailler

Un éleveur de volailles reçoit d'un fournisseur de jeunes canards et de jeunes poulets qu'il élève jusqu'à ce qu'ils aient la taille suffisante à leur commercialisation.

Une volaille est caractérisée par son poids et un numéro d'identification reporté sur une bague qu'elle porte sur sa patte pour des raisons de traçabilité.

Les volailles arrivent à l'élevage à l'âge de deux semaines. Elles sont baguées et enregistrées.

Le prix au kilo du canard et celui du poulet sont deux prix différents, exprimés en "Euros par kilo".

Le prix au kilo est le même pour tous les individus de la même espèce (c'est à dire les canards ou les poulets). Ce prix varie tous les jours.

A l'avenir, il se peut que le prix au kilo des poulets et des canards se calcule différemment en fonction des taxes.

Le prix de vente d'une volaille en particulier est égal à son poids actuel multiplié par son prix au kilo.

Le poids à partir duquel on abat les bêtes est différent pour les canards et les poulets, mais c'est le même pour tous les poulets et le même pour tous les canards.

Ecrivez les classes nécessaires. Il faut pouvoir enregistrer les prix du jour, les poids d'abattage, le poids d'une volaille donnée.

Ecrivez une classe Elevage permettant de représenter l'ensemble des animaux de l'élevage au moyen d'un tableau dynamique (ArrayList).

Des méthodes doivent permettre de sélectionner les animaux à abattre et d'évaluer le prix obtenu pour ces animaux qui doivent être ensuite supprimés du tableau.

Il faut également pouvoir enregistrer les jeunes animaux qui arrivent.

On vous donne ci-dessous l'exemple du main() d'une classe Ferme qui utilise ces classes.

```
package volailler;

public class Ferme {

	//-------------------------------------------------
	public static void main(String[] args){
		// Creer et ajouter quelques animaux
		Elevage laFerme = new Elevage();
```

```
            for    (int i=0; i<15; i++){
                  // Parametre du constructeur de Poulet : poids et
identifiant
                  laFerme.ajouter(new Poulet(0.250,150+i));
            }

            for(int i=0; i<15; i++){
                  laFerme.ajouter(new Canard(0.250,380+i));
            }

            for (int i=0; i<10; i++){
                  laFerme.ajouter(new Poulet(0.250,700+i));
            }
            laFerme.ajouter(new Canard(0.750,825));

            // Changer le poids de quelques animaux
            // 1er paramètre de changePoids : identifiant
            // 2ème paramètre : nouveau poids
            for (int i=0; i<8; i++){
                  laFerme.changePoids(155+i,1.3);
                  laFerme.changePoids(382+i,1.55);
            }

            // Afficher l'etat du volailler
            laFerme.ecrire();

            System.out.println("liste des animaux a abattre: ");
            System.out.println(laFerme.afficherBetesAAbattre());

            System.out.println("Valeur des animaux a abattre: ");
            System.out.println(laFerme.evaluerBetesAAbattre());
```

```
            laFerme.envoyerALAbattoir(); // Retourne aussi un tableau avec
les betes abattues
            laFerme.ecrire();

            System.out.println("Valeur des animaux a abattre: ");
            System.out.println(laFerme.evaluerBetesAAbattre());
      }
}
```

```
------------------------
Exécution du main () :
(affichage de identité / poids  prix)
150 0.25 0.25
151 0.25 0.25
152 0.25 0.25
153 0.25 0.25
154 0.25 0.25
155 1.3 1.3
156 1.3 1.3
157 1.3 1.3
158 1.3 1.3
159 1.3 1.3
160 1.3 1.3
161 1.3 1.3
162 1.3 1.3
163 0.25 0.25
164 0.25 0.25
380 0.25 0.3
381 0.25 0.3
382 1.55 1.8599999999999999
383 1.55 1.8599999999999999
384 1.55 1.8599999999999999
385 1.55 1.8599999999999999
386 1.55 1.8599999999999999
387 1.55 1.8599999999999999
388 1.55 1.8599999999999999
389 1.55 1.8599999999999999
390 0.25 0.3
391 0.25 0.3
392 0.25 0.3
393 0.25 0.3
394 0.25 0.3
700 0.25 0.25
701 0.25 0.25
```

```
702 0.25 0.25
703 0.25 0.25
704 0.25 0.25
705 0.25 0.25
706 0.25 0.25
707 0.25 0.25
708 0.25 0.25
709 0.25 0.25
825 0.75 0.8999999999999999
Valeur des animaux a abattre:
25.279999999999998
++++++++++++++++++++++++++++++++++++++++150 0.25 0.25
151 0.25 0.25
152 0.25 0.25
153 0.25 0.25
154 0.25 0.25
825 0.75 0.8999999999999999
709 0.25 0.25
708 0.25 0.25
707 0.25 0.25
706 0.25 0.25
705 0.25 0.25
704 0.25 0.25
703 0.25 0.25
163 0.25 0.25
164 0.25 0.25
380 0.25 0.3
381 0.25 0.3
702 0.25 0.25
701 0.25 0.25
700 0.25 0.25
394 0.25 0.3
393 0.25 0.3
392 0.25 0.3
391 0.25 0.3
390 0.25 0.3
Valeur des animaux a abattre:
0.0
```

1. Quelles méthodes sont polymorphes ……..
2. Est-ce que vous avez mis des instructions dans la méthode getPrix() de la classe Volaille ?

   …….

   Est-ce logique ? ……...
3. Quelles méthodes sont surchargées ? …….

Vocabulaire

- Surcharge : même nom mais entête différente dans la même classe = overloading
- Redéfinition = même entête de méthode dans une sous-classe = overriding
- Type déclaré vs. type réel

IV. Gestion d'une société de secourisme (OPTIONNEL)

Une société médicale souhaite informatiser la gestion de ses véhicules de secours apportés par les secouristes lorsqu'ils sont appelés en cas d'accident.

Vehicules

Chaque véhicule est caractérisé par un identifiant unique.
Les identificateurs commençent à partir de 100.
Parmi les véhicules de secours que gère la société il y a des engins et des PC (Poste de Commandement).
Un PC (Poste de Commandement) sert de support pour l'organisation des secours. Il reste sur place durant l'intervention et ne peut donc pas transporter de personne blessée.
A chaque engin correspond un nombre de personnes qu'il peut transporter.
Tous les engins permettent de transporter au moins une personne.
Un VA (Véhicule d'Assistance) permet de transporter 1 personne
Un VS (Véhicule de Secours) permet de transporter 2 personnes
La société comporte actuellement 2 PC, 3 VS et 2 VA.
La société prévoit d'acheter dans les prochaines années d'autres types d'engins permettant notamment de transporter plus de deux personnes.
Les méthodes suivantes devront être définies (à vous de trouver dans quelle(s) classe(s)):

- affecter un numéro d'intervention et une distance en km par rapport à la société (pour tous les véhicules)
- embarquer (String nomPersonne1) pour tous les engins
- pour les VS : prévoir aussi une version de la méthode qui permette d'embarquer en même temps 2 personnes : embarquer (String nomPersonne1, String nomPersonne2)
- affecter un hopital de destination (uniquement pour les engins)
- calculer le cout
    - pour le PC, le cout est fixe : 1000 Euros
    - pour les engins, cela dépend :
        - VA : 300 Euros

- VS : le cout est de 600 Euros pour les distances supérieures à 50 km, sinon le cout est de 150 Euros par personne transportée

Paquetages

Vous définirez vos classes dans des paquetages vehicules et vehicules.engins
La classe SocieteMed sera définie dans le paquetage par défaut.
Vous devez mettre les instructions package et import nécessaires pour toutes les classes (y compris SocieteMed pour laquelle vous fournirez aussi attribut(s) et méthode(s) autres que main()).

Définir les classes nécessaires pour gérer les véhicules et la société.
Vous mettrez par défaut les attributs en private mais vous n'avez pas besoin d'écrire les accesseurs.
Cependant, dans le cas de classes héritées, on permettra aux sous-classes d'accéder directement aux attributs de la classe mère.

Questions sur ces classes

1. Donner un exemple de méthode polymorphe. Expliquez. ...
2. Donner un exemple de méthode surchargée. Expliquez. Il n'y en a pas ...
3. Donner un exemple de polymorphisme appliqué au type d'un objet passé en paramètre. Expliquez. ...

Source de la méthode main() de la classe SocieteMed :

```
public static void main (String args []) {
        SocieteMed s = new SocieteMed();

        // Créer les véhicules
        PC pc1 = new PC ();
        PC pc2 = new PC ();
        VS vs1 = new VS ();
        VS vs2 = new VS ();
        VS vs3 = new VS ();
        VA va1 = new VA ();
        VA va2 = new VA ();
```

```
        // Ajouter les véhicules à la société
        s.ajouterVehicule (pc1);
        s.ajouterVehicule (pc2);
        s.ajouterVehicule (vs1);
        s.ajouterVehicule (vs2);
        s.ajouterVehicule (vs3);
        s.ajouterVehicule (va1);
        s.ajouterVehicule (va2);

        // Affecter qq véhicules à une intervention
        int numeroIntervention = 17 ;
        int distance = 35 ;
        pc1.affecterIntervention (numeroIntervention, distance);
        vs1.affecterIntervention (numeroIntervention, distance);
        vs2.affecterIntervention (numeroIntervention, distance);
        va2.affecterIntervention (numeroIntervention, distance);

        // Lors de l'intervention, embarquer quelques personnes
        vs1.embarquer("Dupont");
        vs2.embarquer("Martin", "Durand");

        // Afficher les informations sur tous les véhicules
        System.out.println (s);

        // Calculer le cout de l'intervention
        System.out.println ("Cout total de la journée : " +
s.calculerCout());
  }
```

Execution de la méthode main() de la classe SocieteMed :

```
SocieteMed [vehicules=[
PC [id=100, numeroIntervention=17, distance=35],
```

```
PC [id=101, numeroIntervention=0, distance=0],

VS [nomPersonne2=null, nbPlaces=2, hopital=null, nomPersonne1=Dupont,
nbPersonnesTransportees=1, id=102, numeroIntervention=17, distance=35],
VS [nomPersonne2=Durand, nbPlaces=2, hopital=null, nomPersonne1=Martin,
nbPersonnesTransportees=2, id=103, numeroIntervention=17, distance=35],
VS [nomPersonne2=null, nbPlaces=2, hopital=null, nomPersonne1=null,
nbPersonnesTransportees=0, id=104, numeroIntervention=0, distance=0],

VA [nbPlaces=1, hopital=null, nomPersonne1=null,
nbPersonnesTransportees=0, id=105, numeroIntervention=0, distance=0],
VA [nbPlaces=1, hopital=null, nomPersonne1=null,
nbPersonnesTransportees=0, id=106, numeroIntervention=17, distance=35]]]

Cout total de la journée : 1750.0
```

V. Questions sur l'héritage

Répondre aux questions dans un document (par exemple googledoc) comme vous faites pour les autres exercices.
Programmer les classes sous Eclipse et les tester.

Exercice 1

1) Quelle est la classe mère de toutes les classes en Java ?

2) Est-ce possible de définir une classe C1 qui hérite des attributs d'une classe C2 sans hériter de ses méthodes ? Si oui expliquer comment.

3) En Java, une classe peut-elle hériter de plusieurs classes ?
Si oui, comment ?

4) Soit le programme suivant et son exécution

```
        public class C {
                int i ;
                public static void main(String[] args) {
a                       C o = new C1 ();
b                       ((C1)o).i = 1 ;
c                       ((C)o).i = 2;
d                       System.out.println(o.i);
e                       System.out.println(((C)o).i);
f                       System.out.println(((C1)o).i);
                }
```

```
}

g class C1 extends C {
h       int i ;
}

EXECUTION:
2
2
1
```

Pour chaque ligne numérotée de a à h : expliquer ce qui se passe à l'exécution

Exercice 2

On vous donne de la documentation en annexe 1.

1) Que veut dire le mot clé final devant une déclaration de classe ?
2) Que veut dire le mot clé final devant une déclaration de méthode ?

3) Définir une classe Véhicule qui comporte un attribut entier x et une procédure init () qui initialise l'attribut à -1. *
Faire en sorte que cette méthode ne puisse pas être redéfinie dans les sous-classes.

4) Définir une classe Voiture qui étend la classe Véhicule.
Vous ajouterez un attribut boolean volantDroite ;
Faire en sorte que les développeurs ne puissent pas définir de sous-classe de la classe Voiture.

Annexe 1

Writing Final Classes and Methods

You can declare some or all of a class's methods *final*.

You use the `final` keyword in a method declaration to indicate that the method cannot be overridden by subclasses. The `Object` class does this—a number of its methods are `final`.
You might wish to make a method final if it has an implementation that should not be changed and it is critical to the consistent state of the object.

Methods called from constructors should generally be declared final. If a constructor calls a non-final method, a subclass may redefine that method with surprising or undesirable results.

Note that you can also declare an entire class final. A class that is declared final cannot be subclassed.