POO
Java
2020

PROGRAMMATION ORIENTÉE OBJET (POO)

Application au langage Java

Travaux pratiques

POO

Série N° 2

Classes, Classes abstraites et Interfaces - Héritage

Java

Youssouf EL ALLIOUI

y.elallioui@usms.ma

Exercice 1. Classe Employé

Définir une classe *Employé* caractérisée par les attributs : *matricule*, *nom*, *prénom*, *date de naissance*, *date d'embauche* et *salaire*.

- Définir les *getters* et les *setters* des différents attributs.
- Définir un constructeur sans paramètre et un deuxième avec paramètre afin d'initialiser les attributs par des valeurs fournies.
- Définir une méthode *age* ( ) Qui retourne l'âge de l'employé.
- Définir une méthode *ancienneté* ( ) Qui retourne le nombre d'années d'ancienneté de l'employé.
- Définir une méthode *augmentationDuSalaire*( ) qui augmente le salaire de l'employé en prenant en considération l'ancienneté :

$$\begin{cases} \text{Si Ancienneté} < 5 \text{ ans, alors on ajoute } 2\% \\ \text{Si Ancienneté} < 10 \text{ ans, alors on ajoute } 5\% \\ \text{Sinon, on ajoute } 10\% \end{cases}$$

- Définir la méthode *afficherEmployé*( ) qui affiche les informations de l'employé comme suit :

> *Matricule : […], Nom complet : [NOM Prénom], Âge : […], Ancienneté : […], Salaire : […]*

Le nom doit être affiché en majuscule. Pour le prénom, la première lettre doit être en majuscule, les autres en minuscule.

Écrivez un programme de test pour la classe Employé.

Exercice 2. Représentation d'un point dans l'espace

Réaliser une classe *Point* permettant de représenter un point dans l'espace. Chaque point sera caractérisé par un *nom* (de type *String*) et ses cordonnées x, y et z (sont tous de type *double*). On prévoira :

- Un constructeur recevant en arguments le nom et les cordonnées d'un point,
- Une méthode *affiche* ( ) imprimant, en fenêtre console, le nom du point et ses cordonnées,
- Une méthode *translate* ( ) effectuant une translation définie par la valeur de ses arguments.

- Une méthode $distance$ ( ) calculant la distance entre le point en question et un autre point passé en paramètre.

Écrire un programme de test pour :

- Créer un point,
- Afficher ses caractéristiques,
- Le déplacer et l'afficher à nouveau.
- Calculer la distance entre ce point et un autre point passé en paramètre.

Exercice 3. Implémentation d'une classe Livre

On demande d'implémenter une classe $Livre$ ($réf$, $titre$, $liste\ des\ auteurs$, $année$ et la $maison\ d'édition$) permettant d'offrir les possibilités suivantes :

- Définir la méthode $toString$ () pour pouvoir afficher les caractéristiques d'un $Livre$.
- Des accesseurs et modificateurs, $getter$ et $setter$, permettant de gérer les différentes informations du livre.
- Une fonction $boolean$, $chercherAuteur$ ( ), permettant de déterminer si le livre est écrit par un auteur donné en paramètre.
- Une fonction $boolean$, $chercherTheme$ ( ), permettant de déterminer si le $Livre$ traite un thème bien déterminé, donné sous forme d'une chaîne de caractères, à rechercher dans le titre du $Livre$.

Exercice 4. Premier exemple d'héritage

Écrivez une classe $Bâtiment$ avec les attributs nom et adresse.

- La classe $Bâtiment$ doit disposer de deux constructeurs :
  - Sans paramètre
  - Avec paramètre
- La classe $Bâtiment$ doit contenir des accesseurs et des modificateurs pour atteindre les différents attributs.
- La classe $Bâtiment$ doit contenir une méthode $toString$ () donnant une représentation du Bâtiment.
- Écrire une classe $Maison$ héritant de $Bâtiment$ qui dispose :
  - D'un attribut $nbrePieces$: Le nombre de pièces de la maison.
  - De deux constructeurs (avec et sans paramètres)
- La classe $Maison$ doit contenir des getters et des setters pour les différents attributs. La classe $Maison$ doit contenir une méthode $toString$ () donnant une représentation de la Maison.

Écrivez un programme de test afin de tester les deux classes.

Exercice 5. Attribut statique et héritage

Un compte bancaire possède un code et un solde qui peut être positif (compte créditeur) ou négatif (compte débiteur).

- Chaque compte est caractérisé par un code incrémenté automatiquement le code et le solde d'un compte sont accessibles via les modificateurs et accesseurs dédiés.
- À sa création, un compte bancaire a un solde nul et un code incrémenté.
- Il est aussi possible de créer un compte en précisant son solde initial.
- Utiliser son compte consiste à pouvoir y faire des dépôts et des retraits. Pour ces deux opérations, il faut connaître le montant de l'opération.
- L'utilisateur peut aussi consulter le solde de son compte par la méthode $toString()$.

Un compte $Epargne$ est un compte bancaire qui possède en plus de l'attribut $Taux\ d'Intérêt = 5\%$, une méthode $calculIntérêt()$ qui permet de mettre à jour le solde en tenant compte des intérêts.

Un $CompteEpargne$ est un compte bancaire pour lequel chaque opération de retrait et de versement vaut 20 DH.

Travail à faire :

- Définir la classe $Compte$.
- Définir la classe $CompteEpargne$.
- Définir la classe $ComptePayant$.
- Créer un programme permettant de tester ces classes avec les actions suivantes :
  - Créer une instance de la classe $Compte$, une autre de la classe $CompteEpargne$ et une instance de la classe $ComptePayant$.
  - Faire appel à la méthode $déposer()$ de chaque instance pour déposer une somme quelconque dans ces comptes.
  - Faire appel à la méthode $retirer()$ de chaque instance pour retirer une somme quelconque de ces comptes.
  - Faire appel à la méthode $calculInterêt()$ du compte Épargne.
  - Afficher les soldes des 3 comptes.

Exercice 6. Classe abstraite

Soit la classe abstraite $Employé$ caractérisée par les attributs $matricule$, nom, $prénom$, et $date\ de\ naissance$.

La classe $Employé$ doit disposer des méthodes suivantes :

- Deux constructeurs,
- La méthode $tostring()$,
- La méthode abstraite $getSalaire$ ( ).

Un $ouvrier$ est un $employé$ qui se caractérise par sa $date\ d'entrée$ à $la\ société$.

Tous les $ouvriers$ ont une valeur commune appelée $SMIG = 3000\ DH$

L'ouvrier a un salaire mensuel calculé à l'aide de la formule suivante :

$$salaire = SMIG + (Ancienneté\ en\ année) * 100$$

N.B : le salaire ne doit pas dépasser SMIG*2.

Un $cadre$ est un $employé$ qui se caractérise par un indice.

Le cadre a un salaire qui dépend de son indice :

- 1 : salaire mensuel $12000\ DH$
- 2 : salaire mensuel $14000\ DH$
- 3 : salaire mensuel $16000\ DH$
- 4 : salaire mensuel $18000\ DH$

Les $associées$ de la société sont aussi des $employés$ qui se caractérisent par un chiffre d'affaires, le bénéfice net et un pourcentage qui représente la part de contribution.

Le chiffre d'affaires est commun entre les associées.

Un $associé$ a un salaire annuel qui est égal à $x\%$ du $bénéfice\ net$ de la société :

$$Salaire = BN * x$$

Travail à faire :

- Créer la classe abstraite $Employé$.
- Créer la classe $Ouvrier$, la classe $Cadre$ et la classe $Associé$ et prévoir les Constructeurs et la méthode $toString$ ( ) de chaque classe.

- Implémenter la méthode $getSalaire()$ qui permet de calculer le salaire pour chacune des classes.

Exercice 7. Une application de gestion de vente, utilisation des interfaces

On veut créer une application de gestion de ventes d'un magasin qui vendent des articles de diverses natures.

1) **On aura deux interfaces**
   - Interface Produits qui se périme : l'interface pour les produits qui ont une date de fin de validité

   → **Méthodes :**
   - ❖ Date de péremption : Cette méthode retourne la date de fin de validité
   - ❖ Jours restants : cette méthode retourne le nombre de jours avant la fin de validité

   - Interface susceptible d'être vendue en solde

   → **Méthodes :**
   - ❖ lancer le solde : cette méthode baisse le prix du produit par le pourcentage donné
   - ❖ terminer le solde : cette méthode augmente le prix du produit par le pourcentage donné

2) **On aura une classe générale « Article ».**
   - Attributs :
     - ❖ prix d'achat : le prix pour lequel le supermarché achète le produit
     - ❖ prix de vente : le prix pour lequel le supermarché vend le produit
     - ❖ nom : le nom du produit
     - ❖ fournisseur : le nom du fournisseur du produit
   - deux constructeurs

   → **Méthodes :**
   - ❖ calculateur du taux du rendement (prix de vente- prix d'achat)
   - ❖ affichage des caractéristiques du produit sur l'écran (les prix, le nom, le fournisseur)

3) **On a deux classes dérivées des Articles**

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

   - La classe des articles électroménagers
     - ❖ Attributs supplémentaires : $voltage$ et $marque$
     - ❖ Deux constructeurs

   → **Méthode supplémentaire :**
   - ❖ Affichage des caractéristiques du produit sur l'écran (le prix, le nom, le fournisseur, marque, le voltage)

   Il faut implémenter les interfaces correspondantes à cette classe.

   - La classe des primeurs (fruits et légumes)
     - ❖ Attributs supplémentaires : $date\ de\ péremption$
     - ❖ Deux constructeurs

   → **Méthodes supplémentaires :**
   - ❖ Description des caractéristiques du produit sur l'écran (le prix, le nom, le fournisseur ; date de péremption)

   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 le magasin**
   - Attributs (trois tableaux) :

- ❖ Quantité en stock : tableau pour stocker la quantité par type d'article
- ❖ Les articles électroménagers en nombre de pièces
- ❖ Les primeurs en kilo
- ❖ Produits vendus
- ❖ Produits en stock

▪ Deux constructeurs

→ **Méthodes :**

- ❖ Vendre articles (spécifier le nom et la quantité)
- ❖ S'approvisionner (spécifier le nom et la quantité)
- ❖ Lister les articles en rupture en stock
- ❖ Lister les articles disponibles en stock
- ❖ Quantité en stock par article (fournir le nom comme paramètre)
- ❖ description de l'état du magasin
- ❖ calculateur du taux de rendement

Travail à faire :

- Écrire le code Java des interfaces et des classes présentées ci-dessus.
- Créer une classe $Test$ qui crée un magasin, définit les articles à vendre, effectue le remplissage du stock et simule les achats.