Java : Partie 3
POO : Exercices

Mise à jour d'Août 2013

Auteur : Chouaïb LAGHLAM

...

Sommaire

Préalable

- Dans cette partie «POO», l'affichage, dans les différents exercices, se fait sur la **console** (Une fenêtre Dos sous Windows).
- Créez dans l'espace de travail (workspace) « **Java_010_EspaceTravailCours**», et dans le projet nommé « **prj_Java_020_POO**», un package nommé «**pack_POO_Exos**».
- Chaque exercice vous demandera de créer des classes Java (sans méthode main) et vous devez les tester dans une classe dotée de la méthode main et nommée du nom de l'exercice.

  Exemple :

  Exercice 500 :

  - Il y aura la classe java : Exercice 00500.java avec la méthode main,
  - Il y aura les autres classes java : demandées dans l'exercice et non dotées de la Méthode main

Exercice N° 00500 : ma première classe

Écrire une classe « Point » permettant de représenter un point dans un Plan.
Chaque point sera caractérisé par un nom de type char, une abscisse (de type double) et une ordonnée (de type double).
On précisera :

- ➔ Les attributs en public,
- ➔ Un constructeur recevant en arguments le nom, l'abscisse et l'ordonnée du point,
- ➔ Une méthode « affiche » imprimant (en fenêtre console) le nom du point et ses coordonnées,
- ➔ Une méthode « translate » effectuant une translation du point par 2 valeurs données en argument.

Pas de setteurs ni de getteurs ici.

Écrire un petit programme utilisant cette classe pour créer un point, en afficher les caractéristiques, le déplacer et en afficher à nouveau les caractéristiques.

Exercice N° 00510 : Getteurs et Setteurs

Reprendre la classe « Point » créée précédemment, mettez les attributs en private et dotez-là de getteurs et setteurs,

- ➔ Créez de nouveau un objet point,
- ➔ Testez que l'accès direct aux attributs est interdit,
- ➔ Modifiez les attributs de votre objet par les setteurs,
- ➔ Affichez les attributs de votre objet par les getteurs.

Exercice N° 00520 : Classe technique

Pour les besoins de plusieurs développeurs, vous êtes chargé de fabriquer une classe «**Calculatrice**» ;
Elle n'est pas dotée d'attributs mais des méthodes suivantes:

« **addition** » qui somme deux réels fournis et renvoie le résultat,
« **soustraction** qui soustrait un réel d'un autre réel fournis et renvoie le résultat,
« **division** » qui divise un réel par un autre réel fournis et renvoie le résultat,
« **multiplication** » qui multiplie un réel par un autre réel fournis et renvoie le résultat,
«**racineCarree** qui calcule la racine carrée d'un réel fourni et renvoie le résultat,

Pensez aux contrôles qui s'imposent,
Testez l'appel de toutes ces méthodes dans un programme : Il n'est pas question de créer un objet calculatrice pour invoquer les méthodes citées ci-dessus (vos méthodes doivent être statiques),

Exercice N° 00530 : Plusieurs constructeurs

Créez une classe «**compteBancaire**» :

- Attributs :
  - numéro de compte : chaîne,
  - date création : chaîne,
  - type compte : chaîne (courant, livret A, codevi, ….)
  - taux rémunération : décimal
- les attributs en public,
- pas de setteurs ni getteurs,
- un constructeur qui reçoit numéro, date et type sans le taux,
- un constructeur qui reçoit numéro, date, type et le taux,
- méthode « affiche » qui ne reçoit rien et qui affiche les infos sur le compte.

Testez la création :

- d'un objet compte c1 en appelant le 1er constructeur,
- d'un objet compte c2 en appelant le 2ème constructeur,
- afficher les informations sur l'objet c1.

Exercice N° 00540 : Constructeur, getteurs et setteurs

Reprenez la classe « **compteBancaire »**, mettez les attributs en private et dotez-là de getteurs et de setteurs,
Testez la création d'un objet c25 en appelant l'un des constructeurs,

Question : Que faut-il modifier dans les constructeurs pour respecter l'encapsulation ?

Exercice N° 00550 : Destructeur

On souhaite faire un traitement spécifique lorsqu'un objet est détruit.
Pour tester cette fonctionnalité, modifiez la classe «**compteBancaire »** pour lui ajouter un destructeur,
Ce destructeur se contentera d'afficher « je suis le destructeur de la classe compteBancaire **»,**
Dans votre classe Exercice00550, contenant la méthode main :

- ajouter la méthode «**creeCompte** » qui se contentera de créer un nouvel objet de type **compteBancaire** et elle se termine.
- Dans la méthode «main» :
  - Appeler la méthode **creeCompte** ,
  - Appeler le garbage collector,

Avez-vous le message du destructeur.

Exercice N° 00560 : membres statiques

Créez une classe «**TableRectangulaire** » (un meuble de type table) ainsi :

- ➔ Propriétés d'objets en private :
  - Largeur : entier court,
  - Longueur : entier court,
  - Hauteur : entier court,
  - Matériau : chaîne (en bois, en métal, en ….)
- ➔ Propriétés de classe (propriété statique) :
  - nbreTables : entier court,
- ➔ constructeur :
  il doit comptabiliser les tables au fur et à mesure qu'ils sont créées,
- ➔ getteurs et setteurs,
- ➔ méthode statique « **getNbreTables** » :
  - elle renvoie le nombre de tables créés actuellement en mémoire,
- ➔ un destructeur :
  - il décrémente le nombre de tables créées.

Testez par un programme, le bon fonctionnement de cette classe :

- ➔ créez 2 ou 3 tables et demandez d'afficher le nombre de tables créées,
- ➔ trouvez un moyen de provoquer l'exécution du destructeur et réaffichez le nombre de tables restantes.

Exercice N° 00570 : traduire un cas réel en application orientée objets

Outil de comparaison de coûts pour la société KIES

La société KIES intervient auprès de ses clients pour dépannage,

Le coût du déplacement est calculé en tenant compte de la distance entre l'agence dont relève le client et le site du client ; le tarif kilométrique retenu pour l'intervention dépend de la puissance du véhicule utilisé.

Le coût de la main-d'œuvre est déterminé à partir du coût horaire du technicien chargé de l'intervention et de la durée de celle-ci, toute heure commencée étant entièrement comptabilisée.

Le coût horaire du technicien est obtenu en majorant le coût horaire correspondant à son grade par un coefficient dépendant de son ancienneté dans la société :

– de 5 à 10 ans majoration de 5 %
– de 11 à 15 ans majoration de 8 %
– plus de 15 ans majoration de 12 %

La société KIES a décidé de développer un outil logiciel permettant de comparer le montant du contrat payé par le client et le coût total des interventions relatives à ce contrat. Cet outil sera développé à l'aide d'un langage orienté objets.

Certaines classes et méthodes, nécessaires à ce développement, ont été définies et sont fournies en annexe sous forme textuelle.

Travail à faire

1) Expliquer l'intérêt du mécanisme d'encapsulation et la différence d'utilisation entre une méthode privée et une méthode publique, dans le contexte de la programmation par les objets.

2) Codez en Java les classes suivantes (avec y ajoutant constructeur + getteurs + setteurs + destructeur) :

```
Intervention = classe
  privé
    numéro : entier
    date : date
    durée : entier
    tarifkm : réel/* tarif kilométrique retenu */
    technicien: Employé/* employé ayant effectué l'intervention */
  public
    fonction FraisKm(dist : réel) : réel
/*  La méthode FraisKm de la classe Intervention calcule les frais
kilométriques occasionnés par une intervention, la distance parcourue
étant passée en paramètre. */
    fonction FraisMo() : réel
/*  La méthode FraisMo calcule et retourne les frais de main-d'œuvre
occasionnés par une intervention. */
Fin classe Intervention
```

```
Employé = classe
privé
        numéro : entier
         nom :chaîne
        qualification :Grade
        dateembauche :chaîne
public
        fonction CoûtHoraire() : réel
                /*  La méthode CoûtHoraire détermine et retourne le coût horaire de
                l'employé en fonction de sa qualification et de son ancienneté */
Fin classe Employé
```

```
Grade = classe
  privé
    code : caractère
    libellé :chaîne
    taux : réel
  public
    fonction TauxHoraire() : réel
        /* La méthode TauxHoraire retourne le taux horaire spécifique du grade
        */
   debut
      retourner taux
   fin
Fin classe Grade
```

```
Contrat = classe
  privé
        numéro :entier
        date : chaîne
        cli :Client
        montantcontrat: réel    /* montant payé par le client */
        tabinterventions : tableau (1:500) de Intervention
              nbintervention :entier  /* nombre d'interventions réalisées pour le contrat et présentes dans
                                         tabinterventions */
  public
    fonction Montant() : réel
        /* La méthode Montant retourne le montant du contrat payé par le client
        */
    debut
      retourner montantcontrat
    fin
    fonction Écart() : réel
        /* La méthode Écart détermine et retourne l'écart entre le montant du
        contrat et le coût des interventions effectuées sur ce contrat */
        Fin classe Contrat
```

```
Client = classe
  privé
        numéro      :     entier
        nom    :     chaîne
        adresse :    chaîne
        codepos :    chaîne
        ville    :    chaîne
        nbkm    :    entier          /* distance du client à la société Kès en km */
  public
    fonction Distance() : réel
        /* La méthode Distance retourne la distance, en kilomètres, qui sépare le site du client de la société KIES
        */
        Debut
              retourner nbkm
        fin
Fin classe Client
```

Remarque :

Il existe une classe Java avec une méthode utilitaire qui renvoie le nombre entier entre deux dates.

3) Écrivez le code des méthodes suivantes :
   - **CoûtHoraire** de la classe Employé
   - **FraisMo** de la classe Intervention
   - **FraisKm** de la classe Intervention
   - **Ecart** de la classe Contrat

4) Créez la classe **Exercice00570** avec sa méthode main :

➔ Créez un objet de chaque classe créée si dessus,

➔ Testez vos méthodes en les appelant et en vérifiant que les calculs respectent les règles métiers citées dans l'énoncé.

Exercice N° 00580 : traduire un cas réel en application orientée objets

Stockage et manutention (extrait du cas Caen-Ouistreham sujet 2001)

Lorsqu'un navire entre dans le port de Caen-Ouistreham, pour décharger ou charger de la marchandise (fret) ou les deux, l'échange se fait au travers des infrastructures de stockage (silo, hangar, ...) disposées sur le quai.

Un fret correspond à la cargaison d'un navire. On doit savoir à tout instant la quantité courante de fret pour les opérations de chargement-déchargement. Pour ce qui suit, on ne s'intéresse qu'aux opérations de déchargement.

Une zone de stockage est susceptible de recevoir tout type de fret. Elle est caractérisée par une capacité de stockage disponible. Lorsque sa capacité de stockage est nulle, cela signifie que la zone de stockage est remplie.

Lors d'une opération de déchargement, on cherche sur le quai une zone de stockage. On décharge autant de fret que cette zone peut en recevoir puis, le cas échéant, on cherche une ou plusieurs autres zones pour décharger le reste de la cargaison.

Les classes NAVIRE, STOCKAGE et PORT présentées ci-dessous décrivent la modélisation en objet de cette activité.

```
CLASSE NAVIRE
privé
   noLloyds   : Chaîne              //Numéro du navire
   nomNavire  : Chaîne              // Nom du navire
   libelléFret   : Chaîne    // Libellé du fret
   qtéFret : Entier          // Quantité de fret restant à décharger, dite
quantité courante
public
   obtenirQtéFret() : Entier      //fonction d'accès sur la donnée qtéFret
   décharger(Donnée  qté : Entier) // Diminue la qté courante de qté
   estDéchargé() : Booléen // indique si la qté courante est nulle ou
non
   ...
```

```
CLASSE STOCKAGE
privé
   capaDispo : Entier      // capacité de stockage disponible
public
   obtenirCapaDispo() : Entier // retourne la capacité restant disponible
   stocker(Donnée qté : Entier) // stocke la quantité qté et met à jour
la capacité disponible
   estVide() : Booléen // indique si la zone de stockage est vide ou non
   estRemplie() : Booléen // indique si la zone de stockage est remplie
ou non
   ...
```

```
CLASSE PORT
privé
   tabStock : TABLEAU [1..20] Stockage
public
   déchargement (Donnée/Résultat unNavire : Navire)
   ...
```

Travail à faire

1) Codez en Java les classes ci-dessus,
2) Écrivez le code de la méthode **estDecharge** de la classe NAVIRE,
3) Écrivez le code de la méthode **decharger** de la classe NAVIRE,
   On admet que l'utilisation de la méthode est toujours possible, le contrôle de la quantité de fret disponible étant Fait hors de la méthode,

4) Écrivez le code de la méthode **dechargement** de la classe PORT. Chaque bateau parcourt les zones de stockage dans l'ordre du tableau **tabStock**. Par simplification du problème, on supposer qu'il n'existe qu'un seul fret par navire.