MOOC Intro POO Java
Exercices semaine 7

Exercice 23 : Encore des employés

Une entreprise emploie deux catégories d'employés, des employés **permanents** et des employés **temporaires**. De plus, certains employés temporaires sont des **vendeurs** .

Un employé est caractérisé par les attributs suivants :

- son nom
- son statut (permanent ou temporaire)

Employés permanents

Un employé permanent est caractérisé par les attributs suivants:

- Le nombre de jours déjà travaillés dans le mois.
- Le salaire mensuel fixe (on supposera qu'il y a 20 jours oeuvrés par mois).
- Le nombre d'enfants.
- L'état civil.
- Le montant de la prime mensuelle pour enfants.

Le salaire des employés permanents est constitué du salaire mensuel fixe auquel s'ajoute la prime pour enfants (si l'employé est marié !). On multipliera la prime par le nombre d'enfants si l'employé a plusieurs enfants.

Le *salaire cumulé* pour un employé permanent est proportionnel au *nombre de jours* effectués dans le mois.

Employés temporaires

Les employés temporaires sont caractérisés par:

- Leur salaire horaire.
- Le nombre d'heures déjà effectuées dans le mois.

Les vendeurs sont caractérisés en plus par:

- Le volume de vente (en francs).
- La commission (pourcentage perçu sur la vente).

Les employés temporaires sont payés à l'heure. Ils perçoivent en plus une commission qui est fonction du volume des ventes s'ils sont vendeurs.

Pour les employés temporaires, le *salaire cumulé* est proportionnel aux *heures de travail* et aux *ventes* réalisées dans le mois.

Collection d'employés

Le personnel de l'entreprise est représenté sous la forme d'une collection, l'accès à un élément se faisant par le biais du nom. On suppose que les employés ont tous des noms différents.

Sur la collection représentant les employés, on souhaite pouvoir effectuer les opérations suivantes:

- Afficher le contenu d'une instance d'employé.
- Embaucher un employé dans une des catégories spécifiées.
- Licencier un employé.
- Muter un employé de l'entreprise dans une autre catégorie, tout en conservant le salaire qu'il a cumulé dans sa catégorie d'origine.

Le salaire cumulé dans la catégorie d'origine devra être calculé par le biais d'une méthode, `salaireCumule`, bien choisie. Il devra être converti de façon à être compatible avec sa nouvelle catégorie.

Par exemple, un employé permanent, marié avec un enfant, muté après 15 jours de travail avec un salaire mensuel de 5000 francs et une prime mensuelle pour enfant de 450 francs aura cumulé un salaire égal à:

15 * (5000 + 450)/20 = 4'087,5 francs

Si cet employé est muté dans la catégorie *Vendeur*, avec un salaire horaire de 25 francs et une commission de 10%, le salaire cumulé devra donc être converti soit en heures, soit en ventes, de sorte à être pris en compte dans la nouvelle catégorie.

Si par exemple on convertit l'ensemble du salaire cumulé en ventes, on devra avoir:

10% ventes = 4'087,5 -> ventes = 40'875 francs

L'employé sera donc muté dans la catégorie *Vendeur* avec un volume de ventes de 40'875 francs.

Spécifiez les classes, les attributs et les entêtes des méthodes permettant de rendre compte de l'ensemble des employés et de leur gestion dans la structure indiquée au dessus.

Attention: On impose que la spécification soit faite de telle sorte que la création d'une nouvelle catégorie d'employé puisse se faire **sans aucune modification dans les classes existantes**

- Il doit être clairement indiqué quelles classes et méthodes sont abstraites.
- Les modificateurs d'accès devront être clairement spécifiés.
- La description des classes doit contenir les variables d'instances ainsi que les entêtes des méthodes qui leurs sont spécifiques.
- Vos classes doivent éviter de dupliquer inutilement des méthodes et variables d'instances.
- Pour l'une au moins des classes specifiées, vous devrez indiquer le corps des méthodes nécessaires pour réaliser une mutation.

Exercice 24 : Déménagement

Vous déménagez bientôt et commencez à ranger vos affaires dans des cartons. Pour retrouver rapidement vos objets et savoir dans quel carton vous avez rangé tel ou tel objet, vous décidez d'écrire un programme orienté objet gardant trace de vos rangements.

Un carton porte un numéro. Il est rempli d'objets et/ou d'autres cartons remplis de la même façon. Le contenu d'un carton est l'ensemble des objets qu'il contient (y compris ceux contenus dans des cartons plus petits).

Vous considérerez :

- que chaque objet à un nom;
- qu'un déménagement est un *ensemble* de cartons.

Les fonctionnalités dont vous souhaitez disposer sont les suivantes :

1. mettre quelque chose (un objet ou un autre carton) dans un carton;
2. afficher le contenu d'un carton donné (le nom de tous ses objets);
3. ajouter un carton au déménagement;
4. afficher le contenu de tous les cartons du déménagement.
5. trouver le numéro du carton où se trouve un objet de nom donné : en ne retournant que le numéro du carton le plus externe (et un nombre négatif par exemple si l'objet ne se trouve pas dans les cartons);

Libre à vous maintenant d'implémenter une solution correcte (qui devra toutefois être compatible avec le programme principal fourni dans le fichier `Demenagement.java` (et reproduit ci-dessous). Veillez toutefois à ne pas avoir de duplication dans votre code et à respecter une bonne encapsulation. Pour commencer, vous pouvez dessiner le schéma de la hiérarchie de classes que vous imaginez, puis regroupez les méthodes/attributs communs dans les classes mères. Puis enfin lorsque la hiérarchie sera claire pour vous, vous pouvez coder le programme.

Voici le programme principal fourni :

```
class Demenagement
{

    public static void main(String[] args) {
        //On cree un demenagement qui pourra contenir 2 cartons principaux
        Demenagement demenagement = new Demenagement(2);

        // On cree et remplis ensuite 3 cartons
        // Arguments du constructeur de Box (Carton) :
        // argument 1 : nombre d'items (objets simple ou carton) que le carton peut contenir
        // argument 2 : numero du carton

        // le carton 1 contient des ciseaux
        Box box1 = new Box(1,1);
        box1.addItem(new SimpleItem("ciseaux"));

        // le carton 2 contient un livre
        Box box2 = new Box(1,2);
        box2.addItem(new SimpleItem("livre"));

        // le carton 3 contient une boussole
        // et un carton contenant une echarpe
        Box box3 = new Box(2,3);
        box3.addItem(new SimpleItem("boussole"));
        Box box4 = new Box(1,4);
        box4.addItem(new SimpleItem("echarpe"));
        box3.addItem(box4);

        //On ajoute les trois cartons au premier carton du demenagement
        Box box5 = new Box(3,5);
        box5.addItem(box1);
        box5.addItem(box2);
        box5.addItem(box3);

        //On ajoute un carton contenant 3 objets au demenagement
        Box box6 = new Box(3,6);
        box6.addItem(new SimpleItem("crayons"));
        box6.addItem(new SimpleItem("stylos"));
        box6.addItem(new SimpleItem("gomme"));


        //On ajoute les deux cartons les plus externes au demenagement
```

```
        demenagement.addBox(box5);
        demenagement.addBox(box6);

        //On imprime tout le contenu du demenagement
        demenagement.print();

        //On imprime le numero du carton le plus externe contenant l'objet "echarpe"
        System.out.println("L'echarpe est dans le carton numero " + demenagement.find("echarpe"));
    }
```

Il devrait produire un affichage ressemblant à ce qui suit :

```
Les objets de mon demenagement sont :
ciseaux
livre
boussole
echarpe
crayons
stylos
gomme
L'echarpe est dans le carton numéro 5
```

Exercice 25 : Expressions arithmétiques

Johnny C. est étudiant en première année à l'EPFL. Il doit écrire un programme Java simple permettant d'évaluer des expressions arithmétiques. Ces expressions sont constituées de nombres, d'additions et de multiplications; comme ceci :

```
 2 + 5 * 3
```

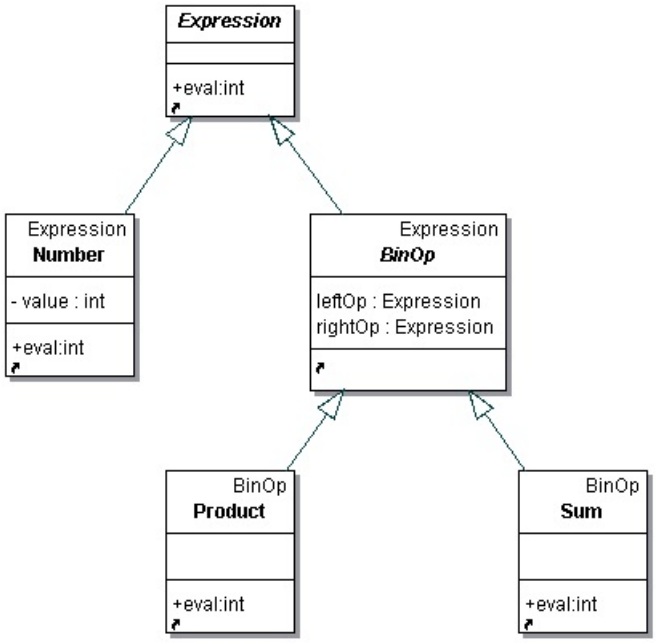
Johnny n'a pas encore très bien absorbé les concepts orientés objets permis par Java et il produit la solution suivante :

```
/** Structure de donnee pour les expressions*/
class Expression {
    public int type;
    public int value;
    public Expression leftOp;
    public Expression rightOp;

    public Expression(int type, int value, Expression leftOp, Expression rightOp) {
        this.type = type;
        this.value = value;
        this.leftOp = leftOp;
        this.rightOp = rightOp;
    }
}

/** Classe principale */
class Arith {

    /** Constantes pour representer les types*/
    public static final int TYPE_NUMBER = 1;
    public static final int TYPE_SUM = 2;
    public static final int TYPE_PROD = 3;

    public static void main(String [] args) {
        // construit l'expression 3 + 2 * 5
        Expression term = new Expression(TYPE_SUM, 0,
            new Expression(TYPE_NUMBER, 3, null, null),
            new Expression(TYPE_PROD, 0,
                new Expression(TYPE_NUMBER, 2, null, null),
                new Expression(TYPE_NUMBER, 5, null, null)));

        System.out.println(evaluate(term));
    }

    /** Evalue recursivement l'expression */
    public static int evaluate(Expression term) {
        switch (term.type) {
            case TYPE_NUMBER:
                return term.value;
            case TYPE_SUM:
                return evaluate(term.leftOp) + evaluate(term.rightOp);
            case TYPE_PROD:
                return evaluate(term.leftOp) * evaluate(term.rightOp);
            default:
                return 0;   //erreur, ne devrait jamais se produire

        }

    }

}
```

Le programme de Johnny fonctionne, mais est écrit dans un style "procédural" plutôt qu'orienté objet. Il peut donc, à cet égard, être amélioré.

1. Réflechissez à la solution de Johnny et essayez d'expliquer pourquoi elle n'est pas très bonne et comment elle pourrait être améliorée.
2. Ecrivez une version améliorée (commencez par lire l'intégralité des recommandations avant de commencer à programmer) :
    - Définissez une classe abstraite `Expression` qui déclarera une méthode abstraite `evaluate`.
    - Créez des classes distinctes pour chaque type d'expressions (`Number`, `Sum`, `Product`) et faites les hériter de `Expression`.
    - Vous pouvez éviter de la duplication inutile de code en créeant une classe abstraite intermédiaire `BinOp` qui contiendra les attributs `leftOp` et `rightOp` représentants respectivement les opérandes gauche et droit d'un expression binaire. Faites hériter `Sum` et `Product` de `Binop`.

- Adaptez la méthode `main` de la classe principale à votre nouvelle structure.



3. Pour comparer la solution originale de Johnny à la votre, faites évoluer votre programme dans le sens suivant :
   - Ajouter la possibilité de générer la représentation de votre expression sous la forme d'une chaîne de caractère. Par exemple si votre expression est le nombre `13`, cette fonctionalité doit vous retourner la chaîne `"13"`. Si c'est une `Sum` avec `12` et `13` comme `leftOp` et `rightOp` respectivement, la fonctionalité en question devra retourner la chaîne `"12 + 13"`.
   - Ajouter un nouveau type d'expression binaire : `Modulo` représentant des expressions binaires dont l'évaluation retourne le reste de la division entière de `leftOp` par `rightOp`.

Portez un regard critique à votre nouvelle version. Si vous avez dû faire du "copier-coller" de portions de votre code pour ajouter ces facilités, c'est qu'il y a encore des problèmes de conception. Essayer de faire en sorte qu'il n'y ait aucune duplication inutile de code.

Une fois satisfait de votre version, essayez d'ajouter les mêmes facilités au programme de Johnny.

Laquelle des deux versions a été la plus facile à modifier ?
