MOOC Intro POO Java
Exercices semaine 5

Exercice 15 : Variables statiques

Pour cet exercice, il ne vous est pas demandé de coder. Il s'agit d'indiquer l'affichage produit par le programme `AfficherCouleur` ci-dessous sans l'exécuter. La classe `FigureGeometrique` se trouve juste après.

```
class AfficherCouleur {
    public static void main(String[] args) {
        FigureGeometrique f1, f2, f3, f4;
        f1 = new FigureGeometrique("Rouge", "Noir");
        f2 = new FigureGeometrique("Bleu", "Blanc");
        f3 = new FigureGeometrique();
        f4 = new FigureGeometrique("Violet", "Orange");

        f2.changerCouleurContour("Noir");

        System.out.println(f3.couleurSurface);
        System.out.println(f3.couleurContour);

        System.out.println(f2.couleurContour);

        System.out.println(FigureGeometrique.nbFigures);
        System.out.println(f2.nbFigures);

        System.out.println(f1.nbFigContNoir);
        System.out.println(f4.nbFigSurfBlanc);
    }
}

class FigureGeometrique {
    String couleurContour = "Jaune";
    String couleurSurface = "Blanc";

    static int nbFigures = 0;
    static int nbFigContNoir = 0;
    static int nbFigSurfBlanc = 0;

    FigureGeometrique() {
    }

    FigureGeometrique(String coulSurf, String coulCont) {
        couleurSurface = coulSurf;
        couleurContour = coulCont;
        if (couleurContour.equals("Noir")) {
            nbFigContNoir++;
        }
        if (couleurSurface.equals("Blanc")) {
            nbFigSurfBlanc++;
        }
        nbFigures++;
    }

    void changerCouleurContour(String couleur) {
        couleurContour = couleur;
    }

    void changerCouleurSurface(String couleur) {
        couleurSurface = couleur;
    }
}
```

Exercice 16 : Boîtes aux lettres (avec de la pub)

Il s'agit ici de reprendre l'exemple de la boîte aux lettres vu il y a quelques séries (vous partirez du corrigé fourni dans le fichier fourno `Commercial.java`) et de l'enrichir en tenant compte des contraintes suivantes :

- il peut exister du courrier commercial;
- tout courrier commercial bénéficie d'une déduction sur son montant d'affranchissement;
- toutes les publicités sont du courrier commercial (déduction de 20% sur le montant d'affranchissement);
- certains colis sont des colis commerciaux (déduction de 15% sur le montant d'affranchissement).

En utilisant la notion d'interface compléter le programme fourni de sorte à tenir compte de ces nouvelles contraintes. Le programme devra pouvoir être testé avec la méthode `main` fournie (après avoir décommenté son corps). Vous devriez obtenir l'affichage suivant :

```
Le montant total d'affranchissement est de 48.4375
Publicité
        Poids : 1500.0 grammes
        Express : oui
        Destination : Les Moilles  13A, 1913 Saillon
        Prix : 12.0 CHF

Publicité
        Poids : 3000.0 grammes
        Express : non
        Destination : Ch. de l'impasse 1, 9999 Nowhere
        Prix : 12.0 CHF

Colis
        Poids : 7000.0 grammes
        Express : non
        Destination : Route de la rape 11, 1509 Vucherens
        Prix : 11.2625 CHF
        Volume : 25.0 litres
        Colis commercial

Colis
        Poids : 2500.0 grammes
        Express : oui
        Destination : Route de la Rameau 14b, 404 Notfound
        Prix : 13.175 CHF
        Volume : 21.0 litres
        Colis commercial

La boite contient 0 courriers invalides
```

Exercice 17 : Primes de risque

Cet exercice vous permettra de concevoir une hiérarchie de classes utilisant la notion d'interface. Il vous servira également de révision pour les notions d'héritage, de classes abstraites et de polymorphisme.

Le directeur d'une entreprise de produits chimiques souhaite gérer les salaires et primes de ses employés au moyen d'un programme Java.

Un employé est caractérisé par son nom, son prénom, son âge et sa date d'entrée en service dans l'entreprise.

Dans le fichier fourni `Salaires.java`, codez une classe abstraite `Employe` dotée des attributs nécessaires, d'une méthode abstraite `calculerSalaire` (ce calcul dépendra en effet du type de l'employé) et d'une méthode `getNom` retournant une chaine de caractère obtenue en concaténant la chaine de caractères "`L'employé `" avec le prénom et le nom.

Dotez également votre classe d'un constructeur prenant en paramètre l'ensemble des attributs nécessaires.

Calcul du salaire

Le calcul du salaire mensuel dépend du type de l'employé. On distingue les types d'employés suivants :

- Ceux affectés à la *Vente*. Leur salaire mensuel est le 20 % du *chiffre d'affaire* qu'ils réalisent mensuellement, plus 400 Francs.
- Ceux affectés à la *Représentation*. Leur salaire mensuel est également le 20 % du *chiffre d'affaire* qu'ils réalisent mensuellement, plus 800 Francs.
- Ceux affectés à la *Production*. Leur salaire vaut le *nombre d'unités* produites mensuellement multipliées par 5.
- Ceux affectés à la *Manutention*. Leur salaire vaut leur *nombre d'heures* de travail mensuel multipliées par 65 francs.

Codez dans votre fichier `Salaires.java` une hiérarchie de classes pour les employés en respectant les conditions suivantes :

- La super-classe de la hiérarchie doit être la classe `Employe`.
- Les nouvelles classes doivent contenir les attributs qui leur sont spécifiques ainsi que le codage approprié des méthodes `calculerSalaire` et `getNom`, en changeant le mot "employé" par la catégorie correspondante.
- Chaque sous classe est dotée de constructeur prenant en argument l'ensemble des attributs nécessaires.
- N'hésitez pas à introduire des classes intermédiaires pour éviter au maximum les redondances d'attributs et de méthodes dans les sous-classes

Employés à risques

Certains employés des secteurs *production* et *manutention* sont appelés à fabriquer et manipuler des produits dangereux.

Après plusieurs négociations syndicales, ces derniers parviennent à obtenir une prime de risque mensuelle.

Complétez votre programme `Salaires.java` en introduisant deux nouvelles sous-classes d'employés. Ces sous-classes désigneront les employés des secteurs *production* et *manutention* travaillant avec des produits dangereux.

Ajouter également à votre programme une interface pour les *employés à risque* permettant de leur associer une *prime mensuelle* fixe de 200.-.

Collection d'employés

Satisfait de la hiérarchie proposée, notre directeur souhaite maintenant l'exploiter pour afficher le salaire de tous ses employés ainsi que le salaire moyen.

Ajoutez une classe `Personnel` contenant une "collection" d'employés. Il s'agira d'une collection polymorphique d'`Employe` - regardez le cours si vous ne voyez pas de quoi il s'agit.

Si vous programmez votre collection au moyen d'un tableau de taille fixe, vous pourrez lui donner la taille maximale de 200 par exemple.

Définissez ensuite les méthodes suivantes à la classe `Personnel` :

- `void ajouterEmploye(Employe)`

  qui ajoute un employé à la collection.

- `void afficherSalaires()`

  qui affiche le salaire de chacun des employés de la collection.

- `double salaireMoyen()`

qui affiche le salaire moyen des employés de la collection.

Testez votre programme avec le main fourni :

```
class Salaires {
    public static void main(String[] args) {
        Personnel p = new Personnel();

        p.ajouterEmploye(new Vendeur("Pierre", "Business", 45, "1995", 30000));
        p.ajouterEmploye(new Representant("Léon", "Vendtout", 25, "2001", 20000));
        p.ajouterEmploye(new Technicien("Yves", "Bosseur", 28, "1998", 1000));
        p.ajouterEmploye(new Manutentionnaire("Jeanne", "Stocketout", 32, "1998", 45));
        p.ajouterEmploye(new TechnARisque("Jean", "Flippe", 28, "2000", 1000));
        p.ajouterEmploye(new ManutARisque("Al", "Abordage", 30, "2001", 45));

        p.afficherSalaires();
        System.out.println("Le salaire moyen dans l'entreprise est de " + p.salaireMoyen() + " francs.");
    }
}
```

Vous devriez obtenir quelque chose comme :

```
Le vendeur Pierre Business gagne 6400.0 francs.
Le représentant Léon Vendtout gagne 4800.0 francs.
Le technicien Yves Bosseur gagne 5000.0 francs.
Le manut. Jeanne Stocketout gagne 2925.0 francs.
Le technicien Jean Flippe gagne 5200.0 francs.
Le manut. Al Abordage gagne 3125.0 francs.
Le salaire moyen dans l'entreprise est de 4575.0 francs.
```

Exercice 18 : Analyse de conception

Pour cet exercice, vous n'aurez à nouveau rien à programmer. Une feuille et un crayon devraient suffire.

Un programmeur, encore débutant, décide de programmer un jeu de stratégie militaire. L'action se déroule dans un monde imaginaire peuplé par deux races: *elfes* et *nains*.

Les nains utilisent des *haches* et les elfes des *arcs*. Il y a différent types de haches et d'arcs qui sont plus ou moins efficaces. Les elfes et les nains peuvent être *cavalier* ou *magicien*.

Au cours du jeu, les joueurs disposent d'une armée composée de plusieurs unités et ont pour but de détruire l'armée ennemie.

Les unités sont caractérisées par leur race, leur points de vie restants et leur statut (mort ou en vie) ainsi que leur arme, et leur compétences (magicien ou cavalier).

Lors d'une première ébauche, notre programmeur modélise le jeu de la façon suivante (fichier fourni) :

```
class Unite {
    private boolean enVie;
    private int pointsDeVie;
}

class Hache {}

class Arc {}

interface Nain {
    public void frappeAvecHache();
}

interface Elfe {
    public void tireFleche();
}

interface Magicien {
    public void lanceSort();
}

interface Cavalier {
    public void chevauche();
}

class NainMagicien extends Unite implements Nain, Magicien {
    private int taille;
    private Hache hache;
    public void frappeAvecHache() {};
    public void lanceSort() {};
}

class NainCavalier extends Unite implements Nain, Cavalier {
    private int taille;
    private Hache hache;
    public void frappeAvecHache() {};
    public void chevauche() {};
}

class ElfeMagicien extends Unite implements Elfe, Magicien {
    private int poids;
    private Arc arc;
    public void tireFleche() {};
    public void lanceSort() {};
}

class ElfeCavalier extends Unite implements Elfe, Cavalier {
    private int poids;
    private Arc arc;
    public void tireFleche() {};
    public void chevauche() {};
}
```

- Il crée une classe `Unite` contenant les attributs `enVie` (oui ou non) et `pointsDeVie`.
- Il crée quatre interfaces `Elfe`, `Nain`, `Magicien` et `Cavalier` et y déclare les méthodes spécifiques aux catégories d'unités qu'elles représentent: des méthodes permettant aux elfes et nains d'utiliser leurs armes spécifiques, ainsi que des méthodes correspondant

aux capacités des magiciens (lancer des sorts) et des cavaliers (chevaucher).

De plus, soucieux de plonger le joueur dans l'univers de ses créatures, il ajoute un attribut `taille` à tous les nains et `poids` à tous les elfes. En effet les nains, qui sont petits, tiennent beaucoup à ce qu'on ne sous-estime pas leur taille et les elfes, qui sont très légers, n'aiment pas qu'on surestime leur poids. Ces attributs seront utilisés pendant le jeu afin que les unités puisse s'envoyer des railleries.

Aidez notre programmeur à répondre aux questions qu'il se pose :

- La classe `Unite` a-t-elle les bons modificateurs ?
- Comment supprimer la duplication des attributs `taille`, `hache` et de la méthode `frappeAvecHache()` dans les classes `NainMagicien` et `NainCavalier` ainsi que la duplication des attributs `arc`, `poids` et de la méthode `tireFleche()` dans les classes `ElfeMagicien` et `ElfeCavalier` ? Il ne faudra pas modifier la classe `Unite` ni créer de nouveaux types.
- Une fois la duplication de code précédente éliminée, est-il aussi possible d'éliminer la duplication des méthodes `lanceSort()` et `chevauche()` ? Toujours sans modifier la classe `Unite` ni créer de nouveaux types.
