MOOC Intro POO Java
Corriges semaine 5

Les corrigés proposés correspondent à l'ordre des apprentissages : chaque corrigé correspond à la solution à laquelle vous pourriez aboutir au moyen des connaissances acquises jusqu'à la semaine correspondante.

Exercice 15 : Variables statiques

Pour ce genre d'exercices, aidez-vous de petits schémas associés à vos objets et décrivant leur contenu, puis déroulez le programme pas à pas en mettant à jour le contenu des objets dans vos schémas.

L'affichage est:

```
Blanc
Jaune
Noir
3
3
1
0
```

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

Une version possible du corrigé est fournie ci-dessous.

```
/* Interface pour représenter le courrier commercial
 */

interface Commercial {

    double deduction();


}


/* Classe pour représenter le courrier
 */

abstract class Courrier {
    // retourne le montant n'ecessaire pour affranchir le courrier
    // en mode d'exp'edition normal

    // on va faire une chose tre`s vilaine parcequ'on ne connait pas les
    // m'ethodes abstraites : on va lui donner un corps arbitrairement
    // d'efini (car on ne sait pas la d'efinir proprement
    // a` ce niveau de la hi'erarchie
    abstract protected double affranchirNormal();
    // la bonne solution consiste a` d'eclarer cette m'ethode comme suit:
    // abstract protected double affranchirNormal();
    // lorsque vous aurez vu le cours 8, expliquez pourquoi...

    // les attributs (communs aux lettres et colis):
    protected double poids;
    protected boolean express;
    protected String adresse;

    // un constructeur possible pour la classe
    public Courrier(double poids, boolean express, String adresse) {
        this.poids = poids;
        this.express = express;
        this.adresse = adresse;
    }

    // retourne le montant n'ecessaire pour affranchir le courrier.
    // elle appelle affranchirNormal et retourne le double de ce montant
    // si le mode d'exp'edition est  express ('eviter la duplication du code
    // qui double le montant dans les m'ethodes affranchir-normal
    // des sous-classes)
    public double affranchir() {
        if (! valide())
        {
            return 0;
        }
        else
        {
            double total = affranchirNormal();
            if (express) {
                total *= 2;
            }
            return total;
        }
    }

    // un courrier est invalide si l'adresse de destination est vide
    // methode utilis'ee par Boite::affranchir et
    // Boite::courriersInvalides
    public boolean valide() {
        return adresse.length() > 0;
    }

    @Override
    public String toString() {
        String s = "";
        if (!valide())
        {
            s+= "(Courrier  invalide)\n";
```

```
        }
        s+= "   Poids : " + poids + " grammes\n";
        s+= "   Express : " + (express ? "oui" : "non") + "\n";
        s+= "   Destination : " + adresse + "\n";
        s+= "   Prix : " + affranchir() + " CHF\n";
        return s;
    }

}

/* Une classe pour repr'esenter les lettres
 */

class Lettre extends Courrier {

    //attributs sp'ecifiques aux lettres:
    protected String format = "";

    public Lettre(double poids, boolean express, String adresse, String format){
        super(poids, express, adresse);
        this.format = format;
    }

    // red'efinit affranchirNormal()
    protected double affranchirNormal() {
        double montant = 0;
        if (format.equals("A4")){
            montant = 2.0;
        } else {
            montant = 3.5;
        }
        montant += poids/1000.0;
        return montant;
    }

    // inutile de red'efinir la méthode valide() pour les  lettres

    @Override
    public String toString() {
        String s = "Lettre\n";
        s += super.toString();
        s += "  Format : " + format + "\n";
        return s;
    }

}
/* Une classe pour repr'esenter les publicit'es
 */

class Publicite extends Courrier implements Commercial {

    public Publicite(double poids, boolean express, String adresse){
        super(poids, express, adresse);
    }

    // redéfinit affranchirNormal()
    protected double affranchirNormal() {
        return poids/1000.0 * 5.0;
    }


    // inutile de red'efinir la méthode valide() pour les  publicités

    @Override
    public String toString() {
        String s = "Publicité\n";
        s += super.toString();
        return s;
    }

    public double deduction() {
        return 0.2;
    }

    @Override
    public double affranchir()
        {
```

```
            double montant = super.affranchir();
            montant -= montant * deduction();
            return montant;
        }

}

/* Une classe pour repr'esenter les colis
 */
class Colis extends Courrier {

    //attributs sp'ecifiques aux colis:
    protected double volume;

    public Colis(double poids, boolean express, String adresse, double volume) {
        super(poids, express, adresse);
        this.volume = volume;
    }

    // redéfinit affranchirNormal();
    protected double affranchirNormal() {
        // affranchit les colis selon une formule pr'ecise
        return 0.25 * volume + poids/1000.0;
    }

    // ici il faut red'efinir  (sp'ecialiser) la re`gle de validit'e des colis
    // un colis est invalide s' il  a une mauvaise adresse
    //ou depasse un certain volume
    public boolean valide(){
        return (super.valide() && volume <= 50);
    }

    @Override
    public String toString() {
        String s = "Colis\n";
        s += super.toString();
        s += "  Volume : " + volume + " litres\n";
        return s;
    }


}


/* Une classe pour les colis commerciaux
 */
class ColisCommercial extends Colis implements Commercial {

    public ColisCommercial(double poids, boolean express, String adresse, double volume) {
        super(poids, express, adresse, volume);
    }

    // redéfinit affranchir() pour appliquer la deduction
    public double affranchir() {
        // affranchit le colis en faisant appel à affranchir_normal de
        // la superclasse puis en lui appliquant la déduction
        double montant = super.affranchir();
        montant -= montant * deduction();
        return montant;
    }

    public double deduction() {
        return 0.15;
    }

    @Override
    public String toString() {
        String s = super.toString();
         s += " Colis commercial\n";

        return s;
    }
}

/*   Une classe pour repr'esenter la boite aux lettre
 */
```

```java
class Boite {

    private Courrier[] contenu;
    private int index;

    // constructeur
    public Boite(int max) {
        contenu = new Courrier[max];
        index = 0;
    }

    // la méthode demand'ee
    public double affranchir() {
        double montant = 0.0;
        for(int i=0; i < index; ++i){
            Courrier c = contenu[i];
            montant += c.affranchir();
        }
        return montant;
    }

    public int size() {
        return index;
    }

    public Courrier getCourrier(int index) {
        if (index < contenu.length)
            return contenu[index];
        else
            return null;
    }

    // autre m'ethode demandée dans l'interface
    // d'utilisation de la classe
    public int courriersInvalides() {
        int count = 0;
        for (int i = 0; i < index; i++) {
            if (!contenu[i].valide())
                count++;
        }
        return count;
    }

    // difficile de fonctionner sans
    public void ajouterCourrier(Courrier  unCourrier) {
        if (index < contenu.length){
            contenu[index] = unCourrier;
            index++;
        } else {
            System.out.println("Impossible d'ajouter un nouveau courrier. Boite pleine !");
        }
    }

    public void afficher() {
        for (int i = 0; i < index; i++) {
            System.out.println(contenu[i]);
        }
    }

}


// PROGRAMME PRINCIPAL (non demandé)
class PosteCommercial {

    public static void  main(String args[]) {
        //Cr'eation d'une boite-aux-lettres
        Boite boite = new Boite(30);

        //Creation de divers courriers/colis..

        Publicite pub1 = new Publicite(1500, true, "Les Moilles  13A, 1913 Saillon");
        Publicite pub2 = new Publicite(3000, false, "Ch. de l'Impasse 1, 9999 Nowhere");

        ColisCommercial colisCom1 = new ColisCommercial(7000, false, "Route de la rape 11, 1509 Vucherens", 25);
        ColisCommercial colisCom2 = new ColisCommercial(2500, true, "Route du Rameau 14b, 404 Notfound", 21);
```

```
        boite.ajouterCourrier(pub1);
        boite.ajouterCourrier(pub2);
        boite.ajouterCourrier(colisCom1);
        boite.ajouterCourrier(colisCom2);


        System.out.println("Le montant total d'affranchissement est de " +
                           boite.affranchir());
        boite.afficher();

        System.out.println("La boite contient " + boite.courriersInvalides()
                           + " courriers invalides");
    }
}
```

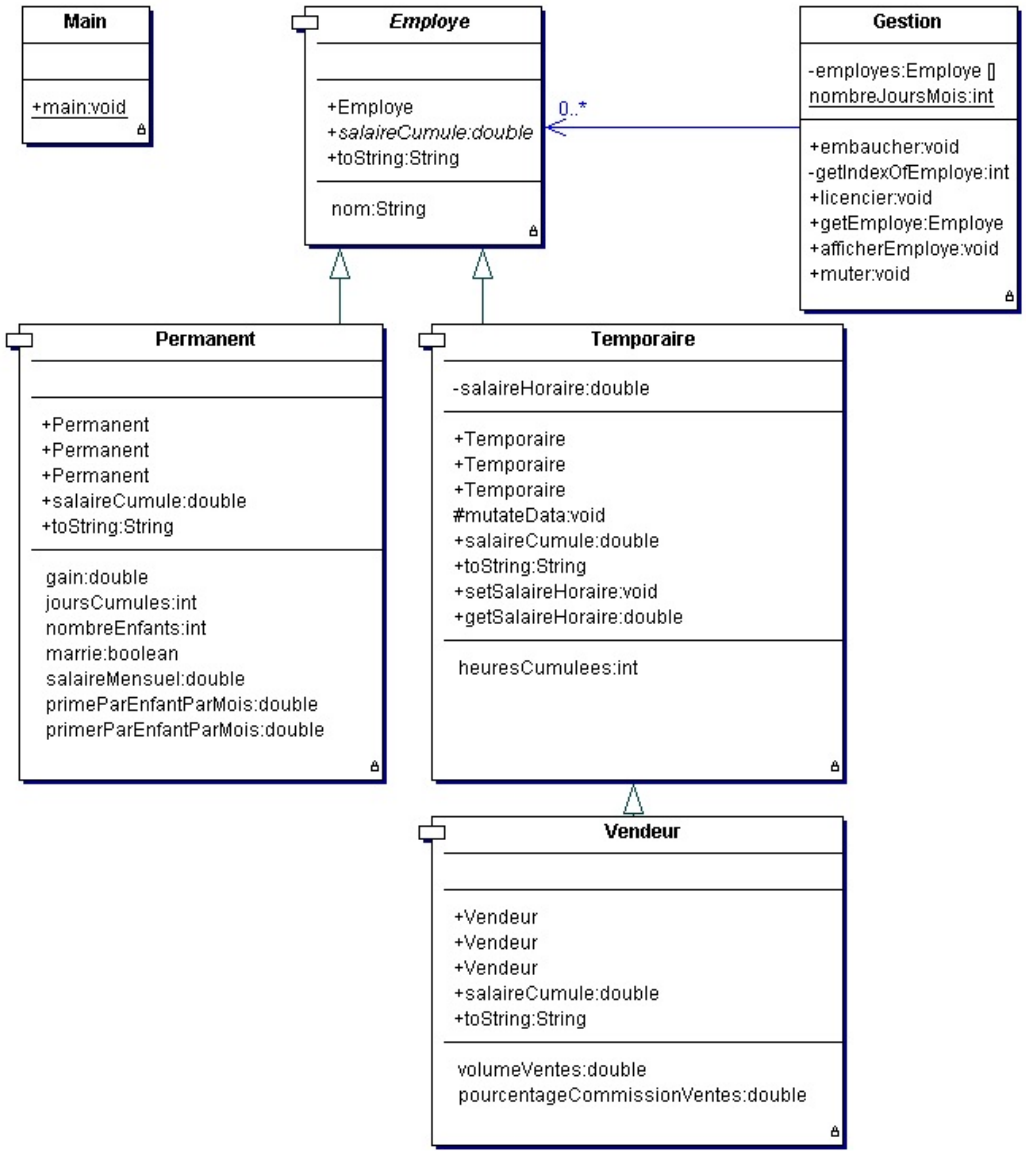
Exercice 17 : Primes de risques

```
/* ********************************************************************
 *  La classe Employe
 * ********************************************************************/
abstract class Employe {
    private String nom;
    private String prenom;
    private int age;
    private String date;


    public Employe(String prenom, String nom, int age, String date) {
        this.nom = nom;
        this.prenom = prenom;
        this.age = age;
        this.date = date;
    }

    public abstract double calculerSalaire();

    public String getTitre()
        {
            return "L'employé " ;
        }

    public String getNom() {
        return getTitre() + prenom + " " + nom;
    }
}

/* ********************************************************************
 *  La classe Commercial (regroupe Vendeur et Représentant)
 * ********************************************************************/
abstract class Commercial extends Employe {
    private double chiffreAffaire;

    public Commercial(String prenom, String nom, int age, String date,
                double chiffreAffaire) {
        super(prenom, nom, age, date);
        this.chiffreAffaire = chiffreAffaire;
    }

    public double getChiffreAffaire()
        {
            return chiffreAffaire;
        }

}

/* ********************************************************************
 *  La classe Vendeur
 * ********************************************************************/
class Vendeur extends Commercial {
    private final static double POURCENT_VENDEUR = 0.2;
    private final static int BONUS_VENDEUR = 400;

    public Vendeur(String prenom, String nom, int age, String date,
            double chiffreAffaire) {
        super(prenom, nom, age, date, chiffreAffaire);
    }

    public double calculerSalaire() {
        return (POURCENT_VENDEUR * getChiffreAffaire()) + BONUS_VENDEUR;
    }

    public String getTitre()
        {
            return "Le vendeur ";
        }

}

/* ********************************************************************
 *  La classe Représentant
 * ********************************************************************/
```

```java
class Representant extends Commercial {
    private final static double POURCENT_REPRESENTANT = 0.2;
    private final static int BONUS_REPRESENTANT = 800;

    public Representant(String prenom, String nom, int age, String date, double chiffreAffaire) {
        super(prenom, nom, age, date, chiffreAffaire);
    }

    public double calculerSalaire() {
        return (POURCENT_REPRESENTANT * getChiffreAffaire()) + BONUS_REPRESENTANT;
    }

    public String getTitre()
        {
            return "Le représentant ";
        }
}

/* *********************************************************************
 *  La classe Technicien (Production)
 * ********************************************************************/
class Technicien extends Employe {
    private final static double FACTEUR_UNITE = 5.0;
    private int unites;

    public Technicien(String prenom, String nom, int age, String date, int unites) {
        super(prenom, nom, age, date);
        this.unites = unites;
    }

    public double calculerSalaire() {
        return FACTEUR_UNITE * unites;
    }

    public String getTitre()
        {
            return "Le technicien ";
        }
}

/* *********************************************************************
 *  La classe Manutentionnaire
 * ********************************************************************/
class Manutentionnaire extends Employe {
    private final static double SALAIRE_HORAIRE = 65.0;
    private int heures;

    public Manutentionnaire(String prenom, String nom, int age, String date,
                    int heures) {
        super(prenom, nom, age, date);
        this.heures = heures;
    }

    public double calculerSalaire() {
        return SALAIRE_HORAIRE * heures;
    }

    public String getTitre()
        {
            return "Le manut. " ;
        }
}

/* *********************************************************************
 *  L'interface d'employés à risque
 * ********************************************************************/
interface ARisque {
    int PRIME = 200;
}

/* *********************************************************************
 *  Une première sous-classe d'employé à risque
 * ********************************************************************/
class TechnARisque extends Technicien implements ARisque {

    public TechnARisque(String prenom, String nom, int age, String date, int unites) {
        super(prenom, nom, age, date, unites);
```

```
    }

    public double calculerSalaire() {
        return super.calculerSalaire() + PRIME;
    }
}

/* *********************************************************************
 *  Une autre sous-classe d'employé à risque
 * *********************************************************************/
class ManutARisque extends Manutentionnaire implements ARisque {

    public ManutARisque(String prenom, String nom, int age, String date, int heures) {
        super(prenom, nom, age, date, heures);
    }

    public double calculerSalaire() {
        return super.calculerSalaire() + PRIME;
    }
}

/* *********************************************************************
 *  La classe Personnel
 * *********************************************************************/
class Personnel {
    private Employe[] staff;
    private int nbreEmploye;
    private final static int MAXEMPLOYE = 200;

    public Personnel() {
        staff = new Employe[MAXEMPLOYE];
        nbreEmploye = 0;
    }

    public void ajouterEmploye(Employe e) {
        ++nbreEmploye;
        if (nbreEmploye <= MAXEMPLOYE) {
            staff[nbreEmploye - 1] = e;
        } else {
            System.out.println("Pas plus de " + MAXEMPLOYE + " employés");
        }
    }

    public double salaireMoyen() {
        double somme = 0.0;
        for (int i = 0; i < nbreEmploye; i++) {
            somme += staff[i].calculerSalaire();
        }
        return somme / nbreEmploye;
    }

    public void afficherSalaires() {
        for (int i = 0; i < nbreEmploye; i++) {
            System.out.println(staff[i].getNom() + " gagne "
                    + staff[i].calculerSalaire() + " francs.");
        }
    }
}

// =====================================================================

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.");

    }
```

}


Main
+main:void
Employe
+Employe
+salaireCumule:double
+toString:String
nom:String
0..*
Gestion
-employes:Employe []
nombreJoursMois:int
+embaucher:void
-getIndexOfEmploye:int
+licencier:void
+getEmploye:Employe
+afficherEmploye:void
+muter:void
Permanent
+Permanent
+Permanent
+Permanent
+salaireCumule:double
+toString:String
gain:double
joursCumules:int
nombreEnfants:int
marrie:boolean
salaireMensuel:double
primeParEnfantParMois:double
primerParEnfantParMois:double
Temporaire
-salaireHoraire:double
+Temporaire
+Temporaire
+Temporaire
#mutateData:void
+salaireCumule:double
+toString:String
+setSalaireHoraire:void
+getSalaireHoraire:double
heuresCumulees:int
Vendeur
+Vendeur
+Vendeur
+Vendeur
+salaireCumule:double
+toString:String
volumeVentes:double
pourcentageCommissionVentes:double

Exercice 18 : Analyse de conception

1. Non. La classe `Unite` devrait être abstraite car elle ne ne correspond à aucune unité concrète du jeu.
2. On peut éviter la duplication en transformant l'interface Nain en une classe abstraite et en y déclarant les membres `taille`, `hache` et `void frappHache()`. On procède de façon analogue pour l'interface `Elfe`.
3. Non car en Java on ne peut pas hériter de deux classes.
