MOOC Intro POO Java
Corriges semaine 3

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 8 : «EPFLien» (héritage)

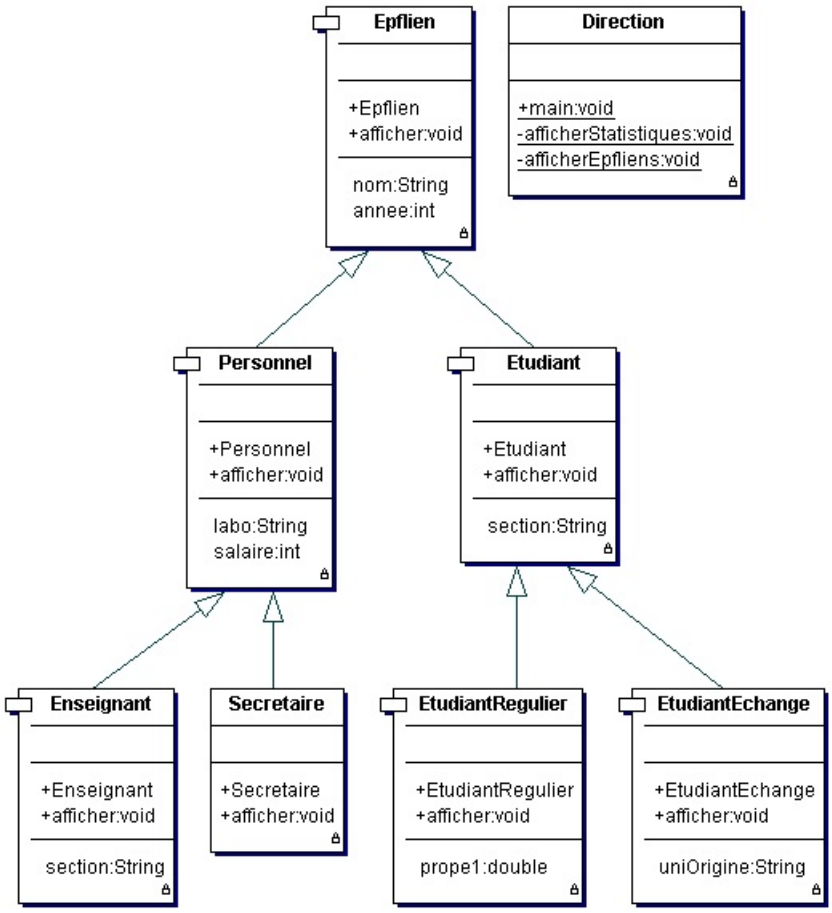
Le code complet est fourni ci-dessous.

```
import java.util.Calendar;
import java.util.ArrayList;

/**
 * Classe principale
   */
class Direction
{
    public static void main(String[] args) {
        Ecole epfl = new Ecole(5);
        epfl.add(new EtudiantRegulier("Gaston Peutimide", 2013, "SSC", 6.0));
        epfl.add(new EtudiantRegulier("Yvan Rattrapeur", 2011, "SSC", 2.5));
        epfl.add(new EtudiantEchange("Bjorn Borgue", 2012, "Informatique", "KTH"));
        epfl.add(new Enseignant("Mathieu Matheu", 1998, "LMEP", 10000, "Physique"));
        epfl.add(new  Secretaire("Sophie Scribona", 2005, "LMT", 5000));
        epfl.afficherStatistiques();
        epfl.afficherEPFLiens();
    }

}

 /**
  * La direction
  */

class Ecole {
    private ArrayList<EPFLien> gens;

    public Ecole(int nbPersonnes) {
        gens = new ArrayList<EPFLien>();
    }

    public void add(EPFLien personne)
        {
            if (personne != null)
            {
                gens.add(personne);
            }
        }

        /**
         * Cette méthode affiche l'ancienneté moyenne des personnes fréquentant  l'école
         * et le nombre d'étudiants parmi eux
         */
    public void afficherStatistiques() {
        int anneeCourante = Calendar.getInstance().get(Calendar.YEAR);
        int nbAnneesTotal = 0;
        int nbEtudiants = 0;
        for (EPFLien epflien : gens) {
            nbAnneesTotal = nbAnneesTotal + (anneeCourante - epflien.getAnnee());
            if (epflien.estEtudiant()){
                ++nbEtudiants;
            }
        }
        System.out.println("Parmi les " + gens.size() + " EPFLiens, " +
                nbEtudiants + " sont des etudiants.");
        double moyen = nbAnneesTotal;
        moyen /= gens.size();
        System.out.println("Ils sont ici depuis en moyenne " + moyen + " ans");
    }
```

```java
    // Cette méthode affiche les caractéristiques des personnes fréquentant  l'école
    public void afficherEPFLiens() {
        System.out.println("Liste des EPFLiens: ");
        for (EPFLien epflien : gens)
            epflien.afficher();
    }
}

/**
  * Les personnes fréquentant l'EPFL
  */
class EPFLien {
    private String nom;
    private int annee;

    public EPFLien(String nom, int annee) {
        this.nom = nom;
        this.annee = annee;
    }

   //Cette méthode affiche les caractéristiques générales d'un EPFLien
    public void afficher() {
        System.out.println("   Nom : " + getNom());
        System.out.println("   Annee : " + getAnnee());
    }

    public String getNom() {
        return nom;
    }

    public int getAnnee() {
        return annee;
    }
    public boolean estEtudiant ()
        {
            return false;
        }

}

/**
  * Les étudiants
  */
class Etudiant extends EPFLien {
    private String section;

    public Etudiant(String nom, int annee, String section) {
        super(nom, annee);
        this.section = section;
    }

    public void afficher() {
        super.afficher();
        System.out.println("   Section : " + getSection());
    }

    public String getSection() {
        return section;
    }

    public boolean estEtudiant()
        {
            return true;
        }

}

/**
  * Les étudiants régulier
  */
class EtudiantRegulier extends Etudiant {
    private double moyenne;

    public EtudiantRegulier(String nom, int annee, String section, double moyenne) {
```

```
        super(nom, annee, section);
        this.moyenne = moyenne;
    }

    public void afficher() {
        System.out.println("Etudiant regulier:");
        super.afficher();
        System.out.println("   Moyenne : " + moyenne);
    }
}

/**
  * Les étudiants  d'échange
  */
class EtudiantEchange extends Etudiant {
    private String uniOrigine;

    public EtudiantEchange(String nom, int annee, String section, String uniOrigine) {
        super(nom, annee, section);
        this.uniOrigine = uniOrigine;
    }

    public void afficher() {
        System.out.println("Etudiant d'echange:");
        super.afficher();
        System.out.println("   Uni d'origine : " + getUniOrigine());
    }

    public String getUniOrigine() {
        return uniOrigine;
    }
}

/**
  * Le personnel de l'EPFL
  */
class Personnel extends EPFLien {
    private String labo;
    private int salaire;

    public Personnel(String nom, int annee, String labo, int salaire) {
        super(nom, annee);
        this.labo = labo;
        this.salaire = salaire;
    }

    public void afficher() {
        super.afficher();
        System.out.println("   Laboratoire : " + getLabo());
        System.out.println("   Salaire : " + getSalaire());
    }

    public String getLabo() {
        return labo;
    }

    public int getSalaire() {
        return salaire;
    }
}

class Enseignant extends Personnel {
    private String section;

    public Enseignant(String nom, int annee, String labo, int salaire, String section) {
        super(nom, annee, labo, salaire);
        this.section = section;
    }

    public void afficher() {
        System.out.println("Enseignant:");
        super.afficher();
        System.out.println("   Section d'enseignement : " + getSection());
    }

    public String getSection() {
```

```
        return section;
    }
}

class Secretaire extends Personnel {
    public Secretaire(String nom, int annee, String labo, int salaire) {
        super(nom, annee, labo, salaire);
    }

    public void afficher() {
        System.out.println("Secretaire:");
        super.afficher();
    }
}
```



Exercice 9 : Boîtes aux lettres (héritage)

Une version possible du corrigé est fournie ci-dessous.

```
/* Classe pour représenter le courrier
 */

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
    public double affranchirNormal(){return 0;};
    // la bonne solution consiste a` d'eclarer cette m'ethode comme suit:
    // abstract private double affranchirNormal();
    // lorsque vous aurez vu les cours de la semaine prochaine, expliquez pourquoi...

    // les attributs (communs aux lettres et colis):
    private double poids;
    private boolean express;
    private 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;
    }

        // un getter pour le poids (car utile dans les sous-classe)
    public double getPoids() {
        return poids;
    }

    // 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:
    private String format = "";

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

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

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

    // redéfinit affranchirNormal()
    public double affranchirNormal() {
        return getPoids()/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;
    }

}

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

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

    public Colis(double poids, boolean express, String adresse, double volume) {
        super(poids, express, adresse);
```

```
        this.volume = volume;
    }

    // redéfinit affranchirNormal();
    public double affranchirNormal() {
        // affranchit les colis selon une formule pr'ecise
        return 0.25 * volume + getPoids()/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 repr'esenter la boite aux lettre
 */

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 Poste {

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

        //Creation de divers courriers/colis..
        Lettre lettre1 = new Lettre(200, true, "Chemin des Acacias 28, 1009 Pully", "A3");
        Lettre lettre2 = new Lettre(800, false, "", "A4"); // invalide

        Publicite pub1 = new Publicite(1500, true, "Les Moilles  13A, 1913 Saillon");
        Publicite pub2 = new Publicite(3000, false, ""); // invalide

        Colis colis1 = new Colis(5000, true, "Grand rue 18, 1950 Sion", 30);
        Colis colis2 = new Colis(3000, true, "Chemin des fleurs 48, 2800 Delemont", 70); //Colis invalide !

        boite.ajouterCourrier(lettre1);
        boite.ajouterCourrier(lettre2);
        boite.ajouterCourrier(pub1);
        boite.ajouterCourrier(pub2);
        boite.ajouterCourrier(colis1);
        boite.ajouterCourrier(colis2);


        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 10 : Puissance 4 (héritage)

Une version simple du Puissance4 vous est fournie. Une bonne extension consiste à essayer de coder des stratégies de jeu un peu plus élaborées.

```
import java.util.Scanner;
/**
 * Classe principale
 */
class Puissance4 {
    protected static Scanner scanner = new Scanner(System.in);
  public static void main(String[] args) {
    System.out.println("Entrez votre nom: ");
    String nom = scanner.nextLine();
    System.out.println("--");

    Partie p = new Partie(new Ordinateur(Jeu.BLEU), new Humain(nom, Jeu.ROUGE));
    p.joue();
  }
}

class Partie {

  private Joueur[] joueurs = new Joueur[2];
  private Jeu jeu;

  public Partie(Joueur joueur1, Joueur joueur2) {
    joueurs[0] = joueur1;
    joueurs[1] = joueur2;
    jeu = new Jeu();
  }

  public void joue() {
    int vainqueur = -1;
    int cJoueur = 0;

    while (vainqueur==-1 && !jeu.estPlein()) {
      joueurs[cJoueur].joue(jeu);
      if (jeu.estPlein()) {
        vainqueur = -1;
      }

      // Si 4 pions sont alignés, on a un vainqueur
      // (même si le jeu est plein!)

      if (jeu.cherche4()) {
    vainqueur = cJoueur;
      }

      // On change de joueur pour l'itération suivante
      cJoueur++;
      cJoueur %= 2;
    }

    System.out.println("La partie est finie.");
    jeu.afficher();
    if (vainqueur == -1) {
      System.out.println("Match nul.");
    } else {
      System.out.println("Le vainqueur est " + joueurs[vainqueur].getNom());
    }
  }
}

class Jeu {
    // final static est une bonne maniere de
    // definir des constantes (voir aux cours prochains)
  public final static int VIDE = 0;
  public final static int BLEU = 1;
  public final static int ROUGE = 2;

  private int taille;
  private int[][] grille;   // 0 = vide, 1 = joueur bleu, 2 = joueur rouge

  public Jeu(int taille) {
```

```
    initJeu(taille);
  }

  public Jeu() {
    initJeu(8);
  }

  private void initJeu(int taille) {
    this.taille = taille;
    grille = new int[taille][taille];
    for (int col = 0; col < taille ; col++) {
      for (int row = 0; row < taille; row++) {
        grille[col][row] = VIDE;
      }
    }
  }

  public boolean joueCoup(int col, int joueur) {
    if ((col < 0) || (col >= taille)) {
      return false;
    }

    // Trouve la première place vide dans la colonne
    for (int ligne = 0; ligne < taille; ligne++) {
      if (grille[col][ligne] == VIDE) {
        grille[col][ligne] = joueur;
        return true;
      }
    }

    // La colonne est pleine
    return false;
  }

  /**
   * Cette méthode vérifie toutes les lignes, colonnes et diagonales pour une série de 4 pions
   * de la même couleur. Si une telle série existe, retourne true.
   *
   * Notez qu'il n'est pas nécessaire de retourner la couleur des 4 pions alignés,
   * puisqu'il s'agit de celle de celui qui vient de jouer.
   * @return true si le jeu contient 4 pions alignés
   */
  public boolean cherche4() {
    // Vérifie les horizontales ( - )
    for (int ligne = 0; ligne < taille; ligne++) {
      if (cherche4alignes(0, ligne, 1, 0)) {
        return true;
      }
    }

    // Vérifie les verticales ( | )
    for (int col = 0; col < taille; col++) {
      if (cherche4alignes(col, 0, 0, 1)) {
        return true;
      }
    }

    // Diagonales (cherche depuis la ligne du bas)
    for (int col = 0; col < taille; col++) {
      // Première diagonale ( / )
      if (cherche4alignes(col, 0, 1, 1)) {
        return true;
      }
      // Deuxième diagonale ( \ )
      if (cherche4alignes(col, 0, -1, 1)) {
        return true;
      }
    }

    // Diagonales (cherche depuis les colonnes gauches et droites)
    for (int ligne = 0; ligne < taille; ligne++) {
      // Première diagonale ( / )
      if (cherche4alignes(0, ligne, 1, 1)) {
        return true;
      }
      // Deuxième diagonale ( \ )
```

```
    if (cherche4alignes(taille - 1, ligne, -1, 1)) {
      return true;
    }
  }

  // On n'a rien trouvé
  return false;
}

/**
 * Cette méthode cherche 4 pions alignés sur une ligne. Cette ligne est définie par
 * le point de départ, ou origine de coordonnées (oCol,oLigne), et par le déplacement
 * delta (dCol,dLigne). En utilisant des valeurs appropriées pour dCol et dLigne
 * on peut vérifier toutes les directions:
 * - horizontale:    dCol = 0, dLigne = 1
 * - vérticale:      dCol = 1, dLigne = 0
 * - 1ère diagonale: dCol = 1, dLigne = 1
 * - 2ème diagonale: dCol = 1, dLigne = -1
 *
 * @param oCol   Colonne d'origine de la recherche
 * @param oLigne Ligne d'origine de la recherche
 * @param dCol   Delta de déplacement sur une colonne
 * @param dLigne Delta de déplacement sur une ligne
 * @return true si on trouve un alignement
 */
private boolean cherche4alignes(int oCol, int oLigne, int dCol, int dLigne) {
  int couleur = VIDE;
  int compteur = 0;

  int curCol = oCol;
  int curRow = oLigne;

  while ((curCol >= 0) && (curCol < taille) && (curRow >= 0) && (curRow < taille)) {
    if (grille[curRow][curCol] != couleur) {
      // Si la couleur change, on réinitialise le compteur
      couleur = grille[curRow][curCol];
      compteur = 1;
    } else {
      // Sinon on l'incrémente
      compteur++;
    }

    // On sort lorsque le compteur atteint 4
    if ((couleur != VIDE) && (compteur == 4)) {
      return true;
    }

    // On passe à l'itération suivante
    curCol += dCol;
    curRow += dLigne;
  }

  // Aucun alignement n'a été trouvé
  return false;
}

/**
 * Vérifie s'il est encore possible de placer des pions
 * @return true si le tableau est plein
 */
public boolean estPlein() {
  // On cherche une case vide. S'il n'y en a aucune, le tableau est plein
  for (int col = 0; col < taille; col++) {
    for (int ligne = 0; ligne < taille; ligne++) {
      if (grille[col][ligne] == VIDE) {
        return false;
      }
    }
  }

  return true;
}

public int getTaille() {
  return taille;
}
```

```
  public void afficher() {
    for (int ligne = taille - 1; ligne >= 0; --ligne) {
      for (int col = 0; col < taille; col++) {
        switch (grille[col][ligne]) {
        case VIDE:
          System.out.print(' ');
          break;
        case ROUGE:
          System.out.print('R');
          break;
        case BLEU:
          System.out.print('B');
          break;
        }
      }
      System.out.println();
    }

    for (int i = 0; i < taille; ++i) {
      System.out.print('-');
    }
    System.out.println();
    for (int i = 1; i <= taille; ++i) {
      System.out.print(i);
    }
    System.out.println();
  }
}

class Joueur {
  private String nom;
  private int couleur;

  public Joueur(String nom, int couleur) {
    this.nom = nom;
    this.couleur = couleur;
  }

  public String getNom() {
    return nom;
  }

  public int getCouleur() {
    return couleur;
  }

  /**
   * Cette méthode joue un coup avec le tableau reçu en paramètre.
   * La méthode est vide car les sous-classes doivent l'implémenter.
   * (Vous verrez prochainement comment gérer ce genre de cas plus proprement)
   * @param jeu Le Jeu avec lequel jouer.
   */
  public void joue(Jeu jeu) {}

}

class Humain extends Joueur {

  public Humain(String nom,int couleur) {
    super(nom, couleur);
  }

  public void joue(Jeu jeu) {
    jeu.afficher();

    boolean valide;
    do {
      System.out.println("Joueur " + this.getNom() + ", entrez un numéro de colonne" +
               "  (entre 1 et " + jeu.getTaille() + ") : ");

      int col = Puissance4.scanner.nextInt(); // on pourrait faire ici la validation de la lecture
      col--;                                  // remet entre 0 et taille-1 (indice à la Java)

      valide = jeu.joueCoup(col, this.getCouleur());
      if (valide == false) {
```

```
        System.out.println("-> Coup NON valide.");
      }
    } while (valide == false);
  }
}

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

class Ordinateur extends Joueur {

  public Ordinateur(int couleur) {
    super("Le programme", couleur);
  }

  public void joue(Jeu jeu) {
    for (int col = 0; col < jeu.getTaille(); col++) {
      if (jeu.joueCoup(col, this.getCouleur())) {
        System.out.println(this.getNom() + " a joué en " + (col + 1));
        return;
      }
    }
  }
}
```
