# NFA032 Programmation objet avec Java

CNAM-Paris-Île-de-France

Première session 2013

**Documents et calculatrice interdits. Le barème est donné à titre indicatif**.

## Exercice 1 : programmation objet (4 points)

### Question 1

Ecrivez une classe représentant une personne avec son nom, son prénom et son numéro d'inscription au répertoire des personnes physiques (NIRPP, couramment appelé numéro de sécurité sociale) qui est un nombre à 13 chiffres. Notez qu'on ne peut pas représenter ce nombre par un int parce que les int ont au plus 10 chiffres. Votre classe comprendra une méthode d'affichage et une méthode qui teste si deux personnes sont la même personne en comparant leurs numéros. Cette méthode ne fera pas d'affichage mais renverra son résultat.

### Question 2

On veut dissocier les informations concernant le numéro NIRPP en plusieurs morceaux : le premier chiffre donne le sexe de la personne (1 pour masculin, 2 pour féminin), les deux chiffres sont les deux derniers chiffres de l'année de naissance, les deux suivants le mois de naissance, les deux suivants pour le département de naissance, les trois suivants la commune de naissance, les trois derniers sont un numéro d'ordre.

Exemple : $1181102045315 = \underbrace{1}_{\text{sexe}}\underbrace{18}_{\text{annee}}\underbrace{11}_{\text{mois}}\underbrace{02}_{\text{departement}}\underbrace{045}_{\text{commune}}\underbrace{315}_{\text{numero}}$

Ecrivez une classe des numéro NIRPP avec 6 variables pour les 6 informations différentes contenues dans un numéro et une méthode qui renvoie le numéro complet. Modifiez la classe des personnes pour prendre en compte la nouvelle représentation des numéros NIRPP.

## Exercice 2 : références (3 points)

```
class Point{
    int x;
    int y;
    Point(int xx, int yy){
        x=xx;
        y=yy;
    }
```

```
}
class Segment{
    Point p1, p2;
    Segment(Point pp1, Point pp2){
        p1=pp1;
        p2=pp2;
    }
}
class Polyedre{
    Point[] tab;
    Polyedre(Point[] tp){
        tp=tab;
    }
}
public class Dessin1{
    public static void main(String[] argv){
        Point p1, p2, p3;
        p1=new Point(1,1);
        p2=new Point(2,2);
        p3=p1;
        Segment seg=new Segment(p1,p2);
        Point[] tabpoint={p1, p2, p3};
        Polyedre pld=new Polyedre(tabpoint);
    }
}
```

Représentez au moyen d'un petit dessin les objets et tableaux existant à la fin de l'exécution du programme donné ci-dessus. Chaque objet sera représenté par un rectangle et chaque référence par une flèche ou une adresse. Attention : on ne vous demande pas un diagramme de classe. C'est chaque objet créé, chaque tableau et chaque référence qui doit être représenté.

## Exercice 3 : exceptions (3 points)

Considérez le code suivant :

```
public class ExoExc1_1 {
   public static void m3(int x, int y){
      if (x==0)
           throw new Err1();
      else if (x<y) {
           throw new Err2();
      }
      Terminal.ecrireStringln("Fin␣normale␣p3");
   }
   public static void m2(int x, int y){
      try {
             m3(x, y);
             Terminal.ecrireStringln("Fin␣normale␣p2");
           } catch (Err2 e) {
             Terminal.ecrireStringln("Recuperation␣p2");
      }
```

```
    }
    public static void m1(int x, int y){
      try {
          m2(x,y);
          Terminal.ecrireStringln("Fin␣normale␣p1");
      } catch (Err1 e ){
          Terminal.ecrireStringln("Recuperation␣p1");
      }
    }

    public static void main(String[] args) {
      int a = Terminal.lireInt();
      int b = Terminal.lireInt();
      m1(a,b);
      Terminal.ecrireStringln("Fin␣programme␣");
    }
}
class Err1 extends Error{}
class Err2 extends Error{}
```

Donnez les messages affichés par l'exécution de ce programme si les valeurs lues pour les variables a et b sont à chaque fois :

1. a=7 et b=5
2. a=0 et b=1
3. a=5 et b=1

## Exercice 4 : listes (4 points)

On donne en annexe un extrait du code des classes `ListeIter` et `ElementListe`.

1. Donnez le code modifié de la classe ElementListe pour que la liste contienne des caractères (char) au lieu de nombres entiers (int).
2. Donnez **les entêtes des méthodes** de la classe `ListeIter` qui doivent être modifiés (et seulement ceux-là).
3. On appelle *répétition* une suite de cellules consécutives contenant la même majuscule. Ecrire une méthode pour supprimer les éléments d'une répétition à l'exception du premier dans une liste L. Exemple : si la liste L contient les lettres : A, A, B, C, C, A, A, A, D, D, cette liste devra, après exécution de la méthode, être transformée en la liste : A, B, C, A, D.
4. Ecrire une méthode qui calcule le nombre de cellules qu'il faut supprimer pour supprimer les répétitions dans une liste. Cette méthode ne modifie pas la liste. Le résultat ne doit pas être affiché, mais renvoyé par la méthode.

## Exercice 5 : programmation et héritage (6 points)

On souhaite modéliser les différentes formes de publications pouvant être réalisées (et regroupées) sur un fil d'actualités des réseaux sociaux. La classe `Post` donnée plus bas, modélise une publication simple,

caractérisée par le nom de son auteur (String) et le nombre de votes attribués par les membres du réseau (bouton "j'aime") sur cette publication. La classe `FilActualite` est à écrire et doit contenir une liste de publications, que vous pourrez modéliser soit par un tableau, soit par un ArrayList de `Post`.

```
public class Post {
   private String auteur;
   private int likes;  // votes sur ce post

   /** Constructeur  **/
   public Post(String auteur){
       auteur = a;   likes = 0;
   }
    public String getAuteur(){ return auteur;  }

    public int getLikes(){ return likes;   }

   /** Incrémente votes "j'aime" sur le post */
    public void Aime(){ likes++;  }

    public void AimePas(){
        if (likes > 0) {likes--;  }
    }
   /** Affiche les details de ce post  */
    public void affiche(){
        System.out.println(auteur);
        if(likes > 0) {
            System.out.println("  -  " + likes + " aiment.");
        }
    }
}
```

## Question 1

On considère deux sortes de publications : les messages texte, et les photos, que l'on souhaite modéliser par deux nouvelles classes `PostMessage` et `PostPhoto`. Ces publications possèdent les caractéristiques d'un `Post`, avec en plus, le texte du message (String) pour une publication message, et le nom du fichier (String) où se trouve l'image et sa légende (String) pour une publication photo. Donnez la définition de ces deux classes en ajoutant dedans, un constructeur, ainsi qu'une redéfinition appropriée de la méthode `affiche()`. Cette méthode devra afficher selon le cas, le message du post, ou le nom du fichier et la légende de l'image, ainsi que les autres caractéristiques du post (nombre de votes, auteur, etc).

## Question 2

Un fil d'actualité regroupe une liste de publications pouvant être des objets `Post` simples, ou `PostMessage` ou `PostPhoto`. Complétez le code de la classe `FilActualite`. Vous pourrez implanter la liste de publications, soit par un tableau, soit par un ArrayList.

```
public class FilActualites {
   // Completer avec  variables necessaires
    /** Construit un fil vide   */
```

```
    public FilActualites()  { // A completer }

    /** Ajoute une publication au fil.  */
    public void ajoutePost(Post post) { // A completer }

    /** Affiche tous les posts, leur auteur + nb de votes  */
    public void affiche() { // A completer }

    /** Affiche le nombre de posts publiés par l'auteur de nom a  */
    public int nombrePostAuteur(String a) { // A completer }

    /** Totalité des votes pour les publications dans ce fil    */
    public int popularite() { // A completer }
}
```

### Question 3

Donnez une méthode main qui déclare un fil d'actualité et y ajoute 3 publications : une simple, une photo et un message texte. Deux parmi ces publications auront comme auteur Geek93. Votre programme doit afficher tous les posts du fil, puis afficher tous les posts de Geek93.

## Annexe : classes ListeIter et ElementListe

```
public class ElementListe {
    private int valeur;
    private ElementListe suivant;
    public ElementListe(int valeur, ElementListe suivant) {
        this.valeur = valeur;
        this.suivant = suivant;
    }
    public ElementListe(int v) {
        this.valeur = v;
        this.suivant = null;
    }
    public int getValeur() {
        return valeur;
    }
    public void setValeur(int valeur) {
        this.valeur = valeur;
    }
    public ElementListe getSuivant() {
        return suivant;
    }
    public void setSuivant(ElementListe suivant) {
        this.suivant = suivant;
    }
}
public class ListeIter {
    ElementListe premier;
```

```
    public ElementListe getPremier() {
        return premier;
    }
    public boolean estVide() { ... }
    public ElementListe getPremier() { ... }
    public void ajouterAuDebut(int v) {
        ElementListe ancienPremier= premier;
        premier= new ElementListe(v,ancienPremier);
    }
    public void ajouterALaFin(int v) {...}
    public int getLongueur() {...}
    public boolean contient(int v) { ...}
    public void retirerPremiereOccurrence(int v) {...}
    public void concatener(ListeIter l) {...}
}
```