Programmation avec des Objets: Cours 4

Classes, Objets, Méthodes

1. Les classes : modularité, encapsulation
2. Champs ou variables associées aux objets
3. Constructeurs
4. Constantes, variables de classe (`static`)
5. Méthodes
6. Fonctions
7. Exemples
8. La méthode `toString`

Les classes, les objets

La programmation avec des classes et des objets permet de:

- Présenter les programmes de manière modulaire
- Obtenir des structures de données composées (enregistrements avec des champs)
- Construire des objets qui se détruisent quand ils ne servent plus
- Utiliser l'héritage (voir cours suivants).

Modularité

- Les classes ont dans cette optique le même rôle que les chapitres d'un livre
- On regroupe les fonctions qui ont trait à un même domaine pour rendre l'ensemble mieux lisible
- Pour utiliser la fonction calcul de la classe Matrice

```
Matrice.calcul(x,y);
```

- On peut aussi mettre des données dans une classe

```
Class MesDonnees{
       static final int N = 1000;
       static final float Pi = 3.1459;}
Class C1 {
     .....
     int n = MesDonnees.N;
     float surface = r * r * MesDonnees.Pi;}
```

Encapsulation: sécurité du logiciel

- Nécessité de séparer l'utilisation d'une fonction de la façon dont elle est réalisée (implantée)
- On parle d'interface, signature d'une fonction
- Le programmeur qui utilise une classe ne doit pas avoir accès aux variables et aux fonctions internes à cette classe
- Permet de modifier la réalisation si on ne touche pas à l'interface

Une classe contient:

- Des constantes
- Des variables de classe
- Des variables associées aux objets (champs)
- Des constructeurs
- Des fonctions (méthodes statiques)
- Des méthodes

Constantes et variables de classe

- Une constante de déclare par: `static final` elle ne peut être modifiée par la suite

```
public static final int Nmax = 1000;
public static final int annee = 2006;
```

- Une variable globale à toutes les fonctions de la classe se déclare par: `static`
- Elle peut être consultée et modifiée par toutes les fonctions de la classe. On les appelle variables de classe ou variables statiques.

```
static int somme = 0;
static int ajouter(int a) {somme += a;}
static int retrancher(int a) {somme -= a;}
```

- On peut les affecter et les évaluer dans la classe en utilisant leur noms.
- A l'extérieur de la classe si elles ont été déclarées `public` en utilisant:
  `NomClasse.nomVariable`
  Cette possibilité est vivement déconseillée

Objets avec champs

- Une classe peut définir un nouveau type
- On déclare les champs comme des variables
  (sans utiliser `static`)
- Les éléments de la classe sont des objets
- Le champ `x` de l'objet `a` est alors noté `a.x`
- Un objet est construit par un constructeur
- Il ya souvent plusieurs constructeurs par classes, ils diffèrent par leur signature

Exemple : les points du plan

- Un point est représenté par son abscisse et son ordonnée
- Un constructeur fabrique l'objet point à partir de ses coordonnées
- Une méthode affiche les coordonnées d'un point
- On a parfois besoin du point $O = (0, 0)$ (origine)
- On veut construire le milieu du segment déterminé par deux points
- Une fonction vérifie si trois points sont alignés

Exemple : les points du plan

```
class Point{
    private double abs;
    private double ord;
    static final double epsilon = 0.2;
    static int nbpoints = 0;
```

Constructeurs

- Un constructeur permet de construire un objet d'une classe en donnant des valeurs à ses champs.
- Un constructeur est une fonction non statique sans résultat qui a le même nom que celui de sa classe
- Pour déclarer un objet de la classe `C`

```
C x;
```

On ne peut pas accéder encore à cet objet

- Pour construire un objet de la classe `C`

```
x = new C(par1, par2, ....)
```

Exemple : les points du plan

```
Point (double x, double y){
    abs = x;
    ord = y;
    nbpoints++;
}

Point() {
    abs = 0;
    ord = 0;
    nbpoints++;
}
```

Les fonctions non statiques ou méthodes

- Déclarées sans `static`.
- Un des paramètres est l'objet qui appelle la méthode
- Cet objet figure avant le nom de la fonction et suivi de `.`
- Exemple `p.proche(p2)`
- Dans la description de la fonction le nom des variables de champ correspond aux champs de l'objet qui a appelé (on peut utiliser aussi `this`).

Méthodes : exemples

```
public double abscisse () {
    return abs;
}
public double ordonnee() {
    return ord;
}
static double absdiff(double a , double b){
    if (a > b) return a - b; else return b-a;
}
public boolean proche (Point u){
    return (absdiff(u.abs, abs) < epsilon) &&
           (absdiff(u.ord, ord) < epsilon) ;
}
```

Fonctions ou méthodes statiques

- Une fonction est statique si elle est formée d'instructions qui ne varient pas suivant les objets
- Elle est alors appelée par son nom à l'intérieur de la classe et par `nomClasse.nomMethode()` à l'extérieur, si elle est publique.

```
public static Point milieu(Point u, Point v) {
    double x = (u.abscisse()+ v.abscisse())/2;
    double y = (u.ordonnee()+ v.ordonnee())/2;
    return new Point (x,y);
}
public static boolean alignes( Point p, Point q, Point r){
     double u = (p.abs - q.abs) * (p.ord - r.ord) -
              (p.abs - r.abs) * (p.ord - q.ord);
     if (u <0) u = -u;
     return (u < epsilon);
}
public String toString(){
    String a = " mon abscisse est: " + abs;
    String b = "   mon ordonnee est: " + ord +"\n";
    return a + b;
    }
```

```
public static void main (String[] args){
    Point u0 = new Point();
    Point u1 = new Point(7.5, 6.3);
    Point u2 = new Point(7.4, 6.4);
    System.out.println (u1.proche(u2)+ "  " +
                        u0.proche(u1));
    Point u3 = milieu(u1,u2);
    System.out.println(u3);
    Point u4 = milieu(u0, u1);
    System.out.println(alignes(u4,u0,u1));
    System.out.println("nombre de points  " + nbpoints);
}
```

```
 % java Point
true  false
 mon abscisse est: 7.45   mon ordonnee est: 6.35

true
nombre de points  5
```

Compléments

- L'instruction `x = y` pour deux objets ne recopie pas `y`, mais effectue une référence de `x` sur `y`.
- Conséquence: un objet paramètre d'appel d'une méthode peut voir les valeurs de ses champs modifiées lors de l'appel.
- Le nom `this` est utilisé pour l'objet qui a appelé une méthode *(à utiliser avec modération)*.
- `this` désigne aussi l'objet en cours de construction dans un constructeur

Compléments

- Le constructeur par défaut qui crée la place sans initialisation existe dans les classes où aucun autre constructeur n'est défini
- L'objet (constante) `null` existe dans toutes les classes

Transformation d'un objet en chaîne de caractères

- Utilise la méthode `public String toString()`
- On peut alors faire un appel à `System.out.println(truc)` qui affiche le résultat de `truc.toString()`.
- L'objet `truc` est alors transformé en chaîne par la méthode `toString()` associée à sa classe, puis affiché.
- Il est bon d'avoir une méthode `toString()` pour chaque classe définie

Un autre exemple de classes: les élèves

Les champs

- Le nom de l'élève: une chaîne de caractères
- La note de l'élève : un entier

Les méthodes

- Affichage d'un élève par son nom et sa note
- Comparaison de deux élèves dont le résultat est booléen

Constructeur

- Donne des valeurs aux deux champs
- Défini par une fonction ayant pour nom `Eleve`

```
class Eleve{
    String nom;
    int total;

    public Eleve(String u, int a) { // Constructeur
         nom = u;
         total = a;
    }
    public  String toString(){
       /* Fait apparaitre a l'ecran le nom
         et la note d'un eleve */
        String u;
       u = "   " + x.nom + "   " + x.total;
       return u;
    }
    public static boolean avant(Eleve u, Eleve v){
       return (u.total >  v.total);}
```

```
    public static void main (String[] args) {
         Eleve u1  = new Eleve("Pierre", 182);
         Eleve u2  = new Eleve("Louis", 137);

         if (avant(u1,u2)){
             System.out.println(u1);
             System.out.println(u2);
        }
        else {
                System.out.println(u2);
                System.out.println(u1);

             }
        }
}
```

```
public static void imprimer(Eleve[] a) {
  for (int i = 0; i < a.length; ++i)
      System.out.println(a[i]);  }

public static void main (String[] args) {
        Eleve[]  tab = new Eleve[4];
        tab[0]  = new Eleve("Alain", 142);
        tab[1]  = new Eleve("Pierre", 182);
        tab[2]  = new Eleve("Louis", 137);
        tab[3]  = new Eleve("Philippe", 181);

       imprimer(tab);
       trier(tab);
       imprimer(tab);
}
```

Rappel: Surcharge

En Java deux méthodes (ou deux constructeurs) peuvent avoir le même nom à condition que les signatures (types des paramètres d'appel) soient différentes ou que ces méthodes appartiennent à des classes différentes.

Exemple: l'opération + est surchargée, désigne à la fois l'addition des nombres et la concaténation de chaînes.