4. Accès aux membres d'une classe

- Java permet plusieurs types d'accès pour les attributs, méthodes et constructeurs d'une classe :

  private : accessible uniquement ds la classe courante ;

  public : accessible dans toutes les classes ;

  (néant) : accessible par les classes du même package ;

Protection des attributs d'un objet

- Autant que possible, les attributs doivent être 'private' ;
- Pour autoriser la lecture d'une variable, on lui associe un accesseur, çàd une méthode qui renvoie la valeur de la variable (exemple: getNom pour lire l'attribut nom) ;
- Pour autoriser la modification d'une variable, on lui associe un modificateur, çàd une méthode qui permet la modification tout en contrôlant sa validité (exemple : setDate pour initialiser l'attribut date) ;

Exemple

- Soit une classe 'Année' qui contient un attribut entier 'an' ;
- Prévoir un constructeur qui initialise 'an' ;
- Prévoir un modificateur pour permettre par la suite la modification de l'attribut avec un contrôle de validité (valeur comprise entre 1950 et 2007) ;
- Prévoir un accesseur pour permettre la lecture de son contenu ;

5. Affectation et comparaison d'objets

- Comme vu précédemment, les variables de type classe contiennent des références sur des objets ;
- Lorsqu'une affectation porte sur une variable de type classe, c'est la référence qui change et non l'objet lui-même !!!

Exemple : Soit la classe Point qui possède un constructeur à 2 arguments.

```
Point a, b ;
a = new Point (3, 5) ;
b = new Point (2, 0) ;
```

Après exécution, on obtient en mémoire ...

(*)

Si on exécute l'instruction :

a = b ;

c'est la référence de a qui devient la référence de b ;

a et b désignent donc le même objet et il n'y a plus aucune référence vers le premier objet !!!

- Il n'existe pas d'instruction qui permet de libérer l'espace inutilisable car Java dispose d'un mécanisme de gestion automatique de la mémoire :

le GARBAGE COLLECTOR

- Le garbage collector est une tâche de la JVM qui :
  - travaille en arrière-plan ;
  - libère la place occupée par les instances non référencées ;
  - compacte la mémoire occupée ;

  Il intervient :

  - quand le système a besoin de mémoire;
  - ou, de temps en temps avec une priorité faible ;

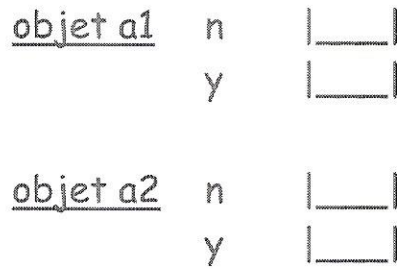
6. Variables et méthodes de classe

a) Les attributs existent dans chacune des instances de la classe. On les appelle des variables d'instance.

Soit la classe :
```
class A
{  int n;
   float y ; }
```

Avec :
```
A a1 = new A( ) ;
A a2 = new A( ) ;
```

On obtient le schéma suivant :



- Java permet de définir des variables de classe, elles n'existeront qu'en un seul exemplaire, quel que soit le nombre d'objets instances de la classe ;
- Il suffit de les déclarer avec l'attribut 'static' ;
- Exemple : classe avec un champs statique destiné à contenir le nombre d'objets déjà créés ;

```
class Obj
{
        public Obj()
        {       nb++ ;
                System.out.println ("Il y a " + nb + " objet(s)") ;
        }
        private static int nb = 0 ;
}
public class TstObj
{
        public static void main (String args[ ])
        {
                Obj a ;
                a = new Obj( ) ;
                Obj b = new Obj( ) ;
                Obj c = new Obj( ) ;
        }
}
```

Qu'affichera ce programme ?

b) De manière analogue, certaines méthodes peuvent avoir un rôle indépendant d'un quelconque objet ;

Une telle méthode est dite méthode de classe ;

Elle ne pourra agir que sur des variables de classe, puisque par nature elle n'est liée à aucun objet en particulier ;

```
class Obj
{
    public Obj( )   // Constructeur
    { nb++ ; }

    // Méthode de classe : statique...

    public static int nbObj ()
    { return nb ; }

    private static int nb = 0 ;
}
```



```
public class TstObj2
{
    public static void main (String args[])
    { Obj a = new Obj() ;
      Obj b = new Obj() ;
      Obj c = new Obj() ;
      System.out.print ("Nbre d'objets = " + Obj.nbObj( ) ) ;

      /* Ici pour faire appel à la méthode nbObj, on ne
         mentionne pas un nom d'objet particulier car la
         méthode est statique...
              on l'appelle avec Obj.nbObj() !!!  */
    }
}
```

- Pour faire appel à une variable de classe ou à une méthode de classe, il n'est pas nécessaire d'instancier un objet de la classe concernée, il suffit de mentionner :

NomDeClasse.VariableDeClasse

ou

NomDeClasse.MéthodeDeClasse( ) ;

- Vous avez déjà, au cours des labos précédents, manipulé des variables et des méthodes statiques sans vous en rendre compte...

- Avez-vous une idée desquelles ?

(*)

Exercices

1) Parmi les attributs suivants de la classe Renault Espace, la version avec toutes les options possibles, distinguez ceux que vous déclareriez comme statiques des autres :

- vitesse
- nombre de passagers
- vitesse maximale
- nombre de vitesses
- capacité du réservoir
- âge
- prix
- couleur

2) Une erreur s'est glissée dans le code suivant, laquelle ?

```
class Test
{   int a ;
    Test (int b)
    { a = b ; }
}
public class PrincipalTest
{   public static void main (String args[])
    {       Test unTest = new Test ( ) ;
    }
}
```

3) Une erreur s'est glissée dans le code suivant, laquelle ?

```
class Test
{   int a ;
    int c ;
    public Test (int b)
    { a = b ; }
    public static int getC()
    { return c ; }
}
public class PrincipalTest
{   public static void main (String args[])
    {  Test unTest = new Test (5) ;  }
}
```


7. Surdéfinition ou surcharge

- On parle de surcharge lorsque plusieurs constructeurs ou méthodes portent le même nom ;
- Dans ce cas, c'est la signature de la méthode qui permet au compilateur de faire la différence (= nombre et type d'arguments) ;

Exemple 1 : Surcharge de constructeur de la classe Point.

```
public class Point
{   private int x, y ;                          // Variables encapsulées...
    public Point (int abs, int ord)             // Constructeur...
        {       x = abs ;
                y = ord ;
        }
    public Point ()                             // 2eme constructeur...
        {       x = 0 ;
                y = 0 ;
        }
}
```


Lors de l'instanciation :

```
Point a = new Point ( ) ;

        // Fait appel au 2ème constructeur...

Point b = new Point (4, 6) ;

        // Fait appel au 1er constructeur...
```

Exemple 2 : Surcharge de la méthode "deplace" de la classe Point.

```
public void deplace ( int dx, int dy )    // deplace (int, int)...
{
    x += dx ;
    y += dy ;
}

public void deplace ( int dx )            // deplace (int)...
{
    x += dx ;
}
```


Lors de l'exécution :

```
Point a = new Point ( ) ;
a.deplace(5) ;
        // Appelle deplace(int)...

a.deplace(2, 2) ;
        // Appelle deplace(int, int)...
```

Nous avons déjà rencontré la surcharge dans la classe Math…