# 3. Les classes et les objets

## 3.1. La notion de classe

La notion de classe généralise la notion de type

On se propose de définir une classe *Point* destinée à manipuler des points

Syntaxe

```
class Id_Classe
{
public :        //contrôle d'accès, voir + loin
        //déclaration des objets et des méthodes de la classe
}
```

où Id_Classe : identificateur de la classe
public : contrôle d'accès à la classe
{} : délimite la classe

Exemple :

```
public class Point
{
        // instruction de definition des champs et methodes de la classe

}
```

### 3.1.1. Définition des champs et des méthodes

Définition :
On appelle méthode les services fournis par toute instance de la classe.

Syntaxe

```
class Id_Classe
{
        //déclaration des méthodes de la classe

        public Type service(type1 arg1, type2 arg2,…,typen argn)
```

```
        {
                //bloc d'instruction
        }
        //déclaration des objets de la classe
        private type1 var1 ;
}
```

Exemple :

```
public class Point
{
        public void initialise(int abs, int ord)
        {
                x = abs ;
                y = ord ;
        }

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

        public void affiche()
        {
                System.out.println(« Point(« +x+ » ,» +y + ») ») ;
        }

        private int x ;         //abscisse
        private int y ;         //ordonnee

}
```

remarques :

- Une variable locale a la classe peut être initialisée lors de sa déclaration

Exemple :

```
public class Point
{
        public void initialise(int abs, int ord)
        {
                x = abs ;
                y = ord ;
        }

        public void deplace(int dx, int dy)
        {
```

```
            x += dx ;
            y += dy ;
        }

        public void affiche()
        {
            System.out.println(« Point(« +x+ » ,» +y + ») ») ;
        }

        private int x = 0;          //abscisse
        private int y = 0 ;         //ordonnee

}
```

- La valeur d'un champ peut être constante s'il est déclaré avec le mot clef *final*. Mais un champ déclaré *final* doit être initialisé au plus tard dans le constructeur. La notion de constructeur sera définie par la suite.

### 3.1.2. Utilisation d'une classe

Syntaxe

```
class Test_Classe
{
        Id_Classe       var_classe ;

        var_classe = new Id_Classe() ;

        var_classe.service(…);
}
```

Exemple

```
public class TestPoint
{
        public static void main(String args[])
        {
            Point    M ;

            M = new Point() ;

            M.initialise(3,5) ;
            M.affiche() ;
        }
}
```

Programme comportant plusieurs classes : un fichier source par classe (existe d'autres possibilités que nous n'aborderons pas ici)

## 3.2. La notion de constructeur

Définition

Comme pour les types fondamentaux, lors de la déclaration d'un objet, le compilateur doit :

- Réserver une quantité de mémoire suffisante pour stocker l'objet.
- Donner une valeur initiale à l'objet.

Le compilateur fournit une fonction d'initialisation par défaut pour chaque classe. Elle est appelée implicitement lors de la définition d'un objet.

Le langage Java offre la possibilité de définir sa propre méthode d'initialisation, méthode appelée constructeur.

Un constructeur est une fonction membre comme les autres avec deux caractéristiques particulières :

- Il porte le même nom que la classe
- Il n'a pas de type de retour

Exemple :

```
public class Point
{
        public Point(int abs, int ord)
        {
                x = abs ;
                y = ord ;
        }

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

        public void affiche()
        {
                System.out.println(« Point(« +x+ » ,» +y + ») ») ;
        }
```

```
        private int x ;          //abscisse
        private int y ;          //ordonnee

}

public class TestPoint
{
        public static void main(String args[])
        {
                Point    M ;

                M = new Point(3.5,2.6) ;

                M.affiche() ;
        }
}
```

Quelques remarques :

- Un constructeur ne fournit aucune valeur de retour
- Une classe peut  ne disposer d'un, plusieurs ou aucun constructeurs.
- Un constructeur peut ne prendre aucun argument en entrée
- Un constructeur ne peut pas être appelé directement :
  exemple : M.Point(3,8) //erreur
- Un constructeur peut appelé un autre constructeur d'une même classe avec le mot clef super : voir plus tard
- Un constructeur peut être déclaré private, alors il ne pourra être appelé de l'extérieur. Il est alors impossible d'instancier un objet sur cette classe.

## 3.3. Les références en java

### 3.3.1. Notion de contrat et d'implémentation

L'encapsulation des données n'est pas obligatoire en java mais fortement conseillée. Une bonne conception orientée objet s'appuie sur la notion de contrat. Cela revient à considérer que la classe est caractérisée par :

- les en-têtes de ses méthodes publiques
- le comportement de ces méthodes

Le reste :

- les champs
- les méthodes privées
- le corps des méthodes publiques

ne sont pas connu de l'utilisateur et constituent l'implémentation de la classe

Le contrat définit ce que fait la classe tandis que son implémentation précise comment elle le fait.

Typologie des méthodes d'une classe :

- le constructeur
- méthodes d'accès ou accessor : fournissent des informations relatives a l'état d'un objet
- méthodes d'altération : modifie l'état de l'objet

### 3.3.2. Affectation et comparaison d'objet

Deux exemples :

Les variables de type classe sont des références

exemple p119 et 120

Initialisation de référence et référence nulle

Rappelons qu'une variable locale d'un type prédéfini ne peut pas être définie sans l'initialiser. Il en est de même pour les variables de type classe.

Exemple :

```
public static void main(String args[])
{
        Point   p;

        p.affiche(); //erreur de compilation
}
```

En revanche, un champ d'un objet est toujours initialisé soit :

- à une valeur nulle
- explicitement
- au sein du constructeur

### 3.3.3. Passage des arguments

Dans différents type de langage de programmation, on rencontre deux façons d'effectuer les transferts entre l'argument effectif et l'argument muet :

Par valeur : le méthode reçoit une copie de la valeur de l'argument effectif.

Par adresse : la méthode reçoit l'adresse de l'argument effectif.

Java emploie systématiquement le passage des arguments par valeur.
Une méthode ne peut pas modifier la valeur d'un argument effectif.

Exemple :

```
class Util
{ public static void echange (int a, int b)  // ne pas oublier static
  { System.out.println ("debut echange : " + a + " " + b) ;
    int c ;
    c = a ;
    a = b ;
    b = c ;
    System.out.println ("fin echange   : " + a + " " + b) ;
  }
}
public class Echange
{ public static void main (String args[])
  { int n = 10, p = 20 ;
    System.out.println ("avant appel   : " + n + " " + p) ;
    Util.echange (n, p) ;
    System.out.println ("apres appel   : " + n + " " + p) ;
  }
}
```

L'unité d'encapsulation est la classe.

### 3.3.4. Autoréférence

Il peut arriver qu'au sein d'une méthode, on est besoin de faire référence à l'objet dans sa globalité. On utilise alors le mot clef this.

Exemple :

```
public boolean coincide(Point P)
{
        return (pt.x == this.x) && (pt.y == this.y);
}

public Point(int x, int y)
{
        this.x = x ;
        this.y = y ;
```

```
        }
```

Appel d'un constructeur au sein d'un autre constructeur

Nous avons vu qu'il n'est pas possible d'appeler un constructeur . Il existe cependant une exception : au sein d'un autre constructeur. On utilise alors le mot clef this

Exemple

```
class Point
{ public Point(int abs, int ord)
  { x = abs ;
    y = ord ;
    System.out.println ("constructeur deux arguments : " + x + " " + y) ;
  }
  public Point()
  { this (0,0) ;  // appel Point (0,0) ; doit etre la premiere instruction
    System.out.println ("constructeur sans argument") ;
  }
  private int x, y ;
}
public class Consthis
{ public static void main (String args[])
  { Point a = new Point (1, 2) ;
    Point b = new Point() ;
  }
}
```

### 3.3.5. La notion de clone

L'affectation des objets de type variable, se limite  a la recopie de référence et non a la recopie de la valeur.
Une bonne démarche est de prévoir dans la classe une méthode destinée à fournir une copie de l'objet.

Exemple :

```
public class Point
{
        public Point(int abs, int ord)
        {
                x = abs ;
```

```
        y = ord ;
    }

    public Point copie()  // renvoie une référence a un point
    {
        Point p=new(x,y) ;
        p.x = x ;
        p.y = y ;
        return p ;
    }

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

    public void affiche()
    {
        System.out.println(« Point(« +x+ » ,» +y + ») ») ;
    }

    private int x ;          //abscisse
    private int y ;          //ordonnee

}
```

Remarque :
Si la classe comporte un champ de classe alors deux possibilités s'offrent à nous :

- La copie superficielle : on recopie la valeur de tous les champs y compris ceux de type classe
- La copie profonde : On recopie la valeur d'un type primitif et pour les champs de types classe, on crée une nouvelle référence à un autre objet du même type et de même valeur
  Il est préférable alors d'avoir pour chaque classe concernée une méthode copie. On parle alors de clonage.

### 3.3.6. Comparaison d'objets

Les operateurs == et != s'appliquent aussi aux objets. Mais attention, dans le cas :

```
Point a,b ;
a==b    vraie si a et b font référence a un seul et même objet
```

et non pas si les valeurs des champs de a et de b sont les mêmes…

### 3.3.7. Le ramasse-miettes

L'operateur new alloue la place mémoire nécessaire pour un objet.

Il n'existe aucun operateur pour libérer l'espace mémoire occupe par un objet dont on a plus besoin. C'est le rôle du Garbage collector ou ramasse-miettes.

Java gère les objets par référence donc a tout instant on connait le nombre de référence a un objet donne. Lorsqu'il n'existe plus aucune référence à un objet, il devient candidat au ramasse-miette.

Rque : On peut créer un objet sans conserver sa référence.
Exemple : (new Point(3,5).affiche()) ;

## 3.4. Autour des méthodes

### 3.4.1. règle d'écriture

**Valeur de retour d'une fonction**

Une méthode peut ne fournir aucun résultat, le mot clef void est alors employé comme type de la valeur de retour
Si une méthode fournit un résultat on parle alors de méthode fonction.

```
public class Point
{
        public Point(int abs, int ord)
        {
                x = abs ;
                y = ord ;
        }



        int getX {return x ;}
        int getY {return y ;}


        private int x ;         //abscisse
        private int y ;         //ordonnee
```

}

**Les arguments d'une méthode**

Les arguments figurant dans l'en-tête de la définition d'une méthode se nomment arguments muets.

Il est possible de déclarer un argument muet avec l'attribut final, dans ce cas sa valeur ne pourra être modifiée à l'intérieur de la fonction

Les arguments fournis lors de l'appel d'une fonction sont quand a eux appelé arguments effectifs

**Variable locale**

La portée d'une variable locale est limitée au bloc ou elle est déclarée

Exemple :

```
void f(int n)
{
        float x ;
        float n ; // erreur de compilation
}
```

.

### 3.4.2. Champ et méthode de classe

**Champ de classe**
Java permet de définir des champs de classe (ou statique) qui n'existent qu'en un seul exemplaire quel que soit le nombre d'instance de la classe. Il suffit alors de les déclarer avec le mot clef static

Exemple :

```
class B
{
        static int n ;
}
```

Exercice : rajouter a la classe point une variable qui permet de compter le nombre de points créés

**Méthode de classe**

De la même manière, il est possible de définir des méthodes de classe. Ces méthodes de classe ont un rôle indépendant d'un quelconque objet. Ces méthodes agissent sur des champs de classe ou les utilisent.
L'appel d'une telle méthode se fait avec le nom de la classe.

Exemple :

```
class B
{
        static int n ;
        private float x;

        public static void f()
        {
                // ici on ne peut agir que sur n
                // on ne peut pas accéder à x
        }
}


B.f() ;
```

Exercice : rajouter à la classe point une méthode de classe qui contient les coordonnes du point origine pour tous les points.

### 3.4.3. Surdéfinition de méthodes

D'une manière générale, on parle de surdéfinition lorsqu'un même symbole possède plusieurs significations différentes, le choix de l'une des significations se faisant en fonction du contexte.

Définition :
On appelle signature d'une méthode la liste des arguments de la fonction respectivement associés à leur type.

Le nom et la signature de la fonction identifient pleinement chaque fonction.

```
public class Point
{
        public Point(int abs, int ord)
        {
                x = abs ;
                y = ord ;
        }
```

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


	private int x ;		//abscisse
	private int y ;		//ordonnee

}

public class Surdef
{
	public static void main(String args[])
	{
		Point a=new Point(1,3);

		a.deplace(1,5);		//appelle deplace(int,int)
		a.deplace(2);		//appelle deplace(int)
		short p=3 ;
		a.deplace(p);	//appelle deplace(short)
		byte b = 2;
		a.deplace(b);	//appelle deplace(short) après conversion de b en short
		int n=5 ;
		a.deplace(n,p);		//appelle deplace(int,short)
		a.deplace(p,n);		//appelle deplace(int,short)
		a.deplace(p,5);		//erreur ambiguite
```

```
        }
}
```

rque : le type de valeur de retour n'intervient pas dans le choix d'une méthode surdéfinie

Exercice : surdéfinir une classe qui donne le min

```
class Surdef
{
        public int min(int a, int b)
        {
                if (a < b)
                        return   a;
                else
                        return b;
        }

        float min(float a, float b)
        {
                        if (a < b)
                        return   a;
                else
                        return b;

        }
}
```

Rque :

- Le type de la valeur de retour ne fait pas parti de la signature.
  Exemple

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

  provoque une erreur : même nom et même signature

- une méthode peut être privée ou publique. Dans tous les cas, elle peut être surdéfinie. Comme les méthodes privées ne sont pas accessible en dehors de la classe, suivant son emplacement, un même appel peut conduire à l'appel de méthodes différentes

  Exemple :

```
class Ecrire
{
        public void f(float x)
        {
                System.out.println(f(float) x = " + x);
        }
        private void f(int n)
        {
                System.out.println(f(int) n = " + n);
        }
        public void g()
        {
                int n=1;
                float x=1.5;

                System.out.println(" dans g");
                f(n);
                f(x);
        }
}
public class test
{
        Ecrire  varE = new Ecrire() ;

        varE.g() ;
        int n=2 ;
        float x = 2.5 ;
        varE.f(n) ;
        varE.f(x) ;
}
```

```
---dans g
f(int) n=1
f(float) x=1.5
---dans f
f(float) n=2.
f(float) x=2.5
```

## 3.5. Les paquetages

La notion de paquetage généralise la notion de classe. Il s'agit de regrouper un certain nombre de classe sous un identificateur commun. Cette notion est proche de celle de bibliothèque.

Cette notion facilite aussi le développement dans le cas de logiciel conséquent. En effet, le risque de créer deux classes de même nom est limite aux seules classes d'un même paquetage.

### 3.5.1. Attribution d'une classe a un paquetage

Un paquetage est caractérisé par un identificateur ou une suite d'identificateurs séparés par un point.

Exemple :

MesClasses
Utilitaire.Mathematique
Utilitaire.M2

L'attribution d'un nom de paquetage se fait au niveau du fichier source. Toutes les classes d'un même fichier source appartiennent à un même paquetage. Il faut alors placer en début de fichier, l'instruction :

package id_Package

### 3.5.2. Utilisation d'une classe d'un paquetage

Il existe deux façons d'appeler une classe appartenant à un autre package :

- citer le nom du paquetage puis le nom de la classe
  Syntaxe :
  ```
  MesClasses.Point       p=new mesClasses.Point(2.5)
  p.affiche() ;
  ```
- en utilisant l'instruction import
  Syntaxe :
  ```
  import MesClasses.Point ; //importe la classe Point du package
  MesClasses
  ```
- en important un paquetage complet
  ```
  import MesClasses.*; // permet d'utiliser toutes les classes du package
  ```
  On peut alors utiliser les classes sans préciser à quel package elles appartiennent.

  Remarques : Deux paquetages contenant des classes de meme nom ne peuvent pas etre importés complets.

### 3.5.3. Les paquetages standards

Les classes standards fournies par Java sont souvent structurées en paquetage

Ex : jawa.awt, java.swing

Il existe un paquetage particulier : java.lang, importe automatiquement par le compilateur. Il permet d'utiliser les classes standards telles que Math, System, Float ou integer sans nécessiter d'instruction import.

### 3.5.4. Droit d'accès

#### 3.5.4.1. Aux classes

Chaque classe dispose d'un droit d'accès :

- avec le mot clé public : la classe est accessible à toutes les autres classes
- sans le mot clé public : la classe est accessible aux classes du même paquetage seulement

#### 3.5.4.2. Aux membres d'une classe

- avec le mot clé public : un membre est accessible de l'extérieur de la classe
- avec le mot clef private : le membre est accessible qu'aux méthodes de la classe
- en l'absence du mot clef public ou private : l'accès au membre est limité au paquetage