Programmation Orientée Objets - JAVA

Erick STATTNER

Maître de Conférences en Informatique

Université des Antilles

erick.stattner@univ-antilles.fr

www.erickstattner.com

Chapitre II. Classes et Objets

1. Types primitifs
2. Classe et organisation
3. Objets et accès
4. Constructeurs
5. Encapsulation et visibilité
6. Attributs et méthodes de classes
7. Comparaison d'objets
8. Destruction des objets

II. Classes et Objets

1. Types primitifs

II. Classes et Objets

1) Types primitifs

Types primitifs

- Comme dans tout langage, possibilité de manipuler des données élémentaires
  - **Entier**:
    - **byte** (1 octet);
    - **short** (2 octets);
    - **int** (4 octets);
    - **long** (8 octets)
  - **Réel**:
    - **float** (4 octets);
    - **double** (8 octets)
  - **Booléen**: **boolean** (true ou false)
  - **Caractère**: **char** (unicode sur 2 octets)
- Ne sont pas considérés comme des objets

II. Classes et Objets

1) Types primitifs

Variables

- Déclaration:
  <type> nom
  int a;
- Initialisation:
  a = 12;
- Déclaration + initialisation:
  int a = 12;
- **ATTENTION: Chaque variable doit être initialisée avant utilisation !**

Tableaux

- **int[] tab = new int[10]**
  tab[0] = 12;

Par convention, le nom d'une variable s'écrit entièrement en minuscule

II. Classes et Objets

2. Classe et organisation

II. Classes et Objets

2) Classe et organisation

Classe VS objet ???

II. Classes et Objets

2) Classe et organisation

La classe

- Permet de créer des types de données plus complexes
- Modélise une entité élémentaire du système
- Garantit que tous les objets crées auront
  - Même structure
  - Même comportement
- Se compose
  - Attributs
  - Méthodes

Par convention, le nom d'une classe commence par une Majuscule
Ex. Point

II. Classes et Objets

2) Classe et organisation

Les attributs

- Données qui caractérisent l'entité modélisée
- Les attributs sont accessibles partout au sein de la classe
  - Visibles et modifiables dans toutes les méthodes de la classe
- Définis par
  - Un type (primitif ou classe)
  - Un nom

<typeAttribut> <nomAttribut>

Par convention, le nom d'un attribut commence par une minuscule

II. Classes et Objets

2) Classe et organisation

Les méthodes

- Définissent le comportement de l'entité modélisée
- Accessibles aux autres méthodes de la classe
- Définies par
  - Un type de retour
  - Un nom
  - Des paramètres
    - Eléments dont la classe a besoin pour réaliser le traitement
    - Accessibles uniquement dans la méthode

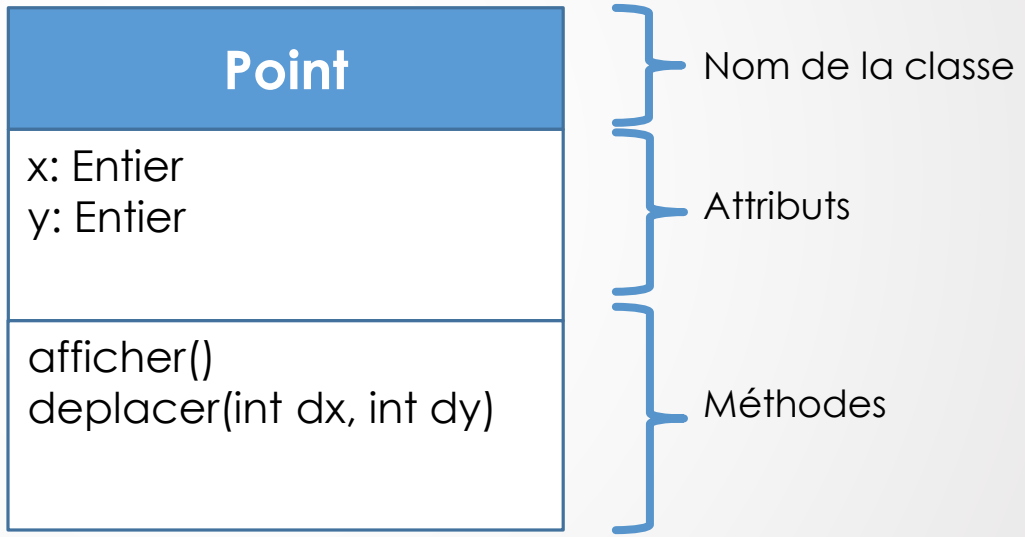
<typeRetour> <nomMethode> (paramètre1, paramètre2 ….)

Par convention, le nom d'une méthode commence par une minuscule

II. Classes et Objets

2) Classe et organisation

Exemple d'un point dans un repère 2D



- **Quel que soit le point crée dans le programme**
  - Un point sera toujours caractérisé par deux entiers: x et y
  - Un point sera toujours en mesure de répondre à 2 actions
    Affiche toi ! Déplace toi !

II. Classes et Objets

2) Classe et organisation

Point
x: Entier y: Entier
afficher() deplacer(int dx, int dy)

Nom de la classe

```
class Point{
      //ATTRIBUTS
      int x;
      int y;

      //METHODES
      void afficher(){
            ...
      }
      void deplacer(int dx, int dy){

           ...
      }
```

Attributs

Méthodes

II. Classes et Objets

2) Classe et organisation

Point
x: Entier y: Entier
afficher() deplacer(int dx, int dy)

```
class Point{
     //ATTRIBUTS
     int x;
     int y;

     //METHODES
     void afficher(){
          System.out.println("x="+x+"y="+y);
     }
     void deplacer(int dx, int dy){
          x = x + dx;
          y = y + dy;
     }
}
```

II. Classes et Objets

2) Classe et organisation

Point
x: Entier y: Entier
afficher() deplacer(int dx, int dy)

Attributs visibles dans toutes les méthodes

Paramètres (visibles uniquement dans la méthode)

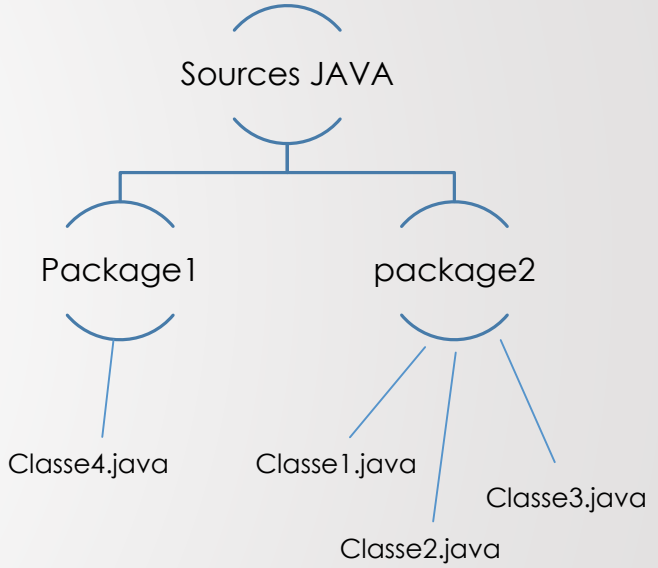
```
class Point{
      //ATTRIBUTS
      int x;
      int y;

      //METHODES
      void afficher(){
            System.out.println("x="+x+"y="+y);
      }
      void deplacer(int dx, int dy){
            x = x + dx;
            y = y + dy;
      }
}
```

II. Classes et Objets

2) Classe et organisation

Package

- Une classe est écrite dans un fichier .java
- **1 classe = 1 fichier**
- **Le fichier doit avoir le même nom que la classe !**
- Les classes sont organisées dans des ***packages***
  - Pour utiliser une classe d'un autre package utiliser **import**
    import package1.maClasse
    import package1.*;
  - Les classes de JAVA sont aussi organisées en package ex. pour utiliser l'instruction de lecture
    import java.util.Scanner;



Par convention, le nom d'un package est entièrement en minuscule

II. Classes et Objets

2) Classe et organisation

Exercice

- Un compte bancaire est modélisé par un identifiant, le nom et le prénom du propriétaire, ainsi que le solde du compte

1. Proposer une classe qui modélise un compte bancaire
2. Implémenter la fonction créditer qui ajoute une somme passée en paramètre au compte

II. Classes et Objets

3. Objets et accès

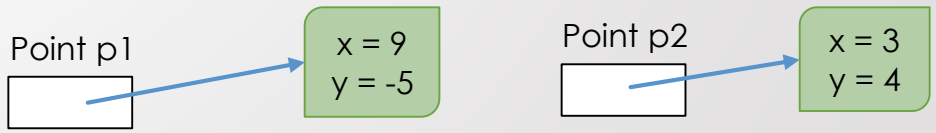
II. Classes et Objets

3) Objets et accès

Un objet

- Un objet est une instance (occurrence) d'une classe
  - Nécessairement conforme au cadre défini dans la classe
  - Chaque attribut à une valeur
- Chaque objet est stocké dans un emplacement mémoire dédié

Point
x: Entier y: Entier
afficher() deplacer(int dx, int dy)



II. Classes et Objets

3) Objets et accès

Création d'un objet

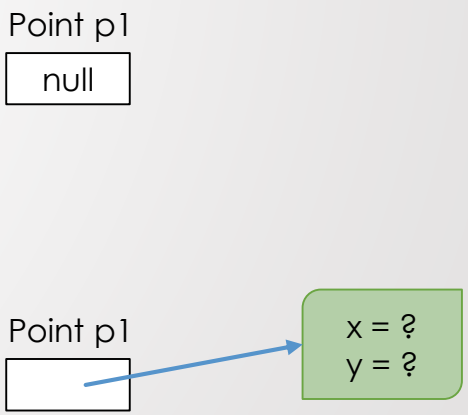
- La création d'un objet est appelée l'***instanciation***
- Utiliser le "***constructeur***" avec le mot clé ***new***
- Deux étapes

1. **Déclaration**
   Définit le nom et le type de l'objet
   A cette étape la référence pointe vers ***null***
   Point p1;

2. **Instanciation**
   Appel au constructeur
   Réservation de l'espace mémoire
   p1 = new Point();

   ou

   Point p1 = new Point();



II. Classes et Objets

3) Objets et accès

Accès aux attributs et méthodes

- Une fois **l'objet** créé, l'accès aux attributs et aux méthodes de l'objet se fait par l'opérateur **.**
- Pour un attribut
  <nomObjet> . <nomAttribut>
  Ex.
  ```
  Point p1 = new Point();
  p1.x = 8;
  p1.y = 4;
  ```
- Pour une méthode
  <nomObjet> . <nomMethode> (parametres)
  Ex.
  ```
  p1.afficher();
  ```

II. Classes et Objets

3) Objets et accès

Accès aux attributs et méthodes

```
Point p1 = new Point();
p1.x = 9;
p1.y = 0;

Point p2 = new Point();
p2.x =  5;
p2.y = 3;

p2.deplacer(2,2);

p2.afficher();

p1.afficher();
```

```
class Point{
     //ATTRIBUTS
     int x;
     int y;

     //METHODES
     void afficher(){
          System.out.println("x="+x+"y="+y)
     }
     void deplacer(int dx, int dy){
          x = x+dx;
          y = y+dy;
     }
}
```

II. Classes et Objets

3) Objets et accès

- **Exercice**

Soit la classe Compte qui modélise un compte bancaire

1. Ajouter la méthode afficher à la classe Compte
2. Ecrire un programme de Test qui crée deux comptes c1 et c2
3. Créditer le compte c1 de 100 euros et le compte c2 de 500 euros
4. Afficher les comptes c1 et c2

II. Classes et Objets

4. Constructeurs

II. Classes et Objets

4) Constructeurs

Construire un objet

- Utiliser un ***constructeur***
- Il existe le constructeur par défaut qui ne possède pas de paramètres
  - Attribut la valeur 0 à tous les attributs de type nombres
  - Attribut la valeur **null** à tous les attributs de type objets
    EX.
    ```
    Point p = new Point();
    System.out.println(p.x);
    ```
    → **??**
- Le constructeur peut être redéfini pour initialiser différement l'objet
  EX.
  ```
  Point p = new Point(3, 4);
  ```

II. Classes et Objets

4) Constructeurs

Exemple du constructeur du point

- Le constructeur est une méthode qui
  - possède EXACTEMENT le même nom que la classe
  - ne possède pas de type de retour
  - possède en paramètre tous les éléments pour initialiser l'objet

```
Point p1 s= new Point (3, 4);

Point p2 = new Point (0, 0);

Point p3 = new Point ();
```

Le constructeur par defaut n'existe plus !

```java
class Point{
    //ATTRIBUTS
    int x;
    int y;

    //CONSTRUCTEURS
    Point(int px, int py){
        x = px;
        y = py;
    }
}
```

II. Classes et Objets

4) Constructeurs

Lever l'ambiguïté entre l'attribut et le paramètre

- Un paramètre peut avoir le même nom qu'un attribut
- Pour les différencier on précède l'attribut de ***this.***
  - **this** fait référence à l'objet courant
- Dans le constructeur

```
point (int x, int y){
        x = x;
        y = y;
}
```

```
point (int x, int y){
        this.x = x;
        this.y = y;
}
```

II. Classes et Objets

4) Constructeurs

Surcharge des méthodes

- En JAVA toutes les méthodes peuvent être surchargées
  - Même nom
  - Mais doivent avoir des paramètres différents
- La bonne méthode est appelée selon les paramètres

II. Classes et Objets

4) Constructeurs

Constructeurs multiples:

- Le bon constructeur est choisi selon les paramètres

Point p1 = new Point(3, 4);

Point p2 = new Point();

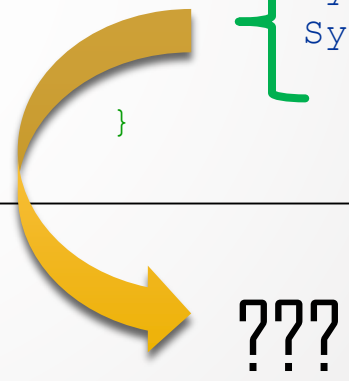
```
class Point{
    //ATTRIBUTS
    int x;
    int y;

    //Construteurs
    Point(int px, int py){
        x = px;
        y = py;
    }
    Point(){
        x = 0;
        y = 0;
    }
}
```

II. Classes et Objets

4) Constructeurs

```
class Point{
     //ATTRIBUTS
     int x; int y;

     //Construteurs
     Point(Point p){
          x = p.x;
          y = p.y;
     }

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

```
public class Test{
          public static void main (String[] args){
                    int a = 8;
                    int b = 4;
                    Point p1 = new Point(a, b);
                    Point p2 = new Point (p1);

                    System.out.println(p1.x);
                    System.out.println(p2.x);

          }
}
```



???

II. Classes et Objets

4) Constructeurs

Exercice:

Soit la classe Compte qui modélise un compte bancaire

1. Ecrire le constructeur de la classe compte
2. Donner un exemple de la création d'un compte en utilisant ce constructeur

II. Classes et Objets

5. Encapsulation et visibilité

II. Classes et Objets

5) Encapsulation et visibilité

En l'état actuel de nos connaissances

- Accès direct aux attributs d'un objet
- Semblables à la notion de structure en C
- Les attributs d'un objet peuvent être modifiés partout dans le code
- **Fortement déconseillé pour éviter les effets de bord (side effect) !**

Solution

- **Principe d'encapsulation**
- Les attributs sont protégés et ne sont pas visibles à l'extérieur de la classe

II. Classes et Objets

5) Encapsulation et visibilité

Visibilité des attributs et des méthodes

- Préciser devant l'attribut / la méthode
  <visibilité> <type> <nomAttribut>
  <visibilité> <typeRetour> <nomMethode>(<param1>, <param2>, ,,,)
- 4 visibilités permettent de protéger les attributs et les méthodes

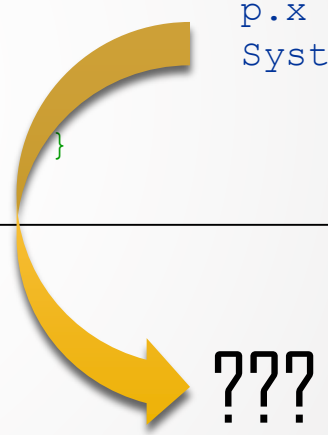
Nom	Mot-clé	Description
publique	**public**	Attribut / Méthode accessible partout
privé	**private**	Attribut / Méthode accessible uniquement au sein de sa propre classe
protégé	**protected**	Attribut / Méthode accessible dans les classes dérivées et les classes du même package
friendly		Visibilité dite « *friendly* » ou « par défaut ». Accès restreint aux classes du même package

II. Classes et Objets

5) Encapsulation et visibilité

```
class Point{
     //ATTRIBUTS
     private int x;
     private int y;

     //Construteurs
     private Point(int xx, int
     yy){
          x = xx;
          y = yy;
     }

     //Methode
     public void afficher(){
          System.out.println("x="+
     x+"y="+y);
     }
}
```

```
public class Test{
          public static void main (String[] args){
                    Point p = new Point (3, 4);

                    p.x = 8;
                    System.out.println(p2.x);

          }
}
```



II. Classes et Objets

5) Encapsulation et visibilité

Accesseurs et modificateurs

- Méthodes qui manipulent les attributs qu'on souhaite rendre accessibles
- **Méthode d'accès** (***getter / accessor***)
  - Renvoyer un attribut privé, sans le modifier
  - En général le nom est la méthode est **get<NomAttribut>**
    Ex. getX()
- **Méthode d'altération** (***Setters / mutator***)
  - Modifier un attribut
  - Prend en paramètre la nouvelle valeur de l'attribut
  - En général, le nom de la méthode est **setXXX**
    Ex. setX(int xx)

Les getters et les setters peuvent être générés automatiquement par Eclipse

II. Classes et Objets

5) Encapsulation et visibilité

```
class Point{
	//ATTRIBUTS
	private int x;
	private int y;

	//Construteurs
	public Point(int xx, int yy){
		x = xx;
		y = yy;
	}
	//Getters et Setters
	public int getX() {
		return x;
	}
	public int getY() {
		return y;
	}
	public void setX (int x) {
		 this.x = x;
	}

	public void setY (int y) {
		 this.y = y;
	}
}
```

```
public class Test{
		public static void main (String[]
args){
				Point p = new Point (3, 4);

				p.setX(8)

				System.out.println(p2.x);

		}
}
```

II. Classes et Objets

5) Encapsulation et visibilité

Possibilité de traiter avant de modifier

```
class Point{
    //ATTRIBUTS
    private int x;
    private int y;

    ...

    public void setX (int x) {
        if(x > 0) {
            this.x = x;
        }

    }

    ...
}
```

II. Classes et Objets

5) Encapsulation et visibilité

Visibilité des classes

- Les classes bénéficient également d'une visibilité
- Déclarée devant le mot-clé **class <visibilité> <nomClasse>**

Modificateur d'accès	Description
final	La classe ne peut pas être étendue par le mécanisme d'héritage. Les classes déclarées final ne peuvent donc pas avoir de classes filles.
private	La classe n'est accessible que dans le fichier dans lequel elle est définie
public	La classe est accessible partout
	La classe est accessible par les classes du même package

Attention, une seule classe public par fichier java !

II. Classes et Objets

5) Encapsulation et visibilité

Ma classe Point correcte

```
public class Point{
	//ATTRIBUTS
	private int x;
	private int y;

	//CONSTRUCTEURS
	public Point(int xx, int yy){
		x = xx;
		y = yy;
	}

	//GETTERS
	public int getX() {
		return x;
	}
	public int getY() {
		return y;
	}
```

```
	//SETTERS
	public void setX (int x) {
		this.x = x;
	}

	public void setY (int y) {
		this.y = y;
	}

	//METHODES
	void deplacer(int dx, int dy){
		x = x+dx;
		y = y+dy;
	}
	void afficher(){
		System.out.println("x="+x+"y="+y);
	}
}
```

II. Classes et Objets

6. Attributs et méthodes de classe

II. Classes et Objets

6) Attributs et méthodes de classe

Commentons quelques instructions:

- Lecture:

```
Point p = new Point(3, 4);
p.afficher();
```

- Ecriture:

```
System.out.println(« toto »);
int val = Math.abs(-9);
```

Qu'observez vous ?

II. Classes et Objets

6) Attributs et méthodes de classe

Un objet est constitué:

- **D'une partie dynamique**
  - Varie durant la vie de l'objet
  - Constitue l'état de l'objet
  - Propre à chaque objet crée

- **D'une partie statique**
  - Commune à toutes les instances de la classe
  - Une modification entraine une modification pour tous les objets de la classe

II. Classes et Objets

6) Attributs et méthodes de classe

La partie statique d'une classe

- Constituée
  - D'atttributs
  - De méthodes
- Attributs et méthodes indépendants des instances
- Communs à tous les objets de la classe
- Utilisables même lorsqu'aucun objet n'est crée
  Ex.
  System.out.println(« toto »)
  Math.random();

II. Classes et Objets

6) Attributs et méthodes de classe

Attributs et méthodes de classes

- Défini en utilisant le mot-clé **static**
  Ex.
  public static int nbPoint
  public static int getNbPoint()

- Pour y accéder, il faut utiliser non pas une référence sur un objet, mais le nom de la classe
  Ex.
  Point.nbPoint = 3;
  int nb = Point.getNbPoint();

Pour tous les points qui seront crées, la valeur **nbPoint** sera la même !

En générale, les attributs et méthodes de classe sont public

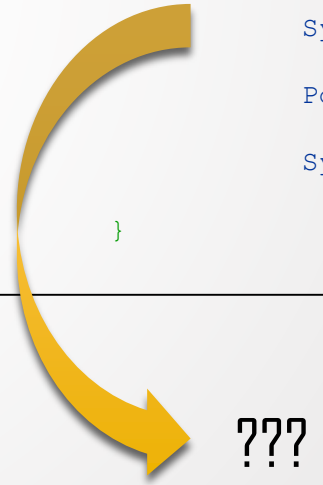
II. Classes et Objets

6) Attributs et méthodes de classe

Exemple d'utilisation: Compter les nombres de points crées

```
public class Point{
	public static int nbPoint = 0;

	//ATTRIBUTS
	private int x;
	private int y;

	//CONSTRUCTEURS
	public Point(int xx, int yy){
		Point.nbPoint = Point.nbPoint + 1;
		x = xx;
		y = yy;
	}

	//GETTERS
	public int getX() {
		return x;
	}
	public int getY() {
		return y;
	}
...
```

```
public class Test{
		public static void main (String[] args){

			System.out.println(Point.nbPoint);

			Point p1 = new Point(0, 0);

			System.out.println(Point.nbPoint);

			Point p2 = new Point(3, 4);

			System.out.println(Point.nbPoint);

		}
}
```



???

II. Classes et Objets

6) Attributs et méthodes de classe

Constantes de classe

- Un attribut de classe peut-être une constante
- Il s'agit de constante liée à une classe
- Défini avec le mot clé **final**
  public final static int VITESSE_MAX = 280

Exercice:

Commenter la méthode

public static void main(String args)

II. Classes et Objets

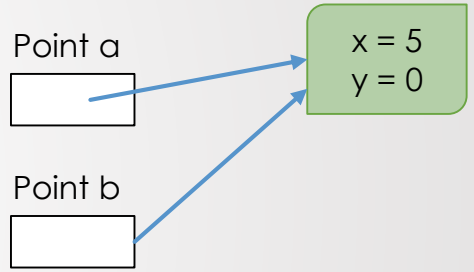
7. Comparaison d'objets

II. Classes et Objets

7) Comparaison d'objets

Affectation d'objet

- Operateur d'égalité revient à copier la référence
  Point a = new Point(5, 0);
  Point b = a;
  Signifie que b et a pointe vers le même objet.
  i.e. Les références sont identiques.



- Toute modification de ***a*** entraine une modification de ***b***, et inversement !

Comparer deux objets

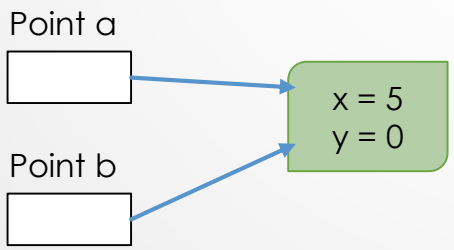
- a == b renvoie la valeur true si les références sont identiques

II. Classes et Objets

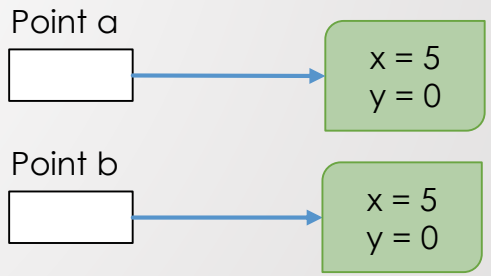
7) Comparaison d'objets

Exercice

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

System.out.println(a == b);
```




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

System.out.println(a == b);
```



II. Classes et Objets

7) Comparaison d'objets

Solution

- Implémenter une méthode de comparaison dédiée
- JAVA fournit déjà un cadre méthode **equals**

```
class Point{
      //ATTRIBUTS
      private int x;
      private int y;

      ...

      public boolean equals(Point p) {
            if(p.x == x && p.y == y) {
                  return true;
            }
            return false;
      }

      ...
}
```

II. Classes et Objets

8. Destruction des objets

II. Classes et Objets

8) Destruction des objets

Destruction des objets

- **Automatiquement lorsque**
  - Par le **Garbage Collector**
  - Plus aucune variable ne référence l'objet
  - Bloc dans lequel il était défini se termine
  - Si l'objet a été affecté à **null**

```
for(int i = 0; i < 100; i++){
        ...
}
//La variable i n'existe que
dans la boucle !
// LA mémoire est libérée
après !
```

- **Manuellement**
  - Sur demande du programmeur par l'instruction ***System.gc()***