Programmation Orienté Objet (POO)

Présenté par : M. Bouderbala

Promotion : 2ème Année LMD Informatique / Semestre N°4

Etablissement : Université de Relizane

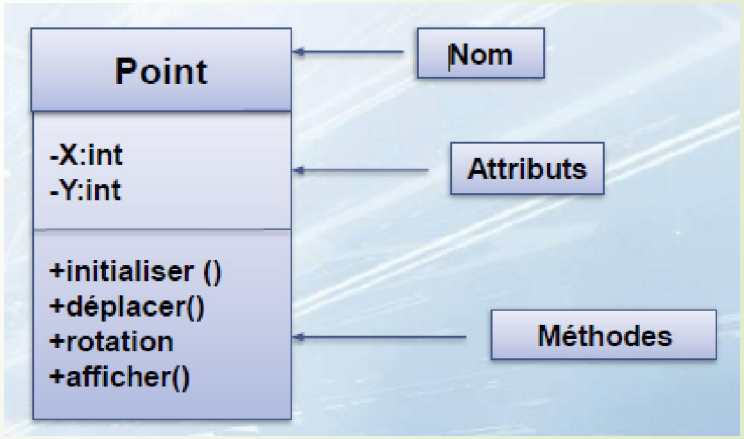
Année Universitaire : 2021/2022

CHAPITRE 2

- Les Classes

La notion de classe

- La classe est une description d'une famille d'objets ayant une même structure et un même comportement.
- Elle est caractérisée par un nom, une composante statique (les attributs) et une composante dynamique (les méthodes).
- Le symbole **– veut dire privée**
- Le symbole **+ veut dire publique**



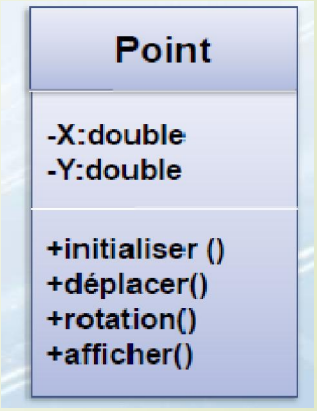
La notion de classe

- La classe peut être considérée comme un **moule** ou un **modèle** à partir du quel on peut créer des objets.
- Un ensemble d'objets ayant les mêmes caractéristiques et les mêmes comportements appartient à la même classe :
  - Tous les Enseignants appartiennent à la classe Enseignant.
  - Tous les étudiants appartiennent à la classe étudiant

- Une classe ne peut pas être utilisée pour exécuter des programmes. Pour cela il faut l'instancier (créer des objets).
- La classe sert simplement à décrire la structure des objets (leurs attributs et leurs méthodes)

La notion de classe

- Supposons qu'un point est représenté par deux coordonnées entières: abscisse et ordonnée.
- Nous devons donc définir deux attributs dans la classe Point
  - **X**
  - **Y**
- Supposons que nous souhaitons pouvoir:
  - Initialiser les coordonnées d'un point.
  - Déplacer ce point dans le plan.
  - Faire une rotation de ce point.
  - Afficher les coordonnées de ce point.
- Nous devons donc disposer de quatre méthodes dans la classe Point
  - Initialiser ; Déplacer ; Rotation ; afficher



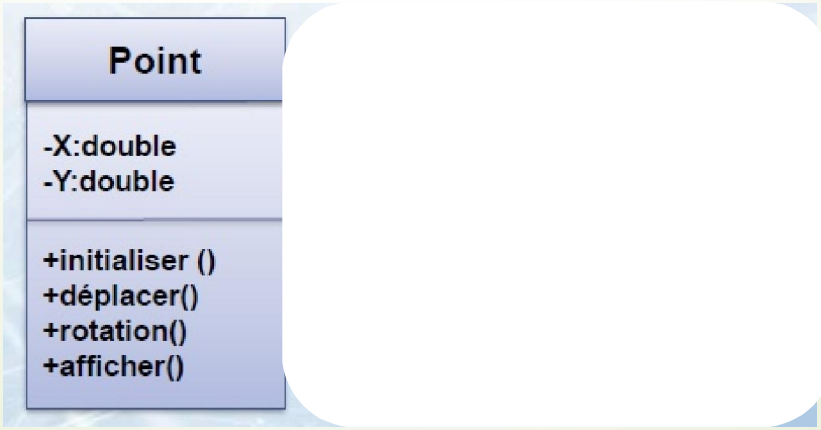
Définition d'une classe Point (code en Java)

Le modèle général de définition d'une classe en Java est :

```
class Point
{
// instructions de définition des champs et des méthodes de la classe
}
```

Les Attributs

- Les attributs sont les données de la classe.
- Un attribut peut être :
  - de type élémentaire: entier, réel (float, double), booléen, caractère.
- Les déclarations des attributs peuvent se faire n'importe où à l'intérieur de la classe, mais en général on les déclare au début ou à la fin de la classe.



Définition des attributs de la classe Point (code en Java)

```
class Point {
Private double x; // abscisse
Private double y; // ordonnée
// définition des méthodes
}
```

- Le mot clé **private** précise que les champs x et y ne sont pas accessibles de l'extérieur de la classe (c à d en dehors de ses propres méthodes). A l'opposé on utilise le mot clé **publique**.

Les méthodes (1)

- Une méthode est un regroupement d'instructions semblable aux fonctions et procédures sauf qu'**elle s'exécute toujours sur un objet précis.**
- Seules les méthodes de la classe ont droit à l'accès aux attributs (privés) de cette classe.
- Une méthode peut être:
  - ✓ **publique:** accessible en dehors de la classe (les autres objets) c.à.d. visible dans l'interface.
  - ✓ **privée:** nécessaire à l'implémentation interne de la classe mais n'est pas visible dans l'interface.

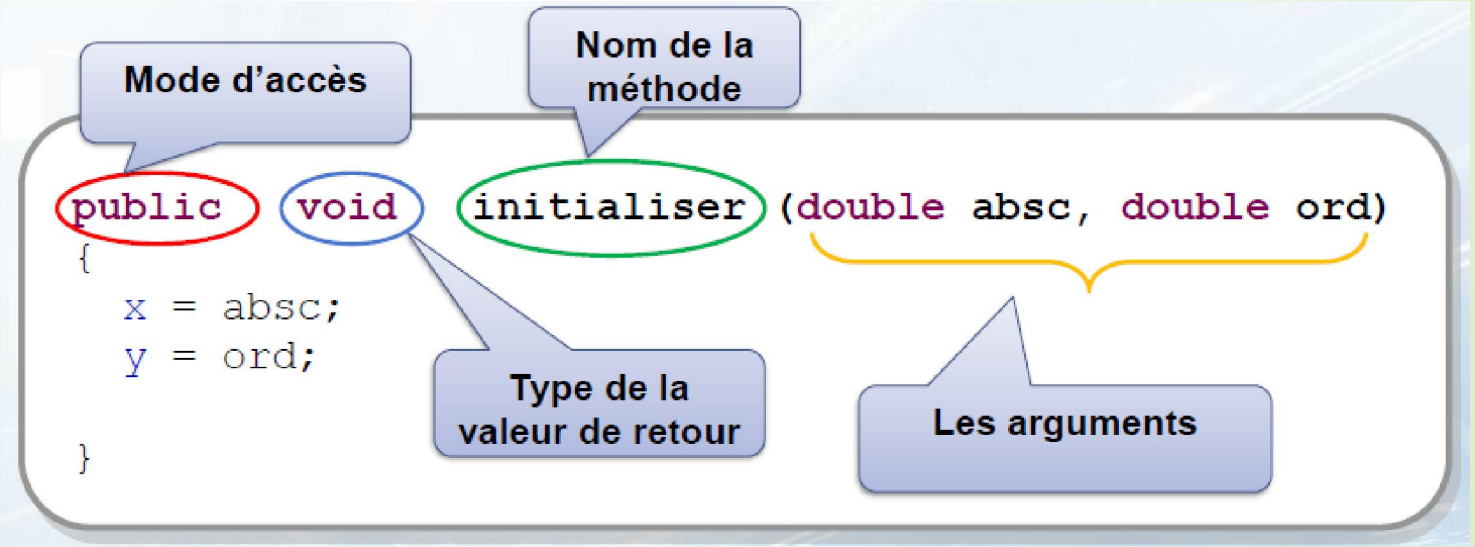
Les méthodes (2)

- Les méthodes ont toujours un type retourné «**Type Retourné».**
- Le **TypeRetourné indique le type de la valeur que la méthode «retourne» quand on l'invoque.**
- C'est le **TypeRoutourné est void signifie que la méthode ne retourne rien (pas de return à la fin, équivalent au procédure)**
- Elle peut recevoir un ou plusieurs arguments entre parenthèses, qu'elle utilisera en cours de son exécution.
- La signature d'une méthode:
  - ✓ le nom de la méthode.
  - ✓ la liste des types de ses arguments.
  - ✓ Pour invoquer une méthode, il faut toujours respecter sa signature.

La notion de classe

Définition des méthodes de la classe Point

(code en Java)



La notion de classe

Le code complet en Java de l'exemple Point

```
class Point {
     private double x;
     private double y;

     public void initialiser (double absc, double ord)
     {x = absc;
      y = ord;
     }
     public void deplacer (double dx, double dy)
     {x = x + dx;
      y = y + dy;
     }
     public void rotation (double alpha)
     {double r = Math.sqrt(x*x+y*y);
      double t = anat2(x,y);
      t += alpha;
      x = r=math.cos(t);
      y = r=math.sin(t);
     }
     public void afficher()
     {
       System.out.println("je suis un point de coordonnées:"+x+" "+y);
     }
}
```

Le nom de la classe

Les attributs

Les méthodes

La notion d'objet

- Les objets sont les moulages fabriqués à partir de la classe.
- Un objet d'une classe est aussi appelé **instance de cette classe.**
- Une fois qu'on a la classe, on peut créer autant d'objets (instances) que l'on veut.
- Ces objets auront:
  - Des identités différentes.
  - Des états définis par les mêmes attributs mais avec des valeurs pouvant être différentes.
  - Un comportement identique (Mêmes méthodes).

Classe et objet

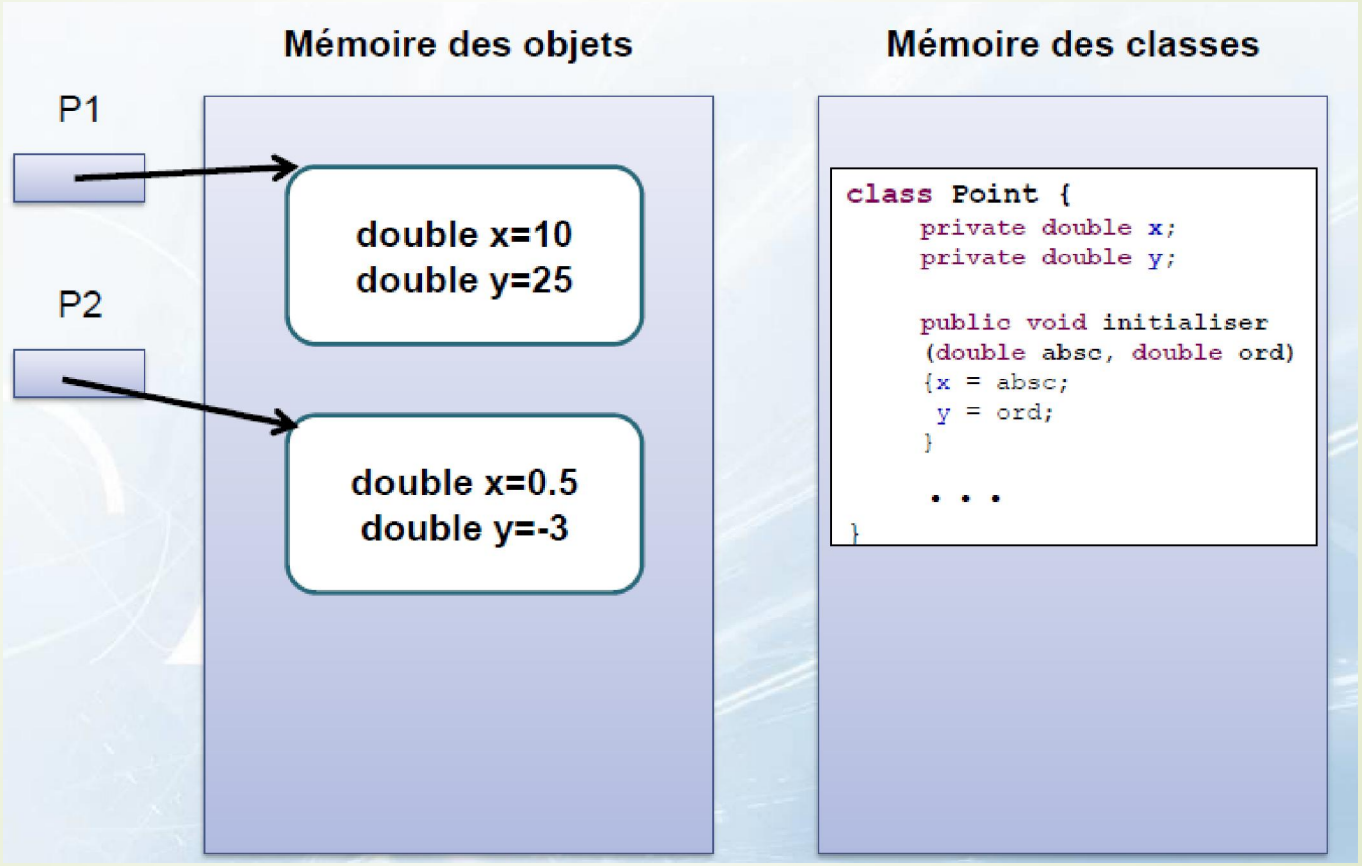
La Classe est :

- un modèle.
- un texte logiciel.
- existe dans le code source.
- abstraite

L'Objet est :

- une concrétisation d'un tel modèle.
- une structure de données créée dynamiquement.
- existe seulement dans la mémoire durant l'exécution
- concret

Classe et objet



Exercice :

Une classe :

Est une sorte de moule ou de matrice à partir duquel sont engendrés les objets réels qui s'appellent des instances de la classe considérée.

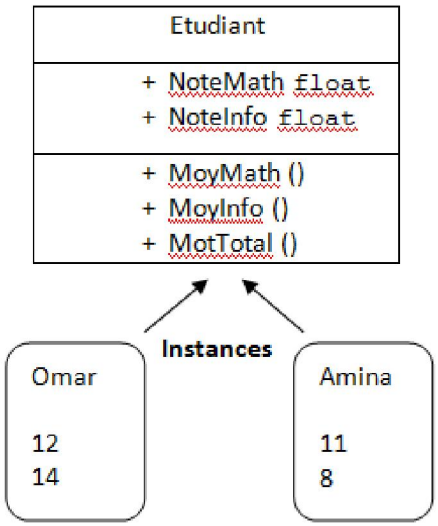
Elle est composée:

•D'attributs (ou champs, ou variables d'instances).

Les attributs de la classe décrivent la structure de ses instances (les objets).

Supposons que chaque étudiant soit caractérisé par sa note en mathématiques (NoteMath) et sa note en informatique (NoteInfo). Un étudiant doit pouvoir effectuer éventuellement des opérations de calcul de ses moyennes dans ces deux matières (MoyMath, MoyInfo) et connaître sa moyenne générale calculée à partir de ces deux notes (MoyTotale).

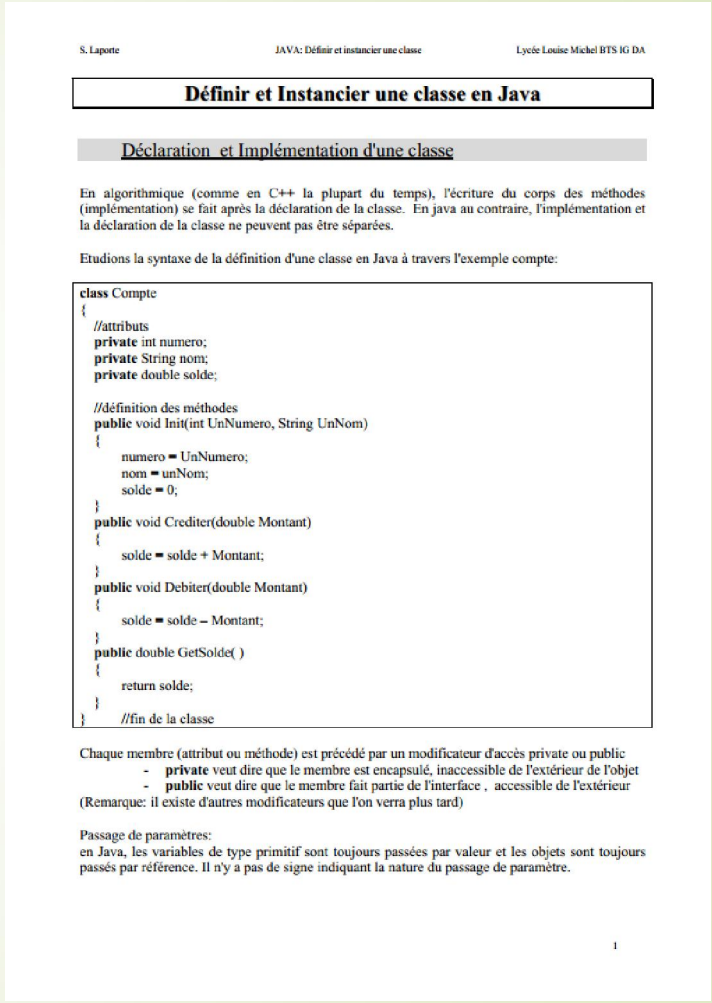
inir La Classes « étudiant »




```
class Etudiant {
    public float NoteMath;
    public float NoteInfo;
    public void  MoyMath(float UneNote){
    ...
    }
    public void MoyInfo(float UneNote){
    ...
    }
    public float MoyTotale(){
    ...
    }
}// fin classe Etudiant
```

La classe Etudiant a été créée
que les attributs *NoteMa*
Les méthodes de cette
exemple *MoyMath*, *MoyInfo*, M
Nous avons créé deux
(deux instances : Omar et Ar
Etudiant.

Exemple : Compte Bancaire



Définir et Instancier une classe en Java

Déclaration et Implémentation d'une classe

En algorithmique (comme en C++ la plupart du temps), l'écriture du corps des méthodes (implémentation) se fait après la déclaration de la classe. En java au contraire, l'implémentation et la déclaration de la classe ne peuvent pas être séparées.

Etudions la syntaxe de la définition d'une classe en Java à travers l'exemple compte:

```
class Compte
{
   //attributs
   private int numero;
   private String nom;
   private double solde;

   //définition des méthodes
   public void Init(int UnNumero, String UnNom)
   {
        numero = UnNumero;
        nom = unNom;
        solde = 0;
   }
   public void Crediter(double Montant)
   {
        solde = solde + Montant;
   }
   public void Debiter(double Montant)
   {
        solde = solde – Montant;
   }
   public double GetSolde( )
   {
        return solde;
   }
}       //fin de la classe
```

Chaque membre (attribut ou méthode) est précédé par un modificateur d'accès private ou public
- **private** veut dire que le membre est encapsulé, inaccessible de l'extérieur de l'objet
- **public** veut dire que le membre fait partie de l'interface , accessible de l'extérieur

(Remarque: il existe d'autres modificateurs que l'on verra plus tard)

Passage de paramètres:
en Java, les variables de type primitif sont toujours passées par valeur et les objets sont toujours passés par référence. Il n'y a pas de signe indiquant la nature du passage de paramètre.

L'instanciation

- **L'instanciation est la concrétisation d'une classe en un objet «concret».**

- Dans nos programmes Java nous allons définir des classes et instancier ces classes en des objets qui vont interagir. Le fonctionnement du programme résultera de l'interaction entre ces objets «instanciés».

- En Programmation Orientée Objet, on décrit des classes et l'application en elle-même va être constituée des objets instanciés, à partir de ces classes, qui vont communiquer et agir les uns sur les autres

L'instanciation

- Pour instancier ou créer un nouvel objet, nous devons lui allouer de l'espace mémoire.
- chaque objet aura son espace où seront stockées les valeurs de ses attributs
- Si nous créons 1000 objets de type Point, nous aurons 1000 emplacements pour la variable x et 1000 emplacements pour la variable y.
- Les méthodes ne sont quant à elles pas dupliquées (inutile).

L'instanciation

- On peut instancier (crée) des objets d'une classe et leur appliquer à volonté les méthodes publiques défini dans la classe.
- Avant de créer un objet, il faut d'abord le déclarer.

Nom_de_la_classe identificateur;

- Cette déclaration ne réserve pas d'emplacement pour un objet de type **Nom_de_la_classe**, mais seulement un emplacement pour une référence à un objet de type **Nom_de_la_classe**.

L'instanciation

- Pour créer un emplacement pour l'objet lui-même, il faut utiliser le mot clé **new.**

  identificateur = new Nom_de_la_classe();

- Dans cette instruction, l'opérateur **new** crée un objet de type **Nom_de_la_classe** et fournit sa référence. Celle-ci est affectée à la variable a déclarée auparavant:

  Nom_de_la_classe identificateur;

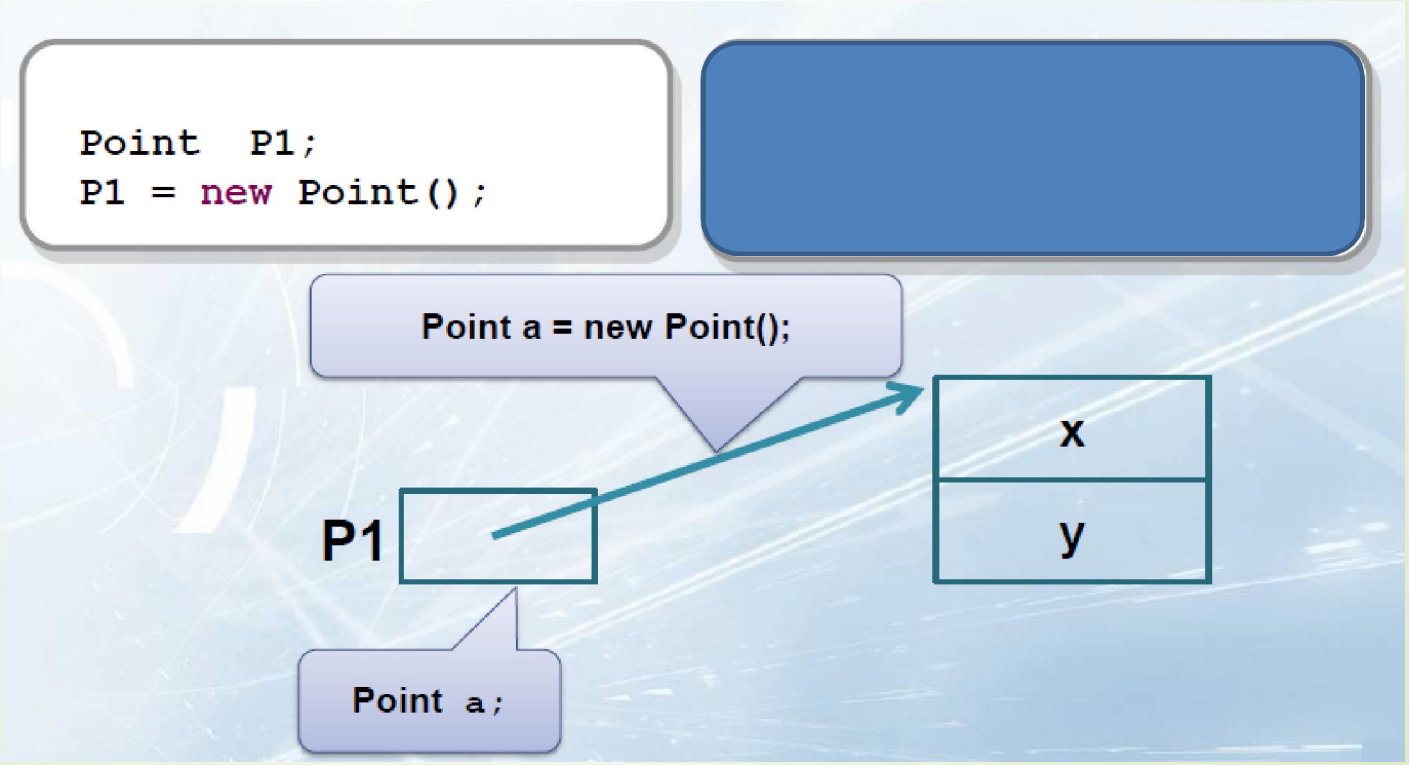
  identificateur = new Nom_de_la_classe();

Est équivalent à:

Nom_de_la_classe identificateur = new Nom_de_la_classe();

L'instanciation

Exemple: la classe Point (exemple en java)



L'utilisation des objets

- Une fois l'objet créé, on peut lui appliquer n'importe quelle méthode définit dans la classe.

Exemple: **P1.initialiser(-3,2);**

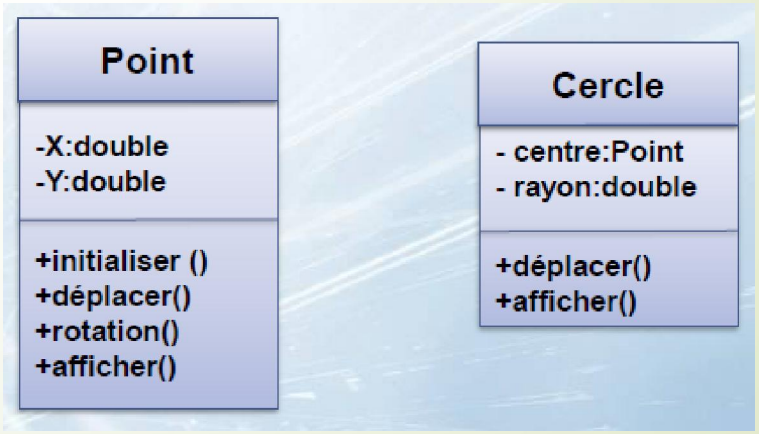
- C'est un appel à la méthode initialiser qui va affecter la valeur -3 au champs **x et la valeur 2 au champs y**

Exemple création et utilisation d'un objet

- ✓ Imaginons que nous souhaitons créer une classe **Cercle permettant de représenter des cercles définis par un centre (objet de type Point) et un rayon de type flottant.**

Les fonctionnalités de cette classe se limitent à :

- L'affichage des caractéristiques d'un cercle (coordonnées du point du centre et le rayon)
- Le déplacement du centre



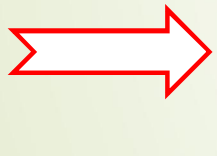
Exemple création et utilisation d'un objet

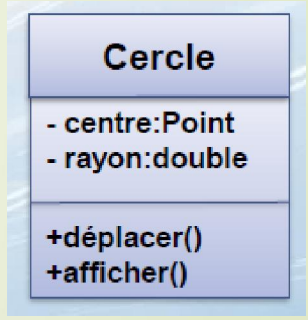
Exemple classe cercle: code en java

```
public class cercle
{ private Point centre;
  private float rayon;

  public Cercle (int x, int y, float r)
  { centre = new Point (x, y);
    rayon = r;
  }

  public void affiche ()
  {// afficher le centre et le rayon du cercle
  }
}
```





Les constructeurs (1)

- Un constructeur est une méthode spéciale utilisée pour crée un nouvel objet. Le constructeur se caractérise par:
  - Il a le même nom que celui de la classe,
  - Ne retourne aucune valeur,
  - N'a aucun type de retour, même pas void
- L'objet obéira à la structure définie dans la classe, c'est-à-dire qu'il aura les attributs et le comportement définis dans la classe. Cela nécessite la réservation d'un espace mémoire pour la mémorisation de l'état.

Les constructeurs (2)

- Le constructeur sert souvent à initialiser les différents attributs de l'objet.
- Le constructeur est **exécuté à la création de l'objet.**
- C'est souvent la méthode la plus surchargée de la classe.
- Une version par défaut est toujours fournie par les langages de programmation (Si le développeur ne définit aucun constructeur, c'est le constructeur par défaut qui est exécuté). **P1 = new Point ();**
- Dès qu'une classe possède au moins un constructeur, le constructeur par défaut ne peut plus être utilisé . Sauf si un constructeur sans arguments a été définit.

Les constructeurs (3)

Retour à l'exemple: Classe Point

```
class Point {
     private double x;
     private double y;

     public Point (double absc, double ord)
     {x = absc;
      y = ord;
     }
     public void deplacer (double dx, double dy)
     {x = x + dx;
      y = y + dy;
     }
     public void rotation (double alpha)
     {double r = Math.sqrt(x*x+y*y);
      double t = anat2(x,y);
      t += alpha;
      x = r=math.cos(t);
      y = r=math.sin(t);
     }
     public void afficher()
     {
       System.out.println("je suis un point de coordonnées:"+x+" "+y);
     }
}
```

C'est le constructeur de la méthode Point. Maintenant, la méthode initialise devient inutile. On peut la supprimer

Les constructeurs (4)

Retour à l'exemple: Classe Point

```
class Point {
     private double x;
     private double y;

     Public Point (double absc, double ord)
     {x = absc;
      y = ord;
     }
     public void deplacer (double dx, double dy)
     {x = x + dx;
      y = y + dy;
     }
     public void rotation (double alpha)
     {double r = Math.sqrt(x*x+y*y);
      double t = anat2(x,y);
      t += alpha;
      x = r=math.cos(t);
      y = r=math.sin(t);
     }
     public void afficher()
     {
       System.out.println("je suis un point de coordonnées:"+x+" "+y);
     }
}
```

Est il permis d'écrire

Point a = new point ()
?

Les constructeurs (5)

Retour à l'exemple: Classe Point

```
class Point {
      private double x;
      private double y;

      public Point ()
      {
      }
      Public Point (double absc, double ord)
      {x = absc;
       y = ord;
      }
      public void deplacer (double dx, double dy)
      {x = x + dx;
       y = y + dy;
      }
      public void rotation (double alpha)
      {double r = Math.sqrt(x*x+y*y);
       double t = anat2(x,y);
       t += alpha;
       x = r=math.cos(t);
       y = r=math.sin(t);
      }
      public void afficher()
      {
        System.out.println("je suis un point de coordonnées:"+x+" "+y);
      }
}
```

Est il permis d'écrire

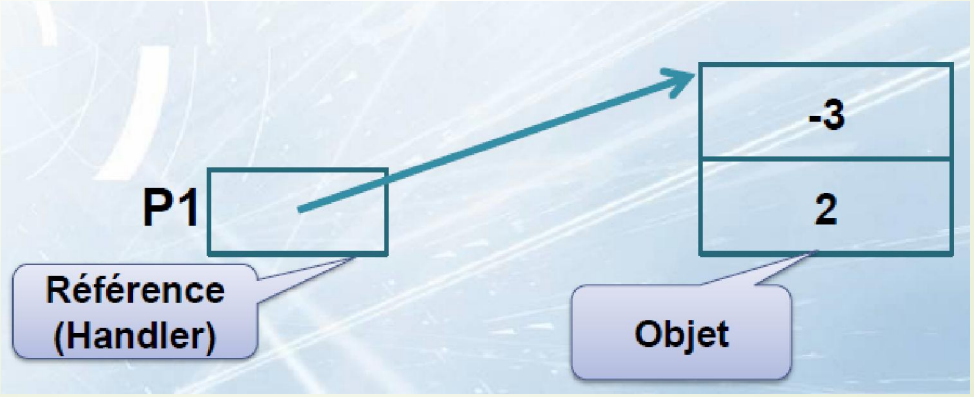
Point a = new point ()
?

Le Destructeur

- Le destructeur est aussi une méthode spéciale de la classe mais qui est exécutée à la mort de l'objet.
- Pour libérer l'espace mémoire occupé par l'objet un destructeur est invoqué (finaliseurs: finalizers en java).
- Java assure la gestion automatique de la mémoire à travers le ramasse-miettes (GarbageCollector en anglais) qui fonctionne selon le principe suivant:
  - ✓ A tout instant, on connait le nombre de références à un objet.
  - ✓ Lorsqu'un objet n'est plus référencé (il n'existe aucune référence sur lui), on est certain que le programme ne pourra plus y accéder. Il est donc possible de libérer l'emplacement correspondant

Le référent d'un objet (1)

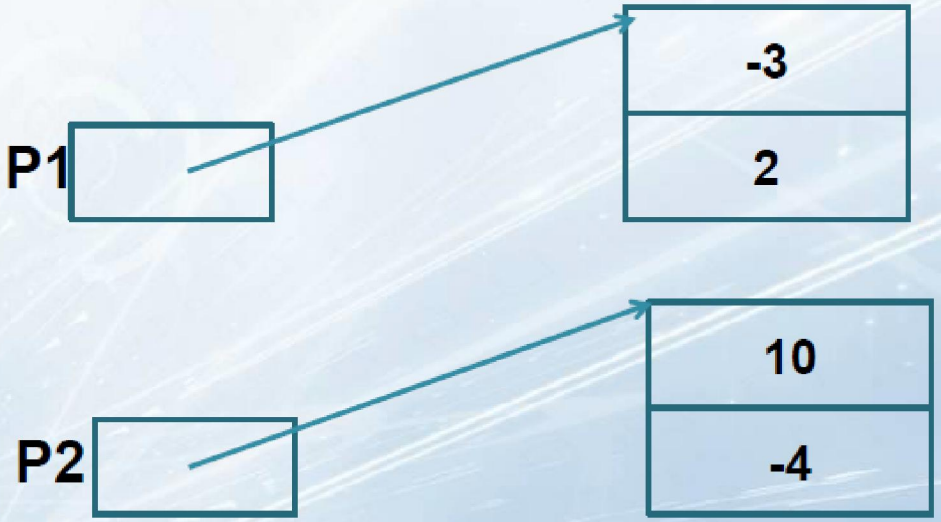
- Chaque objet doit avoir un nom (référent) « qui lui est propre » pour l'identifier.
- La référence (handler) permet d'accéder à l'objet, mais n'est pas l'objet lui-même. Elle contient l'adresse de l'emplacement mémoire dans lequel est stocké l'objet.
- La référence est, en quelque sorte, un pointeur sur l'espace mémoire allouer pour stocker l'état de l'objet.



Le référent d'un objet (2)

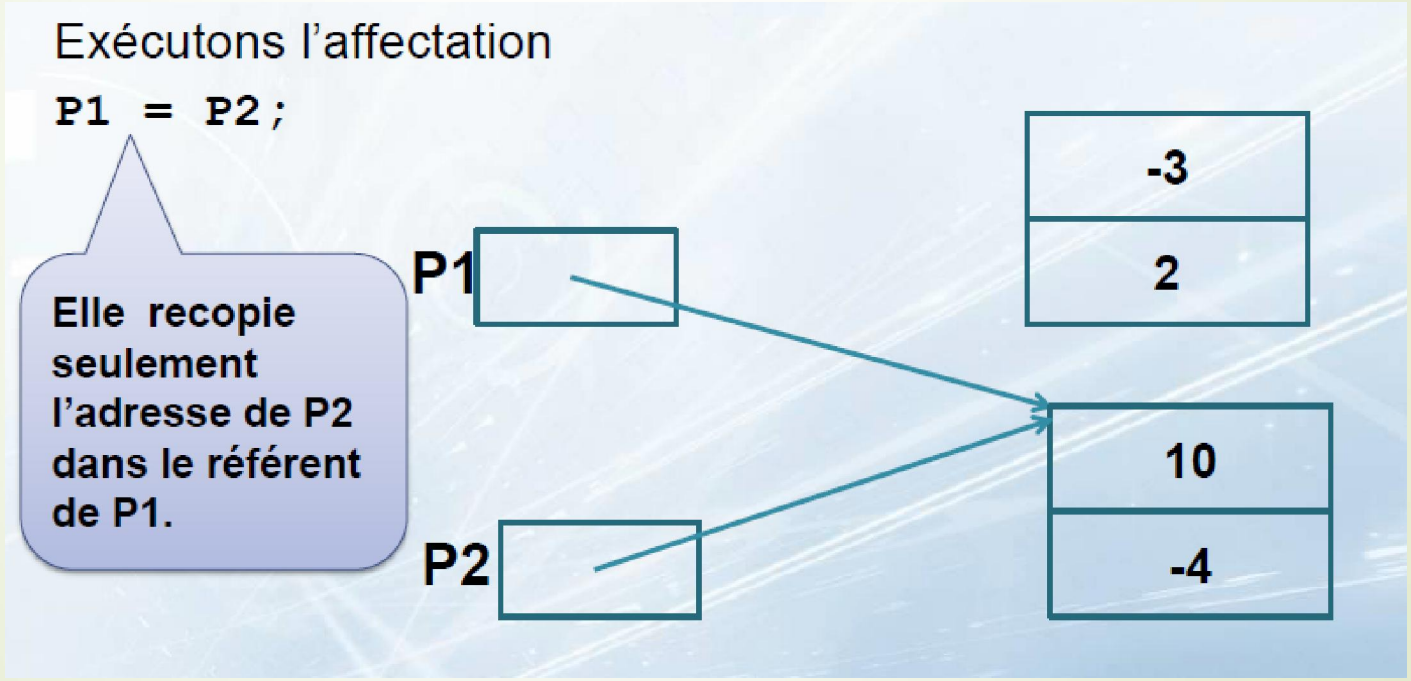
Premier exemple

```
Point P1,P2;
P1 = new Point (-3,2);
P2 = new Point (10,-4);
```



Le référent d'un objet (3)

Premier exemple: Affectation d'objets

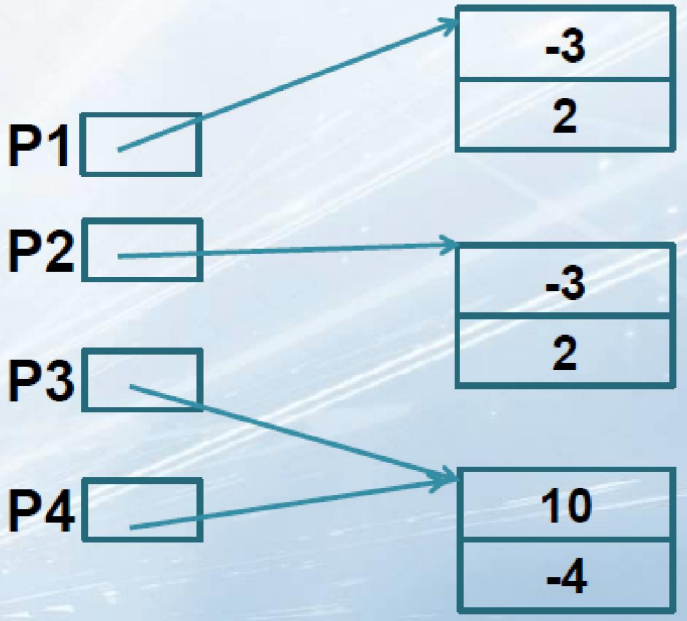


Le référent d'un objet (4)

Premier exemple : Comparison d'objets

```
Point P1, P2, P3,P4;
P1 = new Point (-3,2);
P2 = new Point (-3,2);
P3= new Point (10,-4);
P4= P3;
P1 == P2 ?
P3 == P4 ?
```

Quand on compare 2 variables d'un type primitif on compare leur valeur. Quand on compare 2 objet avec les opérateurs on compare leur référence (leur adresse en mémoire).
La méthode **equals** fait un test sur le contenu de l'objet



L'auto-référence

- Dans le traitement de l'une de ses méthodes un objet peut avoir à s'envoyer un message (pour accéder à un de ses attributs ou invoquer une des ses méthodes). Pour cela, il utilise le mot clé **this.**

- L'utilisation du mot clé **this** permet aussi d'éviter l'ambiguité.
- Si pas d'ambiguité, this peut être omis :

Exemple : le constructeur Point

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

```
public Point(double absc, double ord)
{ x = absc;
  y = ordy;
}
```

Eléments de conception de classes (1)

Parmi les méthodes que comporte une classe, on distingue:

- Les constructeurs
- Les méthodes d'accès ou bien les accesseurs (Accessor, *Getter): fournissent les valeurs de certains champs privés sans les modifier.*
  - Souvent on utilisent des noms de la forme getXXX
- Les méthodes d'altération ou bien les mutateurs (Mutator, *Setter): Modifient les valeurs de certains champs privés.*
  - Souvent on utilise des noms de la forme setXXX.

Getters et Setters

- Deux types de méthodes ont un statut particulier en Java : les *getters* et les *setters* . Afin de respecter le principe d'encapsulation, les champs non statiques d'une méthode sont déclarés private. Afin de pouvoir lire ces champs, et éventuellement d'en modifier la valeur, on ajoute à la classe un getter et un setter pour chacun de ses champs private.
- Le nom de ces deux méthodes est fixé par une convention universellement suivie, et sur laquelle de nombreux frameworks s'appuient. Il convient donc de la respecter !
- Un getter commence par get, suivi d'un nom, qui est en général le nom du champ, commençant par une majuscule. Il retourne une valeur en général du même type que le champ associé.
- Le setter associé commence par set, suivi du même nom que le get. Il prend en paramètre un unique argument, du même type que le type de retour du getter associé.

La notion d'accesseur : **Getters**

- Un accesseur est une méthode permettant de récupérer le contenu d'une donnée membre protégée. Un accesseur, pour accomplir sa fonction :
  - doit avoir comme type de retour le type de la variable à renvoyer
  - ne doit pas nécessairement posséder d'arguments

- Une convention de nommage veut que l'on fasse commencer de façon préferrentielle le nom de l'accesseur par le préfixe *get*, afin de faire ressortir sa fonction première.

```
class A{
private int age;

public int getAge(){
return age;
}
}
```

La notion de mutateur : **Setters**

- Un mutateur est une méthode permettant de modifier le contenu d'une donnée membre protégée. Un mutateur, pour accomplir sa fonction :
  - doit avoir comme paramètre la valeur à assigner à la donnée membre. Le paramètre doit donc être du type de la donnée membre
  - ne doit pas nécessairement renvoyer de valeur (il possède dans sa plus simple expression le type *void*)
- Une convention de nommage veut que l'on fasse commencer de façon préferrentielle le nom du mutateur par le préfixe set.

```
Class A{
private int age;

public void setAge(int nouveauAge){
age = nouveauAge;

}
}
```

Exemples

- +\+\public class Ville.docx
- +\+\public class MoyenDeTransport.docx

Eléments de conception de classes (2)

- Pour une bonne conception de vos classes, il existe quelques règles qui ne sont pas obligatoires mais qu'il est vivement conseillé de suivre.

1. Respecter le principe d'encapsulation en déclarant tous les champs (attributs) privés.
2. S'appuyer sur la notion de **contrat** qui considère qu'une classe est caractérisée par :
   - Les entêtes de ses méthodes publiques (interface)
   - Les comportements de ces méthodes.
3. Le reste (champs, méthodes privées et corps) est appelé **implémentation et doit rester privé.**
4. Le contrat définit ce que fait la classe, alors que l'implémentation décrit comment elle le fait.

Les modificateurs d'accès (1)

- La déclaration d'une classe, d'une méthode ou d'un attribut peut être précédée par un modificateur d'accès (visibilité).
- Un modificateur indique si les autres classes de l'application pourront accéder ou non à l'item qui peut être une classe, une méthode ou un attribut.
- L'utilisation des modificateurs permet au programmeur de contrôler la visibilité des différents items et permet d'empêcher que des actions illégales soient effectuées sur les items.

Les modificateurs d'accès (2)

- Parmi les modificateurs :

- **public:** toutes les classes peuvent accéder à l'item. La déclaration de variables publiques est contraire au principe d'encapsulation.
- **protected:** seules les classes dérivées et les classes du même package peuvent accéder à l'item.
- **Private:** un item private (privé) est accessible uniquement au sein de la classe dans laquelle il est déclaré. Ces éléments ne peuvent être manipulés qu'à l'aide de méthode spécifiques appelés accesseur et mutateur
- **(par défaut):** sans modificateur d'accès, seules les classes du même package peuvent accéder à l'item.

Les modificateurs d'accès (3)

- Autres modificateurs :

- **static:** indique, pour une méthode, qu'elle peut être appelée sans instancier sa classe i.e. indépendamment de tout objet (méthode de la classe).Pour un attribut, qu'ils'agit d'un attribut de classe, et que sa valeur est donc partagée entre les différentes instances de sa classe (variable globale ou variable de classe).

- **final:** une variable déclarée final est en fait une constante,il n'est plus possible de la modifier. Les méthodes déclarées final ne peuvent pas être remplacée dans une sous classe. Les classes déclarées final ne peuvent pas avoir de sous-classe

Surcharge de méthode (1)

- On parle de surcharges (ou surdéfinition) lorsqu'un symbole possède plusieurs significations entre lesquelles on choisit en fonction du contexte.
- En java, il est possible de surcharger les méthodes d'une classe y compris celles qui sont statiques (méthode avec **static).** Plusieurs méthodes peuvent porter le même nom à condition que le nombre et le type de leurs arguments permettent au compilateur d'effectuer son choix,

Surcharge de méthode (2)

Règles générales

A la rencontre d'un appel de méthode, le compilateur cherche toutes les méthodes acceptables et choisit la meilleure si elle existe.

Pour qu'une méthode soit acceptable il faut:

- Qu'elle dispose du nombre d'arguments voulus
- Que les types des arguments effectifs (valeur fournie lors de l'appel) soit compatible avec les types des arguments muets
- Qu'elle soit accessible (une méthode privée ne sera pas acceptable à l'extérieur de la classe)

Surcharge de méthode (3)

Le choix de la méthode se fait comme suit:

- Si aucune méthode n'est acceptable Erreur de compilation
- Si une seule méthode est acceptable, elle sera utilisée
- Si plusieurs méthodes sont acceptables, il choisit la meilleure
- Si il y a ambigüité Erreur de compilation

Passage de paramètres

- Dans les langages de programmation on rencontre deux façons d'effectuer le transfert d'information
  - Par valeur: la méthode reçoit une copie de la valeur de l'argument effectif, elle travaille sur cette copie et la modifie sans que cela ne modifie l'argument effectif
  - Par adresse (ou par référence): la méthode reçoit l'adresse de l'argument effectif sur lequel elle travaille directement et peut modifier sa valeur
- Java transmet toujours les informations par valeur.
  - Que se passe t-il dans le cas où une variable est de type objet ?
  - Qu'est ce qui est transmis et qu'est ce qui peut être modifié ?

Variable de type Objet :

- Dans le cas où la variable manipulée est de type Objet, son nom représente sa référence, donc la méthode reçoit une copie de sa référence.
- La méthode peut donc modifier l'objet concerné