جامعة هواري بومدين
للعلوم والتكنولوجيا
U S T H B

Cours POO
(Programmation Orientée Objet)

S. BOUKHEDOUMA

USTHB – FEI – département d'Informatique
Laboratoire des Systèmes Informatiques -LSI
sboukhedouma@usthb.dz

Introduction

La programmation orientée objet est un style de programmation qui repose sur deux concepts clés:

Le concept de classe
Le concept d'objet

Un objet est créé à partir d'une classe qui le définit

Introduction

Objectifs de la POO

- Faciliter le développement de programmes
- Développer des applications modulaires (créer des classes séparées)
- Faciliter la maintenabilité et l'extension d'applications (ajouter/modifier des classes)
- Favoriser la réutilisation de code (une classe peut être utilisée dans différents programmes)

Principes de la POO

- Encapsulation
- Abstraction
- Héritage
- Polymorphisme

Avantages de la POO

-le principe **d'*encapsulation***: les détails de l'objet sont encapsulés dans une structure de classe, le code constituant l'état et le comportement de l'objet est caché à l'utilisateur

→ diminution du risque d'erreurs

- **la *clarté du code*:** les traitements (les requêtes) sont séparés de la structure des classes. Ceci permettra d'éviter toute duplicité de code.

Avantages de la POO

-***maintenance et évolutivité*:** modifier une classe (ajouter/ supprimer/ modifier) n'affectera pas les autres (du moins, celles qui n'y font pas appel)

- ***réutilisation***: l'indépendance des entités (les classes) les unes des autres permet leur réutilisation dans diverses applications

Plan du cours

Chapitre I – Eléments de base du langage java **(TP)**

Chapitre II – Classes, Objets et Packages

Chapitre III – Modificateurs et Niveaux de visibilité

Chapitre IV – Chaines de caractères et Enumérations **(TP)**

Chapitre V – Héritage et Polymorphisme

Chapitre VI - Classes abstraites et Interfaces

Chapitre VII - Les collections

Chapitre VIII – Les exceptions

جامعة هواري بومدين
للعلوم والتكنولوجيا
U S T H B

Chapitre II

Classes, Objets et Packages

S. BOUKHEDOUMA

USTHB – FEI – département d'Informatique
Laboratoire des Systèmes Informatiques -LSI
sboukhedouma@usthb.dz

Introduction

Un programme java (ou même autre langage orienté objet) se présente sous forme d'un **ensemble d'*objets* créés à partir de *classes***

Ces classes sont définies de manière séparée du programme et peuvent être utilisées dans différents programmes (principe de réutilisation).

Les objets possèdent des caractéristiques propres (***attributs*)** et sont manipulés par des ***méthodes*** (traitements qui sont analogues de *fonctions/procédures* dans la programmation fonctionnelle)

Le concept de classe

Une *classe* est une structure qui factorise la description d'une *infinité* d'entités appelées *objets* de la classe.

Les objets sont des éléments du monde réel de l'étude, il peut s'agir de n'importe quelle entité **selon le domaine d'application**:

Application commerciale: **Client, Produit, Facture,** …

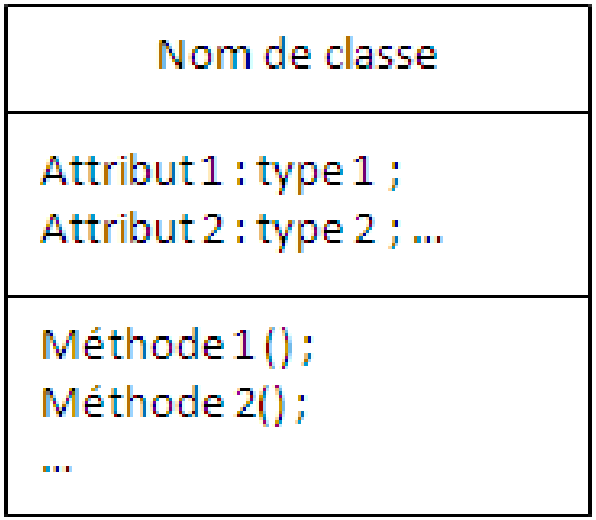
Application de scolarité: **Etudiant, Enseignant, Module, Local, Filière, …**

Application de géométrie: **Point, Carré, Cercle, Forme, …**

les objets de la classe.

Le concept de classe

La structure d'une classe est composée de trois parties :
le *nom de la classe*,
la **liste des *attributs*** décrivant les objets de la classe et
la **liste des *méthodes*** s'appliquant sur les objets de la classe.



Le concept de classe

Un attribut est une propriété de l'objet ayant un type

Exemples:
Matricule, Nom, …
CodProduit, PrixUnitaire, …

La liste des attributs est déterminée en répondant à la question: ***Par quoi peut-on caractériser les objets de cette classe?***

Une méthode est un traitement ayant une fonctionnalité précise sur les objets de la classe

Le concept de classe

Une méthode est un traitement ayant une fonctionnalité précise sur les objets de la classe

La liste des méthodes est déterminée en répondant à la question: ***Quels traitements peut-on effectuer sur les objets de la classe caractériser les objets de cette classe?***

Exemples:

Afficher()
Modifier ()
Calculer ()…

Le concept de classe

Remarque1

Le nom d'une classe **doit être significatif** en désignant clairement l'objet du monde réel (Etudiant, Produit, Cercle, …)

Les noms des attributs et des méthodes doivent être aussi significatifs

Un attribut doit désigner clairement la propriété de l'objet
Une méthode doit désigner clairement le traitement à effectuer.

Le concept de classe

Remarque 2

La classe est une structure qui encapsule:

la partie **« attributs »** : les caractéristiques de l'objet

Et

La partie **« méthodes»**: les traitements à effectuer sur l'objet

⟹ **Principe d'encapsulation**

Ceci permet de protéger les objets des traitements non permis

On dit que la classe est un type structuré amélioré

Le concept de classe

Remarque 2

La classe est une structure qui encapsule:

la partie **« attributs »** : les caractéristiques de l'objet

Et

La partie **« méthodes»**: les traitements à effectuer sur l'objet

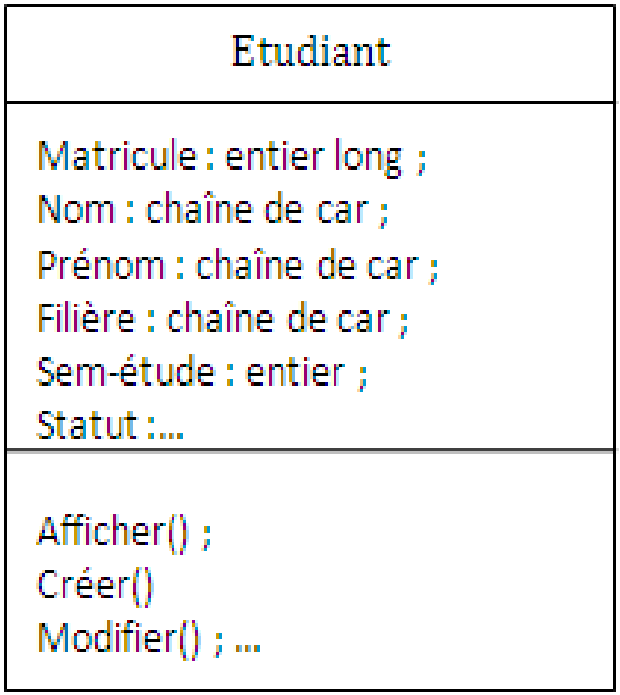
⟹ **Principe d'encapsulation**

Ceci permet de protéger les objets des traitements non permis

On dit que la classe est un type structuré amélioré

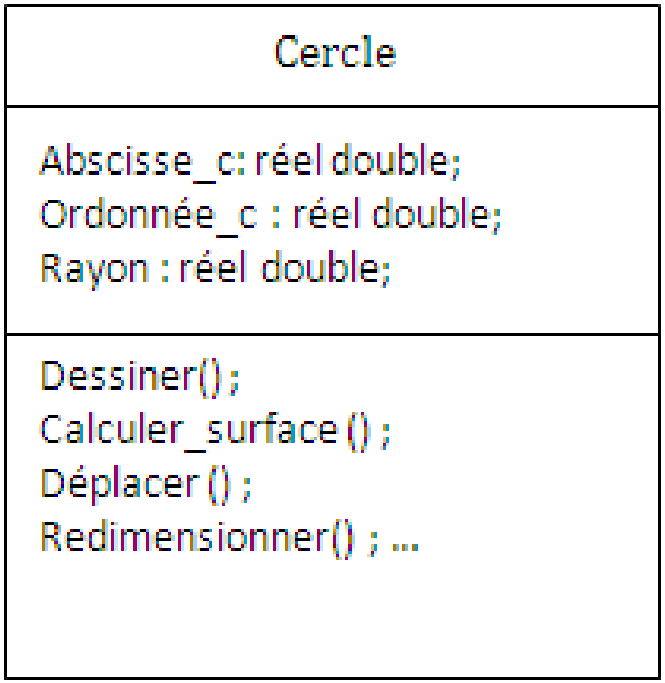
Le concept de classe

Exemple d'une classe Etudiant



Le concept de classe

Exemple d'une classe Cercle



Les propriétés (attributs) d'un cercle sont:

le centre (abscisse, ordonnée)
et le rayon

Les traitements (méthodes) qu'on peut faire sur un cercle sont:

Dessiner
Déplacer
Réduire
Agrandir
...

Le concept de classe

En finalité

Une classe peut être définie par l'équation

Classe = Attributs + Opérations + Instanciation

Attributs: ***propriétés***

Opérations: ***Méthodes***

Instanciation: ***mécanisme qui permet la création des objets de la classe***

Une classe est un type mais pour pouvoir l'utiliser, il faut créer des variables de ce type ⟹ ***créer des objets de la classe***

Le concept d'objet

Un objet **est une entité d'un domaine d'étude** (comme Personne, Employé, Livre, Forme géométrique, Equation, …) possédant **des caractéristiques propres.**

Certaines de ces caractéristiques peuvent changer de valeur dans le temps et provoquent ainsi un **changement d'état de l'objet**.

Un objet est créé à partir d'une classe qui le définit (c'est le type de l'objet)

Un objet est vu comme une ***variable améliorée*** pouvant subir des traitements (opérations)

Le concept d'objet

A la création d'un objet

L'objet a son **propre espace mémoire** comportant les attributs de l'objet et un lien vers toutes les méthodes pouvant s'effectuer sur l'objet
L'objet possède **une identité** (**une référence** attribuée par le système)

Chaque objet est d'un **type précis** (la classe qui le définit)

Dans le jargon de la POO, chaque objet est une *instance* (ou une réalisation) d'une *classe*,

Tous les objets d'un type particulier peuvent subir les mêmes traitements (**appels de méthodes sur les objets**).

Le concept de classe

En finalité

Un objet peut être défini par l'équation

Objet = Identité + Etat + Comportement

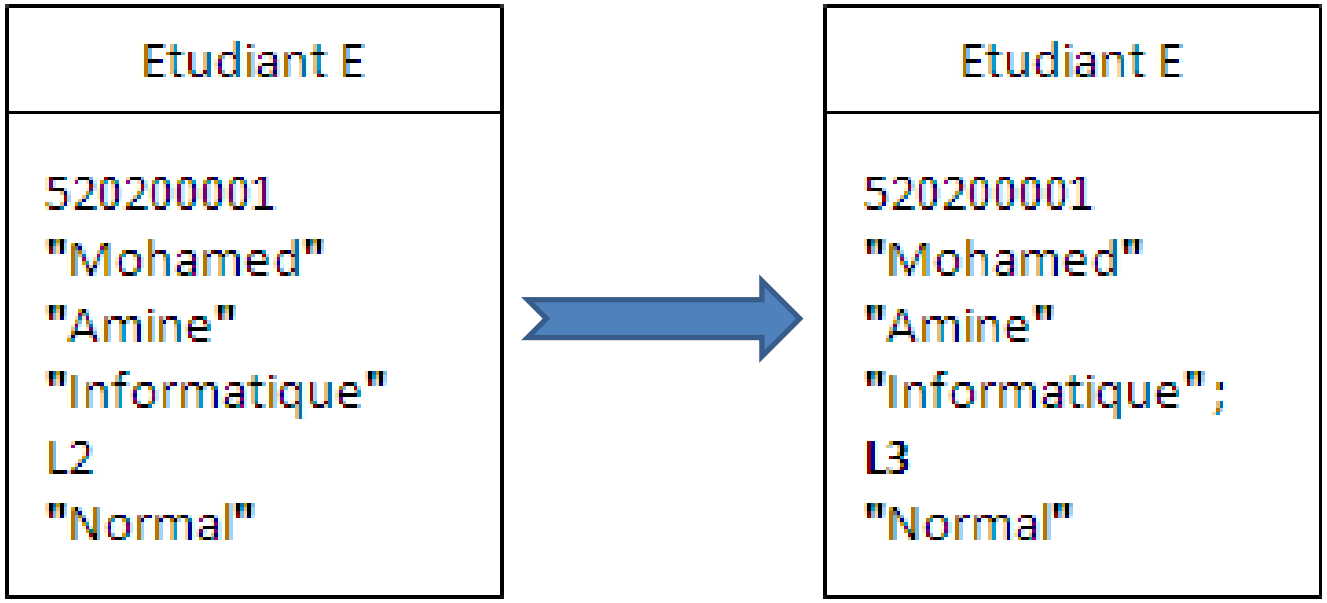
Identité: ***référence***

Etat: ***Valeurs des attributs à un moment donné***

Comportement: ***appel de méthodes sur l'objet, pouvant provoquer un changement d'état de l'objet.***

Le concept de classe

Changement d'état d'un objet



Implémentation d'une classe en java

Syntaxe d'implémentation d'une classe

```
class nom de la classe
 {          type1 att1 ;
            type2 att2 ;...
            typen attn ;
            /* définition de la méthode 1 */
            /* définition de la méthode 2 */
           ...
 }
```

Implémentation d'une classe en java

Exemple

```
class Etudiant
{ /* déclaration des attributs */
 long mat ;
 String nom, prenom ;          /* String est une classe prédéfinie qui implémente
                                  les chaînes de caractères en java */
 String filiere ;
 int sem_etude ;
 String statut ;
 /* définition de la méthode Afficher */
 void Afficher ()
          {System.out.println ("matricule =" + mat) ;
           System.out.println ("nom = " + nom + "& prénom = " + prenom) ;
           System.out.println ("filière = " + filiere) ;
           System.out.println ("semestre = " + sem_etude) ;
           System.out.println ("statut = " + statut) ;
          } // fin de la méthode Afficher

 /* définition de la méthode 2 */
...
}          // fin de définition de la classe Etudiant
```

Implémentation d'une classe en java

Commentaires

String est une classe prédéfinie en java pour la manipulation des chaines de caractères

System.out.print (…) /System.out.println (…)

Est une instruction de java pour l'affichage à l'écran

L'opérateur '+ ' est utilisé pour la concaténation

Par exemple: **System.out.println ("matricule =" + mat);**

Implémentation d'une classe en java

Autre exemple

```
class Point
{ double x, y;  //deux attributs

 // méthodes
 void  affiche()
  { system.out.println ( "les coordonnées du point sont: "
     + " ( " + x + "," + y + " ) " ); }

void déplacer ( double a, double b)
   { x = x+a; y = y+b; }

} // fin de la classe
```

Création d'objets d'une classe

Création d'objets (ou Instanciation)

Un objet **ne peut être utilisé** dans un programme que **s'il est créé.**

L'opération de création d'objet nécessite que la classe qui décrit l'objet soit définie.

En Java, la création d'un objet se fait à l'aide de l'opérateur ***new*** et consiste à :

- ***réserver de l'espace*** pour contenir l'état initial de l'objet
- affecter des valeurs initiales aux propriétés de l'objet (à l'aide d'un ***constructeur***).
- affecter **une *identité* à l'objet** afin de le différencier des autres objets.

Création d'objets d'une classe

Création d'objets (ou Instanciation)

La syntaxe de création d'un objet appelé ***obj*** ***à partir d'une classe*** ***nom_classe*** est la suivante:

nom_classe ***obj*** = ***new*** nom_classe() ;

Exemples

Etudiant **E** = ***new*** Etudiant();
Etudiant **S** = ***new*** Etudiant ();
Point **P1** = ***new*** Point();
Point **P2** = ***new*** Point();

Création d'objets d'une classe

Notion de constructeur

De manière basique, on peut dire que:

Un constructeur est une méthode, d'une classe donnée**, servant à créer des objets* (il s'agit de la méthode invoquée avec l'opérateur ***new).

Un ***constructeur*** ***n'a pas de type de retour***

*Un **constructeur** a nécessairement* ***le même nom que la classe*** *dans laquelle il est défini.*

On peut définir dans un constructeur des instructions d'initialisation des attributs de l'objet.

Création d'objets d'une classe

Notion de constructeur

Si le programmeur ne définit pas un constructeur dans sa classe, java fournit alors un **constructeur par défaut**

Le constructeur par défaut ***a le nom de la classe***

Le constructeur par défaut ***n'a pas de paramètres***

Le constructeur par défaut ***initialise les attributs à des valeurs par défaut (zéro pour les scalaires et null pour les références)***

La définition d'un constructeur explicite annule le constructeur par défaut.

Création d'objets d'une classe

Notion de constructeur

Dans l'écriture

Etudiant E = new Etudiant();

Etudiant() *correspond à l'appel au constructeur par défaut fourni par java*

Point P = new Point();

Point() *correspond à l'appel au constructeur par défaut.*

Définition d'un constructeur

Notion de constructeur

Le programmeur peut définir un constructeur (avec ou sans paramètres) dans sa classe.
Dans ce cas, il faudra appeler le constructeur défini par le programmeur

Plusieurs constructeurs peuvent être définis dans la même classe (avec des signatures différentes) → ***surcharge de constructeurs***

Notion de constructeur

Exemple

```
class Point
{ double x, y;  //deux attributs

 //constructeur défini par le programmeur
 Point (double a, double b)
    { x = a; y =b; }
// méthodes
 void  affiche()
  { system.out.println ( "les coordonnées du point sont: "
    + " ( " + x + "," + y + " ) " ); }
 void déplacer ( double a, double b)
   { x = x+a; y = y+b; }
} // fin de la classe
```

Notion de constructeur

Exemple

Si on veut créer des objets Point en utilisant le constructeur défini dans la classe, on écrira:

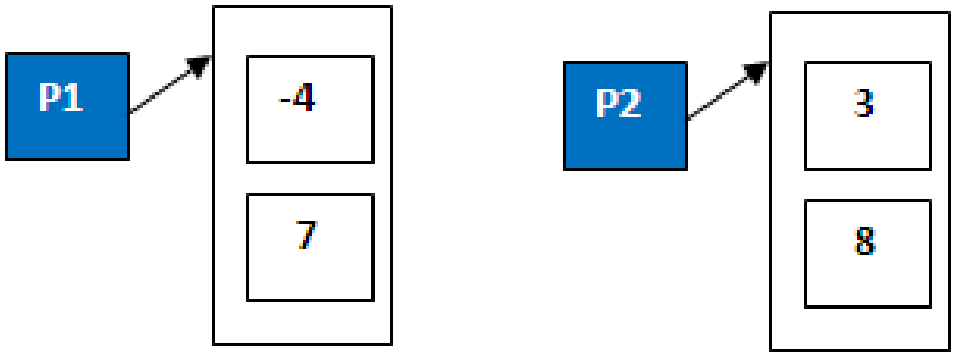
Point P1 = new Point (-4, 7);

Point P2 = new Point (3, 8);

On initialise les objets Point avec les valeurs données en paramètres du constructeur

Notion de constructeur

Schématiquement, on aura



Deux espaces mémoires seront créés pour les deux objets de type Point référencés par P1 et P2, respectivement.

En effet, P1.x = -4 P1.y = 7
P2.x = 3 P2.y =8

Surcharge de constructeurs

La surcharge de constructeurs est le fait de définir plusieurs constructeurs dans la même classe.

Tous les constructeurs auront le même nom → le nom de la classe

Les constructeurs définis dans la même classe auront des signatures différentes, pour lever toute ambiguité

Surcharge de constructeurs

Exemple

```
class Point
{ double x, y;  //deux attributs

 //constructeurs
 Point (double a, double b)
    { x = a; y =b; }

 Point (double a)
   {x= a; y =a/2;}

 Point ()  { }  //constructeur vide (par défaut)
// méthodes
...
}// fin de la classe
```

Surcharge de constructeurs

On a défini, dans la classe Point,

Un constructeur avec deux paramètres

Un constructeur avec 1 seul paramètre

Un constructeur sans paramètres

On pourra écrire:

Point P1 = new Point (3, 9); → **P1 (3, 9)**

Point P2 = new Point (10); → **P2 (10, 5)**

Point P3 = new Point(); → **P3 (0,0)**

Les méthodes

Une méthode est une sorte de « fonction/procédure » dédiée à un traitement spécifique qui pourrait s'appliquer sur n'importe quel objet de la classe.

Une méthode comporte :

-un en-tête (type de retour, nom de la méthode et les paramètres)

- un corps de méthode (ensemble d'instructions)

Les méthodes

Une méthode est une sorte de « fonction/procédure » dédiée à un **traitement spécifique** qui pourrait s'appliquer sur n'importe quel objet de la classe.

Une méthode comporte :

-**un en-tête (type de retour, nom de la méthode et les paramètres)**

- **un corps de méthode (ensemble d'instructions)**

```
Type_résultat Nom_méthode (type1 param1, type2 param2, …)
        { /* instructions */ }
```

Une méthode peut utiliser des variables locales

Les méthodes

Passage de paramètres

Les variables ***de types primitifs*** *sont passées* ***par valeur***, toute modification de la variable dans la méthode n'est pas transmise à l'extérieur.

Les ***tableaux et les objets*** *sont passés* ***par référence*** (adresse), toute modification du tableau ou de l'objet dans la méthode est transmise à l'extérieur.

Les méthodes

Appel de méthodes

Les appels de méthodes sur les objets de la classe, se font ***en utilisant la notation pointée.***

Nom_objet. Nom_méthode **(liste de paramètres d'appel) ;**

Point P1 = new Point (3, 9); **P1.affiche();**

P1.déplacer (3, -2);

Le point P1 aura de nouvelles coordonnées → (6, 7) en appliquant la méthode déplacer définie dans la classe Point (voir exemple diapo suivante)

Les méthodes

Exemple

```
class Point
{ double x, y;  //deux attributs

 //constructeur défini par le programmeur
 Point (double a, double b)
    { x = a; y =b; }
// méthodes
 void  affiche()
  { system.out.println ( "les coordonnées du point sont: "
    + " ( " + x + "," + y + " ) " ); }
 void déplacer ( double a, double b)
   { x = x+a; y = y+b; }
} // fin de la classe
```

Les méthodes

Appel de méthodes

Point P2 = new Point (4, 10); **P2.affiche();**

P2.déplacer (1, 1);

Le point P2 aura de nouvelles coordonnées → (5, 11) en appliquant la méthode déplacer définie dans la classe Point

Surcharge de méthodes

La surcharge de méthode est le fait de définir plusieurs méthodes avec le même nom dans la même classe.

Les méthodes surchargées auront des signatures différentes, pour lever toute ambiguité

Les méthodes

Exemple

```
class Point
{ double x, y;  //deux attributs

 //constructeur défini par le programmeur
 Point (double a, double b)
    { x = a; y =b; }
// méthodes
 void  affiche()
  { system.out.println ( "les coordonnées du point sont: "
    + " ( " + x + "," + y + " ) " ); }
 void déplacer ( double a, double b)
   { x = x+a; y = y+b; }
```

Exemple (suite)

```
void déplacer ( double a)
   { x = x+a;}

void déplacer ()
   { x = x+ 2; y = y+2; }

} //fin de la classe Point
```

La méthode « déplacer » est surchargée dans la classe Point

Les méthodes

Exemple (suite)

```
Point P1 = new Point (3, 9);   P1.affiche();

P1.déplacer (3, -2);  → P1 (6, 7)
P1.déplacer ();       → P1 (8, 9)

Point P2 = new Point (-2, 4);  P2.affiche();
P2.déplacer ();       → P2 (0, 6)
P2.déplacer (5);      → P2 (5, 6)
```

On appelle les différentes méthodes **« déplacer »** sur des objets de la classe Point

Attributs de type « classe d'objets »

Parmi les attributs d'une classe, il est possible d'avoir:

-des attributs de type primitifs (int, float, double,…)

- des attributs de type objet définis par une autre classe

-Aux attributs de type objet, on peut appliquer toutes les méthodes définies dans la classe de ces objets

Attributs de type « classe d'objets »

Exemple

```
class Cercle {
        Point centre ;   // attribut de type Point
        double rayon;

 // constructeur Cercle
Cercle () {
     centre = new Point(1.0, 0.0);
     rayon = 2.0 ; }

      // définition de la méthode déplacer pour le cercle
void déplacer (double  a, double  b) {
         centre.déplacer(a, b);
    // appel de la méthode déplacer de la classe Point }
 }
```

Références d'objets

Au moment de l'exécution d'un programme, lorsqu'une instruction ***new*** est exécutée, un objet en mémoire est créé.
Le nom de l'objet spécifié par le programmeur contient l'adresse mémoire appelée ***référence de l'objet***.
L'objet est alors manipulé à travers sa référence.

En plus des attributs de l'objet, l'espace réservé à l'objet comporte un lien vers les méthodes de la classe.

Références d'objets

Exemple

```
class Point {
      double x,y;
       Point(double x1, double y1)    { x= x1, y = y1 ; }
   }
  class Program_Point
    public static void main (String arg [ ]) {
       Point p = new Point (3.5, 5.0);
       Point q;          // q est juste une référence, pas d'objet créé
       q = p;          // affectation de la réf d'objet p à la référence q
       p. déplacer (1.0, -2.2);  p.affiche(); q.affiche();
// le même objet est affiché deux fois de suite
} }
```

Références d'objets

L'affectation de la valeur de *p* à *q* n'a pas créé un nouvel objet de type *Point* mais simplement une ***référence à l'objet pointé*** par *p*.

C'est le même objet qui est référencé par p et q en même temps.

Pour créer deux objets distincts, il **faut** utiliser l'opérateur ***new*** pour chacun des objets.

L'utilisation des références prend toute son importance lorsque les arguments sont transmis aux méthodes.

La référence « this »

La référence « **this** » est une référence d'objet utilisée pour référencer l'objet à l'intérieur de sa classe.

La référence « this » peut être utilisée pour référencer un attribut d'une classe à l'intérieur de la classe. (Exemple: **this.x** ou **this.y** à l'intérieur de la classe **Point**)

La référence « this » peut être utilisée pour appeler une méthode à l'intérieur de la classe où elle est définie (Exemple: **this.déplacer(2, 3)** à l'intérieur de la classe **Point**)

La référence « this »

Remarque

Cette référence « this » est utilisée notamment lorsqu'on doit écrire une méthode « d'instance » qui manipule deux objets de la classe: un sera référencé par **« this »** et l'autre sera passé en paramètre de la méthode.

A voir plus loin dans les exercices

Vecteur d'objets

Il est possible de regrouper un ensemble d'objets d'une même classe dans un vecteur d'objets

- Il faut créer le vecteur de références vers ces objets
- Créer les objets un par un
- Chaque case du vecteur devra référencer un objet créé

Vecteur d'objets

Exemple: Vecteur d'objets de type Point

```
class TabPoint
 { public  static void  main (String [ ] arg)
      { Point [ ] V = new Point [3];
      /* création d'un vecteur de trois références vers des
objets Point  */
         for (i = 0 ; i< V.length  ; i++)
              V[i] = new Point (i, 2*i);
/* utilisation d'un constructeur avec deux paramètres */
         V[1]. affiche();
   /* on affiche l'objet référencé par V[1]*/
 }    }
```

Compléments

Types primitifs et classes enveloppes

A chaque type primitif (int, float, boolean, …), java fournit une classe enveloppe (Wrapper) qui enveloppe le type.
La classe enveloppe **(Integer, Boolean, Float, Double, Character, Long, …)** du type contient un ensemble de méthodes qui permettent de manipuler le type.

Les classes enveloppes à voir en compléments de cours

Types primitifs et classes enveloppes

Exemple

Il est possible d'obtenir un objet de type **Integer** à partir d'une valeur de type int

Integer obj = new Integer (100);

L'inverse est possible en utilisant la méthode **intValue** de la classe Integer

int a = Integer.**intValue**(obj); → 100

Le ramasse-miettes

Le ramasse-miettes (Garbage Collector) se charge de repérer les objets inutiles (objets créés et non utilisés), et de libérer la mémoire qu'ils utilisent inutilement.

Il opère de façon totalement automatisée. Juste avant de libérer l'espace mémoire d'un objet qui n'est plus référencé, le ramasse-miettes invoque la méthode *finalize* sur cet objet afin de restituer l'espace occupé par l'objet.

La méthode *finalize* de la classe Object (superclasse du langage java) a pour prototype :

protected void finalize() ;