UEF4.3.Programmation Orientée Objet

Mme SADEG

s_sadeg@esi.dz

Ecole nationale Supérieure d'Informatique
(ESI)

Cours précédent

- Présentation du paradigme de programmation orienté objet
- Quelques concepts de base
  - Abstraction
  - Encapsulation
  - Héritage
  - Polymorphisme

Chapitre II

Classes et Objets

La notion de classe

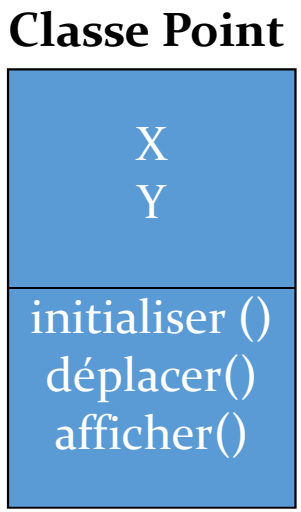
Rappel

- Une *classe* peut être considérée comme un moule ou un modèle à partir duquel on peut créer des objets.
- Un ensemble d'objets ayant les mêmes caractéristiques et les mêmes comportements appartient à la même classe
  - Toutes les voitures appartiennent à la classe voiture
  - 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

Exemple: Classe Point

- 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
  - Afficher les coordonnées de ce point
- Nous devons donc disposer de trois méthodes dans la classe Point
  - initialiser
  - déplacer
  - afficher



La notion de classe

Définition d'une classe Point (code en Java)

Le canevas général de définition d'une classe en Java est

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

le classe

s de la classe Point
Java)

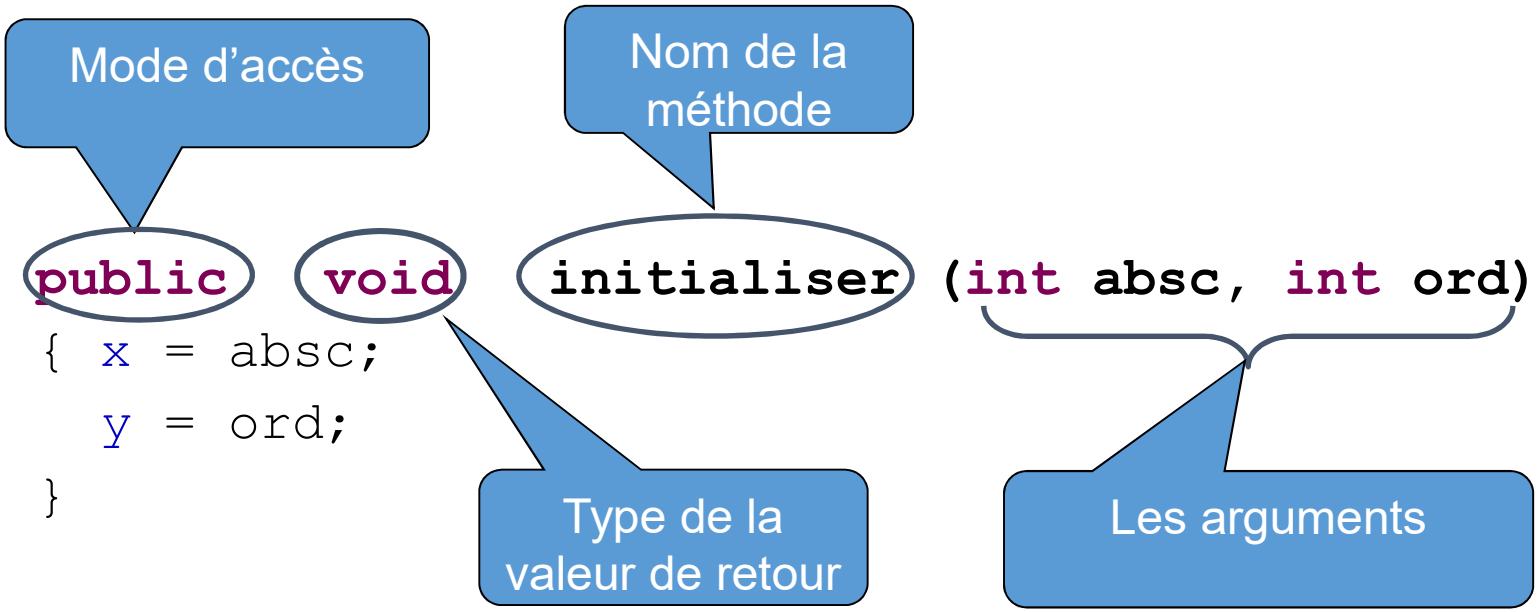
Le mot clé private précise que les champs x et y ne sont pas accessibles de l'extérieur de la classe (c à d en dehors de ses propres méthodes)

```
class Point {
private int x; // abscisse
private int y; // ordonnée
// définition des méthodes
}
```

- Les champs d'une classe peuvent être de type primitif (entier, flottant, caractère ou booléen) ou bien de type objet
- 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

La notion de classe

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



La notion de classe

Exemple classe Point: Le code complet en Java

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

   public void initialiser (int absc, int ord)
   {x = absc;
    y = ord;
   }
   public void deplacer (int dx, int dy)
   {x = x + dx;
    y = y + dy;
   }

   public void afficher()
   {
     System.out.println("je suis un point de coordonnées:"+x+" "+y);
   }
}
```

La notion d'objet

Rappel

- 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)

Notion d'objet

- **Classe = Moule = modèle**
  - Décrit la structure de l'état (les attributs et leurs types)
  - définit le comportement de l'objet (ses méthodes) et son interface (les envois de messages acceptés par l'objet)

- **instance = moulage = objet**
  - Possède un état (valeurs d'attributs) qui correspond à la structure décrite par la classe
  - Ne répond qu'aux envois de messages autorisés par la classe (interface)

Classes et objets

- **classe : abstrait**
  - Il n'y a aucun étudiant qui s'appelle « étudiant »
- **instance : concret**
  - L'étudiant « Halimi amina née le 17 03 1990 »

Création d'un objet

Comment les objets sont-ils créés?

- Pour 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)

Création d'un objet

Exemple: la classe Point

- On peut créer des objets de la classe Point et leur appliquer à volonté les méthodes publiques *initialiser*, *deplacer* et *afficher*
- Avant de créer un objet, il faut d'abord le déclarer.

```
Point  a;
```

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

Création d'un objet

Exemple: classe Point

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

```
a = new Point();
```

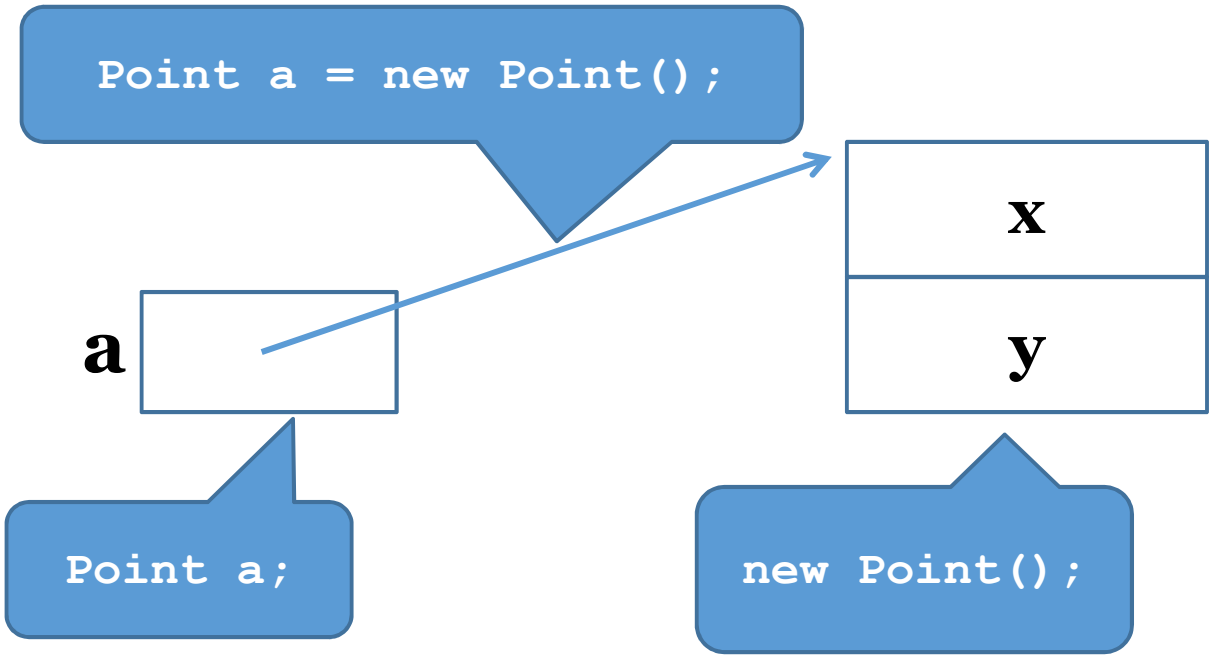
Dans cette instruction, l'opérateur new crée un objet de type point et fournit sa référence. Celle-ci est affectée à la variable *a* déclarée auparavant..

```
Point a;
a = new Point ();
```

```
Point a = new Point();
```

Création d'un objet

Exemple : la classe Point



La notion de constructeur

Classe Point avec constructeur

```
class Point {
private int x;
private int y;
public Point(int absc, int ord)
{ x = absc;
  y = ord;
}
public void deplacer (int dx, int dy)
{x = x + dx;
 y = y + dy;
}
public void afficher()
{
// afficher à l'ecran les valeurs de x et y
}
}
```

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

La notion de constructeur

Définition

- Un constructeur est une méthode spéciale utilisée pour construire un nouvel objet
- 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.
- Le constructeur sert souvent à initialiser les différents attributs de l'objet
- Il a le même nom que celui de la classe
- Ne retourne aucune valeur
- N'a aucun type de retour, même pas `void`

La notion de constructeur

1. Une classe doit-elle disposer d'un constructeur ?

2. Peut on créer un objet d'une classe n'ayant pas de constructeur ?

La notion de constructeur

Classe Point sans constructeur

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

public void deplacer (int dx, int dy)
{x = x + dx;
 y = y + dy;
}
public void afficher()
{
// afficher à l'ecran les valeurs de x et y
}
}
```

Aucun constructeur n'est défini. C'est le constructeur par défaut qui sera invoqué

Création d'un objet

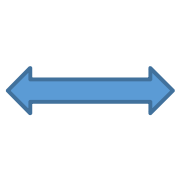
Exemple sans constructeur

En l'absence de constructeur, on peut créer un objet de type Point comme suit:

```
a = new Point();
```

Dans cette instruction, l'opérateur new crée un objet de type point et fournit sa référence. Celle-ci est affectée à la variable *a* déclarée auparavant et les **attributs x et y sont initialisés au valeurs par défaut (0 pour int)**

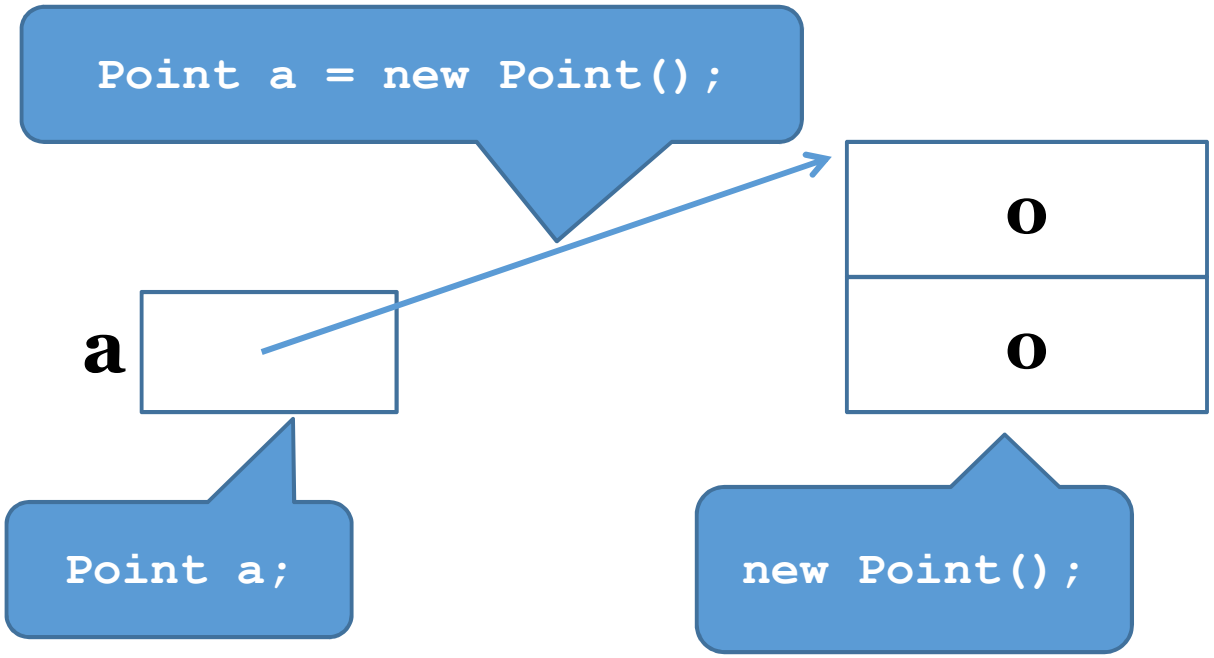
```
Point a;
a = new Point ();
```



```
Point a = new Point();
```

Création d'un objet

Exemple : la classe Point



La notion de constructeur

Attention !

```
class Point {

private int x;
private int y;

public Point(int absc, int ord)
{ x = absc;
  y = ord;
}

public void deplacer (int dx, int dy)
{x = x + dx;
y = y + dy;
}
public void afficher()
{
// afficher à l'ecran les valeurs de x et y
}
}
```

Est il permis d'écrire

```
Point a = new Point ()
```

?

NON !

Que faire ?

La notion de constructeur

Retour à l'exemple: Classe Point

```
class Point {

private int x;
private int y;

public Point()
{
}
public Point(int absc, int ord)
{ x = absc;
  y = ord;
}
public void deplace (int dx, int dy)
{x = x + dx;
y = y + dy;
}
public void affiche()
{
// afficher à l'écran les valeurs de x et y
}
}
```

Point a = new Point();

Correct !

La notion de constructeur

Quelques règles

1. Une classe peut ne disposer d'aucun constructeur (comme dans notre premier exemple de classe Point). Dans ce cas les objets sont créés en utilisant un constructeur par défaut sans arguments comme dans l'instruction `a = new Point ();`

2. Une classe peut disposer de plusieurs constructeurs (qu'est ce que cela vous rappelle t-il ?).

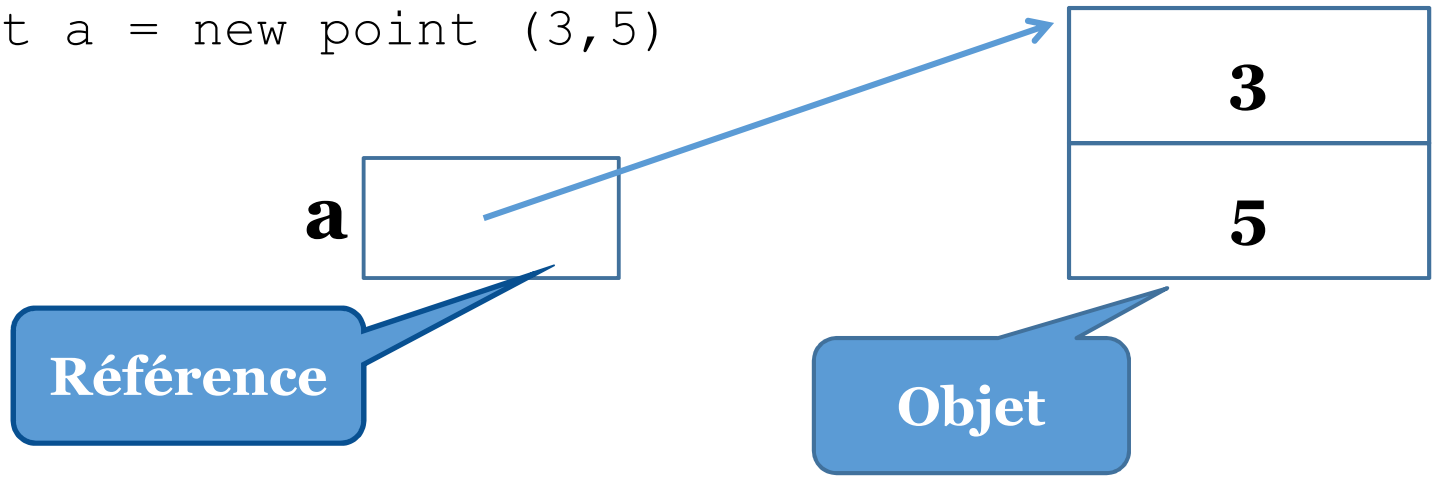
3. Dès qu'une classe possède au moins un constructeur, le constructeur par défaut ne peut plus être utilisé . Sauf si un constructeur sans arguments a été défini.

La notion de référence

- Pour pouvoir utiliser des objets, il faut les référencer
- La référence 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.

Exemple

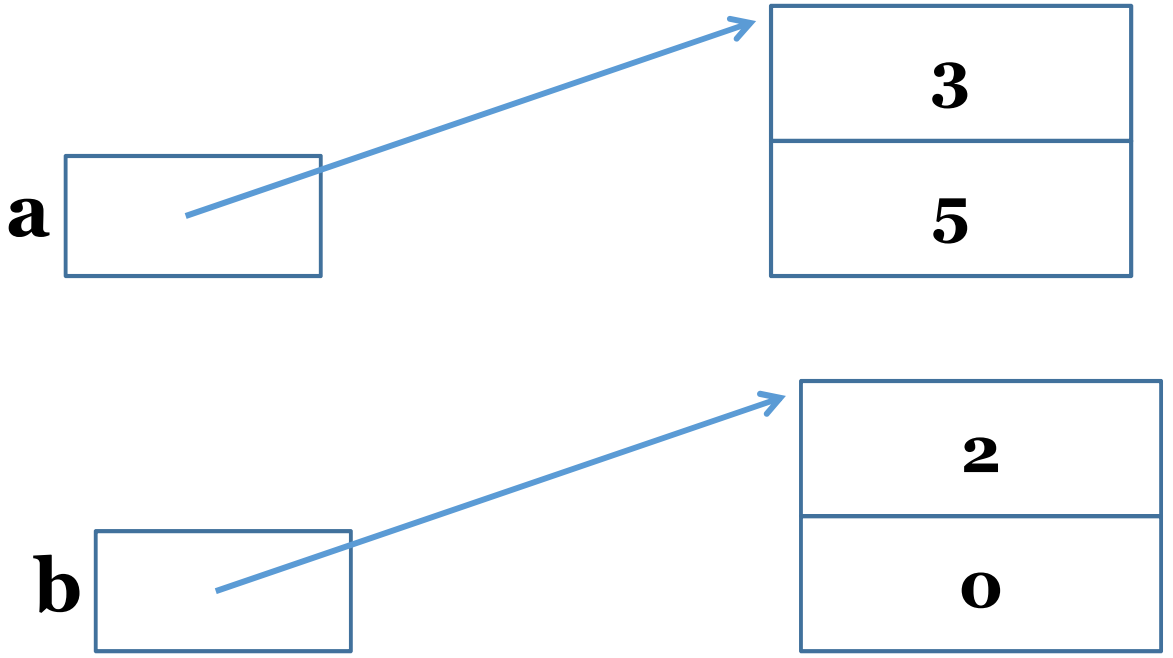
```
Point a = new point (3,5)
```



La notion de référence

Premier exemple

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

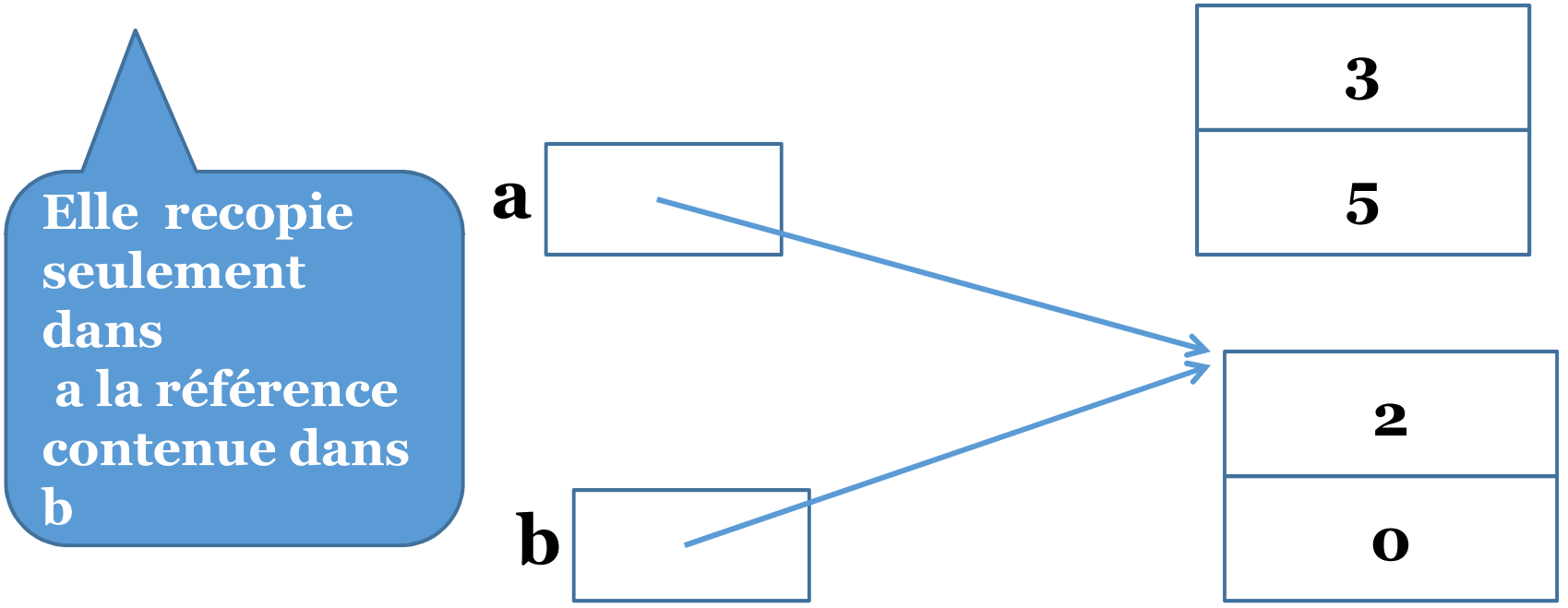


La notion de référence

Premier exemple: Affectation d'objets

Exécutons l'affectation

```
a = b;
```

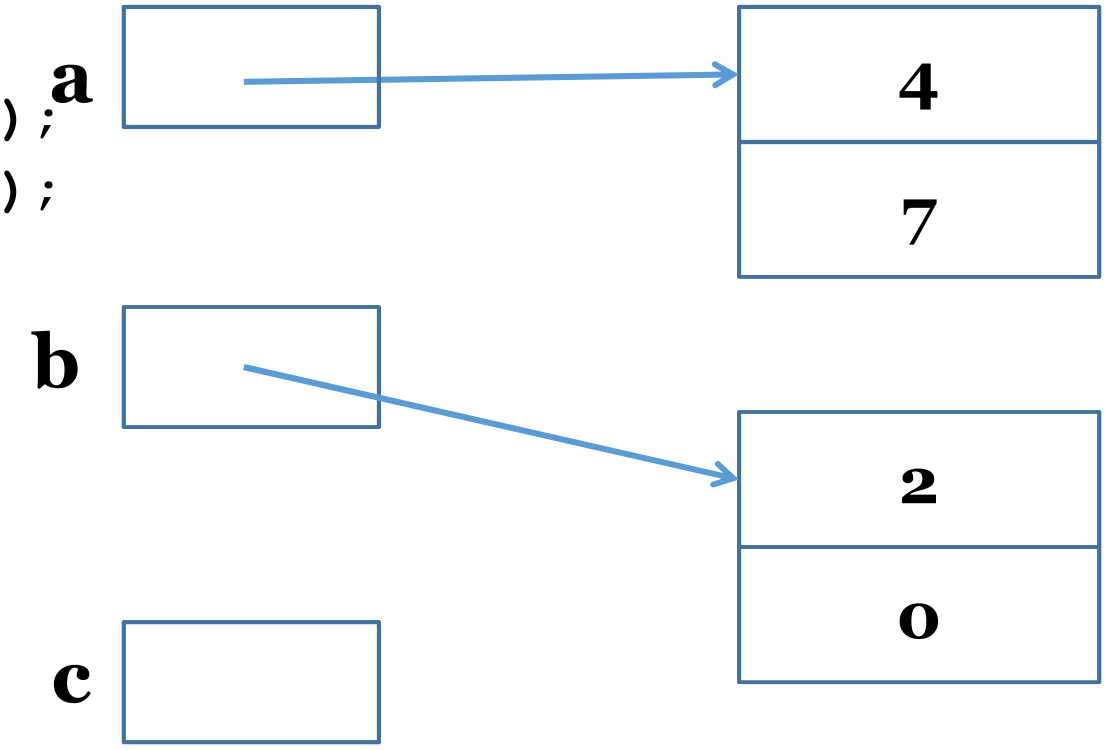


La notion de référence

Deuxième exemple: affectation d'objets

```
Point a, b, c;

a = new Point (4,7);
b = new Point (2,0);
```



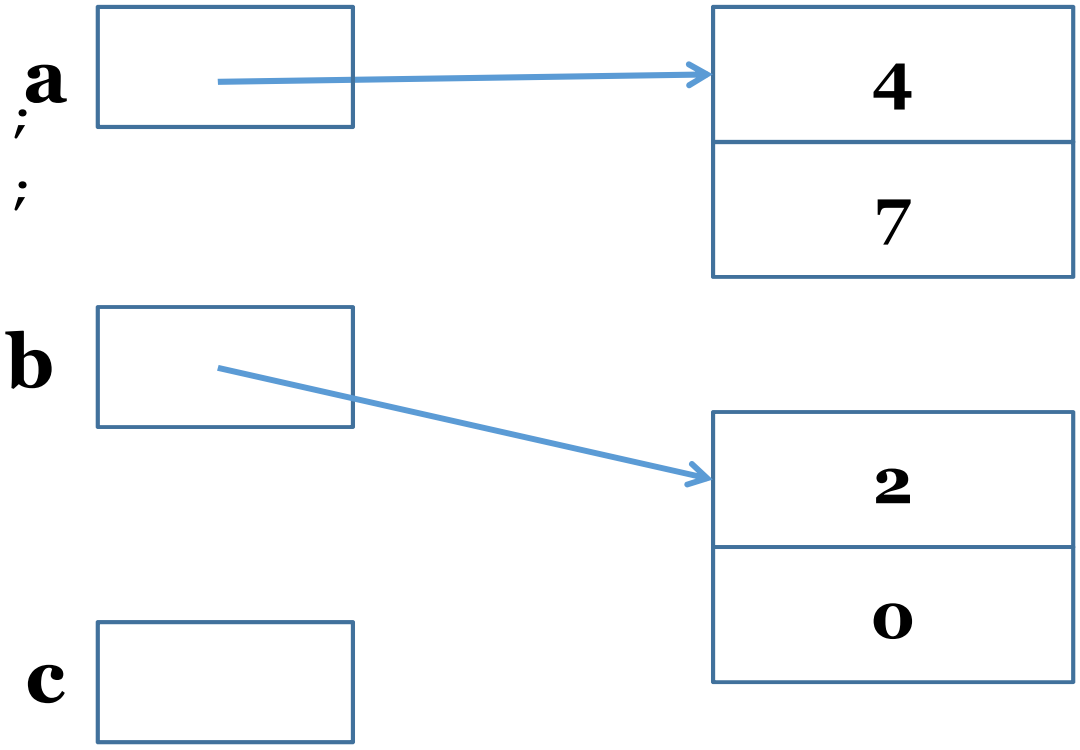
La notion de référence

Deuxième exemple: Comparaison d'objets

```
Point a, b, c;

a = new Point (4,7);
b = new Point (2,0);

c = a;
a = b;
b = c;
```



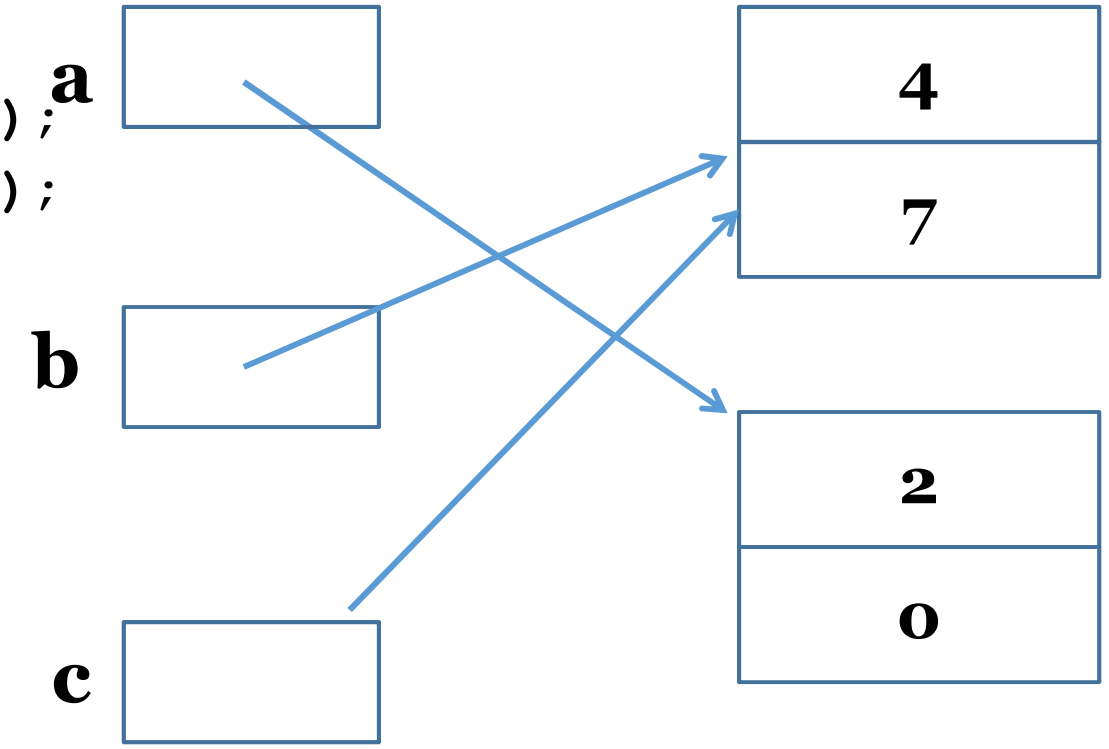
La notion de référence

Deuxième exemple: Comparaison d'objets

```
Point a, b, c;

a = new Point (4,7);
b = new Point (2,0);

c = a;
a = b;
b = c;
```

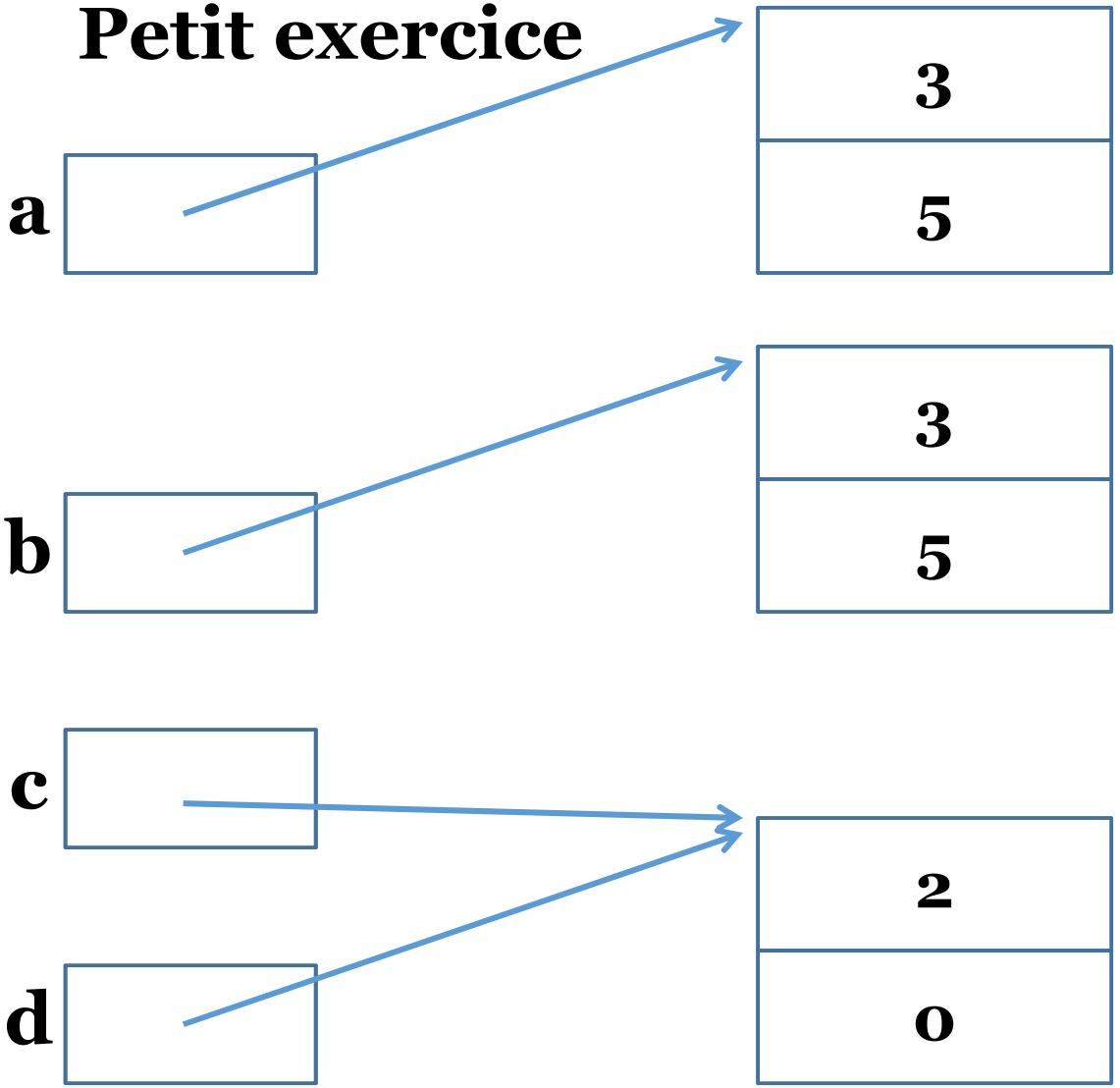


La notion de référence

Petit exercice

a == b ?

c == d ?



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**.

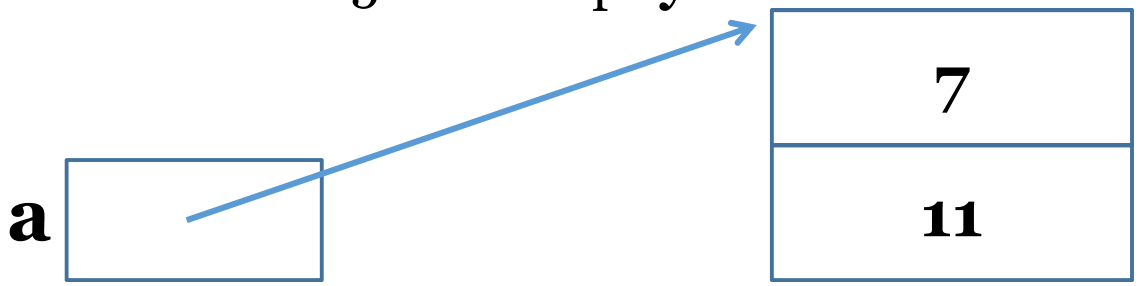
- L'utilisation du mot clé **this** permet aussi d'éviter l'ambiguité

Exemple: le constructeur Point

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

Utilisation d'un objet

- Une fois l'objet créé, on peut lui appliquer n'importe quelle méthode.
- Exemple: `a.deplacer(4,6);`
- C'est un appel à la méthode initialiser qui va affecter la valeur 3 au champs **x** et la valeur 5 au champs **y**



Exercice 7

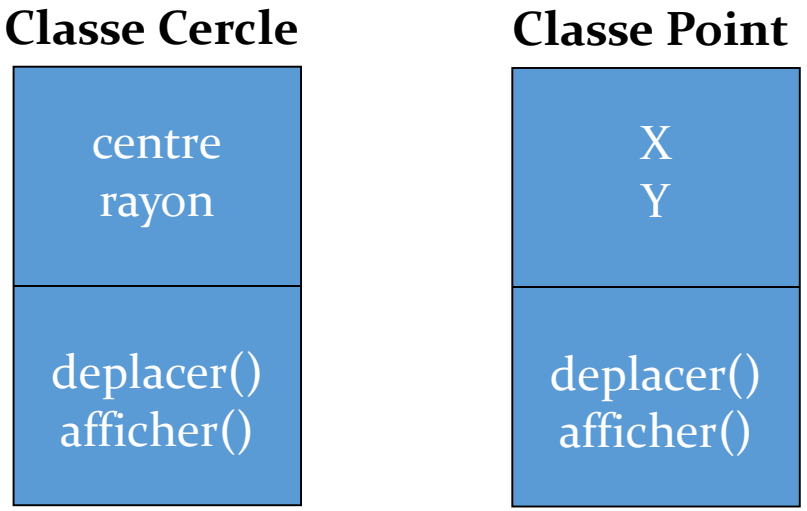
Création et utilisation d'un objet

Exemple : classe cercle

Imaginons que nous souhaitions 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



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

Quelques méthodes particulières

Parmi les méthodes que comporte une classe, on distingue:

- Les constructeurs
- Les méthodes d'accès (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 (Mutator, *Setter*): Modifient les valeurs de certains champs privés.
  - Souvent on utilise des noms de la forme setXXX .

Quelques règles de conception

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 privés.

2. S'appuyer sur la notion de **contrat** qui considère qu'une classe est caractérisée par :
    1. Les entêtes de ses méthodes publiques (interface)
    2. 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.

Le ramasse-miettes (Garbage Collector)

- Pour créer un objet, un programme java utilise l'opérateur new.
- 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 (Garbage Collector 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

Exercice 1 Que fournit le programme suivant ?

```
class Entier{
private int n ;
public Entier (int n) {this. n = n ; }
public void incr (int dn) { n += dn ; }
public void imprime () { System.out.println (n) ; }
}
public class TestEnt{
 public static void main (String args[])
{ Entier n1 = new Entier (2) ; System.out.print ("n1 = "); n1.imprime();
Entier n2 = new Entier (5) ; System.out.print ("n2 = ") ; n2.imprime();
n1.incr(3) ; System.out.print ("n1 = ") ; n1.imprime() ;
System.out.println ("n1 == n2 est " + (n1 == n2)) ;
n1 = n2 ; n2.incr(12) ; System.out.print ("n2 = ") ; n2.imprime() ;
System.out.print ("n1 = ") ; n1.imprime() ;
System.out.println ("n1 == n2 est " + (n1 == n2)) ;
}
}
```

Solution

n1 = 2

n2 = 5

n1 = 5

n1 == n2 est false

n2 = 17

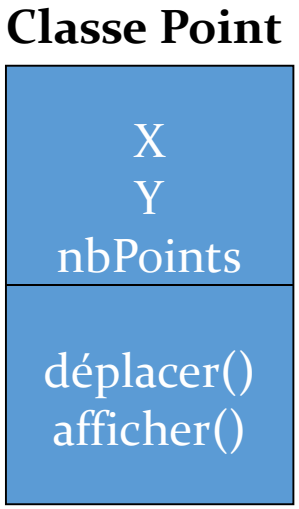
n1 = 17

n1 == n2 est true

Attributs de classe

Exercice: On désire utiliser la classe Point pour créer des points à volonté, mais on veut savoir le nombre de points créés.

Que faut il faire ?



Champs et méthodes de classe

Champs de classes: Solution

```
class Point {
   private int x;
   private int y;
   int nbPoints;

   public Point(int absc, int ord)
   { x = absc;
     y = ord;
     nbPoints ++;
   }
   public void deplacer (int dx, int dy)
   {x = x + dx;
   y = y + dy;
   }
   public void afficher()
   {
   System.out.println("je suis un point de coordonnées: " + x + " " + y);
   }
}
```

Cet attribut sera dupliqué dans chaque objet et aura la valeur 1

Solution incorrecte

Attributs de classe

- À la création d'un objet, On lui crée un exemplaire de chaque attribut.
- On peut définir des attributs qui n'existent qu'en un seul exemplaire pour toutes les instances d'une même classe (donnée globale partagée par tous les objets de la classe).
- Ces attribut sont appelés ***attributs de classe*** ou ***attributs statiques***

Champs et méthodes de classe

Champs de classes: Solution

```
class Point {
   private int x;
   private int y;
   static int nbPoints;

   public Point(int absc, int ord)
   { x = absc;
     y = ord;
     nbPoints ++;
   }
   public void deplacer (int dx, int dy)
   {x = x + dx;
   y = y + dy;
   }
   public void afficher()
   {
   System.out.println("je suis un point de coordonnées: " + x + " " + y);
   }
}
```

méthodes de classe

- De même, on peut définir des *méthodes de classe* (ou *statiques*) qui peuvent être appelée indépendamment de tout objet
- Une méthode de classe a un rôle indépendant d'un quelconque objet. C'est le cas par exemple d'une méthode qui agit uniquement sur les champs de classe
- En java, on utilise là encore, le mot clé `static`.
- Une méthode de classe ne peut en aucun cas agir sur des champs usuels (non statiques)puisqu'elle n'est liée à aucun objet en particulier

Champs et méthodes de classe

Méthodes de classe: retour à l'exemple Point

```
Class Point {
private int x;
private int y;
Static int nbPoints

public Point(int absc, int ord)
{ x = absc;
  y = ord;
  nbPoints ++;
}
public Void deplacer (int dx, int dy)
{  x = x + dx;
   y = y + dy;
}
public static int nbPoints(){
  return nbPoints;
}
}
```

Comment peut on invoquer la méthode nbPoints() ?

int nbp = Point.nbPoints();

Nous n'avons pas besoin de créer un objet pour utiliser cette méthode car c'est une méthode de classe

Exercice 2: Quelles erreurs ont été commises dans ce code ?

```
class A {
static private int p = 20 ;
private int q ;

static int f (int n){ q = n ;}
void g (int n){ q = n ;p = n ;}
}

public class EssaiA{
public static void main (String args[]){
A a = new A() ;
int n = 5 ;
a.g(n) ;
a.f(n) ;
f(n) ;
}
}
```

Solution

- ***l'affectation q=n est* incorrecte, c**ar La méthode statique *f de A ne peut pas agir sur un champ non statique ;*
- Dans la méthode *main, l'appel a.f(n) se réfère à un objet, ce qui est inutile mais toléré. Il serait* cependant préférable de l'écrire *A.f(n).*
- ***l'appel f(n) est incorrect*** *puisqu'il n'existe* pas de méthode *f dans la classe EssaiA. Il est probable que l'on a voulu écrire A.f(n)***.**

Surdéfinition des méthodes

- On parle de surdéfinition (ou surcharge) lorsqu'un symbole possède plusieurs significations entre lesquelles on choisit en fonction du contexte.

- En java, il est possible de surdéfinir les méthodes d'une classe y compris celles qui sont statiques. 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,

Surdéfinition des méthodes

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

   public Point(int absc, int ord)
   { x = absc;  y = ord;
   }
   public void deplacer (int dx, int dy)
   {x = x + dx;
   y = y + dy;
   }

   public void deplacer (int dx)
   {  x + = dx;
   }

   public void deplacer (short dx)
   {x+ = dx;
   }
}
```

Point a = new Point (1,2);
int n = 2; int b = 5; short p = 3;

a.deplacer (n,b); //appelle deplacer (int,int)

a.deplacer (n); //appelle deplacer (int)

a.deplacer (p); // appelle deplacer (short)

Surdéfinition des méthodes

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 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)

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
- S'il y a ambigüité ⟹ Erreur de compilation

Surdéfinition des méthodes

Exemple

```
public void deplace (float dx, int dy)// deplace (float, int)
{…
}
public void deplace (int dx, float dy)
{…
}
```

Quels sont les appels corrects ?

```
Point a = new Point (0,3);
float f; int n;
a.Deplace (f,n)//
a.Deplace (n,f)//
a.Deplace (f,f)//
a.Deplace (n,n)//
```

OK: Appel de deplace (float, int)
Ok: Appel de deplace (int, float)
ERREUR: Pas de méthode deplace (float,float)
ERREUR: Ambigüité

Exercice 3 Quelles erreurs figurent dans ce code ?

```
class Surdef {
public void f (int n) { ..... }
public int f (int p) { ..... }
public void g (float x) { ..... }
public void g (double y) { ..... }
}
```

Solution

- Les deux méthodes *f ont des arguments de même type (la valeur de retour n'intervenant pas* dans la surdéfinition des fonctions). Il y a donc une ambiguïté qui sera détectée dès la compilation de la classe, indépendamment d'une quelconque utilisation.

- La surdéfinition des méthodes *g ne présente pas d'anomalie, leurs arguments étant de types* différents.

Exercice 4

Soit la définition de classe suivante :

```
class A
{  public void f (int n) { ..... }
   public void f (int n, int q) { ..... }
   public void f (int n, double y) { ..... }
}
```

Avec ces déclarations :

A a ; byte b ; short p ; int n ; long q ; float x ; double y ;

Quelles sont les instructions correctes et, dans ce cas, quelles sont les méthodes appelées et les éventuelles conversions mises en jeu ?

a.f(n);
a.f(n, q) ;
a.f(q) ;
a.f(p, n) ;
a.f(b, x) ;
a.f(q, x) ;

Solution

```
a.f(n);       // appel f(int)
a.f(n, q) ; // appel f(int, double) après conversion de q en
                  double
a.f(q) ;      // erreur : aucune méthode acceptable
a.f(p, n) ; // appel f(int, int) après conversion de p en int
a.f(b, x) ; // appel f(int, double) après conversion de b en
              //int et de x en double
a.f(q, x) ; // erreur : aucune méthode acceptable
```

Echange d'informations avec les méthodes

- 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 ?

Echange d'informations avec les méthodes

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.

Cela sera développé en TD à travers des exercices.

- Exercice 5

Exercice 5 Quels résultats fournit ce programme ?

```
class A {
private int n ;
public A (int nn) {n = nn ; }
public int getn () { return n ; }
public void setn (int nn) { n = nn ; }
}
class Util {
public static void incre (A a, int p) { a.setn(a.getn()+p);}
public static void incre (int n, int p) { n += p ;}
}

public class Trans {
public static void main (String args[]) {
A a = new A(2) ;
int n = 2 ;
System.out.println ("valeur de a avant : " + a.getn()) ;
Util.incre (a, 5) ;
System.out.println ("valeur de a apres : " + a.getn()) ;
System.out.println ("valeur de n avant : " + n) ;
Util.incre (n, 5) ;
System.out.println ("valeur de n apres : " + n) ;
}
}
```

Solution

En Java, le transfert des arguments à une méthode se fait toujours par valeur. Mais la valeur d'une variable de type objet est sa référence. D'où les résultats :

valeur de a avant : 2
valeur de a après : 7
valeur de n avant : 2
valeur de n après : 2