Chapitre II:

Classe et Objet

Centre Universitaire de Mila
2ième Année Licence Informatique
Matière: Programmation Orientée Objet
Responsable de la matière: Dr. Sadek Benhammada

1. Syntaxe de déclaration d’une classe

1. Syntaxe de déclaration d'une classe

en-tête

Corps

```
[Modificateurs] class NomDeClasse [extends classe_mere] [implements interfaces]

{
  //Attributs
     [Modificateurs] type nomAttribut_1;
     [Modificateurs] type nomAttribut_2;
       ...
  //Méthodes
     [Modificateurs] typeDeRetour nomDeMethode_1 (params)
    {
       // corps de la méthode;
    }
     [Modificateurs] typeDeRetour nomDeMethode_2 (params)
       {
      // corps de la méthode;
       }
      ...
}
```

1. Syntaxe de déclaration d'une classe

- Une classe se compose de deux parties : un (1) **en-tête** et un (2) **corps**.

1.1. L'en-tête:

- Les **modificateurs** de classe (optionnel) sont : **abstract**, **final**, et la visibilité (**private**, **public**);
- Le mot clé **class** suivie du nom de la classe (obligatoire)**;**
- Le mot-clé **extends** suivie du **nom de la superclasse** (optionnel);
- Le mot-clé **implements** suivie de la liste des noms desinterfaces (optionnel);

Exemples:

```
public class Forme {…}

public class Rectangle extends Forme{…}
```

1.2. Le corps: entouré des accolades ouvrante et fermante (**{…}**), il contient les déclarations des **attributs** et **méthodes:**

1. Syntaxe de déclaration d'une classe

1.3. Déclaration d'un attribut (dans l'ordre) :

- **Modificateurs** (optionnels): ***static***, ***final***, et la visibilité (***private***, ***protected***, ***public***);
- **Type**: Le type est soit:
  - un type de base du langage, (***boolean***, ***byte***, ***short***, ***int***, ***long***, ***float***, ***double***, ***char***, ***void***),
  - ou le nom d'une autre classe du programme.
- **Nom**: nom de l'attribut

Exemples: déclaration d'attribut

```
private int x;
public static final PI=3.14;
```

1. Syntaxe de déclaration d'une classe

1.4. Déclaration d'une méthode : La déclaration d'une méthode est composée de la **signature** et du **corps :**

- **La signature**:
  - **Modificateurs** (optionnels): ***abstract***, ***static***, ***final***, et la visibilité (***private***, ***protected***, ***public***);
  - **Le type de retour** de la méthode;
  - **Nom de la méthode;**
  - Et **Les paramètres de la méthode;**
- **Le corps:** suite d'instructions placées entre { }.

Exemple: déclaration d'une méthode

```
public double somme(double x, double y)  {
      double s=x+y;
      return s;
   }
```

1. Syntaxe de déclaration d'une classe

1.5. Les types de base

Type	longueur	valeurs	Exemple
byte	1 octet	Les entiers entre -128 et +127	byte temperature; temperature = 64;
short	2 octets	Les entiers entre -32768 et +32767	short vitesseMax; vitesseMax = 32000;
int	4 octets	Les entiers entre -2147483648 et 2147483647	int temperatureSoleil; temperatureSoleil = 15600000;
long	8 octets	Les entiers entre - 9223372036854775808 et 9223372036854775807	long anneeLumiere; anneeLumiere=9460700000000000;
float	4 octets	Nombres avec virgule flottante entre 1.401e-045 et 3.40282e+038	float pi; pi = 3.141592653f ;
double	8 octets	Nombres avec virgule flottante entre 2.22507e-308 et 1.79769e+308	double division; division = 0.3333333333334;
char	2 octets	caractère (65000 caractères possibles)	char caractere; caractere = 'A'
boolean	1 bit	valeur logique : true ou false	boolean question; question = true

1.6. Chaines de caractères

- Les chaînes de caractères en java ne correspondent pas à un type de données mais à une classe **`String`**.
- Une chaîne peut donc être déclarée de la façon suivante :

```
String phrase = "Hellow world";
```

- `phrase` **n'est pas une variable** mais un **objet** de la **classe `String`**.
- Java admet l'opérateur + comme opérateur de concaténation de chaines de caractères.
- L'opérateur + permet de concaténer plusieurs chaînes de caractères.

Exemples : Déclaration et concaténation de chaines de caractères

```
String  s1="Hello ";
String  s2="World";
String  s3=s1+s2;// s3==Hellow world
```

1. Syntaxe de déclaration d'une classe

- **Exemple:** Déclaration d'une classe **Point**

```
public class Point {
    // attributs
    private double x; // Abscisse
    private double y; //Ordonnée
    // methodes
    public String toString(){
    return "Point("+x+","+y+")";
    }
}
```

2. Conventions de notation

2. Conventions de notation

1. **Utilisation des noms significatifs** pour les classes, les attributs, les méthodes et les variables. Le nom doit être suffisant pour comprendre ce que fait une méthode par exemple, sans aller voir le détail du code.
2. Les **noms de classes** commencent par une **Majuscule**,

   Exemples:

   ```
   public class Rectangle {…}
   public class Personne{…}
   ```

3. Les noms des **attributs**, des **méthodes** et des **variables** commencent **minuscule**.

   Exemples:

   ```
   private double longueur;  //attribut
   public double surface() {…} //méthode
   ```

2. Conventions de notation

4. Lorsqu'un nom est constitué de plusieurs mots accolés, chacun des noms successifs commence par une majuscule.

   Exemples:

```
public class CompteEnBanque{…} //Classe
public int nombreRoues; //attribut
public double calculerSurface(){…}; //méthode
```

5. Le nom d'une **constante** est en **MAJUSCULE**. Lorsque le nom d'une constante est constitué de plusieurs mots avec les mots séparés par le caractère souligne

   Exemples:

```
public static final double PI=3.14;
public static final int NOMBRE_MAX=100;
```

6. En général, le premier mot d'un nom de méthode est un verbe.

   Exemple:

```
public double calculerSurface()
```

7. Il est fréquent que tous les noms soient en anglais.

3. Déclaration et création d’un objet

3. Déclaration et création d'un objet

3.1. Déclaration d'un objet

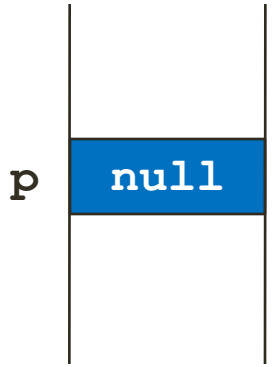
- La déclaration d'un objet est de la forme :

```
NomClasse nomObjet;
```

Exemple: Déclaration d'un objet de la classe Point

```
Point p;
```

- La déclaration de l'objet (**`Point p`**) réserve un emplacement mémoire pour une ***référence*** sur un objet de type Point.



- À ce stade, la valeur de la variable **`p`** est ***`null`***

3. Déclaration et création d'un objet

3.2. Création d'un objet

- Pour que **p** référence effectivement un objet, il faut appeler un **constructeur**.
- Le constructeur est la méthode employée pour créer un objet (*allouer un espace mémoire à l'objet*) d'une classe donnée et éventuellement *initialiser* ses attributs.
- Le constructeur **porte le nom de la classe** et **ne mentionne pas de type de retour.**

Exemple: Constructeur de la classe **Point**

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

  //CONSTRUCTEUR
   public Point(double a,double b)
     {
    x=a;
    y=b;
     }
}
```

3. Déclaration et création d'un objet

3.2. Création d'un objet

- Pour créer un objet on invoque un constructeur à l'aide de l'opérateur **new** qui effectue la réservation de la mémoire et renvoie l'adresse de la zone allouée.

Exemple

```
Point p;//Déclaration de l'objet p
p=new Point(5,3);// Création de l'objet p
```

- Il est possible de réunir la déclaration et la création d'un objet.

Exemple

```
Point p = new Point(5,3);
```

3. Déclaration et création d'un objet

3.2. Création d'un objet

- Il est possible de déclarer plusieurs constructeur pour une même classe (**surcharger le constructeur**).

Exemple:

On peut déclarer un autre constructeur pour la classe *Point*, pour créer les objets dont les valeurs des attributs *x* et y sont égaux.

```
public Point(double a)
{
    x=a;
    y=a;
}
```

3. Déclaration et création d'un objet

3.2. Création d'un objet

- **Le Constructeur par défaut:** Si l'on n'écrit aucun constructeur pour une classe donnée, il est possible d'utiliser le constructeur par défaut qui se contente d'allouer un emplacement mémoire(**il n'initialise pas les attributs**).

- Pour une la classe `Point`, le constructeur par défaut correspond au code suivant :

```
public Point(){}
```

Exemple

Si la classe *Point* n'a pas de constructeurs, on peut écrire :

```
Point p = new Point();
```

Remarque :

- Les attributs d'une classe, s'ils ne sont pas initialisés, se voient affecter automatiquement une valeur par défaut.
- Cette valeur vaut :
  - **0** pour les attributs numériques (**int**, **float**, **double**, etc.),
  - **false** pour les booléens, et
  - **null** pour les objets (Exemple: les attributs de type **String**).

3. Déclaration et création d'un objet

3.3. Le mot clé `this`

- **Le** mot-clé **`this`** sert à référencer dans une méthode l'objet en cours d'utilisation.

- **Exemple:**

```
//Constructeur de la classe Point

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

- L'instruction **`this.`**`x=a;` signifie que l'attribut `x` de l'objet en cours (**`this`**) reçoit la valeur `a`.

3. Déclaration et création d'un objet

3.3. Le mot clé `this`

- Lorsqu'une méthode d'un objet référence un attribut **`x`** de cet objet, l'écriture **`this.x`** est implicite.
- On doit utiliser le mot-clé **`this`** explicitement lorsqu'il y a **conflit d'identificateurs**.

- **Exemple:**

  On doit utiliser le mot-clé **`this`** explicitement, lorsque les même identificateurs sont utilisés pour les attributs et pour les paramètres du constructeur.

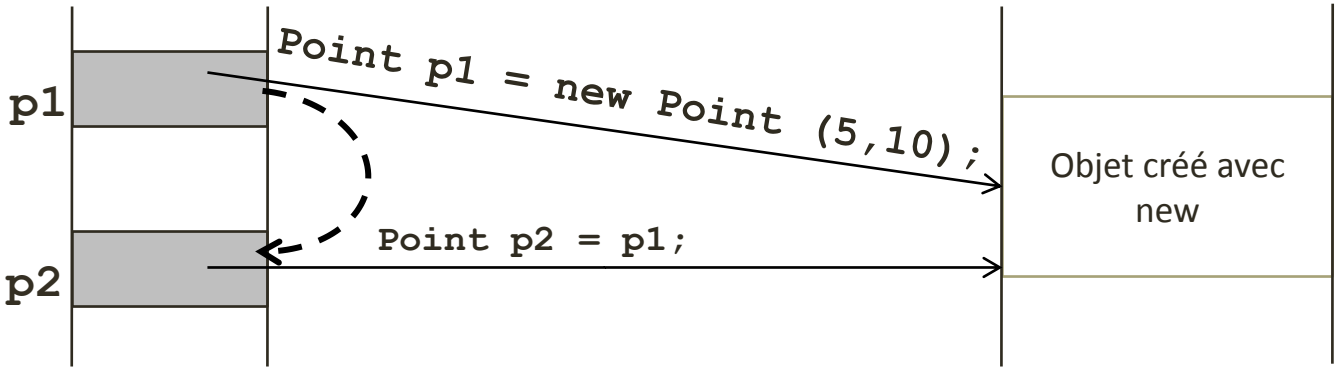
```
//Constructeur de la classe Point

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

3. Déclaration et création d'un objet

3.4. Création d'objets identiques

- On peut avoir besoin de créer deux objets absolument identiques.
- Examinons le code suivant :

```
Point p1 = new Point();
Point p2 = p1;
```



- **p1** et **p2** contiennent la même référence ⇒ **p1** et **p2 pointent le même objet**.
- La modifications des valeurs des attributs de **p1**, modifie aussi les valeurs des attributs de **p2** puisque, en fait, c'est le même objet.

3. Déclaration et création d'un objet

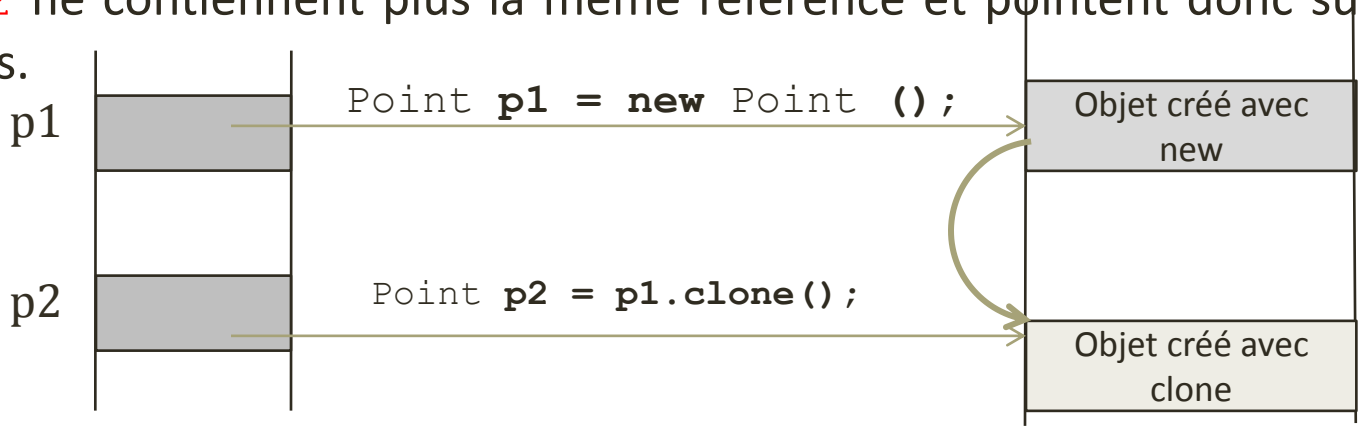
3.4. Création d'objets identiques

1ière solution: La méthode `colone()`

- Pour créer une copie d'un objet, il est possible d'utiliserla méthode `clone();`
- La méthode `clone()` est héritée de la classe `Object` qui est la classe mère de toutes les classes en Java;
- La méthode `clone()` permet de créer un deuxième objet indépendant mais identique à l'objet appelant.
- **Exemple :**

```
Point p1 = new Point();
Point p2 = p1.clone();
```

`p1` et `p2` ne contiennent plus la même référence et pointent donc sur des objets différents.



3. Déclaration et création d'un objet

3.4. Création d'objets identiques

2ième solution: Constructeur de copies

- Une autre solution pour créer des objets identiques consiste à définir un constructeur de copie;
- **Exemple :** Constructeur de copies de la classe `Point`

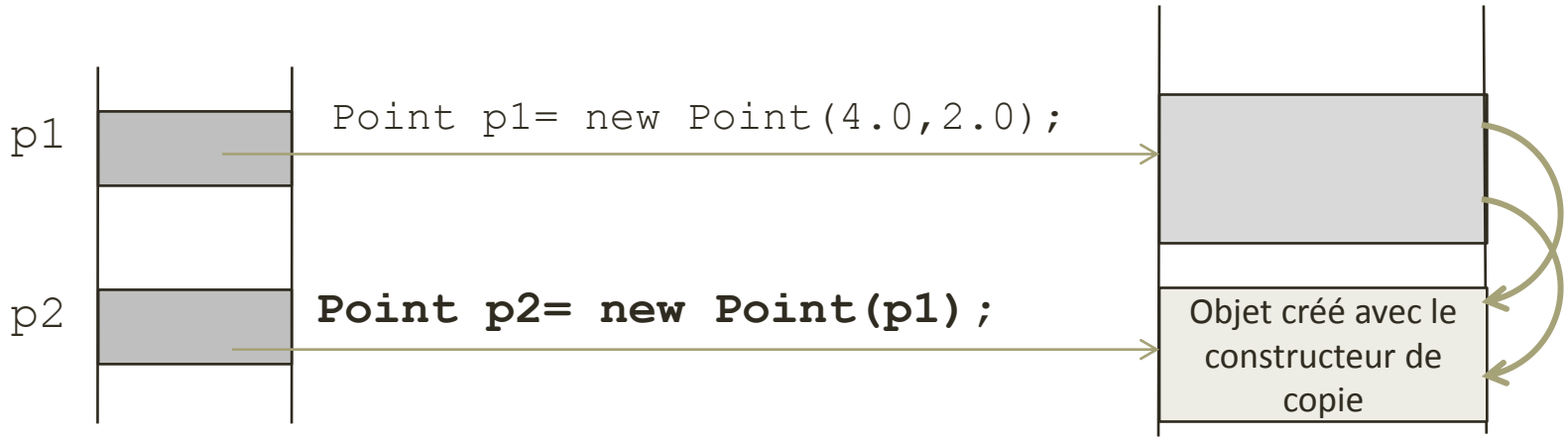
```
public class Point {
  private double x;
  private double y;
     //Constructeur
  public Point(double x, double y)
  {
       this.x=x;
       this.y=y;
  }
   //CONSTRUCTEUR DE COPIES
   public Point (Point p)
   {
      this.x = p.x;
      this.y= p.y;
   }
}
```

3.4. Création d'objets identiques

2ième solution: Constructeur de copies

- **Exemple :** Création d'un objet de la classe `Point` en utilisant le constructeur de copies

```
Point p1= new Point(4.0,2.0);
Point p2= new Point(p1);
```



3. Déclaration et création d'un objet

3.5. Suppression d'objets

- Les objets ne sont pas des éléments statiques et leur durée de vie ne correspond pas forcément à la durée d'exécution du programme.
- La durée de vie d'un objet passe par trois étapes :

1. La déclaration et la création de l'objet.
2. L'utilisation de l'objet en appelant ces méthodes.
3. La suppression de l'objet : elle est automatique en Java grâce au récupérateur de mémoire (`Garbage Collector:` **GC**).

- Le *GC* permet de supprimer automatiquement les objets qui ne sont plus référencés par le programme. En C++, c'est le programmeur qui s'occupe de la suppression des objets inutiles.

3. Déclaration et création d'un objet

3.5. Suppression d'objets

- Les objets ne sont pas des éléments statiques et leur durée de vie ne correspond pas forcément à la durée d'exécution du programme.
- La durée de vie d'un objet passe par trois étapes :

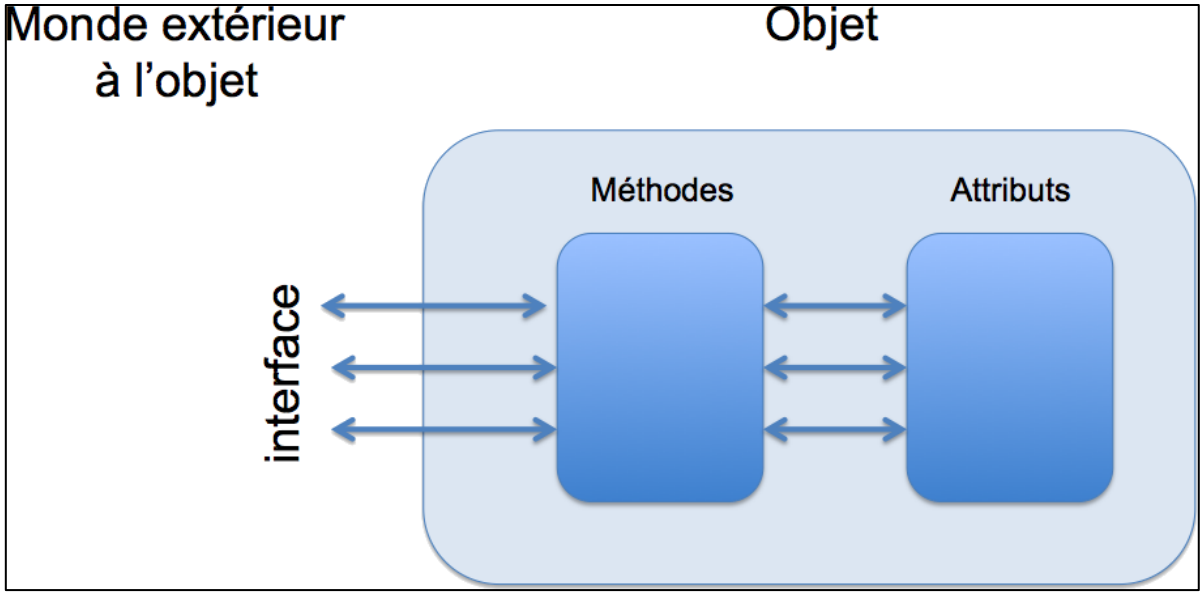
1. La déclaration et la création de l'objet.
2. L'utilisation de l'objet en appelant ces méthodes.
3. La suppression de l'objet : elle est automatique en Java grâce au récupérateur de mémoire (`Garbage Collector:` **GC**).

- Le *GC* permet de supprimer automatiquement les objets qui ne sont plus référencés par le programme. En C++, c'est le programmeur qui s'occupe de la suppression des objets inutiles.

4. L’encapsulation

4. L'encapsulation

- L'encapsulation est la possibilité de masquer une parties des membres d'un objet (attributs et méthodes). c'est-à dire en empêchant l'accès direct à ces membres depuis l'extérieur.
- L'encapsulation permet de ne montrer de l'objet que ce qui est nécessaire à son utilisation.
- La liste des méthodes et des attributs utilisables depuis l'extérieur est appelée l'**interface de la classe.**



4. L'encapsulation

3.1. Contrôle d'accès aux attributs et méthodes

- Pour réaliser l'encapsulation, on dispose d'un ensemble de modificateurs de *contrôle d'accès* aux *classes*, *méthodes* et *attributs*.

- Pour les *méthodes* et *attributs* au sein des classes, le programmeur en Java dispose de 3 niveaux de contrôle d'accès, qu'il fixe à l'aide de 3 modificateurs de visibilité.
  - *public* : les éléments publics sont accessibles sans aucune restriction.
  - *protected* : les éléments protégés ne sont accessibles que depuis la classe et les sous-classes qui en héritent.
  - *private* : les éléments privés ne sont accessibles que depuis la classe elle-même.

- La visibilité par défaut, quand rien n'est spécifié est équivalente à ***public***.

4. L'encapsulation

4.1. Contrôle d'accès aux attributs et méthodes

- **En général:**
  - **Les attributs** d'une classe sont déclarées ***privés*** (**`private`**) ou **protégés** (**`protected`**), ce qui signifie que **seuls des objets de la classe ou de ses sous-classes peuvent les lire et les modifier;**
  - **Les méthodes** sont déclarées ***publiques*** **(`public`),** ce qui signifie que n'importe qu'elle objet peut les appeler;

4. L'encapsulation

4.2. Les accesseurs de lecture et de modification

- Pour lire et modifier les attributs d'un objet, on ajoute à la classe des *méthodes* spécialement conçues à cet effet, que l'on appelle «*Accesseurs*».
- **Accesseurs de lecture**
  - Les **accesseurs de lecture** sont des méthodes qui permettent de **lire** les attributs des l'objets;
  - Les noms des **accesseurs de lecture** commencent généralement par *get* suivi par le nom de l'attribut;

  Exemple: `public String` **`getNom`**`() {return nom}`

- **Accesseurs de modification**
  - **Les accesseurs de modification** sont des méthodes qui permettent de de **modifier** les attributs des l'objets ;
  - Les noms des accesseurs de modification (modificateurs) commencent généralement par *set* suivi par le nom de l'attribut.

  Exemple: `public void` **`setNom`**`(String n) {nom=n;}`

4. L'encapsulation

Exemple

```
public class Point
{
  //Attributs
    private  double x;
    private double y;
  //Accesseurs de lecture
    public double getX(){return x;}
    public double getY(){return y;}
  //Accesseurs de modification
    public void setX(double x){this.x=x;}
    public void setY(double y){this.y=y;}
  //La méthode toString
    public String toString(){
       return "Point("+ x +","+ y +")";
    }
}
```

4. L'encapsulation

Exemple (suite)

Le résultat affiché:

5.0
10.0

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

   Point p= new Point(); /*Création d'un objet Point par utilisation
   du constructeur par défaut */

   p.setX(5.0); /* Utilisation de l'accesseur de modification setX()
   pour initialiser l'attribut x */

   p.setY (10.0); /* utilisation de l'accesseur de modification setY()
   pour initialiser l'attribut y */

   System.out.println(p.getX());/* utilisation de l'accesseur de
   lecture getX() pour afficher la valeur de l'attribut x */

   System.out.println(p.getY ());// utilisation de l'accesseur de
   lecture getY() pour afficher la valeur de l'attribut y
  }
}
```

4. L’encapsulation

4.3. Intérêt des accesseurs

- L’intérêt des accesseurs est de rendre indépendant tout le reste du code de la représentation de l’objet:

- Si on décide de modifier un attribut, il suffit de modifier le code de l’accesseur lui-même, c’est-à-dire une seule ligne du programme, alors qu’il aurait fallu modifier toutes les lignes où l’attribut était utilisé si l’on n’avait pas employé d’accesseur.

4. L'encapsulation

4.3. Intérêt des accesseurs (Exemple)

- Dans la classe **Personne**, on peut déclarer l'attribut **age** avec une visibilité **public**.
- De cette façon tous les objets qui composent le système peuvent accéder et modifier l'attribut **age** des objets de la classe.
- Si on décide de remplacer l'attribut `int age` par **int anneeDeNaissance**, partout où l'on a utilisé l'attribut **age**, il faut modifier le code.

```
public class Personne{
        // attributs
        ...
        public int age;
        // methodes
        …
}
```

```
public class Personne{
        // attributs
        ...
        private int anneeDeNaissance;
        // methodes
        …
}
```

```
Public class UneClasse {
…
Personne p=new Personne(…);
int  x=p.age;
…
p.age=15;

… }
```

```
Public class UneClasse {
…
Personne p=new Personne(…);
int  x=p.age;
…
p.age=15;

… }
```

4. L'encapsulation

4.3. Intérêt des accesseurs (Exemple)

- Si l'on a utilisé **`getAge()`** et **`setAge()`**, alors, il suffit de changer le code des accesseurs.

```
public class Personne{
        // attributs
        ...
        private int age;
        // method
        ...
        public int getAge(){
    return age;
    }
        public void setAge(int a){
    age=a;
    }
}
```

```
import java.util.GregorianCalendar;
public class Personne{
    // attributs
    ...
    private int anneeDeNaissance;
    // method
    //...
   public int getAge(){
   GregorianCalendar d = new GregorianCalendar();
   return d.get(d.YEAR)-anneeDeNaissance;
   }

   public void setAge(int a){
   GregorianCalendar d = new GregorianCalendar();
   anneeDeNaissance=d.get(d.YEAR)-a;
   }
}
```

```
Public class UneClasse {
…
Personne p=new Personne(…);
int  x=p.getAge();
….
p.setAge(14);

… }
```

5. Les packages

5. Les packages

5.1. Définition

- Les classes Java sont regroupées en paquetages (**packages** en anglais);

- Le paquetage est un moyen de modularité qui permet de:

  - Découper une application volumineuse en packages regroupant les classes qui couvrent un même domaine;

  - protéger des attributs et des méthodes;

5. Les packages

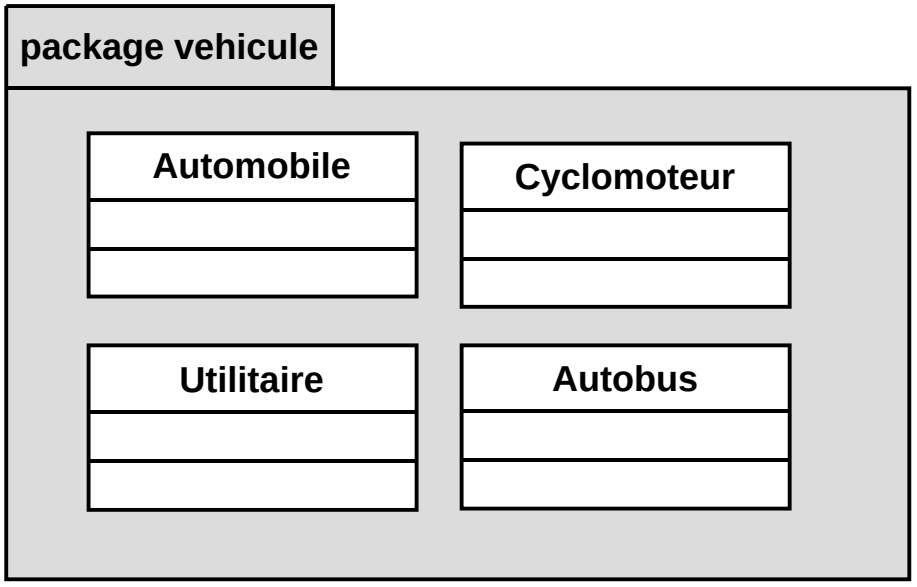
5.2. Attribution d'un nom à un package

- Chaque package porte un nom. Par convention, le nom d'un package commence par une minuscule.
- Toute classe appartenant à un package doit commencer par déclarer son appartenance à ce package, à l'aide de l'instruction :

```
package nomDuPackage;
```

Exemple

```
package vehicules;
public class Automobile{...}

package vehicules;
public class Cyclomoteur{...}

package vehicules;
public class Utilitaire{...}

package vehicules;
public class Autobus{...}
```



5.3. Contrôle d'accès aux classes

- Pour les classes, il n'y a que deux niveaux de visibilité :

1. **Publique**: La classe est visible par les classes de son package, et à l'extérieur du package.
   - La syntaxe pour déclarer une classe publique consiste à écrire :

`public class UneClasse{...}`

Exemple

```
public class Automobile{...}
```

2. **Sans visibilité**: La classe n'est visible que par les classes du *package* dans lequel elle se trouve.
   - La syntaxe consiste à écrire :

`class UneClasse{...}`

Exemple

```
class Point {...}
```

5.3. Utilisation d'un package

- Pour désigner une classe qui est définie dans un autre package, on a le choix entre :

1. Importer la classe :

```
import nomPackage.nomClasse;
```

2. Faire précéder chaque occurrence du nom de la classe par le nom du package dans lequel elle est définie.

Exemple

```
//Declaration de la classe personne
package proprietaire;
public class Personne{...}
```

5. Les packages

Exemple (suite)

Pour désigner au sein du package **vehicule** la classe **Personne** qui appartient au package **proprietaire**, on a le choix entre le code suivant:

```
package vehicule;
import proptietaire.Personne;
public class Automobile{
...
Personne p;
p=new Personne(String prenom, String nom)
...
}
```

Ou bien le code suivant :

```
package vehicule;
public class Automobile{
...
proptietaire.Personne p;
p=new proptietaire.Personne(String prenom, String nom)
...
}
```

5.3. Utilisation d'un package

- Pour importer toutes les classes d'un package :

```
import nomPackage.*;
```

Exemple

On peut remplacer le code suivant:

```
import vehicule.Automobile
import vehicule.Cyclomoteur
import vehicule.Utilitaire
import vehicule.Autobus
```

Par le code suivant:

```
import vehicule.*
```

6. Passage de paramètres

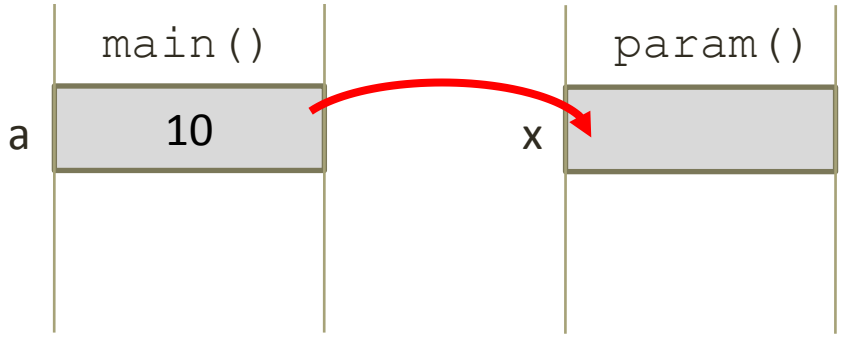
6. Passage de paramètres

- En Java, les paramètres sont toujours **passés par valeur**, c'est à dire que la valeur du paramètre effectif est recopiée dans le paramètre formel correspondant:
  - A l'appel de chaque méthode, de l'espace mémoire local est alloué pour chaque paramètre formel;
  - Les valeurs des paramètres effectif sont copiées avant l'appel de la méthode;
  - Le calcul s'effectue sur les paramètres formels;

6. Passage de paramètres

- **Exemple**

```
public class Test {

  public  void param(double x)
   {
   x=5
   }
  public static void main(String arg[])
   {
   Test test=new Test();
   double a = 10;
   System.out.println("Avant l'appel de param: a="+a);
   test.param(a);
   System.out.println("Aprés l'appel de param: a="+a);

   }
}
```



Résultat affiché:
Avant l'appel de param: a=10.0
Aprés l'appel de param: a=10.0

6. Passage de paramètres

Cas du passage d'un objet

- Quand on passe un objet en paramètres, c'est la référence sur cet objet qui est passée et copiée comme paramètre formel.
- Donc si l'objet est modifié dans la méthode, les modifications seront visibles de l'extérieur.

6. Passage de paramètres

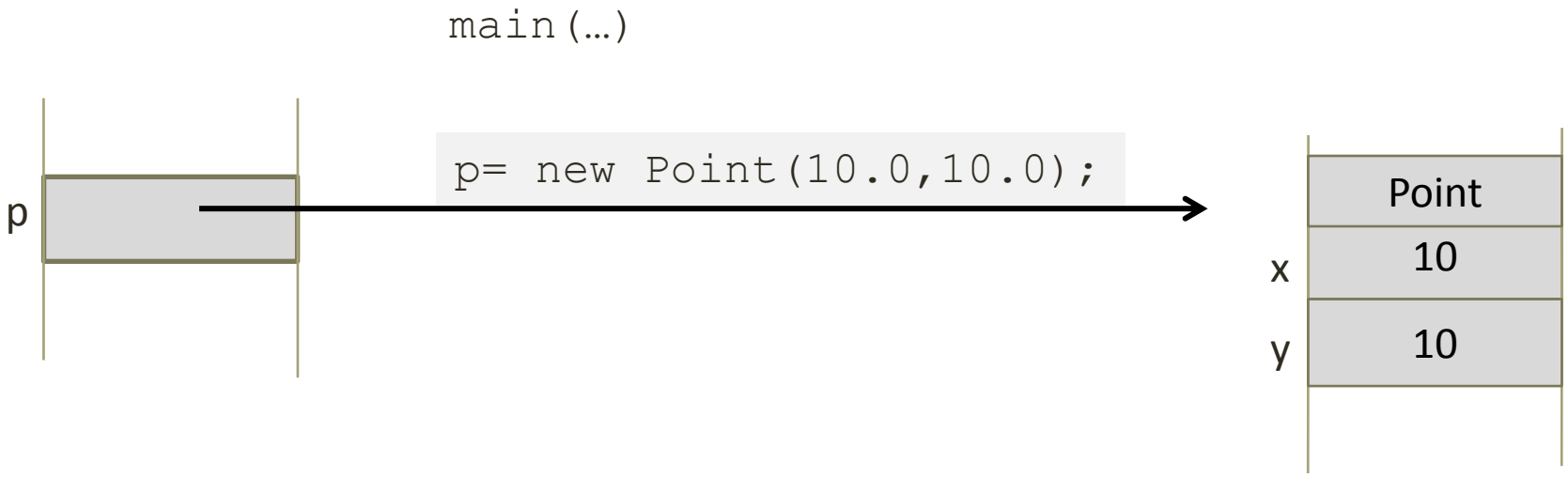
Exemple

```
public class Point{
  private double x;
  private double y;
  public Point(double x, double y){this.x=x; this.y=y; }
  public static void deplacer (Point pt, double dx, double dy){
       pt.x=pt.x+dx;
       pt.y=pt.y+dy;
  }
  public String toString(){ return "Point(" + x +","+ y +")";}

  public static void main(String arg[]){
    Point p= new Point(10.0,10.0);
    System.out.println("Avant l'appel de deplacer "+p.toString());
    deplacer (p,5.0,5.0);
    System.out.println("Aprés l'appel de deplacer "+p.toString());
  }
}
```

Résultat affiché
Avant l'appel de deplacer Point(10.0,10.0)
Aprés l'appel de deplacer Point(15.0,15.0)

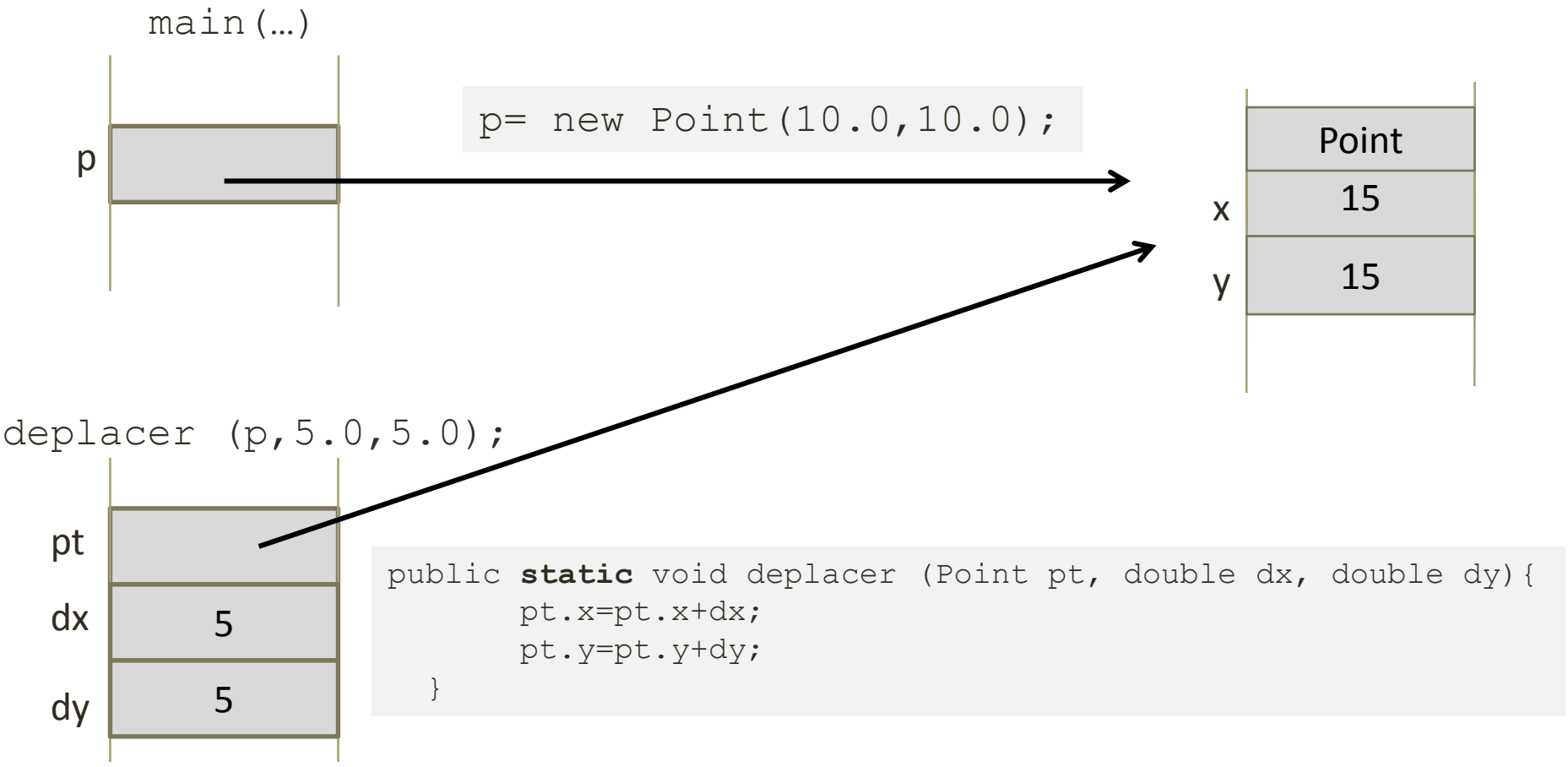
6. Passage de paramètres

Exemple



6. Passage de paramètres

Exemple




```
public static void deplacer (Point pt, double dx, double dy){
        pt.x=pt.x+dx;
        pt.y=pt.y+dy;
   }
```

Résultat affiché
Avant l'appel de deplacer Point(10.0,10.0)
Aprés l'appel de deplacer Point(15.0,15.0)

7. Les éléments statiques

7. Les éléments statiques

7.1. Les attributs statiques (les attributs de classes)

- Les attributs statiques sont définis avec le mot clé `static;`
- Il n'existe qu'une seule copie de l'*attribut statique* pour tous les objets de la classe;
- Si un seul objet modifie la valeur d'un attribut statique, sa valeur sera modifiée pour tous les objets de la classe.
- Pour accéder à un attribut statique, on utilise la notation:

NomClasse.nomAttribut

Exemple

```
public class Voiture
{
    static byte nbRoues = 4;
    private double longueur;
    private byte nbPassagers;
}
```

7. Les éléments statiques

7.1. Les attributs statiques (les attributs de classes)

- Un usage classique de l'attribut statique est donné par l'exemple suivant:

Exemple:

Nous voulions ajouter un attribut d'identification «**id**» à la classe `Personne`, tel que chaque objet de la classe `Personne` aura sa propre valeur pour cette attribut (deux objets ne doivent pas avoir la même valeur pour l'attribut « **id** ».

Solution:

1. Déclaration de l'attribut « **id** » et d'un attribut **static** « **nombre** » initialisé à 0.
2. Dans le constructeur de la classe `Voiture` : Affectation de la valeur de l'attribut «`nombre` » à l'attribut « `id` », et incrémentation de la valeur de l'attribut « nombre » .

7. Les éléments statiques

7.1. Les attributs statiques (les attributs de classes)

- **Exemple (suite)**

```
public class Personne{
//Attributs
        private int id;
        public static int nombre=0;
        private String nom;
//Constructeur
        public Personne(String nom){
          id = nombre;
          nombre++;
          this.nom=nom;
         }
//Methode toString()
        public String toString()
        {
         return "Id:"+ this.id+", Nom:"+this.nom;
        }
}
```

7. Les éléments statiques

```
public class MainClass{
        public static void main(String arg[]){

         Personne p1=new Personne("Ahmed");
         Personne p2=new Personne ("Ali");
         Personne p3=new Personne ("Aicha");

         System.out.println(p1.toString());
         System.out.println(p2.toString());
         System.out.println(p3.toString());

         System.out.println("Nombred'objets="+Personne.nombre);
    }
}
```

Le résultat affiché est:

Id:0, Nom:Ahmed
Id:1, Nom:Ali
Id:2, Nom:Aicha
Nombre d'objets=3

7. Les éléments statiques

7.2. Les méthodes statiques

- L'intérêt des méthodes statiques est de pouvoir être appelé alors qu'on ne dispose pas d'un objet.
- Une méthode statique ne peut utiliser que des attributs et des méthodes statiques.
- Pour appeler une méthode statique :

NomClasse.nomMethode()

- **Exemple**

```
public class Additionneur
{
   public static int somme(int a, int b)
   {
      return (a+b);
   }
}
```

- Pour appeler la méthode `somme`, on aura pas besoin de créer un objet de la classe `Additionneur`, il suffit d'écrire par exemple:

Additionneur.somme(5,10);

7. Les éléments statiques

7.2. Les méthodes statiques

- La méthode `main()` est un exemple des méthodes statique.
- C'est la méthode `main` qui est appelée lorsque la `JVM` doit exécuter une classe particulière.
- La méthode `main()` est statique, c'est donc une méthode appelé par la classe et non pas un objet (Pas d'objet appelant).
  - Pour pouvoir appeler les méthodes d'un objet au sein de la méthode `main()`, il est nécessaire de créer un objet de cette classe au sein de la méthode `main()`.

Exemple

```
public class Personne{
//Attributs
...
//Méthodes
...
public  static void main(String[] args) {
setName("Ali");//erreur, main() n'est executé par aucun objet
Personne pers = new Personne();
pers.setNom("Ali");// correcte
}
}
```

8. La surcharge de méthodes (*overloading*)

8. La surcharge de méthodes (*overloading*)

- La surcharge consiste à définir plusieurs méthodes qui portent le même nom au sein de la même classe.

- Les méthodes qui portent le même nom ont des *signatures* différentes,

- On appelle *signature* d'une méthode l'ensemble constitué du *nom de la méthode* et des *paramètres* qui lui sont passés.

- Deux méthodes d'objets de la même classe qui portent le même nom mais n'ont pas les mêmes paramètres, n'ont pas la même signature et JAVA peut les distinguer.

- Le compilateur choisi la méthode qui doit être appelée en fonction du nombre et des types des paramètres.
- La surcharge permet de simplifier l'interface des classes vis à vis des autres classes.

8. La surcharge de méthodes (*overloading*)

Exemple

```
public class Additionneur{
      public int somme(int a, int b) // 1
      {return (a+b);}
      public int somme(int a, int b, int c) // 2
      {return (a+b+c);}
      public float somme(float a, float b) // 3
      {return (a+b);}
      public float somme(int a, int b) //4
      {return ((float)a+(float)b);} //erreur
      }
```

La déclaration de la méthode 4 provoque une erreur, car, elle a la même signature que la méthode 1.