POO EN JAVA

Pr. Abdelmajid HAJAMI

BTS-DSI

2014

PLAN

- Concepts de la POO
- Présentation de JAVA
- Généralités
- Les types primitifs de JAVA
- Les instructions de contrôle de JAVA
- Les classes et les objets
- Les tableaux
- L'héritage
- Les chaînes de caractères et les types énumérés

CONCEPTS DE LA POO

CONCEPTS DE L'ORIENTÉ OBJET

- En approche orientée objet, le logiciel est considéré comme une collection d'objets qui collaborent entre eux, pour réaliser le métier du client.
- Les objets sont identifiés, et possèdent des caractéristiques statiques et dynamiques

CONCEPTS DE L'ORIENTÉ OBJET

Objet

- Un objet du monde réel est une chose concrète ou abstraite.
- Le monde réel n'est composé que de chose ou objets! **Tout est objet!**

- Un objet informatique est une représentation (un modèle) d'un objet du monde réel. Il modélise une chose abstraite ou concrète
- Un modèle est une abstraction, c-à-d un mécanisme qui permet de ne retenir que les caractéristiques essentielles d'un objet

CONCEPTS DE L'ORIENTÉ OBJET

Classe

- Il s'agit d'un modèle abstrait de données et de traitement représentant un objet réel dans la nature et regroupant des caractéristiques (attributs et méthodes) communes à des objets.
- Un objet est une simple instance de la classe

CONCEPTS DE L'ORIENTÉ OBJET

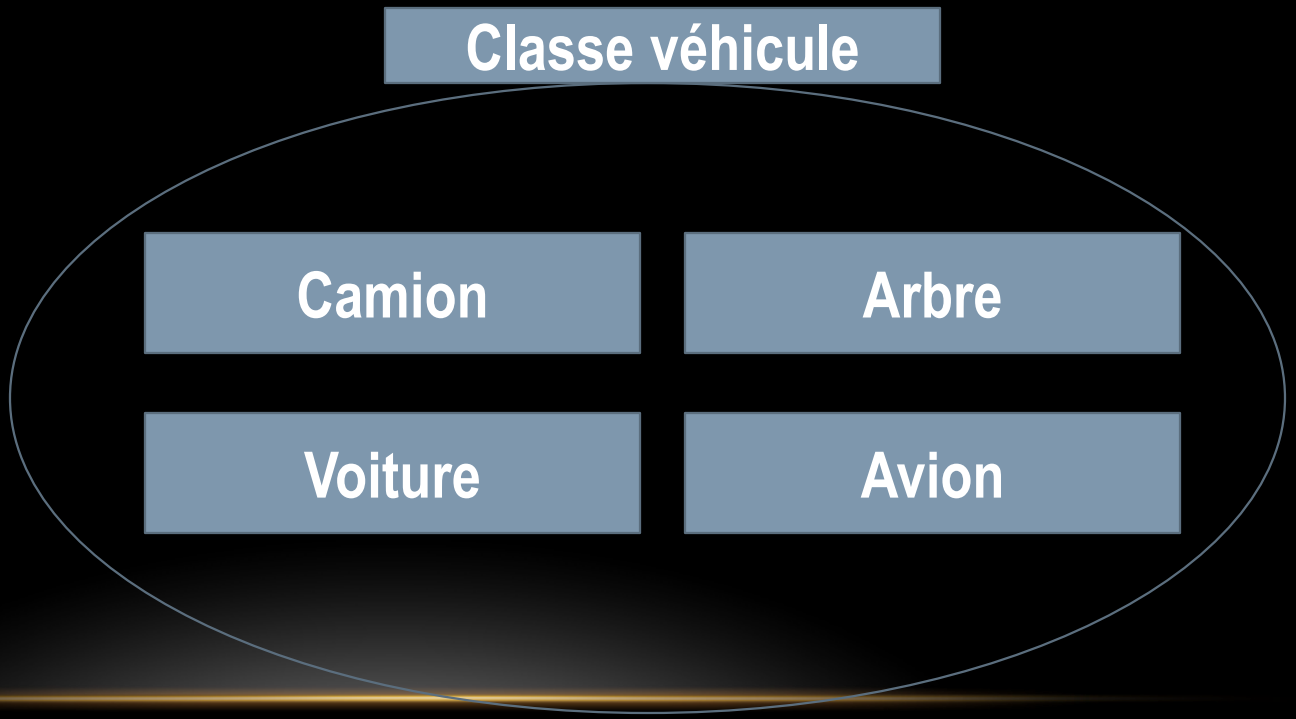
Classe

- En UML, elle est représenté par:

CONCEPTS DE L'ORIENTÉ OBJET

Classe

- Chasser l'intrus :



CONCEPTS DE L'ORIENTÉ OBJET

Identité

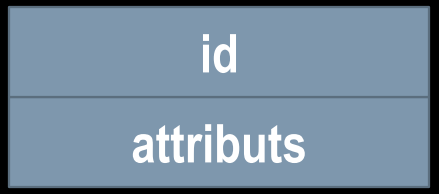
- Tout objet possède une identité , qui permet de l'identifier et de le différencier des autres objets.
- L'id se fait par le choix d'une variable unique qui permet de spécifier l'objet dans un système
- L'id permet de rendre l'objet persistant dans le mapping O/R ou bien avec les annotations de l'API JPA (Java Persistant API)

id

CONCEPTS DE L'ORIENTÉ OBJET

Attributs

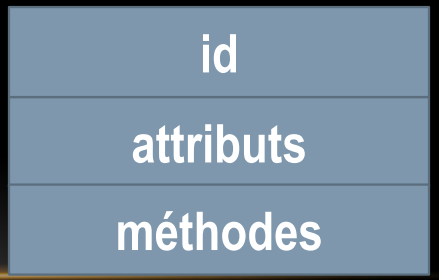
- Ce sont les données qui caractérisent l'objet. Ou des variables qui indiquent l'état de l'objet dans le temps.
- Ils présentent en plus de l'identité, la partie statique d'une classe



CONCEPTS DE L'ORIENTÉ OBJET

Méthodes

- Elles définissent le comportement (behavior) d'un objet, c-à-d l'ensemble des opérations que l'objet est chargé de réaliser dans un logiciel.
- Ces opérations permettent de faire réagir l'objet aux sollicitations extérieurs, ou d'agir sur les autres objets
- Les opérations sont liées aux attributs, car leurs actions peuvent dépondre des valeurs d'attributs ou les modifier.
- L'approche OO associe les données et les traitements au seins de la même classe.



CONCEPTS DE L'ORIENTÉ OBJET

Méthode abstraite

- C'est une opération de la classe avec uniquement une signature et sans corps.
- La signature d'une méthode abstraite décrit le nom, le type de retour, les argument et leurs types

CONCEPTS DE L'ORIENTÉ OBJET

Classe abstraite

- C'est une classe qui possède au moins une méthode abstraite.
- Une classe abstrait *n'est pas instanciable*
- elle représente un modèle de classe comme les interfaces; mais au contraire des interfaces, les classes abstraite peuvent avoir des attributs

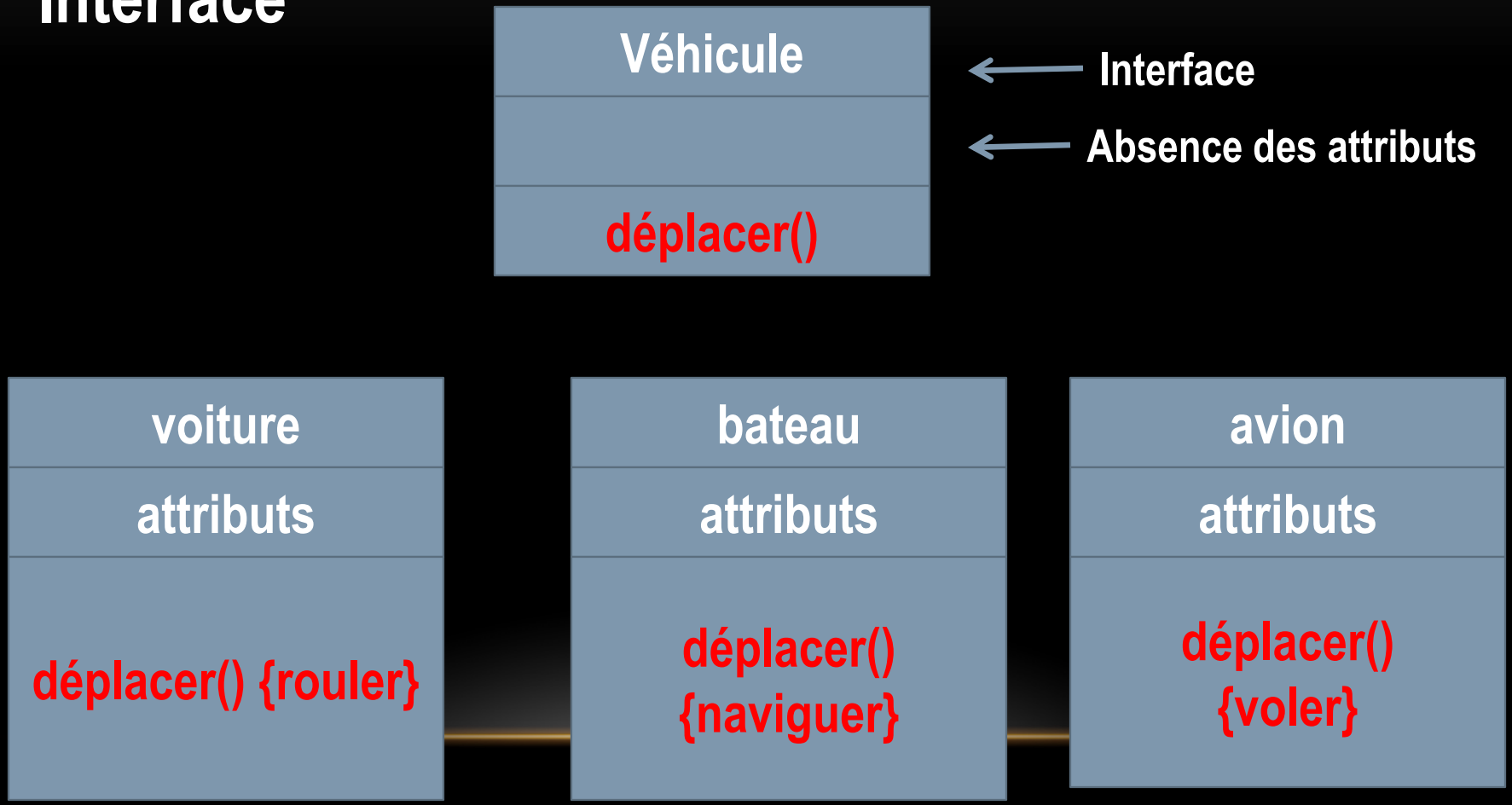
CONCEPTS DE L'ORIENTÉ OBJET

Interface

- C'est un modèle de classe, permettant de rassembler les classes par leur comportements seulement
- Les classes qui implémente une interface, définissent les méthodes abstraite, chacune à sa façon.

CONCEPTS DE L'ORIENTÉ OBJET

Interface



CONCEPTS DE L'ORIENTÉ OBJET

Encapsulation

- Elle consiste à masquer les détails de l'implémentation d'un objet, en définissant des modifications de visibilité à ses attributs et ses méthodes.
- Elle garantit l'intégrité et la sécurité d'accès aux données, car elle interdit l'accès direct aux caractéristiques des objets.

CONCEPTS DE L'ORIENTÉ OBJET

Encapsulation

- Elle est assurée par les modifications de visibilité suivante:

accessibilité	public	private	protected	Package freindly
À l'intérieur de la classe	OUI	OUI	OUI	OUI
Classes du même package	OUI	NON	OUI	OUI
Classes dérivées	OUI	NON	OUI	NON
Classes hors du package	OUI	NON	NON	NON

CONCEPTS DE L'ORIENTÉ OBJET

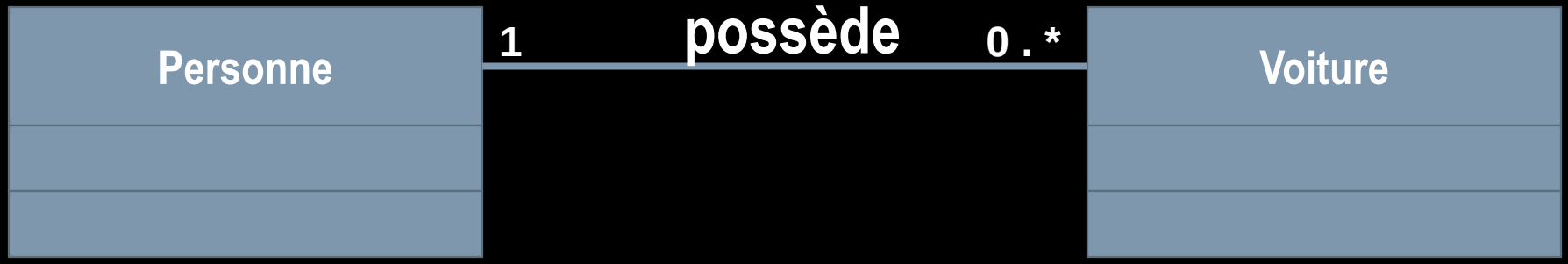
Association

- Une association normale est une ***connexion*** entre classes
- elle possède un ***nom***
- elle est généralement ***navigable*** de manière bidirectionnelle (dans ce cas elle a un nom dans les deux sens si nécessaire)
- elle a une ***multiplicité***
- elle peut être dotée de ***rôles***

CONCEPTS DE L'ORIENTÉ OBJET

Cardinalité (Multiplicité)

- Elle indique le nombre d'instances (objets) d'une classe associées à **une** instance de l'autre classe



CONCEPTS DE L'ORIENTÉ OBJET

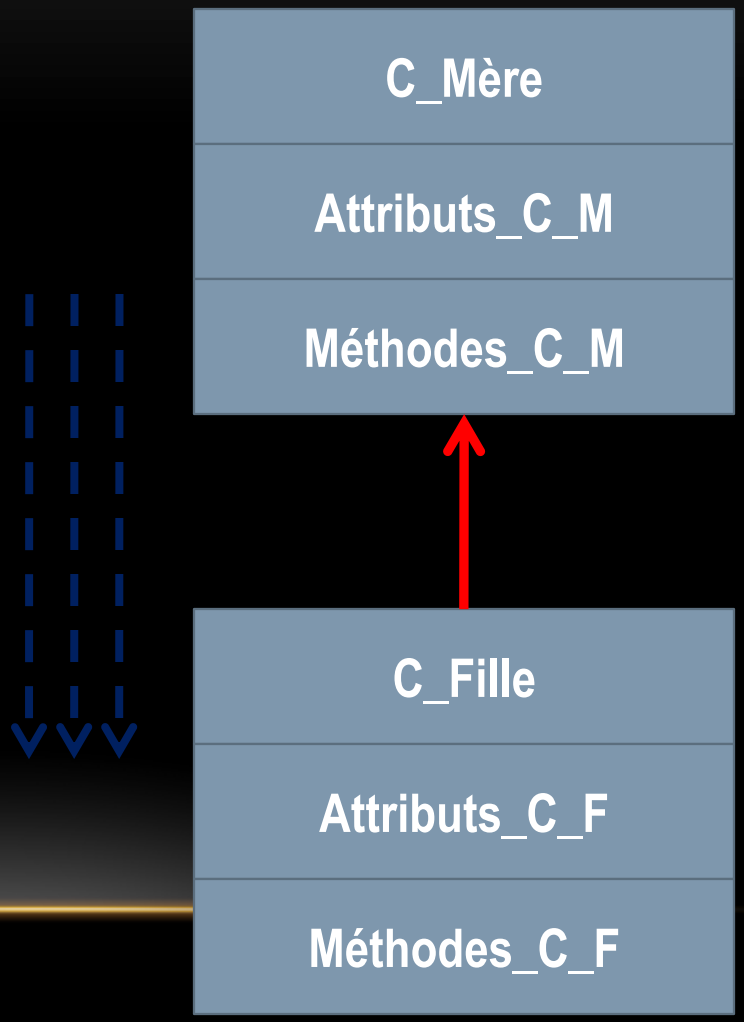
Héritage

- C'est un concept de passage des caractéristiques de la classe **mère** (ses attributs et méthodes) vers la classe **fille** à condition que les contraintes d'encapsulation le permettent (public, protected).
- Une classe peut être dérivable en d'autres classes , à fin d'y ajouter des caractéristiques spécifiques ou d'en adapter certaines (spécialisation).
- Plusieurs classes peuvent être « généralisées » en une classe qui les factorise, à fin de regrouper leurs caractéristiques communes (généralisation).
- L'héritage peut être simple ou multiple.
- L'héritage multiple n'est pas supporté par JAVA et il est supporté par C++

CONCEPTS DE L'ORIENTÉ OBJET

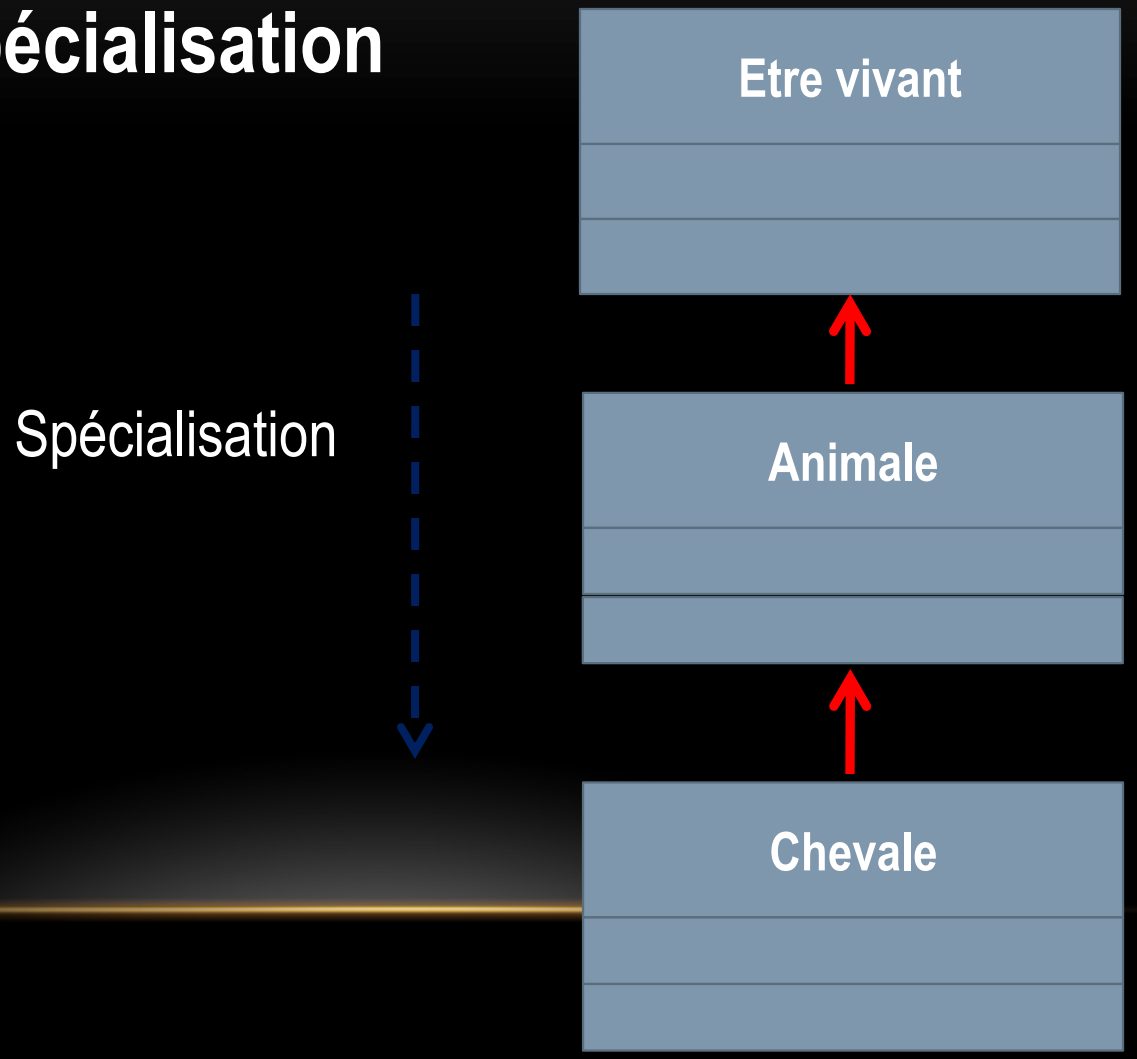
Héritage

Passage des attributs et des méthodes de la classe mère vers la classe fille



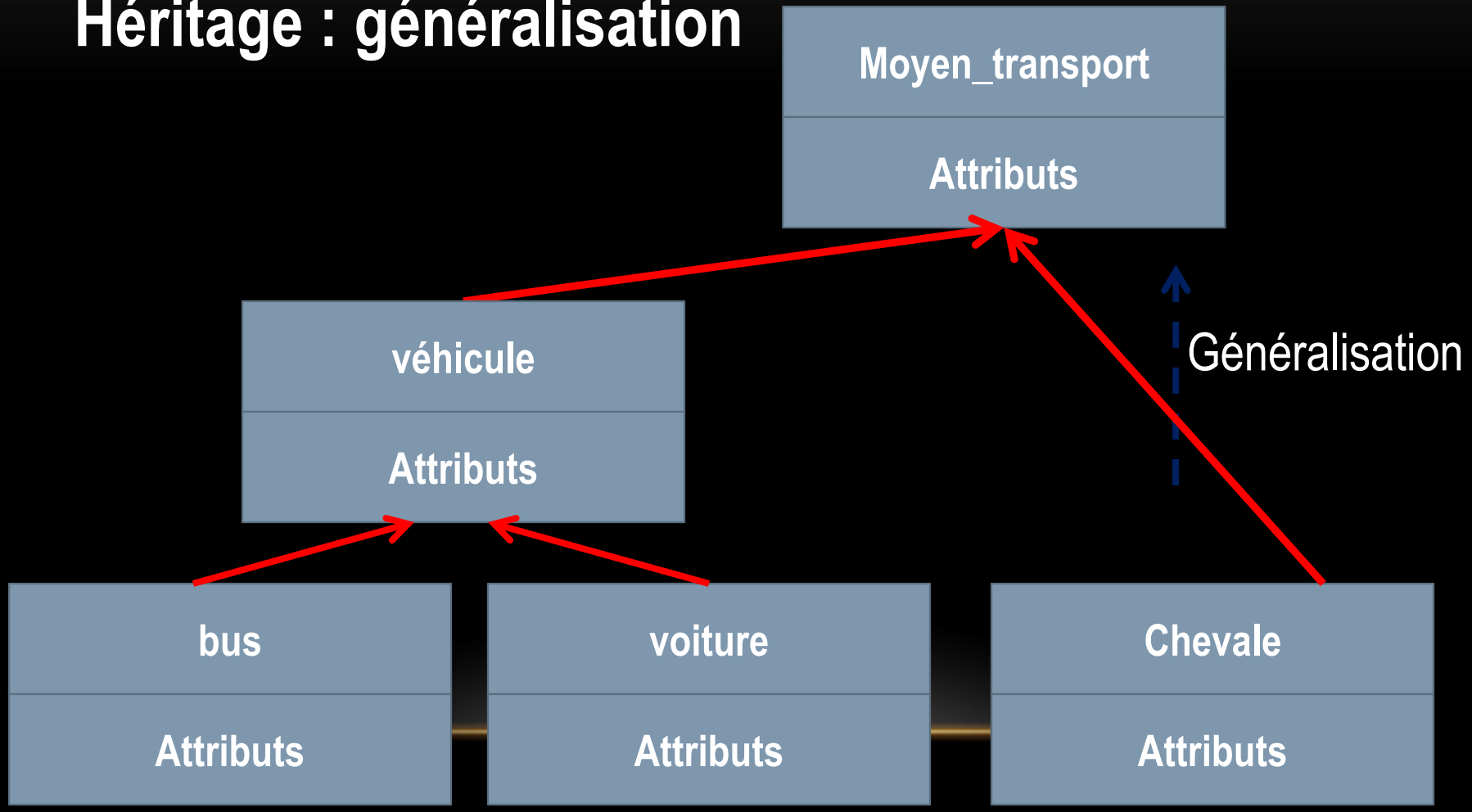
CONCEPTS DE L'ORIENTÉ OBJET

Héritage : spécialisation



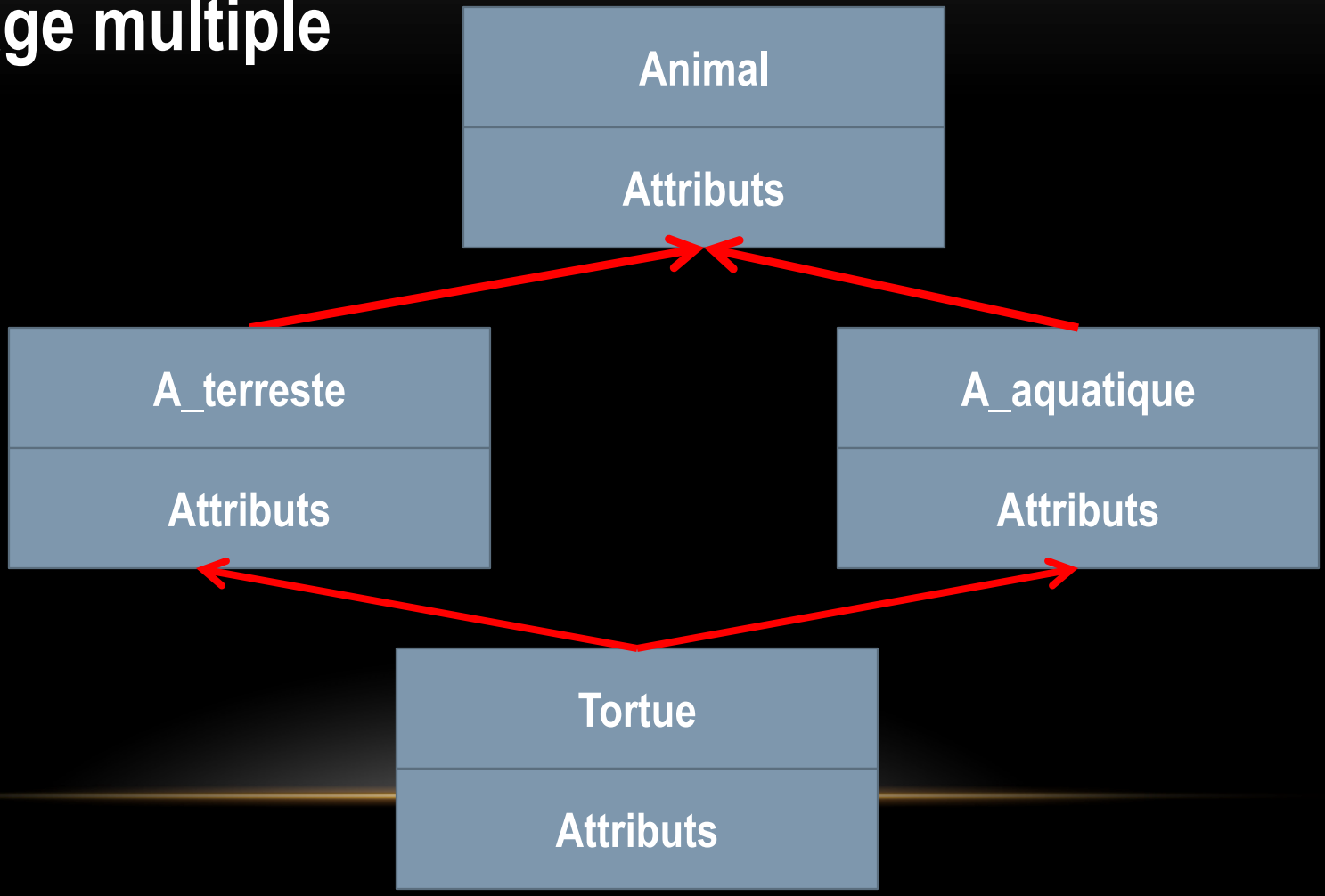
CONCEPTS DE L'ORIENTÉ OBJET

Héritage : généralisation



CONCEPTS DE L'ORIENTÉ OBJET

Héritage multiple



CONCEPTS DE L'ORIENTÉ OBJET

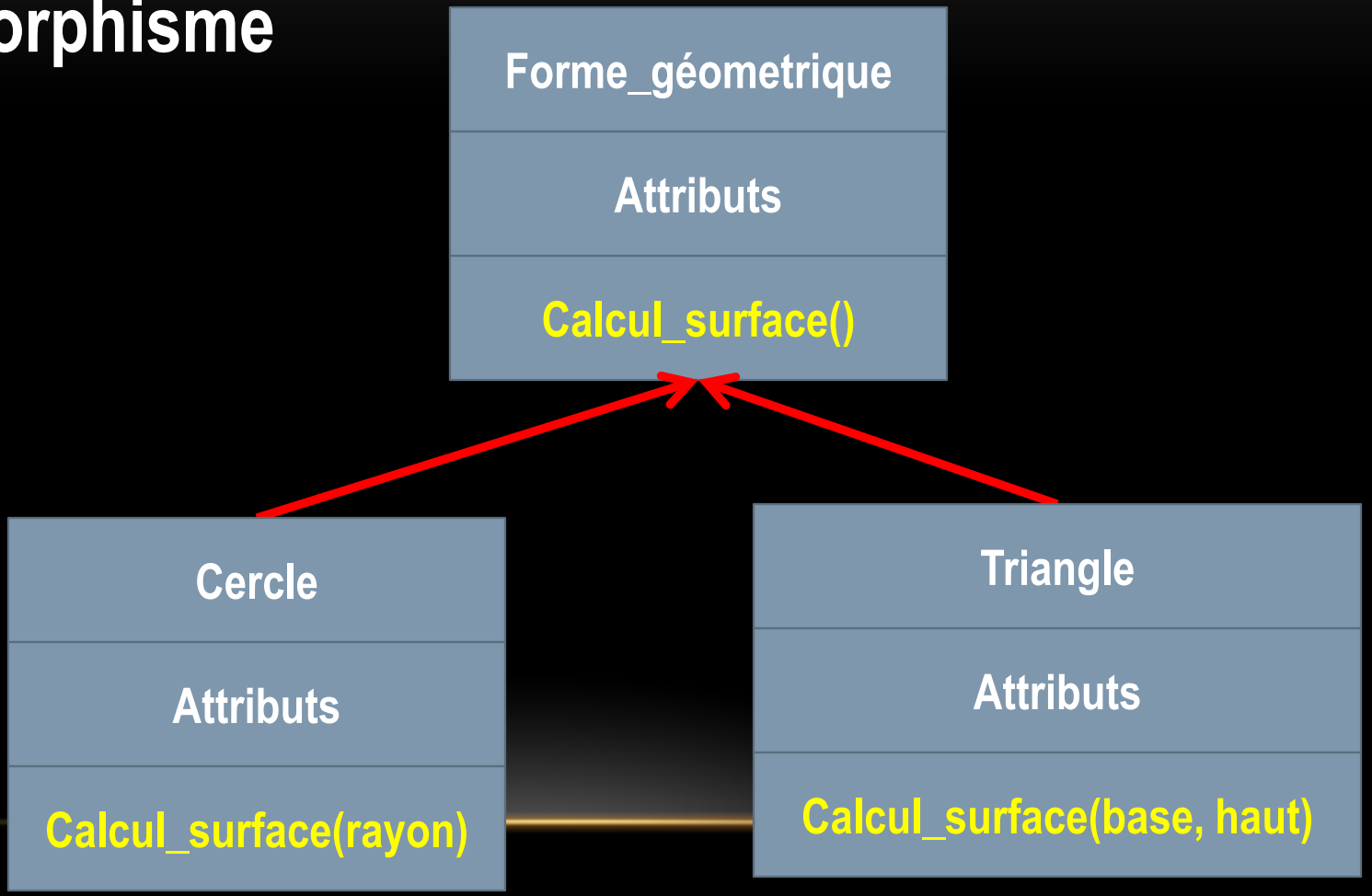
Polymorphisme

Grèc --> Poly : différents; morphisme : forme

- Le concept qui fournit la possibilité d'utiliser la même méthode dans un programme, mais avec des traitements différents, grâce à l'héritage et la redéfinition de la méthode polymorphe dans les classes dérivées

CONCEPTS DE L'ORIENTÉ OBJET

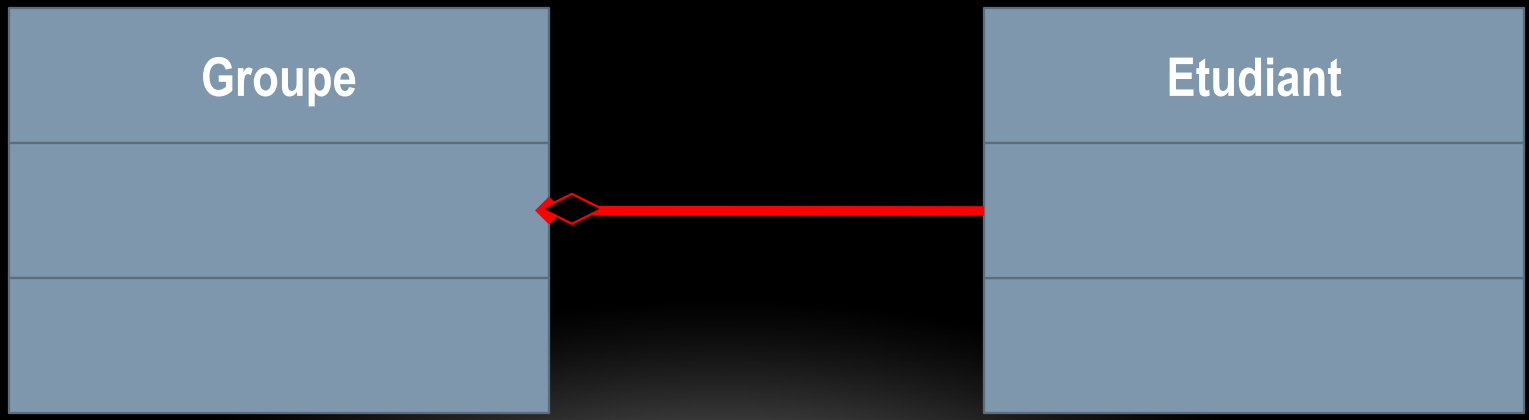
Polymorphisme



CONCEPTS DE L'ORIENTÉ OBJET

Agrégation

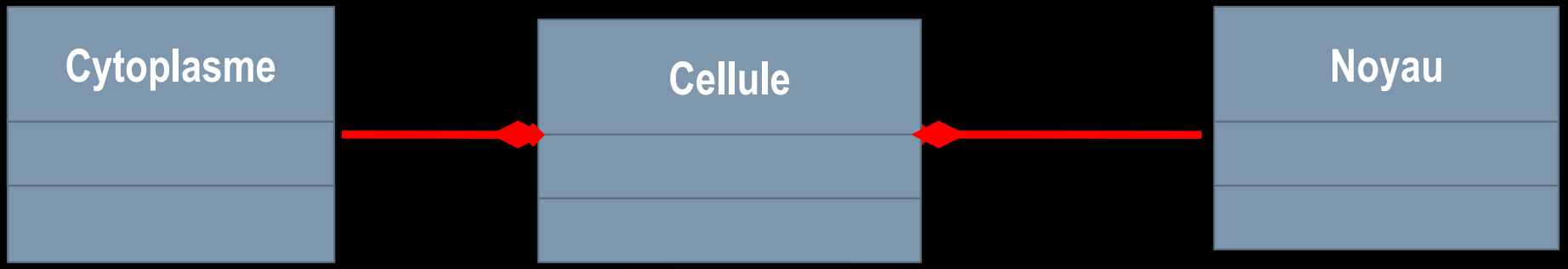
- Il s'agit d'une relation d'inclusion entre deux classes, spécifiant que les objets d'une classe (agrégat) sont des composants (agrégés)de l'autre classe composite.



CONCEPTS DE L'ORIENTÉ OBJET

Composition

- C'est une agrégation forte : la classe composite et les classes composants ont la même durée de vie



PRÉSENTATION DE JAVA

PRÉSENTATION DE JAVA

PHASES DE DEVELOPPEMENT D'UN PROGRAMME JAVA

Phase 1: Edition

- Editeur
  - + vi et emacs sous UNIX
  - + Bloc-notes sous WINDOWS
  - + Environnements de Développement Intégrés (EDI): JBuilder de Borland, NetBeans, Visual Cafe de Symantec, Visual J++ de Microsoft
- Le nom de fichier d'un programme Java se termine toujours par l'extension .java.
- Exemple: Programme.java

PRÉSENTATION DE JAVA

PHASES DE DEVELOPPEMENT D'UN PROGRAMME JAVA

Phase 2: Compilation

- La commande du compilateur Java pour compiler un programme Java et le traduire en byte codes, est javac.

- La compilation génère un fichier possédant le même nom que la classe et contenant les bytes codes avec l'extension .class. Le compilateur génère un fichier compilé pour chaque classe.

Ainsi, si le fichier source contient plusieurs classes, alors plusieurs fichiers ont l'extension .class.

- Exemple: javac Programme.java génère un fichier Programme.class

Mettre l'extension à la suite du nom en respectant la casse du nom de fichier.

Java est sensible à la casse.

PRÉSENTATION DE JAVA

PHASES DE DEVELOPPEMENT D'UN PROGRAMME JAVA

Phase 3: Chargement

- Le chargeur de classe prend le ou les fichiers .class et les transfère en mémoire centrale
- Le fichier .class peut être chargé à partir d'un disque dur de sa propre machine ou à travers un réseau
- 2 types de fichier .class peuvent être chargés: les applications (programmes exécutés sur sa propre machine) et les applets (programmes stockés sur une machine distante et chargés dans le navigateur Web).

- Une application peut être chargée et exécutée par la commande de l'interpréteur Java

java

- Exemple: java Programme

Pas d'extension .class à la suite du nom.

PRÉSENTATION DE JAVA

PHASES DE DEVELOPPEMENT D'UN PROGRAMME JAVA

Phase 4: Vérification

Les byte codes dans une applet sont vérifiés par le vérificateur de byte codes avant leur exécution par l'interpréteur Java intégré au navigateur ou à l'appletviewer. Ce vérificateur vérifie que les byte codes sont conformes aux restrictions de sécurité de Java concernant les fichiers et la machine.

PRÉSENTATION DE JAVA

PHASES DE DEVELOPPEMENT D'UN PROGRAMME JAVA

Phase 5: Exécution

L'ordinateur interprète le programme byte code par byte code.

Les interpréteurs présentent des avantages sur les compilateurs dans le monde Java. En effet, un programme interprété peut commencer immédiatement son exécution dès qu'il a été téléchargé sur la machine cliente, alors qu'un programme source devant subir une compilation supplémentaire entraînerait un délai de compilation avant de pouvoir démarrer son exécution.

Cependant, dans des applets à forte charge de calcul, l'applet doit être compilé pour augmenter la rapidité d'exécution.

GÉNÉRALITÉS

GÉNÉRALITÉS

PROGRAMME ECRITURE CONSOLE

- **Problème: Ecriture d'un texte dans une fenêtre console**
- Fichier: Ecriture.java

```
public class Ecriture
{
    public static void main(String[] args)
    {
     System.out.println("Un programme Java");
    }
}
```

GÉNÉRALITÉS

PROGRAMME ECRITURE CONSOLE

- **Problème: Ecriture d'un texte dans une fenêtre console**
- Fichier: Ecriture.java

```
public class Ecriture
{
        public static void main(String[] args)
        {
         System.out.println("Un programme Java");
        }

}
```

- Exécution:

```
Un programme Java
```

GÉNÉRALITÉS

PROGRAMME ECRITURE FENETRE

- Problème: Ecriture d'un texte dans une fenêtre graphique
- Fichier: EcritureFenêtre.java

```
import javax.swing.*;
public class EcritureFenêtre
{
        public static void main(String[] args)
        {
          JOptionPane.showMessageDialog(null,"Fenêtre Java");
        }
}
```

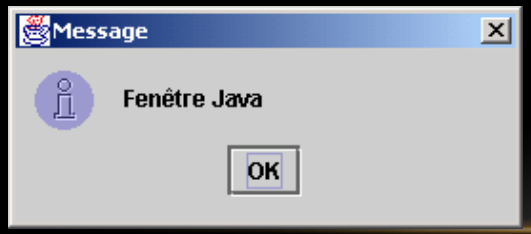
GÉNÉRALITÉS

PROGRAMME ECRITURE FENETRE

- Problème: Ecriture d'un texte dans une fenêtre graphique
- Fichier: EcritureFenêtre.java

```
import javax.swing.*;
public class EcritureFenêtre
{
        public static void main(String[] args)
        {
          JOptionPane.showMessageDialog(null,"Fenêtre
Java");
        }
}
```

Exécution :



GÉNÉRALITÉS

PROGRAMME ECRITURE FENETRE

- Problème: Ecriture d'un texte dans une Applet
- Fichier: EcritureApplet.java

```
import java.awt.*;
import javax.swing.*;

public class EcritureApplet extends JApplet
{
  public void paint(Graphics g)
  {
    g.drawString("APPLET JAVA", 100, 100);
  }
}
```

GÉNÉRALITÉS

PROGRAMME ECRITURE FENETRE

- Problème: Ecriture d'un texte dans une Applet
- Fichier: EcritureApplet.java
- Exécution :


```
import java.awt.*;
import javax.swing.*;

public class EcritureApplet extends JApplet
{
  public void paint(Graphics g)
  {
    g.drawString("APPLET JAVA", 100, 100);
  }
}
```

GÉNÉRALITÉS

PROGRAMME LECTURE

- Problème: Lecture d'un texte d'une fenêtre console
- Fichier: : Lecture.java

Lecture à partir du clavier 1/4

```
import java.io.*;

// Méthodes de lecture au clavier
public class Lecture
{
  // Lecture d'une chaîne
  public static String lireString()
  {
    String ligne_lue = null;
    {
      InputStreamReader lecteur = new InputStreamReader(System.in);
      BufferedReader entree = new BufferedReader(lecteur);
      ligne_lue = entree.readLine();

    try
    }
    catch (IOException err)
    {
      System.exit(0);
    }
    return ligne_lue;
  }
```

Lecture à partir du clavier 2/4

```
// Lecture d'un réel double
 public static double lireDouble()
 {
   double x = 0;
     String ligne_lue = lireString();
     x = Double.parseDouble(ligne_lue);
   try
   {
   }
   catch (NumberFormatException err)
   {
     System.out.println("Erreur de donnée");
     System.exit(0) ;
   }
   return x;
 }
```

Lecture à partir du clavier 3/4

```
// Lecture d'un entier
public static int lireInt()
{
  int n = 0;

  try
  {
    String ligne_lue = lireString();
    n = Integer.parseInt(ligne_lue);
  }
  catch (NumberFormatException err)
  {
    System.out.println ("Erreur de donnée");
    System.exit(0);
  }
  return n;
}
```

Lecture à partir du clavier 4/4

```
// Programme test de la classe Lecture
public static void main (String[] args)
{
  System.out.print("Donner un double:  ");
  double x;
  x = Lecture.lireDouble();
  System.out.println("Résultat " + x);
  System.out.print("Donner un entier: « )
  int n;
  n = Lecture.lireInt();
  System.out.println("Résultat " + n);
 }
}
```

Exécution
Donner un double: 10.01
Résultat 10.01
Donner un entier: 10
Résultat 10

GÉNÉRALITÉS

REGLES GENERALES D'ECRITURE

Identificateurs

Dans un langage de programmation, un identificateur est une suite de caractères servant à désigner les différentes entités manipulées par un programme : variables, fonctions, classes, objets...

GÉNÉRALITÉS

REGLES GENERALES D'ECRITURE

Identificateurs

Par convention

- **Les identificateurs de classes commencent toujours par une majuscule.**
- **Les identificateurs de variables et de méthodes commencent toujours par une minuscule.**
- **Les identificateurs formés par la concaténation de plusieurs mots, comporte une majuscule à chaque début de mot sauf pour le 1er mot qui dépend du type d'identificateur.**
- **Exemple: public class ClasseNouvelle**

GÉNÉRALITÉS

REGLES GENERALES D'ECRITURE

Les mots clés

Certains mots-clés sont réservés par le langage à un usage bien défini et ne peuvent pas être utilisés comme identficateurs.

En voici la liste, par ordre alphabétique :

abstract assert boolean break byte
case catch char class const
continue default do double else
extends final finally float for
goto if implements import instanceof
int interface long native new
null package private protected public
return short static super switch
synchronized this throw throws transient
try void volatile while

GÉNÉRALITÉS

REGLES GENERALES D'ECRITURE

Les séparateurs

- Dans notre langue écrite, les mots sont séparés par un espace, un signe de ponctuation ou une fin de ligne. Il en va quasiment de même en Java.

Ainsi, on ne pourra pas remplacer :

```
int  x,y;
```

par

```
intx,y;
```

En revanche, vous pourrez écrire indifféremment :

```
int n,compte,p,total,valeur ;
```

ou, plus lisiblement :

```
int n, compte, p, total, valeur ;
```

voire :

```
int n,
compte,
p,
total,
valeur ;
```

GÉNÉRALITÉS

REGLES GENERALES D'ECRITURE

Les séparateurs : le format libre

```
// Calcul de racines carrees
// La classe Racines utilise la classe Clavier
public class Racines1  { public
static void main (String[] args)  { final int NFOIS = 5 ;  int i ; double
  x ;    double racx ;    System.out.println ( "Bonjour" ) ;  System.out.println ("Je vais vous calculer " +
  NFOIS  + " racines carrees") ;    for (i=0 ; i < NFOIS ; i++)  { System.out.print ("Donnez un nombre : ") ;
x  = Clavier.lireDouble  ()  ;  if (x < 0.0)  System.out.println (x + " ne possede pas de racine carree")
; else  { racx = Math.sqrt(x) ;  System.out.println (x + " a pour racine carree : "  + racx) ; } }
System.out.println("Travail termine - Au revoir") ;   }}
```

Les séparateurs : le format libre (écriture correcte)

```
// Calcul de racines carrees
// La classe Racines utilise la classe Clavier
public class Racines
{ public static void main (String[] args)
  { final int NFOIS = 5 ;
    int i ;
    double x ;
    double racx ;
    System.out.println ("Bonjour") ;
    System.out.println ("Je vais vous calculer " + NFOIS + " racines carrees") ;
    for (i=0 ; i<NFOIS ; i++)
      { System.out.print ("Donnez un nombre : ") ;
        x  = Clavier.lireDouble ()  ;
 if (x < 0.0)
          System.out.println (x + " ne possede pas de racine carree") ;
         else
          { racx = Math.sqrt(x) ;
            System.out.println (x + " a pour racine carree : "  + racx) ;
          }
      }
    System.out.println ("Travail termine - Au revoir") ;
  }
}
```

GÉNÉRALITÉS

REGLES GENERALES D'ECRITURE

Commentaires

3 formes

Commentaire usuel pouvant s'étendre sur plusieurs lignes: /* … */

Commentaire de fin de ligne s'arrêtant en fin de la ligne: //

Commentaire de documentation pouvant être extraits automatiquement par des programmes utilitaires de création de documentation tels que Javadoc qui génère automatiquement une documentation en format HTML: /** … */

GÉNÉRALITÉS

REGLES GENERALES D'ECRITURE

Commentaires

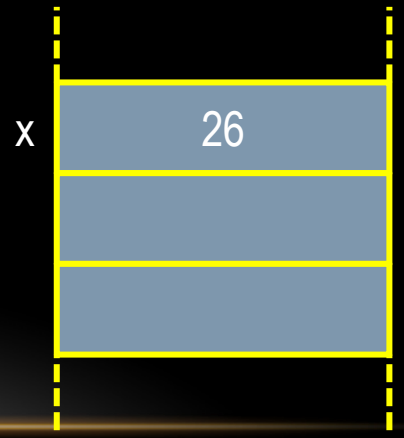
```
/* programme de calcul de racines carrees */
/* commentaire s'etendant
   sur plusieurs lignes
   de programme
*/
/* ================================================== *
 *              UN TITRE MIS EN VALEUR                *
 * ================================================== */
int i ;         /* compteur de boucle */
float x;        /* nombre dont on cherche la racine */
float racx ;    /* pour la racine carre de x */
```

LES TYPES PRIMITIFS DE JAVA

LES TYPES PRIMITIFS DE JAVA

Notion de type

- Dans tout programme, il est nécessaire de stocker, au moins provisoirement, des valeurs, de provenances variées (données d'entrée, résultats intermédiaires, ...)
- Le type d'une valeur permet de différencier la nature de l'information stockée dans une variable.



LES TYPES PRIMITIFS DE JAVA

TYPE BOOLEEN

Déclaration d'une variable booléenne

```
boolean test ;
```

Une variable booléenne prend 2 valeurs: false et true.

Affectation d'une variable booléenne

```
test = (n < m) ;
```

LES TYPES PRIMITIFS DE JAVA

TYPE ENTIER

- Le type entier permet de représenter de façon exacte une partie des nombres entiers relatifs.

Type	Taille (octet)	Valeur minimale	Valeur maximale
byte	1	-128 (Byte.MIN_VALUE)	127 (Byte.MAX_VALUE)
short	2	-32 768 (Short.MIN_VALUE)	32 767 (Short.MAX_VALUE)
int	4	-2 147 483 648 (Integer.MIN_VALUE)	2 147 483 647 (Integer.MAX_VALUE)
long	8	-9 223 372 036 854 775 808 (Long.MIN_VALUE)	9 223 372 036 854 775 807 (Long.MAX_VALUE)

Par défaut: une constante entière est de type int.

LES TYPES PRIMITIFS DE JAVA

TYPE REEL

- Le type réel permet de représenter de façon approchée une partie des nombres réels.

Type7	Taille (octet)	Précision (chiffres significatifs)	Valeur minimale	Valeur maximale
float	4	7	1.402E-45 Float.MIN_VALUE	3.402E38 Float.MAX_VALUE
double	8	15	4.94E-324 Double.MIN_VALUE	1.79E308 Double.MAX_VALUE

Par défaut: une constante réelle est de type double.

LES TYPES PRIMITIFS DE JAVA

TYPE REEL

- Problème: Ecriture d'une variable réelle en utilisant un formatage
- Fichier: EcritureReel.java

```
import java.awt.*;
public class EcritureReel
{
  public static void main(String[] args)
  {
    double  x = 10.123456789;
    System.out.println("x= " + x);
    // au moins 1 chiffre à gauche du point décimal
    // 2 chiffres exactement à droite du point décimal
    DecimalFormat  deuxDecimal = new DecimalFormat("0.00");
    System.out.println("x= " + deuxDecimal.format(x));
  }
}
```

Exécution
x= 10.123456789
x= 10,12

LES TYPES PRIMITIFS DE JAVA

TYPE CARACTERE

Le caractère en Java est représenté en mémoire sur 2 octets en utilisant le code Unicode.

Déclaration d'une variable caractère

```
char c;
```

Une constante caractère est notée entre apostrophe.

Exemple: 'a'

LES TYPES PRIMITIFS DE JAVA

Constantes et expressions constantes

Le mot-clé final

```
final int n = 20 ;
n = n + 5 ;                 // erreur : n a été déclarée final
n = Clavier.lireInt() ;   // idem
```

EXERCICES

1. Ecrire un programme qui permet de permuter deux nombres
2. Ecrire un programme qui lit le prix HT d'un article, le nombre d'articles et le taux de TVA, et qui fournit le prix total TTC correspondant. Faire en sorte que des libellés apparaissent clairement.

LES INSTRUCTIONS DE CONTRÔLE DE JAVA

LES INSTRUCTIONS DE CONTRÔLE DE JAVA

INSTRUCTION if

if (expression_booléenne) instruction_1

[else instruction_2]

LES INSTRUCTIONS DE CONTRÔLE DE JAVA

INSTRUCTION switch

```
switch (expression)
{
  case constante_1: [ suite_instructions_1 ]
  .....................
  case constante_n: [ suite_instructions_n ]
  [ default: suite_instructions ]
}
```

LES INSTRUCTIONS DE CONTRÔLE DE JAVA

INSTRUCTION while

while (expression_booléenne) instruction

INSTRUCTION do while

do instruction while (expression_booléenne);

LES INSTRUCTIONS DE CONTRÔLE DE JAVA

INSTRUCTION for

for ([initialisation]; [expression_booléenne] ; [incrémentations]) instruction

```
public class InstructionFor
{
  public static void main (String args[])
  {
    for (int i=1, j=1; (i <= 5); i++, j+=i)
     {
       System.out.println ("i= " + i + "   j= " + j);
     }
  }
}
```

EXERCICES

1- Ecrire le programme qui affiche le mois correspondant à un nombre compris entre 1 et 12 .

2- Traduire l'algorithme suivant en un programme JAVA

```
Algorithme DeuxChiffres ;
        constante (STOP : entier) ← 99999 ;
        variables uneVal, : entier ;
début
        ecrire ("Entrer un nombre,  ", STOP, " pour finir. ") ;
        lire (uneVal) ;
        tant que uneVal ≠ STOP faire
                si uneVal ≥ 10 et uneVal < 100
                alors ecrire ( uneVal, " est un nombre à deux chiffres. " ) ;
                finsi
        ecrire ("Entrer un autre nombre, " , STOP, " pour finir. ") ;
        lire (uneVal) ;
        fintantque
        ecrire (" fin de l'algorithme ") ;
fin
```

EXERCICES

3- Ecrire le programme de résolution d'une équation du second ordre dans l'ensemble C:

$$aX^2 + bX + c = 0$$

a, b et c sont lus à partir du clavier.

4- Écrire un programme qui détermine le factoriel d'un entier N en utilisant les boucles du type : Pour, Tant que et Répéter.

5- Écrire un algorithme qui calcule la somme des nombres entrés par l'utilisateur tant que cette somme est inférieure à mille.

EXERCICES

6- Ecrire le programme qui calcule la moyenne de N notes.

7- Ecrire un programme calculant la somme des n premiers termes de la "série harmonique", c'est-à-dire la somme :

1 + 1/2 + 1/3 + 1/4 + ... + 1/n

La valeur de n sera lue en donnée

LES CLASSES ET LES OBJETS

LES CLASSES ET LES OBJETS

- La notion de classe généralise la notion de type. La classe comporte des champs (données) et des méthodes.
- La notion d'objet généralise la notion de variable.
- Un type classe donné permet de créer (d'instancier) un ou plusieurs objets du type, chaque objet comportant son propre jeu de données.

LES CLASSES ET LES OBJETS

Définition d'une classe

- Nous nous proposons de définir une classe nommée Point ,destinée à manipuler les points d'un plan.

```
public class Point
{

    // instructions de définition des champs et des méthodes de la classe

}
```

LES CLASSES ET LES OBJETS

Définition des champs : attributs

- Nous supposons qu'un objet de type Point sera représenté par deux coordonnées entières :

```
public class Point
{

        private int x ;    // abscisse
        private int y ;    // ordonnée

}
```

LES CLASSES ET LES OBJETS

Définition des méthodes

```
public class Point
{
        public void initialise (int abs, int ord)
                {       x = abs ;
                        y = ord ;
                }
        public void deplace (int dx, int dy)
                {       x += dx ;
                        y += dy ;
                }
        public void affiche ()
        { System.out.println ("Je suis un point de coordonnées " + x + " " + y) ;
        }

    private int x ;    // abscisse
    private int y ;    // ordonnée
}
```

LES CLASSES ET LES OBJETS

Utilisation de la classe

```
public class TestPoint
{
        public static void main (String args[])
        {       Point a ; // a est un handle (une référence sur un objet)
                 a = new Point() ;
                a.initialise(3, 5) ;   a.affiche() ;
                a.deplace(2, 0) ;      a.affiche() ;
                 Point b = new Point() ;
                b.initialise (6, 8) ;  b.affiche() ;
        }
}
```

Exécution ➔

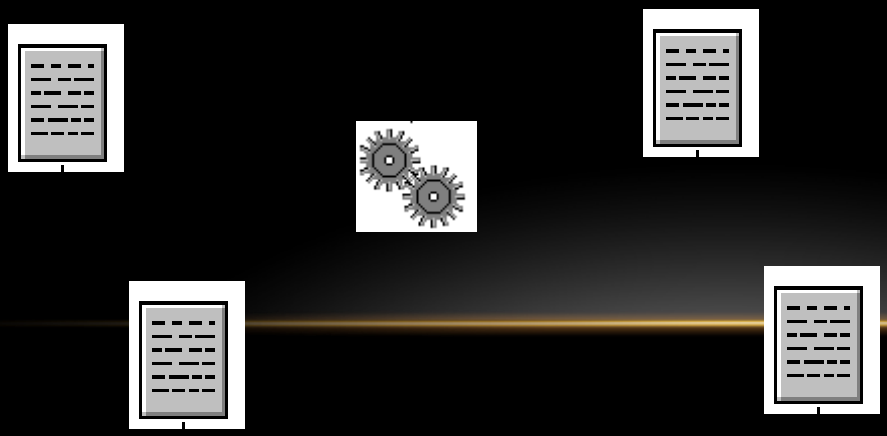
```
Je suis un point de coordonnées 3 5
Je suis un point de coordonnées 5 5
Je suis un point de coordonnées 6 8
```

LES CLASSES ET LES OBJETS

Mise en œuvre d'un programme comportant plusieurs classes

Un fichier source par classe

- Chaque classe est mise dans un fichier à part dans le même dossier :
  - Point.java et TestPoint.java
  - Compiler les fichier Point.java ➔ javac Point.java ➔ point.class
  - Compiler le fichier Testpoint.java ➔ javac TestPoint.java ➔ TestPoint.class
  - Lancer le fichier TestPoint.class ➔ java TestPoint



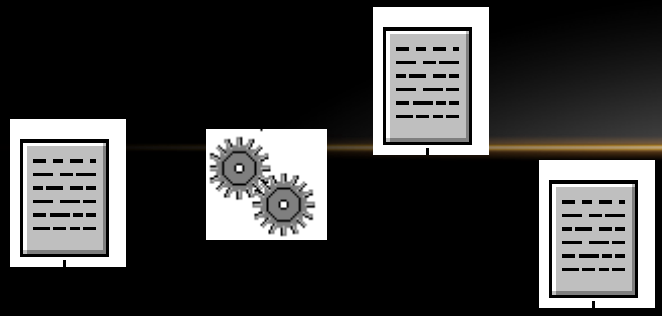
Plusieurs classes dans un seul fichier source

Accessibilité paquetage

Accessibilité JVM

```
class Point
{  public void initialise (int abs, int ord)
                        { x = abs ; y = ord ;
                         }
  public void deplace (int dx, int dy)
                         { x += dx ; y += dy ;
                        }
            public void affiche ()
       { System.out.println ("coordonnées " + x + " " + y) ;
            }
    private int x ;    // abscisse
    private int y ;    // ordonnée
}
```

```
public class TestPoint
{
            public static void main (String args[])
            {           Point a ;
                         a = new Point() ;
                        a.initialise(3, 5) ;   a.affiche() ;
                        a.deplace(2, 0) ;      a.affiche() ;
                         Point b = new Point() ;
                        b.initialise (6, 8) ;  b.affiche() ;
            }
}
```



LA NOTION DE CONSTRUCTEUR

Généralités

```
public class Point
{
        public void initialise (int abs, int ord)
            {       x = abs ;
                    y = ord ;
             }
        ……….
        ……..
}
```

Il est nécessaire de recourir à la méthode initialise pour attribuer des valeurs aux champs d'un objet de type Point.

La notion de constructeur permet d'automatiser le mécanisme d'initialisation d'un objet.

- Un constructeur n'est rien d'autre qu'une méthode, sans valeur de retour, portant le même nom que la classe.
- Un constructeur peut disposer d'un nombre quelconque d'arguments (éventuellement aucun).

LA NOTION DE CONSTRUCTEUR

Exemple de classe comportant un constructeur

```
public class Point
{
        public Point ( int  abs,  int  ord)
            {          x = abs ;
                       y = ord ;
             }
        ……….
        ……..
}
```

Comment utiliser cette classe ?

\\ Une instruction telle que :

Point a = new Point() ; \\ est refusée par le compilateur.

Ici, notre constructeur a besoin de deux arguments. Ceux-ci doivent obligatoirement être fournis lors de la création, par exemple :

Point a = new Point(1, 3) ;

LA NOTION DE CONSTRUCTEUR

Exemple de classe comportant un constructeur

```
public class Point
{
        public Point ( int  abs,  int  ord)
                {          x = abs ;
                           y = ord ;
                }
        ..................
        ..................
}
```

```
public class TstPnt3
{ public static void main (String args[ ])
  { Point a ;
    a = new Point(3, 5) ;
    a.affiche() ;
    a.deplace(2, 0) ;
    a.affiche() ;
    Point b = new Point(6, 8) ;
    b.affiche() ;
  }
}
```

LA NOTION DE CONSTRUCTEUR

Quelques règles concernant les constructeurs

1. Un constructeur ne fournit aucune valeur. Dans son en-tête, aucun type ne doit figurer devant son nom. Même la présence (logique) de void est une erreur

```
class Test
{ .....
  public void Test ()   // erreur de compilation : void interdit ici
  { .....
  }
}
```

2. Une classe peut ne disposer d'aucun constructeur

```
Point a =  new Point() ;   // OK si Point n'a pas de constructeur
```

LA NOTION DE CONSTRUCTEUR

Quelques règles concernant les constructeurs

4. Un constructeur ne peut pas être appelé directement depuis une autre méthode.
   Par exemple, si Point dispose d'un constructeur à deux arguments de type int:

```
Point a = new Point (3, 5) ;
   .....
a.Point(8, 3) ;  // interdit
```

LA NOTION DE CONSTRUCTEUR

Construction et initialisation d'un objet

la création d'un objet entraîne toujours, par ordre chronologique, les opérations suivantes :

- une initialisation par défaut de tous les champs de l'objet,
- une initialisation explicite lors de la déclaration du champ,
- l'exécution des instructions du corps du constructeur.

LA NOTION DE CONSTRUCTEUR

Construction et initialisation d'un objet

Initialisation par défaut des champs d'un objet

Type du champ	Valeur par défaut
boolean	false
char	caractère de code nul
entier (byte, short, int, long)	0
flottant (float, double)	0.0 ou 0.
objet	null

LA NOTION DE CONSTRUCTEUR

Construction et initialisation d'un objet

Initialisation explicite des champs d'un objet

```
class A
{  public A (...) { ..... }   //  constructeur de A
      .....
   private int n = 10 ;
   private int p ;
}
```

L'instruction suivante :

A a = new A (...) ;

entraîne successivement :

- l'initialisation (implicite) des champs n et p de a à 0,
- l'initialisation (explicite) du champ n à la valeur figurant dans sa déclaration, soit 10,
- l'exécution des instructions du constructeur.

LA NOTION DE CONSTRUCTEUR

Construction et initialisation d'un objet

```
public class Init
{ public static void main (String args[])
  { A a = new A() ;
    a.affiche() ;
  }
}
```

```
class A
{ public A()
  { np = n * p ;
    n = 5 ;
  }
  public void affiche()
  { System.out.println ("n = " + n + ", p = " + p + ", np = " + np) ;
  }
  private int n = 20, p = 10 ;
  private int np ;
}
```

Appel du constructeur

Donnez les valeurs affichées :

n = 5 , p = 10 , np = 200

TYPOLOGIE DES MÉTHODES D'UNE CLASSE

Parmi les différentes méthodes que comporte une classe, on a souvent tendance à distinguer :

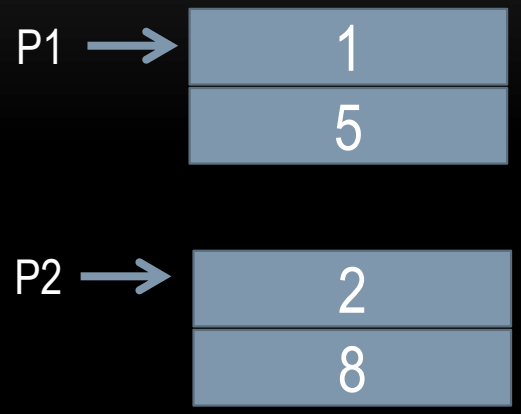
- les constructeurs ;
- les méthodes d'accès (en anglais accessor) qui fournissent des informations relatives à l'état d'un objet, c'est-à-dire aux valeurs de certains de ses champs (généralement privés), sans les modifier ;
- les méthodes d'altération (en anglais mutator) qui modifient l'état d'un objet, donc les valeurs de certains de ses champs.

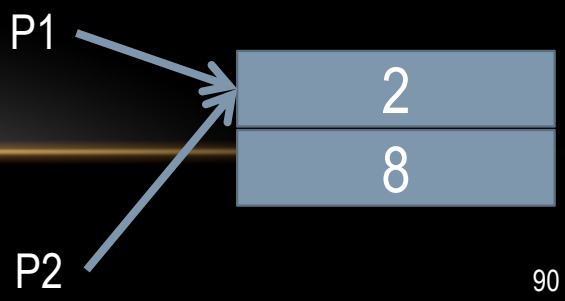
```
public int getX { return x ; }          // getX ou encore getAbscisse
public int getY { return y ; }          // getY ou encore getOrdonnee
public void setX (int abs) { x = abs ; }   // setX ou encore setAbscisse
public void setY (int ord) { x = ord ; }   // setY ou encore setOrdonnee
public void setPosition (int abs, int ord) { x = abs ; y = ord ; }
```

AFFECTATION ET COMPARAISON D'OBJETS

Exemple 1

```
// Soient 2 variables P1 et P2 de type point
Point  P1, P2;
//Soit 2 objets P1 et P2 de type point
P1 = new point(1, 5);
P2 = new point(2, 8);
//L'instruction
P1 = P2 ;
/* affecte à P1 la référence de P2.
Ainsi, P1 et P2 désigne le même objet point(2, 8) et non pas 2 objets de même valeur. */
```





RAMASSE-MIETTES

Garbage Collector

```
Point A;
A = new point(8,12);    // objet avec référence

(new Point(3,5)).affiche() ;    // Objet sans référence
```

candidat au ramasse-miettes

En Java, il n'existe aucun opérateur permettant de détruire un objet dont on n'aurait plus besoin.

En fait, la démarche employée par Java est un mécanisme de gestion automatique de la mémoire connu sous le nom de ramassse-miettes (en anglais Garbage Collector).

On peut activer le garbage collector par l'instruction :

```
System.gc();
```

RÈGLES D'ÉCRITURE DES MÉTHODES

Méthodes ne fournissant aucun résultat

Méthode avec le mot clé void dans son en-tête.
Méthode appelée: objet.méthode(liste arguments).

Méthodes fonction fournissant un résultat

```
public class Point
{           int getX { return x ; }
            int getY { return y ; }
              double distance ()
                        {  double d ;
                           d = Math.sqrt (x*x* + y*y) ;
                           return d ;
                        }
            private int x, y ;
            ..............
}
```

RÈGLES D'ÉCRITURE DES MÉTHODES

Utilisation d'une fonction fournissant un résultat

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

double u, r ;

   .....

u = 2. * a.distance() ;

r = Math.sqrt( a.getX() * a.getX() + a.getY() * a.getY() );
```

RÈGLES D'ÉCRITURE DES MÉTHODES

Les arguments d'une méthode

les arguments figurant dans l'en-tête de la définition d'une méthode se nomment arguments muets (ou encore arguments ou paramètres formels)

```
void f (final int n, double x)
{  .....
   n = 12 ;    // erreur de compilation
   x = 2.5 ;   // OK
   .....
}
```

Les arguments fournis lors de l'appel de la méthode portent quant à eux le nom d' arguments effectifs (ou encore paramètres effectifs).

```
Point p = new Point(...) ;
int n1, n2 ; byte b ; long q ;
p.deplace (n1, n2) ;     // OK : appel normal
p.deplace (b+3, n1) ;    // OK : b+3 est déjà de type int
p.deplace (b, n1) ;      // OK : b de type byte sera converti en int
p.deplace (n1, q) ;      // erreur : q de type long ne peut être converti en int
p.deplace (n1, (int)q) ; // OK
```

RÈGLES D'ÉCRITURE DES MÉTHODES

Propriétés des variables locales

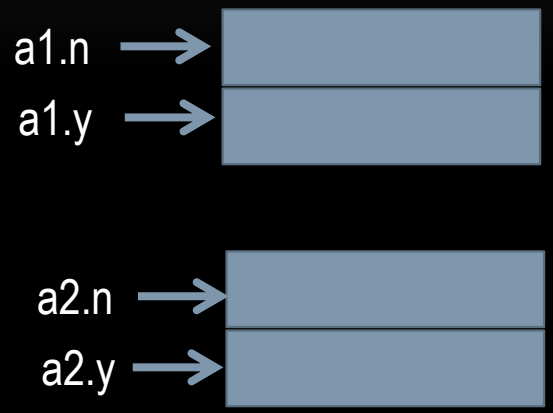
```
void F(int n)
{ float x ;      // variable locale à F
  float n ;      // interdit en Java
  .....
}
void G ()
{ double x ;    // variable locale à G, indépendante de x locale à F
    .....
}
```

RÈGLES D'ÉCRITURE DES MÉTHODES

CHAMPS STATIQUES (Champs de classe)

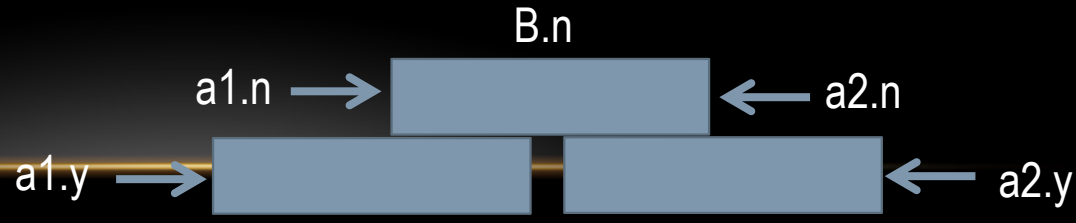
```
class A
{ int n ;
  float y ;
}
```

A a1 = new A(), a2 = new A() ;



Les champs statiques n'existent qu'en un seul exemplaire, indépendamment de tout objet de la classe.

```
class B
{ static int n ;
  float y ;
}
B a1 = new B(), a2 = new B() ;
```



RÈGLES D'ÉCRITURE DES MÉTHODES

CHAMPS STATIQUES

Exemple

```
class Obj {
public Obj() { System.out.print ("++ creation objet Obj ; ") ;
    nb ++ ;
    System.out.println ("il y en a maintenant " + nb) ;
  }
  private static long nb=0 ;
}
public class TstObj
{ public static void main (String args[])
  { Obj a ;
    System.out.println ("Main 1") ;
    a = new Obj() ;
    System.out.println ("Main 2") ;
    Obj b ;
    System.out.println ("Main 3") ;
    b = new Obj() ;
    Obj c = new Obj() ;
    System.out.println ("Main 4") ;
  }
}
```

Exécution

```
Main 1
++ creation objet Obj ; il y en a maintenant 1
Main 2
Main 3
++ creation objet Obj ; il y en a maintenant 2
++ creation objet Obj ; il y en a maintenant 3
Main 4
```

RÈGLES D'ÉCRITURE DES MÉTHODES

METHODES STATIQUES (Méthodes de classe)

une méthode de classe ne pourra en aucun cas agir sur des champs usuels (non statiques)

```
class A
{   .....
  private float x ;        // champ usuel
  private static int n ;   // champ de classe
    .....
  public static void f()   // méthode de classe
  { .....     // ici, on ne peut pas accéder à x, champ usuel,
    .....     //  mais on peut accéder au champ de classe n
  }
}
   .....
A a ;
A.f() ;    // appelle la méthode de classe f de la classe A
a.f() ;    // reste autorisé, mais déconseillé
```

METHODES STATIQUES (Méthodes de classe)

Exemple

```
class Obj
{ public Obj()
  { System.out.print ("++ creation objet Obj ; ") ;
    nb ++ ;
    System.out.println ("il y en a maintenant " + nb) ;
  }
  public static long nbObj ()
  { return nb ;  }
  private static long nb=0 ;
}
public class TstObj2
{ public static void main (String args[])
  { Obj a ;
    System.out.println ("Main 1 : nb objets = " + Obj.nbObj() ) ;
    a = new Obj() ;
    System.out.println ("Main 2 : nb objets = " + Obj.nbObj() ) ;
    Obj b ;
    System.out.println ("Main 3 : nb objets = " + Obj.nbObj() ) ;
    b = new Obj() ;
    Obj c = new Obj() ;
    System.out.println ("Main 4 : nb objets = " + Obj.nbObj() ) ;
  }
}
```

```
Main 1 : nb objets = 0
++ creation objet Obj ; il y en a maintenant 1
Main 2 : nb objets = 1
Main 3 : nb objets = 1
++ creation objet Obj ; il y en a maintenant 2
++ creation objet Obj ; il y en a maintenant 3
Main 4 : nb objets = 3
```

RÈGLES D'ÉCRITURE DES MÉTHODES

initialisation des champs statiques

```
class A
{    .....
   public static void f() ;
     .....
   private static int n = 10 ;
   private static int p ;
}
```

```
A a ;  // aucun objet de type A n'est encore créé, les champs statiques
       // de A sont intialisés : p (implicitement) à 0, n (explicitement) à 10
A.f() ;    // initialisation des statiques de A, si pas déjà fait
```

Bloc d'initialisation statique

```
class A
{ private static int t[] ;
  .....
  static { .....
           int nEl = Clavier.lireInt() ;
           t = new int[nEl] ;
           for (int i=0 ; i<nEl ; i++) t[i] = i ;
         }
  .....
}
```

SURDEFINITION DE MÉTHODES

```
class Point
{ public Point (int abs, int ord)  // constructeur
  { x = abs ;  y = ord ;
  }
public void deplace (int dx, int dy)  // deplace  (int, int)
  { x += dx ; y += dy ;
  }
  public void deplace (int dx)          // deplace  (int)
  { x += dx ;
  }
  public void deplace (short dx)        // deplace  (short)
  { x += dx ;
  }
  private int x, y ;
}
```

Rq :

Le type de la valeur de retour d'une méthode n'intervient pas dans le choix d'une méthode surdéfinie

```
public class Surdef1
{ public static void main (String args[])
  { Point a = new Point (1, 2) ;
    a.deplace (1, 3) ; // appelle deplace (int, int)
    a.deplace (2) ;    // appelle deplace (int)
    short p = 3 ;
    a.deplace (p) ;    // appelle deplace (short)
    byte b = 2 ;
    a.deplace (b) ;    // appelle deplace (short) apres
//conversion de b en short
  }
}
```

SURDEFINITION DE MÉTHODES

CAS DU CONSTRUCTEUR

```
class Point
{
  public Point ()          // constructeur 1 (sans argument)
  { x = 0 ;  y = 0 ;
  }
public Point (int abs)          // constructeur 2 (un argument)
  { x = y = abs ;
  }
  public Point (int abs, int ord ) // constructeur 3 (deux arguments)
  { x = abs ; y = ord ;
  }
  public void affiche ()
  { System.out.println ("Coordonnees : " + x + " " + y) ;
  }
  private int x, y ;
}
```

```
public class Surdef2
{ public static void main (String args[])
  { Point a = new Point () ;    // appelle constructeur 1
    a.affiche() ;
    Point b = new Point (5) ;   // appelle constructeur 2
    b.affiche() ;
    Point c = new Point (3, 9) ;  // appelle constructeur 3
    c.affiche() ;
  }
}
```

```
Coordonnees : 0 0
Coordonnees : 5 5
Coordonnees : 3 9
```