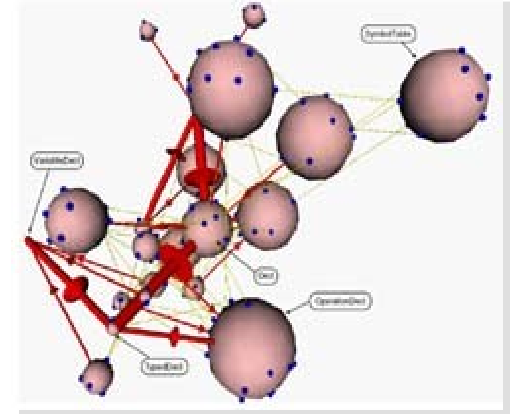
La programmation Orientée Objets et le langage Java

Belgacem BEN HEDIA
ESISAR-INP Grenoble

CITI - INSA Lyon
EPT



Cours 1: Java “sans les objets”

belgacem.ben-hedia@insa-lyon.fr
belgacem.ben-hedia@esisar.inpg.fr
http://www.bbhedia.org

A propos de l'Unité: La programmation Orientée Objets et le langage Java

- CS310 - programmation OO en Java: 10 CM, 4 TD,
- contrôle continu : lors de chacun des 6 TP, rendre un programme
  - ce doit être un programme correct (compilé, testé)
  - envoyer seulement le(s) fichier(s) source(s) (fichiers « .java »)
  - à l'adresse :
- Les cours « diapos » + polycopie de Garetta en ligne « prochainement »

http://www.bbhedia.org/index.php?n=Main.Enseignement

Ressources

- Documentation
  - polycopié de H. Garreta « Le langage Java »
  - documentations diverses (dont utilisation d'eclipse)
  - java.sun.com (logiciel, documentation, etc.)
- Logiciel nécessaire
  - JRE : machine java + bibliothèques, pour exécuter les programmes
  - JDK : compilateur et outils pour développer (contient un JRE)
  - éditeur quelconque (bloc-notes, vi, Jext, etc.)
  - ou, de préférence eclipse, EDI très puissant : www.eclipse.com

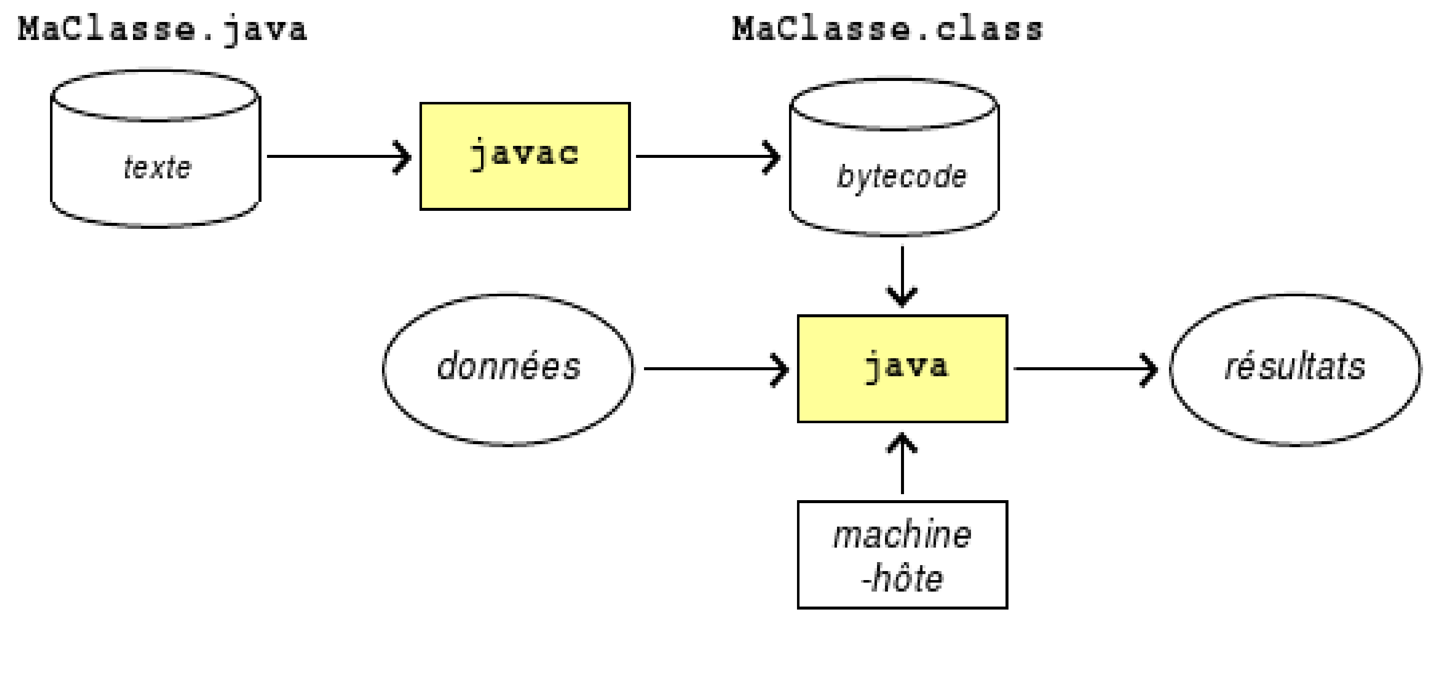
Le langage Java

- James Gosling, vers 1995 (débuts 1990) chez Sun Microsystems Inc.
- pour l'informatique répartie : portabilité, réutilisabilité, sécurité
  - langage orienté objets
  - neuf (aucune compatibilité à assurer)
  - bibliothèque importante (collections, interfaces graphiques, etc.)
  - langage compilé et interprété

Java est compilé et interpété

- compilation: traduction texte java → bytecode
- interprétation: exécution du bytecode par une machine virtuelle

Java (portabilité et sécurité)



Les deux commandes à connaître

- compilation d'un fichier source

  javac <nom de fichier>

  ex : javac Bonjour.java
- exécution d'une classe exécutable

  java <nom de classe>

  ex : java Bonjour

  il doit exister une classe *Bonjour* comportant une certaine méthode *main*
- avec eclipse tout cela devient transparent

L'inévitable « PRINT "HELLO" »

Une classe exécutable complète :

fichier Bonjour.java

```
public class Bonjour {
    public static void main(String[] args) {
        System.out.println("Bonsoir !") ;
    }
}
```

Essai :

```
>javac Bonjour.java
>java Bonjour
Bonsoir !
>
```

Deux erreurs qu'on peut faire

Une classe exécutable complète :

fichier Bonjour.java

```
public class Bonjour {
   public static void main() {
      System.out.println("Bonsoir !") ;
   }
}
```

Essai :

```
>javac Bonjour.java
>java Bonjour.class
Exception in thread "main"
   java.lang.NoClassDefFoundError : Bonjour/class
```

Deux erreurs qu'on peut faire

Une classe exécutable complète :

fichier Bonjour.java

```
public class Bonjour {
   public static void main() {
      System.out.println("Bonsoir !") ;
   }
}
```

Essai :

```
>javac Bonjour.java
>java Bonjour.class
Exception in thread "main"
   java.lang.NoClassDefFoundError : Bonjour/class
```

Une classe exécutable doit posséder une méthode ayant la *signature*

```
public static void main(String[] args)
```

Deux mise en garde...

- La programmation orienté objets (POO) a été développée pour lutter contre la complexité des grosses applications :
  - il est difficile d'en montrer l'intérêt sur de petits exemples
  - l'expérience et l'imagination de l'auditoire sont constamment sollicitées
- Les bibliothèques accompagnant les langages OO sont des usines à gaz dans lesquelles on se perd facilement :
  - recherches dans la documentation : véritables « jeux de pistes »
  - ne pas hésiter à recycler des programmes qui fonctionnent
  - il est nécessaire de beaucoup pratiquer

Approche du langage

- Parlant très (trop ?) simplement, la POO n'est que de la carrosserie :
  - packaging et gestion des droits d'accès aux éléments des programmes
  - mise en commun et réutilisation du code
- Qu'y a-t-il sous cette carrosserie ? essentiellement, le C :
  - même syntaxe de base
  - conservation de la grande majorité des concepts
- Nous connaissons déjà le langage C. Aujourd'hui, nous allons
  - rappeler rapidement les principaux éléments de le programmation en C
  - faire une toute petite approche de la notion de classe

Approche du langage

Toutes les variables doivent être déclarées :

Code erroné

```
float distance(x1, y1, x2, y2) {
   dx = x1 - x2 ;
   dy = y1 - y2 ;
   return Math.sqrt(dx * dx + dy * dy) ;
}
```

il fallait:

Code correcte

```
float distance(int x1, int y1, int x2, int y2) {
   int dx, dy ;
   dx = x1 - x2 ;
   dy = y1 - y2 ;
   return Math.sqrt(dx * dx + dy * dy) ;
}
```

Approche du langage

Toutes les variables doivent être déclarées :

Code erroné

```
float distance(x1, y1, x2, y2) {
   dx = x1 - x2 ;
   dy = y1 - y2 ;
   return Math.sqrt(dx * dx + dy * dy) ;
}
```

ou bien:

Code correcte

```
float distance(int x1, int y1, int x2, int y2) {

   int dx = x1 - x2 ;
   int dy = y1 - y2 ;
   return Math.sqrt(dx * dx + dy * dy) ;
}
```

Approche du langage

L'indentation et la fin de ligne n'ont pas de rôle syntaxique:

Code erroné

```
if (a <= b)
   min = a ;
   max = b ;
else
   min = b ;
   max = a ;
```

il faut écrire:

Code correcte

```
if (a <= b){
   min = a ;
   max = b ;
} else {
   min = b ;
   max = a ;
}
```

Approche du langage

Certaines instructions ont toujours un « ; » au bout, d'autres jamais :

- affectation

```
variable = expression ;
```

- appel de fonction sans resultat

```
fonction(arg1 , arg2 , . . . argk ) ;
```

- instruction conditionnelle

```
if (expression) instruction else instruction
if (expression) instruction
```

- boucle « tant que »

```
while (expression) instruction
```

- boucle « pour »

```
for (expr1 ; expr1 ; expr1 ) instruction
```

Approche du langage

Boucle « tant que »

```
while (expression) instruction
```

Exemple :etant donne u > 0, trouver le plus grand entier k tel que 2 k < u

extrai de code

```
...
p = 1 ;
k = 0 ;
while (p < u) {
   p = 2 * p ;
   k = k + 1 ;   /* ou k++ ; */
}
k = k - 1 ;    /* ou k-- ; */
...
```

Approche du langage

Boucle « pour »

```
for (expr1 ; expr2 ; expr3) instruction
```

– `expr1`: initialisation

– `expr2` : condition de continuation (comme dans while)

– `expr3`: passage à l'itération suivante

Exemple : somme des n éléments d’un tableau de nombres tab :

extrait de code

```
...
somme = 0 ;
for (int i = 0 ; i < n ; i++)
      somme = somme + tab[i] ;
...
```

Une toute petite approche de la notion de classe

- Aujourd'hui nous allons faire une présentation des classes sans parler d'objets,

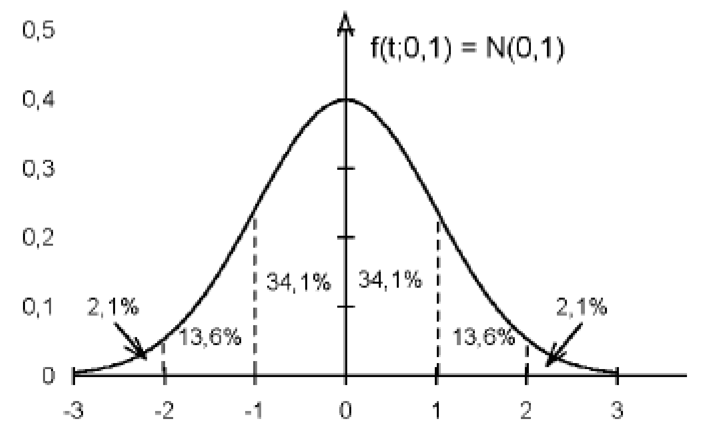
ou « comment faire du C en Java ? »

- Classe =
  - unité de compilation
  - moyen d'allonger les noms des variables et fonctions

    (pour éviter les conflits)
- Pratiquement, cela donne (pour aujourd'hui) :
  - écrire le code à l'intérieur d'une classe (ou plusieurs)
  - qualifier `static` toutes les variables et fonctions
  - qualifier `public` certains éléments imposes
  - faire confiance à C, sauf pour les tableaux

Exemple: simulation de loi normale

- loi uniforme
  - toutes les valeurs ont la même probabilité
  - la fonction `Math.random()` fait cela
- loi normale ou gaussienne
  - les valeurs centrales sont plus probables que les extrêmes



- comment la simuler sur ordinateur ? Il suffit de prendre (les ri sont des tirages uniformes, 0 <= ri <= 1) :

$$y = r1 + r2 + \cdots + r12 - 6$$

Exemple: simulation de loi normale

fichier `SimuLoiNormale.java`

```
public class SimuLoiNormale {
    static double alea() {
        double s = 0 ;
        for (int i = 0 ; i < 12 ; i++)
            s = s + Math.random() ;
        return s - 6 ;
    }
    static int nombre = 100 ;
    public static void main(String[] args) {
        double somme, somcarres ;
        for (int i = 0 ; i < nombre ; i++) {
            double x = alea() ;
            somme = somme + x ;
            somcarres = somcarres + x * x ;
        }
        double moyenne = somme / nombre ;
        double variance = somcarres / nombre - moyenne * moyenne ;
        System.out.println("moyenne " + moyenne) ;
        System.out.println("variance " + variance) ;
    }
}
```

Exemple: simulation de loi normale

Quels sont les membres de la classe `SimuLoiNormale` ?

fichier `SimuLoiNormale.java`

```
public class SimuLoiNormale {
    static double alea() {
        double s = 0 ;
        for (int i = 0 ; i < 12 ; i++)
            s = s + Math.random() ;
        return s - 6 ;
    }
    static int nombre = 100 ;
    public static void main(String[] args) {
        double somme, somcarres ;
        for (int i = 0 ; i < nombre ; i++) {
            double x = alea() ;
            somme = somme + x ;
            somcarres = somcarres + x * x ;
        }
        double moyenne = somme / nombre ;
        double variance = somcarres / nombre - moyenne * moyenne ;
        System.out.println("moyenne " + moyenne) ;
        System.out.println("variance " + variance) ;
    }
}
```

L'identification complète d'une méthode

- `alea` s'appelle en fait

  `SimuLoiNormale.alea()`

- à l'intérieur de sa classe on peut utiliser le nom court
- dans une autre classe il faut employer le nom long

  voyez l'appel de `random` : `Math.random()`

- deux méthode distinctes peuvent avoir le même nom :

  il suffit qu'elles appartiennent `a des classes distinctes

- il n'y a pas de problème à appeler `random` notre fonction :

  `SimuLoiNormale.random()` et `Math.random()`

  les noms complets sont différents

Surcharge des nom des méthodes

- deux méthodes distinctes peuvent avoir le même nom même si elles appartiennent à la même classe :

  il suffit qu’elles aient des signatures distinctes

```
class SimuLoiNormale {
    ...
    static double alea() { ... }
    ...
    static double alea(int k) { ... }
    ...
}
```

- appels de ces méthodes :

```
y = SimuLoiNormale.alea() ; /* la première */
...
y = SimuLoiNormale.alea(20) ; /* la deuxième */
```

Surcharge des nom des méthodes

- un autre exemple

```
class Math {
    static double abs(double x) { ... }
    static float abs(float x) { ... }
    static int abs(int x) { ... }
    ...
}
```

- emploi

```
y = Math.abs(x) ;
...
y = SimuLoiNormale.alea(20) ; /* la deuxième */
```

- encore un exemple

```
System.out.println("Bonjour") ;
System.out.println(unNombre) ;
```

Paquets (package)

- les classes sont un « rangement » des variables et méthodes
- les paquets sont un rangement des classes

```
package mesOutils ;
    class SimuLoiNormale {
        static double alea() {
            ...
        }
    }
```

- la classe s'appelle maintenant : `mesOutils.SimuLoiNormale`
- et la méthode : `mesOutils.SimuLoiNormale.alea()`
- but de tout cela : éviter les conflits de noms (réutilisabilite)

Paquets (package)

- les noms de paquet peuvent contenir « . » :

```
java.awt.event.MouseEvent
```

- Java n'y voit pas une hiérarchie (`java.awt.event` n'est pas un « sous-paquet » de `java.awt`)...
- ...mais les paquets correspondent à des repertoires : les classes de `java.awt.event` doivent etre dans `java/awt/event`
- `java.sun.com` recommande un système de nommage planétaire (cosmique ?) :

```
package fr.univ_mrs.dil.garreta.stats.mesOutils ;
class SimuLoiNormale {
...
```

Instruction `import`

- but : alléger les noms des classes lors de leur utilisation
- on écrit au début du fichier

```
import mesOutils.SimuLoiNormale ; // une classe
import java.awt.event.* ; // tout le paquet
```

- cela permet d'écrire `SimuLoiNormale.alea()`

  au lieu de `mesOutils.SimuLoiNormale.alea()`
- il ne s'agit pas de
  - designer un paquet que, sinon, le compilateur ne trouverait pas,
  - obtenir le droit d'acceder aux classes du paquetage

  mais uniquement de
  - pouvoir utiliser le nom court des classes du paquetage

Structure d’un fichier source Java

- un ou aucun énoncé `package`
- un nombre quelconque d’énoncé `import`
- une (préférable) ou plusieurs classes
- dont une seule est public, elle impose son nom au fichier
- partout ou on veut, des commentaires

fichier `MonCadre.java`

```
   /* auteur : Henri
* date : 20.10.06 */
package mesOutils ;
import java.awt.* ;
import java.awt.event.* ;
public class MonCadre {
le code de la classe est ici
}
```

Convention de nommage

- but : faciliter la lecture des programmes (réutilisabilité)
- noms faits de plusieurs mots : à partir du 2e, commencer chaque mot par une majuscule
- noms des classes : commencer par majuscule

```
class CadrePrincipalDeMonApplication { ...
```

- noms des methodes et variables : commencer par minuscule

```
int x, y ;
double vitesseDuVent ;
```

- paquetages : commencer chaqueelement par minuscule

```
import mesOutils.statistiques.* ;
```

- variables statiques finales : tout en majuscule, avec des «_»

```
static final int NOMBRE_DE_POINTS = 1000 ;
```

Types primitifs

- Nombres entiers

  `byte` : entier sur 8 bits (de −128 à +127)

  `short` : entier sur 16 bits (de −32768 à +32767)

  `int` : entier sur 32 bits

  `long` : entier sur 64 bits
- Nombres décimaux

  `float` : flottant IEEE sur 32 bits (7 chiffres significatifs)

  `double` : flottant IEEE sur 64 bits (15 chiffres significatifs)
- Valeurs booléennes

  `boolean : { false, true }`
- Caractères

  `char` : caractères *`Unicode`* sur 16 bits

Les tableaux du langage

- Tous les tableaux sont dynamiques (crées à l'exécution)
- tous les accès aux tableaux sont contrôles
- exemple : déclaration

```
int[] table ;
```

  (pour le moment, table vaut null)

- création :

```
table = new int[nombreElts] ;
```

- accès aux éléments du tableau :

```
table[i] = -1 ;
```

- un tel accès provoque toujours deux vérifications :
  - `table = null`, sinon `NullPointerException`
  - `0 ≤ i <nombreElts`, sinon `ArrayIndexOutOfBoundsException`

La taille des tableaux

- Exemple :

fichier `TestArgsMain.java`

```
public class TestArgsMain {
   public static void main(String[] args) {
      for (int i = 0 ; i < args.length ; i++)
         System.out.println(args[i]) ;
   }
}
```

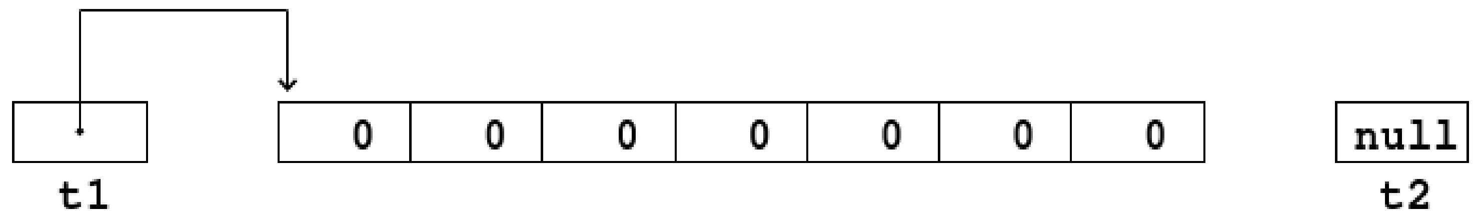
- Essai :

```
> javac TestArgsMain.java
> java TestArgsMain Combien vaut 2+3
Combien
vaut
2+3
>
```

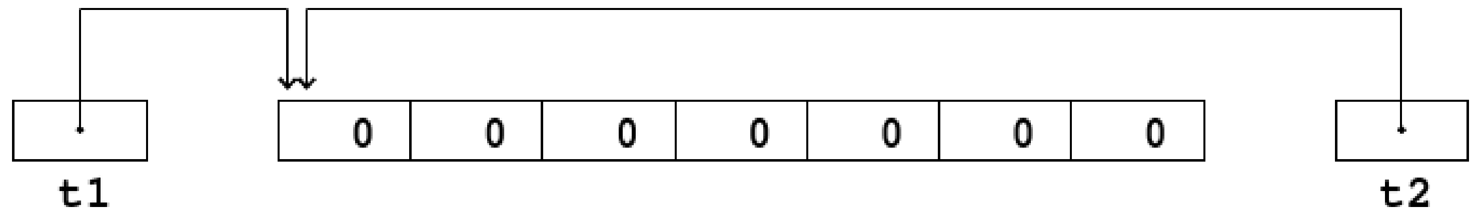
Une variable tableau est une référence

Déclaration : `int[] t1, t2;`

Création : `t1 = new int[7];`



Bien comprendre l'affectation : `t2 = t1;`



Tableaux à deux indices

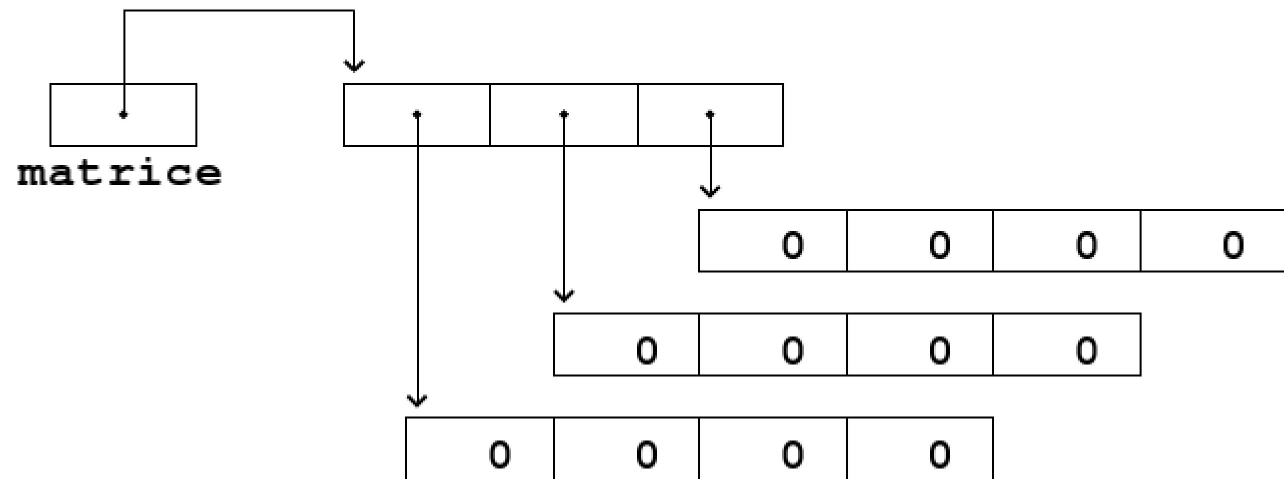
Déclaration

```
double[][] matrice ;
```

Création

```
matrice = new double[nl][nc] ;
```

Exemple (avec `nl = 3, nc = 4`):



Une variable tableau est une référence

On obtient le même résultat avec :

```
double[][] matrice ;
...
matrice = new double[nl][] ;
for (int i = 0 ; i < nl ; i++)
        matrice[i] = new double[nc] ;
```