INTRODUCTION AU LANGAGE JAVA

Moussa LO

UFR de Sciences Appliquées et de Technologie

Université Gaston Berger de Saint-Louis

Introduction à Java

- **Syntaxe de base**
- **Classes et objets**
- **Tableaux / Types énumérés / Classes enveloppes**
- **Packages**
- **Exceptions**
- **Interfaces graphiques**
- **Collections**
- **Persistence (JDBC, JPA)**

Bibliographie Java (1/2)

- **Le langage Java**, K.Arnold, J. Gosling, Thomson, 1996.
- **Java La synthèse – Des concepts objet aux architectures Web**, G. Clavel, et al., Dunod, 2000.
- **Introduction à la programmation objet en Java**, J. Brondeau, Dunod, 1999.
- **Le langage Java – Programmer par l'exemple**, T. Leduc, D. Leduc, Technip, 2000.
- **Algorithmique et programmation objet en Java**, V. Granet, Dunod, 2001.
- **Initiation à l'algorithmique objet**, A. Cardon, C. Dahancourt, Eyrolles, 2001.

Bibliographie Java (2/2)

- **Les Cahiers du Programmeur Java,** E. Pubaret, Eyrolles, 2004.
- **Le langage Java,** Irène Charon, Hermes, 2006**.**
- **Le poly de Java**, H. Garetta, Polycopié, Université de la Méditerranée.
- **Sites Java** :
  - http://www.oracle.com/technetwork/java/
  - http://www.java.com/
  - http://download.oracle.com/javase/tutorial/

GENERALITES

C'est quoi Java ?

- Langage orienté **objet** développé par Sun en 1990.
- Syntaxe proche de C++
- Java est **portable.**
- Grande richesse des APIs proposées : interfaces graphiques pour clients lourds et riches, bases de données, réseau, ...
- Beaucoup de technologies associées : Java EE, Java ME, Java Card, Java TV, Java 3D,
- Plate-forme évolutive :
  - Java 1.0, 1.1, Java 2, 1.3, 1.4, Java 5, Java 6, Java 7
  - Java Community Process (JCP - jcp.org/) : coordonne l'évolution du langage et des technologies associées

C++ / Java

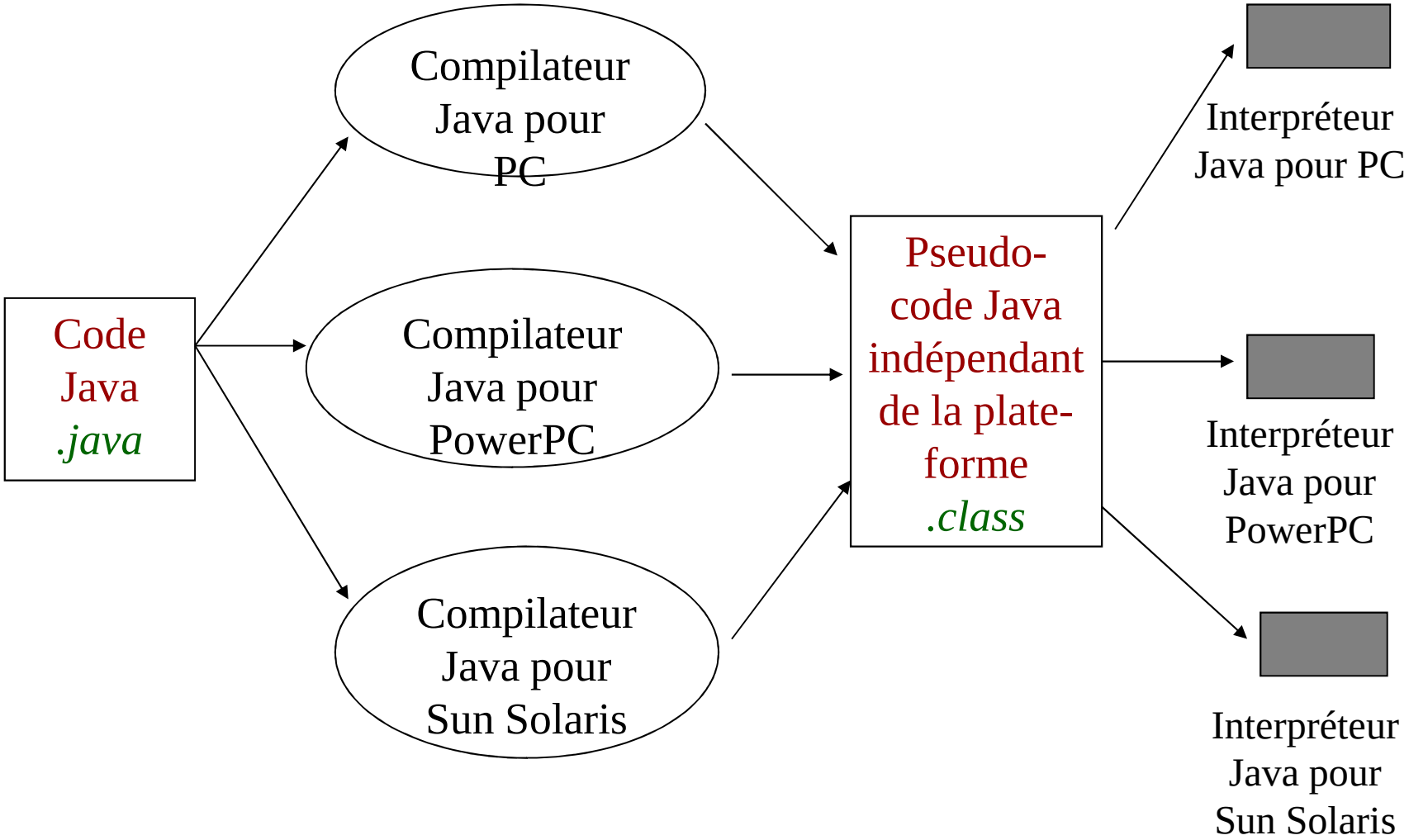
✓ C++ : langage **compilé**

- la compilation du source permet d'obtenir des instructions natives exécutables.
- le code exécutable obtenu est très performant

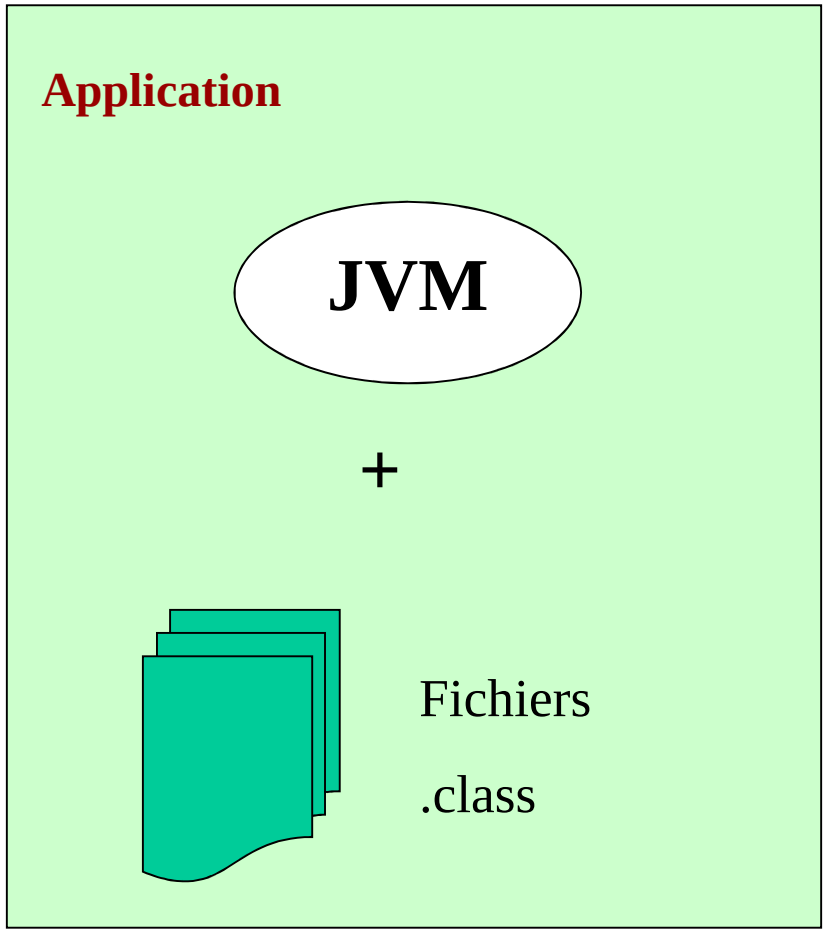
✓ Java : langage **interprété** qui utilise une **machine virtuelle**

- Machine virtuelle : simulation d'un ordinateur réel
- La "compilation" d'un source Java permet d'obtenir un code exécutable, le bytecode, par la machine virtuelle Java.
- La machine virtuelle Java est capable de comprendre les instructions contenu dans le bytecode et de les exécuter effectivement dans l'environnement réel courant.

Java est portable



Application Java



Kit de développement Java (1)

3 choix sont possibles :

✓ Choix 1 : Le **Java (Software) Development Kit (JDK)**

✓(http://www.oracle.com/technetwork/java/)

- référence pour développer en Java.
- comporte tous les outils indispensables à la réalisation et l'exécution d'un programme Java
- totalement gratuit
- ne comporte pas de véritable environnement de développement.

Kit de développement Java (2)

✓ Choix 2 : **Freeware ou shareware**

- offrent une interface graphique au JDK
- des outils supplémentaires (générateurs d'interface...)
- exemple : Eclipse, NetBeans

✓ Choix 3 : **Environnement de développement professionnel**

- très puissants
- très chers…
- exemples: *JBuilder, JCreator*

SYNTAXE DE BASE

Syntaxe de base : les commentaires

- Les commentaires tiennent sur une ligne :

```
// tout le reste de la ligne est un commentaire
```

- Les commentaires sont multilignes :

```
/* ceci est un commentaire
   tenant sur deux lignes
*/
```

- Les commentaires sont destinés au système ***javadoc*** pour la génération d'une documentation API à partir du code :

```
/** ceci est un commentaire spécial destiné au
système JavaDoc */
```

Syntaxe de base : les identificateurs

- Mêmes règles de formation des identificateurs qu'en C.
- Recommandations de SUN : Java Code Conventions (http://www.oracle.com/technetwork/java/codeconv-138413.html)
  - ✓ Le **nom d'une classe ou d'une interface** est fait d'un mot ou de la concaténation de plusieurs mots.

    Chacun commence par une majuscule : Etudiant, EtudiantBoursier, etc.

  - ✓ Les **noms des méthodes et des variables** commencent par une minuscule.

    Lorsqu'ils sont formés d'une concaténation de mots, chacun, sauf le premier, commence par une majuscule : afficher, toString(), nom, numeroInscription, etc.

  - ✓ Les **constantes de classe** (variables static final) sont écrites entièrement en majuscules : Integer.MAX_VALUE

Syntaxe de base : les types de données primitifs

- **`byte`** (8 bits)
- **`short`** (16 bits)
- **`int`** (32 bits)
- **`long`** (64 bits)
- **`float`** (32 bits)
- **`double`** (64 bits)
- **`char`** (16 bits)

✓ ***Possibilité de faire du casting comme en C***

```
double d; int i;
i = (int) d; // troncation de d en un entier
```

Syntaxe de base : L'instruction IF

Syntaxe:

```
if (condition)
   {
   instruction1
  }
   else
   {
   instruction2
   }
```

Exemple:

```
If ( x > 0)
{
System.out.println("positif ");
    x-- ;
}
else
System.out.print("négatif ");
```

Syntaxe de base : L'instruction FOR

Syntaxe:

```
for (initialisation;  condition; expression)
  {
      instructions
  }
```

Exemple:

```
for (int i=0 ; i<5 ; i++)
    System.out.print("Salut!");
```

Syntaxe de base : L'instruction While

Syntaxe:

```
while (condition)
   {
   instructions
   }

do
   {
   instructions
   }
while (condition);
```

Exemple:

```
int i = 10 ;
while ( i>0)
{
     System.out.println(i);
     i-- ;
}
```

Syntaxe de base : L'instruction Switch

Syntaxe:

```
switch (expression)
    {
    case valeur1:
    instructions; break;
    case valeur2:
    instructions; break;
    ...
    default:  instructions
     }
```

Exemple:

```
switch (i) {
    case 10 : System.out.println("x
          égale 10 ") ; break ;
case 100 : System.out.println("x
    égale 100 ") ; break ;
default : System.out.println("x est
    différent de 10 et de 100 ") ;
 }
```

CLASSES ET OBJETS EN JAVA

Définition d'une classe

```
public class NomDeLaClasse {
        déclarations des champs (attributs, propriétés)
        déclarations des méthodes
}

public class Fraction {
        // champs
        public int num ;
        public  int den ;
        // méthodes
        public void afficher () {
                if  (this.num % this.den == 0)
                        System.out.println(this.num/this.den ) ;
                else
                        System.out.println(this.num+"/"+this.den) ;

        }
}
```

Le désignateur (ou auto-référence) this

- **this** est une pseudo-variable qui permet :
  - ✓ de faire référence à l'objet courant (celui qu'on est en train de définir).
  - ✓ ou de désigner ses attributs ou ses méthodes.

- this est généralement utilisé pour lever l'ambiguité sur un identificateur lorsque le paramètre d'une méthode porte le nom de l'attribut de l'objet qu'il est chargé de modifier.

- *Exemple :*

```
void fixerNumerateur (int num) {
        this.num = num ;
}
```

Utilisation d'une classe

```
public class FractionUtil {
        public static void main (String[] args) {
                Fraction f = new Fraction();
                f.num = 8; f.den = 5;
                f.afficher();
        }
}
```

• Instanciation avec l'opérateur *new*

• Java s'occupe de la destruction des objets non utilisés (mécanisme de *garbage collector*).

➔ pas besoin de "destructeurs".

Exemple d'application Java avec le JDK

• Le fichier source prend le nom de la classe qu'il définit avec l'extension ***.java***

==> *FractionUtil.java*.

• La compilation génère un fichier binaire Java avec l'extension ***.class***.

javac FractionUtil.java

==> *FractionUtil.class*

• Ce fichier peut être exécuté grâce à un interpréteur java.

java FractionUtil

Constructeurs spécifiques

```
public class Fraction {
        public int num, den ;

        public Fraction (int n, int d) { this.num=n; this.den=d;}
        public Fraction (int n) { this.num=n; this.den=1;}

        public Fraction inverser() {
                if  (this.num == 0) return null;
                return new Fraction(this.den,this.num);
        }
}
```

• Les divers constructeurs doivent avoir des signatures différentes (*nombre d'arguments ou types des arguments différents*).

La méthode main(1)

• Premier sous-programme exécuté au lancement d'une application Java.

• Signature:

```
public static void main (String[] args) {
    ...
}
```

• Le paramètre est un tableau de chaînes utilisé pour les arguments transmis à partir de la ligne de commande.

La méthode main(2)

Exemple avec arguments:

```
public class Somme
{
        public static void main(String[] args)
        {
         int  x = Integer.parseInt(args[0]);
         int  y = Integer.parseInt(args[1]);
        System.out.println("la somme de "+x+" et "+y+" est:
    "+(x+y));
        }
    }
```

Utilisation:

java Somme 12 14

==> la somme de 12 et 14 est: 26

Exercice 1

Arguments de la ligne de commande

Écrire une application Java qui montre chaque argument qu'elle reçoit sur la ligne de commande.

Chacun des arguments doit être affiché sur une nouvelle ligne, ainsi que sa longueur en caractères.

A la fin, l'application affichera la somme de ces longueurs.

Travaux pratiques

1. *Installer le JDK*
2. *MAJ les variables d'environnement JAVA_HOME et PATH*
3. *Installer un éditeur (Notepad++ par exemple)*
4. *Reprendre l'exercice 1*

Exercice 1 avec des boîtes de dialogues …

```
public static void main(String arg[]){
        String ch = null;
        int sommeLongueurs = 0;
        do {
    s = JOptionPane.showInputDialog("Entrez une chaîne: ");
    JOptionPane.showMessageDialog(null, ch + ": "+ ch.length());
    sommeLongueurs = sommeLongueurs + ch.length();
    int reponse = JOptionPane.showConfirmDialog(null, "Voulez-vous
quitter ?");
        } while (reponse == JOptionPane.NO_OPTION);
        JOptionPane.showMessageDialog(null, " La somme des longueurs
est : "+ sommeLongueurs);
}
```

Entrées/Sorties simplifiées

La méthode printf

• La classe ***java.io.PrintStream*** contient une méthode ***printf*** qui fonctionne presque de la même façon qu'en C.

• ***Exemple*** :

```
int degre = 28;

System.out.printf("Il fait %d degres\n", degre);

// %d indicateur de format pour des entiers

System.out.printf("Le %2$s %1$s", "Java", "langage");

// le 2$ du %2$s indique qu'il s'agit d'ecrire le 2nd paramètre de susbstitution

// le s indique qu'il s'agit d'une chaîne de caractères.
```

• Comme en C on utilise des indicateurs de format commençant par le symbole %. (Consulter la doc de la classe ***java.util.Formatter)***

La classe java.util.Scanner

- La classe **Scanner** permet de lire un flux de données formatées.
- *Exemple*

```
public static void main(String arg[]){
        Scanner entree = new Scanner(System.in);
        System.out.println("Donner votre prenom et votre nom:");
        String prenom = entree.next(); String nom = entree.next();
        System.out.println("Donner votre age:");
        int age = entree.nextInt();
        entree.nextLine();
        System.out.println("Donner une phrase:");
        String phrase = entree.nextLine();
        System.out.printf("%s %s, %d ans, dit %s\n", prenom, nom, age, phrase);
    }
```

La classe java.util.Scanner

• Offre des méthodes pour **lire** un **flux de données** formatées :

☞ **int nextInt()** : lecture de la prochaine valeur de type int.

☞ **long nextLong()** : lecture de la prochaine valeur de type long.

☞ **float nextFloat()** : lecture de la prochaine valeur de type float.

☞ **double nextDouble()** : lecture de la prochaine valeur de type double.

☞ **String nextLine()** : lecture de tous les caractères se présentant dans le flux d'entrée jusqu'à une marque de fin de ligne et renvoi de la chaîne ainsi construite. La marque de fin de ligne est consommée, mais n'est pas incorporée à la chaîne produite.

La classe Scanner

• Offre aussi toute une série de **méthodes booléennes** pour tester le type de la prochaine donnée disponible, sans lire cette dernière :

✓ **boolean hasNext()** : Y a-t-il une donnée disponible pour la lecture

✓ **boolean hasNextLine()** : Y a-t-il une ligne disponible pour la lecture

✓ **boolean hasNextInt()** : La prochaine donnée à lire est-elle de type int

✓ **boolean hasNextLong()** : La prochaine donnée à lire est-elle de type long

✓ **boolean hasNextFloat()** : La prochaine donnée `à lire est-elle de type float

✓ **boolean hasNextDouble()** : La prochaine donn´ee `a lire est-elle de type double

Visibilité, Variables, Tableaux

Visibilité des champs et méthodes

- Le principe d'**encapsulation** permet d'introduire celui de protection des attributs et méthodes.

- **Les Modificateurs de visibilité en Java :**
  - **public** : accessible par toutes les classes. Hérité par les sous classes.
  - **private** : accessible que par les seules méthodes de sa classe. Non hérité.
  - **protected** : accessible par les classes du même *package*. Hérité par les sous classes.
  - ***par défaut* :** accessible par les classes du même *package*. Hérité par les sous classes que si elles se trouvent dans le même package.

Visiblité : exemple

```
public class Etudiant {
        public String nom;
        private  String numIns;

        public Etudiant(String nom) {
                this.nom = nom;
        }

        // methodes accesseurs
        public String getNumIns() { return  this. numIns; }

        public void setNumIns(String numIns) {
                if (numIns==null || numIns.length()!=11) return;
                this.numIns = numIns;
        }

        public void afficher(){
                System.out.println("etudiant :"+ nom);
        }
}
```

Les variables (1)

- **variables d'instance** : elles déterminent les attributs ou l'état d'un objet donné (*attributs d'instance*)
- **variables de classe** : elles déterminent les attributs ou l'état d'une classe donnée (*attributs de classe*)
- **variables locales** : déclarées et utilisées dans les définitions de méthodes.

Syntaxe:

```
Type_de_la_variable identificateur_de_la_variable;
```

Exemples:

```
int annee;

String nom, prenom, adresse;
```

Les variables (2)

```
class Une_Classe {
        ...
        déclarations des variables de classe et d'instance
        ...
        public void une_Methode (déclarations des
        paramètres) {
                ...
                déclarations des variables locales
        }
}
```

Références et objets

• Un **objet** doit être considéré comme accessible par **référence**. L'**identificateur** d'un objet désigne une **référence** à l'objet et non l'objet lui-même.

• Ce sont ces **références** qui sont affectées ou passées comme arguments lors des appels de méthodes.

• *Exemple:*

```
Fraction f1, f2;
f1 = new Fraction();
f2 = f1;
```

f1 et *f2* sont des variables qui se réfèrent au ***même objet***.

Cet objet existe en un ***seul exemplaire*** mais avec ***deux références***.

Les membres de classe

- ✓ **Les attributs ou méthodes sont accessibles:**
  - soit via une instance de la classe: membres d'instance
  - soit via la classe elle-même: membres de classe ou membres statiques.

- ✓ **Les membres de classe :**
  - sont introduits par le mot **static**
  - existent en un seul exemplaire
  - sont partagés par toutes les instances de la classe.

- ✓ **Ils ne peuvent être invoqués que sur la classe.**

Attributs de classe

- Ils sont initialisés au moment du chargement de la classe.
- Ils sont modifiables par tous les objets de la classe.
- Toute modification est répercutée au niveau des autres objets.
- Ils permettent d'éviter la duplication de certains types d'attributs (comme les constantes) dans chaque instance.

Exemple

L'attribut **out** est un attribut de la classe java.lang.System

```
System.out.println(" Salut! ");
```

Les méthodes de classe

- Elles ne peuvent pas modifier des attributs d'instance.
- Lorsqu'une méthode ne manipule que des attributs (ou méthodes) de classe, elle doit être déclarée statique.

Exemples

- exit() est une méthode statique de la classe java.lang.System

```
System.exit();
```

- sqrt() est une méthode statique de la classe java.lang.Math

```
System.out.println(Math.sqrt(100));
```

Les tableaux (1)

✓Les tableaux se comportent un peu comme des objets.

✓Deux syntaxes pour la déclaration:

TypeDesElements[ ] NomDuTableau;

TypeDesElements NomDuTableau[ ];

✓*E*xemple :

int[ ] tab_entiers; *ou* int tab_entiers[ ];
String[ ] tab_chaines;

✓ Le type des éléments du tableau peut être une classe existante.

Les tableaux (2)

- ***Instanciation***

TypeDesElements[ ] NomDuTableau =

new TypeDesElements[TailleDuTableau];

Exemple

```
int[ ] tab_entiers  = new int[15];
String[ ] chaines = new String[3];
```

- *Matrices*

```
TypeDesElements[ ][ ] NomDuTableau;
TypeDesElements NomDuTableau[ ][ ];
```

Exemple

```
int [ ][ ] matrice
ou int matrice [ ][ ];
```

Les tableaux (3)

Exemple de saisie d'un tableau:

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

    int[] t = new int[5];

    Scanner entree = new Scanner(System.in);

    for (int i=0; i<5; i++){
        t[i] = entree.nextInt();
        entree.nextLine();}
}
```

Types énumérés

Déclaration d'un type énuméré (1/2)

- qualifieurs enum identif { identif, identif, ... identif }
- *Exemples :*

✓ **public enum PointCardinal { NORD, SUD, EST, OUEST }**

✓ **public enum JourSemaine { LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI, DIMANCHE }**

☞ Ces déclarations créent 2 nouveaux types publics PointCardinal et JourSemaine.

☞ Elles doivent être écrites dans 2 fichiers séparés, respectivement nommés *PointCardinal.java* et *JourSemaine.java*.

Déclaration d'un type énuméré (2/2)

La déclaration :

```
public enum PointCardinal { NORD, SUD, EST, OUEST }
```

est presque identique à :

```
public class PointCardinal {
    public static final PointCardinal NORD = new PointCard();
    public static final PointCardinal SUD = new PointCard();
    public static final PointCardinal EST = new PointCard();
    public static final PointCardinal OUEST = new PointCard();
}
```

☞ Un type énuméré est donc une classe dont les seules instances sont les valeurs du type énuméré.

La classe java.lang.Enum

• En fait, tout type énuméré est sous-classe de la classe **java.lang.Enum**.

• **java.lang.Enum** introduit quelques méthodes :
String name(), String toString()

Exemple :

```
public enum PointCardinal {
        NORD, SUD, EST, OUEST;
        public String toString() {
            return "<" + super.toString().toLowerCase() + ">";
        }
}
```

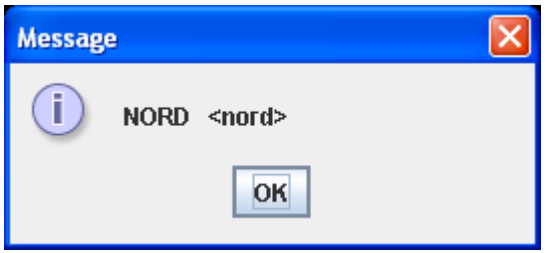
Type énuméré et switch

Une expression de type *enum* peut piloter un *switch* :

```
PointCardinal direction;
...
switch(direction) {
        case NORD:
            System.out.print(PointCardinal.NORD.name());
            break;
        case SUD:
            System.out.print(PointCardinal.SUD.name());
    break;
        …
        default:
            ...
}
```

Type énuméré : Exemple

```
public enum PointCardinal {

        NORD, SUD, EST, OUEST ;

        public String toString() {
                return "<" + super.toString().toLowerCase() + ">";
        }

        public static void main(String arg[]){
                PointCardinal direction = PointCardinal.NORD;

        javax.swing.JOptionPane.showMessageDialog(null,direction.name(
    ) + "   "+ direction);
        }

}
```



Emballage/Deballage
Classes enveloppes

Emballage et Deballage pour les classes enveloppes

• Les données de types primitifs (byte, short, int, long, float, double, char et boolean), ne sont pas des objets en Java.

• A chaque type primitif, correspond une classe dite «**classe enveloppe**» : Byte, Short, Integer, Long, Float, Double, Character et Boolean.

• Chaque instance d'une de ces classes se compose d'une unique donnée membre qui est une valeur du type primitif que l'instance enveloppe.

Emballage et Deballage pour les classes enveloppes

☞ **Emballage** : opération consistant à construire un objet enveloppant une valeur ***v*** d'un type primitif

```
int v;
...
// emballage de v
Integer unInteger = new Integer(v);
```

☞ **Déballage** : opération inverse.

```
...
// déballage de v
v = unInteger.intValue();
```

Emballage et Deballage pour les classes enveloppes

• Java (depuis la version 5) permet de confondre dans l'écriture une **variable** et l'**objet** du type enveloppe associé :

- ✓ là où un **objet** est attendu on peut mettre un **type primitif**
- ✓ là où un **type primitif** est attendu on peut mettre un **objet**

• Le compilateur déduit le type de la variable (par exemple *int* ou *Integer*) selon le contexte.

Emballage et Deballage pour les classes enveloppes

Exemple:

Integer n = 1;	Integer n = new Integer(1);
n = n + 5;	n est considéré comme un int
String s = n.toString();	n est considéré comme un objet de type Integer
System.out.println(s);	
if (n == 10) …	if (n.intValue() == 10) …

La boucle “for each”

La boucle "for each"

✓ « Nouvelle (*Java 5*) » syntaxe de la boucle ***for*** pour parcourir les éléments d'un tableau ou d'une collection.

Exemple 1:

```
// somme des elements d'un tableau
double[ ] t = {3.2, 5.3, 6.2};
double som = 0;
for (double d : t)
        som += d;
// d va prendre successivement les valeurs contenues dans le tableau
```

La boucle “for each”

Exemple 2: affichage d’une matrice.

```
double[][] matrice = new double[NL][NC];
...
for (double[] ligne : matrice) {
    for (double x : ligne)
            System.out.print(x + " ");
    System.out.println();
}
```

PACKAGES

Les packages : définition et utilité

Définition

• Un package est un ensemble de classes et d'autres packages regroupés sous un nom.

• C'est l'adaptation du concept de librairie ou de bibliothèque.

Utilité

• Servent à structurer l'ensemble des classes et interfaces.

• Augmentent la lisibilité des applications en structurant l'ensemble des classes selon une arborescence.

•Faciltent la recherche de l'emplacement physique des classes

• Empêchent la confusion entre des classes de même nom

• etc.

Les packages : généralités

• Lors de la définition d'un fichier source Java, on peut préciser son appartenance à un package en utilisant le mot-clé **package**:

```
package mon_package;
```

• Toutes les classes définies dans ce fichier font ainsi partie du package mon_package.

exemple: `package formesGeo;`

• Si la classe Carre se trouve dans un package formesGeo,

- son nom complet est formesGeo.Carre;
- le fichier Carre.class doit être placé dans un répertoire nommé formesGeo.

Les packages : importation

• Pour pouvoir utiliser une classe d'un package dans un autre fichier, il faut l'importer :

```
import nom_du_package.nom_classe;
import formesGeo.Carre;
```

• On peut importer toutes les classes d'un package en même temps:

```
import nom_du_package.*;
import java.io.*;
```

• Le package java.lang qui gère les types de données et les éléments de base du langage est automatiquement importé.

Les packages : compilation

- Pour compiler une classe contenant l'instruction suivante, il faut indiquer le chemin au compilateur.

```
import monPackage.ClasseImportee;
```

- *soit* **: Option –*classpath* de javac**

javac –classpath *chemin_du_fichier* MaClasse.java

- *soit :* **Variable d'environnement CLASSPATH**

doit contenir le chemin d'accès au répertoire racine du package.

Les packages : exécution

Pour exécuter la commande *java* monPackage.MaClasse :

- *soit* : Se placer dans le répertoire contenant le répertoire monPackage
- *soit* **: Utiliser l'option –classpath de la commande java**
- *soit* **: Variable d'environnement CLASSPATH**
  doit contenir le chemin d'accès au dossier monPackage.

Les packages prédéfinis (1/2)

- Les API Java sont organisées en deux parties :
  - **Java Core API** : API de base implantée dans tout interpréteur Java. Ces classes appartiennent au package "**java**".
  - **Java Standard Extension API** : extensions normalisées au travers d'interfaces et implantées par les éditeurs.

    Ces classes appartiennent aux packages "**javax**" et "**org**".

- Documentation sur toutes l'API Java disponible sur le site officiel du langage :

  pour la version 6 :

  http://download.oracle.com/javase/6/docs/api/

Les packages prédéfinis (2/2)

Les classes sont regroupées par thèmes dans des packages. Exemples :

- **java.lang** : types de données et éléments de base du langage (classes Object, String, Boolean, Integer, Float, Math, System (fonctions système), Runtime (mémoire, processus), etc.)
- **java.util**: structures de données et utilitaires (Dates, Collections, etc.)
- **java.io**: bibliothèque des entrées / sorties, fichiers.
- **java.net**: bibliothèque réseau
- **java.awt**: librairie graphique
- **java.applet**: librairie des applets.
- **java.security**: sécurité (contrôle d'accès, etc.)
- **java.rmi**: applications distribuées.
- **java.sql**: accèes aux BD (JDBC)

*Exercice : **Calculatrice***

Écrire une classe Java Calcul qui effectue des opérations arithmétiques simples (+,-,x,/) sur deux entiers et retourne le résultat.

a. Les arguments seront pris sur la ligne de commande (noter que les blancs entre les nombres et l'opérateur sont nécessaires).

b. Reprendre avec des boîtes de dialogue.

Exemple d'exécution :

java Calcul 3 + 2

5

Exercice : *Facture simple*

Pour calculer rapidement le montant de factures pour les clients d'une boutique, on demande d'écrire un programme Java qui permet de :

1. lire, à partir d'arguments fournis via la ligne de commande, le prénom et le nom du client et une liste de nombres réels représentant les montants des produits achetés ;
2. calculer et d'afficher le montant total de la facture.

Exemple d'utilisation du programme :

java Facture Fatou Cisse 1200 3587.6 1665 895.50

A payer par Fatou Cisse : 7348,1