

Chapitre 1: Langage JAVA

A series of horizontal lines in teal and light blue colors, with varying lengths and offsets, creating a modern, layered effect below the title.

Le Langage JAVA

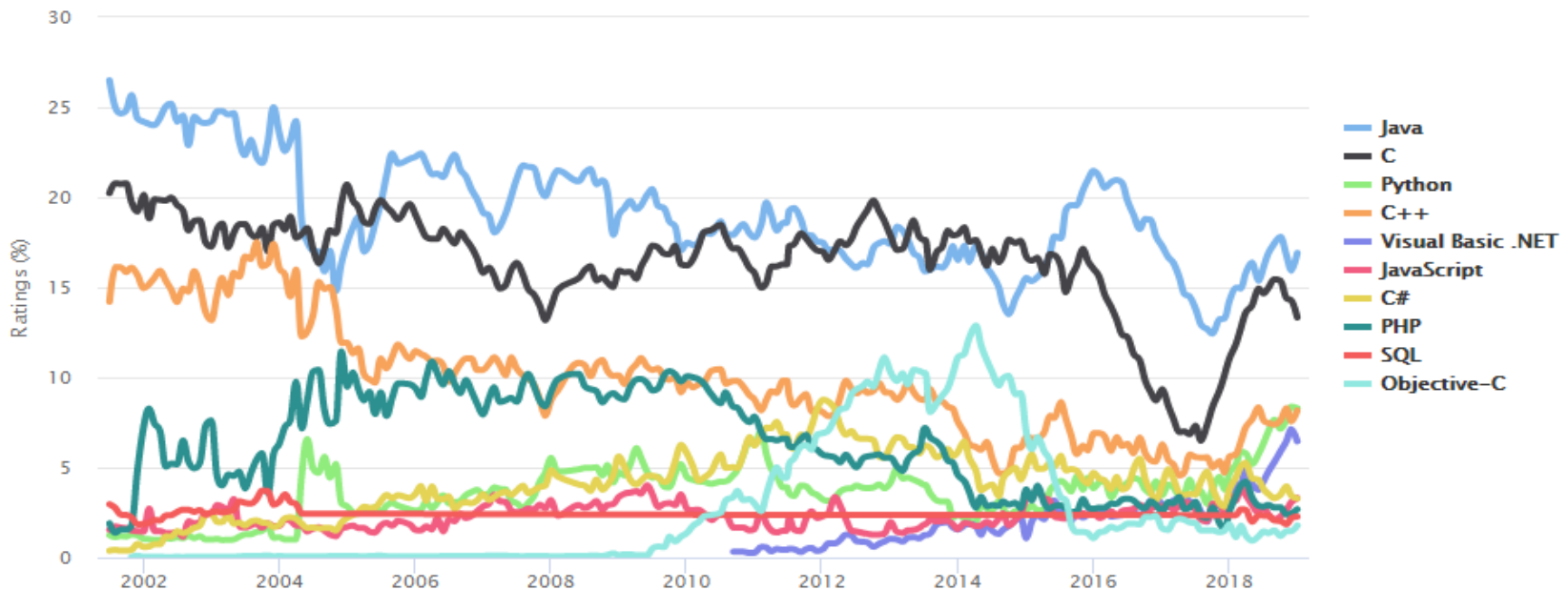
Java c'est quoi?

- Un langage : Orienté objet fortement typé avec classes
 - Un environnement d'exécution (JRE) : Une machine virtuelle et un ensemble de bibliothèques
 - Un environnement de développement (JDK) : Un compilateur et un ensemble d'outils
- La plate-forme et le langage Java sont issus d'un projet de Sun Microsystems datant de 1990.
- Généralement, on attribue sa paternité à trois de ses ingénieurs :
 - James Gosling
 - Patrick Naughton
 - Mike Sheridan
- Une technologie développée par SUN MicrosystemsTM lancée en 1995 - rachetée par Oracle en 2009.

Java c'est quoi?

- Java est devenu aujourd'hui l'un des langages de programmation les plus utilisés.
- Il est incontournable dans plusieurs domaines :
 - **Systèmes dynamiques** : Chargement dynamique de classes
 - **Internet** : Les *Applets* java
 - **Systèmes communicants** : *RMI, Corba, EJB*, etc.

Source: www.tiobe.com



Le Langage JAVA

Java en quelques mots:

- Dans un des premiers papiers* sur le langage JAVA, SUN le décrit comme suit :

«Java : a simple, object-oriented, distributed, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language»

- *White Paper :The Java Language Environment-James Gosling, Henry McGilton -May 1996
- <http://java.sun.com/docs/white/langenv>

Le Langage JAVA

Java en quelques mots:

- **Simple :**
 - Il est toutefois allégé de toutes les sources d'erreurs du C++ (pointeurs, gestion mémoire, etc.) et des fonctionnalités "tordues" et peu utilisées.
- **Objet :**
 - Inspiré du C++, Objective C et Smalltalk , Java est un langage fortement objet.
- **Distribué :**
 - Java contient un ensemble de classes permettant d'utiliser des protocoles TCP/IP tels que HTTP et FTP. L'utilisation des URL se fait aussi "simplement" que l'ouverture de fichiers locaux.
- **Robuste :**
 - Le maximum de vérifications sont faites à la compilation (protection des classes,...).
 - Pas de gestion explicite de la mémoire (GC en thread), pas de pointeur, ..
- **Sécurisé :**
 - Java est très sécurisé (SecurityManager).

Présentation du langage

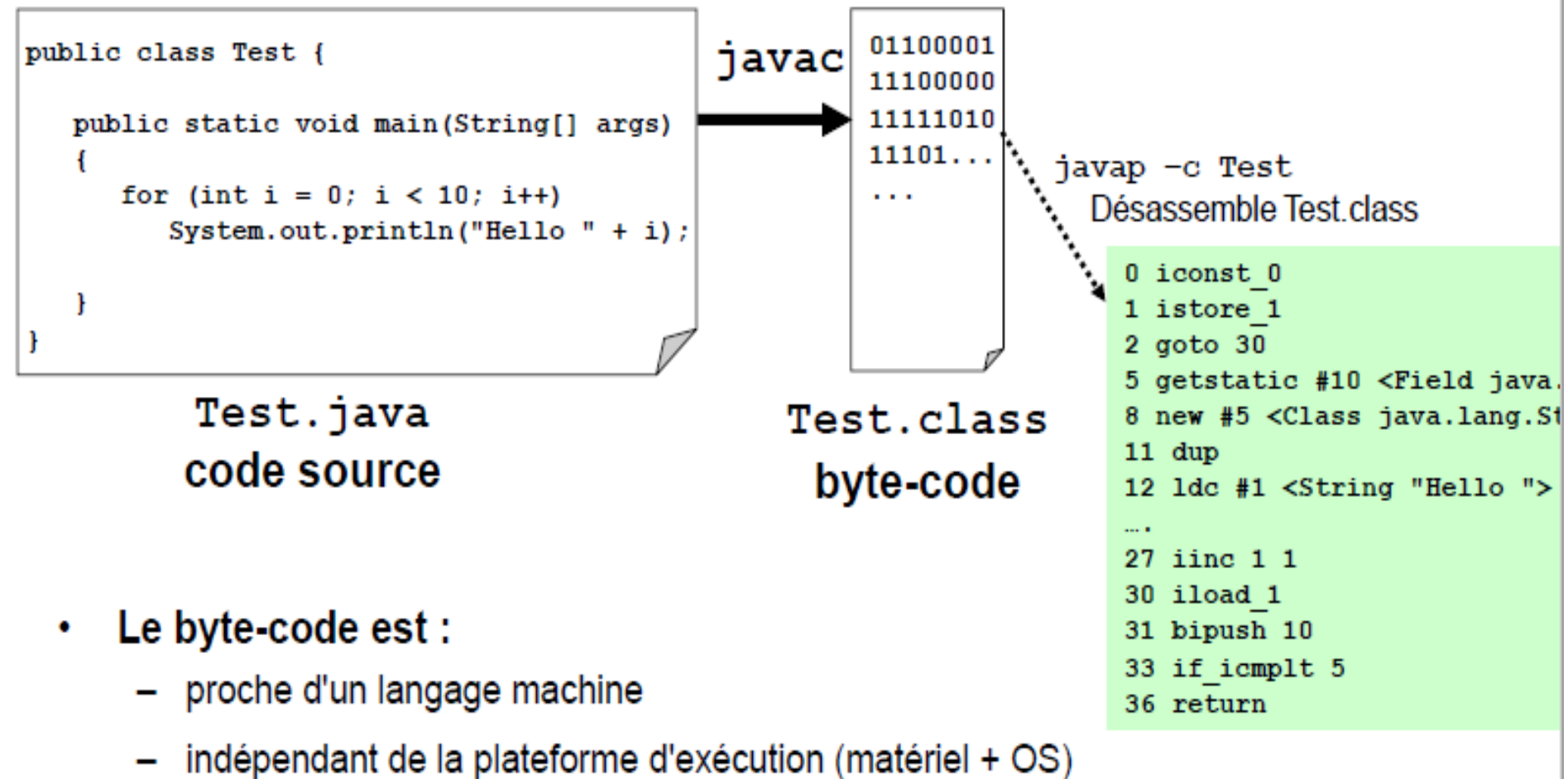
Java en quelques mots:

- **Interprété :**
 - Avantage au développement.
 - Mais est pour l'instant un inconvénient à l'exécution (performances).
- **Indépendant de l'architecture :**
 - C'est la machine virtuelle qui assure cette indépendance.
- **Portable :**
 - Contrairement au C/C++ Java n'est pas dépendant de l'implémentation, les types de base ont des tailles fixes.
- **Multithread :**
 - C'est la possibilité d'exécuter plusieurs tâches simultanément.
- **Dynamique :**
 - Java a été conçu pour fonctionner dans un environnement évolutif.
- **Compact :**
 - L'interpréteur Java n'a besoin que de 215Ko.
 - Chargement dynamique des classes.

Présentation du langage

Un langage compilé / interprété

- **Compilation d'un programme JAVA : génération de byte-code**



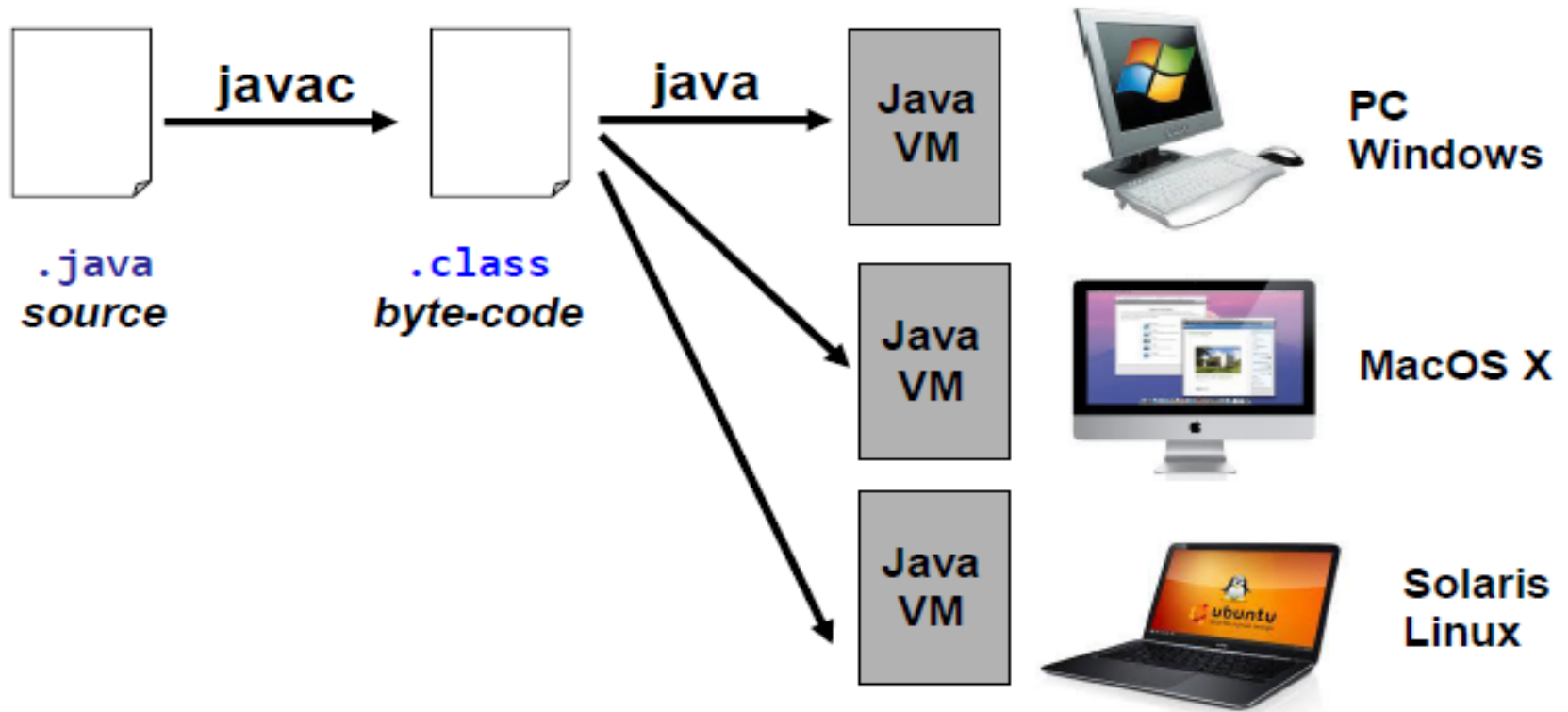
Présentation du langage

La machine virtuelle Java

Exécution d'un programme Java compilé

byte-code assure la portabilité des programmes Java

- langage d'une Machine Virtuelle
- à l'exécution un interpréteur simule cette machine virtuelle



Présentation du langage

La machine virtuelle Java

Avantages de la JVM pour Internet

- Grâce à sa portabilité, le *bytecode* d'une classe peut être chargé depuis une machine distante du réseau, et exécutée par une JVM locale,
- La JVM fait de nombreuses vérifications sur le *bytecode* avant son exécution pour s'assurer qu'il ne va effectuer aucune action dangereuse
- La JVM apporte donc
 - de la souplesse pour le chargement du code à exécuter
 - mais aussi de la sécurité pour l'exécution de ce code
- Les vérifications effectuées sur le *bytecode* et l'étape d'interprétation de ce *bytecode* (dans le langage natif du processeur) ralentissent l'exécution des classes Java
- Mais les techniques « *Just In Time* (JIT) » ou « *Hotspot* » réduisent ce problème : elles permettent de ne traduire qu'une seule fois en code natif les instructions qui sont (souvent pour Hotspot) exécutées

Présentation du langage

Types de programmes Java

- Une application Java

- est composée d'une classe possédant une méthode main() :

```
public static void main (String[] args){  
    //code à exécuter pour initialiser l'application  
}
```
- L'environnement d'exécution dépend de l'OS de la machine
- Pas de restriction au niveau des API

- Une applet Java

- Comprend une classe publique dérivant de java.applet.Applet
- L'environnement d'exécution dépend du browser Web
- Restrictions au niveau des API
 - Généralement pas autorisée à lire ou écrire sur des fichiers locaux.
 - Interdiction d'ouvrir des connections réseaux vers d'autres systèmes que la machine hôte qui a chargé l'applet
 - ...

Présentation du langage

Types de programmes Java: Application

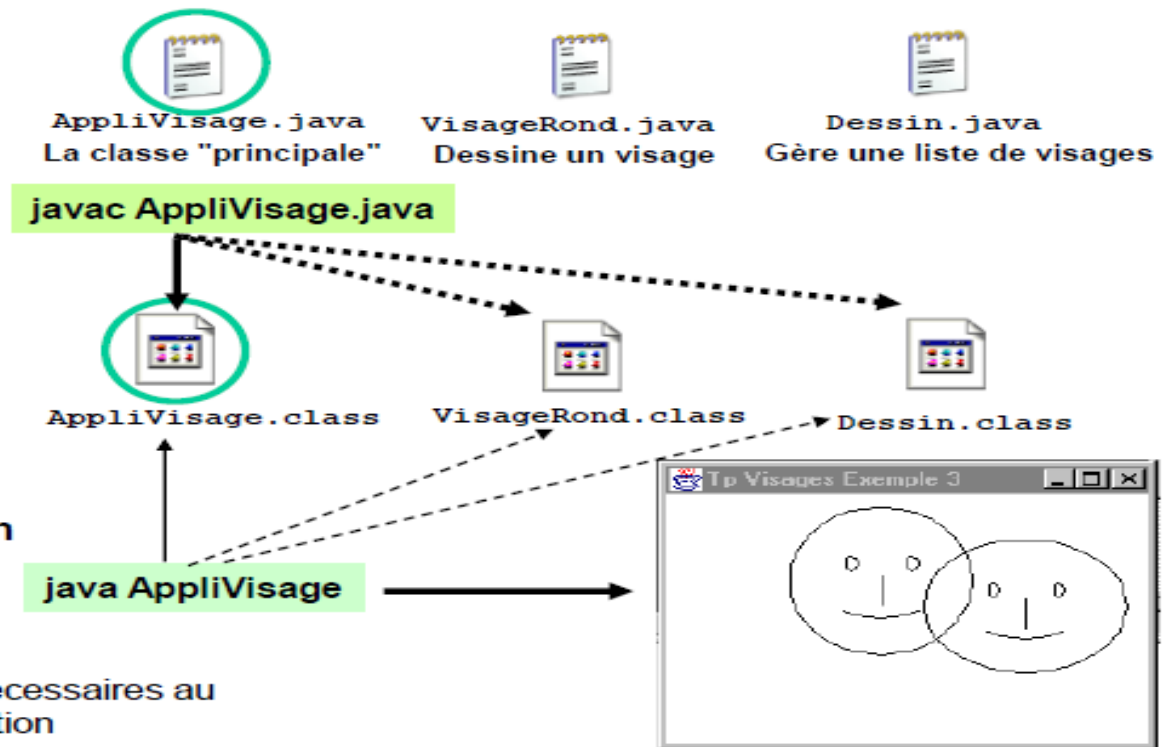
Application est définie par un ensemble de classes dont **une** jouera le rôle de **classe principale**

La compilation de la classe principale entraîne la compilation de toutes les classes utilisées

`javac \simeq make`

Pour exécuter l'application on indique à l'interpréteur **java** le nom de la classe principale

java charge les classes nécessaires au fur et à mesure de l'exécution



- **Application doit posséder une classe principale**

- classe possédant une méthode de signature

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

Tableau de chaînes de caractères
(équivalent à `argc, argv` du C)

- **Cette méthode sert de point d'entrée pour l'exécution**

- l'exécution de l'application démarre par l'interprétation de cette méthode

```
ex: java AppliVisage1
```

Exécute le code défini dans la méthode
main contenue dans le fichier
`AppliVisage1.class`

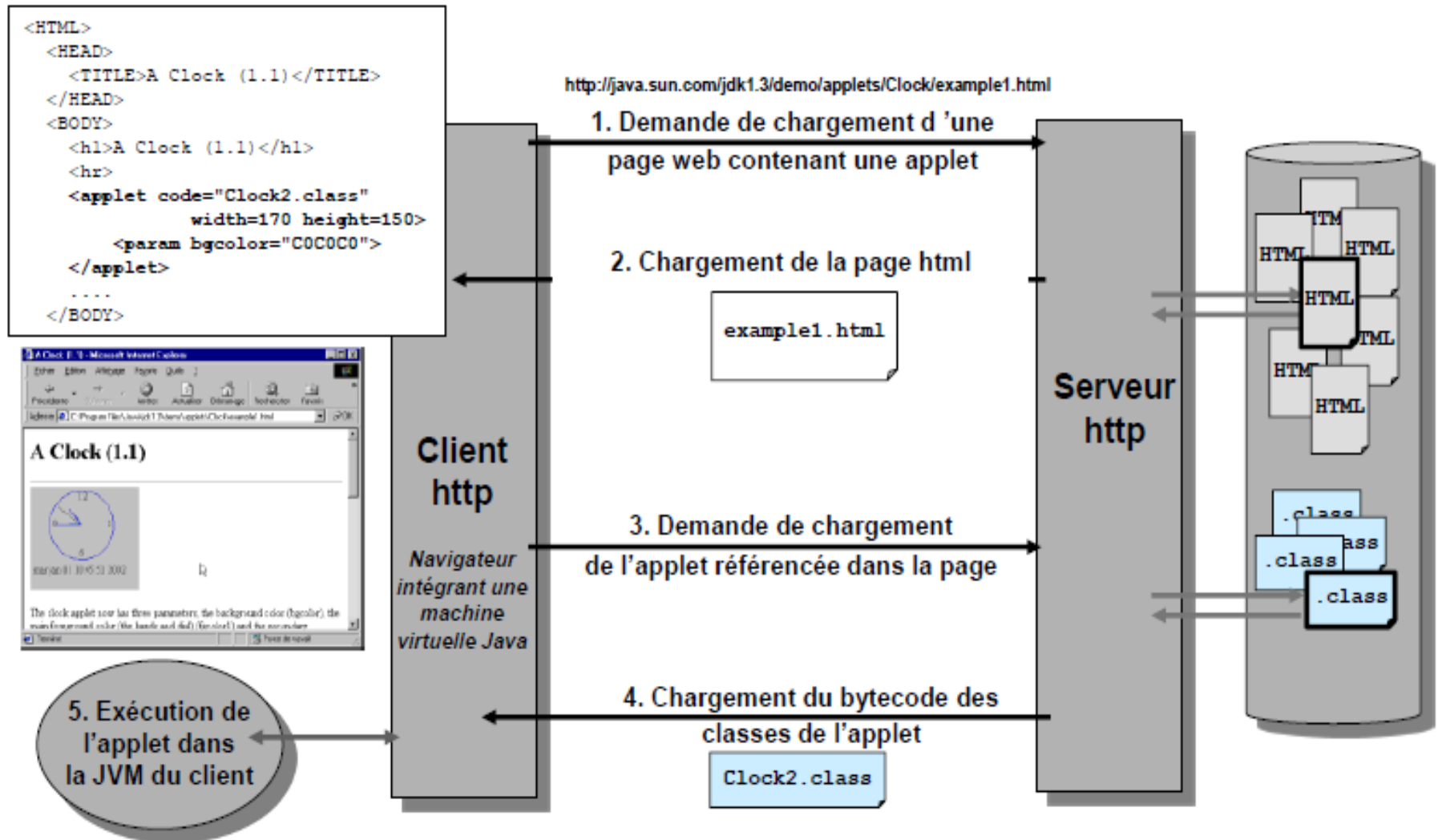
Présentation du langage

Types de programmes Java: Applet

- Classe principale ne possède pas de méthode `main()`
- Hérite de `java.awt.Applet` ou `javax.swing.JApplet`
- Son bytecode réside sur un serveur http
- Elle est véhiculée vers un client http (navigateur Web) via une page html qui contient son url
- Lorsqu'un navigateur compatible Java (avec sa propre machine virtuelle java (JVM)) reçoit cette page HTML, il télécharge le code de la classe et l'exécute sur le poste client
 - l'applet doit posséder un certain nombre de méthodes pour permettre cette exécution
 - `init()`, `start()`, `stop()`, `paint()`, `destroy()`

Présentation du langage

Types de programmes Java: Applet



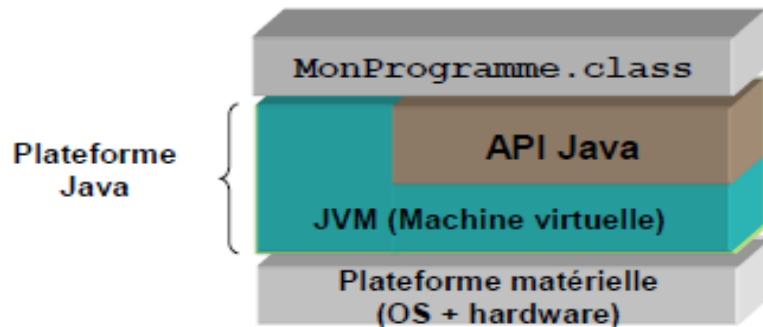
Présentation du langage

La plateforme JAVA

- **Plateforme**

- Environnement matériel et/ou logiciel dans lequel un programme s'exécute.
 - La plus part des plateformes sont la combinaison d'un OS et du matériel sous-jacent (*MS Windows + Intel, Linux + Intel, Solaris + Sparc, Mac Os X + Power PC*)

- **La plateforme Java est entièrement logicielle et s'exécute au dessus des plateformes matérielles**



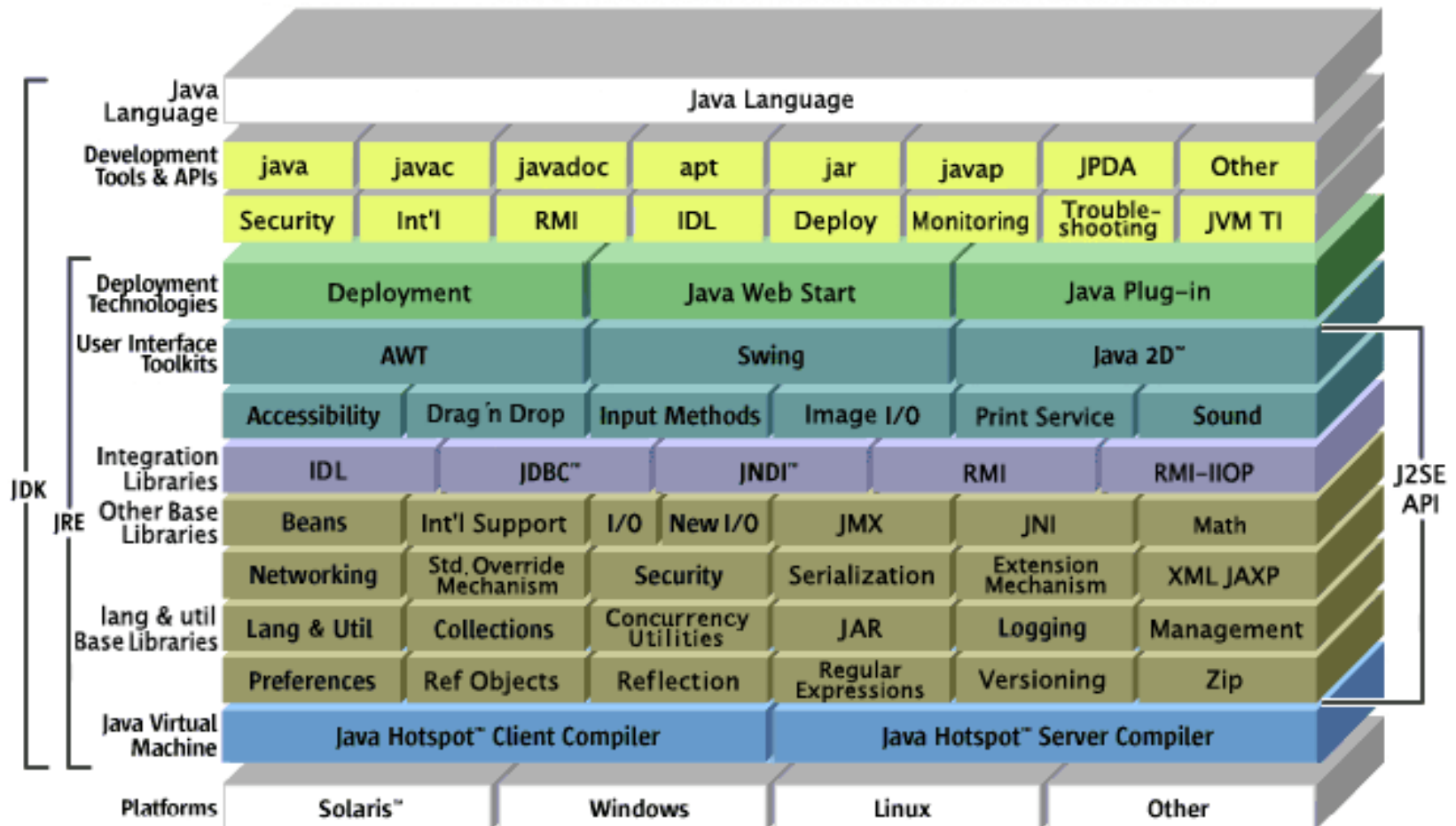
API (Application Programming Interface) Java :
Bibliothèques Java standards sur lesquelles le programmeur peut s'appuyer pour écrire son code

- **API Java**

- (très) vaste collection de composants logiciels (classes et interfaces)
- organisée en bibliothèques (*packages*)
- offre de nombreux services de manière standard (indépendamment de la plateforme matérielle)

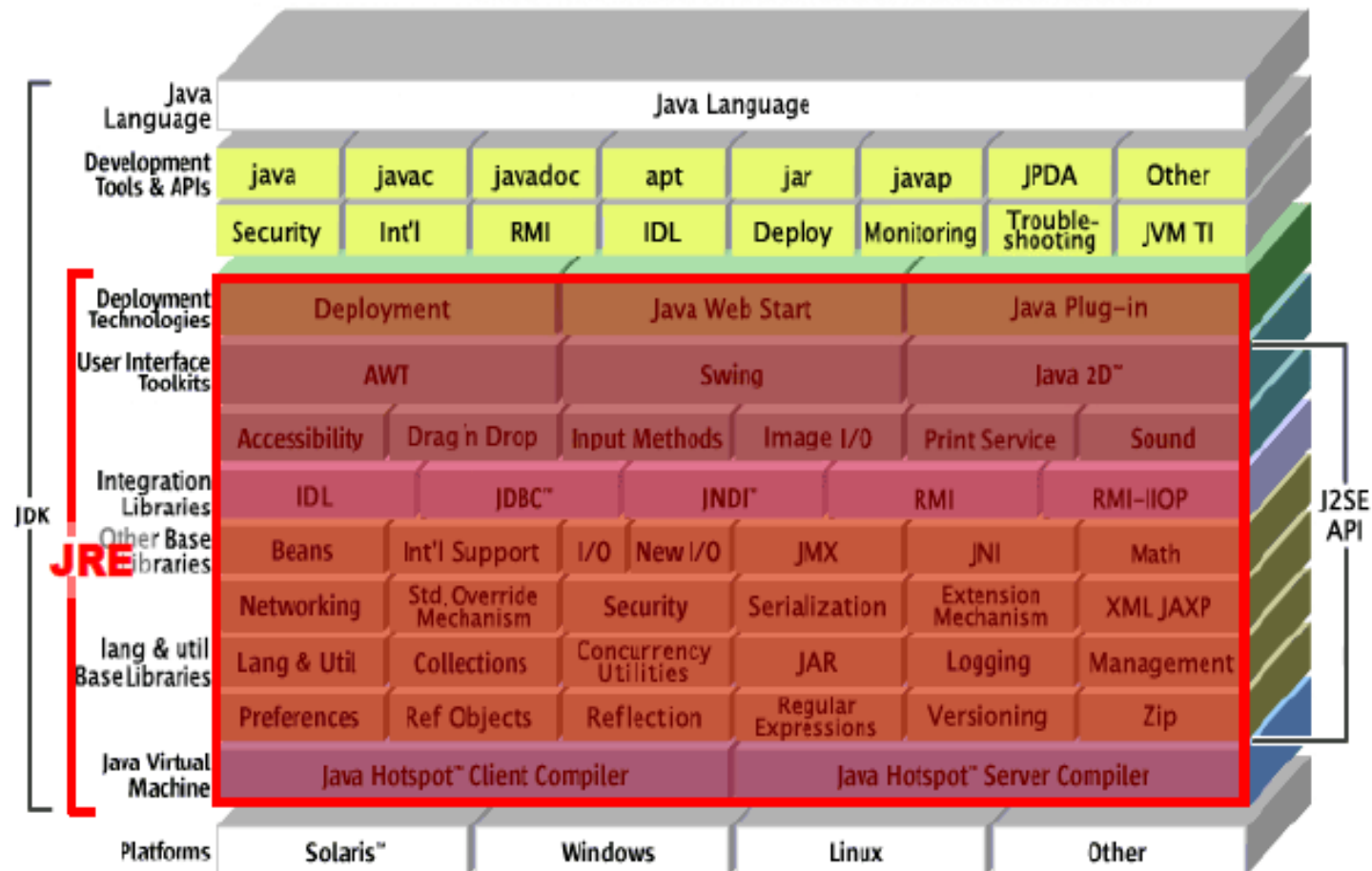
Présentation du langage

La plateforme JAVA



Présentation du langage

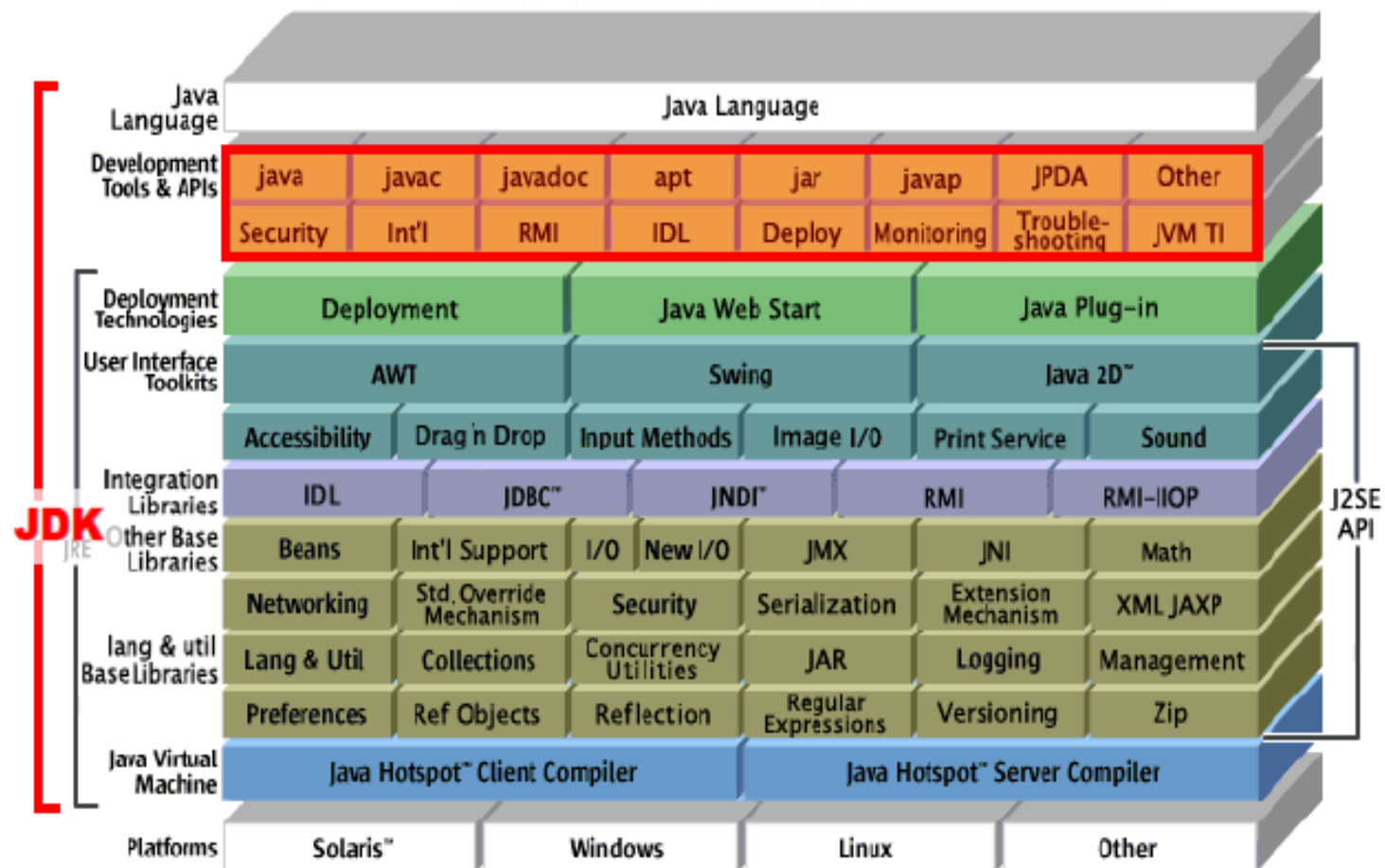
La plateforme JAVA



JRE (Java Runtime Environnement) pour l'exécution de code java compilé

Présentation du langage

La plateforme JAVA



JDK (Java Developer's Kit) outils de base pour le développement d'applications Java

Présentation du langage

La plateforme JAVA

- 3 éditions de Java



Standard Edition JSE

Fourni les compilateurs, outils, runtimes, et APIs pour écrire, déployer, et exécuter des applets et applications dans la langage de programmation Java



Entreprise Edition JEE

Destinée au développement d'applications « d'entreprise » (*«business applications»*) robustes et interopérables. Simplifier le développement et le déploiement d'applications distribuées et articulées autour du web.



Mobile Edition JME

Environnement d'exécution optimisé pour les dispositifs « légers » :

- Carte à puce (smart cards)
- Téléphones mobiles
- Assistants personnels (PDA)

Présentation du langage

Environnements de développement intégrés

- **Nombreux IDE (Integrated Development Environment) pour java**
 - Editeur syntaxique, débogueur, compilateur, exécution
- **Commerciaux**



JCreator
Xinox



WebSphere Studio
Site Developer for Java
IBM



JBuilder
Codegear

ORACLE 10^g
JDEVELOPER

JDeveloper
Oracle

IntelliJIDEA
JetBrains

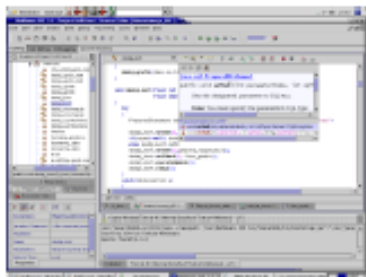
...



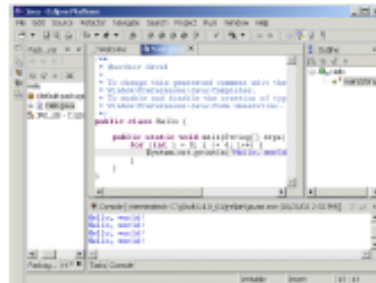
Visual J++
Microsoft

C# .net

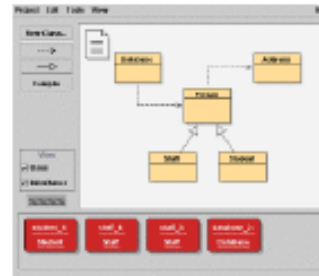
- **Open-source et/ou freeware**



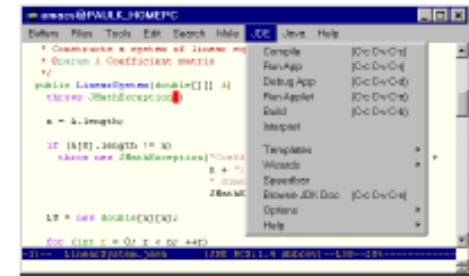
NetBeans
www.netbeans.org



Eclipse
www.eclipse.org



BlueJ
www.bluej.org



Emacs + JDE
<http://sunsite.auc.dk/jde>

Bases du langage Java

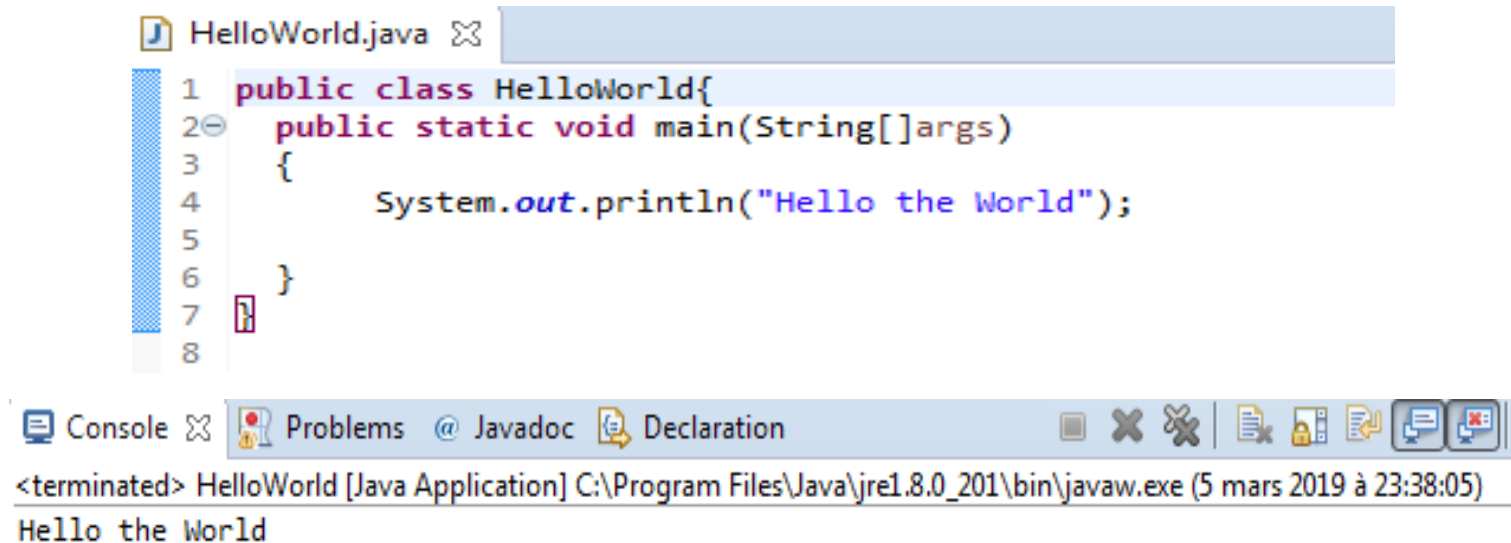
Première application en Java

Deux façons d'écrire des programmes Java:

- En écrivant le code dans un simple éditeur de texte
 - Compilation et exécution du code en ligne de commande DOS
- En utilisant un environnement de développement (IDE)
 - Netbeans
 - Eclipse
 - Jcreator
 - IntelliJIDIA
 - ...

Bases du langage Java

Première application en Java Application HelloWorld



The screenshot shows an IDE window titled 'HelloWorld.java'. The code is as follows:

```
1 public class HelloWorld{
2     public static void main(String[] args)
3     {
4         System.out.println("Hello the World");
5     }
6 }
7
8
```

Below the code editor, there is a toolbar with icons for Console, Problems, Javadoc, and Declaration. The Console tab is active, showing the output: '<terminated> HelloWorld [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (5 mars 2019 à 23:38:05) Hello the World'.

- **public class HelloWorld** : Nom de la classe
- **public static void main** : La fonction principale équivalent à la fonction main du C/C++.
- **String[] argv** : Permet de récupérer des arguments transmis au programme au moment de son lancement
- **System.out.println(« Hello the world, ... »)** : Méthode d'affichage dans la fenêtre console.

Règle de bonne pratique :

1 classe par fichier et 1 fichier par classe

Bases du langage Java

Première application en Java **Application HelloWorld**

- Créer un fichier texte : HelloWorld.java

```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello the World");
    }
}
```

- Compiler le programme : javac HelloWorld.java
- Le compilateur génère le bytecode dans le fichier : HelloWorld.class
- Exécuter l'application : java HelloWorld

« Hello the World » s'affiche à l'écran

Bases du langage Java

Première application en Java

Applet HelloWorld

- Créer un fichier texte : HelloWorldApplet.java

```
import java.applet.Applet;  
import java.awt.Graphics;  
  
public class HelloWorldApplet extends Applet  
{  
    public void paint(Graphics g)  
    {  
        g.drawString("Hello the World", 50, 25);  
    }  
}
```

Importation des classes externes nécessaires
(issues des API Java)

Déclaration de la classe qui hérite de la classe
prédéfinie « Applet »

La méthode paint détermine l’affichage dans la
fenêtre de l’Applet

Écrire à l’écran “Hello the World”

Fermer les accolades

- Compiler le programme : `javac HelloWorldApplet.java`
- Le compilateur génère le bytecode dans le fichier :
`HelloWorldApplet.class`

Bases du langage Java

Première application en Java

Applet HelloWorld

- Les applets s'exécutent dans une page HTML
- Pour intégrer l'applet dans une page HTML, il suffit d'utiliser la balise <APPLET>
- Le paramètre « CODE » de la balise <APPLET> indique le nom de la classe principale de l'applet

```
<HTML>
```

```
<HEAD>
```

```
<TITLE> A Simple Program </TITLE>
```

```
</HEAD>
```

```
<BODY> Here is the output of my program:
```

```
<APPLET CODE="HelloWorldApplet.class" WIDTH=150 HEIGHT=75>
```

```
</APPLET>
```

```
</BODY>
```

```
</HTML>
```

- Ouvrir la page HTML dans un navigateur, l'applet se lance automatiquement au sein de la page

Syntaxe du langage Java

Commentaires

Trois façons d'inclure des commentaires :

- Tout texte entre « `//` » et la fin de la ligne
`// Commentaires sur une seule ligne`
- Tout texte entre « `/*` » et « `*/` »
`/* Commentaires`
`sur un nombre important voire très important`
`de lignes */`
- Les textes entre « `/**` » et « `*/` » sont utilisés pour créer des commentaires que l'exécutable JAVADOC pourra traiter afin de produire une documentation (cf. documentation de l'API Java)
`/** Commentaires destinés`
`à la documentation */`
- Commenter le plus possible et judicieusement
- Commenter clairement(utiliser au mieux les 3 possibilités)
- Chaque déclaration de classe et de membre d'une classe(variable et méthode) doit être commentée avec un commentaire javadoc/`/** ... */`

Syntaxe du langage Java

Identificateurs

- Nommer les classes, les variables, les méthodes, ...
 - Un identificateur Java
 - est de longueur quelconque
 - *commence par une lettre Unicode (caractères ASCII recommandés)*
 - *peut ensuite contenir des lettres ou des chiffres ou le caractère souligné « _ »*
 - *ne doit pas être un mot réservé du langage (mot clé) (**if**, **for**, **true**, ...)*
- [a..z, A..Z, \$, _]{a..z, A..Z, \$, _, 0..9, Unicode}**
- Les noms de classes commencent par une majuscule (ce sont les seuls avec les constantes) :
 - Visage, Object
 - Les mots contenus dans un identificateur commencent par une majuscule :
 - **uneClasse, uneMethode, uneVariable**
 - On préférera **ageDuCapitaine** à **ageducapitaine** ou **age_du_capitaine**
 - Les constantes sont en majuscules et les mots sont séparés par le caractère souligné « _ » :
 - **UNE_CONSTANTE**

Syntaxe du langage Java

Mots – Clés en Java

abstract, boolean, break, byte, case, catch, char, class, continue, default, do, double, else, extends, final, finally, float, for, 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

Syntaxe du langage Java

Types de données en Java

- Deux grands groupes de types de données :
 - types primitifs
 - objets (instances de classe)
- Java manipule différemment les valeurs des types primitifs et les objets : les variables contiennent
 - des valeurs de types primitifs
 - ou des références aux objets

Syntaxe du langage Java

Types primitifs

- Valeur logique
 - boolean (true/false)
- Nombres entiers
 - byte (1 octet), short (2 octets), int (4 octets), long (8 octets)
- Nombres non entiers (à virgule flottante)
 - float (4 octets), double (8 octets).
- Caractère (un seul)
 - char (2 octets) ; codé par le codage Unicode (et pas ASCII)

Syntaxe du langage Java

Types primitifs et valeurs

Type	Taille	Valeurs
<code>boolean</code>	1	<code>true</code> , <code>false</code>
<code>byte</code>	8	-2^7 à $+2^7-1$
<code>char</code>	16	0 à 65535
<code>short</code>	16	-2^{15} à $+2^{15}-1$
<code>int</code>	32	-2^{31} à $+2^{31}-1$
<code>long</code>	64	-2^{63} à $+2^{63}-1$
<code>float</code>	32	1.40239846e-45 à 3.40282347e38
<code>double</code>	64	4.94065645841246544e-324 à 1.79769313486231570e308

Syntaxe du langage Java

Littéral de type numérique

- un **littéral** est une valeur explicite utilisée dans le code source d'un programme.
- On représente les entiers simplement en donnant leur valeur sous forme décimale, binaire, octale ou hexadécimale.
- Ces valeurs sont considérées par Java comme des valeurs de type int. On peut aussi avoir des littéraux de type long ; il suffit de faire suivre le nombre par la lettre l ou L.
- Voici plusieurs littéraux qui représentent tous l'entier 45

```
45           // en décimal (int)
0b101101;    // en binaire (int)
055          // en octal (int)
0x2d         // en hexadécimal (int)

45L          // en décimal (long)
0b101101L;   // en binaire (long)
055L         // en octal (long)
0x2dL        // en hexadécimal (long)
```

Syntaxe du langage Java

Littéral de type numérique

- On peut également écrire un nombre flottant en notation scientifique en le faisant suivre de e ou E suivi de la valeur de l'exposant qui peut être positif ou négatif
- Voici plusieurs littéraux représentant le nombre flottant 25,5, avec le type double pour les deux premiers et float pour les deux derniers :

```
25.5      // en notation décimale (double)
2.55e+1    // en notation scientifique (double)

25.5F     // en notation décimale (float)
2.55e+1F   // en notation scientifique (float)
```

Ajout de JDK 7

- Il est possible d'insérer le caractère « souligné » dans l'écriture des nombres entiers :
1_567_688 ou 0x50_45_14_56
- On peut aussi écrire un nombre en binaire :
ob01010000010001010001010001010110 ou
ob01010000_01000101_00010100_01010110

Syntaxe du langage Java

Littéral de type caractère

- Un caractère Unicode entouré par 2 simples quotes " ' "
 - 'A' 'a' 'ç' '1' '2'
 - \caractère d'échappement pour introduire les caractères spéciaux
 - '\t' tabulation
 - '\n' nouvelle ligne
 - '\r' retour chariot, retour arrière
 - '\f' saut de page
 - ...
 - '\\ - \' - '\"'
- '\u03a9' (\u suivi du code hexadécimal à 4 chiffres d'un caractère Unicode)
 - 'α'

Littéral de type booléen

- Type booléen
 - false
 - true
- Référence inexistante (indique qu'une variable de type non primitif ne référence rien)
 - null

Syntaxe du langage Java

Valeurs par défaut

Si elles ne sont pas initialisées, les variables d'instance ou de classe (pas les variables locales d'une méthode) reçoivent par défaut les valeurs suivantes :

boolean	false
char	' \u0000 '
Entier (byte short int long)	0 0L
Flottant (float double)	0.0F 0.0D
Référence d'objet	null

Syntaxe du langage Java

La pile et le tas

- Java manipule différemment les types primitifs et les objets
- Les variables contiennent
 - des valeurs de types primitifs
 - des références aux objets
- L'espace mémoire alloué à une variable locale est situé dans la **pile**
- Si la variable est d'un type primitif, sa valeur est placée dans la **pile**
- Sinon la variable contient une référence à un objet ; la valeur de la référence est placée dans la pile mais l'objet référencé est placé dans le **tas**
- Lorsque l'objet n'est plus référencé, un « ramasse-miettes » (*garbage collector*, GC) libère la mémoire qui lui a été allouée

Syntaxe du langage Java

Modificateur final

- Le modificateur **final** indique que la valeur de la variable ne peut être modifiée: on pourra lui donner une valeur une seule fois dans le programme.
- Une variable de classe **static final** est constante dans tout le programme exemple : **static final double PI = 3.14;**
- Une variable de classe **static final** peut ne pas être initialisée à sa déclaration mais elle doit alors recevoir sa valeur dans un bloc d'initialisation **static**

Syntaxe du langage Java

Forcer un type en Java

- Java est un langage fortement typé
 - *le type de donnée est associé au nom de la variable, plutôt qu'à sa valeur. (Avant de pouvoir être utilisée une variable doit être déclarée en associant un type à son identificateur).*
 - *la compilation ou l'exécution peuvent détecter des erreurs de typage*
- Dans certains cas, il est nécessaire de forcer le programme à considérer une expression comme étant d'un type qui n'est pas son type réel ou déclaré
- On utilise pour cela le *cast* (transtypage) : *(type-forcé) expression*

```
int x = 10, y = 3;  
// on veut 3.3333.. et pas 3.0  
double z = (double)x / y; // cast de x suffit
```

Syntaxe du langage Java

Cast entre types primitifs

- Un *cast* entre types primitifs peut occasionner une perte de données :
 - conversion d'un **int** vers un **short**

```
int i = 32768;  
short s = (short) i;  
System.out.println(s); → -32767;
```

- Un *cast* peut provoquer une simple perte de précision :
 - conversion d'un **long** vers un **float** peut faire perdre des chiffres significatifs mais pas l'ordre de grandeur

```
long l1 = 9289999999L;  
float f = (float) l1;  
System.out.println(f); → 9.29E8  
long l2 = (long) f;  
System.out.println(l2); → 929000000
```

Syntaxe du langage Java

Cast entre types primitifs

Cast implicite entre types primitifs

- Les affectations entre types primitifs peuvent utiliser un *cast* implicite si elles ne peuvent provoquer qu'une perte de précision (ou, encore mieux, aucune perte)

```
int i = 130;  
double x = 20 * i;
```

Cast explicite entre types primitifs

- Si une affectation peut provoquer une perte de valeur, elle doit comporter un *cast* explicite :

```
short s = 65;           // cas particulier affectation int "petit"  
s = 1000000;           // provoque une erreur de compilation  
int i = 64;  
byte b = (byte)(i + 2); // b = 66  
char c = (char) i;      // caractère dont le code est 64 → '@'  
b = (byte)128;          // b = -128 !
```

Type mismatch: cannot convert from int to short

- L'oubli de ce *cast* explicite provoque une erreur à la compilation .

Syntaxe du langage Java

Cast entre types primitifs

Exemples de casts

- **short s = 1000000; // erreur !**

Cas particulier d'une affectation statique (repérable par le compilateur) d'un **int** « petit » :

- **short s = 65; // pas d'erreur**

Pour une affectation non statique, le *cast* explicite est obligatoire :

- **int i = 60;**
- **short b = (short)(i + 5);**
- Les *casts* de types « flottants » vers les types entiers tronquent les nombres :
- **int i = (int)1.99; // i = 1, et pas 2**

```
byte b1 = (byte)20;  
byte b2 = (byte)15;  
byte b3 = b1 + b2
```

- provoque une erreur (**b1 + b2** est un **int**)
- La dernière ligne doit s'écrire
- **byte b3 = (byte)(b1 + b2);**

Syntaxe du langage Java

Cast entre types primitifs

Problèmes de casts

- Une simple perte de précision ne nécessite pas de *cast* explicite, mais peut conduire à des résultats comportant une erreur importante :
 - **long l1 = 123456789;**
 - **long l2 = 123456788;**
 - **float f1 = l1;**
 - **float f2 = l2;**
 - **System.out.println(f1); // 1.23456792E8**
 - **System.out.println(f2); // 1.23456784E8**
 - **System.out.println(l1 - l2); // 1**
 - **System.out.println(f1 - f2); // 8 !**
- Attention, dans le cas d'un *cast* explicite, la traduction peut donner un résultat totalement aberrant sans aucun avertissement ni message d'erreur :
 - **int i = 130;**
 - **b = (byte)i; // b = -126 !**
 - **int c = (int)1e+30; // c = 2147483647 !**

Syntaxe du langage Java

Casts entre entiers et caractères

- La correspondance **char** → **int**, **long** s'obtient par *cast* implicite

```
char c = '@';  
int i = c; // ⇔ int i = (int) c;  
System.out.println(i); // → 64 le rang du caractère '@' dans le codage
```

- Les correspondances **char** → **short**, **byte**, et **long**, **int**, **short** ou **byte** → **char** nécessitent un *cast* explicite (entiers sont signés et pas les **char**)

```
char c = '@';  
short s = c; ✗ Type mismatch: cannot convert from char to short  
int i = 64;  
char c = i; ✗ Type mismatch: cannot convert from int to char
```

```
short s = (short) c;
```

```
char c = (char) i;
```

Syntaxe du langage Java

Boxing/unboxing

- Le « *autoboxing* » (mise en boîte) automatise le passage des types primitifs vers les classes qui les enveloppent
- Cette mise en boîte automatique a été introduite par la version 5 du JDK
- L'opération inverse s'appelle « *unboxing* »

Syntaxe du langage Java

Opérateurs

- Les plus utilisés
 - Arithmétiques
 - + - * /
 - % (modulo)
 - ++ --(pré ou post décrémentation)
 - Logiques
 - && (et) ||(ou) !(négation)
 - Relationnels
 - == != < > <= >=
 - Affectations
 - = += -= *= ...
- **x++** : la valeur actuelle de **x** est utilisée dans l'expression et *juste après* **x** est incrémenté.
- **+x** : la valeur de **x** est incrémentée et ensuite la valeur de **x** est utilisée
 - **x = 1; y = ++x * ++x; // y a la valeur 6 (2 x 3) !**

Opérateur instanceof

- La syntaxe est :
 - *objet* **instanceof** *nomClasse*
- Exemple : **if (x instanceof Livre)**
- Le résultat est un booléen :
 - **true** si **x** est de la classe **Livre**
 - **false** sinon

Syntaxe du langage Java

Opérateurs

Ordre de priorité des opérateurs

Postfixés	[] . (params) expr++ expr--
Unaires	++expr --expr +expr -expr ~ !
Création et cast	new (type) expr
Multiplicatifs	* / %
Additifs	+ -
Décalages bits	<< >>
Relationnels	< > <= >= instanceof
Egalité	== !=
Bits, et	&
Bits, ou exclusif	^
Bits, ou inclusif	
Logique, et	&&
Logique, ou	
Conditionnel	? :
Affectation	= += -= *= /= %= &= ^= = <<= >>=

Les opérateurs d'égales priorités sont évalués de gauche à droite, sauf les opérateurs d'affectation, évalués de droite à gauche

Syntaxe du langage Java

Déclarations

- Avant toute utilisation dans un programme une variable doit être déclarée
- syntaxe: ***type identificateur***
 - type : un type primitif ou un nom de classe
- Exemples
 - **byte age;**
 - **boolean jeune;**
 - **float poids;**
 - **double x, y ,z;**
- Une variable est accessible (**visible**) depuis l'endroit où elle est déclarée jusqu'à la fin du bloc où sa déclaration a été effectuée

Syntaxe du langage Java

Affectation

- Syntaxe : *lvalue* = *expression*
 - *lvalue* est une expression qui doit délivrer une variable (par exemple un identificateur de variable, élément de tableau..., mais pas une constante)
- Exemples
 - **int age;**
 - **age = 10;**
 - **boolean jeune = true;**
 - **float poids = 71.5f;**
 - **float taille = 1.75f;**
 - **float poidsTaille = poids / taille;**
- Attention en JAVA comme en C, l'affectation est un opérateur. L'affectation peut donc être utilisée comme une expression dont la valeur est la valeur affectée à la variable
 - **i = j = 10;**

Syntaxe du langage Java

Les traitements conditionnels en Java

- On appelle traitement conditionnel une portion de code qui n'est pas exécutée systématiquement.

Le bloc if

En pseudo-code un traitement conditionnel se rédige de la sorte :

```
Si < condition > alors  
  < instructions >  
Fin
```

En java un traitement conditionnel se formule de la sorte :

```
if (<condition>)  
{  
  <instructions>  
}
```


Syntaxe du langage Java

Les traitements conditionnels en Java

- **Exemple**

```
import java.util.*;
public class HelloWorld{
    public static void main(String[]args)
    {
        System.out.println("Saisissez une valeur");
        Scanner sc = new Scanner(System.in);
        int i = sc.nextInt();
        if (i == 0)
        {
            System.out.println("Vous avez saisi une valeur nulle.");
        }
        System.out.println("Au revoir !");
    }
}
```

Syntaxe du langage Java

Les traitements conditionnels en Java

Si ... Alors ... Sinon

Il existe une forme étendue de traitement conditionnel, on la note en pseudo-code de la façon suivante :

```
Si condition alors
    instructions
Sinon
    autresinstructions
Fin
```

En java un traitement conditionnel étendu se formule de la sorte :

```
if (<condition>)
{
    <instructions1>;
}
else
{
    <instructions2>;
}
```

Syntaxe du langage Java

Les traitements conditionnels en Java

- **Exemple**

```
import java.util.*;
public class HelloWorld{
    public static void main(String[]args)
    {
        System.out.println("Saisissez une valeur");
        Scanner sc = new Scanner(System.in);
        int i = sc.nextInt();
        if (i == 0)
        {
            System.out.println("Vous avez saisi une valeur nulle.");
        }
        else
        {
            System.out.println("La valeur que vous avez saisi, " +i +",
n'est pas nulle.");
        }
    }
}
```

Syntaxe du langage Java

Les traitements conditionnels en Java

Switch

```
switch(<nomvariable>)  
{  
  case <valeur_1> : <instructions_1> ; break ;  
  case <valeur_2> : <instructions_2> ; break ;  
  /* ... */  
  case <valeur_n> : <instructions_n> ; break ;  
  default : <instructionspardefaut> ; break;  
}
```

Syntaxe du langage Java

Les traitements conditionnels en Java

- **Exemple**

```
switch(numeroMois)
{
case 1 : System.out.print("janvier") ; break ;
case 2 : System.out.print("fevrier") ; break ;
case 3 : System.out.print("mars") ; break ;
case 4 : System.out.print("avril") ; break ;
case 5 : System.out.print("mai") ; break ;
case 6 : System.out.print("juin") ; break ;
case 7 : System.out.print("juillet") ; break ;
case 8 : System.out.print("aout") ; break ;
case 9 : System.out.print("septembre") ; break ;
case 10 : System.out.print("octobre") ; break ;
case 11 : System.out.print("novembre") ; break ;
case 12 : System.out.print("decembre") ; break ;
default : System.out.print("Je connais pas ce mois...");break;
}
```

Attention, sans **break**,
les instructions du cas
suivant sont exécutées !

Syntaxe du langage Java

Les boucles en Java

Une boucle permet d'exécuter plusieurs fois de suite une même séquence d'instructions.

While : tantque ... faire

```
while(<condition>)  
{  
  <instructions>  
}
```

Syntaxe du langage Java

Les boucles en Java

- **Exemple**

```
public static void main(String[] args){  
    int i = 1;  
    while (i <= 5){  
        System.out.print(i + " ");  
        i++;  
    }  
    System.out.println("\n");  
}
```

N'oubliez pas lorsqu'une boucle fonctionne avec un compteur :

- D'initialiser le compteur avant d'entrer dans la boucle
- D'incrémenter le compteur à la fin du corps
- De contrôler la valeur du compteur dans la condition de boucle

Syntaxe du langage Java

Les boucles en Java

```
public static void main(String[] args)
{
    int i = Integer.parseInt(args[0]);
    int j = 2;
    while (i % j != 0) {
        j++;
    }
    System.out.println("PPD de "
                       + i + " : " + j);
}
```


Syntaxe du langage Java

Les boucles en Java

do ... While : répéter ... jusqu'à

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

Le fonctionnement est analogue à celui de la boucle tant que à quelques détails près :

- la condition est évaluée **après** chaque passage dans la boucle.
- On exécute le corps de la boucle **tant que la condition** est vérifiée.
- On passe toujours **au moins une fois** dans une boucle répéter... jusqu'à

Syntaxe du langage Java

Les boucles en Java

- **Exemple**

```
public static void main(String[]args)
{
    Scanner saisie = new Scanner(System.in);
    int i;
    do
    {
        System.out.print("Saisissez un entier positif ou nul : ");
        i = saisie.nextInt();
        if (i < 0)
            System.out.println("J'ai dit positif ou nul !");
    }
    while (i < 0);
    saisie.close();
    System.out.println("Vous avez saisi " + i + ". ");
}
```

Syntaxe du langage Java

Les boucles en Java

For

```
for(<initialisation> ; <condition> ; <pas>)  
{  
  <instructions>  
}  
http://
```

```
public static void main(String[] args)  
{  
    for (int i = 1; i <= 5 ;i++)  
        System.out.print(i + " ");  
    System.out.println("\n");  
}
```

La gestion du compteur est automatique (initialisation, incrémentation, sortie de boucle).

Il faut connaître à l'avance le nombre d'itérations.

Syntaxe du langage Java

Expression conditionnelle

ExpressionBooléenne ? expression1 : expression2

```
int y = (x % 2 == 0) ? x + 1 : x;
```

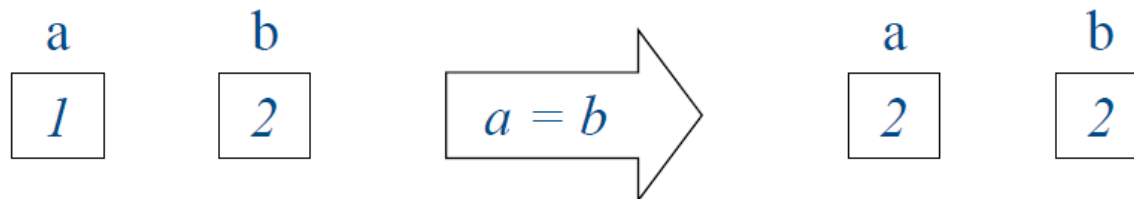
est équivalent à

```
int y;  
if (x % 2 == 0)  
    y = x + 1  
else  
    y = x;
```

Syntaxe du langage Java

Affectation, recopie et comparaison

- Affecter et recopier un type primitif
 - **a=b** : signifie **a** prend la valeur de **b**
 - **a** et **b** sont distincts
 - Toute modification de **a** n'entraîne pas celle de **b**
- Comparer un type primitif
 - **a == b** : retourne **true** si les valeurs de **a** et **b** sont identiques



Syntaxe du langage Java

Entrées/sorties sur console

Affichage sur la console

System.out.print(*chaîne de caractères à afficher*)
System.out.println(*chaîne de caractères à afficher*)

- chaîne de caractères peut être :

- une constante chaîne de caractères (String)

System.out.println("coucou");

- une expression de type String

System.out.println(age);

Ici age est une variable de type int
Elle est automatiquement convertie en String

- une combinaison (concaténation) de constantes et d'expressions de type String. La concaténation est exprimée à l'aide de l'opérateur +

*System.out.println("L'age de la personne est " +
age + " son poids " + poids);*

age (int) et poids (float) sont automatiquement converties en String

Syntaxe du langage Java

Entrées/sorties sur console

Lecture de valeurs au clavier

- utiliser la **classe *Scanner*** du package standard *java.util* (version *JDK 1.5* et supérieur).
- Elle simplifie la lecture de données sur l'entrée standard (clavier) ou dans un fichier.
- Pour utiliser la classe `Scanner`, il faut d'abord l'importer :
 - `import java.util.Scanner;`
- Ensuite il faut créer un objet de la classe `Scanner` :
 - `Scanner sc = new Scanner(System.in);`

Syntaxe du langage Java

Entrées/sorties sur console

Lecture de valeurs au clavier

- Pour récupérer les données, il faut faire appel sur l'objet **sc** aux méthodes décrites ci-dessous. Ces méthodes parcourent la donnée suivante lue sur l'entrée et la retourne :
 - `String next()` : donnée de la classe `String` qui forme un mot,
 - `String nextLine()` : donnée de la classe `String` qui forme une ligne,
 - `boolean nextBoolean()` : donnée booléenne,
 - `int nextInt()` : donnée entière de type `int`,
 - `double nextDouble()` : donnée réelle de type `double`
- Il peut être utile de vérifier le type d'une donnée avant de la lire :
 - `boolean hasNext()` : renvoie `true` s'il y a une donnée à lire
 - `boolean hasNextLine()` : renvoie `true` s'il y a une ligne à lire
 - `boolean hasNextBoolean()` : renvoie `true` s'il y a un booléen à lire
 - `boolean hasNextInt()` : renvoie `true` s'il y a un entier à lire
 - `boolean hasNextDouble()` : renvoie `true` s'il y a un double à lire.

Syntaxe du langage Java

Méthodes

- Méthodes \Leftrightarrow fonctions / procédures
 - Pour factoriser du code
 - Pour structurer le code
 - *Pour servir de «sous programmes utilitaires» aux autres méthodes de la classe*
 - ...
- En java plusieurs types de méthodes
 - Opérations sur les objets (cf. envois de messages)
 - Opérations statiques (méthodes de classe)
 - *Exemples*
Math.random();
LectureClavier.lireEntier();
- Pour le moment nous ne nous intéresserons qu'au second type

Syntaxe du langage Java

Méthodes statiques :

Déclaration

- «Une déclaration de méthode définit du code exécutable qui peut être invoqué, en passant éventuellement un nombre fixé de valeurs comme arguments»

The Java Language Specification J. Gosling, B Joy, G. Steel, G. Bracha

- Déclaration d'une méthode statique

static <typeRetour> nomMethode (<liste de paramètres>)

```
{  
  <corps de la méthode>  
}
```

• exemple

```
static double min(double a, double b) {  
    if (a < b)  
        return a;  
    else  
        return b;  
}
```

Signature de la méthode

Syntaxe du langage Java

Méthodes statiques :

Déclaration

- Déclaration d'une méthode statique

```
static <typeRetour> nomMethode (<liste de paramètres>)  
    {  
        <corps de la méthode>  
    }
```

<typeRetour>

- Quand la méthode renvoie une valeur (fonction) indique le type de la valeur renvoyée
 - *static* **double** min (double a, double b)
 - *static* **int** [] premiers (int n)
- void si la méthode ne renvoie pas de valeur (procédure)
 - *static* void afficher (double [][] m)

Syntaxe du langage Java

Méthodes statiques :

Déclaration

- Déclaration d'une méthode statique

```
static <typeRetour> nomMethode (<liste de paramètres>)  
{  
    <corps de la méthode>  
}
```

(<liste de paramètres>)

- vide si la méthode n'a pas de paramètres
 - `static int lireEntier()`
 - `static void afficher()`
- une suite de couples *type identificateur* séparés par des virgules
 - `static double min(double a, double b)`
 - `static int min (int[] tab)`

Syntaxe du langage Java

Méthodes statiques :

Déclaration

- Déclaration d'une méthode statique
static <typeRetour> nomMethode (*<liste de paramètres>*)
{
 <corps de la méthode>
}
<corps de la méthode>
• suite de déclarations de variables locales et d'instructions

```
static double min(double a, double b) {  
    double vMin; ← Variable locale  
    if (a < b)  
        vMin = a;  
    else  
        vMin = b;  
    return vMin; ← Instruction de retour  
}
```

Syntaxe du langage Java

Méthodes statiques :

Déclaration

<corps de la méthode>

- suite de déclarations de variables locales et d'instructions
- Les variables locales sont des variables déclarées à l'intérieur d'une méthode
 - elles conservent les données qui sont manipulées par la méthode
 - elles ne sont accessibles que dans le bloc dans lequel elles ont été déclarées, et leur valeur est perdue lorsque la méthode termine son exécution

```
static void method1 (...) {  
    int i;  
    double y;  
    int[] tab;  
    ...  
}
```

Possibilité d'utiliser le même identificateur dans deux méthodes distinctes
pas de conflit, c'est la déclaration locale qui est utilisée dans le corps de la méthode

```
static double method2 (...) {  
    double x;  
    double y;  
    double[] tab;  
    ...  
}
```

Syntaxe du langage Java

Méthodes statiques :

Déclaration

- *si la méthode à un type de retour le corps de la méthode doit contenir **au moins** une instruction **return** expression...*

```
static boolean contient(int[] tab, int val b) {
```

```
    boolean trouve = false;
    int i = 0;
    while ((i < tab.length) && (! trouve)) {
        if (tab[i] == val)
            trouve = true;
        i++;
    }
    return trouve;
```

```
}
```

```
for (int i = 0; i < tab.length; i++)
    if (tab[i] == val)
        return true;

return false;
```

Possibilité d'avoir plusieurs instructions **return**

- Lorsqu'une instruction **return** est exécutée retour au programme appelant
- Les instructions suivant le **return** dans le corps de la méthode ne sont pas exécutées

Syntaxe du langage Java

Méthodes statiques :

Déclaration

- **Return** sert aussi à sortir d'une méthode sans renvoyer de valeur (méthode ayant **void** comme type retour)

```
static void afficherPosition(int[] tab, int val) {  
  
    for (int i = 0; i < tab.length; i++)  
        if (tab[i] == val){  
            System.out.println("La position de " + val + " est " + i);  
            return;  
        }  
  
    System.out.println(val + " n'est pas présente dans le tableau");  
}
```


Syntaxe du langage Java

Méthodes statiques :

Invocation

déclaration	appel (invocation)
<pre>static void afficher() { ... }</pre>	<pre>afficher()</pre>
<pre>static double min(double a, double b) { ... }</pre>	<pre>min(10.5,x) avec double x min(x + y * 3,i) avec double x,y int i</pre>
<pre>static boolean contient(int[] tab,int val){ ... }</pre>	<pre>contient(tab1,14) avec int[] tab1 = new int[10] contient(tab2,i) avec int[] tab2 = new int[60] int i</pre>

Syntaxe du langage Java

Méthodes statiques :

- Toute méthode statique d'une classe peuvent être invoquée depuis n'importe quelle autre méthode statique de la classe
 - *l'ordre de déclaration des méthodes n'a pas d'importance*
- Pour invoquer méthode d'une autre classe il faut la préfixer par **NomClasse**.

```
LectureClavier.lireEntier();
```

```
Math.random();
```

```
Arrays.sort(monTableau);
```

- Possibilité d'avoir des méthodes avec des signatures identiques dans des classes différentes

Invocation

```
public class A {  
    static void m1() {  
        ...  
    }  
    static void m2() {  
        m1();  
        m3();  
    }  
    static void m3() {  
        ...  
    }  
}
```

Ne pas oublier import pour les classes d'un autre package que java.lang

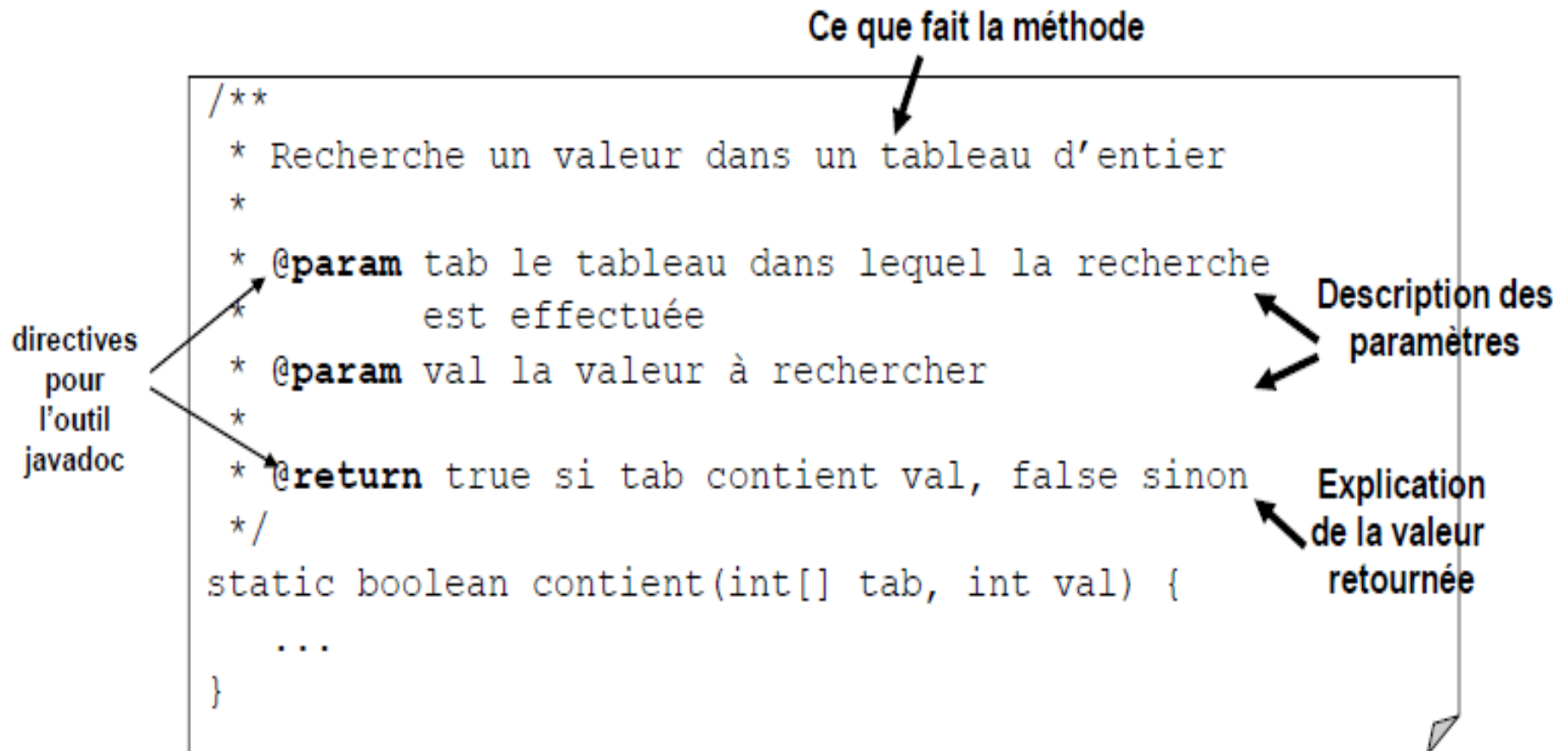
```
import  
java.util.Arrays
```

Syntaxe du langage Java

Méthodes statiques :

Commentaires

- Toute déclaration de méthode doit **TOUJOURS** être précédée de son commentaire documentant (exploité par l'outil javadoc)



Syntaxe du langage Java

Méthodes statiques :

Surcharge

La surcharge survient lorsque on a deux méthodes du même nom mais qui ne prennent pas les mêmes paramètres.

Voici un exemple de surcharge de méthode :

```
public class Test{
    public Test(){
        MaMethode();
        MaMethode(50);
    }
    public void MaMethode(int variable) {
        System.out.println("Le nombre que vous avez passé en paramètre vaut : "
+variable);
    }
    public void MaMethode(){
        System.out.println("Vous avez appelé la méthode sans paramètre ");
    }
}
```

- La définition d'une fonction ou procédure peut s'écrire en fonction d'elle-même
- Par exemple, on peut définir la fonction factorielle ($n! = 1 \times 2 \times 3 \times \dots \times n$) par :
 - $0! = 1$
 - $n! = (n - 1)! \times n$
- Ainsi, $4! = 3! \times 4 = (2! \times 3) \times 4 = ((1! \times 2) \times 3) \times 4$
 $= (((0! \times 1) \times 2) \times 3) \times 4$
 $= (((1 \times 1) \times 2) \times 3) \times 4 = ((1 \times 2) \times 3) \times 4$
 $= (2 \times 3) \times 4 = 6 \times 4 = 24$

$n!$ en Java

```
static long factorielle(int n) {  
    if (n == 0) return 1;  
    return factorielle(n - 1) * n;  
}
```

Syntaxe du langage Java

Le passage de paramètres

- En java, tous les paramètres sont passés par valeur.
- Lorsque une méthode est appelée de nouvelles variables locales sont créées et toute modification ne concernera que ces dernières.
- Certains programmeurs pensent à tort que les variables de types primitifs sont passées par valeur alors que les objets sont passés par référence.
- un contre-exemple

Syntaxe du langage Java

Le passage de paramètres

Pile d'exécution

- Une pile en mémoire est utilisée pour empiler les espaces mémoire des fonctions en cours d'exécution
- Dès qu'une fonction est appelée, son espace mémoire est empilé
- Quand son exécution est terminée, son espace mémoire est dépilé
- Nous allons donc simuler le passage par valeur (**la seule possibilité en Java**) sur l'exemple précédent (méthode echanger)
- Nous verrons ensuite ce qu'aurait donné un passage par adresse

Tableaux en Java

Tableaux

- En java, les tableaux sont statiques
- Les tableaux sont considérés comme des objets
- Fournissent des collections ordonnées d'éléments
- Les éléments d'un tableau peuvent être
 - Des variables d'un type primitif (**int**, **boolean**, **double**, **char**, ...)
 - Des références sur des objets (à voir dans la partie Classes et Objets)
 - Tous les éléments doivent avoir le même type
 - Il faut préciser une taille qu'on ne peut dépasser
- Création d'un tableau
 1. Déclaration : déterminer le type du tableau
 2. Dimensionnement : déterminer la taille du tableau
 3. Initialisation : initialiser chaque case du tableau

Tableaux en Java

Tableaux

1. Déclaration

- La déclaration précise simplement le type des éléments du tableau

int[] monTableau;

- Peut s'écrire également

int monTableau[];

monTableau *null*

Une déclaration de tableau ne doit pas préciser de dimensions

int monTableau[5]; // Erreur



Tableaux en Java

Tableaux

2. Dimensionnement

- Le nombre d'éléments du tableau sera déterminé quand l'objet tableau sera effectivement créé en utilisant le mot clé **new**
- La taille déterminée à la création du tableau est fixe, elle ne pourra plus être modifiée par la suite
- Longueur d'un tableau : **monTableau.length**

```
int[] monTableau;           // Déclaration  
monTableau = new int[3];    // Dimensionnement
```

- La création d'un tableau par **new**
 - Alloue la mémoire en fonction du type de tableau et de la taille
 - Initialise le contenu du tableau à 0 pour les types simples



Tableaux en Java

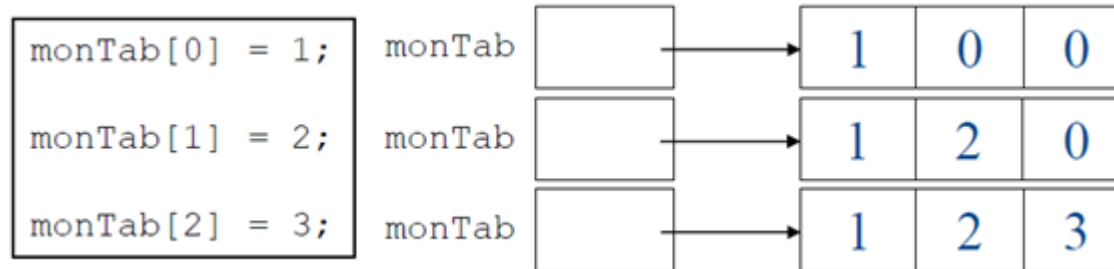
Tableaux

3. Initialisation

- En java les indices commencent à zéro
- L'accès à un élément d'un tableau s'effectue suivant cette forme:

```
monTab[varInt]    // varInt >= 0 et < monTab.length
```

- Java vérifie automatiquement l'indice lors de l'accès (exception levée)



- Autre méthode : en donnant explicitement la liste de ses éléments entre {...}

```
int[] monTab = {1, 2, 3}
```

- est équivalent à :

```
monTab = new int[3];  
monTab[0] = 1; monTab[1] = 2; monTab[2] = 3;
```

Tableaux en Java

Tableaux

Synthèse

1. Déclaration

```
int[] monTableau;
```

2. Dimensionnement

```
monTableau = new int[3];
```

3. Initialisation

```
monTableau[0] = 1;
```

```
monTableau[1] = 2;
```

```
monTableau[2] = 3;
```

- Ou 1, 2 et 3

```
int[] monTab = {1, 2, 3};
```

L'exception

```
Exception in thread "main" java.lang.  
ArrayIndexOutOfBoundsException: 2  
    at org.eclipse.classes.FirstClass.main(  
        FirstClass.java:11)
```

```
for (int i = 0; i < monTableau.length; i++) {  
    System.out.println(monTableau[i]);  
}
```

```
for (int current : monTableau) {  
    System.out.println(current);  
}
```

Tableaux en Java

Tableaux

ForEach

- Boucle « *foreach* » introduite depuis la version 1.5 de Java peut être utilisée pour itérer sur des tableaux

```
/**
 * calcule somme des éléments d'un tableau d'entiers
 * @param a le tableau d'entier
 * @return la somme des éléments de a
 */
public int sum(int[] a) {
    int sum = 0;
    for (int i = 0; i < a.length; i++){
        sum += a[i];
    }
    return sum;
}
```

Boucle for traditionnelle

Avec le nouveau
type de boucle for

Lire : pour chaque x de a

```
public int sum(int[] a) {
    int sum = 0;
    for (int x : a){
        sum += x;
    }
    return sum;
}
```

Tableaux en Java

Tableaux: Multidimensionnels

- tableau dont les éléments sont eux mêmes des tableaux
- un tableau à deux dimensions se déclarera ainsi de la manière suivante :

typeDesElements[][] nomduTableau;

● *exemples*

- `double[][] matrice;`
- `Voxel[][][] cubeVoxels;`

Tableaux en Java

Tableaux: Multidimensionnels

- dimensions du tableau

- *ne sont pas spécifiées à la déclaration (comme pour les tableaux à une seule dimension).*
- *indiquées que lors de la création*
 - *obligatoire que pour la première dimension.*
 - *autres dimensions peuvent n'être spécifiées que lors de la création effective des tableaux correspondants.*

```
double [][] matrice = new double[4][4];
```

**Création d'une matrice
4x4 de réels**

```
double [][] matrice = new double[4][];  
for (int i=0; i < 4; i++)  
    matrice[i] = new double[4];
```

**Les 3 écritures sont
équivalentes**

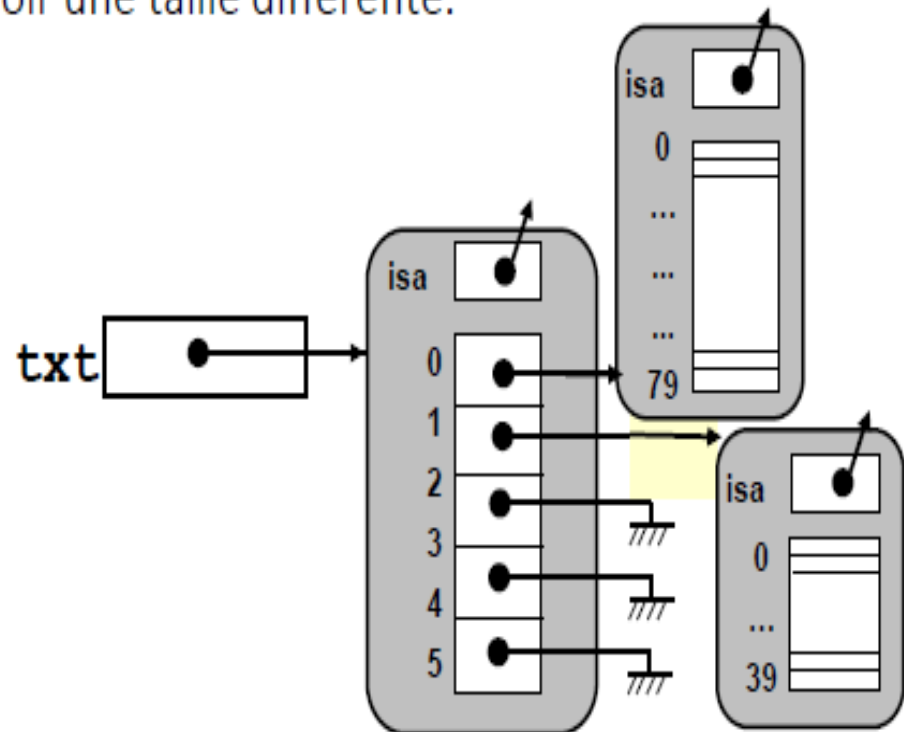
```
double [][] matrice;  
matrice = new double[4][];  
for (int i=0; i < 4; i++)  
    matrice[i] = new double[4];
```


Tableaux en Java

Tableaux: Multidimensionnels

- chaque tableau imbriqué peut avoir une taille différente.

```
char [][] txt;  
txt = new char[6][];  
txt[0] = new char[80];  
txt[1] = new char[40];  
txt[2] = new char[70];  
...
```



Tableaux en Java

Tableaux: Multidimensionnels

- accès aux éléments d'un tableau multidimensionnel (exemple 2d)

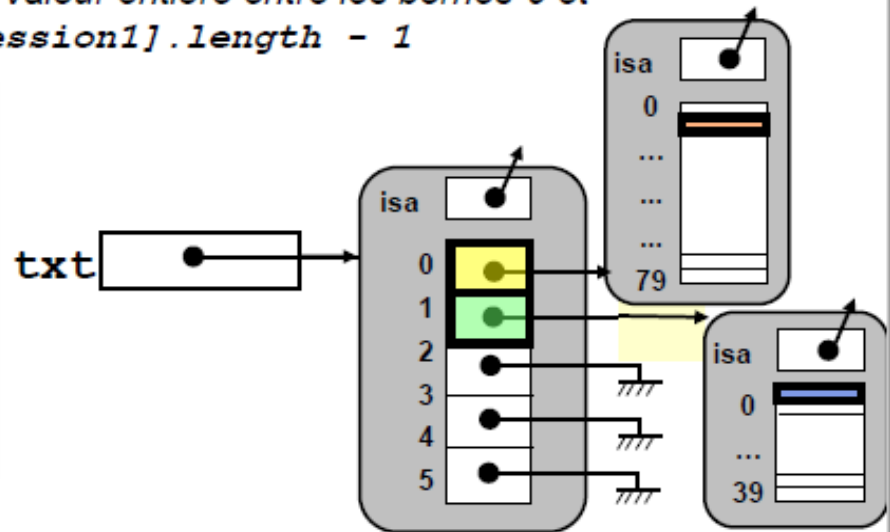
- accès à un élément d'un tableau s'effectue à l'aide d'une expression de la forme :

nomDuTableau[expression1][expression2]

où

- *expression1* délivre une valeur entière entre les bornes 0 et *nomDuTableau.length - 1*
- *expression2* délivre une valeur entière entre les bornes 0 et *nomDuTableau[expression1].length - 1*

```
char[][] txt = new char[5][];  
txt[0] = new char[80];  
txt[1] = new char[40];  
txt[0][1] = 'S';  
txt[1][0] = 'H';
```



Tableaux en Java

Tableaux: Multidimensionnels

- Comme pour tableaux unidimensionnels possible de créer un tableau multidimensionnel en donnant explicitement la liste de ses éléments à la déclaration (liste de valeurs entre accolades)

• *exemples :*

```
int[][] t1 = {  
    { 1, 2, 3, 4, 5},  
    { 6, 7, 8, 9, 10},  
};
```

```
int[][] t1 = new int[2][5];  
t1[0][0] = 1; t1[0][1] = 2; ...  
t1[1][0] = 6; t1[1][1] = 7;...
```

Tableaux en Java

la classe Arrays

package **java.util** définit une classe, **Arrays**, qui propose des méthodes statiques (de classe) pour le tri et la recherche dans des tableaux.

Exemple : tri d'un tableau

```
/// tableau de 1000 réels tirés au hasard
// dans l'intervalle [0..1000[
double[] vec= new double[5];
for (int i = 0; i < vec.length; i++) {
    vec[i] = Math.random()*10;
}
// tri du tableau
Arrays.sort(vec);
// affiche le tableau trié
for (int i = 0; i < vec.length; i++) {
    System.out.print(vec[i] + " ");
}
```

Tableaux en Java

la classe Arrays

- pour chaque type de tableau
 - *Des méthodes de recherche*
 - `int binarySearch(char[] a) , int binarySearch(int[] a)`
`... int binarySearch(Object[] a)`
 - *Des méthodes de tris*
 - `sort(char[] a) , sort(int[] a) sort(Object[] a)`
 - `sort(char[] a, int fromIndex, int toIndex) , ...`
 - *Des méthodes pour remplissage avec une valeur*
 - `fill(char[] a, char val) , fill(int[] a, long val)`
 - `fill(char[] a, char val, int fromIndex, int toIndex) , ...`
 - *Des méthodes de test d'égalité*
 - `boolean equals(char[] a1, char[] a2),`
 - `boolean equals(int[] a1,int[] a2), ...`
- Les tableaux sont des structures de données élémentaires
- Le package **java.util** contient plein de classes « sympa » pour la gestion de structures de données plus évoluées (collections) :
 - *listes*
 - *ensembles*
 - *arbres*

Types énumérés en Java

Types énumérés

Parfois, on a besoin d'une liste de valeurs possibles

exemple : les tailles des vêtements XS, S, M, L, et XL.

utiliser ces symboles et leur associer une valeur (par exemple un **final static int**).

Mais on ne pourra pas utiliser directement comme un type Taille

type énuméré

Types énumérés en Java

Types énumérés

- Pour créer le type énuméré Size, on va donc écrire dans le fichier **Size.java**
- le code ci-dessous :

```
public enum Size{  
    XS,  
    S,  
    M,  
    L,  
    XL  
}
```

- Size est un nouveau type
- Les valeurs sont Size.XS, Size.S, Size.M, Size.L, et Size.XL.
- un type **enum** hérite de la classe Enum, qui possède donc des méthodes
 - values() retourne la liste de valeurs possibles.
 - ordinal() retourne la position de l'instance dans la déclaration du type énuméré

Types énumérés en Java

Types énumérés

- Exemple

```
public static void main(String[] args) {  
  
    for (Size taille: Size.values()){  
        if (taille==Size.L)  
            System.out.println("C'est ma taille : "+taille);  
        else  
            System.out.println(taille + " n'est pas ma taille");  
        }  
    }
```


Types énumérés en Java

Types énumérés

- Exemple avec **Switch**

```
public static void main(String[] args) {  
    Size maTaille = Size.L;  
    double prix = 0;  
  
    switch(maTaille){  
        case XS: prix=4; break;  
        case S: prix = 5; break;  
        case M: prix = 7; break;  
        case L: prix = 9; break;  
        case XL: prix = 10; break;  
    }  
    System.out.println("Le prix est : " + prix);  
}
```