Programmation Orientée Objet

Initiation à Java

Saber HENI
saber.heni02@univ-paris8.fr

Plan du cours

- Chapitre 1 : Introduction au langage Java
- Chapitre 2 : Classes et objets
- Chapitre 4 : Concepts de base de l'OO
- Chapitre 5 : La gestion des exceptions
- Chapitre 6 : Les classes de base dans Java
- Chapitre 7 : Les collections

UNIVERSITÉ PARIS 8
VINCENNES-SAINT-DENIS

Chapitre 1

Introduction au langage Java

Qu'est ce que « Java » ?

- Java est un langage de programmation moderne développé par Sun Microsystems (aujourd'hui racheté par Oracle).

- Un langage de programmation interprété **et** compilé.
  - Langage portable : pseudo-code

- Il ne faut surtout pas le confondre avec JavaScript car Java n'a rien à voir.

Quels types d'application pour « java » ?

- On peut faire de nombreuses sortes de programmes avec Java :
  - des applications, sous forme de fenêtre ou de console ;
  - des applets, qui sont des programmes Java incorporés à des pages web ;
  - des applications pour appareils mobiles, avec J2ME ;
  - et bien d'autres ! J2EE, JMF, J3D pour la 3D...

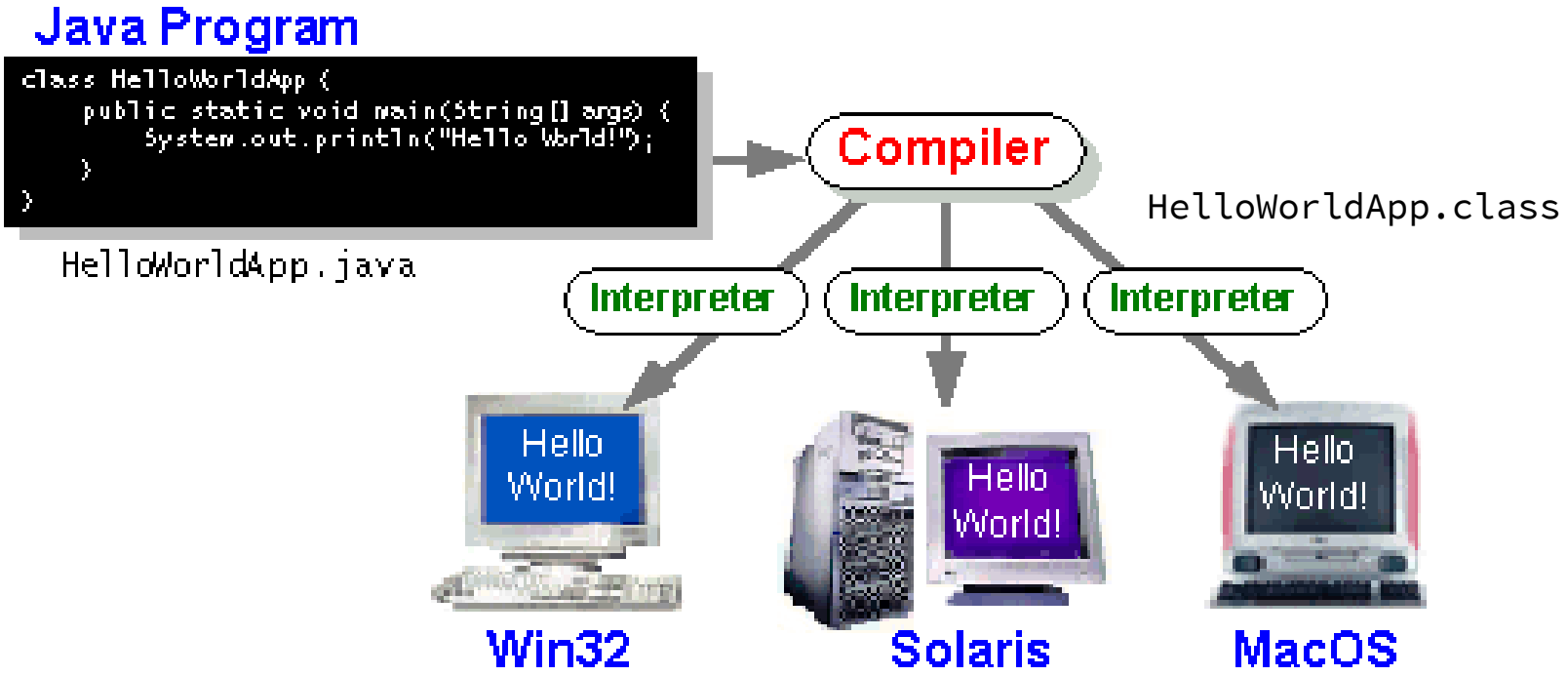
Les différentes versions de Java

- 1996 : JDK 1.0
- 1997 : JDK 1.1
- 1998 : JDK 1.2
- 2000 : JDK 1.3
- 2002 : JDK 1.4
- 2004 : JDK 1.5 (Java 5)
- 2007 : JDK 1.6 (Java 6)
- 2011 : JDK 1.7 (Java 7)
- 2014 : JDK 1.8 (Java 8)

Quelques atouts de java

- Les principaux atouts de java :
  - génère des applications portables,
  - un langage simple et puissant,
  - un langage qui introduit directement la notion de thread,
  - une API très riche,
  - une gestion automatique de la mémoire.

Comment cela fonctionne t'il ?



- L'interprète est une machine virtuelle plus connue sous le nom de « jvm ».

JVM, JRE et JDK?

- JVM
  - La "JAVA virtual machine", ou encore "machine virtuelle JAVA" permet d'interpréter et d'exécuter du code JAVA.
- JRE
  - Le "JAVA Runtime Environment", est traduit par "Environnement d'exécution JAVA". Le JRE contient une JVM et des classes standard de la plate-forme Java et des bibliothèques Java de prise en charge.
- JDK ou SDK
  - JDK pour "JAVA Development Kit", contient en plus d'un JRE, un compilateur java (javac). C'est le compilateur, qui transforme le code programmé en code qui est interprété par la machine virtuelle.

Environnement de développement

- Nous allons utiliser tout au long du cours les outils suivants :
  - Notepad++ (Sous Windows) et gedit (Sous Linux) comme éditeurs pour écrire du code Java.
  - La ligne de commande pour la compilation et l'exécution du code.

- Télécharger et installer le JDK 1.7.
  - Aller sur le site d'Oracle : http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html
- Mettre à jour votre variable d'environnement %PATH% en rajoutant à la fin l'URL du répertoire bin du JDK
  - Exemple d'URL du JDK : C:\Program Files\Java\jdk1.7.0_51\bin

Exemple de compilation - 1

- Il faut savoir qu'en Java :
  - Le code source s'écrit dans une structure qu'on appelle classe.
  - Un fichier source a pour extension **.java**
  - Tous les programmes Java sont composés d'au moins une classe. Elle doit contenir une méthode appelée **main** : ce sera le point de démarrage de notre programme.
  - Un fichier source doit avoir le même nom que la classe publique qu'il contient.
- Soit le code suivant :

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

Exemple de compilation - 2

- Pour compiler
  - Ouvrez une console
  - Positionnez-vous dans le répertoire contenant le fichier source.
  - Tapez

```
>javac Bonjour.java
```

- Pour voir le résultat de la compilation
  - Afficher le contenu du répertoire de compilation.

```
>dir
Bonjour.java
Bonjour.class
```

- Pour excécuter

```
>java Bonjour
Bonjour le monde!
```

Conventions de codage

- Un nom de classe ou interface doit commencer par une majuscule.
  - Ex: class Point, class MasterHandi
- Un nom d'attribut doit commencer par une minuscule.
  - Ex: loopCounter
- Une constante s'écrit en majuscules.
  - Ex: PI, ERROR_MESSAGE

NB :
- ❑ JAVA est sensible à la casse.
- ❑ Ne pas utilisé les noms réservés comme noms de variables ou noms de classes.

Liste des mots réservés (Keywords)

abstract	continue	for	new	switch
assert(3)	default	goto(1)	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum(4)	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp(2)	volatile
const(1)	float	native	super	while

(1) Inutilisé
(2) Ajouté dans la version 1.2
(3) Ajouté dans la version 1.4
(4) Ajouté dans la version 5.0

Littéraux :
`true, false, et null`

Les commentaires

UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS

Commentaire sur une seule ligne

```
// commentaire sur une seule ligne
int N=1; // déclaration du compteur
```

Commentaire sur plusieurs lignes

```
/* commentaires ligne 1
    commentaires ligne 2 */
```

Commentaire de documentation automatique

```
/**
  * commentaire de la methode
  * @param val la valeur à traiter
  * @since 1.0
  * @return la valeur de retour
  * @deprecated Utiliser la nouvelle methode XXX
  */
```

Les variables et les opérateurs

- Une variable est un élément qui stocke des informations de toute sorte en mémoire.
- Le langage binaire (Langage machine) est donc une suite de 0 et de 1 qu'on appelle bit.

Raccourcis	Traduction	Correspondance
b	Bit	C'est la plus petite valeur informatique : soit 0 soit 1
o	Octet	regroupement de 8 bits, par exemple : 01011101
Ko	Kilo Octet	regroupement de 1024 octets
Mo	Mega Octet	regroupement de 1024 ko
Go	Giga Octet	regroupement de 1024 Mo
To	Tera Octet	regroupement de 1024 Go

Les différents types de variables

- Déclaration d'une variable
  - `<Type de la variable> <Nom de la variable> ;`
  - `Puis vous pouvez sur la même ligne affecter une valeur à la variable sinon vous pouvez le faire plus tard.`
- En Java, nous avons deux types de variables :
  - des variables de type simple ou « primitif » ;
  - des variables de type complexe ou des « objets ».
- En Java, toute instruction se termine par un point-virgule « ; »

Les identificateurs en Java

- Un identificateur est une suite de caractères parmi :
  - les lettres (minuscules ou majuscules),
  - Les chiffres,
  - le “blanc souligné” (_),
  - le dollar ($) {Java accepte aussi les autres symboles de devises}.

- Un identificateur ne peut pas commencer par un chiffre.
  - Après la premier caractère, on peut mettre autant de chiffre qu’on veut.

- Une variable est identificateur Java, il suit les mêmes règles et adhère à la convention de code que nous avons vu au début du cours.

Les types primitifs - 1

type	taille (en bits)	intervalle des valeurs	valeur par défaut
byte	8 (1 o)	$[-2^7;2^7[$	0
short	16 (2 o)	$[-2^{15}; 2^{15}[$	0
int	32 (4 o)	$[-2^{31};2^{31}[$	0
long	64 (8 o)	$[-2^{63};2^{63}[$	0
char	16 (2 o)	[\u0000; \uffff]	\u0000
boolean	Inconnue	{false, true}	false
float	32 (4 o)	Dépend de la plateforme	0
double	64 (8 o)	Dépend de la plateforme	0

Littéraux en Java - 1

- 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, octale, hexadécimale ou Binaire (Java 7).
  - 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 la valeur par la lettre l ou L.

```
/* Les littéraux suivants sont
de type int et représentent
l'entier 45 */

45 // en décimal
055 // en octal
0x2d // en hexadécimal
0b101101 // en binaire
```

```
/* Les littéraux suivants
sont de type long et
représentent l'entier 45 */

45L // en décimal
055L // en octal
0x2dL // en hexadécimal
0b101101L // en binaire
```

Littéraux en Java - 2

- Les littéraux flottants sont considérés par défaut comme étant des valeurs de type double.
- On peut ajouter la lettre f ou F derrière le nombre pour qu'ils soient de type **float** et d ou D pour qu'ils soient de type **double.**

```
// Les littéraux suivants sont des doubles et représentent 25.5
25.5 // en notation décimale
2.55e+1 // en notation scientifique

// Les littéraux suivants sont de type float et représentent
25.5
25.5F // en notation décimale
2.55e+1F // en notation scientifique

// Le littéral suivant est de type double et représente 25.0
25D
```

Le type String

- Le type String permet de gérer les chaînes de caractères, c'est-à-dire le stockage de texte.
- Il s'agit d'une variable d'un type plus complexe que l'on appelle objet. Celle-ci s'utilise un peu différemment des variables précédentes :
- `String` commence par une majuscule ! Et lors de l'initialisation, on utilise des guillemets doubles (« " " »).

```
//Première méthode de déclaration
String phrase =  "Ceci est une chaîne";
//Deuxième méthode de déclaration
String chaine = new String ("Une autre chaîne");
```

Les opérateurs - 1

- Java fournit les opérateurs arithmétiques suivant :
  - « + » : permet d'additionner deux variables numériques (mais aussi de concaténer des chaînes de caractères).
  - « - » : permet de soustraire deux variables numériques.
  - « * » : permet de multiplier deux variables numériques.
  - « / » : permet de diviser deux variables numériques.
  - « % » : permet de renvoyer le reste de la division entière de deux variables de type numérique ; cet opérateur s'appelle le **modulo.**

```
int nbre1, nbre2, nbre3;
nbre1 = 1 + 3;
nbre2 = 2 * 6;
nbre3 = nbre2 / nbre1;
nbre1 = 5 % 2;
nbre2 = 99 % 8;
nbre3 = 6 % 3;
```

Les opérateurs - 2

- Un autre exemple

```
int nbre1, nbre2, nbre3;
nbre1 = nbre2 = nbre3 = 0;

nbre1 = nbre1 + 1;
nbre1 = nbre1 + 1;
nbre2 = nbre1;
nbre2 = nbre2 * 2;
nbre3 = nbre2;
nbre3 = nbre3 / nbre3;
nbre1 = nbre3;
nbre1 = nbre1 - 1;
```

Les opérateurs - 3

- Il existe une syntaxe qui raccourcit l'écriture de ce genre d'opérations.
  - Affectation.
  - Post-incrémentation.
  - Pré-incrémentation.

```
int nbre1, nbre2, nbre3, nbre4;
nbre1 = nbre2 = nbre3 = nbre4 = 0;

nbre1 = nbre1 + 1;
nbre2 += 1;
nbre3++; // Post-incrémentation
++nbre4; // Pré-incrémentation
//Affichez les trois nombres
```

Les opérateurs - 4

- Un autre exemple (Post et Pré-incrémentation)

```
int n1=0;
int n2=0;
System.out.println("n1 = " + n1 + " n2 = " + n2);

n1=n2++;
System.out.println("n1 = " + n1 + " n2 = " + n2);

n1=++n2;
System.out.println("n1 = " + n1 + " n2 = " + n2);

n1=n1++;          //attention
System.out.println("n1 = " + n1 + " n2 = " + n2);
```

Les opérateurs - 5

- Opérateurs d'affectation

N°	Opérateur	Exemple	Equivalence	Résultat
1	=	i = 90		**i = 90**
2	+=	i += 20	i = i + 20	**i = 110**
3	-=	i -= 10	i = i - 10	**i = 100**
4	*=	i *= 2.5	i = i * 2.5	**i = 250**
5	/=	i /= 10	i = i / 10	**i = 25**
6	%=	i %= 10	i = i % 10	**i = 5**
7	^=	i ^= 2	i = i ^ 2	**i = 25**
8	++	i++ ou ++i	i = i + 1 et i += 1	**i = 26**
9	--	i-- ou --i	i = i - 1 et i -= 1	**i = 25**

Les opérateurs - 6

Opérateurs logiques

Opérateur	Signification
x > y	Strictement supérieur
x < y	Strictement inférieur
x >= y	Supérieur ou égal
x <= y	Inférieur ou égal
x == y	Egalité
x != y	Différent

Opérateur	Signification
x & y	ET binaire
x ^ y	OU exclusif binaire
x | y	OU binaire
x && y	ET logique (Court-circuit)
x || y	OU logique (Court-circuit)
x & y	ET logique
x | y	OU logique
x ? y : z	L'opérateur ternaire
!x	L'opérateur de négation

La priorité des opérateurs

- Java définit les priorités dans les opérateurs comme suit (du plus prioritaire au moins prioritaire)

Les parenthèses	( )
les opérateurs d'incrémentation	++ --
les opérateurs de multiplication, division, et modulo	* / %
les opérateurs d'addition et soustraction	+ -
les opérateurs de comparaison	< > <= >=
les opérateurs d'égalité	== !=
l'opérateur OU exclusif	^
l'opérateur ET	&
l'opérateur OU	|
l'opérateur ET logique	&&
l'opérateur OU logique	||
les opérateurs d'assignement	= += -=

OU, ET & OU Exclusif binaire

- L'opérateur AND (&) bit à bit retourne la valeur un si les deux bits correspondants des opérandes d'entrée sont à un ; sinon il retourne la valeur zéro.

- L'opérateur OR (|) bit à bit retourne la valeur un si l'un des deux bits correspondants des opérandes d'entrée est à un et retourne la valeur zéro dans le cas où les deux bits sont à zéro.

- L'opérateur EXCLUSIVE OR, ou XOR (^), retourne la valeur un si l'un des deux bits correspondants des opérandes est à un, mais pas les deux.

OU & ET logiques

- En travaillant avec les opérateurs logiques on rencontre un comportement appelé « court-circuit » (&&, ||).
  - Cela signifie que l'évaluation de l'expression sera poursuivie jusqu'à ce que la vérité ou la fausseté de l'expression soit déterminée sans ambiguïté.
  - Certaines parties d'une expression logique peuvent ne pas être évaluées.
- Dans le cas contraire (&, |), l'expression logique sera évaluée en totalité même si la vérité ou la fausseté de l'expression est déjà déterminée.

Et et OU logiques

Exemple (Court-circuit)

```
int i = 5;
int j = 2;
if ((i > 2) || (++j >= 3)) {
        System.out.println (j);
}
```

Exemple (ET et OU logiques)

```
int i = 5;
int j = 2;
if ((i > 2) | (++j >= 3)) {
        System.out.println (j);
}
```

L'opérateur conditionnel ternaire

- Syntaxe
  - `( condition ) ? valeur-vrai : valeur-faux;`
  - Les opérateurs conditionnels ternaires peuvent être imbriqués.

Exemple d'utilisation

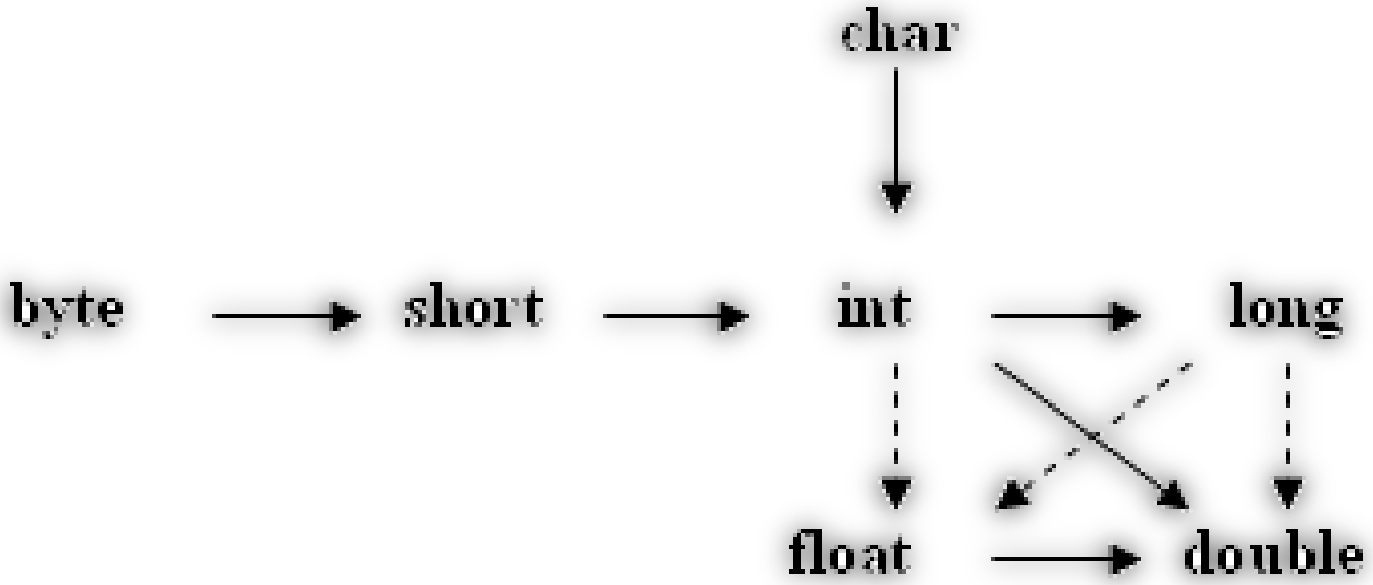
```
int i = 5;
int j = 2;
int max = (i >= j)? i : j;
```

Conversions / transtypage

- Il est parfois intéressant de *convertir* une donnée d'un type primitif vers un autre type primitif.
- Mais il faut faire attention lors de ces conversions car il y a risque de perdre de l'information.
- Java permet la conversion des types ou en d'autre termes, le transtypage.
  - Conversion sans perte d'information
  - Conversion avec perte d'information

Conversion sans perte d'information - 1

- Les conversions vers un type dont la taille de la représentation est plus grande ou égale se font automatiquement sans avoir besoin d'être spécifié.



Conversions automatiques sans perte d'information

Conversions automatiques avec une possible perte d'information

Conversion sans perte d'information - 2

Exemple

```
int n = 123456789;
float f = n; // Perte de précision (int -> float)

System.out.println (f); // f vaut 1.23456792 E8

double d = f; // Aucune perte de précision (float -> double)

long l = n; // Aucune perte de précision (int -> long)
double d2 = 'a'; // Aucune perte de précision (char -> double)
```

Conversion avec perte d'information - 1

- Il y a 22 conversions avec perte d'information.

De	Vers
short	byte ou char
char	byte ou short
int	byte, short ou char
long	byte, short, char ou int
float	byte, short, char, int ou long
double	byte, short, char, int, long ou float

- la conversion du type short vers le type char provoque une perte d'information alors qu'ils utilisent tous les deux 16 bits en mémoire

Conversion avec perte d'information - 2

Exemple 1

```
double i = 1.23;
double j = 2.9999999;
int k = (int)i; //k vaut 1
k = (int)j; //k vaut 2
```

Exemple 2

```
int x = 292;
float f = (float) x; // Conversion sans perte d'information
byte b = (byte) x; // Conversion avec perte d'information
char c = (char) b; // Conversion d'un byte vers un char

System.out.println (f);
System.out.println (b);
System.out.println (c);
System.out.println ((byte) c);
```

Promotion arithmétique - 1

- La *promotion arithmétique* intervient automatiquement lorsque certains opérateurs mathématiques doivent modifier leurs opérandes pour pouvoir exécuter l'opération

- Les **opérateurs arithmétiques** existent en quatre versions double, float, long et int.

Exemple

```
float num = 10.0F;
int denom = 2;
System.out.println (num / denom); // denom est converti en
un float
```

Promotion arithmétique - 2

- Les opérateur qui convertissent leurs opérandes sont :
  - Les opérateurs unaires **-, +, -- et ++**, ils convertissent les types char, byte et short en int automatiquement. Les restes conservent leurs types.
  - Les opérateurs **+, -, *, /, %, <, <=, >, >=, == et !=**. Il faut regarder le type des deux opérandes, et appliquer ces règles dans l'ordre :
    1. Si l'un des opérandes est de type double, l'autre est converti en double ;
    2. Si l'un des opérandes est de type float, l'autre est converti en float ;
    3. Si l'un des opérandes est de type long, l'autre est converti en long ;
    4. Dans tous les autres cas, les deux opérandes sont convertis en int.

Conversion par le compilateur

- Le compilateur est en fait capable de faire des conversions avec perte d'information implicitement. *(Si les tiers sont des littéraux et/ou des constantes)*
  - Seulement, s'il est sûr qu'une telle perte n'aura pas lieu.

Exemple

```
final byte b = 50;
final short s;
s = 25000;
short sum = b + s; // Erreur de compilation
```

Rappel : *les conversions avec perte d'information ne résultent pas toujours en une perte de précision*

Conversion en String

- L'opérateur + appliqué à deux objets de type String désigne la concaténation de chaînes de caractères.
  - Dès que l'un des deux opérandes est de type String, l'autre est converti en String.

Exemple

```
int i = 3;
System.out.println("i = " + i);
System.out.println("i = " + i + 5); // i = 35
```

- La chaîne de caractères constante "i = " est concaténée avec la valeur de la variable i convertie en chaîne de caractères.

Le branchement conditionnel – if – if else

Variante 1

```
if(//condition) {
        //Exécution des instructions si la condition est
        remplie
}
```

Variante 2

```
if(//condition) {
        //Exécution des instructions si la condition est
        remplie
} else {
        //Exécution des instructions si la condition n'est
        pas remplie
}
```

Le branchement conditionnel – else if

Varaiante 3

```
if(//condition) {
        //instructions
}
else if (//condition) {
        //instructions
}
else if (//condition) {
        //instructions
}
else {
        //instructions
}
```

- Entre le premier if et le dernier else, nous pouvons mettre zéro ou plusieurs « else if »

Le branchement conditionnel - exemple

Exemple

```
byte points = 13;
char grade = 'F';
if ((points  <= 20) && (points >=  18)) {
    grade = 'A';
} else {
    if ((points < 18) && (points >= 16)) {
        grade = 'B';
    } else {
        if ((points < 16) && (points >= 14)) {
            grade = 'C';
        } else {
            if ((points < 14) && (points >²= 12)) {
                grade = 'D';
            }
        }
    }
}
```

Le branchement conditionnel – switch - 1

```
switch (variable)
{
        case argument:
                //instructions;
        break;

        case argument:
                 //instructions
                ;
        break;

        default:      …;
}
```

- On ne peut utiliser switch qu'avec des types primitifs d'une taille maximum de 32 bits (byte, short, int, char).
- Depuis la version 5 de Java, l'instruction switch accepte les enums.
- Depuis la version 7 de Java, l'instruction switch accepte les objets de type String en paramètre

Le branchement conditionnel – switch – 2

Exemple

```
char sexe = 'F';
if (sexe == 'M') {
    System.out.println ("Vous êtes un homme");

} else if (sexe == 'F') {
    System.out.println ("Vous êtes une femme");

} else {
    System.out.println ("Vous êtes de sexe indéterminé");

}
```

Le branchement conditionnel - switch

Exemple équivalent au précédent

```
char sexe = 'F';
switch (sexe)
{
    case 'M':
        System.out.println ("Vous êtes un homme");
    break;

    case 'F':
        System.out.println ("Vous êtes une femme");
    break;

    default:
        System.out.println ("Vous êtes de sexe
indéterminé");
}
```

Les boucles - 1

- Le rôle des boucles est de répéter un certain nombre de fois les mêmes opérations
- Une boucle s'exécute tant qu'une condition est remplie

Boucle tant que

```
while (/* Condition */)
{
//Instructions à répéter
}
```

Boucle faire tant que

```
do
{
//Instructions à répéter
} while (/* Condition */);
```

Les boucles - 2

- La boucle for prend les attributs ci-dessous dans sa condition.
  - déclaration des variables et initialisation de ces variables ;
  - spécification de la condition d'arrêt ;
  - incrémentation des variables.

Boucle pour

```
for (initialisation; condition;  modification) {
        …
        // bloc d'instructions
        …
}
```

Les boucles - 3

Exemple 1

```
for(int i = 0 ; i < 10 ; i++){
        System.out.println("i = " + i);
}
```

Exemple 2

```
for(int i = 0, j = 2; (i < 10 && j < 6); i++, j+=2){
        System.out.println("i = " + i + ", j = " + j);
}
```

Exemple 3

```
for(;;){
        System.out.println("La boucle tourne infinément");
}
```

Les débranchements `break` et `continue` - 1

- **`Break`**
  - permet de quitter immédiatement une boucle ou un branchement. Utilisable dans tous les contrôles de flot
- **`Continue`**
  - s'utilise dans une boucle pour passer directement à l'itération suivante

- **`break`** et **`continue`** peuvent s'exécuter avec des blocs nommés.
  - Il est possible de préciser une étiquette pour indiquer le point de retour lors de la fin du traitement déclenché par le break.
  - Une étiquette est un nom suivi d'un caractère deux-points qui définit le début d'une instruction.

Les débranchements `break` et `continue` - 2

Exemple 1

```
for(int i = 0 ; i < 10 ; i++){
	if(i % 2 != 0) {
		continue;
	}
	System.out.println("i = " + i);
}
```

Exemple 2

```
for(int i = 0 ; i < 10 ; i++){
	if(i == 0) {
		break;
	}
	System.out.println("i = " + i);
}
```

Les débranchements break et continue - 3

Exemple 3

```
for(int i = 0 ; i < 10 ; i++){
        for(int j = 0 ; j < 10 ; j++){
                if(i < j) {
                        break;
                }
                 System.out.print("i = " + i);
                 System.out.println(" et j = " + j);
        }
        System.out.println("Inner loop Broken");
}
System.out.println("Outer loop Broken or finished");
```

Les débranchements break et continue - 4

Exemple 4

```
externe :
for(int i = 0 ; i < 10 ; i++){
        for(int j = 0 ; i < 10 ; i++){
                if(j > i) {
                        break;
                }

                System.out.print("i = " + i);
                System.out.println(" et j = " + j);
                if((i == 8)  && (i == j)) {
                        break externe;
                }

        }
        System.out.println("Inner loop Broken");
}
System.out.println("Outer loop Broken");
```

Les tableaux - 1

- Un tableau est une structure de données contenant un groupe d'éléments tous du même type.
  - Le type des éléments peut être un type primitif ou une classe.
  - Lors de la définition d'un tableau les [] spécifiant qu'il s'agit d'un tableau peuvent être placés avant ou après le nom du tableau.

Déclaration

```
<type du tableau> [] <nom du tableau>;
<type du tableau> <nom du tableau> [];
```

- **Exemples**

```
int ti []; // ti est un tableau à une dimension de int
char[] tc; // tc est un tableau à une dimension de char
```

Les tableaux - 2

- La déclaration ressemble beaucoup à celle d'une variable quelconque, si ce n'est la présence de crochets « [ ] »;
- On peut déclarer et initialiser un tableau dans une seule ligne en lui affectant un ensemble de valeurs entre deux accolades « { } ».

Déclaration et initialisation :

```
<type du tableau> [] <nom du tableau> = {v1,v2,…,vn};
    <type> [] <nom> = new <type> []{v1,v2,…,vn};
```

– **Exemples :**

```
int [] tableauEntier        = {0,1,2,3,4,5,6,7,8,9};
double [] tableauDouble     = {0.0,1.0,2.0,3.0,4.0,5.0,6.0};
char [] tableauCaractere    = {'a','b','c','d','e','f','g'};
String [] tableauChaine     = {"chaine1", "chaine2", "chaine3"};
int [] tableau              = new int[]{11,13,17,19,23,29};
tableauEntier               = new int[]{11,13,17,19,23,29};
```

Les tableaux - 3

- En java, nous pouvons aussi créer des tableaux prêts à recevoir des données.
  - Pour cela, on doit allouer de l'espace pour le tableau dans la mémoire.

Déclaration et allocation d'éspace

```
<type> [] <nom> = new <type>[taille du tableau];
```

- **Exemples :**

```
int [] ti = new int [10];  // ti est un tableau de 10 cases
char [] tc = new char [15];  // tc est un tableau de 15 cases
```

- La taille d'un tableau en java est fixe et ne change pas.
- La taille d'un tableau est déterminée par l'atribut ***length***.
- Le premier élément d'un tableau possède l'indice 0.

Les tableaux - 4

- Pour lire ou écrire les valeurs d'un tableau, il faut ajouter l'indice entre crochets ([et]) à la suite du nom du tableau :
  - Pour cela, on doit allouer de l'espace pour le tableau dans la mémoire.

Lecture

```
<nom>[indice de l'élément];
```

Ecriture

```
<nom>[indice de l'élément] = Nouvelle valeur;
```

- **Exemples :**

```
int [] monTableau = {2,3,5,7,11,23,17};
int nb;

monTableau[5] = 13;  // ->  2 3 5 7 11 13 17
nb = monTableau[4];  // 11
nb = monTableau.length; // nb -> 7
```

Les tableaux - 5

- L'attribut **length** d'un tableau donne sa longueur (le nombre d'éléments). Donc pour un tableau nommé **monTableau** l'indice du dernier élément est ***monTableau.length-1***.

- Ceci est particulièrement utile lorsque nous voulons parcourir les éléments d'un tableau.

  - **Exemple :**

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

Les tableaux - 6

- **Java 5** fournit un moyen plus court de parcourir un tableau.
  - L'exemple suivant réalise le traitement sur ***monTableau*** :

- **Exemple :**

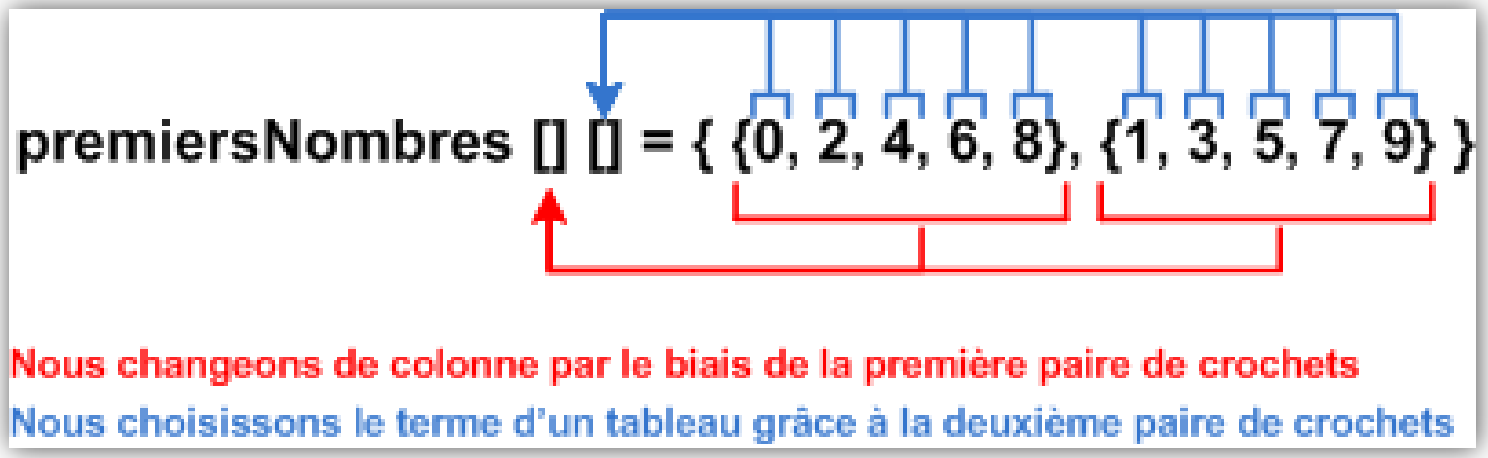
```
for (int element : monTableau){
        System.out.println(element);
   // traitement
}
```

- La variable ***element*** contient une copie de ***monTableau[i]***. Avec des tableaux contenant des **variables primitives**, toute modification de ***element*** n'aura aucun effet sur le contenu du tableau.

Les tableaux multidimensionnels - 1

- Un tableau multidimensionnel n'est rien d'autre qu'un tableau contenant au minimum deux tableaux.
  - Java permet la création et la manipulation de tableaux de plusieurs dimensions.
  - Il faut juste mettre autant de [] que le nombre de dimensions.

Exemple



Les tableaux multidimensionnels - 2

Exemple

```
int[][][] troisD = { { {1, 2, 3}, { 4, 5, 6}, { 7, 8, 9} },
                     { {10, 11, 12}, {13, 14, 15}, {16, 17, 18} },
                     { {19, 20, 21}, {22, 23, 24}, {25, 26, 27} } };
```

Exercice

Donnez un exemple de déclaration et d'initialisation d'un tableau à 5 dimensions (2*2*2*2*2). Le contenu du tableau sera une série de valeurs commençant par 1.

Les tableaux multidimensionnels - 3

- Exemple, pour allouer une matrice de 5 lignes de 6 colonnes :

```
int[][] matrice=new int[5][];
for (int i=0 ; i<matrice.length; i++) {
    matrice[i]=new int[6];
}
```

- Java permet de résumer l'opération précédente en :

```
int[][] matrice=new int[5][6];
```

Les tableaux multidimensionnels - 4

- La première version montre qu'il est possible de créer un tableau de tableaux n'ayant pas forcément tous la même dimension.

- On peut également remplir le tableau à la déclaration et laisser le compilateur déterminer les dimensions des tableaux, en imbriquant les accolades :

```
int[][] matrice =
    {
        { 0, 1, 4, 3 } , // tableau [0] de int
        { 5, 7, 9, 11, 13, 15, 17 } // tableau [1] de int
    };
```

Les tableaux multidimensionnels - 5

- Pour déterminer la longueur des tableaux, on utilise également l'attribut length :

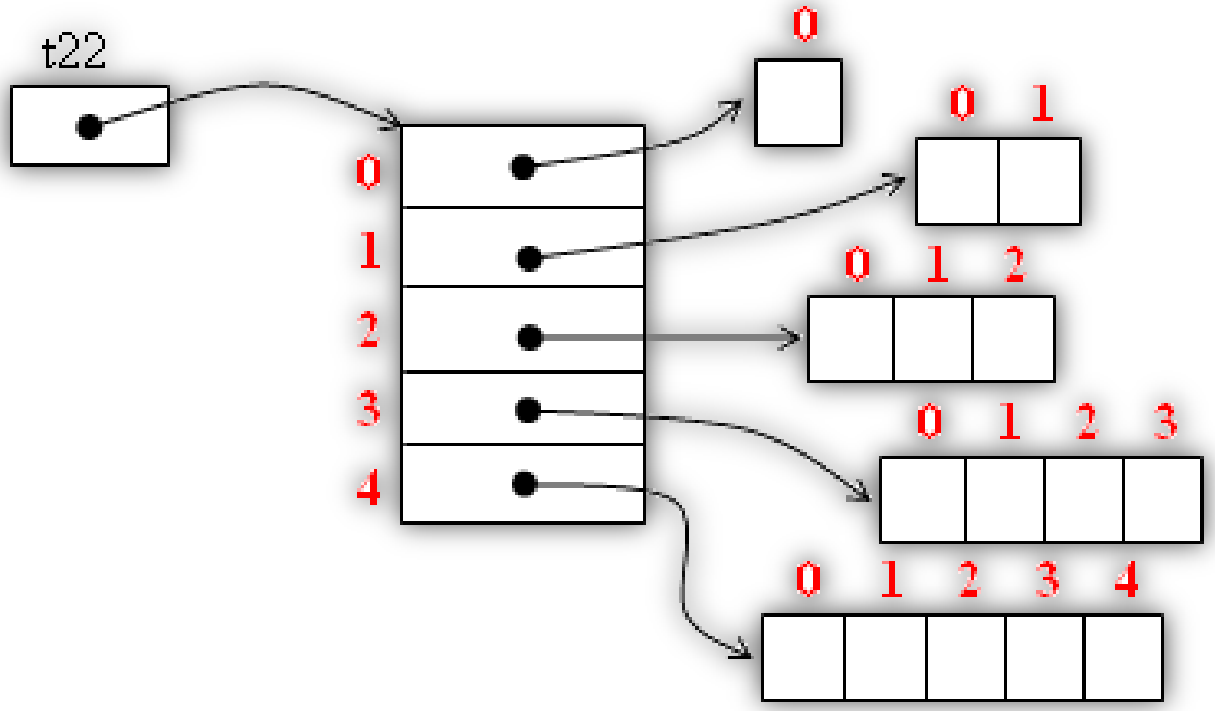
```
matrice.length      //  2
matrice[0].length   //  4
matrice[1].length   //  7
```

- De la même manière que précédement, on peut facilement parcourir tous

```
for(int i=0; i<matrice.length; i++) {
        for(int j=0; j<matrice[i].length; j++) {
                //Action sur matrice[i][j]
        }
}
```

Les tableaux multidimensionnels - 6

- **Exercice :**
- Ecrivez le code java permettant d'avoir un tableau à double dimensions ayant la structure suivante :



Argument de la ligne de commande - 1

- Dans l'en-tête d'une méthode main apparait ce qui s'appelle un paramètre qui est ci-dessous nommé args :

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

  - On peut nommer ce paramètre selon son propre choix.
  - L'en-tête pourrait aussi être :

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

  - Cet argument est de type tableau de String ou encore tableau de chaînes de caractères.
- Grâce à ce paramètre, on peut "passer des arguments à la méthode main".

Ligne de commande

```
> java NomDeLaClasse arg0 arg1 … argN
```

Argument de la ligne de commande - 2

Code source

```
public static void main (String [] args) {
        System.out.println (args[0]);
        System.out.println (args[1]);
        …
        System.out.println (args[args.length-1]);

        //ou

        for (String arg : args) {
                System.out.println (arg);
        }
}
```

UNIVERSITÉ PARIS 8
VINCENNES-SAINT-DENIS

Java

つづく

to be continued ...