

CHAPITRE 2 : LES ELEMENTS DU LANGUAGE JAVA

Dr. A. Nejeoui



INTRODUCTION

Objectifs :

- Comprendre la structure d'une application Java
- Manipulation des variables de types primitifs
- Comprendre le Dépassement
- Manipulation des variable de types references
- Les opérateurs (arithmétiques, logiques,)
- Les Expressions et les Instructions

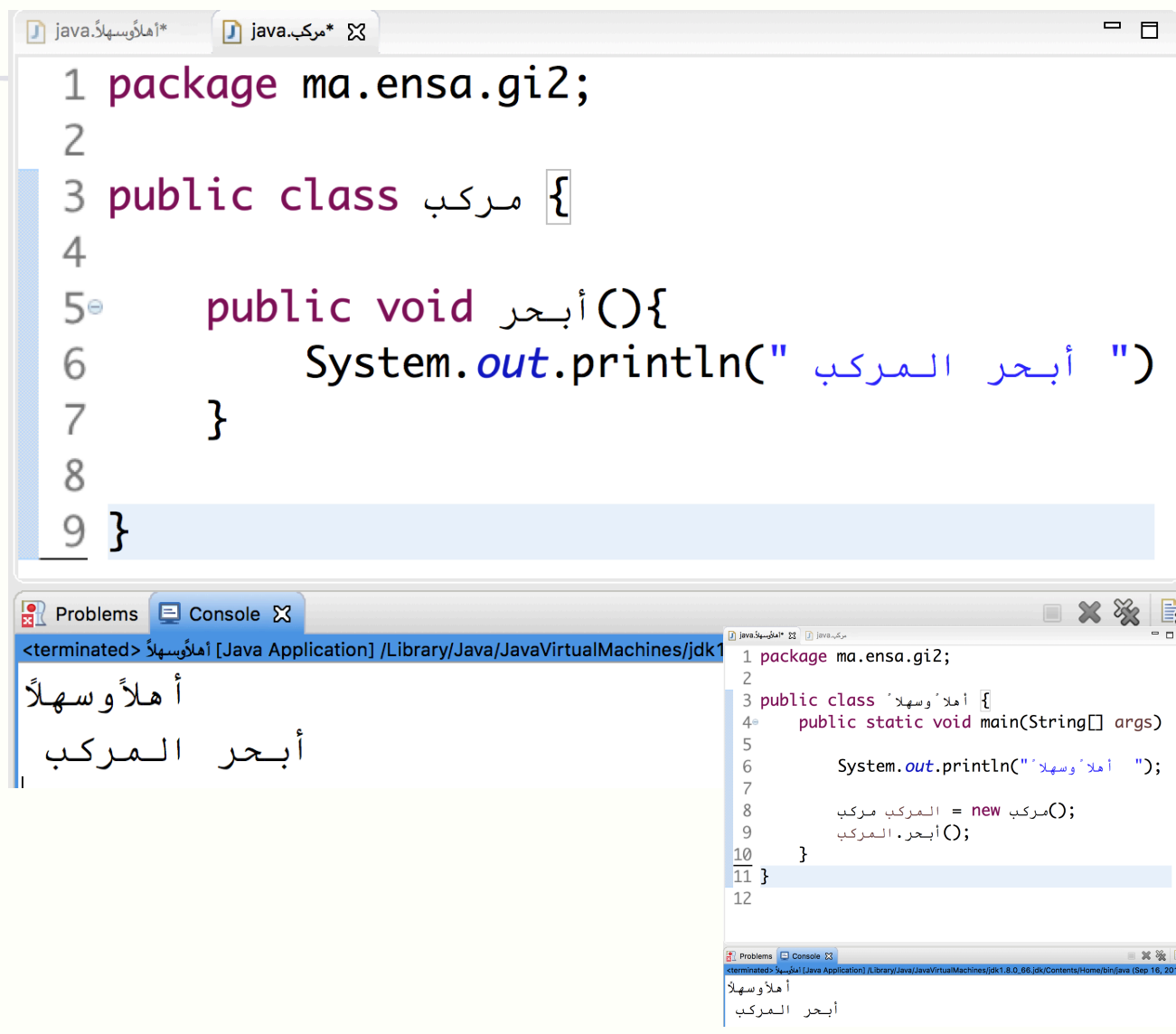


INTRODUCTION

- Instructions de contrôle
 - if - else
 - L'instruction switch
 - Boucle for
 - Boucle while
 - Boucle do-while
 - break et continue
 - try-catch-finally
 - return



STRUCTURE D'UNE APPLICATION JAVA



The screenshot shows an IDE with two windows. The top window, titled 'java.أهلا وسهلا*', contains the following Java code:

```
1 package ma.ensa.gi2;
2
3 public class مركب {
4
5     public void أبحر() {
6         System.out.println(" أبحر المركب ")
7     }
8
9 }
```

The bottom window, titled 'Problems Console', shows the output of the program:

```
<terminated> أهلا وسهلا [Java Application] /Library/Java/JavaVirtualMachines/jdk1
أبحر المركب
```

Below the console, another code snippet is visible, showing a `main` method that creates an instance of the `مركب` class and calls its `أبحر` method:

```
1 package ma.ensa.gi2;
2
3 public class أهلا وسهلا {
4     public static void main(String[] args)
5     {
6         System.out.println(" أهلا وسهلا ");
7
8         مركب المركب = new مركب();
9         المركب.أبحر();
10    }
11 }
12
```

UNE APPLICATION JAVA EST UN ENSEMBLE DE CLASSES ET INTERFACES DONT UNE AU MOINS CONTIENT LA MÉTHODE MAIN. LES CLASSES SONT COMPOSÉS DE MÉTHODES ET ATTRIBUTS. UNE MÉTHODE EST UN ENSEMBLE D'INSTRUCTIONS. IL Y A 3 TYPES D'INSTRUCTIONS, INSTRUCTIONS :

- DE DECLARATION
- D'EXPRESSION
- DE CONTRÔLE DE FLOT D'EXÉCUTION



Package

Hierarchy

APPLICATION

src

(default package)

VolcanoApplication.java

VolcanoApplication

main(String[])

VolcanoRobot.java

VolcanoRobot

speed

status

temperature

checkTemperatur

showAttributes()

JRE System Library [jdk]

JavaProgramming

myproj

VolcanoRobot.java

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

lass VolcanoRobot {

String status;

int speed;

float temperature;

void checkTemperature() {

if (temperature > 660) {

status = "returning home";

speed = 5;

}

}

void showAttributes() {

System.out.println("Status: " + status);

System.out.println("Speed: " + speed);

System.out.println("Temperature: " + temperature)

}

VolcanoApplication.java

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

class VolcanoApplication {

public static void main(String[] arguments) {

VolcanoRobot dante = new VolcanoRobot();

dante.status = "exploring";

dante.speed = 2;

dante.temperature = 510;

dante.showAttributes();

System.out.println("Increasing speed to 3.");

dante.speed = 3;

dante.showAttributes();

System.out.println("Changing temperature to 670.

dante.temperature = 670;

dante.showAttributes();

System.out.println("Checking the temperature.");

dante.checkTemperature();

dante.showAttributes();

}

Console

Problems

Javadoc

Declaration

<terminated> VolcanoApplication [Java Application] C:\Sun\SDK\jdk\bin\javaw.exe

Status: exploring

Speed: 2

Temperature: 510.0

Increasing speed to 3.

Status: exploring

Speed: 3

Temperature: 510.0

Changing temperature to 670.

Status: exploring

Speed: 3

Temperature: 670.0

Checking the temperature.

Status: returning home

Speed: 5

Temperature: 670.0

Ensemble d'instructions

Une Méthode

Méthode main

Instanciation d'un objet de type VolcanoRobot

POO JAVA: GIZ

LES DECLARATIONS



LES VARIABLES DE TYPES PRIMITIFS

Type	Taille	Min	M
byte / Byte	8	-128	127
short / Short	16	-32768	32767
char / Character	16	U+0000	U+FFFF
int / Integer	32	-2147483648	2147483647
long / Long	64	-9223372036854775808	9223372036854775807
float / Float	32	1.4E-45	3.4028235E+38
double / Double	64	4.9E-324	1.7976931348623157E308
boolean / Boolean	-	-	-

UNE VARIABLE EST UNE ZONE MEMOIRE IDENTIFIÉE PAR UN IDENTIFICATEUR.

CHAQUE VARIABLE POSSÈDE UN NOM ET UN TYPE ET ELLE EST VISIBLE DANS SA PORTÉE.

EN JAVA ON DISTINGUE ENTRE :

- LES VARIABLES DE TYPE PRIMITIF
- LES VARIABLES DE TYPE REFERENCE



OVERFLOW / UNDERFLOW

```
java.أهلا وسهلا. java.مركب. Test.java Other.java OverflowAndUnderflow.java
3 public class OverflowAndUnderflow {
4     public static void main(String[] args) {
5         System.out.println(Double.MAX_VALUE + " " + Double.MIN_VALUE);
6         System.out.println(Float.MAX_VALUE + " " + Float.MIN_VALUE);
7         int n = 2000000000;
8         System.out.println(n * n);
9         System.out.println(Integer.MIN_VALUE - 1 == Integer.MAX_VALUE);
10        double d = 9.9768945E303;
11        double d2 = 9.9768945E303;
12        for (int i = 0; i < 4; i++)
13            System.out.println((d *= 1.87E1) + " " + (d2 /= 23.87E200));
14    }
15 }
```

Console

<terminated> OverflowAndUnderflow [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_66.jdk/Contents/Home/bin/java (Sep 18, 2017, 11:52:55 AM)

```
1.7976931348623157E308 4.9E-324
3.4028235E38 1.4E-45
-1651507200
true
1.8656792715E305 4.1796793045664015E102
3.4888202377049997E306 1.7510177229017183E-99
6.524093844508349E307 7.3356419057466204E-301
Infinity 0.0
```



OVERFLOW / UNDERFLOW

Un entier int est stocké sur 32-bits, le résultat $n*n$ est 4,000,000,000,000,000,000 dont la représentation binaire est sur 64-bits :

-- Les bits du poids le plus fort -- 00110111 10000010 11011010 11001110 -- Les bits du poids le plus faible-
10011101 10010000 00000000 00000000

32-bit n'est pas suffisant pour stocker le nombre, les 4 octets du poids le plus fort sont tronqués, le nombre est donc remplacé par les 4 octets du poids le plus faible : 10011101 10010000 00000000 00000000 qui représente le nombre décimal 1651507200 et comme le bit du poids le plus fort est égal à 1, le nombre est négatif donc on aura -1651507200

Un double est stocké sur 64-bits La valeur Max = Double.MAX_VALUE = 1.7976931348623157E308

La valeur Min = Double.MIN_VALUE = 1.7976931348623157E308

Un float est stocké sur 32-bits La valeur Max = Float.MAX_VALUE = 3.4028235E38

La valeur Min = Float.MIN_VALUE = 1.4E-45

Le résultat d'un Overflow pour les types double et float donne INFINITY

Le résultat d'un Underflow pour les types double et float donne 0.0



NOM D'UNE VARIABLE

Un identificateur :

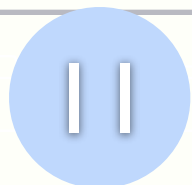
- Doit commencer par une Lettre Java (`Character.isJavaIdentifierStart(int)` doit retourner true)
- Doit contenir seulement des caractères de type Lettre Java ou chiffres Java (`Character.isJavaIdentifierPart(int)` doit retourner true)
- Doit être différent des **mots réservés du langage Java**
- Doit être différent des littérales : **true**, **false** et **null**.
- Peut avoir une taille quelconque : mais de préférence il faut choisir des noms de variables significatifs et courts.

Par convention : le nom d'une variables commence par une lettre minuscule.



50 MOTS-RÉSERVÉS JAVA

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



VARIABLES DE TYPE REFERENCE

Toute variable de type différent des 8 types primitifs est dite de type référence. Exemple :

```
String nom="Hicham";
```

```
Date date;
```

```
Gadget helicopter = new Helicopter();
```

l'API Java 8 contient plus de 4000 classes.

On peut initialiser les variables de type reference par affectation d'une reference d'un type compatible ou via l'appel d'un constructeur d'un type compatible.



INITIALISER UNE VARIABLE

une variable de type primitif peut être initialisée par une littérale d'un type compatible.

- `byte index=109;`
- `short numero=34567;`
- `float x = 35.09f;`
- `boolean flag=true;`
- `int x = 35;`
- `double d=3.56;`
- `long mils=1234567890L;`
- `char جيم = '\u062C';`
- `char tradeM = '\u2122';`



PORTÉE D'UNE VARIABLE

```
1 package ma.ensa.g12;  
2  
3 import java.text.ParseException;  
4 import java.text.SimpleDateFormat;  
5 import java.util.Date;  
6  
7 public class Portee {  
8     private int membre1 = 15;  
9     private Date membre2;  
10  
11     public void uneMethode(String parametre1, boolean parametre2){  
12         double local = 2.5;  
13         if (membre1 > local && parametre2)  
14             try {  
15                 membre2 = new SimpleDateFormat("dd-MM-yyy").parse(parametre1);  
16             } catch (ParseException e) {  
17                 e.printStackTrace();  
18             }  
19     }  
20 }
```

Definition :

La portée d'une variable c'est la section du code source dans laquelle la variable est visible.

Java définit quatre niveaux de portée :

- Les variables membre
- Les paramètres d'une méthode
- Les variables locales
- Les paramètres de gestion des exceptions : accessibles dans le block catch



LA CLASSE STRING

- La classe String est une classe immuable qui représente des chaînes de caractères immuables.
 - length()
 - charAt(int index)
 - split(String sep)
 - intern()
 - substring(int beginIndex, int endIndex)
 - match(String regex)



OPÉRATEURS

Il y'a plusieurs type d'opérateur en Java :

- Opérateurs arithmétiques
- Opérateurs conditionnels et relationnels
- Opérateur logiques
- Opérateurs sur les bits.
- Opérateurs d'affectation



OPÉRATEURS ARITHMÉTIQUES

Opérateur	Utilisation	Description
+	$op1 + op2$	Ajoute de op1 à op2
-	$op1 - op2$	Soustrait op2 de op1
*	$op1 * op2$	Multiplie op1 par op2
/	$op1 / op2$	divise op1 par op2
%	$op1 \% op2$	Calcul le reste de la division de op1 par op2



OPÉRATEURS UNAIRES

Opérateur	Utilisation	Description
+	+op	Change le type de op à int si c'était un byte, short, ou char
-	-op	Négation arithmétique de op

Opérateur	Utilisation	Description
++	op++	Incrément op par 1; il est évalué par la valeur de op avant incrémentation .
++	++op	Incrément op par 1; il est évalué par la valeur de op après incrémentation .
--	op--	Décrément op par 1; il est évalué par la valeur de op avant décrémentation .
--	--op	décrément op par 1; il est évalué par la valeur de op après décrémentation .



OPÉRATEURS RELATIONNELS

Opérateur	Utilisation	Retourne true si
>	op1 > op2	op1 est plus grand que op2
>=	op1 >= op2	op1 est plus grand ou égal à op2
<	op1 < op2	op1 est inférieur à op2
<=	op1 <= op2	op1 est inférieur ou égal à op2
==	op1 == op2	op1 et op2 sont égaux
!=	op1 != op2	op1 et op2 sont différents

Un opérateur relationnel compare deux opérandes et détermine la relation entre eux



OPÉRATEURS CONDITIONNELS

Opérateur	Utilisation	Retourne true si
&&	op1 & & op2	op1 et op2 sont vrais tous les deux, l'évaluation de op2 est conditionnelle
	op1 op2	L'un des opérandes op1 ou op2 est vrai, l'évaluation de op2 est conditionnelle
!	! op	op est faux
&	op1 & op2	op1 et op2 sont vrais tous les deux, l'évaluation de op2 et op1 est nécessaire
	op1 op2	L'un des opérandes op1 ou op2 est vrai, l'évaluation de op2 et op1 est nécessaire
^	op1 ^ op2	Si op1 et op2 sont différents



OPÉRATEURS D'AFFECTATION

Opérateur	Utilisation	Equivalence
<code>+=</code>	<code>op1 += op2</code>	<code>op1 = op1 + op2</code>
<code>-=</code>	<code>op1 -= op2</code>	<code>op1 = op1 - op2</code>
<code>*=</code>	<code>op1 *= op2</code>	<code>op1 = op1 * op2</code>
<code>/=</code>	<code>op1 /= op2</code>	<code>op1 = op1 / op2</code>
<code>%=</code>	<code>op1 %= op2</code>	<code>op1 = op1 % op2</code>
<code>&=</code>	<code>op1 &= op2</code>	<code>op1 = op1 & op2</code>
<code> =</code>	<code>op1 = op2</code>	<code>op1 = op1 op2</code>
<code>^=</code>	<code>op1 ^= op2</code>	<code>op1 = op1 ^ op2</code>



LES OPÉRATEURS SUR LES BITS

Opérateur	Utilisation	Description
&	op1 & op2	après promotion numérique binaire & est appliqué sur chaque bit
	op1 op2	après promotion numérique binaire est appliqué sur chaque bit
^	op1 ^ op2	après promotion numérique binaire ^ est appliqué sur chaque bit
~	~op	~op égale (-op)-1
<<	Décalage à gauche	7 << 2 donne 28
>>	Décalage à droite signée	7 >> 2 donne 1
>>>	n >>> s Décalage à droite non signée	si n est un entier négatif ($n \gg s$) + (2 << 31-s) si n est un long négatif ($n \gg s$) + (2L << 63-s) -7 >>> 1 donne 2147483644 -7L >>> 1 donne 9223372036854775804
>>=	op1 >>= op2	op1 = op1 >> op2
<<=	op1 <<= op2	op1 = op1 << op2
>>>=	op >>>= op2	op = op >>> op2

Si l'un des opérandes n'est pas de type primitif intégral (byte, short, char, int, long) l'expression génère une erreur de compilation.

Dans le cas des opérateurs (&, |, ^) la promotion numérique binaire est appliquée sur les deux opérandes.

La promotion numérique binaire n'est pas appliquée sur les deux opérandes dans le cas des opérateurs de décalage de bits (>>, <<, >>>), par contre la promotion numérique binaire est appliquée sur chaque opérande séparément avant d'appliquer les opérateurs de décalage des bits.



AUTRES OPÉRATEURS

Opérateur	Description
?:	Abréviation de l'instruction if-else
[]	Déclaration des tableaux (arrays), création, accès aux éléments.
.	Pour former des nom composés
(<i>params</i>)	Détermine une liste des paramètres séparés par des virgules.
(<i>type</i>)	Converti une valeur au type spécifié
new	Pour créer un nouveau objet ou nouveau tableau
instanceof	Détermine si son premier opérande est une instance de son second opérande.



COMPLÉMENT À DEUX

- Java utilise la représentation complément à deux pour représenter les nombres entiers.
- un nombre positifs est représenté par ça représentation binaire ordinaire.
- un nombre négatif est obtenu alors par inversion des bits du nombre positif puis ajouter 1 et ignorer le dépassement.



EXPRESSIONS, INSTRUCTION ET BLOCS



EXPRESSIONS

- Les expressions sont utilisées pour calculer et assigner les valeurs aux variables, et pour aider à contrôler le flot d'exécution du programme.
- Le rôle d'une expression est double: faire les calculs indiqués par les éléments de l'expression et retourner le résultat du calcul.

Définition

Une expression est une série d'opérandes, opérateurs et appels de méthodes qui est évaluée à une seule valeur.



EXPRESSIONS

Quelques exemples d'expressions

- `a=b+c`
- `System.out.println("Bonjour")`
- `Character.isJavaIdentifierPart(ch)`
- `a++`
- `ch = str.charAt(0)`



EVALUATION DES EXPRESSIONS

l'opérande à gauche est évalué en premier :

- `int a=5;`
- `a= (a=10)+a;`
- `System.out.println(a)`
- `a+=(a=10);`



`System.out.println(a)`

POO JAVA: GI2

VALEUR D'UNE EXPRESSIONS

Expression	Action	Valeure Retournée
ch = 'a'	affecte le caractere 'a' à la variable ch	la valeur de ch après affectation de ('a')
"La valeur MAX d'un type short est " + Short.MAX_VALUE	Concatène la chaîne de caractères "La valeur MAX d'un type short est " et la valeur de Short.MAX_VALUE après conversion au type string	La chaîne de caractères : La valeur MAX d'un type short est 32767
Character.isJavaIdentifierPart(ch)	Appelle la méthode .isJavaIdentifierPart	La valeur retournée par la méthode : true

Chaque expression réalisé une operation et retourne une valeur.



LES INSTRUCTIONS

Les instructions sont équivalentes aux phrases dans les langages naturelles.

Une instruction forme une unité complète d'exécution. Il y'a trois types d'instructions:

- Les instructions d'expressions
- Les instructions de déclarations
- Les instructions de contrôle de flux



INSTRUCTIONS D'EXPRESSIONS

Toute expression terminée par un point virgule est une instruction d'expression.

Exemples :

- `aValue = 8933.234; // instruction d'affectation`
- `aValue++; // instruction d'incrémentation`
- `System.out.println(aValue); //appel de méthode`
- `Integer integerObject = new Integer(4); // instruction d'instantiation`



INSTRUCTIONS DE DECLARATION

Une instruction de declaration permet de declarer une variable de type primitif ou de type reference.

Exemples :

- `int somme;`
- `char uneLettre;`
- `Voiture voiture;`
- `Object obj;`



INSTRUCTIONS DE CONTRÔLE DE FLOT D'EXÉCUTION

- if
- if-else
- switch
- boucles : for, while et do-while
- break
- continue
- try-catch-finally
- return



IF - ELSE

La syntaxe générale :

```
if (expression booléenne) {  
    bloc1 }  
else {  
    bloc2  
}
```

La condition doit être évaluable en true ou false et elle est **obligatoirement** entourée de **parenthèses**.

La partie commençant par **else** est facultative.

Les points-virgules sont obligatoires après chaque instruction et interdits après }.

Si un bloc ne comporte qu'une seule instruction, les accolades qui l'entourent peuvent être omises.



SINON PENDANT 'DANGLING ELSE'

```
1 package ma.ensa.gi2;  
2  
3 public class Dangling {  
4  
5     public static void main(String[] args) {  
6         int a=5;  
7         if(a< -10) System.out.println("a < -10");  
8         if(a>5) System.out.println("a>5");  
9         else System.out.println(" ? ");  
10    }  
11 }
```



L'OPÉRATEUR TERNAIRE

`() ? ... : ...`

- `int a=5, b=20, max;`
- `max = (a>b)?a:b ;`
- `System.out.println(max);`



SWITCH

La syntaxe générale :

```
switch (expression entière ou caractère ou chaîne de caractère ou  
    Enumeration) {  
case i:  
case j:  
[bloc d'instructions]  
break;  
case k :  
.....  
default:  
}
```

Rq:

Le type de la variable d'une instruction case doit être char, byte, short, int, String ou Enum.



SWITCH

```
1 package ma.ensa.gi2;
2 public class Switch {
3     public static void main(String[] args) {
4         myEnum l=myEnum.V1;
5         switch(l){
6             case V1: System.out.println("V1");break;
7             case V2: System.out.println("V2");break;
8             case V3: System.out.println("V3");break;
9             default :System.out.println("Default");
10        } } }
```

myEnum.java

```
1 package ma.ensa.gi2;
2 public enum myEnum{
3     V1, V2, V3
4 }
```



SWITCH

```
1 import java.util.Scanner;
2
3 class Switch {
4     public static void main(String[] arguments) {
5         Scanner scanf = new Scanner(System.in);
6         int mois, nbJours, annee;
7         System.out
8             .println("saisir le mois sous form d'un entier entre 1 et 12");
9         mois = scanf.nextInt();
10        System.out.println("saisir l'année ");
11        annee = scanf.nextInt();
12        switch (mois) {
13            case 1: case 3: case 5: case 7: case 8: case 10: case 12: nbJours = 31; break
14            case 4:
15            case 6:
16            case 9:
17            case 11:
18                nbJours = 30; break;
19            case 2:
20                if ((annee % 4 == 0) && !(annee % 100 == 0) || (annee % 400 == 0))
21                    nbJours = 29;
22                else
23                    nbJours = 28; break;
24            default: nbJours = 0;
25        }
26        System.out.println("le nombre de jours du mois " + mois + " est : "
27            + nbJours + " jours");
28    }
}
```



BOUCLE FOR

```
1 package ma.ensa.gi2;  
2 public class For {  
3     public static void main(String[] args) {  
4         int somme = 0, max = 2;  
5         int k = -5;  
6         for (int i = 0, j = 100; i <= max && j > 0; i++, j--, k += 15)  
7             { somme += i * j - k;  
8             System.out.println("i=" + i + " j=" + j + " k=" + k +  
9                 " somme = " + somme);  
10        }  
11    }  
12 }
```

Markers Properties Servers Data Source Explorer Snippets Console Progress Debug History

<terminated> For [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_66.jdk/Contents/Home/bin/java (Sep 21, 2017, 12:39:36 PM)

i=0	j=100	k=-5	somme = 5
i=1	j=99	k=10	somme = 94
i=2	j=98	k=25	somme = 265

Syntaxe Générale :

```
for ( liste expressions1; Expression  
      logique ; liste expressions3){  
    bloc  
}
```

C'est la boucle contrôlée, utilisée
pour répéter N fois un même
bloc d'instructions

liste expressions1 précise la valeur
initiale des variables de
contrôle (ou compteurs)

Expression logique, la condition à
satisfaire pour rester dans la
boucle

liste expressions3, une liste
d'actions à réaliser à la fin
de chaque boucle (en
général, l'actualisation des
compteurs).



BOUCLE WHILE

```
int max = 100, i = 0, somme = 0;
while (i <= max) {
    somme += i;    // somme = somme + i
    i++;
}
```

Attention de ne pas mettre de point virgule après la condition.

C'est une erreur non détectable par le compilateur mais affecte la sémantique du programme.

L'itération while

```
while (expression) {
    bloc
}
```

Elle permet de répéter un bloc d'instructions TANT QUE la condition est vraie.

La sortie de boucle est effectuée aussitôt que la condition est évaluée fausse.

Le test de cette condition est vérifié au début de chaque boucle, si bien que le bloc peut ne pas être exécuté.



BOUCLE DO-WHILE

```
int i = 100, j = 0, somme = 0 ;
```

```
do{
```

```
    somme += j;
```

```
    j++;}
```

```
while (j <= i); //Si oublié génère une erreur de  
compilation
```

A la sortie de la boucle, la variable somme contient la somme des 100 premiers entiers.

Cette structure est très proche de la structure while. Sa syntaxe est :

```
do{
```

```
    instructions;}
```

```
    while (condition);
```

Dans cette boucle **faire_tant_que**, la condition est évaluée après l'exécution du corps de la boucle. Elle est au minimum exécutée une fois même si la condition à tester est fausse au départ



INSTRUCTIONS DE RUPTURE DE BOUCLE

Pas de goto; en Java .. mais il existe des moyens pour sortir ou effectuer des "sauts" en rompant le cours normal d'une itération. Avec les instructions

break [label] : on quitte définitivement le bloc courant (et on passe à la suite).

continue [label] : on saute les instructions du bloc situés à la suite (et on passe à l'itération suivante).

Si on indique break [label], ou continue [label], où label est une étiquette qui marque un endroit dans le code, le saut s'effectue relativement au bloc marqué par l'étiquette.



BREAK

```
1 import java.util.Scanner;
2 class BreakDemo {
3     public static void main(String[] args) {
4         Scanner scanf = new Scanner(System.in);
5         int produitNuméro;
6         double amount, total;
7         System.out.println("Vous pouvez acheter jusqu'à 10 produits, mais");
8         System.out.println("la prix total ne doit pas excéder $100.");
9         total = 0;
10        for (produitNuméro = 1; produitNuméro <= 10; produitNuméro++) {
11            System.out.print("Entrer le prix du produit #" + produitNuméro
12                + ": $");
13            amount = scanf.nextDouble();
14            total = total + amount;
15            if (total >= 100) {
16                System.out.println("Vous avez dépensé tout votre argent.");
17                break;
18            }
19            System.out.println("Le prix total est $" + total);
20            System.out.println("Vous pouvez acheter jusqu'à "
21                + (10 - produitNuméro) + " produits.");
22        }
23        System.out.println("Vous avez dépensé $" + total);
24    }
25 }
```

- L'instruction break a deux formes: **avec un label** ou **sans label**, on a déjà vu la forme sans label avec switch. Dans ce cas break termine l'instruction switch et transfère le flux de contrôle immédiatement à l'instruction qui suit switch
- Vous pouvez aussi utiliser l'instruction break pour terminer une boucle for, while ou do-while. Le programme suivant BreakDemo contient une boucle for qui permet de calculer la somme des produits achetés .



BREAK AVEC ÉTIQUETTE

```
1 package ma.ensa.gi2;
2 public class BreakAvecLibelle {
3     public static void main(String[] args) {
4         bcl1:
5         {   for (int i = 0; i < 5; i++) {
6             for (int j = 0; j < 5; j++) {
7                 if (j == 2)
8                     break;
9                 if (i == 2)
10                    break bcl1;
11                System.out.println("i=" + i + "    j=" + j);
12            }
13        }
14        System.out.println("Fin");
15    }
16 }
17 }
```



CONTINUE

```
2 public class Continue {
3     public static void main(String[] args) {
4         bcl1:    for (int i=0; i<5; i++){
5                 for (int j=0; j<5; j++){
6                     if (j==2) continue;
7                     if (i==2) continue bcl1;
8                 System.out.println("i="+i+"    j="+j);
9                 }
10            }
11        System.out.println("Fin");
12    }
13 }
```

Cette instruction modifie aussi le déroulement normal d'une boucle. Elle permet de sauter les instructions qui suivent continue et de redémarrer le flot d'instructions au niveau de l'évaluation de la condition de la boucle.

Contrairement à l'instruction break qui fait **quitter** la boucle, on saute les instructions qui suivent continue puis on **continue** l'exécution de l'itération. L'instruction continue peut aussi se combiner avec une étiquette.



TRY-CATCH-FINALLY

```
1 package ma.ensa.gi2;
2 import java.text.*;
3 import java.util.Date;
4 public class TryCatchFinally {
5     public static void main(String[] args) {
6         Date date=null;
7         try{
8             date=new SimpleDateFormat(args[0]).
9                 parse(args[1]);
10        }catch (ParseException e) {
11            e.printStackTrace();
12        }
13        finally {
14            date=new Date();
15        }
16    }
17 }
```



RETURN

Cette instruction termine **immédiatement** l'exécution des méthodes. On peut donc considérer que c'est aussi une instruction de contrôle.

Le flux de contrôle est transféré à l'instruction qui suit immédiatement l'appel de la méthode. L'instruction return a deux formes : une forme qui retourne une valeur et une autre qui ne retourne aucune valeur. Pour retourner une valeur vous pouvez tout simplement placer la valeur (ou l'expression qui calcul cette valeur) après le mot clés **return**; le type de données de la valeur retournée doit correspondre au type de la méthode lors de la déclaration. Lorsque la méthode est déclarée de type void, utilisez tout simplement **return**;



CONCLUSION

Dans ce chapitre on a présenté les elements de base du langage JAVA :

- Les Variables de types primitifs et références
- Les opérateurs (arithmétiques, logiques, décalage de bits)
- Les expressions
- Les instructions
 - de déclaration
 - d'expression
 - de contrôle de flot d'executions

