
2017-2018
Premiers cours MLL5U20
(extraits de “Partie 1”)

JSE : Java Platform Standard Edition

version actuelle : JSE 8

A compter du 21/9/2017 : JSE 9

JRE : Java Runtime Environment

machine virtuelle + bibliothèques de base

JDK : Java Development Kit

JRE + compilateur

Premières applications autonomes

```
package up5.mi.pary.jt.hello;  
// un premier programme  
/* la version JAVA du classique  
   Hello World  
*/
```

// Ceci est un commentaire finissant en fin de ligne

/* ceci est un commentaires pouvant encadrer
un nombre quelconques de caractères
sur un nombre quelconque de lignes */

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

Hello World !

noms de paquets

**Les paquets dont le nom commence par
"java."
sont réservés à Oracle**

**Le concepteur de ce cours
utilise des paquets
dont le nom
commence par "up5.mi.pary."**

**Le nom de paquetage doit être choisi de telle manière
qu'il identifie sans ambiguïté
la personne ou la société.**

Quelques paquetages du Java Development Kit (JDK)

Le **J**ava **D**evelopment **K**it (**JDK**) désigne un ensemble de bibliothèques logicielles de base du langage de programmation Java, ainsi que les outils avec lesquels le code Java peut être compilé....

Wikipedia

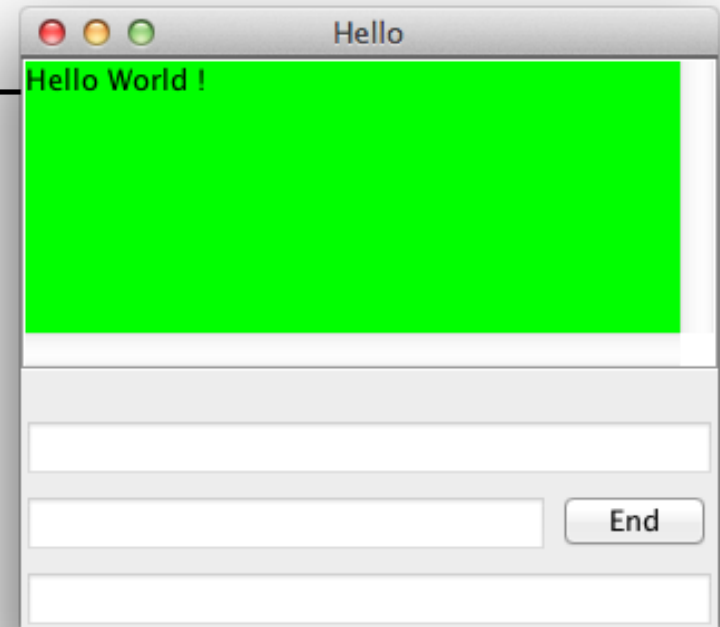
Nom du paquetage	rôle des classes du paquetage
java.lang	les classes de base
java.io	les entrées sorties
java.util	les utilitaires
java.net	communication réseau
javafx.application	application javafx
javafx.event	les événements

Hello World avec la classe Terminal

```
package up5.mi.pary.jt.term;

import up5.mi.pary.term.Terminal;

public class HelloWorld {
    public static void main(String [] tArg) {
        Terminal term = new Terminal("Hello",300,200);
        term.println("Hello World !");
        term.end( );
    }
}
```



`new Terminal("Hello",400,400)` permet de créer un terminal de taille 400x400

La fonction **println** de Terminal s'utilise comme `System.out.println`

La fonction **end()** permet de signaler visuellement la fin de l'exécution du programme.

exemple : calcul du carré de la somme de deux nombres

Le programme demande un entier **1**
puis un autre entier **2** puis affiche le
carré de la somme **3**

calcul du carré de la somme

donner un entier:

1

donner un entier:

5

OK

calcul du carré de la somme

donner un entier: 5
donner un autre entier:

2

donner un autre entier:

7

OK

calcul du carré de la somme

donner un entier: 5
donner un autre entier: 7
le carré de la somme de 5 et de 7 vaut 144

3

End

calcul du carré de la somme de deux nombres

```
package up5.mi.pary.jt.term;
import up5.mi.pary.term.Terminal;

public class TestTerminal {

    public static void main(String [] args) {

        Terminal term = new Terminal("calcul du carré de la somme",300,300);

        int a = term.readInt("donner un entier:");

        int b= term.readInt("donner un autre entier:");

        term.println("le carré de la somme de "+a+  

                           " et de "+b+" vaut "+(a+b)*(a+b));
        term.end();
    }
}
```

donner un entier: 5

donner un autre entier: 7

le carré de la somme de 5 et de 7 vaut 144

La documentation en ligne de la classe Terminal

La documentation en ligne de la classe Terminal permet d'avoir connaissance des diverses fonctionnalités offertes afin de pouvoir les utiliser dans les programmes que nous écrivons.

IMPORT

les déclarations **import**
permettent, dans le programme,
d'écrire le nom simple des classe (exemple : Terminal)
au lieu du nom complet (exemple : up5.mi.pary.term.Terminal)

C'est une facilité syntaxique
(il n'y a pas d'inclusion de texte comme avec l'instruction include en C).

Syntaxes :

import *nomCompletDeClasse*; // pour l'import d'une classe

import *nomDePaquetage.**; // pour l'import de toutes les classes d'un paquetage

Remarque :

impossible de dire par exemple :

```
import up5.*.*;
```

Seuls les deux formes mentionnées ci-dessus sont possibles !

IMPORT implicites

Sont importées implicitement :

- toutes les classes du paquetage java.lang
- toutes les classes du paquetage courant

```
package dupond.essai;  
// les deux imports suivants sont implicites  
//et il n'est donc pas utile de les mentionner
```

```
import java.lang.*; // java.lang est un package contenant des  
                    // classes très souvent utilisées (System Math String ...)
```

```
import dupond.essai.*; //les classes du paquetage courant sont aussi importées  
implicitement  
// (le paquetage courant est celui de la classe définie dans le fichier)  
public class ...
```

Classes Java

Les classes Java peuvent être classées en deux catégories :

1. les "vraies" classes au sens de la Programmation Orientée Objet

grâce auxquelles on peut créer des objets et qui définissent des fonctionnalités pour manipuler ces objets

exemple :

la classe Terminal qui permet de créer et d'utiliser des terminaux

la classe String qui permet de créer et d'utiliser des chaînes de caractères

2. les classes (que nous appellerons classes d'utilitaires) dont la raison d'être est de regrouper des fonctions (appelées fonctions statiques)

similaires aux fonctions que l'on rencontre dans les langages non orienté objet et des constantes

exemples :

- la classe Math définit des fonctions comme abs, sqrt, min, max et des constantes comme PI
- la classe HelloWorld avec sa fonction main

La classe Math : un exemple de classe d'utilitaires

```
package java.lang;

/** The class Math contains methods for performing basic numeric operations
such as the elementary exponential, logarithm, square root, and trigonometric
functions.
*/

public final class Math {
// Static fields

/** The double value that is closer than any other to e, the base of the natural
logarithms */
public static final double E = 2.7182818284590452354;

/** Returns the absolute value of an int value.*/
public static int abs(int a){ return (a < 0) ? -a : a;}

/** Returns the arc cosine of an angle, in the range of 0.0 through pi.*/
public static double acos(double a){...}

/** Returns the square root of a double value.*/
public static double sqrt(double a){...}
...}
```

un autre exemple de classe d'utilitaires

```
package up5.mi.pary.jc.statique;
```

```
public class Exemple {
```

```
/**@return le carré de la somme de deux entiers*/
```

```
public static int carreDeLaSomme(int a,int b){
```

```
    int somme=a+b;
```

```
    int carre = somme * somme;
```

```
    return carre;
```

```
}
```

```
/** @return la factorielle d'un entier */
```

```
public static int fact(int n){
```

```
    int res;
```

```
    if (n == 0)
```

```
        res = 1;
```

```
    else res = n* fact(n-1);
```

```
    return res;
```

```
}
```

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

```
    System.out.println("Le carré de 5 et de 7 vaut "+carreDeLaSomme(5,7));
```

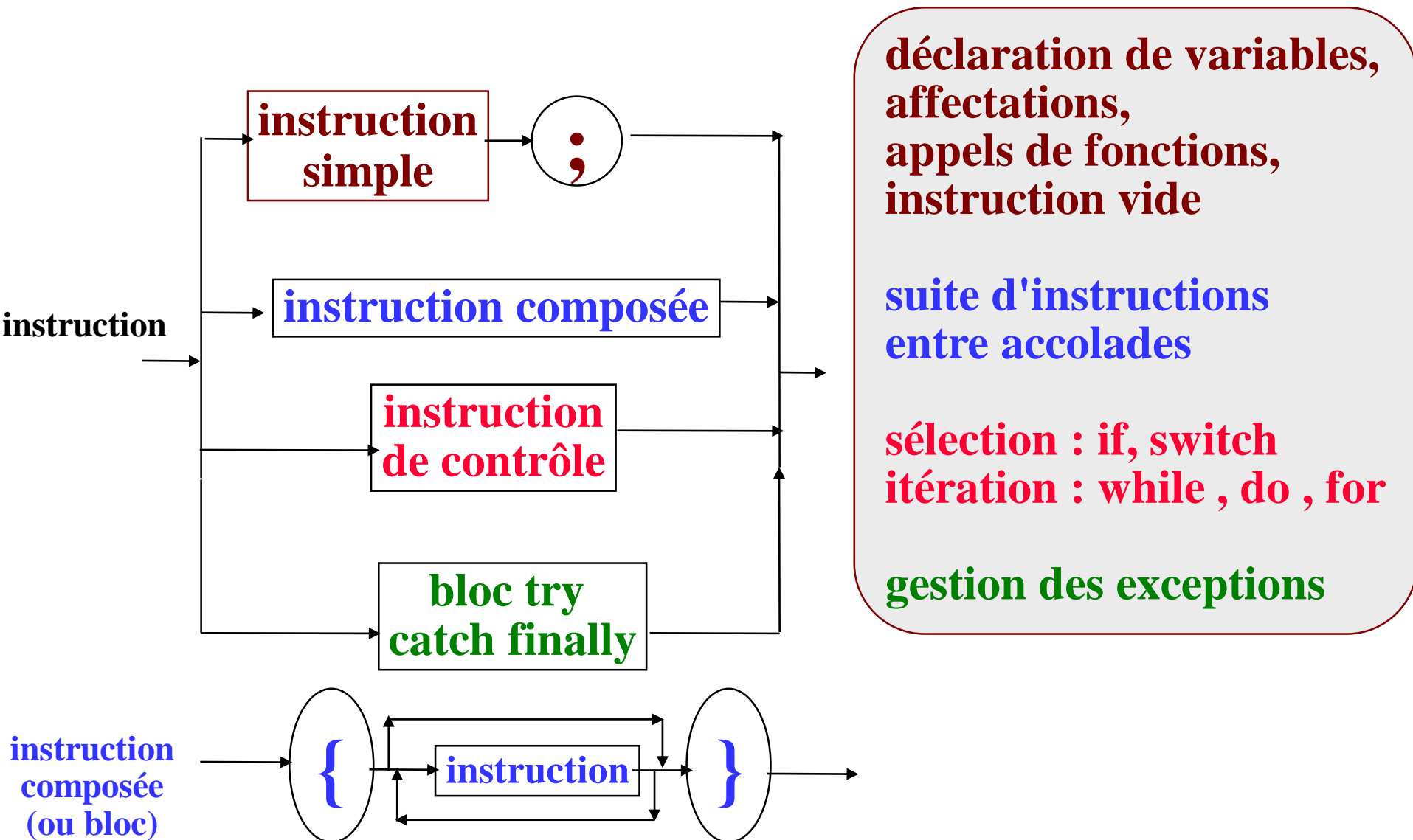
```
    System.out.println("fact(5)=" +fact(5));
```

```
}}
```

En-tête de la fonction

**Bloc
d'instructions
définissant
la fonction**

Différentes catégories d'instructions Java



Opérateurs d'affectation

=

est l'opérateur général d'affectation
Les autres opérateurs d'affectation
permettent d'alléger l'écriture
pour certains cas particuliers

+= -=
***= /= % =**
et avec tous les
opérateurs binaires

+=

a += exp ;équivaut à **a = a + exp;**

-=

a -= exp ;équivaut à **a = a - exp;**

--
++

--

a--; équivaut à **a= a-1;**

++

a++; équivaut à **a= a+1;**

Les types simples

2 catégories de types de données en JAVA :

**LES 8 TYPES SIMPLES
(tous prédéfinis)**

byte short int long // les entiers
float double // les réels
boolean // les booléens
char // les caractères

LES CLASSES

exemples de classes:

java.lang.**String**

java.util.**Date**

java.util.**List**

**Les éléments représentables par un type simple ne dépendent
ni de la machine, ni du compilateur**

Les constantes numériques littérales

nombre	type	
523	int	
3 000 000 000	long	(> 0x7FFFFFFF)
523L	long	fini par L ou l
7.45	double	double par défaut
7.45F	float	fini par F ou f
1D	double	fini par D ou d
6.023E23	double	

nombre	base	
377	décimal	
0377	octal	commence par 0
0xAF5	hexadécimal	commence par 0x

Les booléens

2 valeurs booléennes

true

false

Le résultat d'un test
est de type boolean

```
int x = 9;  
boolean b = true;  
boolean c = (x>5);
```

Attention

Pas de conversions automatiques entre entiers et booléens

```
boolean b = 1;  
if (1) {...};
```

Les opérateurs booléens

Symbole	Correspond à	Exemple	Évaluation en court circuit
!	Négation logique	! (x > 5)	
&	Et logique	(x>1)&(x<8)	non
&&	Et logique	(x>1)&&(x<8)	oui
	Ou logique	(x>10) (x<8)	non
	Ou logique	(x>10) (x<8)	oui
^	Ou exclusif	(x>1)^(y<8)	

Les opérateurs && et || réalisent les mêmes opérations que & et | mais avec une évaluation en court-circuit:
le second argument n'est alors évalué que lorsque sa valeur est susceptible de modifier le résultat

Les caractères en JAVA

UTILISE LE STANDARD UNICODE
et permet de coder 65536 caractères (2 octets)

Caractères "Unicode" de '\u0000' to '\uffff' (sur 2 octets)

De '\u0000' à '\u007f' (0 à 127) : identiques aux codes ASCII

De '\u0080' à '\u00ff' (128 à 255): codes Latin-1

codage des caractères de 0 à 127

code ASCII

chiffres	'0': 48 '1': 49 '2':50 '9': 57
majuscules	'A': 65 'B' : 66 'C' : 67 ... 'Z' : 90
minuscules	'a' : 97 'b': 98 'c': 99 ... 'z': 122
autres	' ' : 32 '#': 35 '<' : 60 '{': 123

Exemple de fonctions statiques élémentaires 1 : fonctions rendant un résultat (suite)

```
/** rend le cube de 'x'*/
```

```
public static double cube(double x){  
    return x*x*x;  
}
```

```
/** rend 'x' puissance 6 */
```

```
public static double puissance6(double x){  
    double x3 = cube(x); // on utilise la fonction définie ci-dessus  
    return x3*x3;  
}
```

Exemple de fonctions statiques élémentaires 2 : fonctions ne rendant pas de résultat

```
/** affiche un message de bienvenue pour une personne
 * connaissant le 'nom' et le 'prenom'*/
public static void afficherBienvenue(String nom,String prenom){
    System.out.println("Bonjour, "+prenom+" "+nom);
}
```

```
/** affiche le menu pour un programme gérant des comptes */
public static void afficheMenu( ){
    System.out.println("1 : consulter le solde");
    System.out.println("2 : enregistrer une nouvelle opération");
    System.out.println("3 : consulter l'historique du compte");
}
```

void Les fonctions ne rendant pas de résultat ont **void** comme type de retour

L'instruction simple « return »

provoque la terminaison de la fonction
et le retour à la fonction appelante

Obligatoire si la fonction a une valeur de retour

return <expression>;

expression est la valeur de retour de la fonction

```
public static double cube(double a){  
    return(a*a*a);  
}
```

Rarement utilisé si la fonction n'a pas de valeur de retour

return;

le return est souvent implicite

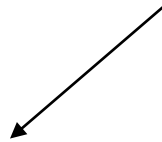
La fonction se termine "naturellement"

```
public static void afficherBienvenue(String nom,String prenom){  
    System.out.println("Bonjour, "+prenom+" "+nom);  
    return; // facultatif}
```


choix des identificateurs

le nom des identificateurs (paramètres, variables, fonctions, ...) doit pouvoir être **justifié**

```
public static void saluerUtilisateur(String nomUtilisateur)
```



Salue l'utilisateur du programme

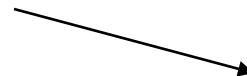


le nom de l'utilisateur
du programme

```
//class up5.mi.pary.term.Terminal  
public String readString(String message)
```



pour lire une chaîne de caractères



le message à afficher

CONVENTIONS SUR LES IDENTIFICATEURS

Types d'identificateurs	convention	exemple
classe	le premier symbole est une majuscule H elloWorld	S ystem
variable, fonction	le premier symbole est une minuscule p rintln	n om
constante	tout en majuscule	PI

Pour séparer les mots composant un identificateur, on met des majuscules.
exemple : read**S**tring Hello**W**orld

APPEL DE FONCTIONS statiques

L'appel d'une fonction statique est effectué comme suit :
<nom de la classe> . <nom de la fonction>(<arguments>)

```
package up5.mi.pary.jt.algo;  
public class MathUtil {  
    public static double cube(double x){  
        return x*x*x;  
    }  
}
```

MathUtil.java

```
package up5.mi.pary.jt.puis6;  
public class TestPuis6 {  
    public static void main(String[] args){  
        Terminal term = new Terminal("puissance 6",400,400);  
        double x = term.readInt("donner un nombre");  
        double y = up5.mi.pary.jt.algo.MathUtil.cube(x) ;  
        term.println("la puissance sixième de "+x+" est "+y*y);  
    }  
}
```

TestPuis6.java

**Rq: Si la classe est importée (explicitement ou implicitement),
le nom simple de la classe suffit.**

APPEL DE FONCTIONS statiques

Si la classe est importée (explicitement ou implicitement),
le nom simple de la classe suffit.

```
package up5.mi.pary.jt.puis6;  
import up5.mi.pary.jt.algo.MathUtil;  
public class TestPuis6 {  
    public static void main(String[ ] args){  
        Terminal term = new Terminal("puissance 6",400,400);  
        double x = term.readInt("donner un nombre");  
        double y = MathUtil.cube(x) ;  
        term.println("la puissance sixième de "+x+" est "+y*y);  
    }  
}
```

TestPuis6.java

Appel à une fonction statique de la même classe

```
package up5.mi.pary.jt.algo;
public class MathUtil{

    public static int cube(int x){
        return x*x*x;
    }

    public static void main(String[] args){
        Terminal term = new Terminal("cube",400,400);
        int x = term.readInt("donner un entier");
        int y = cube(x); // ou MathUtil.cube(x)
        term.println("le cube de "+x+" est "+y);
    }
}
```

L'appel à une fonction statique de la même classe peut se faire en mentionnant simplement le nom de la fonction

Les conditionnelles

if (<condition>) <instruction>

if (<condition>) <instruction> else <instruction>

switch (<expression>){

 case <valeur> : <instructions>

 ...

}

if (<test>) <instruction> else <instruction>

if (<test>) /* toujours entre parenthèses */
<instruction> /*exécutée si test vrai*/
else **<instruction>** /*exécutée si test faux*/
partie else facultative

if avec else

```
if (r>0)
    System.out.println("positif");
else
    System.out.println("négatif ou nul");
```

L'instruction à choix multiples :SWITCH

```
Terminal term = new Terminal("switch",400,400);
char c = term.readChar("Repondre Oui ou Non");
switch (c){
    case 'O': term.println("Vous m'avez répondu Oui" );break;
    case 'N': term.println(" Vous m'avez répondu Non" );break;
    default: term.println(" Je n'ai pas compris" );break;
}
```

Repondre Oui ou Non
O
Vous m'avez répondu Oui

```
Terminal term = new Terminal("switch",400,400);
char c = term.readChar("Repondre Oui ou Non");
switch (c){
    case 'O':
    case 'o' : term.println("Vous m'avez répondu Oui" );break;
    case 'n' :
    case 'N': term.println(" Vous m'avez répondu Non" );break;
    default: term.println(" Je n'ai pas compris" );break;
}
```

Repondre Oui ou Non
O
Vous m'avez répondu Oui

SWITCH : un exemple avec une expression entière

```
Terminal term = new Terminal("switch",400,400);
int n = term.readInt("Repondre 1 ou 2");
switch (n){
    case 1 : term.println("Vous m'avez repondu 1" );break;
    // surtout pas case '1' avec des apostrophes car cela désigne alors un caractère !
    case 2: term.println("Vous m'avez repondu 2" );break;
    default: term.println("Je n'ai pas compris" );break;
}
```

Attention : ne pas confondre 1 et '1' : le compilateur Java ne signale pas l'erreur

traduire une instruction switch en if

```
switch (c){  
  case 'O':  
  case 'o' : term.println("Vous m'avez répondu Oui" );break;  
  case 'n' :  
  case 'N': term.println(" Vous m'avez répondu Non" );break;  
  default: term.println(" Je n'ai pas compris" );break;  
}
```

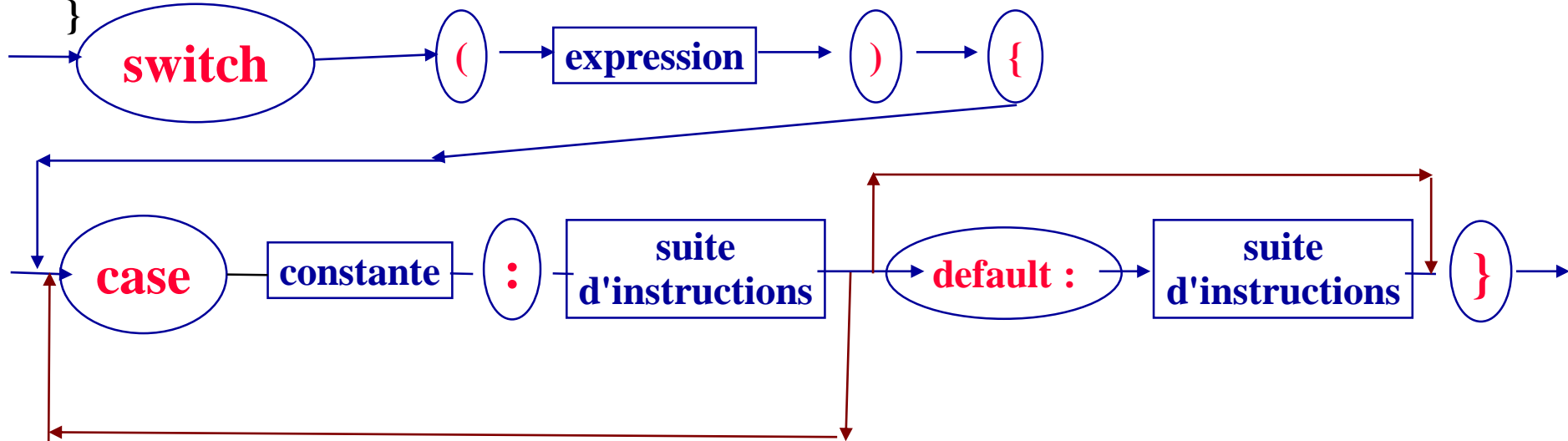
est équivalent à

```
if ((c=='O')||(c=='o'))  
  term.println("Vous m'avez répondu Oui" );  
else if ((c=='N')||(c=='n'))  
  term.println(" Vous m'avez répondu Non" );  
else term.println(" Je n'ai pas compris" );
```

remarques : || est l'opérateur de disjonction et se lit "OU"
== est l'opérateur d'égalité

SWITCH: syntaxe

```
switch (c){  
  case 'O': term.println("Vous m'avez répondu Oui");break;  
  case 'N': term.println(" Vous m'avez répondu Non" );break;  
  default: term.println(" Je n'ai pas compris" );break;  
}
```



restriction

Le type de l'expression testée doit être un type entier ou "char"
Depuis java7, également String

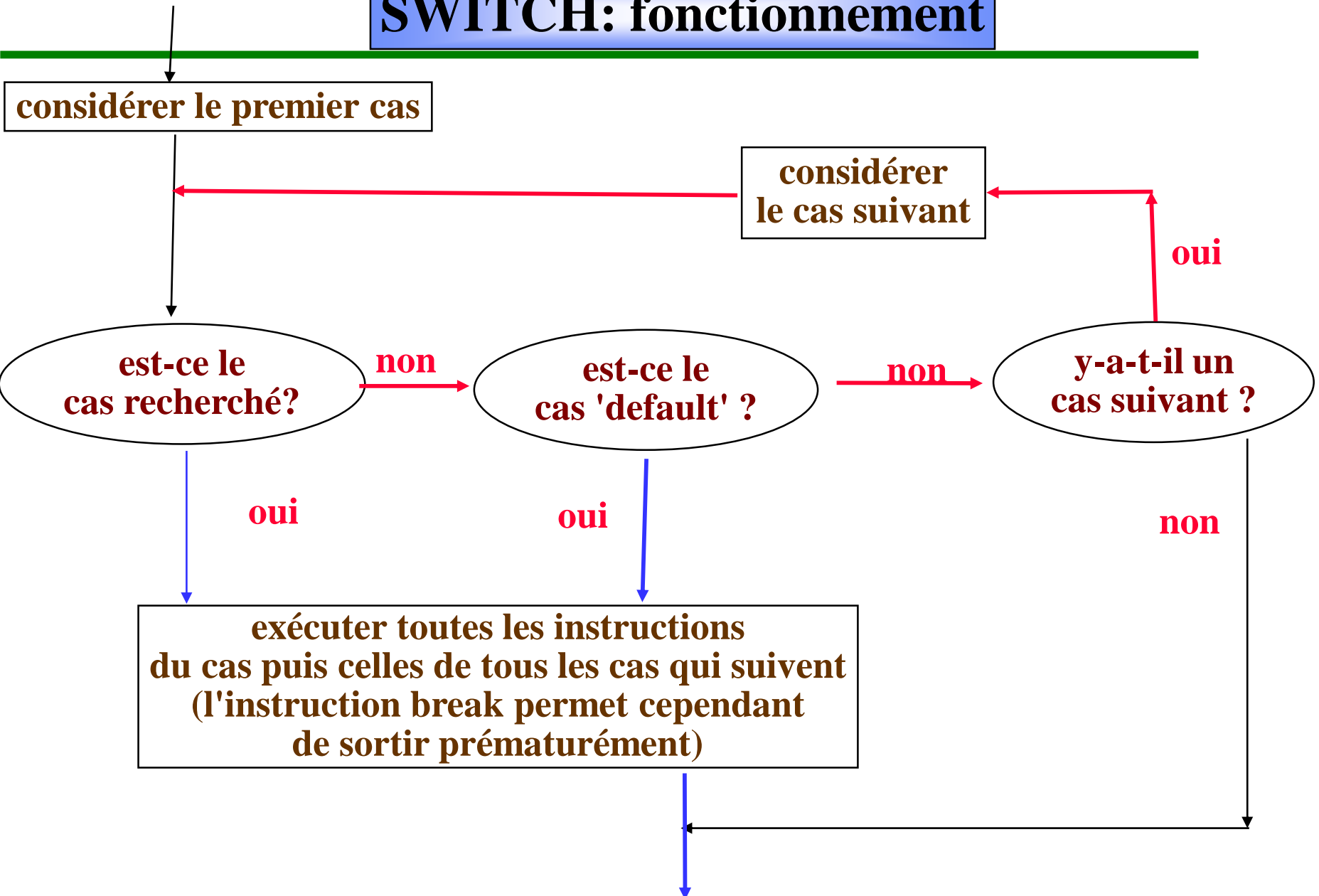
fonctionnement

Lorsqu'un cas est sélectionné, toutes les instructions
qui le suivent sont exécutées,
y compris celles des cas suivants!

break

L'instruction simple 'break' permet
de sortir prématurément de l'instruction switch

SWITCH: fonctionnement



while

```
while (<test>)  
    <instruction>
```



TANT QUE le test est vérifié
exécuter l'instruction

while : puissance nième d'un nombre entier

un exemple d'utilisation dans une fonction

```
/**@return la puissance entière d'un double */  
public static double puissance (double n, int p){  
    int i=1;  
    double res=1;  
    while (i<=p){  
        res=res*n;  
        i=i+1;  
    }  
    return res;  
}
```

```
public static void main(String [ ] args){  
    System.out.println("2^4="+puissance (2,4));}
```

2^4=16

do ... while

do <instruction>

while (<test>) ; // point virgule obligatoire



exécuter l'instruction

TANT QUE le test est vérifié

différence avec l'instruction while

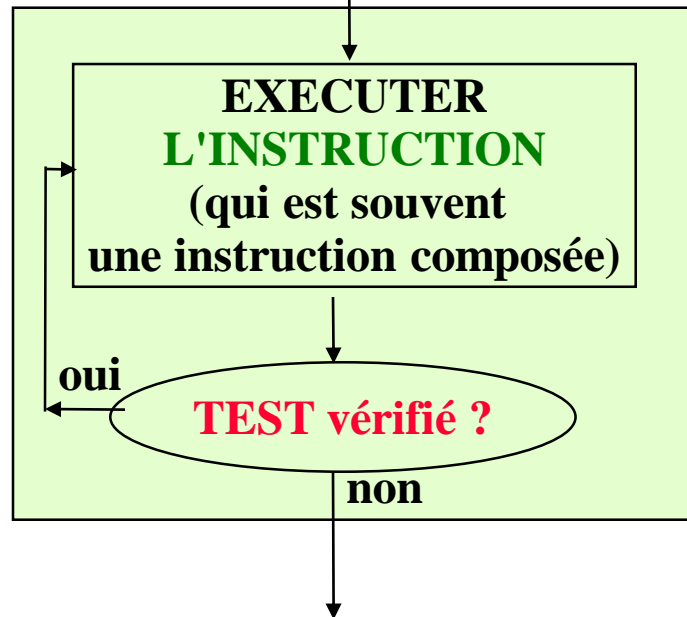
**LE TEST EST REALISE
EN FIN DE BOUCLE**

```
do <instruction> while (<test>) ;
```

exemple

```
int annee;  
do {  
    annee = term.readInt('Donner une année >= 1880');  
}  
while (annee < 1880);
```

fonctionnement



utilisation

dans les cas (assez peu fréquents)
où au moins une itération doit être effectuée

for

```
for (<expInit>;<expTest>;<expIncr>)  
    <instruction>
```

Permet de traduire des énoncés comme :

Pour i variant de i_{\min} à i_{\max} faire <instruction>

Répéter n fois <instruction>

Pour i variant de i_{\min} à i_{\max} et tant que test vérifié faire <instruction>

for (<expInit>;<expTest>;<expIncr>) <instruction>

Pour i variant de i_{\min} à i_{\max} faire <instruction>

p=1;

Pour i variant de 1 à n faire p=p*i;

```
int p = 1;  
for (int i=1 ; i<= n ; i++)  
    p=p*i;
```

Répéter n fois <instruction>

répéter 40 fois écrire "-"

```
for (int i=1 ; i<= 40 ; i++)  
    System.out.println('-');
```

Autre utilisation de la boucle for

Pour i variant de i_{\min} à i_{\max} et tant que test vérifié
faire **<instruction>**

$p=1$;

Pour i variant de 1 à n et tant que $p < 1000$ faire $p=p*i$;


```
int p = 1;  
for (int i=1 ; ( i<= n)&& (p<1000) ; i++)  
    p=p*i;
```

syntaxe

for (**<expInit>**;**<expTest>**;**<expIncr>**) **<instruction>**

for : la variable de boucle

i est souvent déclarée à l'intérieur de l'instruction for



```
for (int i=1;i<=5;i++)  
    System.out.println("'" + i + " " + i*i*i );
```

La variable i est locale à la boucle for

1	1
2	8
3	27
4	64
5	125



remarque

on aurait pu aussi écrire:

```
int i;  
for (i=1;i<=5;i++) // la variable i n'est plus locale  
    System.out.println("'" + i + " " + i*i*i );  
System.out.println(i); // on peut l'utiliser après la boucle !
```

conventions:

Sauf exception, la variable de boucle est déclarée dans l'instruction for.

Les tableaux

**Un tableau est une collection ordonnée
de variables du même type**

exemple :

tableau de 8 float

tableau de 5 String

tableau de 10 Terminal

**Chacune de ces variables est repérée par son
indice (qui est sa position dans le tableau):
si un tableau comporte n éléments,
les indices vont de 0 à $n-1$**

Les tableaux

nombre d'éléments: l'attribut length

On peut accéder au nombre d'éléments d'un tableau grâce à l'attribut "length"

```
String [ ] args= new String[12];  
System.out.println(args.length);  
// args.length désigne la longueur  
// du tableau args
```

12



On ne peut pas modifier la valeur de "length" :
c'est un attribut constant

~~args.length = 4;~~

Valeurs par défaut des variables

Lorsqu'un tableau est créé, une valeur par défaut est affectée à chacun des éléments de ce tableau.

type	valeur
boolean	false
char	\u0000
byte,short,int,long	0
float,double	0.0
type «classe »	null

AFFICHAGE DES ELEMENTS D'UN TABLEAU

```
/** affiche les éléments du tableau 'tab' sur le terminal 'term'*/  
public static void afficher(Terminal term,int [] tab){  
    for (int i = 0; i < tab.length ; i++){  
        term.print(tab[i]);  
        if (i!=tab.length-1) term.print(" ");  
    }  
    term.println();  
}  
  
public static void main (String[] args){  
    int [ ] tabInt = {32,12,23,21,5};  
    Terminal term = new Terminal("affichage de tableaux",400,400);  
    afficher(term , tabInt);  
}
```


SAISIE DES ELEMENTS D'UN TABLEAU

```
/* rend un tableau de 'n' entiers demandés
à l'utilisateur à l'aide du terminal 'term'*/
public static int [ ] saisieTabInt(Terminal term,int n){
int[] tab =new int[n];/* le résultat est un tableau de n éléments*/
for (int i = 0; i < n ; i++)
    tab [i] = term.readInt("");/* saisie de l'élément d'indice i*/
return tab;
}

public static void main (String [ ] args){
    Terminal term = new Terminal("Saisie et affichage d'un tableau");
    int [ ] tabInt=saisieTabInt(term,5);
    afficher(term,tabInt);
}
```

SOMME DES ELEMENTS D'UN TABLEAU

```
/* rend la somme des éléments du tableau 'tab' */  
public static int somme (int [ ] tab){  
int res=0;  
for (int i = 0; i < tab.length ; i++)  
    res=res + tab[i] ;  
return res;  
}
```

```
public static void main (String [ ] args){  
    int [ ] tabInt = {32,12,23,21,5};  
    System.out.println(somme (tabInt) );  
}
```

93

TEST DE L'APPARTENANCE D'UN ELEMENT A UN TABLEAU

```
int [ ] t = {1,2,3,4,5};
```

3 est-il un élément de t ?

t [0] == 3 ? non

t [1] == 3 ? non

t [2] == 3 ? oui 3 est un élément de t

12 est-il un élément de t ?

t [0] == 12 ? non

t [1] == 12 ? non

t [2] == 12 ? non

t [3] == 12 ? non

t [4] == 12 ? non

tant que il reste des éléments à tester
et que on n'a pas trouvé l'élément cherché
tester l'élément suivant

```
trouve=false;  
i=0; /*indice de l'élément à tester*/  
tant que (i<t.length)  
    et (! trouve) // (trouve ==false)  
    si t[i] est l'élément cherché  
        alors trouve=true  
    sinon i=i+1;
```

TEST DE L'APPARTENANCE D'UN ELEMENT A UN TABLEAU

```
trouve=false;
i=0; /*indice de l'élément à tester*/
tant que (i<t.length)
    et (! trouve) // (trouve ==false)
    si t[i] est l'élément cherché
        alors trouve=true
        sinon i=i+1;
```

```
/* teste si 'e' est un élément du tableau 't'*/
```

```
public static boolean appartient(int [ ] tab , int elt){
```

```
boolean trouve=false;
```

```
int i=0;
```

```
while ((i<tab.length)&&(!trouve)){
```

```
    if (tab [i] == elt)
```

```
        trouve=true;
```

```
    else i++;
```

```
    }
```

```
return trouve;
```

```
}
```

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

```
    Terminal term = new Terminal("test de l'appartenance à un tableau",400,400);
```

```
    int [ ] tab= saisieTabInt(5);
```

```
    int x=term.readInt("quel nombre dois-je chercher ?");
```

```
    if (appartient(tab,x))
```

```
        term.println( x+" appartient au tableau ");
```

```
}
```