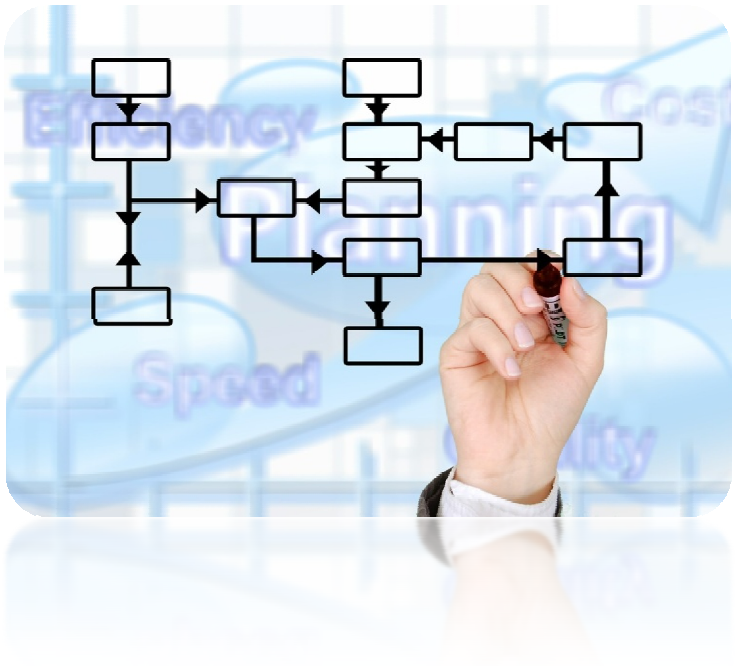
Objectifs : savoir créer un programme simple en utilisant le langage Java et le JDK.



1- Java et installation du JDK

2- Créer et exécuter un programme

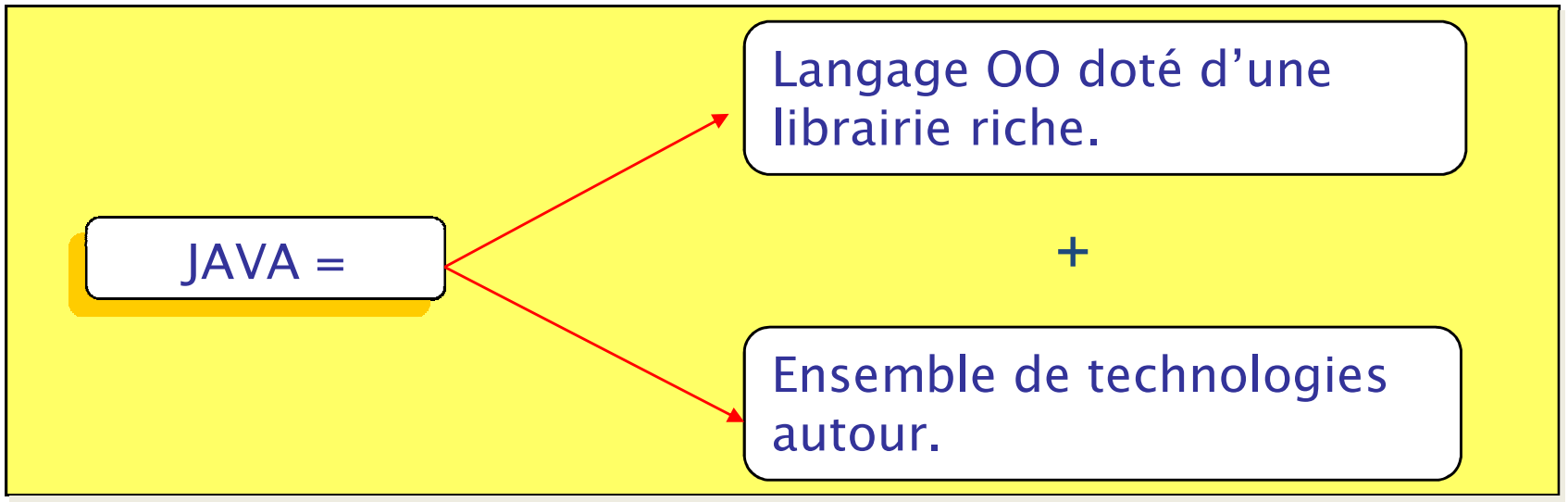
3- Variables, types et opérateurs

4- Coder les structures alternatives

5- Coder les structures répétitives

Le langage Java est un langage orienté objet généraliste et très répandu.

1 Qu'est-ce que le langage Java?



Le code Java est exécuté par une machine virtuelle

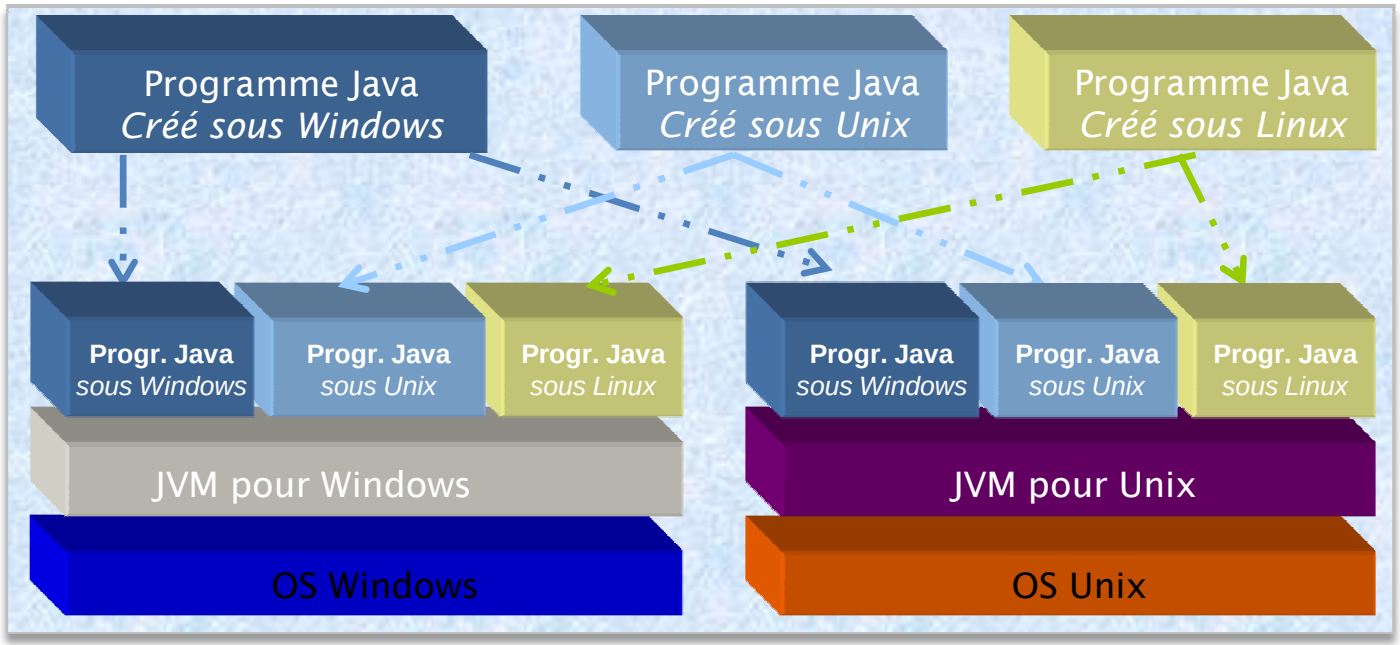
1 Qu'est-ce que le JRE et le JDK

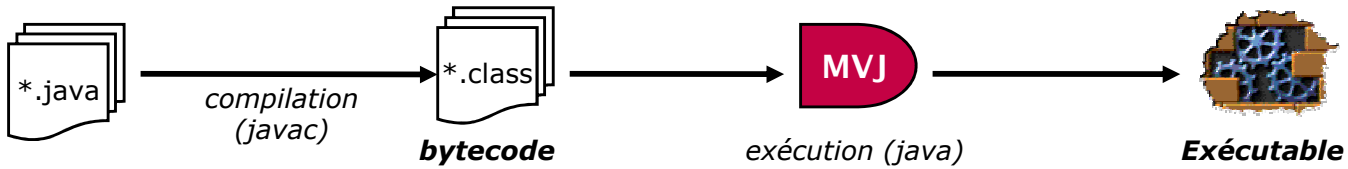
Logiciel	Signification
JRE (JVM)	**Java Virtual Machine (JVM) = Java Runtime Environment (JRE)** API formant une plateforme d'exécution des programmes Java et jouant le rôle d'interface entre les programmes et le système d'exploitation.
JDK	**Java Development Kit (JDK)** Il s'agit du JRE + Outils de mise au point de programmes.

Exemples d'outils fournis :
javac : Compilateur
java : Pour lancer l'exécution d'une application Java
javadoc : Pour la documentation

JVM = Machine virtuelle Java

2 Principe de fonctionnement





Le JDK contient toutes les fonctions nécessaires pour commencer à développer.

3 Installer le JDK sous Windows

Pour commencer à coder en Java, installez l'environnement de développement Java (JDK) :

- Télécharger et installer le JDK:
  https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Windows x86	197.34 MB	jdk-8u191-windows-i586.exe
Windows x64	207.22 MB	jdk-8u191-windows-x64.exe

- Créer la variable d'environnement JAVA_HOME, contenant le chemin vers le JDK installé. Par exemple sous Windows : **JAVA_HOME** vers **C:\Program Files\Java\jdk1.8.0_45**
- Pour vérifier la version installée : **c:\>java -version**
- A la variable **PATH** ajoutez le chemin du compilateur : **;%JAVA_HOME%\bin**

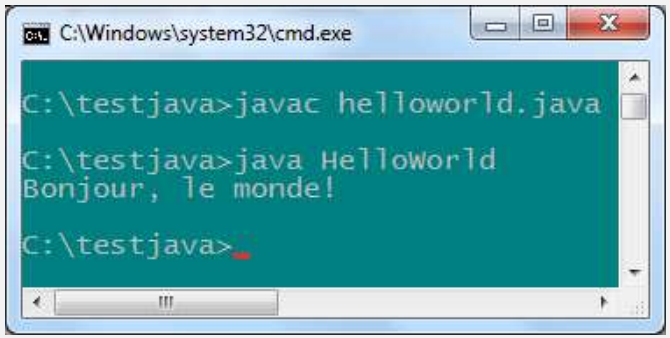
Installez-vous Notepad++ pour coder

1 Premier programme : HelloWorld

- Créer le fichier **c:\testjava\HelloWorld.java** dans votre éditeur Notepad++.
- Lancer une fenêtre MS DOS en tapant **cmd** dans la recherche Windows.
- A l'invite tapez **: cd c:\testjava** puis ensuite : **javac helloworld.java**
- Lancer ensuite : **java HelloWorld** pour exécuter votre code.

```
public class HelloWorld {

    // Méthode main = démarre le programme
    public static void main(String[] args) {
        System.out.println("Bonjour, le monde!");
    }
}
```



HelloWorld.java

Quelques conventions d'écriture pour le code Java.

1 Commenter votre code Java

- Pour un commentaire sur une ligne, utilisez deux caractères slash : **//**
- Si le commentaire est multiligne, encadrez-le entre **/*** et ***/**.
- Un commentaire relatif à la documentation de code (**Javadoc**) sera encadré par **/**** et ***/**.

Quelques conventions d'écriture pour le code Java.

2 Instructions, casse

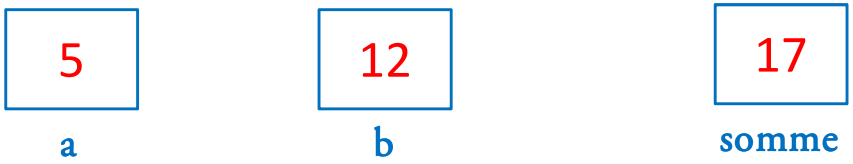
- Une instruction (expression) doit se terminer par un point-virgule : « **;** ».
- Un **bloc d'instruction** peut contenir une ou plusieurs instructions. Utilisez l'accolade ouvrante « **{** » pour débuter le bloc, et l'accolade fermante « **}** » pour le fermer.
- **Java est sensible à la casse**. Donc la variable **mavar** sera différente de la variable **Mavar**.

Java est fortement typé. Pour toute variable créée, il faut préciser son type.

3 Variables, identificateur

Rappelons qu'une variable est une case mémoire permettant de stocker des données.

Elle doit avoir un **identificateur** et être d'un certain **type**.



4 Une constante

Une constante ne change pas dans le programme. ***Il ne faut pas la déclarer plusieurs fois***.

Chaque variable doit avoir un type.

5 Déclarer des variables : syntaxe

- L'identificateur doit être précédé du type.
- On peut affecter directement la variable ou le faire plus tard.
- L'identificateur débute par une **minuscule**, et ne doit pas contenir de caractère spécial.

  S'il est composé de plusieurs mots, il faut utiliser la **notation camelisée**.
- Si on veut créer une constante, faire précéder la déclaration du modificateur **final**.
- On peut faire plusieurs déclarations sur une même ligne.

```
1  // Une variable
2  int monEntier;
3
4  // Plusieurs variables
5  int tonEntier, sonEntier, leurEntier;
6
7  // Initialisation
8  int autreEntier = 5 ;
9
10 // Une constante
11 final int unFlottant;
```

Quelques variables

Java gère plusieurs types.

6 Tableau des différents types

On a 3 catégories de types **« primitifs »** : les booléens, les nombres et les caractères. Ces types ne sont pas des classes.

Type	Taille (bits)	Signe	Valeur par défaut	Format	Commentaire
byte	8	signé	0	IEEE*	+ Les types primitifs passent par **valeur**. + Les objets passent **aussi par valeur** : une copie de leur référence est transmise.
short	16	signé	0	IEEE	
int	32	signé	0	IEEE	
long	64	signé	0	IEEE	
float	32	signé	0.0F	IEEE	
double	64	signé	0.0D	IEEE	
char	16	non signé	0x00	Unicode2	
boolean	8	non signé	false		

* L'IEEE est une norme pour la représentation des nombres à virgule flottante en binaire.

Java gère plusieurs types.

7 Les types arithmétiques (nombres)

Ci-contre quelques déclarations de nombres.

- La valeur des entiers de plus de 32 bits (long) finissent par le symbole **L** ou **l**.
- Les valeurs des réels peuvent :

  + être en simple précision.
  + terminer par **d** ou **D** pour un double.
  + terminer par **f** ou **F** pour un float.

```
// short, int
short monshort =32000;
int minInt = 50000;

// long
long monLong = 30000000;
long = autreLong = 5L

// float, double
float monFlot = (float)2.5;
float autreFloat = 2.5F;
double monDouble = 6.5e10;
```

Quelques déclarations

Java gère plusieurs types.

8 Les caractères et les chaines

Les caractères sont de type primitif **char**, alors que les chaînes de caractères ne le sont pas. Les chaînes sont de type **String** = ce sont des instances de la classe String.

- Un caractère ou une chaîne est placé entre guillements.
- Quelques caractères d'échappement :

Caractère	Signification
\n	Ligne suivante
\r	Paragraphe suivant
\t	Tabulation
\'	Apostrophe
\"	Guillemet
\\	Antislash

```
1  // Une chaine
2  String  maChaine    = "Hello world \n";
3
4  // Caractères
5  char    monChar = 'c';
6  char    autreChar   = 48;
7
8  // Entier
9  int     monEntier   = 'a';
```

Quelques déclarations

Java gère plusieurs types.

9 Le type boolean (booléen)

Les booléens, qui ne sont pas des nombres, peuvent prendre soit la valeur **true**, soit la valeur **false**. **false** est la valeur par défaut d'un booléen.

```
1  // Booléens
2  boolean monBool = true;
3  boolean $monBool; //Correct, mais à éviter
4
5  //Ces exemples ne compilent pas
6  int 8i; //Ne compile pas
7  char    val%;   //Que 0-9, a-z, A-Z, _ et $
8  double  switch; //switch est un mot réservé
```

Quelques déclarations

10 Les différents opérateurs en Java

Pour manipuler, évaluer les variables et les données qu'elles contiennent, on emploie des opérateurs et on construit des expressions.

- Arithmétiques : Utilisés pour réaliser les calculs mathématiques
- Affectations : Affectent des valeurs à des variables
- Comparaison : Comparent des opérandes et retourne une valeur booléenne
- Logique : Pour accomplir des opérations booléennes sur des opérandes booléens
- Ternaire : Opérateur issu du langage C.
- Concaténation : Accomplit la concaténation de chaînes
- D'instance : Vérifie si une donnée est d'une certaine instance.

11 Opérateurs arithmétiques binaires

Ils permettent de faire des calculs. Dans les exemples, la valeur initiale sera toujours égale à **11**.

Signe	Nom	Signification	Exemple	Résultat
+	plus	addition	x + 3	14
-	moins	soustraction	x - 3	8
*	multiplé	multiplication	x * 2	22
/	divisé	division	x / 2	5.5
%	modulo	Reste division	x % 5	1
=	a la valeur	affectation	x = 5	5

```
15  // Variables
16  int x = 100; int y = 200;
17
18  // Addition
19  int somme = x + y;
20
21  // Soustraction
22  int difference = y - x;
23
24  // Multiplication
25  int produit = x * y;
26
27  // Division
28  int quot = x / y;
29
30  // Modulo
31  int reste = x % y;
```

Exemple

12

Opérateurs arithmétiques unaires

Ils permettent de faire des calculs. Dans les exemples, **x** est égal à **3**.

Signe	Description	Exemple	Signification	Résultat
++x	Pré-incrémente	y = ++x	3 puis plus 1	y = 4 x = 4
-+x	Pré-décrémente	y = --x	3 puis moins 1	y = 2 x = 2
x++	Post-incrémente	y = x++	3 puis plus 1	y = 3 x = 4
x--	Post-décrémente	y = x--	3 puis moins 1	y = 3 x = 2
-	Inverse	-x	L'inverse	X = -3

13

Opérateurs d'affectation (opérateurs associatifs)

Ils permettent d'assigner une valeur à une variable. Si **x** vaut **11** et **y** vaut **5** :

Signe	Description	Exemple	Signification	Résultat
=	affectation	y = x	y contient 5	y = 5 x = 5
+=	plus égal	x += y	x = x + y	x = 16
-=	moins égal	x -= y	x = x - y	x = 6
*=	multiplé égal	x *= y	x = x * y	x = 55
/=	divisé égal	x /= y	x = x / y	x = 2.2
%=	modulo égal	x %= y	x = x % y	x = 1

14 Opérateurs de comparaison

Ils permettent de faire des comparaisons. A l'issue de la comparaison de deux opérandes, une valeur booléenne **true** ou **false** est retournée. Nous considérons **x** = **11**.

Signe	Nom	Exemple	Résultat
==	égale	x == 11	true
!=	différent	x != 11	false
>	supérieur à	x > 11	false
<	inférieur à	x < 11	false
>=	supérieur ou égal à	x >= 11	true
<=	inférieur ou égal à	x <=11	true

Exemple

```
1  boolean resultat;
2
3  // true si var1 > var2, false var2 > var1
4  resultat = var1 > var2;
```

15 Opérateurs logiques

Les opérateurs logiques sont utilisés pour savoir si deux opérandes booléens sont égaux.

Signe	Nom	Exemple	Résultat
&&	et	(condition) && (condition2)	condition1 **ET** condition2
||	ou	(condition) || (condition2)	condition1 **OU** condition2
!	non	!x	false si **x** contient **true**
^	ou exclusif	a ^ b	vrai si seul a ou b vrai

16 Opérateur ternaire (issu de C)

Syntaxe : **(condition) ? (expression1) : (expression2) ;** ----->

Renvoie **expression1** si condition est vraie et e**xpression2** dans le cas contraire.

```
33  // Opérateur ternaire
34  int min = (val1 < val2) ? val1 : val2;
```

Exemple

17 Concaténation de chaines

On peut mettre bout à bout des chaines en utilisant l'opérateur de concaténation **+**.

```
1  // Création de chaines
2  String nom = "DUPONT";
3  String prenom = "Jean";
4
5  // Concaténation
6  String identite = nom + "  "  + prenom;
7
8  // Affichage
9  System.out.println("Je suis : " + identite);
```

Concaténation

18 Opérateur instanceof

Il permet de déterminer la classe d'appartenance d'un objet donné. Il retourne un booléen.

```
1  // Création d'un objet
2  Livre monlivre = new Livre('TINTIN', 400);
3
4  if(monlivre instanceof Livre){ // true
5      monlivre.getInfos();
6  }
```

instanceof

Le casting = transtypage : permet de convertir un type en un autre.

19

Les conversions de type

Il y a 4 contextes de conversion (cast) :

- Conversion explicite
- Affectation
- Appel de méthode
- Promotion arithmétique

```
1   double f = 3.14;
2   int i = 0, j = 0;
3   short monShort = 1;
```

```
5   // Conversion explicite
6   j = (int) f;     // float -> int
7   float ff = (float) 3.14;
```

```
9   // Affectation
10  i = monShort;    // short -> int
```

```
12  // Appel de méthode : int obj.m(int i)
13  obj.m(monShort);     //short -> int
```

```
17  /*
18     Promotion arithmétique :
19     divison d'un entier et d'un flottant : l'entier i
20     est converti en flottant, puis la division
21     flottante est calculée
22  */
23  f = i / (double)j;  //f vaut 0.333…
```

Le casting = transtypage : permet de convertir un type en un autre.

20 Les conversions de type (suite)

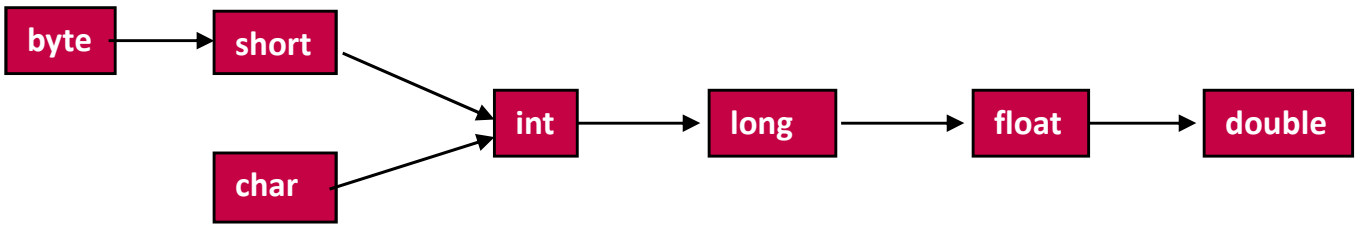
- Certaines conversions provoquent une perte de valeur : **float** en **int**, int en short, short en byte.
- Le type **boolean** ne peut être converti en entier.
- Le type **int** ne peut pas etre traité comme un booléen.

--->

```java
int i = 3;

// Impossible en JAVA !!!!
if ( i ) {
    // Ne peut pas compiler
    …..
}

// Correct en Java
if ( i != 0) {
        …..
}
```



Ce sont des structures permettant de prendre des décisions

1 Structure if à une seule instruction

```
1  // Une seule instruction
2  int  a = 1;
3  if ( a == 1) System.out.println("C'est OK.");
4  else System.out.println("C'est KO.");
```

2 Structure if classique

```
6  // If classique
7  int  a = 1;
8  if ( a == 1) {
9      System.out.println("Instruction 1.");
10     System.out.println("Instruction 2.");
11 }
```

3 Structure if .. else

```
14 // If avec else
15 int  a = 1;
16 if ( a == 1) {
17     System.out.println("Instructions1.");
18 }
19 else{
20     System.out.println("Instructions2.");
21 }
```

4 Structure if .. elseif .. else

```
24 // If else if else
25 int  a = 5;
26 if ( a == 1) { a++;}
27 else if ( a < 1) { a--;}
28 else if ( a > 10) { a--;}
29 else{System.out.println("Exécuté.");}
```

Ce sont des structures permettant de prendre des décisions

5 Structure if avec imbrication

```
32  // Imbrication de if
33  int  a = 5;
34  if ( a == 1) {
35      a++;
36
37      // if imbriqué
38      if ( a == 4) {
39          a = a * 2;
40      }
41  }
42  else{
43      System.out.println("Exécuté.");
44  }
```

Ce sont des structures permettant de prendre des décisions

6 Structure switch

```
47  // Switch = branchement
48  String role = "ADMIN";
49  switch(role){
50      case "VISIT": System.out.println("Visiteur.");
51                      break;
52
53      case "ADMIN": System.out.println("Boss.");
54                    break;
55
56      default:      System.out.println("Yes.");
57                    break;
58  }
```

Ces structures permettent de répéter des actions.

1 Boucles while et do..while

```
61  // while
62  int a = 1;
63  while(a < 10){
64      System.out.println(a);
65      a++;
66  }
67
68  // do..while
69  int a = 1;
70  do{
71      System.out.println(a);
72      a++;
73  }while(a < 10);
```

Remarques

- Eviter d'être enfermé dans la boucle
- **while :** on n'entre pas forcément dans la boucle.
- **do..while :** on entre au moins 1 fois dans la boucle.
- Très souvent on initialise 1 ou plusieurs variables en début de boucle.
- **while** et **do..while** : conseillés si on ne sait pas *a priori* combien de fois boucler.

Nota :

- **break** : sortir de la boucle
- **continue** : saute au pas suivant.
- Si on a **une seule instruction**, on peut omettre le bloc d'accolades.

Ces structures permettent de répéter des actions.

2 Boucles for et for (each)

```
76  // for
77  for(int a = 0; a < 10 ; a++){
78      System.out.println(a);
79  }
80
81  // for (each) / depuis Jdk 1.5
82  int tableau[] = {1, 2, 3, 4};
83  for(int element : tableau){
84      System.out.println(element);
85  }
```

Remarques

- Eviter d'être enfermé dans la boucle
- **for :** l'initialisation, la condition et l'incrémentation se font dans le for.
- **for** : conseillé si on sait *a priori* combien de fois boucler.
- **for (each)** : efficace pour parcourir tableaux, listes et objets.

Nota :

- **break** : sortir de la boucle
- **continue** : saute au pas suivant.
- Dans **int a**, **a** est local à la boucle et non visible à l'extérieur.