



Chapitre1

F. Karami



Présentation et Généralités de Java

Pourquoi JAVA?



- Le langage Java est un langage capable de s'exécuter sur n'importe quelle plate-forme car c'est d'une part un langage compilé et d'autre part un langage interprété.
- Le code source Java est transformé en de simples instructions binaires.
- (Byte Code= Instructions générées par le compilateur qu'un ordinateur abstrait peut exécuter).



Pourquoi JAVA?



- **Robuste et sûr :**
- *Peu de pièges.*
- *Pas de pointeurs*
- *Compilateurs très stricts car toutes les valeurs doivent être initialisées.*
- *Le traitement des exceptions est obligatoire.*
- *Les erreurs à l'exécution sont vérifiées tout comme les limites des tableaux.*



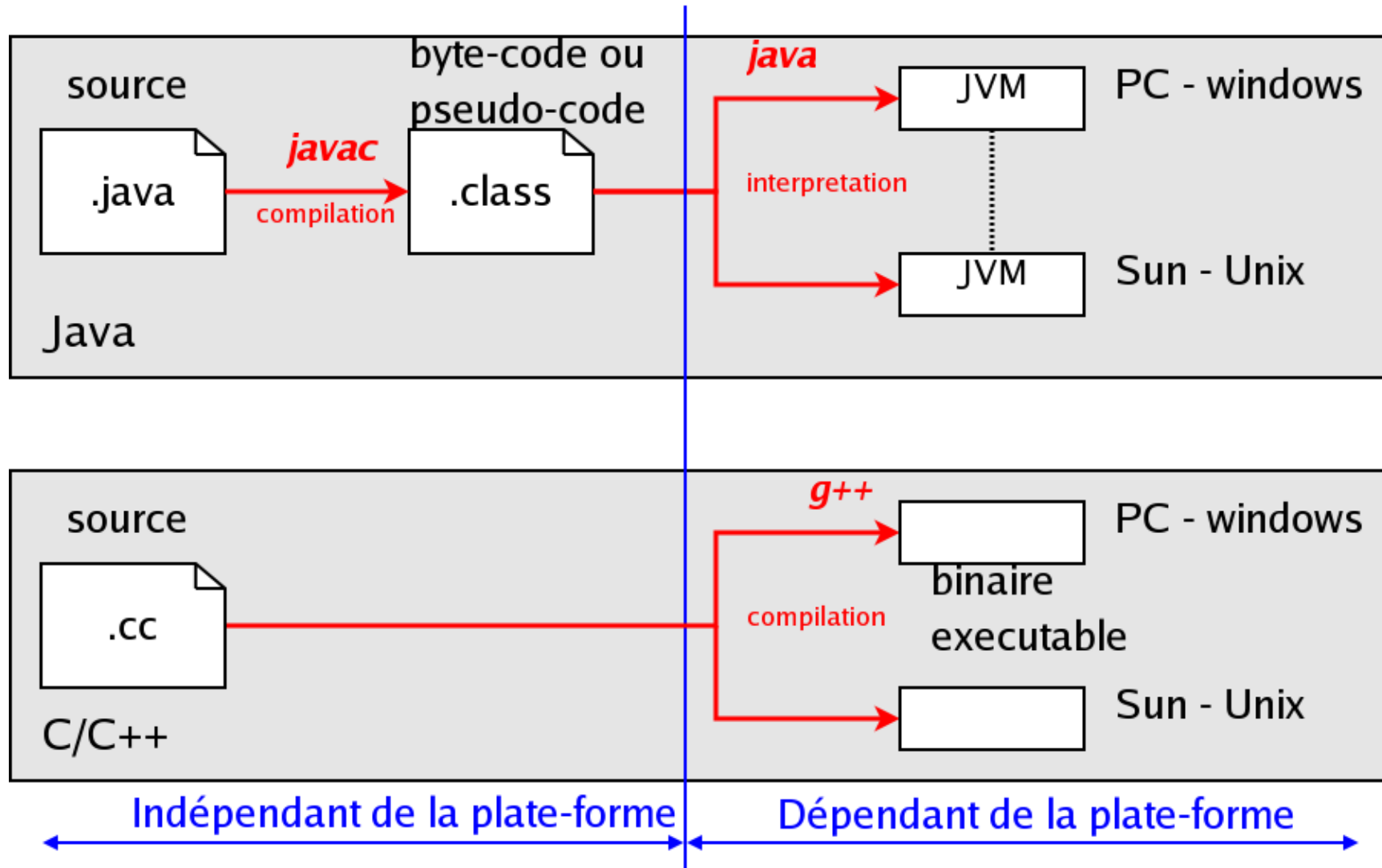
Pourquoi JAVA?



- **Sécurisé :**
- Allocation de la mémoire dynamique due au *GARBAGE COLLECTOR* (Ramasse Miettes).



Java : principe fondateur de sa portabilité





Choix de Java



- ***Java est un langage de programmation objet***
- ***Il est portable sur la plupart des plates-formes***
- ***C'est un langage généraliste ayant un très vaste d'application (réseau, base de données, calcul scientifique, etc)***
- ***Il intègre une interface graphique de haut niveau***
- ***Il existe de nombreuses bibliothèque de programmes dans des domaines très variés.***
- ***Le code produit (il s'agit d'un pseudo-code ou byte-code) indépendant de la plate forme utilisée.***
- ***Les programmes Java peuvent être exécutés sous forme d'applications indépendantes ou distribuées à travers le réseau et exécutées par un navigateur Internet sous forme d'applets.***
- ***Mais, il est moins rapide que C++ (calcul scientifique)***



Syntaxe Java



- Le code source d'un programme Java est contenu dans **plusieurs** fichiers d'extension .java
 - Une seule classe publique par fichier;
 - Le nom du fichier doit être le même que celui de la classe publique;
 - Par convention, le nom d'une classe commence toujours par une majuscule.

Le code source d'une classe commence par le mot-clé **class** suivi de son contenu :

```
class <nom de la classe> {  
    <contenu de la classe>  
}
```



Contenu d'une classe (6)



- Une classe est composée de plusieurs **membres** dont chacun est soit :
 - un **attribut** : variable typée
 - une **méthode** (ou **opération**) : ensemble d'instructions de traitement

```
class CompteBancaire {  
    String proprietaire;  
    double solde;  
  
    double getSolde() {  
        return solde;  
    }  
  
    void credite(double val) {  
        solde = solde + val;  
    }  
}
```

Attributs {

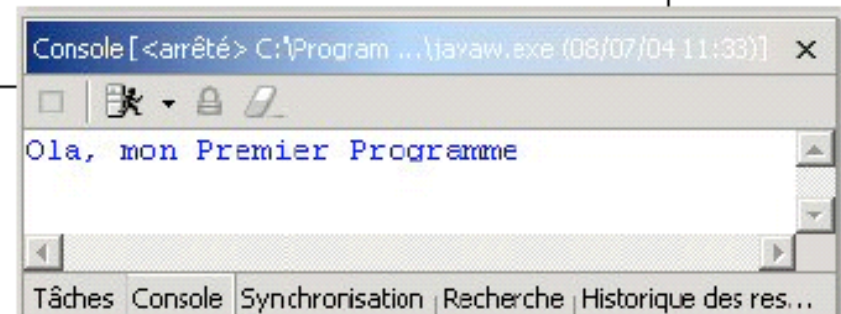
Méthodes {



Premier programme java



```
public class PremierProg {  
  
    public static void main (String[] argv) {  
        System.out.println("Ola, mon Premier Programme");  
    }  
}
```



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



Éléments du Langage

Syntaxe

Éléments du Langage



- **Commentaires**
- **Variables**
- **Constantes**
- **Types primitifs**
- **Les opérateurs et les expressions**
- **Les instructions de contrôles**



Commentaires



3 sortes de commentaires

- bloc

```
/* le code qui suit fait des choses tellement  
intéressantes qu'il faut plusieurs lignes rien  
que pour le décrire */
```

```
int france = 3;
```

```
int brazil = 0;
```

- fin de ligne

```
int sénégal = 1; // pas de commentaire
```

```
int france = 0; // non plus
```

- Généralisation de commentaire javadoc (nous le verrons plus tard)



Éléments du Langage

Variables
Constantes

Variables (1)



- Une variable, c'est une case mémoire
- Il faut
 - un nom
 - un type
- Java est un langage fortement typé



Variables (2)



- Déclaration des variables...

```
int qiFilles = 120;
```

déclaration de type



Variables (2)



- Déclaration des variables...

```
int qiFilles = 120;
```

nom de variable



Variables (2)



- Déclaration des variables...

```
int qiFilles = 120;
```

initialisation

Variables (2)



- Déclaration des variables...

```
int qiFilles = 120;
```



fin d'expression



Variables (3)



- Il faut initialiser les variables...

```
int qiFilles = 120;  
int qiGarcons; // alors, c'est combien ?  
double nbMoyen = (nombreF * qiFilles + nombreG *  
    qiGarcons) / (nombreF + nombreG)
```

- ...sinon (en Java)
 - initialisation par défaut (variables d'instance)
- ...sinon (en bien d'autres langages)
 - une valeur aléatoire




Variables (4)



- Nom de variable
 - Constitué de
 - lettres : a z E R t Y . . .
 - nombres : 3 1 4 5 . . .
 - certains autres caractères : _
 - Commence par lettre (minuscule par convention)

Variables (5)



```
int ru496, point3, maVariableAMoi; // ok  
 int 123hop, pour%cent; // non !
```

```
int mes_notes, Pi; // oui, mais non-standard  
int mesNotes; // mieux, usage standard
```

– Min- et majuscules sont différents

```
 int moiMeme, moimeme, Moimeme;
```

Constantes (1)



```
static final double L_PAR_100KM = 8.2;  
static final double PRIX = 1.013;  
double km = 437.3;  
System.out.println(km / 100.0 * L_PAR_100KM *  
    PRIX);
```

- Sont des constantes

L_PAR_100KM

PRIX // si seulement c'était vrai...



Constantes (2)

- Par convention, nom tout en majuscules

```
static final double L_PAR_100KM = 8.2;  
static final double PRIX = 5.8;
```

- Mot clé **final** interdit réaffectation de valeur
 - Mot clé **static**, nous le verrons plus tard



```
static final double PRIX = 5.8;
```

```
PRIX = 6.2; // nonono
```




Éléments du Langage

Types primitifs

Types primitifs



Types « primitifs »

- caractères
- numériques
 - `int`
 - `double`
- logique
 - `boolean`



Types Primitifs

Numérique (1)



- Tous les types d'entiers

byte 8 bit -128 .. 127

short 16 bit -32768 .. 32767


int 32 bit -2147483648 .. 2147483647

long 64 bit -9223372036854775808 .. 9223372036854775807

- Initialisés à 0 par défaut
- Entiers sont `int` par défaut

```
int i = -42;
```

```
int total = i + 17; // 17 est un int
```

```
 byte botal = i * 10; // faux résultat
```

Types Primitifs Numérique (2)




- Tous les types de flottants

float 32 bit $\pm 3.40283247E+38$.. $\pm 1.40239846E-45$

double 64 bit $\pm 1.79769313486231570E+308$..
 $\pm 4.94065645841246544E-324$

- Initialisés à 0.0

- Flottants sont **double** par défaut

 `double d = 3.14159; // ok`
`float f = 3.14159; // nonono !`
`float x = 3.14159f; // ok, faut spécifier`

- **double** est à préférer à **float**

Types Primitifs

Numérique (3)



```
int pieces10centimes = 5;  
int pieces20centimes = 12;  
  
double valeurJaunes = pieces10centimes * 0.1  
+ pieces20centimes * 0.2;
```


- Les entiers **int**
 - Des entités indivisibles
 - nombres cardinaux ou ordinaux
- Les flottants **double**
 - Des entités d'échelle continue
 - nombres réels

Types Primitifs

Numérique (4)



- Conversion automatique
 - permis s'il n'y a pas de perte de précision

```
short s = -39;  
int t = s + 356; // ok 16 bits -> 32 bits  
 s = t - 12; // nonono 32 bits -> 16 bits
```

- Conversion manuelle (*cast*)

```
int t = 356;  
short s = (short) (t - 30); // oui, c'est exprès  
t = (int) 365.25; // oui, sous ma responsabilité
```

Types Primitifs

Logique



- Le type boolean
- Valeurs
`true`
`false`
- Traité plus tard (voir Décisions)



Les opérateurs et les expressions

Types Primitifs

Arithmétique (1)



- Opérations élémentaires

addition	<code>a + b</code>
soustraction	<code>i - j</code>
multiplication	<code>quant * prix</code>
division	<code>qi / nbCafes</code>
modulo	<code>qi % nbCafes</code>
exponentiation	<code>Math.pow(x, n)</code>

Types Primitifs

Arithmétique (2)



- Affectation

```
t = a + b - c * d / e; // a + b - ((c * d) / e)
t = a + (b - c) * d / (e - 1);
```

- Utiliser () pour regrouper des termes

Types Primitifs

Arithmétique (3)



- Incrément / décrétement
 - l'expression du type...

```
heure = heure + 1;
```

- ...est si banale qu'il y a le raccourci

```
heure++;
```

Types Primitifs

Arithmétique (4)



<pre>int i = 0; i++;</pre>		<pre>int i = 0; i = i + 1;</pre>
<pre>int t = i++;</pre>		<pre>int t = i; i = i + 1;</pre>
<pre>int s = ++i; int r = --i;</pre>	équivalent à	<pre>i = i + 1; int s = i; i = i - 1; int r = i;</pre>
<pre>r += i;</pre>		<pre>r = r + i;</pre>
<pre>s *= i; i /= 2;</pre>		<pre>s = s * i; i = i / 2;</pre>

Types Primitifs

Arithmétique (5)



- **Autres fonctions**

Math.sin(x) sinus x (en radians)

Math.cos(x) cosinus x (en radians)

Math.tan(x) tangente x (en radians)

Math.exp(x) e^x

Math.log(x) $\ln x$, $x > 0$

Math.ceil(x) plus petit entier $\geq x$

Math.floor(x) plus grand entier $\leq x$

Math.abs(x) valeur absolue $|x|$

Décisions

Typage



- Type **boolean**
 - Prend des valeurs
 - **true**
 - **false**

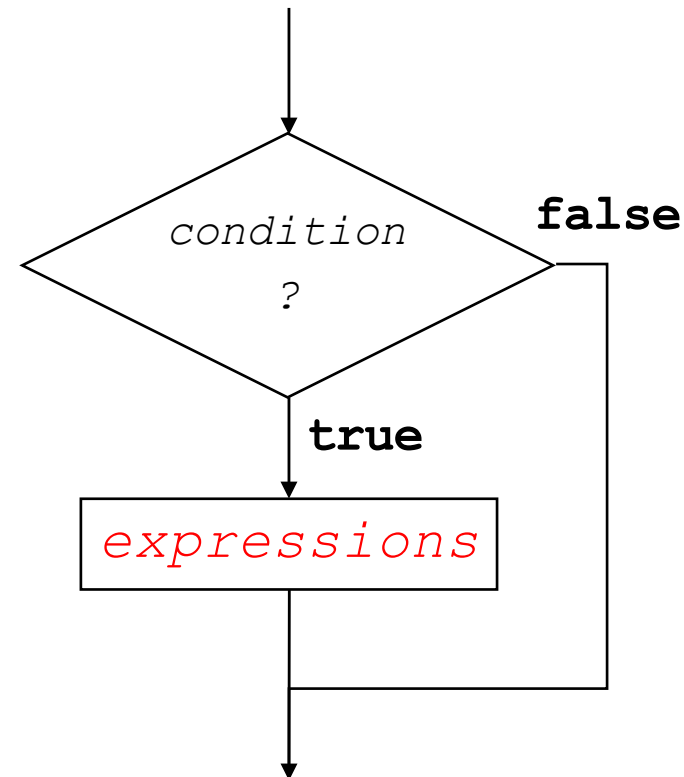
```
boolean javaEstFacile = true;
```

Décisions (1)



- Interruption de la progression linéaire du programme suite à une décision
 - Basée sur une condition qui peut être

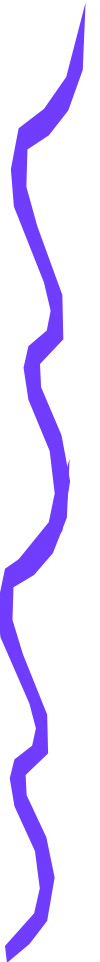
- **true**
- **false**



Décisions (2)



- Conditions basées sur des opérateurs relationnels
 - e.g. « égal à », « supérieur à »...
- Opérateurs relationnels combinées par opérations logiques
 - e.g. « et », « ou »...
- Traduction en code par des structures
 - **if**
 - **while**
 - **for**
 - **do**



Opérateurs Relationnels



Math	Java	Description
$>$	<code>></code>	Supérieur β
\geq	<code>>=</code>	Supérieur ou Égal β
$<$	<code><</code>	Inférieur β
\leq	<code><=</code>	Inférieur ou Égal β
$=$	<code>==</code>	Egalité
\neq	<code>!=</code>	Inégalité

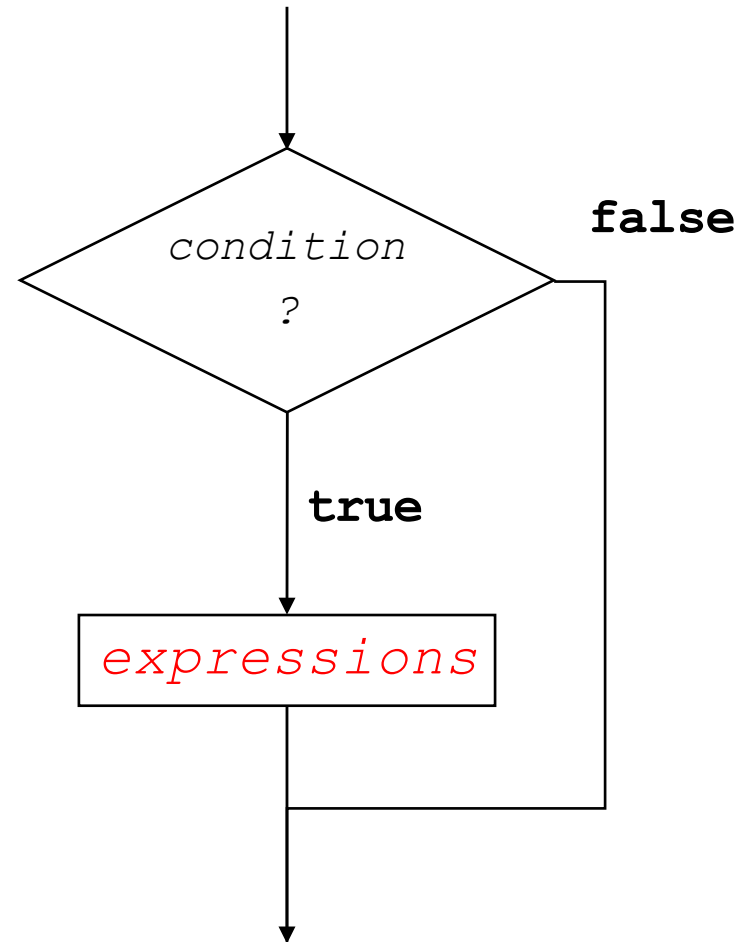
Décisions

if



```
if (condition) {  
    expressions  
}
```

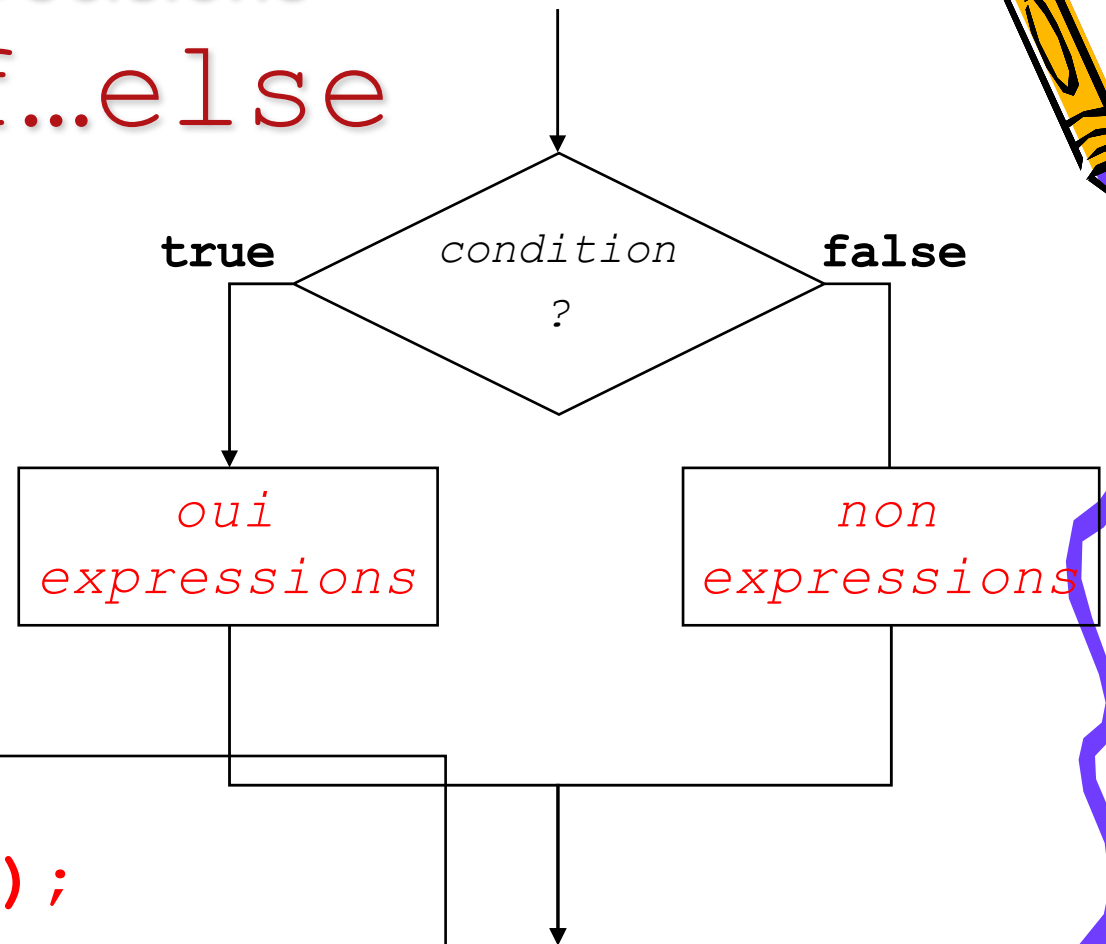
```
stopVoiture();  
if (piecesJaunes <= 0) {  
    baisseVitre();  
    System.out.println(  
        "Désolé, mec !");  
    leveVitre();  
}  
demarreEnTrombe();
```



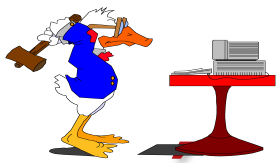
Décisions if...else



```
if (condition) {  
    oui expressions  
} else {  
    non expressions  
}
```



```
if (r >= 0) {  
    sqrt = Math.sqrt(r);  
} else {  
    System.err.println("Erreur");  
}
```



Décisions



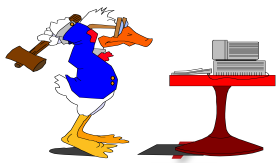
Égalité – le Piège du Débutant

- Lequel est le bon ?

```
if (vitesse = 160) {  
    points == -8;  
}
```

```
if (vitesse == 130) {  
    points = -6;  
}
```

```
if (vitesse = 110) {  
    points = -4;  
}
```



Décisions



Égalité – le Piège du Débutant

- Lequel est le bon ?

```
if (vitesse = 160) {  
    points == -8;  
}
```

```
if (vitesse == 130) {  
    points = -6;  
}
```

```
if (vitesse = 110) {  
    points = -4;  
}
```

- Evidemment...

```
if (vitesse == 130) {  
    points = -6;  
}
```

- **Ne pas confondre**
 - Affectation =
 - Égalité ==

Opérations Logiques



- Et : `&&`
- Ou : `||`
- Négation : `!`
- Ordre de priorité
 - `!` précède `&&` précède `||`
 - `a || !b && c` est équivalent à `a || ((!b) && c)`
 - `()` peuvent rendre l'expression plus claire

Décisions

& &



- Fonctionnalité

a	b	a && b
true	true	true
true	false	false
false	?	false

```
int vent = Console.in.readInt();
String type = "Ouragan";

if (vent < 64 && vent >= 56) {
    type = "Violente tempête";
}

if (vent < 56 && vent >= 48) {
    type = "Tempête";
}
```

Décisions

||

a	b	a b
true	?	true
false	true	true
false	false	false

- Fonctionnalité

```
int beaufort = Console.in.readInt();

if (beaufort < 0 || beaufort > 12) {
    System.err.println("Entrée erronée");
} else {
    fait quelquechose
}
```


Décisions

!



- Fonctionnalité

a	!a
true	false
false	true

```
int beaufort = Console.in.readInt();

if (!(beaufort > 9)) {
    System.out.println("Sortons");
} else {
    System.out.println("Au plumard");
}
```

Variables Logiques



- Type fondamental **boolean**

```
boolean beau = true;
boolean riche = true;

if (beau && riche) {
    System.out.println("Ça vaut mieux !");
}
```

- Valeurs possibles : **true false**



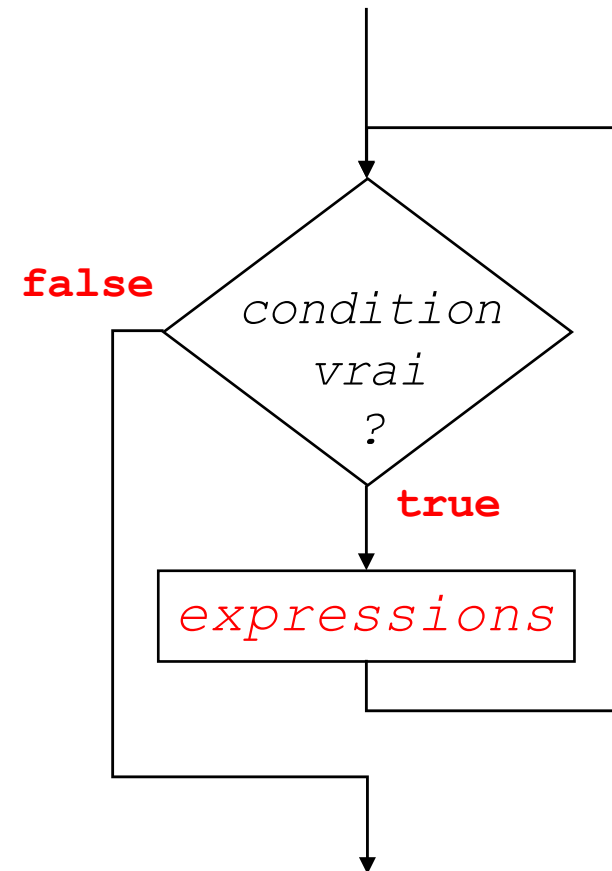
Les instructions de contrôles de Java

Itération while (1)



- Boucle tant que (*while*) une condition est vraie

```
while (condition) {  
    expressions  
}
```



Itération

while (2)



- La boucle la plus souvent rencontrée

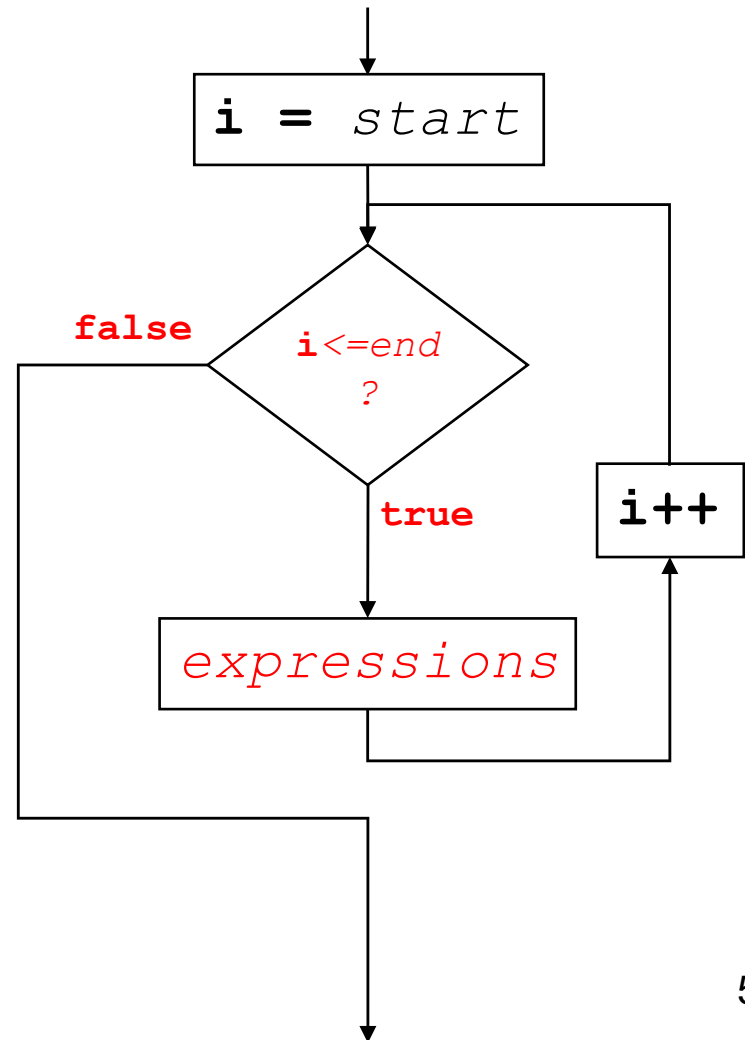
```
i = valInit;  
while (i <= valeurFin) {  
    expressions  
    i++;  
}
```

- Si souvent qu'elle s'écrit...

Itération for (1)



```
for (i = start; i <= end; i++)  
{  
  expressions  
}
```



Itération for (2)



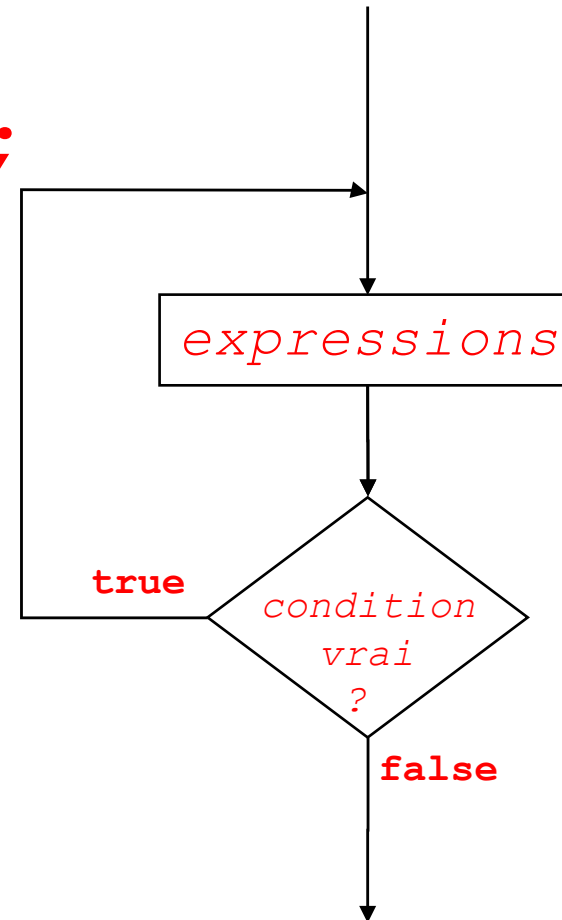
- Exemple - pour calculer $n! = 1 \times 2 \times 3 \times \dots \times n$

```
public int factoriel(int n) {  
    int facteur;  
    int produit = 1;  
    for (facteur = n; facteur > 0; facteur--)  
    {  
        produit = produit * facteur;  
    }  
    return produit;  
}
```

Itération do



```
do {  
    expressions  
} while (condition);
```



Itération

Conditions



- Conditions de bouclage
 - Compteur
 - `i < 100`
 - Sentinelle
 - `valeurEntree != 0`
 - Flag
 - `fait != true`
 - Borne
 - `montant < 0.5`

Itération

Compteur



- Passer par toutes les valeurs

```
for (int i = 1; i <= 100; i++) {  
    System.out.println(i);  
}
```

Itération

Sentinelles



- En attente d'une condition particulière

```
do {  
    int valeurEntree =  
    Console.in.readInt();  
    if (valeurEntree != -1) {  
        fait quelquechose  
    }  
} while (valeurEntree != -1)
```



Itération Flag



- Signale une condition

```
boolean fait = false;  
while (!fait) {  
    essaie de faire le nécessaire  
}
```

Itération Borne



- Limite à ne pas dépasser

```
double montant = soldeInitiale;  
while (montant > 0.5) {  
    dépenser (presque) sans compter  
}
```