PSEUDO-CODE ET LANGAGE JAVA : LES STRUCTURES DE CONTRÔLES : LA SÉQUENCE ET L'ALTERNATIVE

INTRODUCTION

Un programme n'est pas toujours qu'une suite d'instructions exécutées les unes à la suite des autres. Ils arrivent très fréquemment d'exécuter un traitement sous certaines **conditions**. Ou alors ils arrivent de devoir **répéter** plusieurs fois un même traitement.

On dispose en algorithmique et dans tous les langages de programmation de **structures de contrôles** que sont :

- **LA SÉQUENCE : suite d'opérations (affectation, lecture, écriture…)**
- **L'ALTERNATIVE : les tests**
- **L'ITÉRATION : les boucles**

À partir de ces 3 structures de contrôles, tous les programmes peuvent être écrits. Chaque structure peut s'imbriquer les unes dans les autres.

Nous allons aborder dans ce cours les notions algorithmiques sur les structures de contrôles, et sa traduction en langage Java.

LA SÉQUENCE

Définition : Suite d'action exécutée en **séquence** (les unes à la suite des autres). Le processeur exécute les instructions dans l'ordre dans lequel elles apparaissent dans le programme : l'exécution est **séquentielle**

Une fois que le programme a fini une instruction, il passe à la suivante. Tant qu'une instruction n'est pas terminée, il attend avant de continuer. Certaines instructions sont bloquantes. C'est le cas par exemple de la **saisie.** On attend que l'utilisateur saisisse une valeur au clavier avant de continuer.

Pseudo langage :	Comportement :
… action1 action2 action3 … (Séquence A)	

Exemple de séquence :

Algo	Langage Java

i ← 3
a ← 4
i ← i + a
Afflicter (a)
	int i=3;
	int a=4;
	i=i+a;
	System.out.println(a);

L'ALTERNATIVE OU CONDITIONNELLE

Tous les algorithmes ne s'exécutent pas en séquence. Parfois, il est nécessaire qu'un programme n'exécute pas toutes les instructions. C'est le rôle de la structure conditionnelle : exécuter une séquence sous certaines conditions. Les alternatives possibles sont le :

- SI…ALORS
- SI… ALORS …SINON
- SI…ALORS… SINON SI… SINON
- SUIVANT

C'EST QUOI UNE CONDITION ?

Une condition est une expression logique qui donne un résultat de type booléen. Elle est composée de trois éléments :

(Expr1) OP (Expr2)

Ou **Expr1 ou Expr2** peut être :

- Une valeur fixe ou littéral : 2 , 3.14 , "Chaine", 'c', true, false, etc …
- Une variable : a, b, age, poids, etc…
- Une expression : age -10 , a + b, etc …

Et **OP** un opérateur de comparaison :

En Algo	NON	ET	OU	=	<> ou ≠	<	≤	>	≥
En JAVA	!	&&	||	==	!=	<	<=	>	>=

Les conditions (ou expressions conditionnelles) peuvent aussi être complexes formés de plusieurs conditions simples ou variables booléennes reliées entre elles par les opérateurs logiques ET, OU, et NON. Pour plus de visibilité, il est conseillé d'utiliser les parenthèses.

Exemple : (a +3 = b ET c < 0) OU (a = c *2 ET b <> c)

Dans l'exemple les expressions booléennes soulignées en rouge sont évaluées en premier et donne soit un résultat vrai ou faux, puis celles soulignées en vert, et pour terminer en bleu. Pour évaluer correctement ce type d'expression il est indispensable de connaître ses tables de vérité du ET et du OU :

A	B	A **ET** B	A **OU** B
VRAI	VRAI	**VRAI**	**VRAI**
VRAI	FAUX	FAUX	**VRAI**
FAUX	VRAI	FAUX	**VRAI**
FAUX	FAUX	FAUX	FAUX

Le NON inverse le résultat d'une expression booléenne. Pour inverser l'expression précédente, on écrira :

NON((a +3 = b ET c < 0) OU (a = c *2 ET b <> c))

Cette expression peut être remplacé par la suivante où les ET sont remplacé par des OU, les OU par des ET, les < par >=, les = par des <>,…. Ce qui donne :

(a+3 <> b OU c >=0) ET (a <> c*2 OU b=c)

Exercice : pour chaque expression logique donner la valeur de la variable booléenne b :

```
int i=10,j=20,k=3,l=8,m=4,n=20 ;
boolean b ;

b=(i>j) ;               // l'expression logique (i>j) est évaluée. Le résultat est affecté à b
b=(j==(n-i)) ;          // une expression logique peut comporter des calculs (n-i)
b=(i<j) || (j==k) ;
b=(i<j)&&(k>l) ;
b= !((i<j)&&(k>l)) ;
b= (i!=j) ;
```

LA STRUCTURE SI… ALORS

Cette structure est utilisée si on veut exécuter une instruction seulement si une condition est vraie et ne rien faire si la condition est fausse.

Pseudo langage :	Comportement :	EN JAVA:

SI condition évalué à VRAI **ALORS**
action 1
action 2 — Séquence A
FIN SI
Séquence B
		if(condition évaluée à VRAI) **{**
		instruction1 ;
		instruction2 ;
		}
		ou
		if(condition VRAI)
		instruction ;

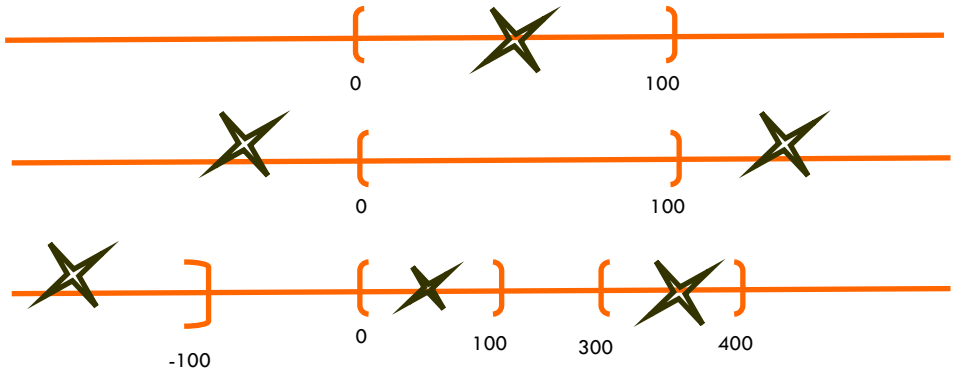
Remarque : dans la syntaxe du Java on note 2 syntaxes possibles. Lorsque la séquence A contient 2 instructions ou plus il est **obligatoire** de mettre les **{…}**, ce qui n'est pas le cas lorsqu'il y a une seule instruction. Les () autour du test sont obligatoires.

Conseil : lorsqu'on débute la programmation, pour ce simplifier le codage on mettra systématiquement les {…}.

EXERCICE

- Dans un programme de facturation, on veut effectuer une remise de 10 % si le montant de la facture dépasse 800 euros. Si le montant est inférieur à 800 euros, on veut juste afficher le montant tel quel.

  Si le montant est supérieur à 800 euros, il faut prendre en compte la remise et calculer le nouveau montant.
- Écrire un algorithme qui demande un nombre à l'utilisateur et l'informe ensuite si le nombre est pair. Dans le cas où le nombre est impair, aucun affichage n'est réalisé.
- Écrire un algorithme qui demande un nombre et informe ensuite l'utilisateur si le nombre est compris dans l'intervalle [2 – 20]. Dans le cas contraire, aucun affichage n'est effectué.
- Écrire un algorithme permettant de savoir si un nombre est dans les intervalles suivants :



SI… ALORS… SINON

Dans le déroulement d'un algorithme, on doit souvent choisir entre 2 actions, suivant une condition. C'est ce que permet le SI…ALORS…SINON. Dans ce cas au moins une séquence sera exécutée soit A, soit B. Le SINON est exécuté si la condition est évaluée à FAUX

Pseudo langage :	Comportement :	EN JAVA:
SI condition évalué à VRAI **ALORS** action 1 action 2 — Séquence A **SINON** action 3 action 4 — Séquence B **FIN SI** Séquence C		**if**(condition VRAI) **{** instruction1 ; instruction2 ; **} else {** Instruction3 ; Instruction4 ; **}**

Remarque :

- Le symbole { marque le début d'un bloc d'instruction, et } marque la fin du bloc. Utilisé avec un IF, le bloc est rattaché à ce dernier. Le deuxième bloque d'instruction est attaché au ELSE.
- La position des accolades en Java a peu d'importance. Il s'agit de conventions de codage. La convention choisie permet ici de gagner 3 lignes ce qui n'est pas négligeable tout en gardant une bonne lisibilité. Cette dernière notion est ici importante. Nous aurions pu aussi écrire les 2 codes suivants. Dans le 2ème cas le code est difficile à lire. On remarque également l'indentation des instructions par rapport au bloc d'instruction toujours par souci de lisibilité

```
if(condition VRAI)
{
      instruction1 ;
      instruction2 ;
}
else
{
      Instruction3 ;
      Instruction4 ;
}
```

```
if(condition VRAI){    instruction1 ;
      instruction2 ;}else{
Instruction3 ;
      Instruction4 ;
}
```

EXERCICE

1. Écrire un message précisant si la valeur d'une variable n saisit par l'utilisateur est positive ou négative
2. Dans un programme de facturation, on veut effectuer une remise de 10 % si le montant de la facture dépasse 800 euros. Dans tous les cas un message affichera s'il y a remise ou non.
3. On souhaite afficher la valeur absolue de la différence entre deux nombres entiers a et b.
4. Écrire un algorithme qui demande de saisir deux nombres à l'utilisateur ainsi qu'une lettre représentant l'opération à effectuer. Si la lettre est **s** ou **S** (comme somme), la somme doit être calculée et affichée. Si la lettre est un autre caractère, le produit doit être calculé et affiché.

SI…ALORS…SINON SI… SINON

Dans le cas précédent (SI… ALORS… SINON) 2 séquences peuvent être exécutées. Mais il y a des tas de situations où 2 voies ne suffisent pas.

Exemple : Un programme doit donner l'état de l'eau selon sa température. 3 affichages sont possibles : solide, liquide ou gazeuse.

Une première version de cet algorithme pourrait être :

```
VAR
       Temperature : Entier
DEBUT
   ECRIRE "Entrez la température de l'eau :"
   SASIR Temperature
   SI Temperature <= 0 ALORS
          ECRIRE "C'est de la glace"
   FINSI
   SI Temperature > 0 ET Temperature < 100 ALORS
           ECRIRE "C'est du liquide"
   FINSI
   SI Temperature >= 100  ALORS
          ECRIRE "C'est de la vapeur"
   FINSI
FIN
```

Il serait ainsi bien plus rationnel d'imbriquer les tests ainsi.

```
VAR
       Temperature : Entier
DEBUT
   ECRIRE "Entrez la température de
l'eau :"
   LIRE Temperature
   SI Temperature <= 0 ALORS
         ECRIRE "C'est de la glace"
   SINON
         SI Temperature < 100 ALORS
              ECRIRE "C'est du liquide"
         SINON
              ECRIRE "C'est de la vapeur"
         FINSI
   FINSI
FIN
```

```
VAR
       Temperature : Entier
DEBUT
   ECRIRE "Entrez la température de l'eau :"
   LIRE Temperature
   SI Temperature <= 0  ALORS
         ECRIRE "C'est de la glace"
   SINON SI Temperature < 100  ALORS
         ECRIRE "C'est du liquide"
   SINON
         ECRIRE "C'est de la vapeur"
   FINSI
FIN
```

Pseudo langage :	Comportement :	EN JAVA:

SI condition évalué à VRAI **ALORS**
action 1
action 2 — Séquence A
 	
SINON SI condition VRAI **ALORS**
action 3
action 4 — Séquence B
SINON SI
SINON SI
...
SINON — Séquence N
FIN SI
Séquence C
		if(condition VRAI) **{**
		instruction1 ;
		instruction2 ;
		}
		else if (condition VRAI) **{**
		Instruction3 ;
		Instruction4 ;
		}
		else if (condition VRAI){
		Instruction5 ;
		Instruction6 ;
		}
		else{
		
		}

Toutes ses structures alternatives peuvent s'imbriquer les unes dans les autres. Les possibilités sont infinies.

Exemple :

```
si… alors
        si… alors
                …
        fin si
sinon si… alors
        si… alors
        …
        sinon
                si… alors
                        …
                fin si
        fin si
fin si
```

EXERCICE

Modifier l'algorithme qui demande de saisir deux nombres à l'utilisateur ainsi qu'une lettre représentant l'opération à effectuer suivant le cahier des charges suivant : si la lettre est s ou S (comme somme), la somme doit être calculée et affichée. Si la lettre est un p ou P, le produit doit être calculé et affiché. Dans tout autre cas, afficher un message d'erreur.

REMARQUES

Remarque 1 : Il faut structurer le programme de façon à le rendre plus lisible. Il ne faut pas hésiter à utiliser les accolades et les indentations pour plus de clarté.

Exemple ce qu'il ne faut pas faire : comme il n'y a pas d'accolade ce programme est faux. Corrigez-le.

```
// vérification que le nombre rentré est inférieur à 10 ou supérieur à 100
// et affichage d'un message si OK
int x ;
System.out.println("Entrer un nombre inférieur à 10 ou supérieur à 100 : ») ;
x=sc.nextInt() ;
if(x > 10)
        if(x>100)
                alert("Supérieur à 100 \n merci ") ;
        else
                alert("inférieur à 10, \n merci ") ;
```

Remarque 2 : Il ne faut pas confondre l'opérateur (=) d'affectation avec l'opérateur (==) de comparaison.

Exemple :

```
int i = 10 ;
if (i = 2)      // Ici = est l'opérateur d'affectation ; on affecte à i la valeur 2
                // Le programme ne compile pas. La raison est simple : ce qui est
                // attendu entre parenthèse est un boolean, or (a) est un int : Erreur
                // ATTENTION : d'autres langages ne font pas cette vérification (C,
                // JS) et passe à la compilation, mais provoque plus tard une erreur
                // à l'exécution.
{   ... }
```

Il faut donc écrire :

```
if (i == 2)   // == est l'opérateur d'égalité. Le résultat est une valeur booléenne (ici FAUX)
              // puisque i vaut 10
```

LE CHOIX MULTIPLE : LE SUIVANT

La structure SUIVANT permet de choisir le traitement à effectuer en fonction de la valeur ou de l'intervalle de valeur d'une variable ou d'une expression. Cette structure permet de remplacer avantageusement une succession de structures SI…ALORS.

Dans ce cas il n'y a pas d'évaluation booléenne.

Suivant la valeur de la variable le cas correspondant est exécuté. Les autres cas sont ignorés. On parle d'exécution conditionnelle d'un cas parmi N possibles.

Pseudo langage :	Comportement :	En JAVA :

Suivant variable **faire**
  cas 1 :
  action1
  action2 — Séquence A
  cas 2 :
  action3
  action4 — Séquence B
  ...
  autre :
  action5 — Séquence N
finSuivant
		switch (variable)
		{
		  case valeur1 :
		  instructions1 ;
		  instructions2 ;
		  break ;
		  case valeur2 :
		  instruction3 ;
		  break ;
		  default :
		  instructionN ;
		  break ;
		}

Il est possible d'imbriquer plusieurs cas qui effectuent le même traitement

```
Cas3 à cas 4 :
        action1
        action2

Ou bien

Cas 1, Cas 2, Cas 3 :
        action1
        action2
```

```
switch (variable)
{
    case 3:case 4 :
        instructions1 ;
        instructions2 ;
        break ;
    case 2 :
        instruction3 ;
        break ;
   default :
        instructionN ;
        break ;
}
```

Règles à respecter concernant le switch

- Les types de variables autorisés dans le switch sont : byte, Byte, short, Short, int, Integer, char, Character, String(depuis la version 7 de Java) . Les autres types de données (float, double, long) ne sont pas acceptés.
- Ce qui est mis dans le « case » doit être compatible avec le type de donnée choisie dans le switch

```
case "2": erreur si dans le switch la variable est un int
```

- La valeur d'un « case » doit être une constante ou un littéral. Les variables ne sont pas autorisées.

```
case b+1: erreur si b n'est pas une constante (déclaré avec final)
```

- L'instruction *default* est facultative et peut apparaître n'importe où dans le bloc switch, même si la convention est de le faire apparaitre en dernier. Dans ce cas le break est inutile.
- L'instruction break est importante. Si elle est omise, cela ne génère pas d'erreur de compilation, mais attention, l'exécution se poursuit avec le « case » suivant ce qui peut être un comportement attendu.

EXERCICE

- Écrire un algorithme permettant d'afficher en toutes lettres le mois en fonction de son numéro. Par exemple si mois vaut 3 on affichera "Mars". Coder ce programme en JAVA.
- Ecrire un algorithme permettant d'effectuer une opération arithmétique. L'utilisateur choisit un nombre, un opérateur, un nombre. En fonction de l'opérateur, l'opération adéquate est exécutée.
- Reprendre l'exercice des mois. Cette fois-ci si l'utilisateur saisi 3 on affichera tous les mois restant jusqu'à la fin de l'année.
- Écrire un algorithme permettant d'effectuer une opération sur une variable selon le menu suivant :

  Choix 1 : ajouter 5

  Choix 2 : ajouter 3 et multiplier par 5

  Choix 3 : soustraire 2 et diviser par 4