



Traduction du pseudo-code utilisé en logique de programmation vers Java

➤ Les commentaires


Pseudo-code	
// commentaire sur une seule ligne	// commentaire sur une seule ligne
	/* un long commentaire sur plusieurs lignes */

➤ Les variables


Déclarer une variable (sans initialisation)

Pseudo-code	
VAR nomVariable : typeVariable	typeVariable nomVariable ;
<ul style="list-style-type: none">○ N : type Numérique○ T : type Texte○ C : type Caractère○ B : type Booléen	<ul style="list-style-type: none">○ byte : type entier sur 1 byte○ short : type entier court○ int : type entier○ long : type entier long○ float : type nombre décimal sur 32 bits○ double : type nombre décimal sur 64 bits○ String : type chaîne de caractère (texte) NB : le « S » majuscule de « String »○ char : type caractère○ boolean : type booléen

Déclarer une variable (avec initialisation)

Pseudo-code	
VAR nomVariable : typeVariable ← valeur	typeVariable nomVariable = valeur;
Exemples	
VAR nombreEntier : N ← 3	int nombreEntier = 3 ;
VAR nombreDecimal : N ← 15,2	float nombreDecimal = 15.2 ; NB : le « . » séparant la partie entière et la partie décimale du nombre
VAR texte : T ← "Bonjour"	String texte = "Bonjour" ;

Affecter une valeur à une variable (préalablement déclarée)


Pseudo-code	
nomVariable ← valeur	nomVariable = valeur;
Exemples	
nombreEntier ← 5	nombreEntier = 5 ;
nombreDecimal ← 8,3	nombreDecimal = 8.3 ;
texte ← "Au revoir !"	texte = "Au revoir !" ;

Remarques :


- Si une variable n'est pas préalablement déclarée avant une affectation, une erreur se produit lors de la compilation.
- Il faut que la valeur affectée soit du même type que celle utilisée lors de la déclaration

Affichage et saisie


Affichage

Pseudo-code	
AFFICHER ...	<pre>System.out.println(...);</pre> <ul style="list-style-type: none"> NB : un retour à la ligne est automatiquement ajouté après ce que l'on a affiché. L'affichage suivant débutera donc sur une nouvelle ligne.
AFFICHER ...	<pre>System.out.print(...);</pre> <ul style="list-style-type: none"> NB : un retour à la ligne N'est PAS automatiquement ajouté après ce que l'on a affiché. Par défaut, l'affichage suivant NE débutera donc PAS sur une nouvelle ligne.
Symbole de concaténation : ,	<p>Symbole de concaténation : +</p> <ul style="list-style-type: none"> Remarque : « + » est aussi le symbole de l'addition quand les 2 opérandes sont des nombres.
Exemples	
AFFICHER "Bonjour"	<pre>System.out.println("Bonjour");</pre> <p>// affiche à l'écran : Bonjour</p>
AFFICHER texte	<pre>System.out.println(texte);</pre> <p>// affiche à l'écran le contenu de la variable texte</p>
AFFICHER 3 + 2	<pre>System.out.println(3+2);</pre> <p>// affiche à l'écran : 5</p>
AFFICHER "3 + 2 =", 3 + 2	<pre>System.out.println("3 + 2 =" + (3 + 2));</pre> <p>// affiche à l'écran : 3 + 2 = 5</p> <p>NB : les parenthèses pour que l'addition soit effectuée avant la concaténation</p>

Saisie


Pseudo-code	
	<pre>import java.util.Scanner;</pre> <p>// ligne à inclure au début du fichier avant la ligne commençant par le mot-clé « class »</p>
SAISIR monCaractere	<pre>monCaractere = new Scanner(System.in).next();</pre> <ul style="list-style-type: none"> NB : monCaractere est une variable qui aura été préalablement déclarée avec le type : char
SAISIR maChaine	<pre>maChaine = new Scanner(System.in).nextLine();</pre> <ul style="list-style-type: none"> NB : maChaine est une variable qui aura été préalablement déclarée avec le type : String
SAISIR nombreEntier	<pre>nombreEntier = new Scanner(System.in).nextByte();</pre> <ul style="list-style-type: none"> NB : nombreEntier est une variable qui aura été préalablement déclarée avec le type : byte <pre>nombreEntier = new Scanner(System.in).nextShort();</pre> <ul style="list-style-type: none"> NB : nombreEntier est une variable qui aura été préalablement déclarée avec le type : short <pre>nombreEntier = new Scanner(System.in).nextInt();</pre> <ul style="list-style-type: none"> NB : nombreEntier est une variable qui aura été préalablement déclarée avec le type : int <pre>nombreEntier = new Scanner(System.in).nextLong();</pre> <ul style="list-style-type: none"> NB : nombreEntier est une variable qui aura été préalablement déclarée avec le type : long
SAISIR nombreAVirgule	<pre>nombreAVirgule = new Scanner(System.in).nextFloat();</pre> <ul style="list-style-type: none"> NB : nombreAVirgule est une variable qui aura été préalablement déclarée avec le type : float <pre>nombreAVirgule = new Scanner(System.in).nextDouble();</pre> <ul style="list-style-type: none"> NB : nombreAVirgule est une variable qui aura été préalablement déclarée avec le type : double
SAISIR monBooleen	<pre>monBooleen = new Scanner(System.in).nextBoolean();</pre> <ul style="list-style-type: none"> NB : nombreBooleen est une variable qui aura été préalablement déclarée avec le type : boolean

➤ **Types de données booléennes**

Pseudo-code	
VRAI FAUX	true false
NON	not
ET	& <ul style="list-style-type: none"> NB : les 2 opérandes de & sont d'abord évalués. Ensuite une conclusion est tirée.
ET	&& <ul style="list-style-type: none"> NB : le second opérande de && ne sera évalué que si le premier est vrai (si le premier opérande est faux, il est inutile de perdre du temps à évaluer le second puisque le résultat sera faux quelle que soit la valeur du second opérande)
OU	 <ul style="list-style-type: none"> NB : les 2 opérandes de sont d'abord évalués. Ensuite une conclusion est tirée.
OU	 <ul style="list-style-type: none"> NB : le second opérande de ne sera évaluée que si le premier est faux (si le première opérande est vrai, il est inutile de perdre du temps à évaluer le second puisque le résultat sera vrai quelle que soit la valeur du second opérande)
=	== <ul style="list-style-type: none"> NB : 2 X le symbole « = » pour la comparaison. Pour rappel le 1X « = » correspond à l'affectation
≠	!=
<	<
<=	<=
>	>
>=	>=

L'alternative


SI (if)

Pseudo-code	
SI expressionBoolenne ALORS // instructions à exécuter si la // condition est vraie FINSI	if expressionBoolenne { // instructions à exécuter si la // condition est vraie }
SI expressionBoolenne ALORS // instructions à exécuter si la // condition est vraie SINON // instructions à exécuter si la // condition est fausse FINSI	if expressionBoolenne { // instructions à exécuter si la // condition est vraie } else { // instructions à exécuter si la // condition est fausse }

Remarque :


Les boucles imbriquées sont bien entendu aussi possible en Java
 (pour effectuer un « SINON SI », on écrit « else if »)

CAS OU (switch)


Pseudo-code	
CAS OU varNumerique CAS valeur1 // instructions à exécuter si // varNumerique=valeur1 CAS valeur2 // instructions à exécuter si // varNumerique=valeur2 AUTRES CAS // instructions à exécuter si // varNumerique est autre FIN CAS OU	swicth varNumerique{ case valeur1 : // instructions à exécuter si // varNumerique=valeur1 break ; case valeur2 : // instructions à exécuter si // varNumerique=valeur2 break ; default : // instructions à exécuter si // varNumerique est autre }

➤ **Les boucles**


TANT QUE (while)

Pseudo-code	
TANT QUE expressionBoolenne // instructions à répéter tant que la // condition est vraie FIN TANT QUE	while expressionBoolenne { // instructions à répéter tant que la // condition est vraie }

do ... while


Pseudo-code	
	do { // instructions à répéter tant que la // condition est vraie }while expressionBoolenne ; NB : ne pas oublier « ; » après l'expression booléenne

REPETER ... JUSQU'À CE QUE


Pseudo-code	
REPETER // instructions à répéter jusqu'à ce // que la condition soit vraie // (tant que la condition est fausse JUSQU'À CE QUE expressionBoolenne	

Astuce : on peut aisément transformer une boucle **REPETER ... JUSQU'À CE QUE** écrite en pseudo-code en une boucle **do ... while** écrite en Java en inversant la valeur de l'expression booléenne avec un « not ».

POUR (for)


Pseudo-code	
POUR A ALLANT DE B A C PAR PAS DE D // instructions à répéter un nombre // connu de X FIN TANT QUE	for (A=B ; A <= C ; A = A + D){ // instructions à répéter un nombre // connu de X }

➤ **Fonctions sur le type texte**


Pseudo-code	
<pre>// supposons que la variable « texte » a // été déclarée et initialisée longueur(texte) caract(texte, position) sousChaine(texte, début, fin)</pre>	<pre>// supposons que la variable « texte » a // été déclarée et initialisée texte.length(); texte.charAt(position -1); Remarque : en Java, contrairement au pseudo-code, le premier caractère est en 0 (d'où le « -1 » pour généraliser) texte.substring(début -1, fin -1); Même remarque</pre>

➤ **Les tableaux**

Déclaration de tableaux 1D (ou vecteurs)

Pseudo-code	
VAR identifiant : type [exprInt] NB : type sont les types de variables déjà vus dans la déclaration d'une variable. NB2 : exprInt est une expression dont le résultat est un entier.	type [] identifiant = new type [exprInt] ; NB : type sont les types de variables déjà vus dans la déclaration d'une variable. NB2 : exprInt est une expression dont le résultat est un entier.
Exemples	
VAR tableau : N [6] VAR liste : T [5]	int [] tableau = new int [6] ; String [] liste = new String [5] ;


Déclaration de tableaux 2D (ou matrices)

Pseudo-code	
VAR identifiant : type [exprInt, exprInt] NB : type sont les types de variables déjà vus dans la déclaration d'une variable. NB2 : exprInt est une expression dont le résultat est un entier.	type [][] identifiant = new type [exprInt][exprInt]; NB : type sont les types de variables déjà vus dans la déclaration d'une variable. NB2 : exprInt est une expression dont le résultat est un entier.
Exemple	
VAR tableau : N [6,5]	int [][] tableau = new int [6] [5] ;

Remarque :

Il est possible de créer des tableaux à 3 dimensions, 4 dimensions et plus.

Affichage de la valeur d'une case d'un tableau

Pseudo-code	
Exemples	
AFFICHER tableau[3]	System.out.println(tableau [2] ;
AFFICHER tableau[4,2]	System.out.println(tableau [3][1]) ;

Remarques :

- Dans le pseudo code, l'indice de la première case est 1, en Java, l'indice de la première case est 0.
- Boucler sur un tableau de 6 cases fera parcourir les indices de 0 à 5 inclus.
- **System.out.println(tableau [2])** affichera la 3ème case d'un tableau, celle qui a l'indice 2.
- **tab.length()** permet de donner la longueur d'un tableau en Java.
- Il est possible en Java de déclarer et de remplir un tableau en même temps :
int tableau = {6,8,47,31} ;
 crée un tableau de 4 cases (indices 0 à 3) déjà rempli de 4 nombres entiers.

