**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   * + N : type Numérique   + T : type Texte   + C : type Caractère   + B : type Booléen | typeVariable nomVariable ;   * + 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 chaine 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  VAR nombreDecimal : N 🡨 15,2  VAR texte : T 🡨 ″Bonjour″ | int nombreEntier = 3 ;  float nombreDecimal = 15.2 ; NB : le « . » séparant la partie entière et la partie décimale du nombre  String texte = ″Bonjour″ ; |

Affecter une valeur à une variable (préalablement déclarée)

|  |  |
| --- | --- |
| Pseudo-code |  |
| nomVariable 🡨 valeur | nomVariable = valeur; |
| Exemples | |
| nombreEntier 🡨 5  nombreDecimal 🡨 8,3  texte 🡨 ″Au revoir !″ | nombreEntier = 5 ;  nombreDecimal = 8.3 ;  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  …  AFFICHER  …  Symbole de concaténation :  , | System.out.println(…) ;   * 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.   System.out.print(…) ;   * 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 : +   * Remarque : « + » est aussi le symbole de l’addition quand les 2 opérandes sont des nombres. |
| Exemples | |
| AFFICHER  ″Bonjour″  AFFICHER  texte  AFFICHER  3 +2  AFFICHER  ″3 + 2 =″, 3 + 2 | System.out.println(″Bonjour″) ;  // affiche à l’écran : Bonjour  System.out.println(texte) ;  // affiche à l’écran le contenu de la variable texte  System.out.println(3+2) ;  // affiche à l’écran : 5  System.out.println(″3 + 2 = ″+ (3 + 2)) ;  // affiche à l’écran : 3 + 2 = 5  NB : les parenthèses pour que l’addition soit effectuée avant la concaténation |

Saisie

|  |  |
| --- | --- |
| Pseudo-code |  |
|  | import java.util.Scanner;  // ligne à inclure au début du fichier avant la ligne commençant par le mot-clé « class » |
| SAISIR  monCaractere  SAISIR maChaine  SAISIR nombreEntier  SAISIR nombreAVirgule  SAISIR monBooleen | monCaractere = new Scanner(System.in).next();   * NB : monCaractere est une variable qui aura été préalablement déclarée avec le type : char   maChaine = new Scanner(System.in).nextLine();   * NB : maChaine est une variable qui aura été préalablement déclarée avec le type : String   nombreEntier = new Scanner(System.in).nextByte();   * NB : nombreEntier est une variable qui aura été préalablement déclarée avec le type : byte   nombreEntier = new Scanner(System.in).nextShort();   * NB : nombreEntier est une variable qui aura été préalablement déclarée avec le type : short   nombreEntier = new Scanner(System.in).nextInt();   * NB : nombreEntier est une variable qui aura été préalablement déclarée avec le type : int   nombreEntier = new Scanner(System.in).nextLong();   * NB : nombreEntier est une variable qui aura été préalablement déclarée avec le type : long   nombreAVirgule = new Scanner(System.in).nextFloat();   * NB : nombreAVirgule est une variable qui aura été préalablement déclarée avec le type : float   nombreAVirgule = new Scanner(System.in).nextDouble();   * NB : nombreAVirgule est une variable qui aura été préalablement déclarée avec le type : double   monBooleen = new Scanner(System.in).nextBoolean();   * NB : nombreBooleen est une variable qui aura été préalablement déclarée avec le type : boolean |

* ***Types de données booléennes***

|  |  |
| --- | --- |
| Pseudo-code |  |
| VRAI  FAUX  NON  ET  ET  OU  OU  =  ≠  <  <=  >  >= | true  false  not  &   * NB : les 2 opérandes de & sont d’abord évalués. Ensuite une conclusion est tirée.     &&   * 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)   |   * NB : les 2 opérandes de | sont d’abord évalués. Ensuite une conclusion est tirée.   ||   * NB : le second opérande de || ne sera évaluée que si l 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)   ==   * 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  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  }  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 |  |
| // supposons que la variable « texte » a  // été déclarée et initialisée  longueur(texte)  caract(texte, position)  sousChaine(texte, début, fin) | // 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 |

* ***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] ;s  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]  AFFICHER tableau[4,2] | System.out.println(tableau [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.

* ***/***
* ***/***
* ***/***