



TD Séquentiel - Rappels de base

Résumé

Ce TD a pour but de

1 Algorithmes séquentiels	3
1.1 Le pseudo-code	3
2 Variables et types	3
2.1 Les types autorisés en algo	3
2.2 Les types équivalents en java	4
2.3 Java est un langage fortement typé.	4
2.4 Déclaration de variables en algo	7
2.5 Déclaration de variables en Java	9
3 Opérateurs et expressions	10
3.1 Les opérateurs arithmétiques élémentaires en algo	10
3.2 Les opérateurs arithmétiques élémentaires en Java	11
3.3 Les opérateurs DIV et MOD en algo	11
3.4 Les opérateurs DIV et MOD en Java	11
3.5 Les fonctions mathématiques complexes en algo	11
3.6 Les fonctions mathématiques complexes en Java	12
4 L'affectation d'une valeur à une variable	12
4.1 Affectation interne en algo	12
4.2 Affectation interne en Java	13
4.3 Affectation externe en algo	13



4.4	Affectation externe en Java	14
4.5	Communication des résultats en algo	15
4.6	Communication des résultats en Java	15

1 Algorithmes séquentiels

Revoyons ici les bases du pseudo-code et leur traduction en Java.

1.1 Le pseudo-code

Le pseudo-code ou Langage de Description des Algorithmes (LDA en abrégé) est un langage formel et symbolique utilisant :

- des noms symboliques destinés à représenter les objets sur lesquels s'effectuent des actions ;
- des opérateurs symboliques ou des mots-clés traduisant les opérations primitives exécutables par un exécutant donné ;
- des structures de contrôle types.

2 Variables et types

Nous savons que les opérations que l'ordinateur devra exécuter portent sur des éléments qui sont les **données** du problème.

Lorsqu'on attribue un **nom** et un **type** à ces données, on parle alors de **variables**.

Dans un algorithme, une variable conserve toujours son nom et son type, mais peut changer de **valeur**.

- Le **nom** d'une variable permet de la caractériser et de la reconnaître ;
- le **type** d'une variable décrit la nature de son contenu.

2.1 Les types autorisés en algo

Dans un premier temps, les seuls types utilisés sont :

- **entier** pour les nombres entiers ;
- **réel** pour les nombres réels ;
- **caractère** pour les différentes lettres et caractères (par exemple ceux qui apparaissent sur un clavier : 'a', '1', '#', etc.)
- **chaîne** pour les variables contenant un ou plusieurs caractère(s) ou aucun (la chaîne vide) (par exemple : "Bonjour", "Bonjour le monde", "a", "", etc.)
- **booléen** les variables de ce type ne peuvent valoir que **vrai** ou **faux**

Le type des données

Quel(s) type(s) de données utiliseriez-vous pour représenter :

- une date du calendrier ? _____
- un moment dans la journée ? _____
- le prix d'un produit en grande surface ? _____
- votre nom ? _____
- vos initiales ? _____
- votre adresse ? _____

2.2 Les types équivalents en java

Les équivalents Java des types donnés en algo sont

- `int` pour le type **entier** pour les nombres entiers ;
- `double` pour le type **réel** pour les nombres réels ;
- `char` pour le type **caractère** pour les différentes lettres et caractères
- `String` pour le type **chaîne** pour les variables contenant un ou plusieurs caractère(s) ou aucun (la chaîne vide)
- `boolean` pour le type **booléen** les variables de ce type ne peuvent valoir, en Java, que `true` ou `false`

Les commandes de base

Quel(s) type(s) de données utiliseriez-vous pour représenter :

- une date du calendrier ? _____
- un moment dans la journée ? _____
- le prix d'un produit en grande surface ? _____
- votre nom ? _____
- vos initiales ? _____
- votre adresse ? _____

2.3 Java est un langage fortement typé.

Java est un langage fortement typé.

Toute donnée a un type. Quels types ?

- **primitifs prédéfinis** : entier, réel, booléen (logique) ;
- **références prédéfinis** : tableaux, `String`, . . . ;
- **références définis par le programmeur**

Les types primitifs

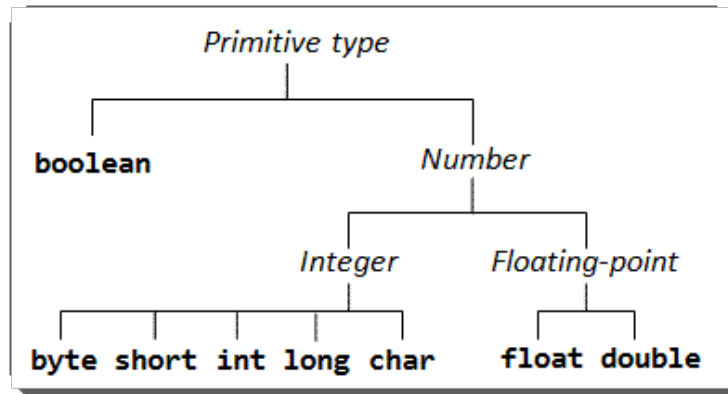


FIGURE 1 – primitifs.png

Les types primitifs numériques entiers

Les types primitifs numériques entiers sont **byte**, **short**, **int** et **long** (**char** sera vu à part),

- ce sont des nombres signés (en complément à 2)
- ils sont codés sur
 - 8 bits pour un **byte** ;
 - 16 bits pour un **short** ;
 - 32 bits pour un **int** ;
 - 64 bits pour un **long** ;
- ils comprennent donc les valeurs :
 - -128 à 127 pour un **byte** ;
 - -32768 à 32767 pour un **short** ;
 - -2147483648 à 2147483647 pour un **int** ;
 - -9223372036854775808 à 9223372036854775807 pour un **long** ;

Les littéraux pour les types primitifs numériques entiers

Un **littéral**, c'est une notation pour représenter une valeur fixée directement dans un code source.

Un **littéral numérique décimal** est

- une suite de chiffres
- éventuellement séparés par `_` pour la lisibilité
- suivi éventuellement du suffixe (`L` ou `l`) qui distingue un **int** d'un **long**
- pas de **byte** ou **short**, un littéral est **int** (ou **long**)

Un **littéral numérique octal** est

- une suite de chiffres de 0 à 7
- précédé de 0.

Un **littéral numérique hexadécimal** est

- une suite de chiffres et lettres a,b,c,d,e,f (minuscules/majuscules)
- précédé de 0x ou 0X

Un **littéral numérique binaire** est

- une suite de chiffres 0 et 1
- précédé de 0b ou 0B

Par exemple, la quantité 100 de type `int` peut s'écrire

- 100
- 1_0_0
- 0144
- 01_44
- 0x64
- 0b110_0100

Le type numérique caractère

`char` est

- un caractère Unicode codé en UTF16
- un entier non signé sur 16 bits
- assimilé à un entier

Pour un littéral de type `char` : un caractère entre single quote. Par exemple : `'a'`, `'1'`, `' '`, ...

Il existe des séquences d'échappement pour représenter certains caractères spéciaux :

- `\n` pour représenter un **saut de ligne**
- `\t` pour représenter une **tabulation**
- `\'` pour représenter le caractère `'`
- `\"` pour représenter le caractère `"`

Les types primitifs numériques flottants

Les types primitifs numériques flottants sont `float` et `double`.

- ils respectent la norme IEEE754 ;
- ils sont codés sur
 - 32 bits pour un `float` ;
 - 64 bits pour un `double` ;
- on utilisera plus souvent le type `double` ;
- il s'agit d'une **modélisation** de la notion mathématique.

Les littéraux pour les types primitifs numériques flottants

Un **littéral**, c'est une notation pour représenter une valeur fixée directement dans un code source.

Un **littéral numérique flottant** se compose de parties optionnelles (mais pas ensemble) :

1. une partie entière
2. .
3. partie décimale
4. E ou e
5. exposant
6. suffixe (en l'absence de suffixe : un double)

Par exemple : 1.2E3, 1.F, .1, 1e-2d, 1f

Par contre : 1, .E1, E1 ne sont pas des littéraux pour les types primitifs numériques flottants

Le type primitif booléen

`boolean`

- aussi appelé logique ;
- il peut prendre 2 valeurs :
 - `true` ;
 - `false` ;
- il prend 1 octet en mémoire.

Les types références prédéfinis : les chaînes de caractères

`String`

- il s'agit de caractères entre double quote : par exemple : "Java", "Bonjour tout le monde", "a",... ;
- est un type référence prédéfini ;
- Attention `char` \neq `String`

2.4 Déclaration de variables en algo

La **déclaration** d'une variable est l'instruction qui définit son **nom** et son **type**.

`num1, num2 : entiers`

L'ensemble des instructions de la forme `variable1, variable2, . . . : type` forme la partie d'un algorithme nommée **déclaration des variables**.

La déclaration des informations apparaîtra toujours en **début** d'algorithme, ou dans un bloc annexe appelé dictionnaire des variables ou encore dictionnaire des données.

Valeur initiale ?

Attention, lors de la déclaration d'une variable, **celle-ci n'a pas de valeur !** Nous verrons que c'est l'instruction d'affectation qui va servir à donner un contenu aux variables déclarées.

Comment nommer correctement une variable ?

Le but est de trouver un nom qui soit suffisamment **court**, tout en restant **explicite** et ne prêtant **pas à confusion**.

Ainsi *num1* est plus approprié pour désigner le premier numérateur que *zozo1*, *tintin*, *bidule* ou *premierNumérateur*. De même, ne pas appeler *den* la variable représentant le numérateur.

Il faut aussi tenir compte que les langages de programmation imposent certaines limitations (parfois différentes d'un langage à l'autre) ce qui peut nécessiter une modification du nom lors de la traduction.

Voici quelques règles et limitations traditionnelles dans les langages de programmation :

- Un nom de variable est généralement une suite de caractères alphanumériques d'un seul tenant (pas de caractères blancs) et ne commençant jamais par un chiffre. Ainsi *x1* est correct mais pas *1x*.
- Pour donner un nom composé à une variable, on peut utiliser le «tiret bas » ou underscore (par ex. *premier_numérateur*) mais on déconseille d'utiliser le signe «- » qui est plutôt réservé à la soustraction. Ainsi, dans la plupart des langages, *premier-numérateur* serait interprété comme la soustraction des variables *premier* et *numérateur*.
- Une alternative à l'utilisation du tiret bas pour l'écriture de noms de variables composés est la notation «chameau » (*camelCase* en anglais), qui consiste à mettre une majuscule au début des mots (généralement à partir du deuxième), par exemple *premierNombre* ou *dateNaissance*.
- Les indices et exposants sont proscrits.
- Les mots-clés du langage sont interdits (par exemple *for*, *if*, *while* pour Java et Cobol) et on déconseille d'utiliser les mots-clés du pseudo-code (tels que *Lire*, *Afficher*, *pour*. . .)

- Certains langages n'autorisent pas les caractères accentués (tels que à, ç, ê, ø, etc.) ou les lettres des alphabets non latins mais d'autres oui ; certains font la distinction entre les minuscules et majuscules, d'autres non. En algorithmique, nous admettons, dans les noms de variables les caractères accentués du français, par ex. : durée, intérêts, etc.

Comment déclarer

Quelle instruction permet de déclarer :

- le jour d'une date ? _____
- l'heure d'un moment ? _____
- le prix d'un produit en grande surface ? _____
- votre nom ? _____

2.5 Déclaration de variables en Java

La **déclaration** d'une variable est l'instruction qui définit son **nom** et son **type**.

```
int num1;
```

L'ensemble des instructions de la forme **type nom**; (où le **type** peut être un type primitif, un type référence prédéfini ou un type référence défini par l'utilisateur) forme la **déclaration des variables**.

La déclaration des informations apparaîtra en **début** de bloc.

Valeur initiale ?

Attention, comme en algo, lors de la déclaration d'une variable, **celle-ci n'a pas de valeur** ! Nous verrons que c'est l'instruction d'affectation qui va servir à donner un contenu aux variables déclarées.

Comment nommer correctement une variable ?

Le but est de trouver un nom qui soit suffisamment **court**, tout en restant **explicite** et ne prêtant **pas à confusion**.

Il faut aussi tenir compte des limitations du langage.

- Un nom de variable est généralement une suite de caractères alphanumériques d'un seul tenant (pas de caractères blancs) et ne commençant jamais par un chiffre. Ainsi **x1** est correct mais pas **1x**.
- un nom de variable ne peut comporter que des lettres, des chiffres, les caractères **_** et **\$**.

- L'écriture de noms de variables composés est la notation «chameau» (camelCase en anglais), qui consiste à mettre une majuscule au début des mots (généralement à partir du deuxième), par exemple `premierNombre` ou `dateNaissance`.
- Pour donner un nom composé à une variable entièrement en majuscules, on peut utiliser le «tiret bas» ou underscore (par ex. `PREMIER_NUMERATEUR`).
- Les mots-clés du langage sont interdits (par exemple `for`, `if`, `while`)

Comment déclarer

Quelle instruction permet de déclarer :

- le jour d'une date ? _____
- l'heure d'un moment ? _____
- le prix d'un produit en grande surface ? _____
- votre nom ? _____

3 Opérateurs et expressions

Les **opérateurs** agissent sur les **variables** et les **constantes** pour former des **expressions**.

Une expression est donc une combinaison cohérente de variables, de constantes et d'opérateurs, éventuellement accompagnés de parenthèses.

3.1 Les opérateurs arithmétiques élémentaires en algo

Ce sont les opérateurs binaires bien connus :

- `+` addition
- `-` soustraction
- `*` multiplication
- `/` division réelle

Ils agissent sur des variables ou expressions à valeurs entières ou réelles.

Plusieurs opérateurs peuvent être utilisés pour former des expressions plus ou moins complexes, en tenant compte des règles de calcul habituelles, notamment la priorité de la multiplication et de la division sur l'addition et la soustraction.

Il est aussi permis d'utiliser des parenthèses, par exemple `a - (b + c * d) / x`.

Tout emploi de la division devra être accompagné d'une réflexion sur la valeur du dénominateur, une division par 0 entraînant toujours l'arrêt d'un algorithme.

3.2 Les opérateurs arithmétiques élémentaires en Java

Ce sont les opérateurs binaires bien connus :

- + addition
- - soustraction
- * multiplication
- / division réelle si au moins un des deux opérandes est réel
- / division entière si les deux opérandes sont entiers

Ils agissent sur des variables ou expressions à valeurs entières ou réelles.

Plusieurs opérateurs peuvent être utilisés pour former des expressions plus ou moins complexes, en tenant compte des règles de calcul habituelles, notamment la priorité de la multiplication et de la division sur l'addition et la soustraction.

Il est aussi permis d'utiliser des parenthèses, par exemple $a - (b + c * d)/x$.

Tout emploi de la division devra être accompagné d'une réflexion sur la valeur du dénominateur, une division par 0 entraînant un arrêt du programme.

3.3 Les opérateurs DIV et MOD en algo

Ce sont deux opérateurs très importants qui ne peuvent s'utiliser qu'avec des variables entières :

- DIV division entière
- MOD reste de la division entière

3.4 Les opérateurs DIV et MOD en Java

Ce sont deux opérateurs très importants qui ne peuvent s'utiliser qu'avec des variables entières :

- / division entière si les deux opérandes sont entiers
- % reste de la division entière

3.5 Les fonctions mathématiques complexes en algo

- L'élévation à la puissance sera notée $**$ ou $^$.
- Pour la racine carrée d'une variable x nous écrirons \sqrt{x} . Attention, pour ce dernier, de veiller à ne l'utiliser qu'avec un radicant positif!
Exemple : $(-b + \sqrt{(b * *2 - 4 * a * c)}) / (2 * a)$
- À votre avis, pourquoi ne pas avoir écrit «4ac »et «2a » ?

- Si nécessaire, on se permettra d'utiliser les autres fonctions mathématiques sous leur forme la plus courante dans la plupart des langages de programmation (exemples : $\sin(x)$, $\tan(x)$, $\log(x)$, $\exp(x)$. . .)

3.6 Les fonctions mathématiques complexes en Java

L'essentiel des fonctions mathématiques se trouvent dans la classe `Math`.

- `Math.pow(a,b)` est l'élévation à la puissance a^b
- `Math.sqrt(x)` est la racine carrée d'une variable x , \sqrt{x} . Attention, pour ce dernier, de veiller à ne l'utiliser qu'avec un radicant positif, sinon, une exception sera générée !
- Si nécessaire, les autres fonctions mathématiques `Math.sin(x)`, `Math.tan(x)`, `Math.log(x)`, `Math.exp(x)`. . .
- Toutes les informations se trouvent dans l'API (docs.oracle.com/javase/8/docs/api/), à la classe `Math`.

4 L'affectation d'une valeur à une variable

Cette opération est probablement l'opération la plus importante. En effet, une variable ne prend son sens réel que si elle reçoit à un moment donné une valeur. Il y a deux moyens de donner une valeur à une variable.

4.1 Affectation interne en algo

On parle d'**affectation interne** lorsque la valeur d'une variable est «calculée» par l'exécutant de l'algorithme lui-même à partir de données qu'il connaît déjà : `nomVariable ← expression` (une expression est une combinaison de variables et d'opérateurs). **L'expression a une valeur.**

Les exemples d'affectation sont-ils corrects ?

- ☐ `somme ← nombre1 + nombre2`
- ☐ `denRes ← den1 * den2`
- ☐ `cpt ← cpt + 1`
- ☐ `delta ← b**2 - 4*a*c`
- ☐ `maChaine ← "Bonjour"`
- ☐ `test ← a = b`
- ☐ `somme + 1 ← 3`
- ☐ `somme ← 3n`

Remarques

- Il est de règle que le résultat de l'expression à droite du signe d'affectation (\leftarrow) soit de même type que la variable à sa gauche. On tolère certaines exceptions :
 - `varEntière \leftarrow varRéelle` : dans ce cas le contenu de la variable sera la valeur tronquée de l'expression réelle.
Par exemple si «nb »est une variable de type entier, son contenu après l'instruction `nb \leftarrow 15/4` » sera 3.
 - `varRéelle \leftarrow varEntière` : ici, il n'y a pas de perte de valeur.
 - `varChaine \leftarrow varCaractère` : équivalent à `varChaine \leftarrow chaine(varCaractère)`.
 - Le contraire n'est évidemment pas accepté.
- Seules les variables déclarées peuvent être affectées, que ce soit par l'affectation externe ou interne !
- Nous ne mélangerons pas la déclaration d'une variable et son affectation interne dans une même ligne de code, donc pas d'instructions hybrides du genre `x \leftarrow 2 : entier` ou encore `x : entier(0)`.
- Pour l'affectation interne, toutes les variables apparaissant dans l'expression doivent avoir été affectées préalablement. Le contraire provoquerait un arrêt de l'algorithme.

4.2 Affectation interne en Java

On parle d'**affectation interne** lorsque la valeur d'une variable est «calculée »par l'exécutant de l'algorithme lui-même à partir de données qu'il connaît déjà : `nomVariable = expression`; (une expression est une combinaison de variables et d'opérateurs). **L'expression a une valeur.**

Les exemples d'affectation sont-ils corrects ?

- ☐ `somme = nombre1 + nombre2;`
- ☐ `denRes = den1 * den2;`
- ☐ `cpt = cpt + 1;`
- ☐ `delta = b**2 - 4*a*c;`
- ☐ `maChaine = "Bonjour";`
- ☐ `somme + 1 = 3;`
- ☐ `somme = 3n;`

4.3 Affectation externe en algo

On parle d'**affectation externe** lorsque la valeur à affecter à une variable est donnée par l'utilisateur qui la communique à l'exécutant quand celui-ci

le lui demande : cette valeur est donc externe à la procédure (l'ordinateur ne peut la deviner lui-même!)

L'affectation externe est donc la primitive qui permet de recevoir de l'utilisateur, au moment où l'algorithme se déroule, une ou plusieurs valeur(s) et de les affecter à des variables en mémoire.

Nous noterons : `lire liste_de_variables_à_lire`

Exemples

```
lire nombre1, nombre2
```

```
lire num1, den1, num2, den2
```

L'exécution de cette instruction provoque une pause dans le déroulement de l'algorithme ; l'exécutant demande alors à l'utilisateur les valeurs des variables à lire. Ces valeurs viennent donc de l'extérieur ; une fois introduites dans le système, elles sont affectées aux variables concernées et l'algorithme peut reprendre son cours. Les possibilités d'introduction de données sont nombreuses : citons par exemple l'encodage de données au clavier, un clic de souris, le toucher d'un écran tactile, des données provenant d'un fichier, etc.

4.4 Affectation externe en Java

On parle d'**affectation externe** lorsque la valeur à affecter à une variable est donnée par l'utilisateur qui la communique à l'exécutant quand celui-ci le lui demande : cette valeur est donc externe à la procédure (l'ordinateur ne peut la deviner lui-même!)

L'affectation externe est donc la primitive qui permet de recevoir de l'utilisateur, au moment où l'algorithme se déroule, une ou plusieurs valeur(s) et de les affecter à des variables en mémoire.

Nous noterons :

```
import java. util .Scanner;
// ...
Scanner clavier = new Scanner(System.in);
// ...
int nombre1 = clavier. nextInt ();
```

Les différentes lectures possibles sont :

- pour un entier : `clavier.nextInt()` ;
- pour un réel : `clavier.nextDouble()` ;
- pour un booléen : `clavier.nextBoolean()` ;
- pour un mot : `clavier.next()`
- pour une ligne : `clavier.nextLine()` ;

- pour un caractère : `clavier.next.charAt(0)` ;

4.5 Communication des résultats en algo

L'instruction de communication des résultats consiste à donner à l'extérieur (donc à l'utilisateur) la valeur d'un résultat calculé au cours de l'exécution de l'algorithme.

Nous noterons : **afficher expression** ou **liste de variables séparées par des virgules** qui signifie que la valeur d'une expression (ou celles des différentes variables mentionnées) sera fournie à l'utilisateur (par exemple par un affichage à l'écran ou par impression sur listing via l'imprimante, etc.).

Exemples

afficher `nb1`

afficher "Le premier nombre vaut ", `nb1`, " et le seconde nombre vaut ", `nb2`

Remarques :

- Ce ne serait pas une erreur fondamentale de remplacer lire par recevoir ou afficher par écrire. Il n'y a évidemment pas de confusion possible à partir du moment où l'on sait qu'il s'agit de primitives d'échange entre l'extérieur et l'ordinateur exécutant la procédure, mais par principe, il est conseillé d'utiliser une syntaxe commune et limitée à un petit nombre de mots-clés.
- Comme pour l'affectation interne, on ne peut afficher que des expressions dont les variables qui la composent ont été affectées préalablement.

4.6 Communication des résultats en Java

L'affichage de la valeur d'une variable `var` est donnée par l'instruction

```
System.out.println (var);
```

Si nous voulons précéder l'affichage de la valeur d'une variable `var` par un message, nous utiliserons l'instruction

```
System.out.println ("La variable var vaut " + var);
```