OFPPT

ROYAUME DU MAROC

مكتب التكوين المهني وإنعاش الشغل

Office de Formation Professionnelle et de la Promotion du Travail

DIRECTION RECHERCHE ET INGENIERIE DE FORMATION

RESUME THEORIQUE
&
GUIDE DES TRAVAUX PRATIQUES

MODULE N°3 :

Titre : Codification d'un algorithme et programmation procédurale

SECTEUR : **Tertiaire**

SPECIALITE : **Technicien Spécialisé en Développement Informat**

NIVEAU : **TS**

Mars 2006

REMERCIEMENT

La DRIF remercie les personnes qui ont contribué à l'élaboration du présen

Pour la supervision :

MME.BENNANI WAFAE	DIRECTRICE CDC TERTIAIRE & TIC
M. ESSABKI NOURDDINE	CHEF DE DIVISION CCFF

Pour la conception :

- JELLAL ABDELILAH	Formateur animateur au CDC Tertiaire & TIC
- KHLIFA AIT TALEB	Formateur à l'ITA GUELMIM

Pour la validation :

Les utilisateurs de ce document sont invités à communiquer à la DRIF toutes les remarques et suggestions afin de les prendre en considération pour l'enrichissement et l'amélioration de ce programme.

Said Slaoui
DRIF

OBJECTIF OPERATIONNELS DE PREMIER NIVEAU
DE COMPORTEMENT

COMPORTEMENT ATTENDU

Pour démontrer sa compétence, le stagiaire doit **codifier un algorithme et utiliser un langage procédural** selon les conditions, les critères et les précisions qui suiven

CONDITIONS D'EVALUATION

- Travail individuel effectué avec un PC équipé d'un environnement de développement :
- éditeur de texte
- Le langage de programmation Java
- une interface homme machine graphique(type Windows) n'est pas indispensable
- l'utilisation d'un formalisme de représentation des algorithmes est obligatoire.

CRITERES GENERAUX DE PERFORMANCE

- Utilisation des commandes appropriées.
- Respect du temps alloué.
- Respect des règles d'utilisation du matériel et logiciel Informatique.

PRECISIONS SUR LE COMPORTEMENT ATTENDU	CRITERES PARTICULIERS DE PERFORMANCE
A. Structurer le programme à codifier	• Analyse judicieuse des composants du programme. • Enumération des différentes instructions du programme • Structuration correcte de l'enchaînement des instructions dans un diagramme • Définition juste du format d'un algorithme • Définition juste des instructions d' E/S
D. Utiliser les instructions de base d'un algorithme	• Utilisation appropriée des instructions conditionnelles • Utilisation juste du syntaxe de la boucle • Utilisation judicieuse des notions fondamentales d'un tableau : - Notion de tableau une dimension - Notion de tableau multi - dimensions • Déclaration juste des tableaux • Affectation correcte des tableaux • Utilisation pertinente des tests booléens • Pratique approprié de la recherche dichotomique • Imbrication juste des structures répétitives et alternatives • Regroupement correct des instructions adéquates en fonctions et procédures cohérentes et réutilisables
C. Utiliser les fichiers	• Structuration correcte des données au sein d'un fichier texte • Définition judicieuse du type d'accès • accès séquentiel • accès direct (ou aléatoire) • Ouverture correcte d'un fichier texte pour : • Lecture • Ecriture
D. Traduire l'algorithme dans le langage de programmation JAVA	• Codification correcte de l'algorithme selon les instructions du langage JAVA • Utilisation judicieuse du compilateur (messages) et des outils de déboguage • Test de l'appel d'un sous programme • Correction éventuelle des erreurs

Partie 01 : Algorithme

1. INTRODUCTION

1.1. Notion de programme

Si l'on s'intéresse aux applications de l'ordinateur, on s'aperçoit qu'elles sont très nombreuses. En voici quelques exemples :

- Etablissement de feuille de payes, de factures
- Gestion de stocks
- Calcul de la trajectoire d'un satellite
- Suivi médical de patients dans un hôpital
- …

Un ordinateur pour qu'il puisse effectuer des tâches aussi variées il suffit de le programmer. Effectivement l'ordinateur est capable de mettre en mémoire un programme qu'on lui fournit puis l'exécuter.

Plus précisément, l'ordinateur possède un ensemble limité d'opérations élémentaires qu'il sait exécuter. Un programme est constitué d'un ensemble de directives, nommées instructions, qui spécifient :

- les opérations élémentaires à exécuter
- la façon dont elles s'enchaînent.

Pour s'exécuter, un programme nécessite qu'on lui fournisse ce qu'on peut appelé « informations données » ou plus simplement « données ». En retour, le programme va fournir des « informations résultats » ou plus simplement résultats.

Par exemple un programme de paye nécessite des informations données : noms des employés, situations de famille, nombres d'heures supplémentaires, etc… Les résultats seront imprimés sur les différents bulletins de paye.

1.2. Le processus de la programmation

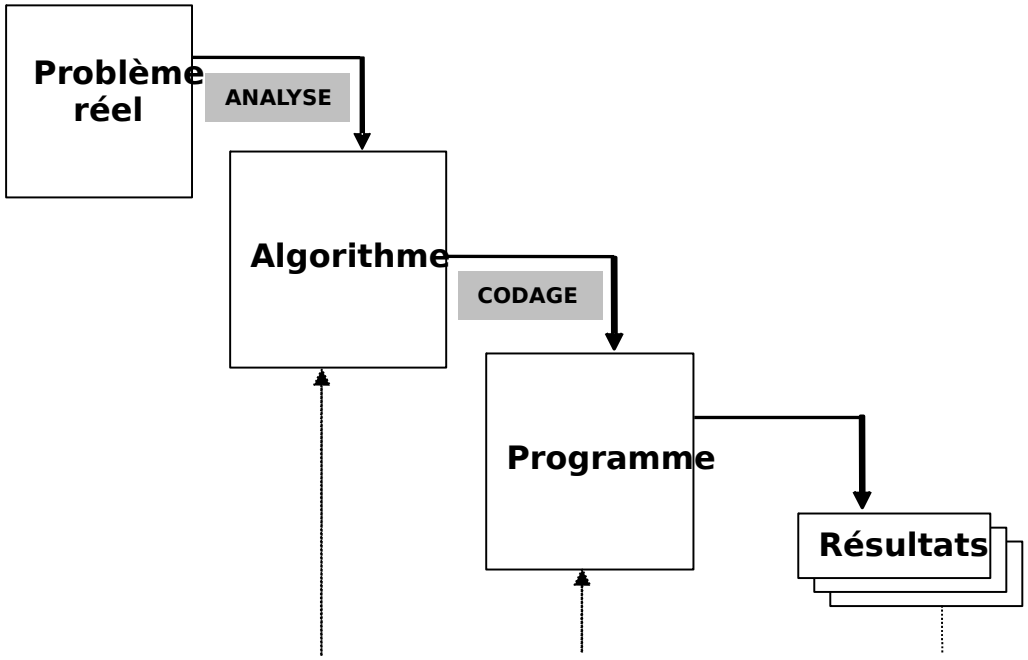
La programmation consiste, avant tout, à déterminer la démarche permettant d'obtenir, à l'aide d'un ordinateur, la solution d'un problème donné.

Le processus de la programmation se déroule en deux phases :

- dans un premier temps, on procède à ce qu'on appelle l'analyse du problème posé ou encore la recherche d'un algorithme[1] qui consiste à définir les différentes étapes de la résolution du problème. C'est la partie essentielle dans le processus de programmation. Elle permet de définir le contenu d'un programme en termes de données et d'actions.
- Dans un deuxième temps, on exprime dans un langage de programmation donné, le résultat de l'étape précédente. Ce travail, quoi qu'il soit facile, exige le respect strict de la syntaxe du langage de programmation.

Lors de l'étape d'exécution, il se peut que des erreurs syntaxiques sont signalées, ce qui entraîne des corrections en général simple ou des erreurs sémantiques plus difficiles à déceler. Dans ce dernier cas, le programme produit des résultats qui ne correspondent pas à ceux escomptés : le retour vers l'analyse sera alors inévitable.

[1] Un algorithme est une suite d'actions que devra effectuer un ordinateur pour arriver à un résultat, à partir d'une situation donnée.



Les différentes étapes du processus de programmation

Donc, la résolution d'un problème passe tout d'abord par la recherche d'un algorithme.
L'objectif de ce cours est de vous fournir les éléments de base intervenant dans un algorithme : variable, type, instructions d'affectation, de lecture, d'écriture, structures.

2. LES VARIABLES

2.1. La notion de variable

Dans un programme informatique, on va avoir en permanence besoin de stocker provisoirement en mémoire des valeurs. Il peut s'agir de données issues du disque dur ou fournies par l'utilisateur (frappées au clavier). Il peut aussi s'agir de résultats obtenus par le programme, intermédiaires ou définitifs. Ces données peuvent être de plusieurs types (on en reparlera) : elles peuvent être des nombres, du texte, etc. Dès que l'on a besoin de stocker une information au cours d'un programme, on utilise une variable.
Une variable est un nom qui sert à repérer un emplacement donné de la mémoire, c'est à dire que la variable ce n'est qu'une adresse de mémoire.
Cette notion contribue considérablement à faciliter la réalisation des programmes. Elle permet de manipuler des données sans avoir à se préoccuper de l'emplacement qu'elles occupent effectivement en mémoire. Pour cela, il vous suffit tout simplement de leur choisir un nom. Bien entendu, la chose n'est possible que parce qu'il existe un programme de traduction (compilateur, interpréteur) de votre programme dans le langage machine ; c'est lui qui attribuera une adresse à chaque variable.
Le programmeur ne connaît que les noms A, MONTANT, RACINE… Il ne se préoccupe pas des adresses qui leur sont attribuées en mémoires.
Le nom (on dit aussi identificateur) d'une variable, dans tous les langages, est formé d'une ou plusieurs lettres ; les chiffres sont également autorisés à condition de ne pas apparaître au début du nom. La plupart des signes de ponctuation sont exclus en particulier les espaces.
Par contre, le nombre maximum de caractères autorisés varie avec les langages. Il va de deux dans certains langages jusqu'à quarante.
Dans ce cours, aucune contrainte de longueur ne vous sera imposée. De même nous admettrons que les lettres peuvent être indifférents des majuscules ou des minuscules.

Remarque : Pour les noms des variables choisissez des noms représentatifs des informations qu'ils désignent ; ainsi MONTANT est un meilleur choix que X pour désigner le montant d'une facture.

Une variable peut être caractérisé aussi par sa valeur. A un instant donné, une variable ne peut contenir qu'une seule valeur. Bien sûr, cette valeur pourra évoluer sous l'action de certaines instructions du programme.
Outre le nom et la valeur, une variable peut être caractérisée par son type. Le type d'une variable définit la nature des informations qui seront représentées dans les variables (numériques, caractères…).
Ce type implique des limitations concernant les valeurs qui peuvent être représentées. Il limite aussi les opérations réalisables avec les variables correspondantes. Ainsi, les opérations arithmétiques (addition, soustraction, multiplication, division) possibles des variables numériques, n'ont aucun sens pour des variables de type caractères. Par contre les comparaisons seront possibles pour les deux types.

2.2. Déclaration des variables

La première chose à faire tout au début de l'algorithme, avant de pouvoir utiliser des variables, c'est de faire la déclaration des variables.
Lorsqu'on déclare une variable, on lui attribue un nom et on lui réserve un emplacement mémoire. La taille de cet emplacement mémoire dépend du type de variable. C'est pour cette raison qu'on doit préciser lors de la déclaration le type du variable.
La syntaxe d'une déclaration de variable est la suivante :

VARIABLE nom : TYPE

ou

VARIABLES nom1, nom2,… : TYPE

2.3. Types de variables

2.3.1. Type numérique

Commençons par le cas très fréquent, celui d'une variable destinée à recevoir des nombres.
Généralement, les langages de programmation offrent les types suivants :

- ENTIER

Le type entier désigne l'ensemble des nombres entiers négatifs ou positifs dont les valeurs varient entre -32 768 à 32 767.
On écrit alors :

VARIABLES i, j, k : ENTIER

- REEL

Le type réel comprend les variables numériques qui ont des valeurs réelles. La plage des valeurs du type réel est :
$-3{,}40 \times 10^{38}$ à $-1{,}40 \times 10^{45}$ pour les valeurs négatives
$1{,}40 \times 10^{-45}$ à $3{,}40 \times 10^{38}$ pour les valeurs positives
On écrit alors :

VARIABLES x, y : REEL

Remarque : Le type de variable choisi pour un nombre va déterminer les valeurs maximales et minimales des nombres pouvant être stockés dans la variable. Elle détermine aussi la précision de ces nombres (dans le cas de nombres décimaux).

2.3.2. Type chaîne

En plus, du type numérique on dispose également du type chaîne (également appelé caractère ou alphanumérique).
Dans une variable de ce type, on stocke des caractères, qu'il s'agisse de lettres, de signes de ponctuation, d'espaces, ou même de chiffres. Le nombre maximal de caractères pouvant être stockés dans une seule variable chaîne dépend du langage utilisé.
On écrit alors :

VARIABLE nom, prenom, adresse : CHAINE

Une chaîne de caractères est notée toujours soit entre guillemets, soit entre des apostrophes.
Cette notation permet d'éviter les confusions suivantes :

- Confondre un chiffre et une suite de chiffres. Par exemple, 423 peut représenter le nombre 423 (quatre cent vingt-trois), ou la suite de caractères 4, 2, et 3.
- La confusion qui consiste à ne pas pouvoir faire la distinction entre le nom d'une variable et son contenu.

Remarque : Pour les valeurs des variables de type chaîne, il faut respecter la casse. Par exemple, la chaîne ″Salut″ est différente de la chaîne ″salut″.

2.3.3. Type booléen

Dans ce type de variables on y stocke uniquement des valeurs logiques VRAI ou FAUX, TRUE ou FALSE, 0 ou 1.
On écrit alors :

VARIABLE etat : BOOLEEN

2.3.4. Opérateurs et expressions

2.3.4.1. Opérateurs

Un opérateur est un signe qui relie deux variables pour produire un résultat.
Les opérateurs dépendent des types de variables mis en jeu.
Pour le type numérique on a les opérateurs suivants :

+ : Addition
- : Soustraction
***** : Multiplication
/ : Division
^ : Puissance

Tandis que pour le type chaîne, on a un seul opérateur qui permet de concaténer deux chaînes de caractères. Cet opérateur de concaténation est noté &.
Par exemple : la chaîne de caractères ″Salut″ concaténer à la chaîne ″tout le monde″ donne comme résultat la chaîne ″Salut tout le monde″.

2.3.4.2. Expressions

Une expression est un ensemble de variables (ou valeurs) reliées par des opérateurs et dont la valeur du résultat de cette combinaison est unique.
Par exemple :

7 5+4 x + 15 – y/2 nom & prenom

où x et y sont des variables numériques (réels ou entiers) et nom et prenom sont des variables chaîne.

Dans une expression où on y trouve des variables ou valeurs numériques, l'ordre de priorité des opérateurs est important. En effet, la multiplication et la division sont prioritaires par rapport à l'addition et la soustraction.
Par exemple, 12 * 3 + 5 donne comme résultat 41.
Si l'on veut modifier cette ordre de priorité on sera obligé d'utiliser les parenthèse.
Par exemple, 12 * (3 + 5) donne comme résultat 96.

2.3.5. L'instruction d'affectation

L'instruction d'affection est opération qui consiste à attribuer une valeur à une variable. On la notera avec le signe ←.
Cette instruction s'écrit :

VARIABLE← valeur

Par exemple : MONTANT ← 3500.
On dit qu'on affecte (ou on attribue) la valeur 3500 à la variable numérique MONTANT.
Si dans une instruction d'affectation, la variable à laquelle on affecte la valeur et la valeur affectée ont des types différents, cela provoquera une erreur.
On peut aussi attribuer à une variable la valeur d'une variable ou d'une expression de façon générale.
On écrit :

VARIABLE← EXPRESSION

Par exemple :

A ← B
A ← B * 2 + 5

Dans ce cas, l'instruction d'affectation sera exécutée en deux temps :

- D'abord, on calcule la valeur de l'expression
- On affecte la valeur obtenue à la variable à gauche.

On peut même avoir des cas où la variable de gauche qui figure dans l'expression à droite.
Par exemple :

A ← A + 5

Dans cette exemple, après l'exécution de l'instruction d'affectation la valeur de la variable A sera augmenter de 5.

Remarque :

Dans une instruction d'affection on a toujours :

- à gauche de la flèche d'affection un nom de variable
- à droite de la flèche d'affectation une valeur ou une expression
- l'expression à droite de la flèche doit être du même type que la variable située à gauche.

Si dans une instruction d'affectation une ces points n'est pas respecté, cela engendra une erreur.
Il est à noter que l'ordre dans lequel sont écrites les instructions est essentiel dans le résultat final.

Exemple :

CAS I	CAS II
A ← 15	A ← 30
A ← 30	A ← 15

Après exécution des deux instructions d'affection, la valeur de A sera :

- Cas I : 30
- Cas II : 15

Exercices

1. Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

```
Variables A, B : Entier
Début
        A ← 1
B       ← A + 3
        A ← 3
Fin
```

2. Quelles seront les valeurs des variables A, B et C après exécution des instructions suivantes ?

```
Variables A, B, C : Entier
Début
        A ← 5
B       ← 3
C       ← A + B
A       ← 2
C       ← B – A
Fin
```

3. Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

```
Variables A, B : Entier
Début
 A      ← 5
B       ← A + 4
A       ← A + 1
B       ← A – 4
Fin
```

4. Quelles seront les valeurs des variables A, B et C après exécution des instructions suivantes ?

```
Variables A, B, C : Entier
Début
 A      ← 3
B       ← 10
C       ← A + B
B       ← A + B
A       ← C
Fin
```

5. Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

```
Variables A, B : Entier
Début
 A      ← 5
B       ← 2
A       ← B
B       ← A
Fin
```

Questions : les deux dernières instructions permettent-elles d'échanger les deux valeurs de B et A ? Si l'on inverse les deux dernières instructions, cela change-t-il quelque chose ?

6. Ecrire un algorithme permettant d'échanger les valeurs de deux variables A et B, et ce quel que soit leur contenu préalable.

7. On dispose de trois variables A, B et C. Ecrivez un algorithme transférant à B la valeur de A, à C la valeur de B et à A la valeur de C (toujours quels que soient les contenus préalables de ces variables).

8. Que produit l'algorithme suivant ?

```
Variables A, B, C : Caractères
Début
 A  ← "423"
B   ← "12"
C   ← A + B
Fin
```

9. Que produit l'algorithme suivant ?

```
Variables A, B : Caractères
Début
 A  ← "423"
B   ← "12"
C   ← A & B
Fin
```

Solutions

1.

Après exécution de l'instruction	La valeur des variables est :
A ← 1	A = 1 B = ?
B ← A + 3	A = 1 B = 4
A ← 3	**A = 3 B = 4**

2.

Après exécution de l'instruction	La valeur des variables est :
A ← 5	A = 5 B = ? C = ?
B ← 3	A = 5 B = 3 C = ?
C ← A + B	A = 5 B = 3 C = 8
A ← 2	A = 2 B = 3 C = 8
C ← B – A	**A = 2 B = 3 C = 1**

3.

Après exécution de l'instruction	La valeur des variables est :
A ← 5	A = 5 B = ?
B ← A + 4	A = 5 B = 9
A ← A + 1	A = 6 B = 9
B ← A – 4	**A = 6 B = 2**

4.

Après exécution de l'instruction	La valeur des variables est :
A ← 3	A = 3 B = ? C = ?
B ← 10	A = 3 B = 10 C = ?
C ← A + B	A = 3 B = 10 C = 13
B ← A + B	A = 3 B = 13 C = 13
A ← C	**A = 13 B = 13 C = 13**

5.

Après exécution de l'instruction	La valeur des variables est :
A ← 5	A = 5 B = ?
B ← 2	A = 5 B = 2
A ← B	A = 2 B = 2
B ← A	**A = 2 B = 2**

Les deux dernières instructions ne permettent donc pas d'échanger les deux valeurs de B et A, puisque l'une des deux valeurs (celle de A) est ici écrasée.
Si l'on inverse les deux dernières instructions, cela ne changera rien du tout, hormis le fait que cette fois c'est la valeur de B qui sera écrasée.

6. L'algorithme est :

```
Début
        C ← A
A       ← B
B       ← C
Fin
```

On est obligé de passer par une variable dite temporaire (la variable C).

7. L'algorithme est :

```
Début
        D ← C
C       ← B
        B ← A
A       ← D
Fin
```

En fait, quel que soit le nombre de variables, une seule variable temporaire suffit.

8. Il ne peut produire qu'une erreur d'exécution, puisqu'on ne peut pas additionner des caractères.

9. On peut concaténer ces variables. A la fin de l'algorithme, C vaudra donc "42312".

3. LES INSTRUCTIONS DE LECTURE ET ECRITURE

Considérons le programme suivant :

```
VARIABLE A : ENTIER
Début
        A ← 12 ^ 2
Fin
```

Il permet de calculer le carré de 12.
Le problème de ce programme, c'est que, si l'on veut calculer le carré d'un autre nombre que 12, il faut réécrire le programme.

D'autre part, la machine calcule le résultat et l'utilisateur qui fait exécuter ce programme ne saura jamais que ce résultat correspond au carré de 12.
C'est pour cela qu'il faut introduire des instructions qui permettent le dialogue avec l'utilisateur.
En effet, il existe une instruction qui permet à l'utilisateur de faire entrer des valeurs au clavier pour qu'elles soient utilisées par le programme. La syntaxe de cette instruction de lecture est :

LIRE NomVariable

Lorsque le programme rencontre une instruction LIRE, l'exécution du programme s'interrompt, attendant la saisie d'une valeur au clavier.

Dès que l'on frappe sur la touche ENTER, l'exécution reprend.

Une autre instruction permet au programme de communiquer des valeurs à l'utilisateur en les affichant à l'écran. La syntaxe de cette instruction d'écriture est :

ECRIRE NomVariable

ou de façon générale

ECRIRE Expression

Remarque : Avant de lire une variable, il est fortement conseillé d'écrire des libellés à l'écran, afin de prévenir l'utilisateur de ce qu'il doit frapper. La même chose pour l'instruction d'écriture.

Exemple :

```
Variables A, CARRE : Réels
DEBUT
    Ecrire 'Entrez un nombre'
    Lire A
    CARRE ← A * A
    Ecrire 'Le carré de ce nombre est : '
    Ecrire CARRE
FIN
```

Exercices

1. Quel résultat produit le programme suivant ?

```
VARIABLES Val, Double : ENTIERS
Début
    Val ← 231
Double  ← Val * 2
    ECRIRE Val
    ECRIRE Double
Fin
```

2. Ecrire un programme qui demande deux nombres entiers à l'utilisateur, puis qui calcule et affiche le somme de ces nombres.

3. Ecrire un programme qui lit le prix HT d'un article, le nombre d'articles et le taux de TVA, et qui fournit le prix total TTC correspondant. Faire en sorte que des libellés apparaissent clairement.

4. Ecrire un programme qui lit une valeur et qui nous calcule l'inverse de cette valeur.

5. Le surveillant général d'un établissement scolaire souhaite qu'on lui écrit un programme qui calcule, pour chaque élève, la moyenne des notes des cinq matières. Ces matières sont avec leur coefficient :

MATIERE	COEFFICIENT
Math	5

Physique 5	
Français 4	
Anglais 2	
Histoire - Géographie	2

Corrections

1. On verra apparaître à l'écran :

```
231
462
```

2. Le programme est :

```
VARIABLES A, B, SOMME : ENTIERS
Début
    ECRIRE 'Entrez le premier nombre'
    Lire A
    ECRIRE 'Entrez le deuxième nombre'
    Lire B
    SOMME ← A + B
    ECRIRE 'La somme de ces deux nombres est : '
    ECRIRE SOMME
Fin
```

Remarque : On peut remplacer les deux derniers lignes par :

```
    ECRIRE 'La somme de ces deux nombres est : ', SOMME
```

3. Le programme est :

```
VARIABLES pht, ttva, pttc : REELS
VARIABLE nb : ENTIER
    Début
        ECRIRE "Entrez le prix hors taxes :"
        LIRE pht
        ECRIRE "Entrez le nombre d'articles :"
        LIRE nb
        ECRIRE "Entrez le taux de TVA :"
        LIRE ttva
        Pttc ← nb * pht * (1 + ttva)
        ECRIRE "Le prix toutes taxes est : ", ttva
    Fin
```

4. Le programme est :

```
VARIABLES x, inverse : REELS
    Début
        ECRIRE "Entrez une valeur :"
        LIRE x
        inverse ← 1 / x
        ECRIRE "L'inverse est : ", inverse
    Fin
```

5. Le programme est :

```
VARIABLES mat, phy, ang, fra, hg, moyenne : REELS
    Début
        ECRIRE "Entrez la note de math :"
```

```
        LIRE mat
        ECRIRE "Entrez la note de physique :"
        LIRE phy
        ECRIRE "Entrez la note de  français :"
        LIRE fra
        ECRIRE "Entrez la note d'anglais :"
        LIRE ang
        ECRIRE "Entrez la note d'histoire-Géo :"
        LIRE hg
moyenne         ← ((mat + phy) * 5 + fra * 4 + (ang
+ hg) * 2) / 18
        ECRIRE "La moyenne est : ", moyenne
Fin
```

4. LA STRUCTURE ALTERNATIVE

4.1. Les conditions simples

Une condition simple consiste en une comparaison entre deux expressions du même type.
Cette comparaison s'effectue avec des opérateurs de comparaison. Voici la liste de ces opérateurs accompagnés de leur signification dans le cas des types numérique ou chaîne :

Opérateur	Signification numérique	Signification chaîne
=	égal à	égal à
< >	différent	différent
<	inférieur	placé avant dans l'ordre alphabétique
>	supérieur	placé après dans l'ordre alphabétique
<=	inférieur ou égal	placé avant dans l'ordre alphabétique ou égal
>=	supérieur ou égal	placé après dans l'ordre alphabétique ou égal

Pour la comparaison du type chaîne c'est l'ordre alphabétique qu'est utilisé dans le cas où l'on compare deux lettres majuscules ou minuscules. Mais si l'on compare majuscules et minuscules, il faut savoir que les majuscules apparaissent avant les minuscules. Ainsi, par exemple : "M" < "m".

4.2. Les conditions complexes

Certains problèmes exigent parfois de formuler des conditions qui ne peuvent pas être exprimées sous la forme simple vu en dessus. A cet effet, la plupart des langages autorisent des conditions formées de plusieurs conditions simples reliées entre elles par ce qu'on appelle des opérateurs logiques. Ces opérateurs sont : **ET**, **OU** et **NON**.

- Pour que la condition complexe,

$$condition1 \ \textbf{ET} \ condition2$$

soit VRAI, il faut impérativement que la $condition1$ soit VRAI et que la $condition2$ soit VRAI.

- Pour que la condition

$$condition1 \ \textbf{OU} \ condition2$$

soit VRAI, il suffit que $condition1$ soit VRAI ou $condition2$ soit VRAI. Il est à noter que cette condition complexe sera VRAI si $condition1$ et $condition2$ sont VRAI.

- Le NON inverse une condition :

$$\textbf{NON}(condition)$$

est VRAI si condition est FAUX, et il sera FAUX si condition est VRAI.

D'une manière générale, les opérateurs logiques peuvent porter, non seulement sur des conditions simples, mais aussi sur des conditions complexes. L'usage de parenthèses permet dans de tels cas de régler d'éventuels problèmes de priorité. Par exemple, la condition :

(a < 0 **ET** b > 1) **OU** (a > 0 **ET** b > 3)

est VRAI si l'une au moins des conditions entre parenthèses est VRAI.

4.3. La structure alternative

Supposons que nous avons besoin, dans un programme, d'afficher un message précisant que la valeur d'une variable est positive ou négative. Avec les instructions de base que nous avons vu (celles qui permettent la manipulation des variables : affectation, lecture, écriture), on ne peut pas. Il faut introduire une des instructions de structuration du programme (ces instructions servent à préciser comment doivent s'enchaîner chronologiquement ces instructions de base) qui donne la possibilité d'effectuer des choix dans le traitement réalisé. Cette instruction s'appelle la structure alternative. Sa syntaxe est :

SI *condition* **ALORS**
bloc 1 d'instructions
SINON
bloc 2 d'instructions
FIN SI

Si la condition mentionnée après **SI** est VRAI, on exécute le *bloc1 d'instructions* (ce qui figure après le mot **ALORS**); si la condition est fausse, on exécute le *bloc2 d'instructions* (ce qui figure après le mot **SINON**).

Exemple :

SI a > 0 **ALORS**
ECRIRE "valeur positive"
SINON
ECRIRE "valeur négative"
FIN SI

Dans ce programme, on vérifie si la valeur de a est supérieure à 0, on affichera le message "valeur positive". Dans le cas contraire, il sera affiche le message "valeur négative".

La structure alternative peut prendre une autre forme possible où l'une des parties du choix est absente. Elle s'écrit dans ce cas :

SI condition ALORS
bloc d'instructions
FIN SI

Exemple : Dans un programme de calcul du montant d'une facture, on applique une remise de 1% si le montant dépasse 5000 Dhs. Nous écrirons :

SI montant > 5000 **ALORS**
montant ← montant * 0.99
FIN SI

4.4. Les structures alternatives imbriquées

Il peut arriver que l'une des parties d'une structure alternative contienne à son tour une structure alternative. Dans ce cas, on dit qu'on a des structures alternatives imbriquées les unes dans les autres.

Exemple : Ecrire un programme qui donne l'état de l'eau selon sa température.

Variable Temp : **Entier**
Début
Ecrire "Entrez la température de l'eau :"
Lire Temp
Si Temp =< 0 **Alors**

```
            Ecrire "C'est de la glace"
        Sinon
        Si Temp < 100 Alors
            Ecrire "C'est du liquide"
        Sinon
            Ecrire "C'est de la vapeur"
        Finsi
        FinSi
Fin
```

On peut aussi écrire :

```
Variable Temp : Entier
Début
Ecrire "Entrez la température de l'eau :"
Lire Temp
Si Temp =< 0 Alors
   Ecrire "C'est de la glace"
Finsi
Si Temp > 0 Et Temp < 100 Alors
   Ecrire "C'est du liquide"
Finsi
Si Temp > 100 Alors
   Ecrire "C'est de la vapeur"
Finsi
Fin
```

La première version est plus simple à écrire et plus lisible. Elle est également plus performante à l'exécution. En effet, les conditions se ressemblent plus ou moins, et surtout on oblige la machine à examiner trois tests successifs alors que tous portent sur la même chose, la valeur de la variable Temp. Mais aussi, et surtout, nous avons fait des économies sur le temps d'exécution de l'ordinateur. Si la température est inférieure à zéro, celui-ci écrit « C'est de la glace » et passe directement à la fin, sans être ralenti par l'examen des autres possibilités.

4.5. Autre forme

Dans des langages de programmation, la structure alternative peut prendre une autre forme qui permet d'imbriquée plusieurs. Sa syntaxe est :

```
SELON expression
        valeur1 : action1
        valeur2 : action2
        …
        valeurN : actionN
        SINON : action
FIN SELON
```

Si *expression* est égale à *valeuri*, on exécute actioni et on passe à la suite de l'algorithme. Sinon on exécute action et on passe à la suite de l'algorithme.

Exercices

1. Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si leur produit est négatif ou positif (on laisse de côté le cas où le produit est nul). Attention toutefois : on ne doit pas calculer le produit des deux nombres.

2. Ecrire un algorithme qui demande trois noms à l'utilisateur et l'informe ensuite s'ils sont rangés ou non dans l'ordre alphabétique.

3. Ecrire un algorithme qui demande un nombre à l'utilisateur, et l'informe ensuite si ce nombre est positif ou négatif (on inclut cette fois le traitement du cas où le nombre vaut zéro).

4. Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si le produit est négatif ou positif (on inclut cette fois le traitement du cas où le produit peut être nul). Attention toutefois, on ne doit pas calculer le produit !

5. Ecrire un algorithme qui demande l'âge d'un enfant à l'utilisateur. Ensuite, il l'informe de sa catégorie :
- « Poussin » de 6 à 7 ans
- « Pupille » de 8 à 9 ans
- « Minime » de 10 à 11 ans
- « Cadet » après 12 ans

6. a partir d'un montant lu, on détermine un montant net par application d'une remise de :
- 1% si le montant est compris entre 2000 et 5000 Dhs (valeurs comprises)
- 2 % si le montant est supérieur à 5000 Dhs.

Solutions

1. Le programme est :

```
Variables m, n : Entier
Début
    Ecrire "Entrez deux nombres : "
    Lire m, n
    Si m * n > 0 Alors
        Ecrire "Leur produit est positif"
    Sinon
        Ecrire "Leur produit est négatif"
    Finsi
Fin
```

2. Le programme est :

```
Variables a, b, c : Caractère
Début
    Ecrire "Entrez successivement trois noms : "
    Lire a, b, c
    Si a < b et b < c Alors
        Ecrire "Ces noms sont classés alphabétiquement"
    Sinon
        Ecrire "Ces noms ne sont pas classés"
    Finsi
Fin
```

3. Le programme est :

```
Variable n : Entier
Début
    Ecrire "Entrez un nombre : "
    Lire n
    Si n < 0 Alors
        Ecrire "Ce nombre est négatif"
    SinonSi n = 0 Alors
        Ecrire "Ce nombre est nul"
    Sinon
        Ecrire "Ce nombre est positif"
```

```
        Finsi
Fin
```

4. Le programme est :

```
Variables m, n : Entier
Début
        Ecrire "Entrez deux nombres : "
        Lire m, n
        Si m = 0 OU n = 0 Alors
                Ecrire "Le produit est nul"
        SinonSi (m < 0 ET n < 0) OU (m > 0 ET n > 0) Alors
                Ecrire "Le produit est positif"
        Sinon
                Ecrire "Le produit est négatif"
        Finsi
Fin
```

5. Le programme est :

```
Variable age : Entier
Début
        Ecrire "Entrez l'âge de l'enfant : "
        Lire age
        Si age >= 12 Alors
                Ecrire "Catégorie Cadet"
        SinonSi age >= 10 Alors
                Ecrire "Catégorie Minime"
        SinonSi age >= 8 Alors
                Ecrire "Catégorie Pupille"
        SinonSi age >= 6 Alors
                Ecrire "Catégorie Poussin"
        Finsi
Fin
```

6. Le programme est :

```
Variables montant , taux , remise : Réels
Début
        Ecrire "Entrez le montant : "
        Lire montant
        Si montant <2000 Alors
                taux ← 0
        Sinon
        Si montant ≤ 5000 Alors
                taux ← 1
        Sinon
                taux ← 2
        Fin SI
        Fin Si
        Montant ← montant * (1 – taux / 100)
        Ecrire "Le montant net est : " , montant
Fin
```

5. LES STRUCTURES REPETITIVES

Reprenons le programme du surveillant général qui calcule la moyenne des notes. L'exécution de ce programme fournit la moyenne des notes uniquement pour un seul élève. S'il l'on veut les moyennes de 200 élèves, il faut ré exécuter ce programme 200 fois. Afin d'éviter cette tâche fastidieux d'avoir ré exécuter le programme 200 fois, on peut faire recourt à ce qu'on appelle les **structures répétitives**. On dit aussi les **structures itératives** ou **boucles**.
Une structure répétitive sert à répéter un ensemble d'instructions. Il existe trois formes de structures répétitives : **POUR**, **TANT QUE**, **REPETER**.

5.1. La structure POUR

Cette structure permet de répéter des instructions un nombre connu de fois. Sa syntaxe est :

POUR *compteur* = val_*initial* **A** *val_final* **PAS DE** *incrément*
Instructions à répéter
FIN POUR

compteur c'est ce qu'on appelle **compteur**. C'est une variable de type entier.
val_*initial* et *val_final* sont respectivement les valeur initiale et final prise par le compteur. Ce sont des valeurs entières.
incrément est la valeur d'augmentation progressive du compteur. La valeur par défaut du pas est de 1. Dans de telle on peut ne pas le préciser.

Remarques :

Pour un pas positif, la valeur négative doit être inférieure à la valeur finale. Pour un pas négatif, valeur négative doit être supérieure à la valeur finale.
Si la valeur initiale est égale à la valeur finale, la boucle sera exécutée une seule fois.

Exemple : Réécrivons le programme du surveillant général de façon qu'il puisse calculer les moyennes de 200 élèves.

```
VARIABLES mat, phy, ang, fra, hg, moyenne : REELS

VARIABLE i : ENTIER
    Début
POUR                i = 1 A 200
            ECRIRE      "Entrez la note de math :"
                LIRE mat
                ECRIRE "Entrez la note de physique :"
                LIRE phy
                ECRIRE "Entrez la note de  français :"
                LIRE fra
                ECRIRE "Entrez la note 'anglais :"
                LIRE ang
                ECRIRE "Entrez la note d'histoire-Géo :"
                LIRE hg
                moyenne ← ((mat + phy) * 5 + fra * 4 + (ang
                + hg) * 2) / 18
                ECRIRE "La moyenne est : ", moyenne
            FIN POUR
    Fin
```

Exercices

1. Ecrire un algorithme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre, présentée comme suit (cas où l'utilisateur entre le nombre 7) :

Table de 7 :

7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
…
7 x 10 = 70

2. Ecrire un algorithme qui demande un nombre de départ, et qui calcule la somme des entiers jusqu'à ce nombre. Par exemple, si l'on entre 5, le programme doit calculer :
1 + 2 + 3 + 4 + 5 = 15

3. Ecrire un algorithme qui demande un nombre de départ, et qui calcule sa factorielle.
NB : la factorielle de 8, notée 8 ! vaut 1 x 2 x 3 x 4 x 5 x 6 x 7 x 8

4. Ecrire un algorithme qui demande successivement 20 nombres à l'utilisateur, et qui lui dise ensuite quel était le plus grand parmi ces 20 nombres :

Entrez le nombre numéro 1 : 12
Entrez le nombre numéro 2 : 14
…
Entrez le nombre numéro 20 : 6

Le plus grand de ces nombres est : 14

Modifiez ensuite l'algorithme pour que le programme affiche de surcroît en quelle position avait été saisie ce nombre :
C'était le nombre numéro 2

5. Ecrire un algorithme qui :
- lit d'abord une valeur
- ensuite il va lire successivement 20 nombres.
- enfin il va déterminer combien de fois la première valeur a été saisie (sans compter la première saisie).

Solutions

1. Le programme est :

```
Variables i , valeur : Entiers
DEBUT
    Lire valeur
    POUR i = 1 A valeur
        Ecrire valeur & " X " & i & " = " & valeur * i
    FIN POUR
FIN
```

2. Le programme est :

```
Variables i , valeur , somme : Entiers
DEBUT
    Lire valeur
    somme ← 0
    POUR i = 1 A valeur
        somme ← somme + i
    FIN POUR
    Ecrire "La somme des " & valeur & " premiers entiers est : " & somme
```

FIN

3. Le programme est :

```
Variables i , valeur , factoriel : Entiers
DEBUT
    Lire valeur
    factoriel ← 1
    POUR i = 1 A valeur
        factoriel ← factoriel * i
    FIN POUR
    Ecrire "Le factoriel de " & valeur & " est : " & factoriel
Fin
```

4. Le programme est :

```
Variables i , a , max , pmax : Entiers
DEBUT
    Ecrire « Entrez le nombre numéro 1 »
    Lire a
    max ← a
    pmax ← 1
    POUR i = 2 A 20
        Ecrire « Entrez le nombre numéro » , i
        Lire a
        SI a > max ALORS
            max ← a
            pmax ← i
FIN SI
    FIN POUR
    Ecrire « Le plus grand nombre est : » , max
    Ecrire « Sa position est : » , pmax
FIN
```

5. Le programme est :

```
Variables i , a , b , S : Entiers
DEBUT
    Ecrire « Entrez un chiffre : »
    Lire a
    S ← 0
    POUR i = 1 A 20
        Ecrire « Entrez un nombre : »
        Lire b
        SI a = b ALORS
            S ← S + 1
FIN SI
    FIN POUR
    Ecrire « Le nombre de fois de saisie de » , a , « est : » , S
FIN
```

5.2. La structure TANT QUE

Cette structure permet de répéter les instructions **tant qu**'une condition est satisfaite. Sa syntaxe est :

```
TANT QUE condition
 Instructions à répéter
FIN TANT QUE
```

condition c'est une condition qu'on appelle parfois condition d'arrêt. Cette condition est testée avant la première exécution.
Cette structure diffère de la première par le fait qu'on va répéter des instructions pour un nombre de fois inconnu au préalable.

Exemple : Reprenant toujours le programme de notre surveillant. S'il ne sait pas combien de moyennes à calculer on ne pourra pas utiliser la structure **POUR**. Dans ce cas on est obligé d'utiliser la structure **TANT QUE**. Le programme sera alors :

```
Variables mat, phy, ang, fra, hg, moyenne : Réels
Variable reponse : Chaîne
DEBUT
        reponse ← "o"
        TANT QUE reponse = "o"
                Ecrire "Entrez la note de math :"
                Lire mat
                Ecrire "Entrez la note de physique :"
                Lire phy
                Ecrire "Entrez la note de  français :"
                Lire fra
                Ecrire "Entrez la note d'anglais :"
                Lire ang
                Ecrire "Entrez la note d'histoire-Géo :"
                Lire hg
                moyenne  ← ((mat + phy) * 5 + fra * 4 + (ang + hg) * 2) / 18
                Ecrire "La moyenne est : ", moyenne
                Ecrire "Voulez-vous continuer oui (o) /non (n) ?"
                Lire reponse
        FIN TANT QUE
FIN
```

Exercices

1. Ecrire un algorithme qui demande à l'utilisateur un nombre compris entre 1 et 3 jusqu'à ce que la réponse convienne.

2. Ecrire un algorithme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message : « Plus petit ! », et inversement, « Plus grand ! » si le nombre est inférieur à 10.

3. Ecrire un algorithme qui demande un nombre de départ, et qui ensuite affiche les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.

4. Ecrire un algorithme qui demande successivement des nombres à l'utilisateur, et qui lui dise ensuite quel était le plus grand parmi ces nombres et quel était sa position. La saisie des nombres s'arrête lorsque l'utilisateur entre un zéro.

5. Lire la suite des prix (en dhs entiers et terminée par zéro) des achats d'un client. Calculer la somme qu'il doit, lire la somme qu'il paye, et déterminer le reste à rendre.

Solutions

1. Le programme est :

```
Variable a : Réel
Début
```

```
Tant Que a < 1 OU a > 3
 Ecrire « Veuillez Saisir une valeur comprise entre 1 et 3 »
 Lire a
 Fin                TantQue
      Fin
```

2. Le programme est :

```
      Variable a : Réel
      Début
            Lire a
Tant Que a < 10 OU a > 20
 Si a < 10 Alors
  Ecrire « Plus grand ! »
 Sinon
                              Ecrire « Plus petit ! »
 Fin Si
 Lire a
Fin Tant Que
      Fin
```

3. Le programme est :

```
      Variable a , i : Réel
      Début
Ecrire « Entrez un nombre »
Lire a
i                ← a + 1
Tant Que i < a + 10
 Ecrire i
 i                          ← i + 1
Fin Tant Que
      Fin
```

4. Le programme est :

```
      Variables i , a , max , pmax : Entiers
      DEBUT
            Ecrire « Entrez le nombre numéro 1 »
            Lire  a
            max ← a
            pmax ← 1
            i ← 1
            TANT QUE a <> 0
                  i ← i + 1
                  Ecrire  « Entrez le nombre numéro » , i
                  Lire a
                  SI a > max ALORS
                        max ← a
                        pmax ← i
 FIN SI
            FIN TANT QUE
            Ecrire « Le plus grand nombre est : » , max
            Ecrire « Sa position est : » , pmax
      FIN
```

5. Le programme est :

```
      Variables prixlu , mdu , mpaye , reste : Entiers
```

```
DEBUT
 Ecrire      «Entrez le prix »
        Lire prixlu
        mdu ← 0
        mdu ← mdu + prixlu
        TANT QUE prixlu <> 0
                Ecrire « Entrez le prix »
                Lire prixlu
                mdu ← mdu + prixlu
        FIN TANT QUE
        Ecrire « Entrez le prix payé»
        Lire mpaye
        reste ← mpaye - mdu
        Ecrire « Le reste est : » , reste
FIN
```

5.3. La structure REPETER

Cette structure sert à répéter des instructions **jusqu'à** ce qu'une condition soit réalisée. Sa syntaxe est :

```
REPETER
 Instructions                    à répéter
JUSQU'A                 condition
```

Considérons le programme suivant :

```
Variables a , c : Entiers
DEBUT
        REPETER
                Lire a
                c ← c * c
                Ecrire c
        JUSQU'A a = 0
        Ecrire « Fin »
FIN
```

Les mots **REPETER** et **JUSQU'A** encadrent les instructions à répéter. Cela signifie que ces instructions doivent être répéter autant de fois jusqu'à ce que la variable **a** prennent la valeur 0.
Notez bien que le nombre de répétition dans cette structure n'est indiqué explicitement comme c'est la cas de la structure TANT QUE. Il dépend des données que l'on fournit au programme.
Pour bien expliciter cela, voyons ce que produira ce programme si l'on lui fournit successivement les valeurs 2, 4, 0. Le résultat se présentera ainsi :

```
4
16
0
Fin
```

Exercices

1. Ecrire un algorithme qui demande successivement des nombres à l'utilisateur, et qui calcule le nombre de valeurs saisies. La saisie des nombres s'arrête lorsque l'utilisateur entre le caractère « n » ou « N ».

2. Ecrire un algorithme qui demande successivement des nombres à l'utilisateur, et qui calcule leur moyenne. La saisie des nombres s'arrête lorsque l'utilisateur entre un zéro.

3. Modifiez l'algorithme de l'exercice 1, de façon qu'il nous renseigne sur le nombre des valeurs positives et sur le nombre des valeurs négatives. Ne comptez pas les valeurs nuls.

4. Ecrire un algorithme qui lit les caractères saisies par l'utilisateur. A la fin ce programme nous affichera la phrase saisie. La saisie des caractères s'arrête lorsqu'on tape point « . ». Pour l'utilisateur veut insérer un espace il lui suffit de tapez sur 0. Par exemple si l'utilisateur tape successivement les caractères « b » , « o », « n », « j », « o », « u », « r » , « t », « o », « u », « s », « . » , il nous affichera la chaîne « bonjour tous ».
Mais si il tape « b » , « o », « n », « j », « o », « u », « r » , « 0 », « t », « o », « u », « s », « . » , le programme affichera « bonjour tous ».

Solutions

1. le programme est :

```
Variables a , compteur : Entiers
Variable reponse : Chaîne
DEBUT
        compteur ← 0
        REPETER
         Ecrire      « Entrez un nombre : »
                Lire a
                compteur ← compteur + 1
                Ecrire « Voulez-vous continuez Oui/Non ? »
                Lire reponse
        JUSQU'A reponse = « N » ou reponse = « n »
        Ecrire « Le nombre de valeurs saisies est : » , compteur
FIN
```

2. Le programme est :

```
Variables a , somme , moyenne , compteur : Entiers
DEBUT
        compteur ← 0
somme           ← 0
        REPETER
         Ecrire      « Entrez un nombre : »
                Lire a
                compteur ← compteur + 1
                somme ← somme + a
        JUSQU'A a = 0
        Moyenne ← somme / compteur
        Ecrire « La moyenne de valeurs saisies est : » , moyenne
FIN
```

3. le programme est :

```
Variables a , npos , nneg : Entiers
Variable reponse : Chaîne
DEBUT
        npos ← 0
        nneg ← 0
        REPETER
         Ecrire      « Entrez un nombre : »
                Lire a
                SI a > 0 ALORS
                        npos ← npos + 1
                SINON
                        SI a < 0 ALORS
                                nneg ← nneg + 1
  FIN SI
```

```
    FIN SI
                Ecrire « Voulez-vous continuez Oui/Non ? »
                Lire reponse
            JUSQU'A reponse = « O » ou reponse = « o »
            Ecrire « Le nombre de valeurs positives saisies est : » , npos
            Ecrire « Le nombre de valeurs négatives saisies est : » , nneg
FIN
```

4. Le programme est :

```
    Variables caractere , phrase : Chaînes
    DEBUT
        phrase ← « »
        REPETER
         Ecrire      « Entrez une caractère : »
            Lire caractère
            SI caractere = « 0 » ALORS
                caractere ← «  »
            FIN SI
            phrase ← phrase +caractere
        JUSQU'A caractere = « . »
        Ecrire « La phrase résultante est : » , phrase
FIN
```

6. LES TABLEAUX

6.1. Les tableaux à une seul dimension

Imaginez que l'on veuille calculer la moyenne des notes d'une classe d'élèves. Pour l'instant on pourrait l'algorithme suivant :

```
    Variables somme, nbEleves, Note, i : Réels
    DEBUT
        somme ← 0
        Ecrire " Nombre d'eleves :"
        Lire nbEleves
        POUR i = 1 A nbEleves
            Ecrire " Note de l'eleve numero ", i , " : "
            Lire Note
            somme ← somme + Note
        FIN POUR
        Ecrire " La moyenne est de :", somme / nbEleves
    FIN
```

Si l'on veut toujours calculer la moyenne des notes d'une classe mais en gardant en mémoire toutes les notes des élèves pour d'éventuels calculs (par exemple calculer le nombre d'élèves qui ont des notes supérieurs à la moyenne). Dans ce cas il faudrait alors déclarer autant de variables qu'il y a d'étudiants. Donc, si l'on a 10 élèves il faut déclarer 10 variables et si l'on a N il faut déclarer N variables et c'est pas pratique. Ce qu'il faudrait c'est pouvoir par l'intermédiaire d'une seule variable stocker plusieurs valeurs de même type et c'est le rôle des tableaux.

Un **tableau** est un ensemble de valeurs de même type portant le même nom de variable. Chaque valeur du tableau est repérée par un nombre appelé **indice**.

Les tableaux c'est ce que l'on nomme un **type complexe** en opposition aux types de données simples vus précédemment. La déclaration d'un tableau sera via la syntaxe suivante dans la partie des déclarations :

Tableau *nom_tableau*(*nombre*) : **Type**

nom_tableau : désigne le nom du tableau

nombre : désigne le nombre d'éléments du tableau. On dit aussi sa taille
Type : c'est le type du tableau autrement dit le type de tous ces éléments

Exemples :

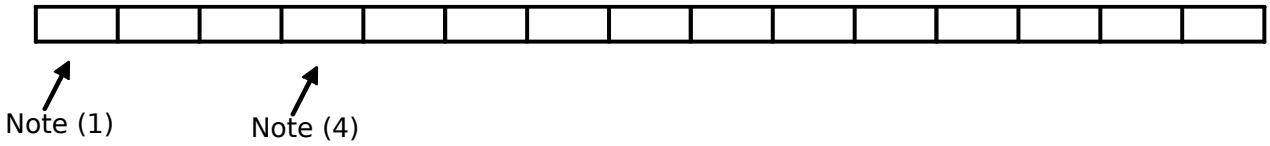
Tableau Note (20) : **Réel**

Note (20) est un tableau qui contient vingt valeurs réelles.

Tableau nom (10) , prenom (10) : **Chaîne**

Nom (10) et prenom (10) sont deux tableaux de 10 éléments de type chaîne.

Un tableau peut être représenté graphiquement par (exemple Note (15)) :



Si l'on veut accéder (en lecture ou en écriture) à la i ème valeur d'un tableau en utilisant la syntaxe suivante :

nom_tableau (*indice*)

Par exemple si X est un tableau de 10 entiers :

- X (2) ← - 5
  met la valeur -5 dans la 2 ème case du tableau
- En considérant le cas où *a* est une variable de type Entier, a ← X (2)
  met la valeur de la 2 ème case du tableau tab dans a, c'est- à- dire 5
- **Lire** X (1)
  met l'entier saisi par l'utilisateur dans la première case du tableau
- **Ecrire** X (1)
  affiche la valeur de la première case du tableau

Remarques :

- Un tableau possède un nombre maximal d'éléments défini lors de l'écriture de l'algorithme (les bornes sont des constantes explicites, par exemple 10, ou implicites, par exemple MAX). Ce nombre d'éléments ne peut être fonction d'une variable.
- La valeur d'un indice doit toujours :
  - être un nombre entier
  - être inférieure ou égale au nombre d'éléments du tableau

Exercices

1. Considérons les programmes suivants:

```
Tableau X (4) : Entier
DEBUT
X        (1) ← 12
X        (2) ← 5
       X (3) ← 8
X        (4) ← 20
FIN
```

```
Tableau voyelle (6) : Chaîne
DEBUT
voyelle        (1) ← « a »
voyelle        (2) ← « e »
       voyelle (3) ← « i »
voyelle        (4) ← « o »
```

voyelle (5) ← « u »
voyelle (6) ← « y »
FIN

Donner les représentations graphiques des tableaux X (4) et voyelle (6) après exécution de ces programmes.

2. Quel résultat fournira l'exécution de ce programme :

```
Variable i : Entier
Tableau C (6) : Entier
DEBUT
        POUR i = 1 A 6
Lire C (i)
FIN POUR
        POUR i = 1 A 6
C               (i) ← C (i) * C (i)
FIN POUR
        POUR i = 1 A 6
Ecrire C (i)
FIN POUR
FIN
```

Si on saisit successivement les valeurs : 2 , 5 , 3 , 10 , 4 , 2.

3. Que fournira l'exécution de ce programme :

```
Tableau suite (8) : Entier
Variable    i : Entier
DEBUT
Suite (1) ← 1
Suite (2) ← 1
        POUR i = 3 A 8
suite           (i) ← suite (i - 1) + suite (i - 2)
FIN POUR
        POUR i = 1 A 8
Ecrire suite (i)
FIN POUR
FIN
```

4. Soit T un tableau de vingt éléments de types entiers. Ecrire le programme qui permet de calculer la somme des éléments de ce tableau.

5. Soit T un tableau de N entiers. Ecrire l'algorithme qui détermine le plus grand élément de ce tableau.

6. Ecrire un programme qui permet de lire 100 notes et de déterminer le nombre de celles qui sont supérieures à la moyenne.

7. Soit T un tableau de N entiers. Ecrire l'algorithme qui détermine simultanément la position du plus grand élément et la position du plus petit élément du tableau.

8. Soit T un tableau de N réels. Ecrire le programme qui permet de calculer le nombre des occurrences d'un nombre X (c'est-à-dire combien de fois ce nombre X figure dans le tableau T).

9. On dispose des notes de 25 élèves ; chaque élève peut avoir une ou plusieurs notes mais toujours au moins une. Ecrire un programme permettant d'obtenir la moyenne de chaque élève lorsqu'on lui fournit les notes. On veut que les données et les résultats se présentent ainsi :

Notes de l'élève numéro 1
12
12
-1
Notes de l'élève numéro 2
……
Notes de l'élève numéro 25
15
-1
Moyennes
Elève numéro 1 : 11
……
Elève numéro 25 : 15
Moyenne de la classe : 12.3

Les parties italiques correspondent aux données tapées par l'utilisateur. La valeur -1 sert de critère de fin de notes pour chaque élève.

Solutions

1. La représentation graphique du tableau X (4) après exécution du premier programme est :

12 5	8 20		

La représentation graphique du tableau voyelle (4) après exécution du deuxième programme est :

a e	i o u y				

2. L'exécution du programme nous affichera successivement à l'écran :

```
      4
25
9
100
      16
4
```

3. L'exécution du programme nous affichera successivement à l'écran :

```
      1
1
2
3
5
8
13
21
```

4. Le programme est :

Variables i , somme : **ENTIERS**

```
        Tableau T (N) : ENTIER
        DEBUT
 somme ← 0
                POUR i = 1 A N
 somme                          ← somme + T (i)
 FIN POUR
 Ecrire « La somme de tous les éléments du tableau est : » , somme
FIN
```

5. Le programme est :

```
Variables       i , max : ENTIERS
        Tableau T (N) : ENTIER
        DEBUT
 max ← T (1)
i                ← 1
                REPETER
  i ← i + 1
 SI T (i)  > max ALORS
  max                           ← T (i)
 FIN SI
JUSUQ'A i = N
 Ecrire « La somme de tous les éléments du tableau est : » , somme
FIN
```

6. Le programme est :

```
Variables       i , somme , moyenne , nsup : Réels
        Tableau Note (100) : Réel
        DEBUT
 somme ← 0
                POUR i = 1 A 100
 Lire Note (i)
 somme                          ← somme + Note (i)
 FIN POUR
 Moyenne ← somme / 100
nsup                ← 0
                POUR i = 1 A 100
 SI Note (i) > moyenne ALORS
  nsup                          ← nsup + 1
 FIN SI
 FIN POUR
 Ecrire « Le nombre de notes supérieures à la moyenne est : » , nsup
FIN
```

7. Le programme est :

```
Variables       i , pmax , pmin : Entiers
        Tableau T (N) : Réel
        DEBUT
 max ← T (1)
 min ← T (1)
pmax                ← 1
 pmin ← 1
i                ← 1
                REPETER
  i ← i + 1
 SI T (i)  > max ALORS
```

```
max                          ← T (i)
pmax                         ← i
                FIN SI
SI T (i) < min ALORS
min                          ← T (i)
pmin                         ← i
                FIN SI
JUSUQ'A i = N
Ecrire « La position du plus grand élément du tableau est : » , pmax
Ecrire « La position du plus petit élément du tableau est : » , pmin
FIN
```

8. Le programme est :

```
Variables      X ,i,Compt : Réels
       Variable Compt :ENTIER
       Tableau T (N) : Réel
       DEBUT
Lire           X

            POUR i=1 JUSQU'A i=N

SI T (i) =X ALORS
Compt                        ← compt+1
FIN SI
FIN POUR
Ecrire « Le nombre d'occurrences de cet éléments du tableau est : » , compt
FIN
```

9. Le programme est :

```
       Variables i , note , nnote , snote , smoyenne , cmoyenne : Entiers
       Tableau moy (25) : Réel
       DEBUT
POUR           i = 1 A 25
Ecrire « Notes de l'élève numéro » , i
snote                        ← 0
nnote                        ← 0
REPETER
                             Lire note
SI note <> -1 ALORS
snote                                ← snote + note
nnote                                ←nnote + 1
FIN SI
                JUSQU'A note = -1
                moy (i) = snote / nnote
                smoyenne = smoyenne + moy (i)
FIN POUR
Ecrire « Moyennes »
POUR i = 1 A 25
Ecrire « Elève numéro » , i , « : » , moy (i)
FIN POUR
                cmoyenne = smoyenne / 25
Ecrire « Moyenne de la classe : » , cmoyenne
FIN
```

6.2. Les tableaux dynamiques

Il arrive fréquemment que l'on ne connaisse pas à l'avance le nombre d'éléments que devra comporter un tableau. Bien sûr, une solution consisterait à déclarer un tableau avec une taille très grande. Cela pourra avoir comme conséquence soit que cette taille ne nous nous suffira pas ou qu'une place mémoire immense sera réservée sans être utilisée.
Afin de surmonter ce problème on a la possibilité de déclarer le tableau sans préciser au départ son nombre d'éléments. Ce n'est que dans un second temps, au cours du programme, que l'on va fixer ce nombre via une instruction de re-dimensionnement : **Redim**.

Exemple :

On veut saisir des notes pour un calcul de moyenne, mais on ne sait pas combien il y aura de notes à saisir. Le début de l'algorithme sera quelque chose du genre :

```
Tableau    Notes () : Réel
Variable   nb en Entier
 DEBUT
  Ecrire          "Combien y a-t-il de notes à saisir ?"
  Lire            nb
  Redim           Notes(nb-1)
                  …
 FIN
```

Exercices

1. Insertion d'un élément dans un tableau

Soit T un tableau de N éléments. Ecrire un programme qui permet d'insérer un élément x à la position i du tableau T.

2. Suppression d'un élément du tableau

Soit T un tableau de N éléments. Ecrire un programme qui permet de supprimer un élément x du tableau.

Solutions

1. Le programme est :

```
Tableau T () : Entier
Variables i, x, j : Entier
DEBUT
 Ecrire « Donnez la dimension du tableau »
 Lire   N
 Redim T (N)
        POUR j = 1 A N
 Lire T (j)
        FIN POUR
 Ecrire  « Entrez le nombre à insérer »
 Lire x
        Ecrire « Entrez la position où insérer ce nombre »
 Lire i
        Redim T (N +1)
        j = N
TANT QUE j ≥ i
                T (j+1)  = T (j)
j               = j - 1
FIN TANT QUE
        T (i) = x
```

Dans ce programme on a travaillé avec un seul tableau dynamique. On peut aussi travailler avec le tableau T à dimension fixe et en définir un autre qui recevra tous les éléments de T plus l'élément à insérer. Le programme dans ce cas est :

```
Tableau T (N) : Entier
Tableau Tr (N+1) : Entier
Variables i , x , j , k : Entier
DEBUT
        POUR j = 1 A N
 Lire T (j)
        FIN POUR
 Ecrire      « Entrez le nombre à insérer »
 Lire x
        Ecrire « Entrez la position où insérer ce nombre »
 Lire i
        j = 1
        k = 1
TANT QUE k ≤ N + 1
SI k                    ≠ i ALORS
 Tr (k)                          ← T (j)
 j                          ← j + 1
                    SINON
 Tr (k) = x
FIN SI
                    k = k + 1
FIN TANT QUE
```

2. Le programme est :

```
Tableau T (N) : Entier
Tableau Tr () : Entier
Variables i , x , j : Entier
DEBUT
        POUR j = 1 A N
 Lire T (j)
        FIN POUR
 Ecrire      « Entrez le nombre à supprimer »
 Lire x
 j          ← 0
POUR i = 1 A            N
 SI T (i) ≠ x ALORS
                        j ← j + 1
  ReDim Tr (j)
                        Tr (j)  =  T (i)
 FIN SI
FIN POUR
```

Dans ce programme on a considéré deux tableaux, le tableau T à dimension fixe et le tableau Tr dynamique. Il est aussi possible de travailler avec un seul tableau dynamique.

```
Tableau T () : Entier
Variables x, j , k , N : Entiers
DEBUT
 Ecrire « Donnez la dimension du tableau »
 Lire       N
 Redim T (N)
```

```
        POUR j = 1 A N
    Lire T (j)
        FIN POUR
    Ecrire      « Entrez le nombre à supprimer »
    Lire x
        j = 1
TANT QUE j              ≤ N
 SI T (j)  = x ALORS
 POUR k = j A N - 1
  T                             (k) = T (k + 1)
 FIN POUR
 N                          ← N – 1
 ReDim T (N)
SINON
 j                          ← j + 1
 FIN SI
FIN TANT QUE
```

6.3. Les tableaux multidimensionnels

Nous avons vu qu'un tableau à une dimension correspond à une liste ordonnée de valeurs, repérée chacune par un indice.

Dans tous les langages de programmation, il est possible de définir des tableaux à deux dimensions (permettant par exemple de représenter des matrices). Ainsi, on pourra placer des valeurs dans un tableau à deux dimensions et cela consiste comme dans le cas des tableaux à une dimension à donner un nom à l'ensemble de ces valeurs. Chaque valeur est alors repérée par deux indices qui précise la position.

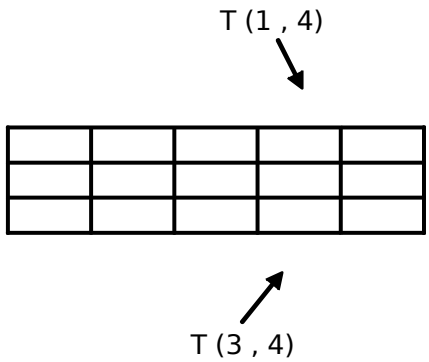
On déclare un tableau à deux dimensions de la façon suivante :

Tableau *nom_tableau* (*i*, *j*) : **Type**

nom_tableau : désigne le nom du tableau
i : désigne le nombre de lignes du tableau
j : désigne le nombre de colonnes du tableau
Type : représente le type des éléments du tableau

Exemple :

Soit T (3 , 5) un tableau d'entiers. On peut représenter graphiquement par :



T (1 , 4) et T(3 , 4) sont deux éléments du tableau. Entre parenthèse on trouve les valeurs des indices séparées par une virgule. Le premier sert à repérer le numéro de la ligne, le second le numéro de la colonne.

On accède en lecture ou en écriture à la valeur d'un élément d'un tableau à deux dimensions en utilisant la syntaxe suivante :

Nom_tableau (*i* , *j*)

Par exemple si T est défini par : **Tableau** T (3 , 2) : Réel

- T (2 , 1) ← -1.2
  met la valeur -1.2 dans la case 2,1 du tableau
- En considérant le cas où a est une variable de type Réel, a ← T (2 , 1)
  met -1.2 dans a

Par extension, on peut aussi définir des tableaux à n dimensions. Leur déclaration sera à l'image de celle des tableaux à deux dimensions, c'est- à- dire :

Tableau *nom_tableau* (*i , j , k , ….*) : **Type**

Par exemple :

Tableau X (10 , 9 , 5) : **Entier**

Ainsi que leur utilisation :

- X (2 , 1 , 3) ← 10
- a ← X (2 , 1 , 3)

Exercices

1. Considérons le programme suivant :

```
Tableau X (2 , 3) : Entier
Variables i , j , val : Entiers
DEBUT
val                ← 1
POUR i = 1 A 2
 POUR j = 1 A 3
  X (i , j)                ← val
  val                      ← val + 1
 FIN POUR
FIN POUR
POUR i = 1 A 2
 POUR j = 1 A 3
  Ecrire X (i , j)
 FIN POUR
FIN POUR
```

a. Que produit l'exécution de ce programme.

b. que produira ce programme si l'on remplace les derniers lignes par :

```
POUR          j = 1 A 3
POUR i = 1 A 2
 Ecrire X (i , j)
FIN POUR
FIN POUR
```

2. Quel résultat fournira ce programme :

```
Tableau X (4 , 2) : Entier
Variables k , m : Entiers
DEBUT
POUR k = 1 A 4
 POUR m = 1 A 2
  X (k , m)                ← k + m
 FIN POUR
FIN POUR
POUR k = 1 A 4
 POUR m = 1 A 2
  Ecrire X (k , m)
 FIN POUR
FIN POUR
```

3. Soit T un tableau à deux dimensions de vingt lignes et cinquante colonnes.
a. Ecrire un algorithme qui permet de calculer la somme de tous les éléments du tableau.
b. Ecrire l'algorithme qui permet de compter le nombre des éléments strictement positifs.
c. Ecrire l'algorithme permettant d'obtenir la somme des éléments positifs (spos) et la somme des éléments négatifs (sneg) de ce tableau.
d. Ecrire l'algorithme qui détermine la plus grande valeur des éléments du tableau.
e. Ecrire l'algorithme qui détermine simultanément l'élément le plus grand du tableau ainsi que sa position.

Solutions

1. L'exécution du programme donnera :
Les deux premières boucles du programme permettent de remplir le tableau. Ainsi la représentation graphique sera :

1 2 3		
4 5 6		

Les deux dernières boucles permettent d'afficher ces six éléments. Le résultat sera donc :

Question a

1
2
3
4
5
6

Question b

1
4
2
5
3
6

2. Les deux premières boucles de ce programme permettent de remplir le tableau. La représentation graphique sera ainsi :

2 3	
3 4	
4 5	
5 6	

Les dernières boucles permettent d'afficher les huit éléments du tableau. Le résultat est :

2
3
3
4
4
5
5
6

3. Soit T (20 , 50) un tableau de réels.
a. L'algorithme qui calcule la somme de tous les éléments du tableau est :

```
        Tableau T (20 , 50) : Réel
        Variables i , j : Entiers
Variable      som : Réel
        DEBUT
 som ← 0
POUR i = 1 A 20
 POUR j = 1 A 50
  som                          ← som + T (i , j)
 FIN POUR
FIN POUR
Ecrire « La somme de tous les éléments du tableau est : » , som
```

b. L'algorithme qui compte le nombre des éléments strictement positifs est :

```
        Tableau T (20 , 50) : Réel
        Variables i , j : Entiers
Variable      npos : Réel
        DEBUT
 npos ← 0
POUR i = 1 A 20
 POUR j = 1 A 50
  SI T (i , j) > 0 ALORS
   npos                                ← npos + 1
  FIN SI
 FIN POUR
FIN POUR
            Ecrire « Le nombre des éléments strictement positifs du tableau est : » , npos
```

c. L'algorithme permettant d'obtenir la somme des éléments positifs (spos) et la somme des éléments négatifs (sneg) de ce tableau est :

```
Tableau T (20 , 50) : Réel
        Variables i , j : Entiers
Variable      spos , sneg : Réel
        DEBUT
 spos ← 0
sneg              ← 0
POUR i = 1 A 20
 POUR j = 1 A 50
  SI T (i , j) > 0 ALORS
   spos                                ← spos + T (i , j)
  SINON
   sneg                                ← sneg + T (i , j)
  FIN SI
 FIN POUR
FIN POUR
Ecrire « La somme des éléments positifs du tableau est : » , spos
Ecrire « La somme des éléments négatifs du tableau est : » , sneg
```

d. L'algorithme qui détermine la plus grande valeur des éléments du tableau est :

```
Tableau T (20 , 50) : Réel
        Variables i , j : Entiers
Variable      max : Réel
        DEBUT
 max ← T (1 , 1)
POUR i = 1 A 20
```

```
POUR j = 1 A 50
 SI T (i , j) > max ALORS
  max                          ← T (i , j)
 FIN SI
 FIN POUR
FIN POUR
Ecrire « Le plus grand élément du tableau est : » , max
Ecrire « la position de l'élément i «= »,imax, «et j=» jmax
```

e. L'algorithme qui détermine simultanément l'élément le plus grand du tableau ainsi que sa position est :

```
Tableau T (20 , 50) : Réel
      Variables i , j , imax , jmax : Entiers
Variable      max : Réel
      DEBUT
max ← T (1 , 1)
POUR i = 1 A 20
 POUR j = 1 A 50
 SI T (i , j) > max ALORS
  max                          ← T (i , j)
  imax                         ← i
  jmax                         ← j
 FIN SI
 FIN POUR
FIN POUR
Ecrire « Le plus grand élément du tableau est : » , max
```

7. LES STRUCTURES

Imaginons que l'on veuille afficher les notes d'une classe d'élèves par ordre croissant avec les noms et prénoms de chaque élève. On va donc utiliser trois tableaux (pour stocker les noms, les prénoms et les notes). Lorsque l'on va trier le tableau des notes il faut aussi modifier l'ordre les tableaux qui contiennent les noms et prénoms. Mais cela multiplie le risque d'erreur. Il serait donc intéressant d'utiliser ce qu'on appelle **les structures**.

Les structures contrairement aux tableaux servent à rassembler au sein d'une seule entité un ensemble fini d'éléments de type éventuellement différents. C'est le deuxième type complexe disponible en algorithmique.

A la différence des tableaux, il n'existe pas par défaut de type structure c'est-à-dire qu'on ne peut pas déclarer une variable de type structure. Ce qu'on peut faire c'est de construire toujours un nouveau type basé sur une structure et après on déclare des variables sur ce nouveau type.

La syntaxe de construction d'un type basé sur une structure est :

```
TYPE          NomDuType = STRUCTURE
 attribut1          : Type
 attribut2          : Type
 .                  . .
 attributn          : Type
FIN          STRUCTURE
```

Le type d'un attribut peut être :

- Un type simple
- Un type complexe
  - Un tableau
  - Un type basé sur une structure

Exemple :

```
TYPE        Etudiant = STRUCTURE
            nom : chaîne
prenom :          chaîne
note :          Réel
FIN         STRUCTURE
```

Dans cet exemple on a construit un type *Etudiant* basé sur une structure. Cette structure a trois attributs (on dit aussi champ) : nom, prenom et note.

```
TYPE Date       = STRUCTURE
jour :          Entier
mois :          Entier
annee           : Entier
FIN         STRUCTURE
```

Dans ce deuxième exemple, on a construit un type *Date* basé sur une structure. Cette structure a aussi trois attributs : jour, mois et annee.

Après on peut déclarer des variables basé sur ce type. Par exemple :

Variable Etud : **Etudiant**

Donc Etud est une variable de type Etudiant.

Il est possible de déclarer un tableau d'éléments de ce type Etudiant par exemple. On pourra écrire donc :

Tableau Etud (20) : **Etudiant**

Etud (1) représente le premier étudiant.

Maintenant, pour accéder aux attributs d'une variable dont le type est basé sur une structure on suffixe le nom de la variable d'un point « . » suivi du nom de l'attribut. Par exemple, dans notre cas pour affecter le nom "Dinar" à notre premier étudiant, on utilisera le code suivant :

Etud (1).nom = « Dianr »

Exercices

1. Définissez la structure « Stagiaire » constituée des champs suivants :

Champ	Type
Nom	Chaîne
Prénom	Chaîne
Datenais	Structure

Le champ « Datenais » est aussi une structure dont les champs sont :

Champ	Type
Jour	Entier
Mois	Entier
Année	Entier

Ecrivez ensuite l'algorithme qui permet de lire et après afficher le nom, prénom et date de naissance d'un stagiaire.

2. On souhaite gérer les notes d'un étudiant. Pour cela on va définir la structure « Etudiant » dont les champs sont :

Champ	Type
Nom	Chaîne
Prénom	Chaîne
Note	Tableau de 3 éléments
Moyenne	Réel

Ecrire l'algorithme qui permet de lire les informations d'un étudiant (nom, prénom et notes), de calculer sa moyenne et d'afficher à la fin un message sous la forme suivante :

« La moyenne de l'étudiant Dinar Youssef est : 12.45 »

où « Dinar » et « Youssef » sont les noms et prénoms lus et 12.45 est la moyenne calculée.

3. Modifier l'algorithme de l'exercice précédent de façon que l'on puisse gérer les notes de 50 étudiants.

Solutions

1. L'algorithme est :

```
TYPE Date = STRUCTURE
 Jour        : Entier
 Mois        : Entier
 Annee       : Entier
FIN STRUCTURE
TYPE Stagiaire = STRUCTURE
 Nom         : chaîne
 Prenom      : chaîne
 Datenais    : Date
FIN STRUCTURE
Variable stag : Stagiaire
DEBUT
Ecrire « Entrez les information du stagiaire »
Ecrire « Entrez le nom »
Lire stag.Nom
Ecrire « Entrez le prénom »
Lire stag.Prenom
Ecrire « Entrez le jour de naissance »
Lire stag.Date.Jour
Ecrire « Entrez le mois de naissance »
Lire stag.Date.Mois
Ecrire « Entrez l'année de naissance »
Lire stag.Date.Annee
Ecrire « Le nom du stagiaire est : » , stag.Nom
Ecrire « Son prénom est : » , stag.Prenom
Ecrire «Sa date de naissance est :», stag.Date.Jour , «/», stag.Date.Mois, «/», stag.Date.Annee
```

2. L'algorithme est :

```
TYPE Etudiant = STRUCTURE
 Nom         : Chaîne
 Prenom      : Chaîne
        Note (3) : Réel
        Moyenne : Réel
FIN STRUCTURE
Variable i : Entier
Variable som : Réel
Variable etud Etudiant
DEBUT
```

Ecrire « Entrez les information de l'étudiant »
Ecrire « Entrez le nom »
Lire etud.Nom
Ecrire « Entrez le prénom »
Lire etud.Prenom
Ecrire « Entrez la première note »
Lire Etud.Note (1)
Ecrire « Entrez la deuxième note »
Lire etud.Note (2)
Ecrire « Entrez la troisième note »
Lire etud.Note (3)
som ← 0
POUR i = 1 **A** 3
som ← etud.Note (i)
FIN POUR
etud.Moyenne = som / 3
Ecrire «La moyenne de l'étudiant » , etud.Nom , « » , etud.Prenom , « est : » , etud.Moyenne

3. L'algorithme est :

```
TYPE Etudiant = STRUCTURE
 Nom          : Chaîne
 Prenom       : Chaîne
 Note(3)      : Réel
        Moyenne : Réel
FIN STRUCTURE
Variable i , j : Entier
Variable som : Réel
Variable etud(50) : Etudiant
DEBUT
Ecrire « Entrez les information des étudiants »
POUR j = 1 A 50
Ecrire « Entrez le nom »
Lire etud(j).Nom
Ecrire « Entrez le prénom »
Lire etud(j).Prenom
Ecrire « Entrez la première note »
Lire etud(j).Note (1)
Ecrire « Entrez la deuxième note »
Lire etud(j).Note (2)
Ecrire « Entrez la troisième note »
Lire etud(j).Note (3)
som ← 0
POUR i = 1 A 3
som ← etud(j).Note (i)
FIN POUR
etud (j).Moyenne = som / 3
FIN POUR
POUR j = 1 A 50
 Ecrire «La moyenne de l'étudiant » , etud(j).Nom , «  » , etud(j).Prenom , « est : » ,
 etud(j).Moyenne
FIN POUR
```

8. LES FONCTIONS ET PROCEDURES

En programmation, donc en algorithmique, il est possible de décomposer le programme qui résout un problème en des sous-programmes qui résolvent des sous parties du problème initial. Ceci permettra d'améliorer la conception du programme et ainsi sa lisibilité.
L'utilisation des sous-programmes s'avère utile aussi dans le cas où on constate qu'une suite d'actions se répète plusieurs fois.
Il existe deux types de sous-programmes les fonctions et les procédures. Un sous- programme est obligatoirement caractérisé par un nom (un identifiant) unique.
Le nom d'un sous-programme comme le nom d'une variable doit :

- Contenir que des lettres et des chiffres
- Commencer obligatoirement par une lettre

Le programme qui utilise un sous- programme est appelé **programme appelant**. Un sous-programme peut être invoqué n'importe où dans le programme appelant en faisant référence à son nom.
Un programme ainsi doit suivre la structure suivante :

Définition des constantes
Définition des types
Déclaration des variables
Définition des sous- programmes

DEBUT

Instructions du programme principal

FIN

8.1. Les fonctions

Une fonction est un sous-programme qui retourne un **seul** résultat. Pour définir une fonction on utilise la syntaxe suivante :

FONCTION nom_fonction (Argument1 : **Type** , Argument2 : **Type** ,….) : **Type**
Déclarations
DEBUT
Instructions de la fonction
nom_fonction ← Valeur renvoyée
FIN

On constate que la déclaration d'une fonction revient à lui préciser un nom, un type ainsi qu'une liste d'arguments.
Un argument (appelé **paramètre formel**) d'un sous- programme est une variable locale particulière qui est associée à une variable ou constante du programme appelant. Puisque qu'un argument est une variable locale, il admet un type.
Lorsque le programme appelant appelle le sous-programme il doit indiqué la variable ou la constante de même type, qui est associée au paramètre.
Par exemple, le sous-programme *sqr* permet de calculer la racine carrée d'un réel. Ce sous-programme admet un seul paramètre de type réel positif.
Le programme qui utilise *sqr* doit donner le réel positif dont il veut calculer la racine carrée, cela peut être :

- une variable, par exemple *a*
- une constante, par exemple 5.25

Les arguments d'une fonction sont en nombre fixe (≥ 0).
Une fonction possède un seul type, qui est le type de la valeur retournée qui est affecté au nom de la fonction.
Une fois la fonction définie, il est possible (en fonction des besoins) à tout endroit du programme appelant de faire appel à elle en utilisant simplement son nom suivi des arguments entre parenthèses.
Les parenthèses sont toujours présentes même lorsqu'il n'y a pas de paramètre.
Les arguments spécifiés lors de l'appel de la fonction doivent être en même nombre que dans la déclaration de la fonction et des types prévus. Dans le programme appelant ces arguments sont appelés **paramètres effectifs**.

La valeur ainsi renvoyée par la fonction peut être utilisée dans n'importante quelle expression compatible avec son type.

Exemple :

```
        FONCTION Calcul (x : Réel , y : Réel , z : Réel) : Réel
 Variable a : Entier
        DEBUT
 a                ← 3
 Calcul                ← (x + y + z)  * a
        FIN
```

Ça c'est un exemple de déclaration de fonction. Cette fonction appelée « Calcul » est de type réel et elle admet trois arguments de type réel.
Maintenant voici un exemple de programme complet :

```
 FONCTION Calcul (x : Réel , y : Réel , z : Réel) : Réel
 Variable a : Entier
        DEBUT
 a                ← 3
 Calcul                ← (x + y + z)  * a
        FIN
 Variables      i , j , k , b : Réels
 DEBUT
  Lire i
 Lire j
 Lire k
 b                ← Calcul (i , j , k) + 2
 Ecrire b
 FIN
```

Dans ce programme on fait appel a une fonction. Sa valeur est utilisée dans une expression.

Exercice

1. Définir la fonction « Somme » qu'on lui passe deux valeurs de type entier et qui renvoie comme valeur la somme des valeurs reçues.

2. Définir la fonction « Absolue » qui renvoie la valeur absolue d'une valeur qu'on lui passe comme paramètre.

3. Définir la fonction « Inverse » qui renvoie l'inverse d'une valeur qu'on lui passe comme paramètre.

4. Définir la fonction « Max » qui renvoie le maximum de deux valeurs.

5. Ecrivez un programme qui lit trois scores et qui utilise la fonction définie dans l'exercice précédent pour déterminer le meilleur score et l'afficher après.

Solutions

1. La définition de la fonction « Somme » est :

```
        FONCTION Somme (x : Réel , y : Réel ) : Réel
        DEBUT
 Somme                ← x + y
        FIN
```

2. La définition de la fonction « Absolue » est :

```
        FONCTION Absolue (x : Réel) : Réel
        DEBUT
 SI x > 0 ALORS
  Absolue                        ← x
```

```
SINON
 Absolue                    ← -1 * x
FIN SI
      FIN
```

3. La définition de la fonction « Inverse » est :

```
      FONCTION Inverse (x : Réel) : Réel
      DEBUT
SI x ≠ 0 ALORS
 Inevrse                    ← 1 /x
FIN SI
      FIN
```

4. La définition de la fonction « Max » est :

```
      FONCTION Max (x : Réel , y Réel) : Réel
      DEBUT
SI x > y ALORS
 Max                        ← x
SINON
 Max                        ← y
FIN SI
      FIN
```

5. Le prgramme est :

```
      FONCTION Max (x : Réel , y Réel) : Réel
      DEBUT
SI x > y ALORS
 Max                        ← x
SINON
 Max                        ← y
FIN SI
      FIN
      Variables score1 , score2 , score3 , meil_score : Réels
      DEBUT
            Ecrire « Entrez les troix scores : »
            Lire score1
            Lire score2
            Lire score3
            meil_score ← Max (Max (score1 , score2) , score3)
            Ecrire « Le meilleur score est : » , meil_score
      FIN
```

8.2. Les variables locales et globales

La **portée** d'une variable est l'ensemble des sous- programmes où cette variable est connue c'est-à-dire que les instructions de ces sous-programmes peuvent utiliser cette variable.

Une variable définie au niveau du programme principal (problème appelant) est appelée **variable globale**. Sa portée est totale : **tout** sous-programme du programme principal peut utiliser cette variable.

Cependant, une variable définie au sein d'un sous-programme est appelée **variable locale**. La portée d'une variable locale est uniquement le sous-programme qui la déclare.

Remarque :

Lorsque le nom d'une variable locale est identique à une variable globale, la variable globale est localement masquée. Dans ce sous-programme la variable globale devient inaccessible.

Exemple

Soit le programme suivant :

```
        Fonction Surface (a : Réel) : Réel
        Variables valeur , resultat : Réels
        DEBUT
 valeur          ← 3.14
 resulat         ← valeur * a
 Surface          ← resultat
        FIN
Variable    rayon : Réel
DEBUT
 Ecrire          « Entrez le rayon du cercle : »
Lire rayon
Ecrire « La surface de cette cercle est : » , Surface (rayon)
        FIN
```

Les variables *valeur* et *resultat* déclarées dans la fonction *Surface* sont locales.
Considérons presque le même programme sauf que la variable *valeur* est déclarée maintenant dans le programme appelant.

```
Fonction Surface (a : Réel) : Réel
        Variables resultat : Réels
        DEBUT
 resulat         ← valeur * a
 Surface          ← resultat
        FIN
Variable    valeur , rayon : Réel
DEBUT
 valeur ← 3.14
 Ecrire          « Entrez le rayon du cercle : »
Lire rayon
Ecrire « La surface de cette cercle est : » , Surface (rayon)
        FIN
```

Dans ce deuxième programme seule la variable *resultat* est locale. Tandis que la variable *valeur* est devenue globale. On peut par conséquent accéder à sa valeur dans la fonction *Surface*

8.3. Les passage de paramètres

Il existe deux types d'association (que l'on nomme **passage de paramètre)** entre le(s) paramètre(s) du sous-programme (fonction ou procédure) et variable(s) du programme appelant :

- **Passage par valeur**
- **Passage par adresse**

Dans le cas où l'on choisit pour un paramètre effectif un passage par valeur, la valeur de ce paramètre effectif ne change pas même si lors de l'appel du sous-programme la valeur du paramètre formel correspondant change. On peut dire que dans ce cas le paramètre effectif et le paramètre formel ont font deux variables différents qui ont seulement la même valeur. C'est la type de passage par défaut.
Dans le cas où l'on choisit pour un paramètre effectif un passage par adresse, la valeur de ce paramètre effectif change si lors de l'appel du sous-programme la valeur du paramètre formel correspondant change. On peut dire que dans ce cas le paramètre effectif et le paramètre formel ont font deux variables qui ont le même adresse (par conséquent valeur). Pour préciser qu'il s'agit d'un passage par adresse, il faut soulignés les paramètres concernés lors de la définition du sous-programme.

Exemple

Considérons les deux programmes suivants :

Programme 1

```
        Fonction Calcul (a : Réel) : Réel
```

```
        DEBUT
Calcul          ← a * 2
a               ← a – 1
        FIN
        Variable x : Réel
        DEBUT
x               ← 3
Ecrire Calcul (x)
Ecrire x
        FIN
```

Programme 2

```
        Fonction Calcul (a : Réel) : Réel
        DEBUT
Calcul          ← a * 2
a               ← a – 1
        FIN
        Variable x : Réel
        DEBUT
x               ← 3
Ecrire Calcul (x)
Ecrire x
        FIN
```

Dans le premier programme on a un passage de paramètre par valeur et dans le deuxième on a un passage de parametrespar adresse. Le premier programme affichera le résultat suivant :
6
3
car même si la valeur de a change celle de x non.
Tandis que le deuxième programme il affichera :
6
2
la valeur de x changera car celle de a a changée.

8.4. Les procédures

Les procédures sont des sous- programmes qui ne retournent **aucun** résultat. Elles admettent comme les fonctions des paramètres.
On déclare une procédure de la façon suivante :

```
PROCEDURE nom_procedure (Argument1 : Type , Argument2 : Type ,….)
Déclarations
        DEBUT
                Instructions de la procédure
        FIN
```

Et on appelle une procédure comme une fonction, en indiquant son nom suivi des paramètres entre parenthèses.

Exercice

1. Ecrire une procédure qui reçoit la longueur et la largeur d'une surface et qui affiche la valeur de la surface. Donnez à cette procédure le nom « Surface ».
2. Ecrire une procédure qui permet d'échanger les valeurs de deux variables. Appelez cette procédure « Echanger ».
3. On dispose d'une phrase dont les mots sont séparer par des point virgules. Ecrivez une procédure qui permet de remplacer les points virgules par des espaces. On suppose qu'on dispose des fonctions suivantes :

- Longueur : permet de calculer la longueur d'une chaîne de caractères.

Utilisation : Longueur (chaîne)

- Extraire : permet d'extraire une partie (ou la totalité) d'une chaîne.

Utilisation : Extraire (chaîne , position_debut, longueur)

Paramètre : chaîne de laquelle on fait l'extraction
position_debut la position à partir de laquelle va commencer l'extraction
longueur désigne la longueur de la chaîne qui va être extraite.

Solutions

1. Le programme qui définit la procédure « Surface » est :

```
PROCEDURE Surface (longueur : Réel , largeur : Réel)
        Variable s : Réel
DEBUT
s                   ← longueur * largeur
Ecrire « La surface obtenu est : » , s
        FIN
```

2. Le programme qui définit la procédure « Echanger » est :

```
PROCEDURE Echanger (x : Réel , y : Réel)
        Variable z : Réel
DEBUT
z                   ← x
x                   ← y
y                   ← z
        FIN
```

3. Le programme de cette procédure est :

```
PROCEDURE Changer (chaine____ : Chaîne)
        Variables i , l : Entier
        Variables caract , schaine : Chaîne
DEBUT
l                   ← Longueur (chaine)
                  schaine = « «  »
POUR i = 1 A l
 caract                          ← Extraire (chaine , i , 1)
 SI caract = « ; » ALORS
  caract = «  »
  FIN                       SI
 schaine                          ← schaine & caract
 FIN                  POUR
chaine                  ← schaine
FIN
        Variable chaine : Chaîne
        Variable i : Entier
DEBUT
chaine              ← « bonjour;tout;le;monde »
changer              (chaine)
Ecrire chaine
        FIN
```

9. LA RECHERCHE BINAIRE (RECHERCHE DICHOTOMIQUE)

Soit T un tableau de N éléments ordonnés et x un élément de même type que les éléments de T. Il s'agit d'examiner la présence de x dans T. Comme le tableau est ordonné, il satisfait la spécification suivante :

$\forall i \in [1, N-1]$ T (i) $\leq$ T (i+1)

Au lieu de faire une recherche linéaire [2], on décompose le tableau en deux sous-tableaux T1 et T2 et trois cas peuvent se produire :

- x est trouvé la recherche est terminé.
- la recherche continue dans T1.
- la recherche continue dans T2.

Exemple

Soit le tableau suivant T :

1	2	3	4	5	6	7	8
3	7	8	8	11	15	20	24

Vous constatez que le tableau T est déjà ordonné. On va voir comment s'applique la méthode de recherche binaire pour rechercher si x = 20 existe dans le tableau T.

On divise l'intervalle des indices [1,8] en deux intervalles [1,4] et [5,8]. On obtient deux tableaux T1 et T2.

1	2	3	4	5	6	7	8
3	7	8	8	11	15	20	24

x $\notin$ T1 | x $\in$ T2

La recherche va continuer dans le tableau T2 puisque x (20) est plus supérieur que le plus grand élément de T1. Donc l'intervalle de recherche devient [5,8] et on va le diviser à son tour en deux intervalles [5,6] et [7,8].

5	6	7	8
11	15	20	24

x $\notin$ T1 | x $\in$ T2

De même, la recherche va continuer dans T2. L'intervalle de recherche devient [7,8]. On le divise en deux intervalles [7,7] et [8,8].

Finalement, x est trouvé.

7	8
20	24

x $\in$ T1 | x $\notin$ T2

Le programme de la recherche dichotomique est le suivant :

```
Tableau T(N) : Entier
Variables inf , sup , milieu , x : Entier
        Variable Trouve : Booléen
 DEBUT
 Trouve              ← Faux
 inf = 1
 sup                 ← N
        TANT QUE inf ≤ sup ET Trouve = Faux
```

[2] Parcourir tout le tableau T du premier élément au dernier.

```
milieu          ← (inf + sup) Div 2     ' Div est la division entière
SI T(milieu) = x ALORS
 Trouve                 ← Vrai
SINON
 SI T(milieu) < x alors
  inf                   ← inf + 1
 SINON
  sup                   ← milieu –1
 FIN                SI
 FIN SI
FIN      TANTQUE
      SI Trouve = Vrai ALORS
Ecrire « L'élément » , x , « existe dans T »
      SINON
Ecrire « L'élément » , x , « n'existe pas dans T »
      FIN SI
FIN
```

10. LES ALGORITHMES DE TRI

Trier les éléments d'un tableau revient à ordonner tous ces éléments selon un ordre croissant ou décroissant.
Soit T un tableau de N éléments muni d'une relation d'ordre ≤. Trier ce tableau c'est construire un algorithme qui devra satisfaire à la spécification suivante :

$$\forall i \in [1, N-1] \quad T(i) \leq T(i+1)$$

Dans ce paragraphe on va traiter plusieurs algorithmes de tri : tri par sélection, tri par bulle, tri par comptage, tri par insertion, tri par shell.

10.1. Tri par bulle

Principe

Ce tri permet de faire remonter petit à petit un élément trop grand vers la fin du tableau en comparant les éléments deux à deux.
Si un élément d'indice *j* est supérieur à un élément d'indice i+1 on les échange et on continue avec le suivant. Lorsqu'on atteint le fin du tableau on repart du début. On s'arrête lorsque tous les éléments du tableau sont bien placés c'est-à-dire qu'on aura aucun changement d'éléments à effectuer.

Algorithme

```
      Tableau T(N) : Entiers
      Variables j , nc : Entiers
DEBUT
 REPETER
  nc                    ← 0
  POUR j = 1 A (N-1)
   SI T(j) > T(j+1) ALORS
    nc                          ← nc +1
    z                           ← T(j)
    T(j)                        ← T(j+1)
    T(j+1)                      ← z
    FIN SI
   FIN                  POUR
 JUSUQU'A nc = 0
FIN
```

Exemple

Soit le tableau suivant :

52	10	1	25	

Boucle REPETER	Etat du tableau			Valeur de nc
Itération 1	10 52 1 25			3
	10 1 52 25			
	10 1 25 52			
Itération 2	1 10 25 52			1
	1 10 25 52			
	1 10 25 52			
Itération 3	1 10 25 52 0			

10.2. Tri par sélection

Principe : Soit T un tableau de N éléments. On cherche le plus petit élément du tableau et on le place à la première position. Après, on cherche le plus petit dans les (N-1) qui reste et on le place en deuxième position et ainsi de suite.

52 10 1 25 62 3 8 55 3 23
1 52 10 25 62 3 8 55 3 23
1 3 52 10 25 62 8 55 3 23
1 3 3 52 10 25 62 8 55 23
1 3 3 8 52 10 25 62 55 23
1 3 3 8 10 52 25 62 55 23
1 3 3 8 10 23 52 25 62 55
1 3 3 8 10 23 25 52 62 55
1 3 3 8 10 23 25 52 62 55
1 3 3 8 10 23 25 52 55 62

Algorithme :

```
        POUR i ALLANT DE 1 A 9
FAIRE
 Petit                  Å  TAB (i)
        POUR j ALLANT DE i A 10
FAIRE
 Si                 (TAB (j) < petit) ALORS petit Å  TAB (j) ; position Å  j FSI
FinPour
        POUR j ALLANT DE position A i+1 PAS –1
FAIRE
TAB(j)          Å  TAB (j-1) ;
FinPour
TAB          (i) Å  petit ;
FinPour
```

5.4.2- Le tri bulle :

```
FAIRE
Inversion          Å  FAUX
        POUR i ALLANT DE 1 A 9
FAIRE
                Si (TAB (i) > TAB (i+1))
         ALORS
          Tampon                Å  TAB (i) ;
          TAB                (i) Å  TAB (i+1) ;
```

```
 TAB                (i+1) Å  Tampon
 Inversion                Å  VRAI
FSI
FinPour
JUSQUA (inversion = FAUX) ;
```

5.4.3- Le tri par permutation :

```
        POUR i ALLANT DE 1 A 9
FAIRE
                SI (TAB (i+1) < TAB (i))
 ALORS
  Abouger                        Å  TAB (i+1)
  j                        Å  1 ;
                          TanQue ((j < i) ET (TAB (j) < TAB (i+1)))
  Faire                          j Å  j+1
  FTQ
                          POUR k ALLANT DE i+1 A j+1 PAS –1
  Faire
   TAB (k)                                Å  TAB (k-1)
  FinPour
  TAB                          (j) Å  abouger
 FSI
Fin          Pour
```

5.4.4- Le tri par comptage :

```
        POUR i ALLAN DE 1 A 10
FAIRE
 RES                (i) Å  0
 NB                (i) Å  0
                POUR j ALLANT DE 1 A 10
 FAIRE
  Si                          TAB (j) < TAB (i) ALORS NB (i) Å  NB (i) + 1 FSI
 FinPour
FinPour
        POUR i ALLANT de 1 A 10
FAIRE
 j                Å  NB (i)
                TantQue RES (j) <> 0
 Faire
  j                          Å  j+1
 FTQ
 RES                (j) Å  TAB (i)
FinPour
```

5.4.5- Le tri alphabétique :

```
        POUR nbmots ALLANT DE 1 A 10
Faire
                AFFICHER « Entrer le mot suivant »
 LIRE                MOT
```

```
Pluspetit               Å VRAI
j              Å 1
               TANTQUE (pluspetit ET (j < nbmots))
Faire
 k                     Å 1
                       TantQue ((MOTS (j, k) = MOT (k)) ET k <= 20)
 Faire
  k                          Å k+1
 FTQ
                       Si (MOTS (j, k) < MOT (k))
 ALORS
  j                          Å j+1
 SINON
  Pluspetit                        Å FAUX
 FSI
FTQ
               Si (j < nbmots)
ALORS
                       POUR i ALLANT DE nbmots A j+1 PAS –1
 Faire
  POUR                             k ALLANT DE 1 A 20
  Faire
                MOTS (i, k)                Å MOTS (i-1, k)
  FinPour
        FinPour
       FSI
              POUR k ALLANT DE 1 A 20
       Faire
        MOTS                (j, k) Å MOT (k)
       FinPour
      FinPour
      POUR i ALLANT DE 1 A nbmots
      Faire
       AFFICHER          MOTS (i)
      FinPour
```

Partie 02 : JAVA

1. Présentation

Java est un langage de programmation à usage général, évolué et orienté objet dont la syntaxe est proche du C. Il existe 2 types de programmes en Java : les applets et les applications. Une application autonome (stand alone program) est une application qui s'exécute sous le contrôle direct du système d'exploitation. Une applet est une application qui est chargée par un navigateur et qui est exécutée sous le contrôle d'un plug in de ce dernier.

Ce chapitre contient plusieurs sections :

- Les caractéristiques
- Bref historique de Java
- Les différentes éditions et versions de Java
- Un rapide tour d'horizon des API et de quelques outils
- Les différences entre Java et JavaScript
- L'installation du JDK

1.1. Les caractéristiques

Java possède un certain nombre de caractéristiques qui ont largement contribué à son énorme succès :

Java est interprété	le source est compilé en pseudo code ou byte code puis exécuté par un interpréteur Java : la Java Virtual Machine (JVM). Ce concept est à la base du slogan de Sun pour Java : WORA (Write Once, Run Anywhere : écrire une fois, exécuter partout). En effet, le byte code, s'il ne contient pas de code spécifique à une plate-forme particulière peut être exécuté et obtenir quasiment les même résultats sur toutes les machines disposant d'une JVM.
Java est indépendant de toute plate-forme	il n'y a pas de compilation spécifique pour chaque plate forme. Le code reste indépendant de la machine sur laquelle il s'exécute. Il est possible d'exécuter des programmes Java sur tous les environnements qui possèdent une Java Virtual Machine. Cette indépendance est assurée au niveau du code source grâce à Unicode et au niveau du byte code.
Java est orienté objet	comme la plupart des langages récents, Java est orienté objet. Chaque fichier source contient la définition d'une ou plusieurs classes qui sont utilisées les unes avec les autres pour former une application. Java n'est pas complètement objet car il définit des types primitifs (entier, caractère, flottant, booléen,...).
Java est simple	le choix de ses auteurs a été d'abandonner des éléments mal compris ou mal exploités des autres langages tels que la notion de pointeurs (pour éviter les incidents en manipulant directement la mémoire), l'héritage multiple et la surcharge des opérateurs, ...
Java est fortement typé	toutes les variables sont typées et il n'existe pas de conversion automatique qui risquerait une perte de données. Si une telle conversion doit être réalisée, le développeur doit obligatoirement utiliser un cast ou une méthode statique fournie en standard pour la réaliser.
Java assure la gestion de la mémoire	l'allocation de la mémoire pour un objet est automatique à sa création et Java récupère automatiquement la mémoire inutilisée grâce au garbage collector qui restitue les zones de mémoire laissées libres suite à la destruction des objets.

Java est sûr	la sécurité fait partie intégrante du système d'exécution et du compilateur. Un programme Java planté ne menace pas le système d'exploitation. Il ne peut pas y avoir d'accès direct à la mémoire. L'accès au disque dur est réglementé dans une applet. Les applets fonctionnant sur le Web sous soumises aux restrictions suivantes dans la version 1.0 de Java : * aucun programme ne peut ouvrir, lire, écrire ou effacer un fichier sur le système de l'utilisateur * aucun programme ne peut lancer un autre programme sur le système de l'utilisateur * toute fenêtre créée par le programme est clairement identifiée comme étant une fenêtre Java, ce qui interdit par exemple la création d'une fausse fenêtre demandant un mot de passe * les programmes ne peuvent pas se connecter à d'autres sites Web que celui dont ils proviennent.
Java est économe	le pseudo code a une taille relativement petite car les bibliothèques de classes requises ne sont liées qu'à l'exécution.
Java est multitâche	il permet l'utilisation de threads qui sont des unités d'exécution isolées. La JVM, elle même, utilise plusieurs threads.

Les programmes Java exécutés localement sont des applications, ceux qui tournent sur des pages Web sont des applets.

Les principales différences entre une applet et une application sont :

- les applets n'ont pas de méthode main() : la méthode main() est appelée par la machine virtuelle pour exécuter une application.
- les applets ne peuvent pas être testées avec l'interpréteur mais doivent être intégrées à une page HTML, elle même visualisée avec un navigateur disposant d'un polug in sachant gérer les applets Java, ou testées avec l'applet viewer.

1.2. Bref historique de Java

Les principaux événément de la vie de Java sont les suivants :

Année	Evénements
1995	mai : premier lancement commercial
1996	janvier : JDK 1.0
1996	septembre : lancement du JDC
1997	février : JDK 1.1
1998	décembre : lancement de J2SE et du JCP
1999	décembre : lancement J2EE
2000	mai : J2SE 1.3
2002 J2SE	1.4
2004 J2SE	5.0

1.3. Les différentes éditions et versions de Java

Sun fourni gratuitement un ensemble d'outils et d'API pour permettre le développement de programmes avec Java. Ce kit, nommé JDK, est librement téléchargeable sur le site web de Sun http://java.sun.com ou par FTP

Le JRE (Java Runtime Environment) contient uniquement l'environnement d'exécution de programmes Java. Le JDK contient lui même le JRE. Le JRE seul doit être installé sur les machines où des applications Java doivent être exécutées.

Depuis sa version 1.2, Java a été renommé Java 2. Les numéros de version 1.2 et 2 désignent donc la même version. Le JDK a été renommé J2SDK (Java 2 Software Development Kit) mais la dénomination JDK reste encore largement utilisée, à tel point que la dénomination JDK est reprise dans la version 5.0. Le JRE a été renommé J2RE (Java 2 Runtime Environment).

Trois éditions de Java existent :

- J2ME : Java 2 Micro Edition
- J2SE : Java 2 Standard Edition
- J2EE : Java 2 Entreprise Edition

Sun fourni le JDK, à partir de la version 1.2, sous les plate-formes Windows, Solaris et Linux.

La version 1.3 de Java est désignée sous le nom Java 2 version 1.3.

La version 1.5 de Java est désignée officiellement sous le nom J2SE version 5.0.

La documentation au format HTML des API de Java est fournie séparément. Malgré sa taille imposante, cette documentation est indispensable pour obtenir des informations complètes sur toutes les classes fournies. Le tableau ci dessous résume la taille des différents composants selon leur version pour la plate-forme Windows.

	Version 1.0	Version 1.1	Version 1.2	Version 1.3	Version 1.4	Version 5.0
JDK compressé		8,6 Mo	20 Mo	30 Mo	47 Mo	44 Mo
JDK installé				53 Mo	59 Mo	
JRE compressé			12 Mo	7 Mo		14 Mo
JRE installé				35 Mo	40 Mo	
Documentation compressée			16 Mo	21 Mo	30 Mo	43,5 Mo
Documentation décompressée			83 Mo	106 Mo	144 Mo	223 Mo

1.3.1. Java 1.0

Cette première version est lancée officiellement en mai 1995.

1.3.2. Java 1.1

Cette version du JDK est annoncée officiellement en mars 1997. Elle apporte de nombreuses améliorations et d'importantes fonctionnalités nouvelles dont :

- les Java beans
- les fichiers JAR
- RMI pour les objets distribués
- la sérialisation
- l'introspection
- JDBC pour l'accès aux données
- les classes internes
- l'internationalisation
- un nouveau modèle de sécurité permettant notamment de signer les applets
- JNI pour l'appele de méthodes natives
- ...

1.3.3. Java 1.2

Cette version du JDK est lancée fin 1998. Elle apporte de nombreuses améliorations et d'importantes fonctionnalités nouvelles dont :

- un nouveau modèle de sécurité basé sur les policy
- les JFC sont incluses dans le JDK (Swing, Java2D, accessibility, drag & drop ...)
- JDBC 2.0
- les collections
- support de CORBA
- un compilateur JIT est inclus dans le JDK
- de nouveaux format audio sont supportés
- ...

Java 2 se décline en 3 éditions différentes qui regroupent des APIs par domaine d'application :

- **Java 2 Micro Edition (J2ME)** : contient le nécessaire pour développer des applications capable de fonctionner dans des environnements limités tels que les assistants personnels (PDA), les téléphones portables ou les systèmes de navigation embarqués
- **Java 2 Standard Edition (J2SE)** : contient le nécessaire pour développer des applications et des applets. Cette édition reprend le JDK 1.0 et 1.1.
- **Java 2 Enterprise Edition (J2EE)** : contient un ensemble de plusieurs API permettant le développement d'applications destinées aux entreprises tel que JDBC pour l'accès aux bases de données, EJB pour développer des composants orientés métiers, Servlet / JSP pour générer des pages HTML dynamiques, ... Cette édition necessite le J2SE pour fonctionner.

Le but de ces trois éditions est de proposer une solution reposant sur Java quelque soit le type de développement à réaliser.

1.3.4. J2SE 1.3

Cette version du JDK apporte de nombreuses améliorations notamment sur les performances et des fonctionnalités nouvelles dont :

- JNDI est inclus dans le JDK
- hotspot est inclus dans la JVM
- ...

La rapidité d'exécution a été grandement améliorée dans cette version.

1.3.5. J2SE 1.4 (nom de code Merlin)

Cette version du JDK, lancée début 2002, apporte de nombreuses améliorations notamment sur les performances et des fonctionnalités nouvelles dont :

- JAXP est inclus dans le JDK pour le support de XML
- JDBC version 3.0
- new I/O API pour compléter la gestion des entrée/sortie
- logging API pour la gestion des logs applicatives
- une API pour utiliser les expressions régulières
- une api pour gérer les préférences utilisateurs
- JAAS est inclus dans le JDK pour l'authentification
- un ensemble d'API pour utiliser la cryptographie
- l'outil Java WebStart
- ...

Cette version ajoute un nouveau mot clé au langage pour utiliser les assertions : assert.

1.3.6. J2SE 5.0 (nom de code Tiger)

La version 1.5 du J2SE est spécifiée par le JCP sous la JSR 176. Elle devrait intégrer un certain nombre de JSR dans le but de simplifier les développements en Java.

Ces évolutions sont réparties dans une quinzaine de JSR qui seront intégrées dans la version 1.5 de Java.

JSR-003 JMX	Management API
JSR-013 Decimal	Arithmetic
JSR-014 Generic	Types
JSR-028 SASL	
JSR-114	JDBC API Rowsets
JSR-133	New Memory Model and thread
JSR-163 Profiling	API
JSR-166 Concurrency	Utilities
JSR-174	Monitoring and Management for the JVM
JSR-175 Metadata	facility
JSR-199 Compiler	APIs
JSR-200	Network Transfer Format for Java Archives
JSR-201	Four Language Updates
JSR-204 Unicode	Surrogates
JSR-206	JAXP 1.3

La version 1.5 de Java est désignée officiellement sous le nom J2SE version 5.0.

1.3.7. Les futures versions de Java

Version	Nom de code	Date de sortie
1.4	Merlin	2001
1.4.1 Hopper 2002		
1.4.2 Mantis 2003		
1.5 Tiger 2004		
1.5.1 Dragonfly ???		
1.6 Mustang ???		

1.3.8. Le résumé des différentes versions

	Java 1.0	Java 1.1	Java 1.2	J2SE 1.3	J2SE 1.4	J2SE 5.0
Nombre de de packages	8	23	59	76	135	166
Nombre de classes	201	503	1520	1840	2990	3280

1.3.9. Les extensions du JDK

Sun fourni un certains nombres d'API supplémentaires qui ne sont pas initialement fournies en standard dans le JDK. Ces API sont intégrées au fur et à mesure de l'évolution de Java.

Extension	Description
JNDI	Java Naming and directory interface Cet API permet d'unifier l'accès à des ressources. Elle est intégrée a Java 1.3
Java mail	Cette API permet de gérer des emails. Elle est intégrée a la plateforme J2EE.
Java 3D	Cette API permet de mettre en oeuvre des graphismes en 3 dimensions
Java Media	Cette API permet d'utiliser des composants multimédia
Java Servlets	Cette API permet de créer des servlets (composants serveurs). Elle est intégrée a la plateforme J2EE.
Java Help	Cette API permet de créer des aides en ligne pour les applications
Jini	Cette API permet d'utiliser Java avec des appareils qui ne sont pas des ordinateurs
JAXP	Cette API permet le parsing et le traitement de document XML. Elle est intégré a Java 1.4

Cette liste n'est pas exhaustive.

1.4. Un rapide tour d'horizon des API et de quelques outils

La communauté Java est très productive car elle regroupe :

- Sun, le fondateur de Java
- le JCP (Java Community Process) : c'est le processus de traitement des évolutions de Java dirigé par Sun. Chaque évolution est traitée dans une JSR (Java Specification Request) par un groupe de travail constitué de différents acteurs du monde Java
- des acteurs commerciaux dont tous les plus grands acteurs du monde informatique excepté Microsoft

- la communauté libre qui produit un très grand nombre d'API et d'outils pour Java

Ainsi l'ensemble des API et des outils utilisables est énorme et évolue très rapidement. Les tableaux ci dessous tentent de recenser les principaux par thème.

J2SE 1.4

	Java Bean	RMI	IO	Applet
	Reflexion Collection		Logging	AWT
	Net (réseau)	Preferences	Security	JFC
	Internationalisation	Exp régulière		Swing

Les outils de Sun

	Jar Javadoc Java	Web Start	JWSDK	

Les outils libres (les plus connus)

	Jakarta Tomcat	Jakarta Ant	JBoss	Apache Axis
	JUnit	Eclipse		

Les autres API

Données	Web Entreprise		XML	Divers
JDBC JDO	Servlets JSP JSTL Jave Server Faces	Java Mail JNDI EJB JMS JMX JTA RMI-IIOP Java IDL JINI JXTA	JAXP SAX DOM JAXM JAXR JAX-RPC SAAJ JAXB	JAI JAAS JCA JCE Java Help JMF JSSE Java speech Java 3D

Les API de la communauté open source

Données	Web Entreprise	XML	Divers	
OJB Cator hibernate	Jakarta Struts Webmacro Expresso Barracuda Turbine		Apache Xerces Apache Xalan Apache Axis JDOM DOM4J	Jakarta Log4j Jakarta regexp

1.5. Les différences entre Java et JavaScript

Il ne faut pas confondre Java et JavaScript. JavaScript est un langage développé par Netscape Communications.

La syntaxe des deux langages est très proche car elles dérivent toutes les deux du C++.

Il existe de nombreuses differences entre les deux langages :

	Java	Javascript
Auteur	Développé par Sun Microsystems	Développé par Netscape Communications
Format	Compilé sous forme de byte-code	Interprété
Stockage	Applet téléchargé comme un élément de la page web	Source inséré dans la page web
Utilisation	Utilisable pour développer tous les types d'applications	Utilisable uniquement pour "dynamiser" les pages web
Execution	Executé dans la JVM du navigateur	Executé par le navigateur
POO Orienté	objets	Manipule des objets mais ne permet pas d'en définir
Typage	Fortement typé	Pas de contrôle de type
Compléxité du code	Code relativement complexe	Code simple

1.6. L'installation du JDK

Le JDK et la documentation sont librement téléchargeable sur le site web de Sun : http://java.sun.com

1.6.1. L'installation de la version 1.3 DU JDK de Sun sous Windows 9x

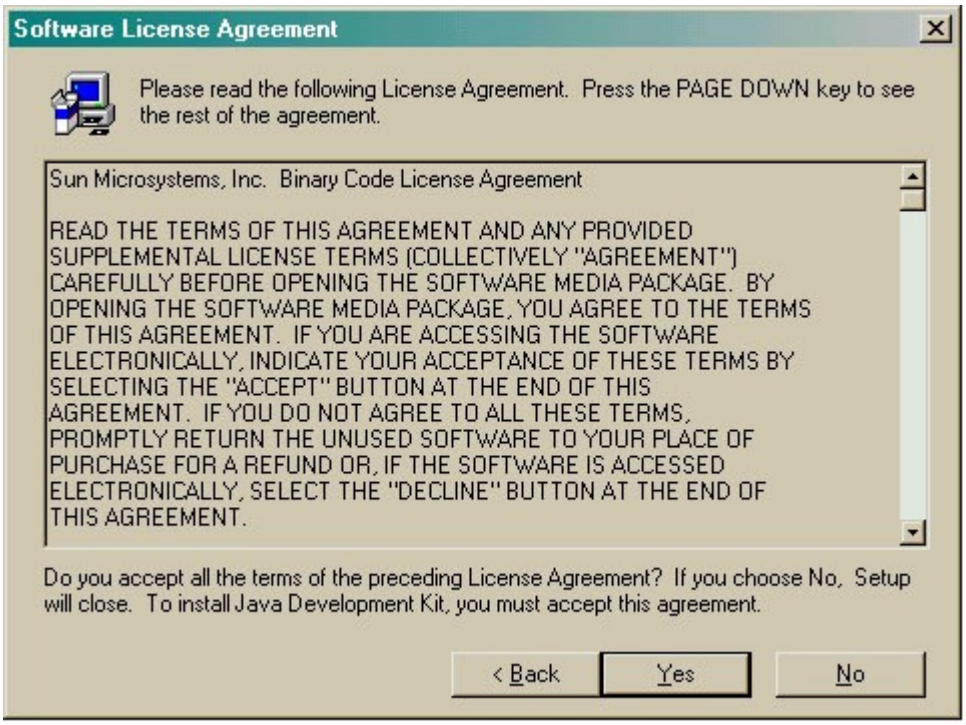
Pour installer le JDK 1.3 sous Windows 9x, il suffit de télécharger et d'exécuter le programme : j2sdk1_3_0-win.exe

Le programme commence par désarchiver les composants.

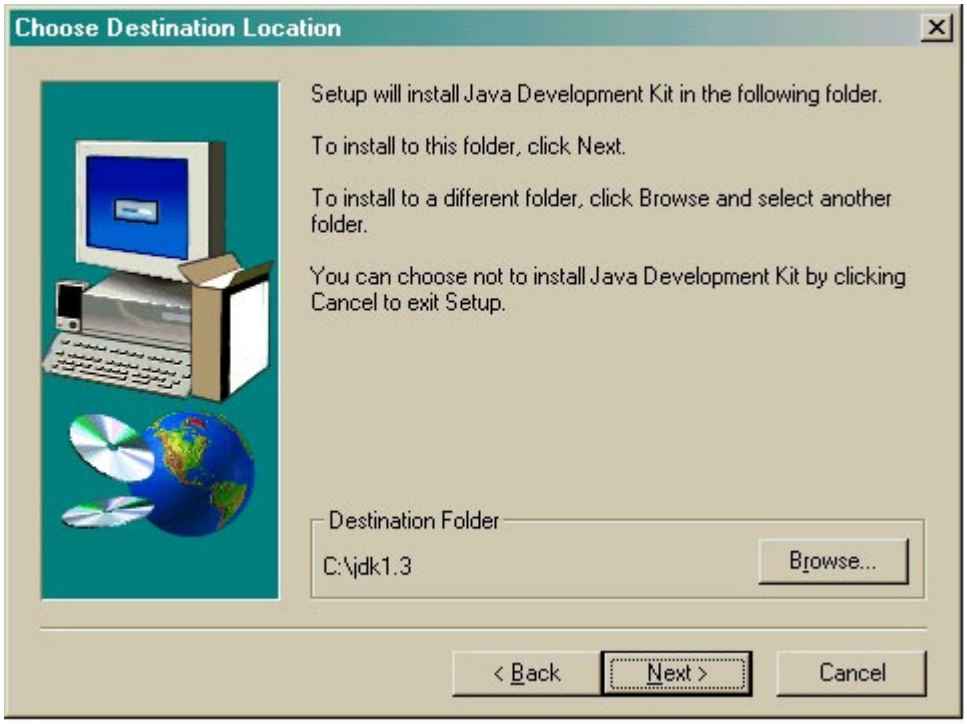
Le programme utilise InstallShield pour guider et réaliser l'installation.

L'installation vous souhaite la bienvenue et vous donne quelques informations d'usage.

L'installation vous demande ensuite de lire et d'approuver les termes de la licence d'utilisation.



L'installation vous demande le répertoire dans lequel le JDK va être installé. Le répertoire proposé par défaut est pertinent car il est simple.



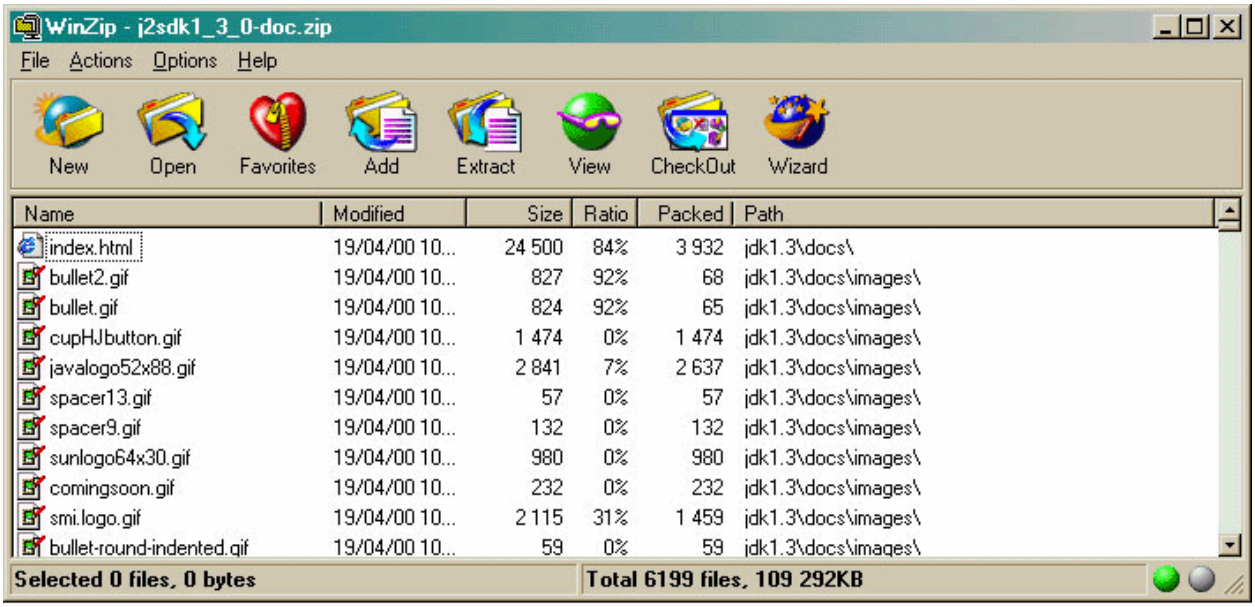
L'installation vous demande les composants à installer :

- Program Files est obligatoire pour une première installation
- Les interfaces natives ne sont utiles que pour réaliser des appels de code natif dans les programmes Java
- Les démos sont utiles car ils fournissent quelques exemples
- les sources contiennent les sources de la plupart des classes Java écrites en Java. Attention à l'espace disque nécessaire à cet élément

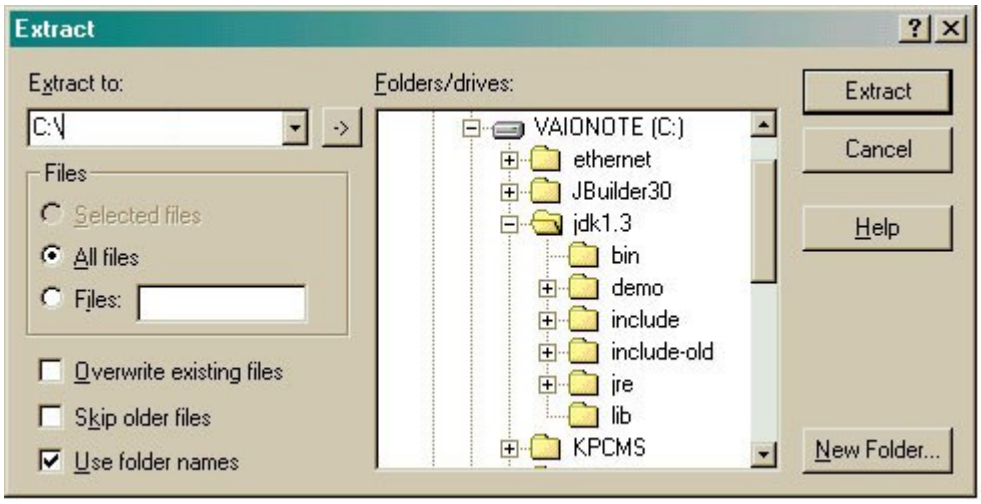
L'installation se poursuit par la copie des fichiers et la configuration du JRE.

1.6.2. L'installation de la documentation de Java 1.3 sous Windows

L'archive contient la documentation sous forme d'arborescence dont la racine est jdk1.3\docs.



Si le répertoire par défaut a été utilisé lors de l'installation, il suffit de décompresser l'archive à la racine du disque C:\.



Il peut être pratique de désarchiver le fichier dans un sous répertoire, ce qui permet de reunir plusieurs versions de la documentation.

1.6.3. La configuration des variables système sous Windows 9x

Pour un bon fonctionnement du JDK, il est recommandé de paramétrer correctement deux variables systèmes : la variable PATH qui définit les chemins de recherche des exécutables et la variable CLASSPATH qui définit les chemins de rechercher des classes et bibliothèques Java.

Pour configurer la variable PATH, il suffit d'ajouter à la fin du fichier autoexec.bat :

Exemple :
SET PATH=%PATH%;C:\JDK1.3\BIN

Attention : si une version antérieure du JDK était déjà présente, la variable PATH doit déjà contenir un chemin vers les utilitaires du JDK. Il faut alors modifier ce chemin sinon c'est l'ancienne version qui sera utilisée. Pour vérifier la version du JDK utilisée, il suffit de saisir la commande java -version dans une fenêtre DOS.

La variable CLASSPATH est aussi définie dans le fichier autoexec.bat. Il suffit d'ajouter une ligne ou de modifier la ligne existante définissant cette variable.

Exemple :
SET CLASSPATH=C:\JAVA\DEV;

Dans un environnement de développement, il est pratique d'ajouter le . qui désigne le répertoire courant dans le CLASSPATH surtout lorsque l'on n'utilise pas d'outils de type IDE. Attention toutefois, cette pratique est fortement déconseillée dans un environnement de production pour ne pas poser des problèmes de sécurité.

Il faudra ajouter par la suite les chemins d'accès aux différents packages requis par les développements afin de les faciliter.

Pour que ces modifications prennent effet dans le système, il faut redemarrer Windows ou exécuter ces deux instructions sur une ligne de commande DOS.

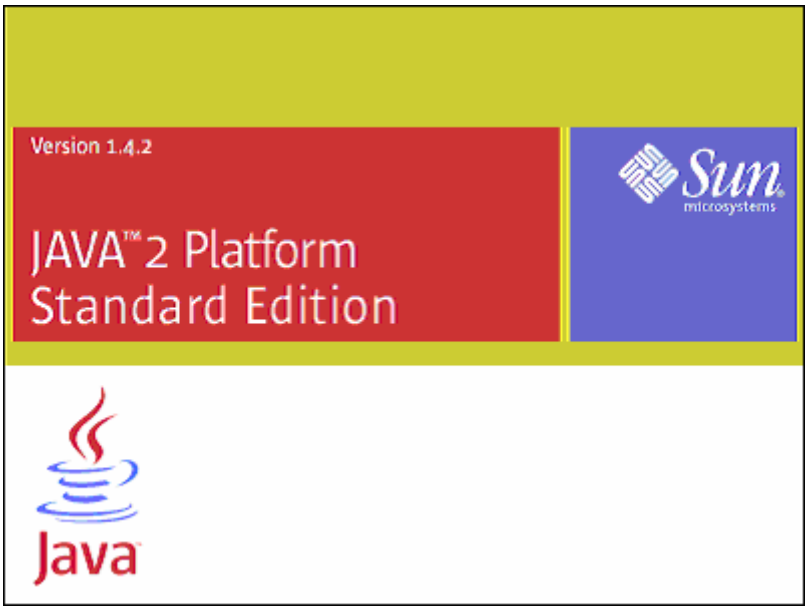
1.6.4. Les éléments du JDK 1.3 sous Windows

Le répertoire dans lequel a été installé le JDK contient plusieurs répertoires. Les répertoires donnés ci-après sont ceux utilisés en ayant gardé le répertoire par défaut lors de l'installation.

Répertoire	Contenu
C:\jdk1.3	Le répertoire d'installation contient deux fichiers intéressants : le fichier readme.html qui fourni quelques informations et des liens web et le fichier src.jar qui contient le source Jßava de nombreuses classes. Ce dernier fichier n'est présent que si l'option correspondante a été cochée lors de l'installation.
C:\jdk1.3\bin	Ce répertoire contient les exécutables : le compilateur javac, l'interpréteur java, le débuggeur jdb et d'une façon générale tous les outils du JDK.
C:\jdk1.3\demo	Ce répertoire n'est présent que si l'option nécessaire a été cochée lors de l'installation. Il contient des applications et des applets avec leur code source.
C:\jdk1.3\docs	Ce répertoire n'est présent que si la documentation a été décompressée.
C:\jdk1.3\include et C:\jdk1.3\include-old	Ces répertoires ne sont présents que si les options nécessaires ont été cochées lors de l'installation. Il contient des fichiers d'en-tête C (fichier avec l'extension .H) qui permettent de faire interagir du code Java avec du code natif
C:\jdk1.3\jre	Ce répertoire contient le JRE : il regroupe le nécessaire à l'exécution des applications notamment le fichier rt.jar qui regroupe les API. Depuis la version 1.3, le JRE contient deux machines virtuelles : la JVM classique et la JVM utilisant la technologie Hot spot. Cette dernière est bien plus rapide et c'est elle qui est utilisée par défaut. Les éléments qui composent le JRE sont séparés dans les répertoires bin et lib selon leur nature.
C:\jdk1.3\lib	Ce répertoire ne contient plus que quelques bibliothèques notamment le fichier tools.jar. Avec le JDK 1.1 ce répertoire contenait le fichier de la bibliothèque standard. Ce fichier est maintenant dans le répertoire JRE.

1.6.5. L'installation de la version 1.4.2 du JDK de Sun sous Windows

Télécharger sur le site java.sun.com et exécuter le fichier j2sdk-1_4_2_03-windows-i586-p.exe.



Un assistant permet de configurer l'installation au travers de plusieurs étapes :

- La page d'acceptation de la licence (« Licence agreement ») s'affiche
- Lire la licence et si vous l'acceptez, cliquer sur le bouton radio « I accept the terms in the licence agreement », puis cliquez sur le bouton « Next »
- La page de sélection des composants à installer (« Custom setup ») s'affiche, modifiez les composants à installer si nécessaire puis cliquez sur le bouton « Next »
- La page de sélection des plug in pour navigateur (« Browser registration ») permet de sélectionner les navigateurs pour lesquels le plug in Java sera installé, sélectionner ou non le ou les navigateurs détecté, puis cliquez sur le bouton « Install »
- L'installation s'opère en fonction des informations fournies précédemment
- La page de fin s'affiche, cliquez sur le bouton « Finish »

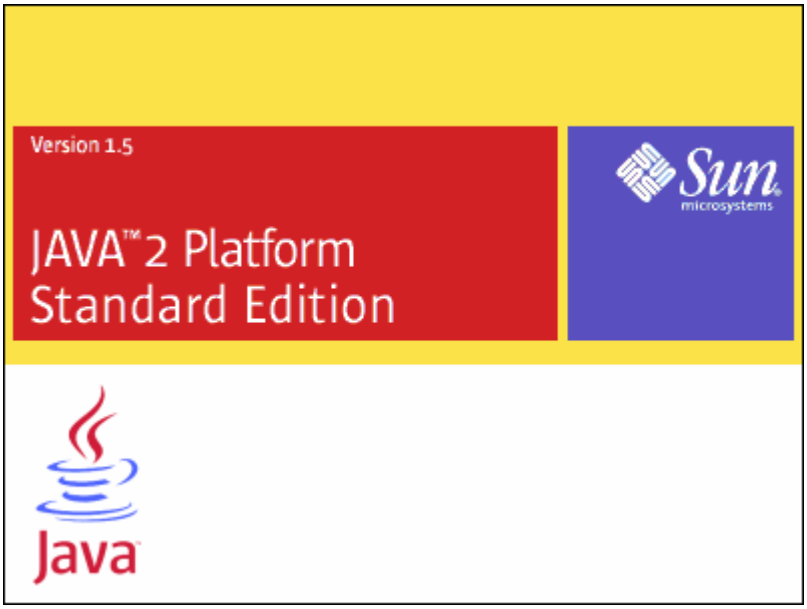
Même si ce n'est pas obligatoire pour fonctionner, il est particulièrement utile de configurer deux variables systèmes : PATH et CLASSPATH.

Dans la variable PATH, il est pratique de rajouter le chemin du répertoire bin du JDK installé pour éviter à chaque appel des commandes du JDK d'avoir à saisir leur chemin absolu.

Dans la variable CLASSPATH, il est pratique de rajouter les répertoires et les fichiers .jar qui peuvent être nécessaire lors des phases de compilation ou d'exécution, pour éviter d'avoir à les préciser à chaque fois.

1.6.6. L'installation de la version 1.5 beta 1 du JDK de Sun sous Windows

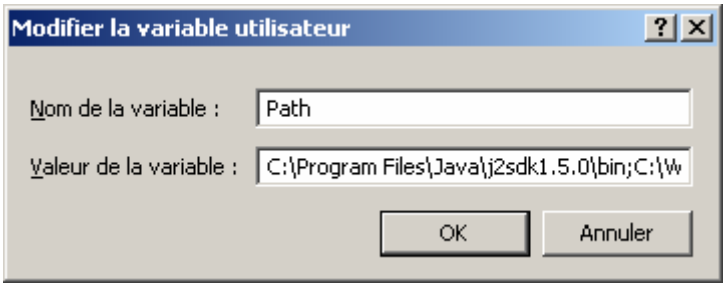
Il faut télécharger sur le site de Sun et exécuter le fichier j2sdk-1_5_0-beta-windows-i586.exe



Un assistant guide l'utilisateur pour l'installation de l'outil.

- Sur la page « Licence Agreement », il faut lire la licence et si vous l'acceptez, cochez le bouton radio « I accept the terms in the licence agreement » et cliquez sur le bouton « Next »
- Sur la page « Custom Setup », il est possible de sélectionner/désélectionner les éléments à installer. Cliquez simplement sur le bouton « Next ».
- La page « Browser registration » permet de sélectionner les plug-ins des navigateurs qui seront installés. Cliquez sur le bouton « Install »
- Les fichiers sont copiés.
- La page « InstallShield Wizard Completed » s'affichage à la fin de l'installation. Cliquez sur « Finish ».

Pour faciliter l'utilisation des outils du J2SE SDK, il faut ajouter le chemin du répertoire bin contenant ces outils dans la variable Path du système.



Il est aussi utile de définir la variable d'environnement JAVA_HOME avec comme valeur le chemin d'installation du SDK.

2. Les techniques de base de programmation en Java

N'importe quel éditeur de texte peut être utilisé pour éditer un fichier source Java.

Il est nécessaire de compiler le source pour le transformer en J-code ou byte-code Java qui sera lui exécuté par la machine virtuelle.

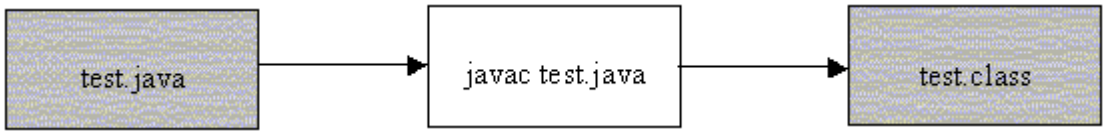
Il est préférable de définir une classe par fichier. Le nom de la classe publique et le fichier qui la contient doivent être identiques.

Pour être compilé, le programme doit être enregistré au format de caractères Unicode : une conversion automatique est faite par le JDK si nécessaire.

Ce chapitre contient plusieurs sections :

- La compilation d'un code source : cette section présente la compilation d'un fichier source.
- L'exécution d'un programme et d'une applet : cette section présente l'exécution d'un programme et d'une applet.

2.1. La compilation d'un code source



Pour compiler un fichier source il suffit d'invoquer la commande javac avec le nom du fichier source avec son extension .java

javac NomFichier.java

Le nom du fichier doit correspondre au nom de la classe principale en respectant la casse même si le système d'exploitation n'y est pas sensible

Suite à la compilation, le pseudo code Java est enregistré sous le nom *NomFichier.class*

2.2. L'exécution d'un programme et d'une applet

2.2.1. L'exécution d'un programme

Une classe ne peut être exécutée que si elle contient une méthode main() correctement définie.

Pour exécuter un fichier contenant du byte-code il suffit d'invoquer la commande java avec le nom du fichier source sans son extension .class

java NomFichier

3. La syntaxe et les éléments de bases de Java

Ce chapitre contient plusieurs sections :

- Les règles de base : cette section présente les règles syntaxiques de base de Java.
- Les identificateurs : cette section présente les règles de composition des identificateurs.
- Les commentaires : cette section présente les différentes formes de commentaires de Java.

- La déclaration et l'utilisation de variables : cette section présente la déclaration des variables, les types élémentaires, les formats des type élémentaires, l'initialisation des variables, l'affectation et les comparaisons.
- Les opérations arithmétiques : cette section présente les opérateurs arithmétique sur les entiers et les flottants et les opérateurs d'incrémentation et de décrémentation.
- La priorité des opérateurs : cette section présente la priorité des opérateurs.
- Les structures de contrôles : cette section présente les instructions permettant la réalisation de boucles, de branchements conditionnels et de débranchements.
- Les tableaux : cette section présente la déclaration, l'initialisation explicite et le parcours d'un tableau
- Les conversions de types : cette section présente la conversion de types élémentaires.
- La manipulation des chaines de caractères : cette section présente la définition et la manipulation de chaîne de caractères (addition, comparaison, changement de la casse ...).

3.1. Les règles de base

Java est sensible à la casse.

Les blocs de code sont encadrés par des accolades. Chaque instruction se termine par un caractère ';' (point virgule).

Une instruction peut tenir sur plusieurs lignes

exemple
char

code

=

'D';
L'indentation est ignorée du compilateur mais elle permet une meilleure compréhension du code par le programmeur.

3.2. Les identificateurs

Chaque objet, classe, programme ou variable est associer à un nom : l'identificateur qui peut se composer de tous les caractères alphanumériques et des caractères _ et $. Le premier caractère doit être une lettre, le caractère de soulignement ou le signe dollar.

Rappel : Java est sensible à la casse.

Un identificateur ne peut pas appartenir à la liste des mots réservé du langage Java :

abstract	assert (JDK 1.4)	boolean	break	byte
case catch char			class const	
continue default	do double else			
extends false final	finally float			
for goto if			implements	import
instanceof int		interface	long native	
new null package			private protected	

public return short			static super	
switch synchronized this throw throws				
transient true		try void volatile		
while				

3.3. Les commentaires

Ils ne sont pas pris en compte par le compilateur donc ils ne sont pas inclus dans le pseudo code. Ils ne se terminent pas par un ;.

Il existe trois types de commentaire en Java :

Type de commentaires	Exemple
commentaire abrégé	// commentaire sur une seule ligne int N=1; // déclaration du compteur
commentaire multiligne	/* commentaires ligne 1 commentaires ligne 2 */
commentaire de documentation automatique	/** * commentaire de la methode * @param val la valeur a traiter * @since 1.0 * @return Rien * @deprecated Utiliser la nouvelle methode XXX */

3.4. La déclaration et l'utilisation de variables

3.4.1. La déclaration de variables

Une variable possède un nom, un type et une valeur. La déclaration d'une variable doit donc contenir deux choses : un nom et le type de données qu'elle peut contenir. Une variable est utilisable dans le bloc ou elle est définie.

La déclaration d'une variable permet de réserver la mémoire pour en stocker la valeur.

Le type d'une variable peut être :

- soit un type élémentaire dit aussi type primitif déclaré sous la forme **type_élémentaire variable**;
- soit une classe déclarée sous la forme **classe variable** ;

exemple
long nombre;
int compteur;
String chaine;
Rappel : les noms de variables en Java peuvent commencer par un lettre, par le caractère de soulignement ou par le signe dollar. Le reste du nom peut comporter des lettres ou des nombres mais jamais d'espaces.

Il est possible de définir plusieurs variables de même type en séparant chacune d'elles par une virgule.

Exemple
int jour, mois, annee ;
Java est un langage à typage rigoureux qui ne possède pas de transtypage automatique lorsque ce transtypage risque de conduire à une perte d'information.

Pour les objets, il est nécessaire en plus de la déclaration de la variable de créer un objet avant de pouvoir l'utiliser. Il faut réserver de la mémoire pour la création d'un objet (remarque : un tableau est un objet en Java) avec l'instruction **new**. La libération de la mémoire se fait automatiquement grâce au garbage collector.

Exemple
MaClasse instance; // déclaration de l'objet
instance = new maClasse(); // création de l'objet
OU MaClasse instance = new MaClasse(); // déclaration et création de l'objet
Exemple
int[] nombre = new int[10];
Il est possible en une seule instruction de faire la déclaration et l'affectation d'une valeur à une variable ou plusieurs variables.

Exemple
int i=3 , j=4 ;

3.4.2. Les types élémentaires

Les types élémentaires ont une taille identique quelque soit la plate-forme d'exécution : c'est un des éléments qui permet à Java d'être indépendant de la plate-forme sur lequel le code s'exécute.

Type	Désignation	Longueur	Valeurs	Commentaires
boolean	valeur logique : true ou false	1 bit	true ou false	pas de conversion possible vers un autre type
byte	octet signé	8 bits	-128 à 127	
short	entier court signé	16 bits	-32768 à 32767	
char	caractère Unicode	16 bits	\u0000 à \uFFFF	entouré de cotes simples dans du code Java
int	entier signé	32 bits	-2147483648 à 2147483647	
float	virgule flottante simple précision (IEEE754)	32 bits	1.401e-045 à 3.40282e+038	
double	virgule flottante double précision (IEEE754)	64 bits	2.22507e-308 à 1.79769e+308	
long entier	long	64 bits	-9223372036854775808 à 9223372036854775807	

Les types élémentaires commencent tous par une minuscule.

3.4.3. Le format des types élémentaires

Le format des nombres entiers :

Les types byte, short, int et long peuvent être codés en décimal, hexadécimal ou octal. Pour un nombre hexadécimal, il suffit de préfixer sa valeur par 0x. Pour un nombre octal, le nombre doit commencer par un zéro. Le suffixe l ou L permet de spécifier que c'est un entier long.

Le format des nombres décimaux :

Les types float et double stockent des nombres flottants : pour être reconnus comme tel ils doivent posséder soit un point, un exposant ou l'un des suffixes f, F, d, D. Il est possible de préciser des nombres qui n'ont pas le partie entière ou décimale.

Exemple
float pi = 3.141f;
double v = 3d
float f = +.1f , d = 1e10f;
Par défaut un littéral est de type double : pour définir un float il faut le suffixer par la lettre f ou F.

Exemple
double w = 1.1;


Attention : float pi = 3.141; // erreur à la compilation

Le format des caractères :

Un caractère est codé sur 16 bis car il est conforme à la norme Unicode. Il doit être entouré par des apostrophes. Une valeur de type char peut être considérée comme un entier non négatif de 0 à 65535. Cependant la conversion implicite par affectation n'est pas possible.

Java
```
/* test sur les caractères */
class test1 {
   public static void main (String args[]) {
      char code = 'D';
      int index = code - 'A';
      System.out.println("index = " + index);
   }
}
```

3.4.4. L'initialisation des variables

Exemple
int nombre; // déclaration
nombre = 100; //initialisation
OU int nombre = 100; //déclaration et initialisation
En Java, toute variable appartenant à un objet (définie comme étant un attribut de l'objet) est initialisée avec une valeur par défaut en accord avec son type au moment de la creation. Cette initialisation ne s'applique pas aux variables locales des méthodes de la classe.

Les valeurs par défaut lors de l'initialisation automatique des variables d'instances sont :

Type	Valeur par défaut
boolean	false
byte, short, int, long	0
float, double	0.0
char	\u000
classe	null


Remarque : Dans une applet, il est préférable de faire les déclarations et initialisation dans la méthode init().

3.4.5. L'affectation

le signe = est l'opérateur d'affectation et s'utilise avec une expression de la forme variable = expression. L'opération d'affectation est associatif de droite à gauche : il renvoie la valeur affectée ce qui permet d'écrire :

x = y = z = 0;
Il existe des opérateurs qui permettent de simplifier l'écriture d'une opération d'affectation associée à un opérateur mathématique :

Opérateur	Exemple	Signification
=	a=10	équivalent à : a = 10
+=	a+=10	équivalent à : a = a + 10
-=	a-=10	équivalent à : a = a - 10
*=	a*=10	équivalent à : a = a * 10
/=	a/=10	équivalent à : a = a / 10
%=	A%=10	reste de la division
^=	a^=10	équivalent à : a = a ^ 10
<<=	A<<=10	équivalent à : a = a << 10 a est complété par des zéros à

		droite
>>>= a>>=10		équivalent à : a = a >> 10 a est complété par des zéros à gauche
>>>=	a>>>=10	équivalent à : a = a >>> 10 décalage à gauche non signé

Attention : Lors d'une opération sur des opérandes de types différents, le compilateur détermine le type du résultat en prenant le type le plus précis des opérandes. Par exemple, une multiplication d'une variable de type float avec une variable de type double donne un résultat de type double. Lors d'une opération entre un opérande entier et un flottant, le résultat est du type de l'opérande flottant.

3.4.6. Les comparaisons

Java propose des opérateurs pour toutes les comparaisons :

Opérateur	Exemple	Signification
>	a > 10	strictement supérieur
<	a < 10	strictement inférieur
>=	a >= 10	supérieur ou égal
<	a < 10	inférieur ou égal
==	a == 10	Egalité
!=	a != 10	diffèrent de
&	a & b	ET binaire
^	a ^ b	OU exclusif binaire
|	a | b	OU binaire
&	a && b	ET logique (pour expressions booléennes) : l'évaluation de l'expression cesse dès qu'elle devient fausse
||	a || b	OU logique (pour expressions booléennes) : l'évaluation de l'expression cesse dès qu'elle devient vraie
? :	a ? b : c	opérateur conditionnel : renvoie la valeur b ou c selon l'évaluation de l'expression a (si a alors b sinon c) : b et c doivent retourner le même type

Les opérateurs sont exécutés dans l'ordre suivant à l'intérieure d'une expression qui est analysée de gauche à droite:

- incréments et décréments
- multiplication, division et reste de division (modulo)
- addition et soustraction
- comparaison
- le signe = d'affectation d'une valeur à une variable

L'usage des parenthèses permet de modifier cet ordre de priorité.

3.5. Les opérations arithmétiques

Les opérateurs arithmétiques se notent + (addition), - (soustraction), * (multiplication), / (division) et % (reste de la division). Ils peuvent se combiner à l'opérateur d'affectation

Exemple

nombre += 10;

3.5.1. L'arithmétique entière

Pour les types numériques entiers, Java met en oeuvre une sorte de mécanisme de conversion implicite vers le type int appelée promotion entière. Ce mécanisme fait partie des règles mise en place pour renforcer la sécurité du code.

Exemple
short x= 5 , y = 15;
x = x + y ; //erreur à la compilation

Incompatible type for =. Explicit cast needed to convert int to short.
x = x + y ; //erreur à la compilation
^
1 error
Les opérandes et le résultat de l'opération sont convertis en type int. Le résultat est affecté dans un type short : il y a donc risque de perte d'informations et donc erreur à la compilation est émise. Cette promotion évite un débordement de capacité sans que le programmeur soit pleinement conscient du risque : il est nécessaire, pour régler le problème, d'utiliser une conversion explicite ou cast

Exemple
x = (short) (x + y);
Il est nécessaire de mettre l'opération entre parenthèse pour que ce soit son résultat qui soit converti car le cast a une priorité plus forte que les opérateurs arithmétiques.

La division par zéro pour les types entiers lève l'exception ArithmeticException

Exemple
```
/* test sur la division par zero de nombres entiers */
class test3 {
   public static void main (String args[]) {
      int valeur=10;
      double résultat = valeur / 0;
      System.out.println("index = " + résultat);
   }
}
```

3.5.2. L'arithmétique en virgule flottante

Avec des valeurs float ou double, la division par zéro ne produit pas d'exception mais le résultat est indiqué par une valeur spéciale qui peut prendre trois états :

- indéfini : Float.NaN ou Double.NaN (not a number)
- indéfini positif : Float.POSITIVE_INFINITY ou Double.POSITIVE_INFINITY, + ∞
- indéfini négatif : Float.NEGATIVE_INFINITY ou Double.NEGATIVE_INFINITY, + ∞

Conformément à la norme IEEE754, ces valeurs spéciales représentent le résultat d'une expression invalide NaN, une valeur supérieure au plafond du type pour infini positif ou négatif.

X Y		X/Y	X%Y
valeur finie	0	+ ∞ NaN	
valeur finie	+/- ∞ 0		x
0 0 NaN			NaN
+/- ∞ valeur	finie	+/- ∞ NaN	
+/- ∞ +/-	∞ NaN		NaN

Exemple
/* test sur la division par zero de nombres flottants */

```
class test2 {
   public static void main (String args[]) {
      float valeur=10f;
      double résultat = valeur / 0;
      System.out.println("index = " + résultat);
   }
}
```

3.5.3. L'incrémentation et la décrémentation

Les opérateurs d'incrémentation et de décrémentation sont : n++ ++n n-- --n

Si l'opérateur est placé avant la variable (préfixé), la modification de la valeur est immédiate sinon la modification n'a lieu qu'à l'issu de l'exécution de la ligne d'instruction (postfixé)

L'opérateur ++ renvoie la valeur avant incrémentation s'il est postfixé, après incrémentation s'il est préfixé.

Exemple

```
System.out.println(x++); // est équivalent à
System.out.println(x); x = x + 1;

System.out.println(++x); // est équivalent à
x = x + 1; System.out.println(x);
```

Exemple
/* test sur les incrementations prefixees et postfixees */

```
class test4 {
   public static void main (String args[]) {
      int n1=0;
      int n2=0;
      System.out.println("n1 = " + n1 + " n2 = " + n2);
      n1=n2++;
      System.out.println("n1 = " + n1 + " n2 = " + n2);
      n1=++n2;
      System.out.println("n1 = " + n1 + " n2 = " + n2);
      n1=n1++;       //attention
      System.out.println("n1 = " + n1 + " n2 = " + n2);
   }
}
```

Résultat

```
int n1=0;
int n2=0; // n1=0 n2=0
```

n1=n2++; // n1=0 n2=1
n1=++n2; // n1=2 n2=2
n1=n1++; // attention : n1 ne change pas de valeur

3.6. La priorité des opérateurs

Java définit les priorités dans les opérateurs comme suit (du plus prioriotaire au moins prioritaire)

les parenthèses	( )
les opérateurs d'incrémentation	++ --
les opérateurs de multiplication, division, et modulo	* / %
les opérateurs d'addition et soustraction	+ -
les opérateurs de décalage	<< >>
les opérateurs de comparaison	< > <= >=
les opérateurs d'égalité	== !=
l'opérateur OU exclusif	^
l'opérateur ET	&
l'opérateur OU	|
l'opérateur ET logique	&&
l'opérateur OU logique	||
les opérateurs d'assignement	= += -=

Les parenthèses ayant une forte priorité, l'ordre d'interprétation des opérateurs peut être modifié par des parenthèses.

3.7. Les structures de contrôles

Comme quasi totalité des langages de développement orienté objets, Java propose un ensemble d'instructions qui permettent de d'organiser et de structurer les traitements. L'usage de ces instructions est similaire à celui rencontré dans leur équivalent dans d'autres langages.

3.7.1. Les boucles

```
while ( boolean )
{
   ... // code a exécuter dans la boucle
}
```

Le code est exécuté tant que le booléen est vrai. Si avant l'instruction while, le booléen est faux, alors le code de la boucle ne sera jamais exécuté

Ne pas mettre de ; après la condition sinon le corps de la boucle ne sera jamais exécuté

```
do {
  ...
} while ( boolean )
```
Cette boucle est au moins exécuté une fois quelque soit la valeur du booléen;

```
for ( initialisation; condition; modification) {
   ...
}
```
Exemple
```
for (i = 0 ; i < 10; i++ ) { .... }
for (int i = 0 ; i < 10; i++ ) { .... }
for ( ; ; ) { ... } // boucle infinie
```
L'initialisation, la condition et la modification de l'index sont optionels.

Dans l'initialisation, on peut déclarer une variable qui servira d'index et qui sera dans ce cas locale à la boucle.

Il est possible d'inclure plusieurs traitements dans l'initialisation et la modification de la boucle : chacun des traitements doit être séparé par une virgule.

Exemple
```
for (i = 0 , j = 0 ; i * j < 1000;i++ , j+= 2) { ....}
```
La condition peut ne pas porter sur l'index de la boucle :

Exemple
```
boolean trouve = false;
for (int i = 0 ; !trouve ; i++ ) {
   if ( tableau[i] == 1 )
   trouve = true;
   ... //gestion de la fin du parcours du tableau
}
```
Il est possible de nommer une boucle pour permettre de l'interrompre même si cela est peu recommendé :

Exemple
```
int compteur = 0;
boucle:
while (compteur < 100) {

   for(int compte = 0 ; compte < 10 ; compte ++) {
      compteur += compte;
      System.out.println("compteur = "+compteur);
      if (compteur > 40) break boucle;
   }
}
```

3.7.2. Les branchements conditionnels

```
if (boolean) {
   ...
} else if (boolean) {
```

```
   ...
} else {
   ...
}
swith (expression) {
   case constante1 :
      instr11;
      instr12;
      break;

   case constante2 :
      ...
   default :
      ...
}
```

On ne peut utiliser switch qu'avec des types primitifs d'une taille maximum de 32 bits (byte, short, int, char).

Si une instruction case ne contient pas de break alors les traitements associés au case suivant sont exécutés.

Il est possible d'imbriquer des switch

L'opérateur ternaire : (condition) ? valeur-vrai : valeur-faux

Exemple

```
if (niveau == 5) // equivalent à total = (niveau ==5) ? 10 : 5;
total = 10;
else total = 5 ;
System.out.println((sexe == "H") ? "Mr" : "Mme");
```

3.7.3. Les débranchements

break : permet de quitter immédiatement une boucle ou un branchement. Utilisable dans tous les contrôles de flot

continue : s'utilise dans une boucle pour passer directement à l'itération suivante

break et continue peuvent s'excuter avec des blocs nommés. Il est possible de préciser une étiquette pour indiquer le point de retour lors de la fin du traitement déclenché par le break.

Une étiquette est un nom suivi d'un deux points qui définit le début d'une instruction.

3.8. Les tableaux

Ils sont dérivés de la classe Object : il faut utiliser des méthodes pour y accéder dont font partie des messages de la classe Object tel que equals() ou getClass().

Le premier élément d'un tableau possède l'indice 0.

3.8.1. La déclaration des tableaux

Java permet de placer les crochets après ou avant le nom du tableau dans la déclaration.

Exemple
int tableau[] = new int[50]; // déclaration et allocation

OU int[] tableau = new int[50];

OU int tab[]; // déclaration
tab = new int[50]; //allocation
Java ne supporte pas directement les tableaux à plusieurs dimensions : il faut déclarer un tableau de tableau.

Exemple
float tableau[][] = new float[10][10];
La taille des tableaux de la seconde dimension peut ne pas être identique pour chaque occurrence.

Exemple
int dim1[][] = new int[3][];
dim1[0] = new int[4];
dim1[1] = new int[9];
dim1[2] = new int[2];
Chaque élément du tableau est initialisé selon son type par l'instruction new : 0 pour les numériques, '\0' pour les caractères, false pour les booléens et nil pour les chaines de caractères et les autres objets.

3.8.2. L'initialisation explicite d'un tableau

Exemple:
int tableau[5] = {10,20,30,40,50};
int tableau[3][2] = {{5,1},{6,2},{7,3}};
La taille du tableau n'est pas obligatoire si le tableau est initialisé à sa création.

Exemple:
int tableau[] = {10,20,30,40,50};
Le nombre d'élément de chaque lignes peut ne pas être identique :

Exemple:
```
int[][] tabEntiers = {{1,2,3,4,5,6},
                      {1,2,3,4},
                      {1,2,3,4,5,6,7,8,9}};
```

3.8.3. Le parcours d'un tableau

Exemple:
for (int i = 0; i < tableau.length ; i ++) { ... }
La variable **length** retourne le nombre d'éléments du tableau.

Pour passer un tableau à une méthode, il suffit de déclarer les paramètres dans l'en tête de la méthode

Exemple:
```
public void printArray(String texte[]){ ...
}
```
Les tableaux sont toujours transmis par référence puisque se sont des objets.

Un accès à un élément d'un tableau qui dépasse sa capacité, lève une exception du type java.lang.arrayIndexOutOfBoundsException.

3.9. Les conversions de types

Lors de la déclaration, il est possible d'utiliser un cast :

Exemple:
int entier = 5;
float flottant = (float) entier;
La conversion peut entrainer une perte d'informations.

Il n'existe pas en Java de fonction pour convertir : les conversions de type se font par des méthodes. La bibliothèque de classes API fournit une série de classes qui contiennent des méthodes de manipulation et de conversion de types élémentaires.

Classe	Role
String	pour les chaines de caractères Unicode
Integer	pour les valeurs entières (integer)
Long	pour les entiers long signés (long)
Float	pour les nombres à virgules flottante (float)
Double	pour les nombres à virgule flottante en double précision (double)

Les classes portent le même nom que le type élémentaire sur lequel elles reposent avec la première lettre en majuscule.

Ces classes contiennent généralement plusieurs constructeurs. Pour y accéder, il faut les instancier puisque ce sont des objets.

Exemple:
String montexte;
montexte = new String("test");
L'objet montexte permet d'accéder aux méthodes de la classe java.lang.String

3.9.1. La conversion d'un entier int en chaîne de caractère String

Exemple:
int i = 10;
String montexte = new String();
montexte =montexte.valueOf(i);
valueOf est également définie pour des arguments de type boolean, long, float, double et char

3.9.2. La conversion d'une chaîne de caractères String en entier int

Exemple:
String montexte = new String(« 10 »);
Integer nomnombre=new Integer(montexte);
int i = monnombre.intValue(); //convertion d'Integer en int

3.9.3. La conversion d'un entier int en entier long

Exemple:
int i=10;
Integer monnombre=new Integer(i);
long j=monnombre.longValue();

3.10. La manipulation des chaines de caractères

La définition d'un caractère :

Exemple:
char touche = '%';
La définition d'une chaîne :

Exemple:
String texte = "bonjour";
Les variables de type String sont des objets. Partout ou des constantes chaines figurent entre guillemets, le compilateur Java génère un objet de type String avec le contenu spécifié. Il est donc possible d'ecrire :

Exemple:
String texte = « Java Java Java ».replace('a','o');
Les chaines ne sont pas des tableaux : il faut utiliser les méthodes de la classes String d'un objet instancié pour effectuer des manipulations.

Il est impossible de modifier le contenu d'un objet String contruit à partir d'une constante. Cependant, il est possible d'utiliser les méthodes qui renvoient une chaîne pour modifier le contenu de la chaine

Exemple:
String texte = « Java Java Java »;
texte = texte.replace('a','o');
Java ne fonctionne pas avec le jeu de caractères ASCII ou ANSI, mais avec Unicode (Universal Code). Ceci concerne les types char et les chaines de caractères. Le jeu de caractères Unicode code un caractère sur 2 octets. Les caractères 0 à 255 correspondent exactement au jeu de caractères ASCII

3.10.1. Les caractères spéciaux dans les chaines

Caractères spéciaux	Affichage
\' Apostrophe	
\" Guillemet	
\\ anti	slash
\t Tabulation	
\b retour	arrière (backspace)
\r retour	chariot
\f	saut de page (form feed)
\n	saut de ligne (newline)
\0ddd	caractère ASCII ddd (octal)
\xdd	caractère ASCII dd (hexadécimal)
\udddd	caractère Unicode dddd (hexadécimal)

3.10.2. L'addition de chaines

Java admet l'opérateur + comme opérateur de concaténation de chaines de caractères.

L'opérateur + permet de concatener plusieurs chaines. Il est possible d'utiliser l'opérateur +=

Exemple:

String texte = " ";
texte += " Hello ";
texte += " World3 ";
Cet opérateur sert aussi à concatener des chaînes avec tous les types de bases. La variable ou constante est alors convertie en chaîne et ajoutée à la précédente. La condition préalable est d'avoir au moins une chaîne dans l'expression sinon le sinon '+' est évalué comme opérateur mathématique.

Exemple:
System.out.println(" La valeur de Pi est : "+Math.PI);
int duree = 121;
System.out.println(" durée = " +duree);

3.10.3. La comparaison de deux chaines

Il faut utiliser la méthode equals()

Exemple:
String texte1 = "texte 1 ";
String texte2 = "texte 2 ";
if (texte1.equals(texte2))...

3.10.4. La détermination de la longueur d'une chaine

La méthode length() permet de déterminer la longueur d'une chaine.

Exemple:
String texte = "texte";
int longueur = texte.length();

3.10.5. La modification de la casse d'une chaine

Les méthodes Java toUpperCase() et toLowerCase() permettent respectivement d'obtenir une chaîne tout en majuscule ou tout en minuscule.

Exemple:
String texte = "texte ";
String textemaj = texte.toUpperCase();

4. Les fichiers :

Les entrées/sorties en Java

La langage Java, comme tout langage de programmation digne de ce nom, fournit une API de manipulation de flux de données. Quasiment toutes les classes de cette API sont localisées dans le package **java.io.*** , à l'exception des classes de manipulation de flux sur sockets. Pour ces dernières, vous les trouverez dans le package **java.net.*** .

Afin d'appréhender au mieux les choses, nous allons diviser notre étude en plusieurs sous-parties.

- Dans un premier temps, nous allons nous focaliser sur comment manipuler un fichier (indépendamment des données qu'il contient). Pour ce faire, nous utiliserons la classe **java.io.File** . Elle expose un grand nombre de méthodes : nous en verrons les plus utiles.
- En second lieu, nous allons voir comment manipuler les flux de données. Plus précisément, nous allons voir comment exploiter les données contenues dans un fichier. Un grand nombre de classes peuvent alors être utilisées en fonction des besoins.
- Enfin, nous finirons cette présentation en parlant de sérialisation. Ce concept consiste en la possibilité de prendre un objet en mémoire et d'envoyer son état (la valeur de ses attributs) sur un flux de données (pourquoi pas vers un fichier, ou mieux encore, vers une socket).

Chaque chapitre est agrémenté de nombreux exemples de code, afin de faciliter au mieux votre apprentissage. Ce cours est maintenu en permanence. Si vous détectez des erreurs, ou si vous avez des suggestions à apporter, dans le but d'améliorer sa qualité, n'hésitez pas à me contacter par email.

4.1 Manipulations de fichiers

Nous allons, dans cette section, voir comment on manipule les fichiers en Java. Comme il est dit dans l'introduction, nous n'allons pas maintenant voir comment manipuler les données que contient le fichier : il nous faudra pour ce faire utiliser les classes de flux (streams en anglais), ce que nous verrons dans la section suivante. En revanche, nous allons voir quelles sont les possibilités de manipulation de fichiers tels que le renommage, la suppression, connaître les droits d'un fichier en terme de sécurité, ...

Toutes ces possibilités vous sont offertes par l'intermédiaire de la classe **File** du package **java.io**. Nous terminerons cette section en analysant un exemple de code permettant de lister le contenu d'un répertoire.

La classe java.io.File.

La classe File permet donc de manipuler un fichier sur le disque dur. Elle propose à cet effet, un grand nombre de méthodes. L'exemple suivant montre un exemple simple d'utilisation de cette classe : on y créé un objet File basé sur un fichier, puis on l'utilise pour connaître

certaines de ces caractéristiques (Droits d'accès, longueur du fichier). Au final, le fichier est supprimé.

```
File f = new File("fichier.mp3");

System.out.println(f.getAbsolutePath() + f.getName());

if (f.exists()) {
    System.out.println(f.getName() + " : " +
        (f.canRead()  ? "r" : "-") +
        (f.canWrite() ? "w" : "-") + " : " +
        f.length()
    );
    f.delete();
}
```

Le tableau suivant présente certaines méthodes couramment utilisées. Pour chacune d'entre elles, un court descriptif vous est proposé. Pour de plus amples informations sur cette classe, vous pouvez toujours vous référer à la documentation en ligne de l'API fournie pas Sun : http://java.sun.com/j2se/1.4/docs/api/index.html.

String getName();	Retourne le nom du fichier.
String getPath();	Retourne la localisation du fichier en relatif.
String getAbsolutePath();	Idem mais en absolu.
String getParent();	Retourne le nom du répertoire parent.
boolean renameTo(File newFile);	Permet de renommer un fichier.
boolean exists() ;	Est-ce que le fichier existe ?
boolean canRead();	Le fichier est t-il lisible ?
boolean canWrite();	Le fichier est t-il modifiable ?
boolean isDirectory();	Permet de savoir si c'est un répertoire.
boolean isFile();	Permet de savoir si c'est un fichier.
long length();	Quelle est sa longueur (en octets) ?
boolean delete();	Permet d'effacer le fichier.
boolean mkdir();	Permet de créer un répertoire.
String[] list();	On demande la liste des fichiers localisés dans le répertoire.

Parcours du contenu d'un répertoire

Pour finaliser la présentation de cette classe, vous pouvez consulter ce dernier exemple. La classe Scan ne contient qu'une méthode statique : le main. Celle-ci accepte en paramètre le nom d'un dossier à consulter. On en demande la liste de tout ce qu'il contient et on affiche chacune des entrées en spécifiant s'il s'agit d'un dossier ou d'un fichier.

```
import java.io.*;
```

```
public class Scan {

    public static void main(String[] args) throws Exception
{
        if (args.length != 1) {
            System.out.println("Veuillez saisir un dossier
!");
            System.exit(0);
        }

        File f = new File(args[0]);
        String [] files = f.list();

        for(int i =0; i < files.length; i++) {
            if (new File(args[0] + "\\" +
files[i]).isDirectory()) {
                System.out.print("Rep : ");
            } else {
                System.out.print("Fil : ");
            }
            System.out.println(args[0] + "\\" + files[i]);
        }
    }
}
```

Conclusion

La classe **java.io.File** vous permet donc de manipuler le système de fichiers. Vous pouvez ainsi créer, supprimer, déplacer ou obtenir des informations aussi bien sur des fichiers que sur des dossiers.

4.2 Manipulations de flux de données

Nous allons, dans cette section, nous intéresser aux flux (streams en anglais) de données, à proprement parlé. Il est vrai qu'il existe plusieurs types de flux de données : des flux basés sur des fichiers, sur des sockets réseaux, sur la console de l'application, ... Mais dans tous les cas, leurs utilisations seront similaires. Ce sont les concepts d'héritage et d'interfaces qui permettent d'obtenir cette abstraction.

Pour initier cette présentation, nous allons donc commencer à parler des flux de données prédéfinis. Puis nous présenterons les principales classes du package **java.io** (comme vous allez le voir, il y a de quoi faire). Enfin nous étudierons un exemple concret permettant de réaliser une copie de fichiers.

Les flux de données prédéfinis.

Il existe, comme dans presque tous les autres langages, trois flux prédéfinis. Ces trois flux sont associés à l'entrée standard d'une application, sa sortie standard ainsi que sa sortie standard d'erreurs. Ils sont respectivement nommés **System.in**, **System.out** et **System.err**. Nous pouvons d'ores et déjà signaler que System.in est une instance de la classe **InputStream**, alors que les deux autres flux sont des instances de la classe **PrintStream**.

```
try {
    int c;
    while((c = System.in.read()) != -1) {
        System.out.print(c);  nbc++;
    }
} catch(IOException exc) {
    exc.printStackTrace();  // En fait
exc.printStackTrace(System.err);
}
```

Pour ce qui est de la manipulation du flux de sortie, il est préférable de ne pas directement utiliser la classe **InputStream**. En effet, elle ne propose que des méthodes élémentaires de récupération de données. Préférez au contraire la classe **BufferedReader**. Nous reviendrons ultérieurement, dans ce document, sur la notion de **Reader** et de **Writer**. Pour l'heure, sachez simplement générer un objet **BufferedReader** à partir de **System.in**, comme le montre l'exemple suivant. Cela vous permettra de facilement récupérer des chaînes de caractères saisies sur la console par l'utilisateur.

```
Reader reader = new InputStreamReader(System.in);
BufferedReader keyboard = new BufferedReader(reader);

System.out.print("Entrez une ligne de texte : ");
String line = keyboard.readLine();
System.out.println("Vous avez saisi : " + line);
```

Topographie des classes de flux de données

Le tableau suivant vous montre brièvement quelques classes proposées dans le package **java.io**. Comme vous pouvez le remarquer le tableau est constitué de quatre zones distinctes. La colonne de gauche contient quelques noms de classes de flux de lecture, contrairement à la colonne de droite qui, elle, contient des classes de flux d'écriture.

En plus, le tableau est aussi divisé en deux lignes. La première contient des classes qui manipulent des octets. Ces classes existent plus ou moins depuis la version originale du JDK (Java Development Kit). Mais certaines de leurs possibilités (de leurs méthodes) ont rapidement été dépréciées. En effet, les méthodes permettant l'acquisition de chaînes de caractères présentent l'inconvénient de ne pas être portables d'un système à un autre, ce qui pour Java est inacceptable. N'oubliez pas que la norme ASCII (American Standard Code for Information Interchange) ne spécifie que 128 caractères. Or, nos codes accentués (par exemple) n'en font pas partie. Il existe, au final, plusieurs dérivés d'ASCII (utilisés d'un système à un autre) ne partageant pas tous les mêmes codes.

En Java, le problème se règle via l'utilisation de système Unicode (16 bits). Il peut spécifier, théoriquement, jusqu'à 65536 caractères. On est donc tranquille pour l'échange d'informations textuelles entre des systèmes hétérogènes. Mais, attention : dans la majorité des cas, les flux sont des flux 8 bits. Qu'est ce que signifie donc, dans ce cas, l'utilisation de **Reader** ou de **Writer**. En fait c'est simple, ces classes permettent de réaliser des transformations de code d'un système ASCII dérivé vers Unicode et réciproquement. Ainsi, par exemple, **System.in** est et restera un flux 8 bits, mais l'utilisation d'un **Reader** permettra de transformer les caractères ASCII (ou dérivé) en code Unicode en tenant compte du système de codage utilisé sur le poste. Ces classes sont apparues à partir du JDK 1.1.

Notez bien que le tableau suivant n'est pas exhaustif, bien au contraire. Pour s'en convaincre, jetez un oeil sur la documentation du JDK pour le package **java.io**.

	Flux d'entrées	Flux de sorties
JDK 1.0 **Flux d'octets (8 bits)**	InputStream +- FileInputStream +- DataInputStream +- BufferedInputStream +- . . .	OutputStream +- FileOutputStream +- DataOutputStream +- BufferedOutputStream +- . . .
JDK 1.1 **Flux de caractères (16 bits)**	Reader +- FileReader +- BufferedReader +- StringReader +- . . .	Writer +- FileWriter +- BufferedWriter +- StringWriter +- . . .

Encore une petite chose : initialement, quasiment tous vos flots seront de types **InputStream** ou **OutputStream** (ou de classes dérivées). Comment faire pour en obtenir un **Reader** ou un **Writer**. En fait c'est simple : deux classes de transition vous sont proposées. Elles se nomment **InputStreamReader** et **OutputStreamWriter**. Elles permettent, respectivement, de transformer un **InputStream** en un **Reader** et un **OutputStream** en un **Writer**. A titre d'exemple, je vous rappelle juste le code permettant de générer un flot d'acquisition de chaînes de caractères à partir de la console.

```
Reader reader = new InputStreamReader(System.in);
BufferedReader keyboard = new BufferedReader(reader);

System.out.print("Entrez une ligne de texte : ");
String line = keyboard.readLine();
System.out.println("Vous avez saisi : " + line);
```

Exemple d'utilisation des classes de flux

Maintenant que l'on voit un peu mieux à quoi correspondent telle ou telle classe, il nous faut comprendre comment elles s'utilisent. En effet, pour obtenir le flux adapté à vos besoins, il faut souvent passer par plusieurs constructions intermédiaires.

Les exemples de code suivants montrent comment cumuler les constructions de classes de flux pour arriver au résultat escompté. Le but final est d'obtenir un flot sur fichier, bufférisé, permettant de manipuler des données typées.

```
File f = new File("fichier.mp3");
FileInputStream fis = new FileInputStream(f);
BufferedInputStream bis = new BufferInputStream(fis);
DataInputStream dis = new DataInputStream(bis);

int a = dis.readInt();
short s = dis.readShort();
boolean b = dis.readBoolean();
```

Il est important de comprendre que l'ordre de construction ne peut en aucun cas être changé. En effet, l'objet final à utiliser se doit d'être de type DataInputStream. En effet, c'est sur ce type que les méthodes attendues sont définies. Le second exemple montre le même exemple, mais pour écrire dans un flux.

```
File f = new File("fichier.mp3");
FileOutputStream fos = new FileOutputStream(f);
BufferedOutputStream bos = new BufferedOutputStream(fos);
DataOutputStream dos = new DataOutputStream(bos);

int a = 10;       dos.writeInt(a);
short s = 3;      dos.writeShort(s);
boolean b = true; dos.writeBoolean(b);
```

Un cas concret : réaliser une copie de fichier en Java

L'exemple que je vous propose permet de réaliser une copie de fichier. Pour ce faire, votre programme attend que le nom du fichier source et le nom du fichier de destination soient renseignés sur la ligne de commande servant à lancer le programme. A titre indicatif, voici un exemple de commande servant à démarrer le programme en sachant que si le nombre de paramètres n'est pas correct, le programme vous informera de qu'elle est sa bonne utilisation (son usage).

```
> java Copy sourceFile.txt destFile.txt
```

Ensuite, le programme réalise la copie à proprement parler. Deux points me semblent être à préciser. Premièrement, nous allons travailler sur des flux binaires et non des flux textuels : il est donc judicieux de choisir les classes dérivées de **InputStream** et de **OutputStream**. Secondo, la manipulation de flux peut aboutir à lever des exceptions. Il faut donc spécifier ce que l'on fera d'une éventuelle exception. Ici, j'ai choisi de mettre un bloc **try ... catch** et d'afficher la pile des appels de méthodes (mais un **throws Exception** sur le prototype du main en aurait fait autant).

Dans le but de vous montrer un maximum de choses, je récupère la taille du fichier grâce à l'objet de type File. Mais j'aurais pu coder la boucle de lecture des octets jusqu'à obtenir la valeur -1 (c'est elle qui vous est renvoyée en cas de fin de fichier). Dans le but d'accélérer le traitement, je place des buffers (des tampons) sur les flux basés sur les fichiers. Dans ce cas, il faut absolument que vous fermiez les flux bufférisés et non les flux simples (en fait les derniers objets de flux créés). Pour le reste, il me semble que les choses sont suffisamment simples.

```
import java.io.*;

public class Copy {

    public static void main (String[] argv)   {
        // Test sur le nombre de paramètres passés
        if (argv.length != 2) {
            System.out.println("Usage> java Copy sourceFile
destinationFile");
            System.exit(0);
 }
```

```
       try {
            // Préparation du flux d'entrée
            File sourceFile = new File(argv[0]);
            FileInputStream fis = new
FileInputStream(sourceFile);
            BufferedInputStream bis = new
BufferedInputStream(fis);
            long l = sourceFile.length();

            // Préparation du flux de sortie
            FileOutputStream fos = new
FileOutputStream(argv[1]);
             BufferedOutputStream bos = new
BufferedOutputStream(fos);

            // Copie des octets du flux d'entrée vers le
flux de sortie
            for(long i=0;i<l;i++) {
                bos.write(bis.read());
            }

            // Fermeture des flux de données
            bos.flush();
            bos.close();
            bis.close();

 }        catch (Exception e) {
            System.err.println("File access error !");
            e.printStackTrace();
 }

 System.out.println("Copie        terminée");
    }
}
```

Conclusion

Nous avons donc, dans cette section, étudié comment manipuler les classes de flux proposées en Java. En fait, il y a toute une arborescence de classes qui vous est proposée. Chacune de ces classes présente certaines caractéristiques que vous pouvez cumuler par constructions successives.

Nous avons aussi vu que trois objets de flux sont initialement définis pour toutes applications : **System.in**, **System.out** et **System.err**. Vous pouvez, eux aussi, les spécialiser. Finalement nous avons mis en oeuvre un exemple permettant de réaliser une copie de fichier : il nous a donc fallu créer des objets de flux basés sur des fichiers mais en plus bufférisés.

4.3 La sérialisation en Java

La sérialisation consiste à pouvoir prendre un objet en mémoire et à en sauvegarder l'état sur un flux de données (vers un fichier, par exemple). Ce concept permettra aussi de reconstruire, ultérieurement, l'objet en mémoire à l'identique de ce qu'il pouvait être initialement. La sérialisation peut donc être considérée comme une forme de persistance de vos données.

Présentation du concept

Les problématiques

Mais cela n'est pas aussi simple qu'il n'y parait. En effet, quelques problèmes font que la sérialisation n'est pas si simple à mettre en oeuvre. Pour s'en convaincre, dite vous que le langage C++ (par exemple) ne fournit pas, de base, la sérialisation. Certes, certaines librairies (les MFC, pour ne citer qu'elles) proposent le concept, mais pas avec le même aboutissement qu'en Java.

Le premier problème réside dans le fait que les attributs (en effet, ce sont eux qu'il faut sauvegarder pour sauvegarder l'état de l'objet) ne sont pas tous de types scalaires (int, float, double, ...). Certains sont de types agrégats (sous-objets ou tableaux). Or, en Java, les agrégats sont obligatoirement stockés dans le tas (la mémoire autre que la pile d'exécution) : ils sont donc inévitablement pointés. Or, qu'est ce que cela veut dire que de sérialiser un pointeur ? Ne serait-ce pas plutôt l'objet pointé qu'il faudrait sauvegarder ?

A partir de cette question, une autre en découle tout naturellement. Comment savoir si le type d'un attribut est un type scalaire ou un agrégat ?

Une dernière question vous est peut être venue à l'esprit : comment le mécanisme de sérialisation va fonctionner sur deux objets qui se pointent mutuellement ? Il ne faudrait, en effet, pas sauvegarder les mêmes objets indéfiniment.

Les solutions

En fait, il n'y a pas des solutions, mais une solution à tous ces problèmes : **la réflexion**. La réflexion permet, par programmation, d'obtenir des informations sur un type de données chargé en mémoire. Ces informations permettent de décrire les attributs, les méthodes et les constructeurs de la classe considérée.

Ces informations descriptives sont stockées dans les **méta-classes**. Une méta-classe est donc un type qui en identifie un autre. En fait, si l'on y réfléchit bien, les concepts de méta-classes et de réflexion sont les clés de voûte de tout le système Java (Java Bean, JNI, RMI, ...).

A titre d'information, le concept d'introspection peut être vu comme un aboutissement de la réflexion. Ce que traite en plus l'introspection, ce sont les conventions de codage définies en Java. Ainsi, à partir des méthodes exposées par une classe on peut aussi en déduire ses propriétés (les gets et les sets) et ses évènements (les écouteurs). Mais cela n'apporte rien de plus pour ce qui est de la sérialisation.

Pour que l'on puisse manipuler les méta-classes, il faut que le compilateur joigne la table des symboles au fichier de byte code générer. C'est ce que ne sait pas faire C++ : les méta-classes en sont donc plus difficiles à fournir (certain compilateurs y arrivent malgré tout en mode debug).

Pour pouvoir utiliser la réflexion, l'environnement Java vous fournit la classe **java.lang.Class**, ainsi qu'un bon nombre d'autres classes utilitaires localisées dans le package **java.lang.reflect**. A titre d'exemple, je vous suggère de tester le programme suivant. Il affiche

l'ensemble des attributs d'une classe en précisant si, pour chaque attribut, son type est scalaire ou s'il s'agit d'un agrégat.

```
import java.lang.reflect.*;

public class MetaDonnees {
    public static void main(String[] args) {
        Class metaClass = javax.swing.JButton.class;
        /* ou Class metaClass = new JButton().getClass(); */

        Field attributes[] = metaClass.getFields();
        for(int i=0; i<attributes.length; i++) {
            boolean scalar =
attributes[i].getType().isPrimitive();
            System.out.print(scalar ? "Scalar" : "Object");
            System.out.print(" : " + attributes[i].getName()
+ " de type ");
            System.out.println(attributes[i].getType());
        }
    }
}
```

Le support de sérialisation en Java

Utilisation de la sérialisation

Maintenant que vous avez bien compris les pré-requis, nous pouvons nous focaliser sur la mise en oeuvre de la sérialisation via l'API de Java. Tout ce que vous devez savoir est localisé dans le package **java.io**. On y trouve notamment deux classes : **ObjectInputStream** et **ObjectOutputStream**. Ces deux classes proposent, respectivement, les méthodes **readObjet** et **writeObject**. Ce sont ces méthodes qui vont vous permettre la sérialisation. Pour toutes les problématiques précédentes, vous ne vous occupez de rien : tout est pris en charge par ces deux classes.

L'exemple de code suivant tente de vous montrer la puissance du mécanisme. Pour ce faire, la méthode saveWindow construit une fenêtre contenant différents composants graphiques (zone de saisie de texte, boutons, arborescence, ...) puis sauvegarde l'objet de fenêtre sur le disque. A partir de là, tous les autres composants doivent être aussi sérialisés. La méthode loadWindow, permet, quant à elle, l'opération inverse : elle reconstruit la fenêtre et tous les éléments initialement contenus. La méthode "main" se charge, elle, d'acquérir les commandes de la part de l'utilisateur et de les traiter.

En guise de test, demandez d'abord de sauvegarder une fenêtre. Regardez alors la taille du fichier "window.ser" sur le disque (pour ma part 19,7 Ko). Puis demandez consécutivement, de recharger la fenêtre quelques fois : on peut considérer le fichier comme un modèle servant à régénérer autant d'instance que souhaité. Tapez *exit* pour sortir de la fenêtre.

```
import java.io.*;
import java.awt.*;
import javax.swing.*;

public class Serialisation {
    private final static Reader reader = new
InputStreamReader(System.in);
    private final static BufferedReader keyboard = new
```

```
BufferedReader(reader);

    // Permet de créer une fenêtre et de la sérialiser dans
un fichier.
    public void saveWindow() throws IOException {
        JFrame window = new JFrame("Ma fenêtre");
        JPanel pane = (JPanel)window.getContentPane();
        pane.add(new JLabel("Barre de status"),
BorderLayout.SOUTH);
        pane.add(new JTree(), BorderLayout.WEST);
        JTextArea textArea = new JTextArea("Ceci est le
contenu !!!");
        textArea.setBackground(Color.GRAY);
        pane.add(textArea, BorderLayout.CENTER);

        JPanel toolbar = new JPanel(new FlowLayout());
        toolbar.add(new JButton("Open"));
        toolbar.add(new JButton("Save"));
        toolbar.add(new JButton("Cut"));
        toolbar.add(new JButton("Copy"));
        toolbar.add(new JButton("Paste"));

        pane.add(toolbar, BorderLayout.NORTH);
        window.setSize(400,300);

        FileOutputStream fos = new
FileOutputStream("window.ser");
        ObjectOutputStream oos = new
ObjectOutputStream(fos);
        oos.writeObject(window);
        oos.flush();
        oos.close();
    }

    // Permet de reconstruire la fenêtre à partir des
données du fichier.
    public void loadWindow() throws Exception {
        FileInputStream fis = new
FileInputStream("window.ser");
        ObjectInputStream ois = new ObjectInputStream(fis);
        JFrame window = (JFrame)ois.readObject();
        ois.close();

        window.setVisible(true);
    }

    // Permet de saisir différentes commandes. Testez
plusieurs load
    // consécutifs : plusieurs fenêtres doivent apparaîtrent
    public static void main(String[] args) throws Exception
{
        Serialisation object = new Serialisation();

        while(true) {
            System.out.print("Saisir le mode d'execution
(load ou save) : ");
            String mode = keyboard.readLine();

            if (mode.equalsIgnoreCase("exit")) break;
            if (mode.equalsIgnoreCase("save"))
```

```
object.saveWindow();
            if (mode.equalsIgnoreCase("load"))
object.loadWindow();
        }

        System.exit(0);
    }
}
```

Coder une classe sérialisable

Nous venons de voir comment sauver ou recharger un objet sur un flux de données. Nous allons, maintenant voir comment coder une classe sérialisable. En effet, par défaut, vos classes ne permettent pas de sauvegarder l'état d'un objet sur un flux de données. Il faut explicitement le demander. Par contre, pour une grande majorité des classe du J2Sdk (Java 2 Software Development Kit), elles ont été définies comme étant sérialisables. C'est pour cela que l'exemple précédent a fonctionné sans aucun problème.

En fait, les choses sont très simples. Il faut simplement marquer votre classe comme étant sérialisable. On peut caricaturer en disant que le compilateur préfère avoir votre confirmation avant d'entreprendre quoi que se soit : en effet, un objet peut théoriquement avoir une taille phénoménale.

Pour ce faire, il vous suffit d'implémenter l'interface **java.io.Serializable**. Du coup, vous paniquez : il n'y a aucune raison ! Cette interface ne contient aucune méthode : vous n'avez rien d'autre à faire. Le but de la manipulation est simplement de marquer l'objet comme étant sérialisable : comme vous avez codé l'implémentation de l'interface, le compilateur travaille maintenant en confiance (il a vos confirmations) et va se débrouiller pour le reste. L'extrait de code suivant montre une classe dont les attributs pourront être envoyés vers un flux de données.

```
import java.io.*;
import java.util.*;
public MaClasse implements Serializable {
    private String monPremierAttribut;
    private Date monSecondAttribut;
    private long monTroisièmeAttributs;

    // . . . d'éventuelles méthodes . . .
}
```

Dans certains cas subtils, vous pourriez avoir besoin qu'un attribut ne soit pas sérialisé pour une classe donnée. Cela est réalisable sans de réelle difficulté. Il suffit de rajouter le qualificateur **transient** à l'attribut en question. Le code suivant définit une classe dont le second attribut ne sera jamais sérialisé.

```
import java.io.*;
import java.util.*;
public MaClasse implements Serializable {
    private String monPremierAttribut;
    private transient Date monSecondAttribut;  // non
sérialisé
    private long monTroisièmeAttributs;
```

```
    // . . . d'éventuelles méthodes . . .
}
```

Un piége à éviter

Pour terminer cette section, il faut que je vous mette en garde. Il y a une source potentielle de fuites mémoire avec le mécanisme de sérialisation : j'en ai d'ailleurs déjà fait les frais. En fait ce n'est pas réellement un bug : il faut juste bien comprendre comment marchent les classes **ObjectInputStream** et **ObjectOutputStream**.

Pour détecter les cycles de pointeurs et donc éviter de sérialiser plusieurs fois un objet dans un flux, les deux classes précédentes gèrent des vecteurs (java.util.Vector). Un tel conteneur, peut contenir autant d'objets (de pointeurs) que ce qu'il y a de mémoire disponible. A chaque fois qu'un objet est manipulé par l'intermédiaire de l'une des deux classes de flux d'objet, sa référence est stockée dans le vecteur associé.

Dans un de mes développements, j'ai eu besoin de coder un client et un serveur TCP/IP. Pour communiquer, j'ai défini des nombreuses classes dérivant toutes d'une même classe mère nommée Message. Cette classe définit une méthode abstraite *process*. Chacune des autres classes la redéfinit. Le client génère ainsi des objets de messages divers et les envoie au serveur. Le client est aujourd'hui codé de manière similaire à l'extrait de code qui suit.

```
try {
    OutputStream os = sockService.getOutputStream();
    ObjectOutputStream oos = new ObjectOutputStream(os);

    while(true) {
        Message msg = this.createMessage();
        oos.writeObject(msg)
        oos.reset();
    }
} catch(Exception e) {
    e.printStackTrace();
}
```

Le problème, auquel je me suis heurté, réside dans le fait que j'avais omis la ligne marquée en gras. Elle permet de réinitialiser le vecteur contenu dans l'objet de flot. Ainsi, il ne croît pas indéfiniment et le programme n'arrive donc plus à une erreur de type **java.lang.OutOfMemoryError**. Pensez à tenir compte de cette problématique, en comprenant bien qu'elle n'apparaît que si votre connexion réseau est utilisée pour un grand nombre de transfert d'objets.

Conclusion

Nous avons donc, au terme de cette section, vu ce que propose le concept de sérialisation. Nous avons présenté les différents problèmes inhérents à ce concept. Ce qu'il faut simplement en retenir, c'est que l'environnement Java prend tout à sa charge par l'intermédiaire de la réflexion.

Deux classes principales vous permettent de mettre en oeuvre cette sérialisation : **java.io.ObjectInputStream** et **java.io.ObjectOutputStream**. Mais attention : pour qu'un objet puisse être manipulé sur un flux d'objet, il faut qu'il soit marqué comme étant

sérialisable. Pour ce faire il se doit d'implémenter l'interface **java.io.Serializable** : les classes du J2Sdk implémentent quasiment toutes cette interface.

Enfin, une problématique existe : si vous ne gérez pas correctement vos flux, des débordements de mémoires sont envisageables.

Exercices

Enoncé 1 : Ecrire un algorithme qui permet la saisie d'un montant HT d'une facture, puis de calculer et afficher un total TTC qui est égal au montant HT + TVA selon les conditions suivantes :

- Si le montant HT est inférieure a 5000, alors la TVA est égal à 7%.
- Sinon si le montant HT est entre 5000 et 10000, alors la TVA est égal à 14%.
- Sinon si le montant HT est supérieur a 10000, alors la TVA est égal à 20%.

Enoncé 2 : Ecrire un algorithme qui permet de saisir un nombre au clavier puis de déterminer s'il est premier ou non.

Règle : on dit qu'un nombre est premier s'il n'est divisible que par 1 et lui-même.

Enoncé 3 : Ecrire un algorithme qui permet de ranger dix nombres dans un tableau T d'une dimension, puis les afficher avec leur somme et leur moyenne.

Enoncé 4

Ecrire un algorithme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message : « Plus grand ! », et inversement, « Plus petit ! » si le nombre est inférieur à 10.

Enoncé 5 :

Ecrire les programme qui permettent de calculer les sommes suivantes, en utilisant les schémas : pour, tant que et répéter :

- S1= 3+6+9+…+n
- S2= 2+4+6+…+n
- S3=1 !+2 !+3 !+….+n !
- S4=1/1 !+1/2 !+1/3 !+…+1/n !
- S5=n !/((p !*(n-p) !)

Enoncé 6 :

Quelles sont les valeurs de a et i affichées par l'algorithme suivant:

```
entier a,i;
début
        aÅ  0; iÅ 0;
Répéter
 i        Å  i+1;
 a        Å  a+i;
 i        Å  2*i+1;
jusqu'à i >=10;
écrire(a,i);
fin
```

Enoncé 7 :

Réaliser la fonction « Nvoy ()» qui reçoit une chaine de caractères en paramètre et renvoie le nombre de voyelles contenues dans cette chaine. Une voyelle fait partie des caractères suivant : 'a', 'e', 'i', 'o', 'u', 'y'

Enoncé 8 :

Etablir un algorithme qui permet de saisir les éléments d'un tableau de 20 réels; et affiche le nombre des valeurs positives, le nombre des valeurs négative, le nombre des valeurs nulles.

L'algorithme fait appel à deux procédures.

Enoncé 9 :

Etablir un algorithme qui permet de trier les éléments d'un tableau de 20 réels; et calcule et affiche la moyenne de ses éléments.
L'algorithme fait appel à une procédure trie et une fonction moyenne.

Réaliser sur PC les algorithmes demandés dans les exercices suivants en utilisant le java :

1) **Compter de 5 en 5**
   Ecrire un programme qui compte de 5 en 5 de 0 jusqu'à 100 puis affiche la somme des nombres trouvés.

2) **Minimum, maximum et somme**
   Ecrire un programme qui place 3 nombres passés en paramètres dans un tableau, puis calcule et affiche le minimum, le maximum et la somme de ces nombres.

3) **Mois**
   Ecrire un programme qui lit le nombre entier entre 1 et 12 passés en paramètre et qui affiche le nom du mois correspondant.
   On pourra utiliser un tableau ou l'instruction switch.

Ex1:

Concevoir un programme permettant de calculer le salaire net d'un employé sachant que ce dernier et constitué des valeurs suivantes :
Le salaire de base (SB), une prime (P), une retenue (R) tel que :

SB= nombre d'heures * taux horaire
P= taux de prime *SB
R= taux de retenue*SB

EX2:

Ecrire un algorithme qui permet de remplir un tableau M de deux dimensions (trois lignes et cinq colonnes) avec les nombres de 1 à 15.

1 2 3 4 5			
6 7 8 9 10			
11 12 13 14 15			

Ex3

Quel résultat produira cet algorithme ?

```
Début
Tableau T(3, 1) en Entier
Variables k, m, en Entier
Pour k  0 à 3
   Pour m  0 à 1
      T(k, m)  k + m
   m Suivant
k Suivant
Pour k  0 à 3
```

Pour m 0 à 1

Ecrire T(k, m)

m **Suivant**

k **Suivant**

Fin

Ex4

Ecrivez un algorithme qui trie un tableau dans l'ordre décroissant.
Vous écrirez bien entendu deux versions de cet algorithme, l'une employant le tri par sélection, l'autre le tri à bulles.

Ex 5

Le président d'un Club sportif a soumis chaque joueur du club à un questionnaire, lui demandant de citer trois de ses coéquipiers qu'il juge les plus utiles pour le club. au point de vue motivation. On propose de stocke les réponses dans des tableaux.
Les tableaux doivent contenir comme informations

- Le nom du joueur ayant répondu au questionnaire (le nombre des joueurs est 31) ;
- Les noms des trois joueurs choisis par le précédent.

Travail à faire

1. Déclarer les tableaux nécessaires pour l'application

Ecrire les algorithmes permettant de :

2. Saisir le nom et: les réponses de chaque joueur.
3. Chercher et afficher les joueurs ayant choisi un Joueur donné, dont on saisit le nom au clavier.
4. Trier la liste des joueurs par ordre alphabétique
5. Calculer dans un nouveau tableau pour chaque joueur le nombre de votes qu'il a reçus.
6. Trier la liste des joueurs par ordre décroissant du nombre de votes reçus.
7. Afficher la liste des joueurs qui ont reçu plus que 10 votes (Nom joueur. Nombre de votes).