




***Votre avenir  
nous engage***

***Découverte de la  
programmation en  
Java  
Support de formation***

## Objectif

- Ce document ne nécessite aucun pré requis en algorithmique ou en langage informatique : il guide le stagiaire dans la découverte des notions de base de la programmation, en s'appuyant sur le langage Java
- Il doit être utilisé en complément d'un cours d'algorithmique
- Il ne fait pas double emploi avec un cours complet sur le langage Java, ou la programmation objet
- Documents associés :
  - **Algorithmique Support de Formation AFPA** : propose une progression avec des durées indicatives, pour l'apprentissage de l'algorithmique.
  - **Algorithmique Cours AFPA** : référence du pseudo langage auquel on se réfère pour présenter les structures algorithmiques du langage Java
  - **Le livre de Java premier langage : pour les vrais débutants en programmation**. Anne Tasso, aux éditions Eyrolles. Certains exercices et thèmes sont repris ou s'inspirent de ce livre. Le symbole  y renvoie toujours. La deuxième édition de ce livre (**Le livre de Java premier langage 2<sup>ème</sup> édition**, même auteur, même éditeur) sera référencé avec les numéros de pages suivis d'un astérisque. Exemple:



« Exécuter un programme » : pp. 14-17 ou pp. 19-23\*

indique qu'il faut étudier les pages 14 à 17 du livre dans sa première édition, ou les pages 19 à 23 dans sa deuxième édition.

- Pour préciser certains points du langage Java : consulter **Pour Commencer avec Java**, document Afpa qui va déjà loin dans l'étude du langage !

## Mode d'emploi

Les chapitres comportent généralement 4 parties :

- les objectifs
- les points clés des thèmes abordés
- des exemples de base, testés dans l'environnement Sun et Microsoft
- des énoncés d'exercices

### Notations utilisées dans la description des points clés



Travail bibliographique à effectuer indiquant les références des pages à lire, dans « Le livre de Java premier Langage ». Des questions vous permettront de vérifier si vous avez compris l'essentiel.



Exercice pratique de programmation, en langage Java.



Point important qui mérite d'être souligné



A éviter à tout prix !

Document AFPA

# CHAPITRE 1

## Développement de programme en Java

### Objectifs

Découverte du développement d'applications en Java : dans un environnement intégré (Eclipse), et en mode texte à l'aide du JDK Sun.

Notons que vous pouvez avoir un autre environnement intégré que Eclipse ( mais celui-ci est complet, ouvert et évolutif grâce à la possibilité d'ajouts de nombreux Plug-in ). Attention dans leur version d'évaluation certains environnements ( JBuilder par exemple ) ne donnent pas la possibilité de développer des applications pour la console DOS. Nous ne pourrons pas utiliser ces environnements pour la programmation de nos algorithmes.

Un environnement de développement intégré permet de profiter d'un certain nombre d'outils d'écriture du code, de test et de mise au point. Editeur colorisé, vérification de syntaxe en ligne, mise en place de la charpente d'instructions complexes, débogueur, ...

L'environnement de Sun a deux avantages : il est gratuit, et c'est une référence. Par contre il n'offre pas (dans sa distribution gratuite) d'environnement de développement intégré (EDI). Nous sommes alors obligé d'utiliser un éditeur de texte ( bloc note par exemple ).

### Points clé



#### « Naissance d'un programme » : pp. 9-17 ou pp. 13-23\*

- Sans s'attacher aux détails de la syntaxe Java, repérer la structure générale d'un programme, distinguer le point d'entrée du programme, les commentaires, les instructions d'entrée-sortie (*System.out.print* et *Lire.d* ...). Ces points seront exposés dans le chapitre suivant.
- *Quel est l'intérêt du byte-code et de la machine virtuelle Java par rapport à la technique de compilation et d'édition de lien utilisée dans les langages classiques ?*



#### Compilation et exécution avec le JDK Sun

- La démarche est exactement la même que sous Unix (voir « Le Livre de Java premier langage », p. 16)
- Recopier le fichier *Lire.Java* dans votre répertoire de travail : ce fichier simplifiera l'écriture des premiers programmes Java en fournissant des fonctions de lecture clavier : *Lire.d*, *Lire.i*...
- Compiler le fichier *Lire.java*

- D'extension .java, les fichiers sources doivent avoir impérativement le même nom que la classe **public** qu'ils contiennent : avec le Bloc-Notes, entrez le programme *Cercle* (p. 13 ou pp. 17-18\*) dans un fichier *Cercle.java*.

- Dans une fenêtre « Invite de Commande », compiler ce fichier par la commande *javac* :

```
javac Cercle.java
```

- Vérifier que votre répertoire de travail contient les deux classes issues de la compilation : *Lire.class*, et *Cercle.class*
- Exécuter le programme grâce à la machine virtuelle Java (tenir compte des minuscules et majuscules dans le nom de la classe)

```
java Cercle
```

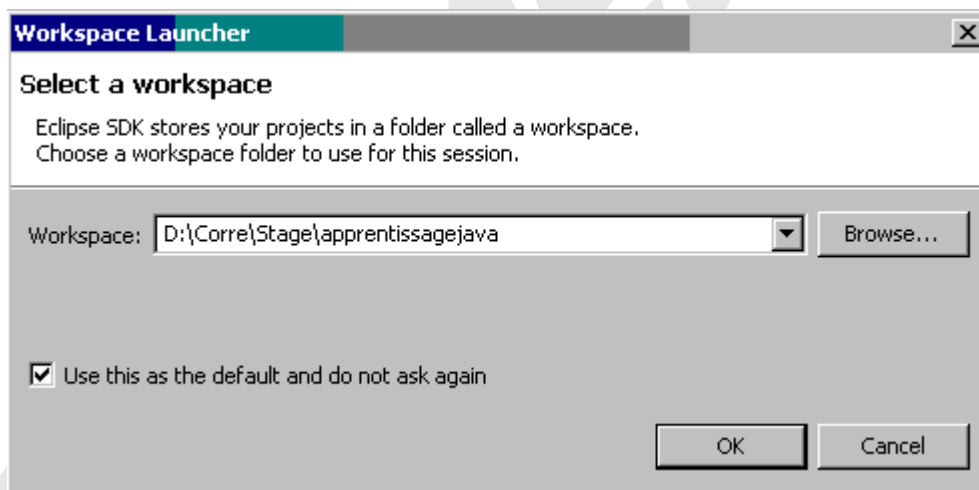


## Compilation et exécution avec Eclipse

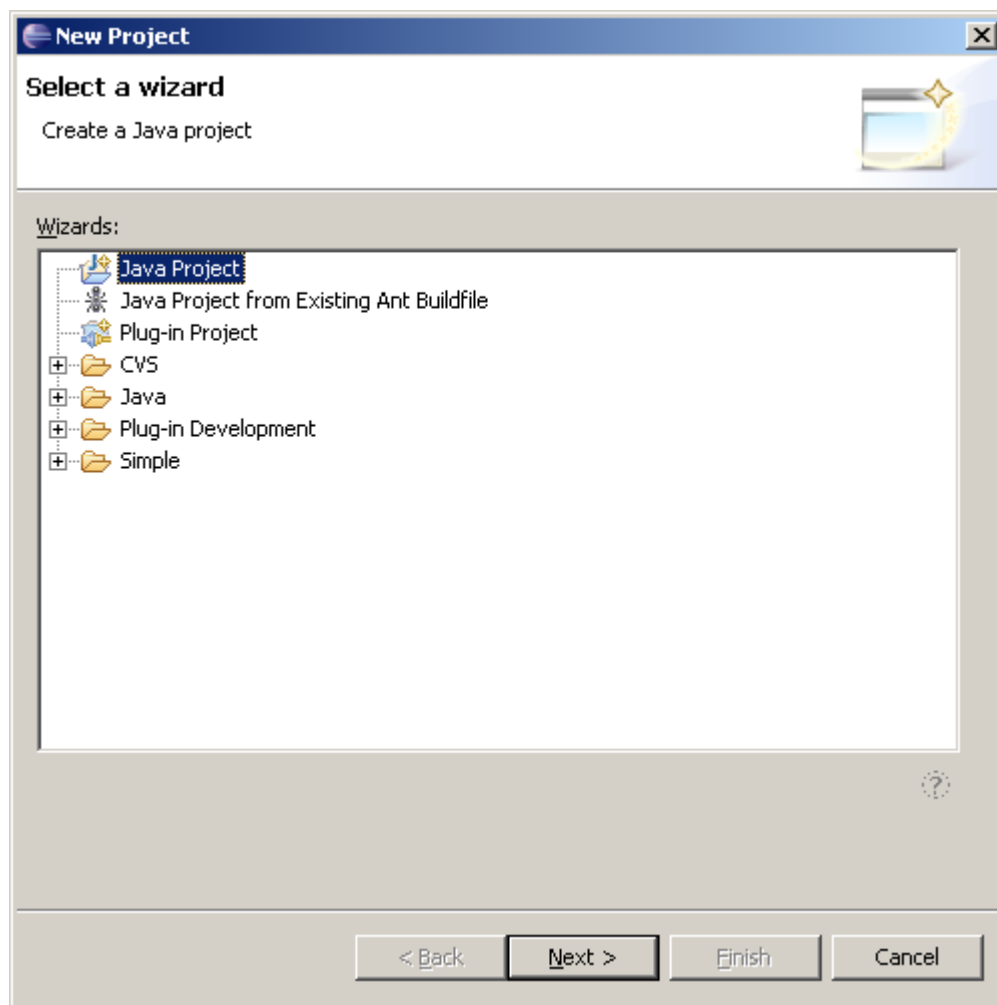
- 1° Lancer l'environnement *Eclipse* à partir de l'icône de votre bureau.



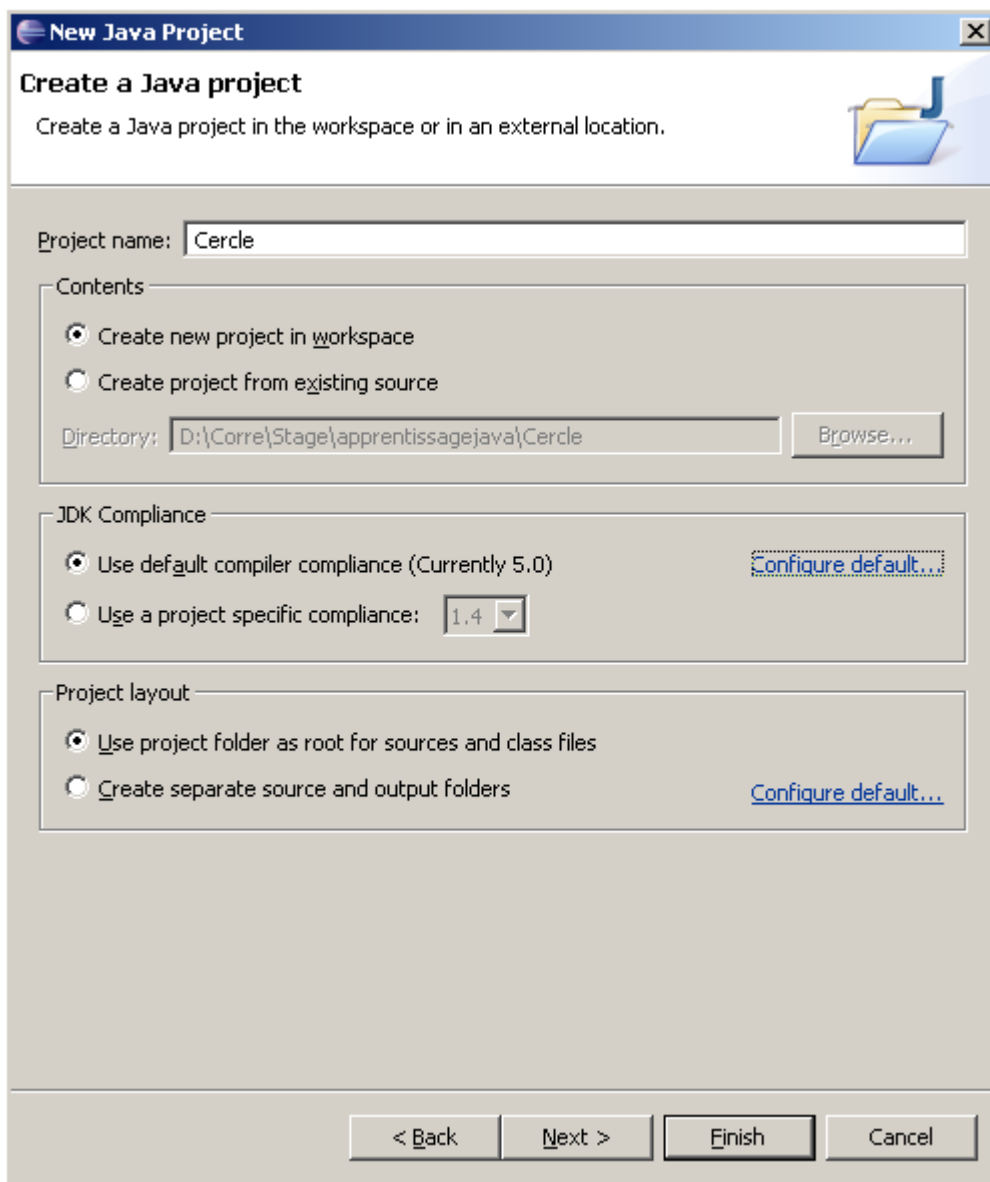
- 2° Définir l'espace de travail sur votre disque utilisateur ( Workspace ).



- 3° Créer un nouveau projet (menu « *File* »/ « *NEW* »/ « *Project* »),



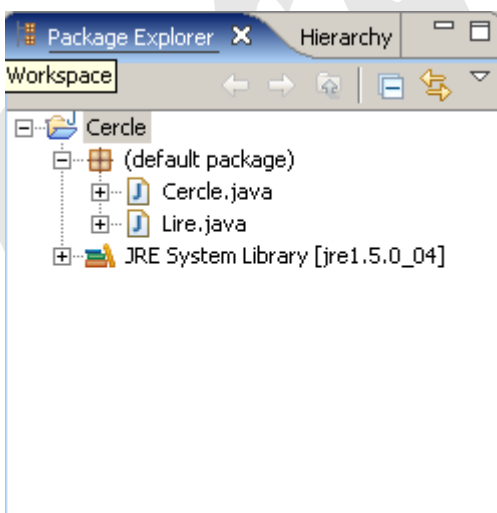
choisir un Java Project.



lui donner un nom (*Cercle*) et modifier la version de compilateur par défaut ( la plus évoluée ).

Appuyez sur le bouton Finish. Un nouveau projet appelé Cercle est créé.

4° Copier les fichiers fournis en exemple dans le répertoire Cercle sur le projet ( Lire.java et Cercle.java ).



⇒ Dans la fenêtre « *Package Explorer* », cliquer sur le nom du projet (*Cercle*) Puis double cliquer sur le nom du fichier *Cercle.java* pour l'ouvrir.

⇒ Pour créer une classe supplémentaire on utiliserait alors l'icône 

```
/* Classe      : Cercle
   Auteur      : Eyrolles, revu par Lécu Régis
   Mise à jour : 12 février 2001
   Fonction    : ce programme calcule le périmètre et la surface
                 à partir de son rayon
*/
public class Cercle
{

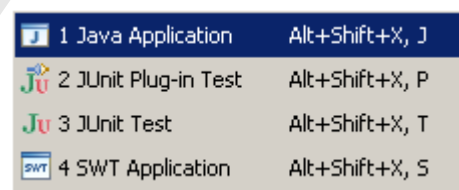
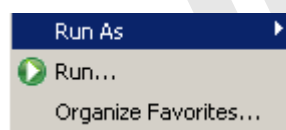
    public static void main(String [] argument)
    {
        double rayon ;      // rayon en m
        double perimetre;    // périmètre en m

        // Etape 1 : lecture du rayon
        System.out.print("Valeur du rayon : ");
        rayon = Lire.d();

        // Etape 2 : calcul et affichage du périmètre
        perimetre = 2 * Math.PI * rayon;
        System.out.println ();
        System.out.println(" Le cercle de rayon "+ rayon);
        System.out.println("      a pour perimetre : "+ perimetre);

    }
}
```

5° Il vous suffit maintenant d'exécuter l'application *Cercle*. Sa première exécution de fera en utilisant l'icône , mais attention la flèche noire seulement .




Voici l'exécution attendue du programme :

```
Valeur du rayon : 45
|
Le cercle de rayon 45.0
      a pour perimetre : 282.7433388230814
```





Pour des exécutions successives de la même application vous utiliserez  , mais la flèche blanche sur fond vert.



## Compatibilité entre les deux environnements

Pour illustrer l'intérêt du « byte code », lancer le fichier *Cercle.class* qui est issu de la compilation dans l'environnement Microsoft, avec la machine virtuelle JAVA de Sun :

```
java Cercle
```

Vérifier que le code s'exécute de la même manière que dans l'environnement intégré.



## Votre premier développement « sans filet »

- En vous inspirant directement du code fourni, rajouter une fonctionnalité de calcul et d'affichage de la surface du cercle.
- Afficher la fonctionnalité et la version du programme, dès son démarrage :

```
**** P rim tre et surface du Cercle (V1.0, 12/01/01) ****
```

On conservera cette saine habitude dans tous les d veloppements suivants, pour  viter de mettre au point avec le code d'un autre programme, ou avec le code du mois pr c dent !

- A rer la pr sentation du r sultat de l'ex cution du programme, en ins rant des retours   la ligne :  
*System.out.println()*

*Note : le fichier Lire.java est une bo te   outils qui contient un certain nombre de fonctions permettant de lire des informations simples au clavier. En cas de mauvaise saisie, ces fonctions arr tent le fonctionnement du programme en affichant un message d'erreur.*

*La fonction c de la bo te   outils Lire permet de lire un caract re. Exemple :*  
*char car = Lire.c() ;*

*La fonction i permet de lire un entier au clavier, d permet de lire un double, etc...*

*Ces fonctions ont  t  programm es pour que les saisies au clavier ne puissent pas avoir d'influence entre elles. C'est   dire que si on lit un caract re, et que l'utilisateur en frappe trois, valid s par la touche Entr e, les caract res non lus dispara tront avant la prochaine saisie.*

## CHAPITRE 2

### LES TYPES DE BASE ET LES ENTREES/SORTIES

#### Objectifs

Ce chapitre reprend de façon systématique ce qui a été observé en réalisant le programme *Cercle* :

- déclaration et utilisation des variables.
- saisies clavier et les affichages écran de base.

#### Points clé

##### *Structure d'un programme Java*

- Un programme Java de base consiste en une **classe** qui contient au minimum une **fonction** de nom réservé **main** : ce nom désigne à la machine virtuelle Java, le « point d'entrée » (début) du programme.

```
public static void main ( String [] args )
```

Nous verrons plus tard ce que signifie tout le décorum autour de main. Pour l'instant nous nous contenterons de le recopier à l'identique ( car c'est obligatoire ).

- Comme toutes les fonctions, le main regroupe une suite d'instructions, entre un marqueur de début { et un marqueur de fin }

##### *Les commentaires*

- Un commentaire est un texte libre ignoré par le compilateur, qui améliore la lisibilité du programme
- 2 types de commentaires :

façon C :     /\* <- marque début du commentaire  
                  le commentaire peut tenir sur plusieurs lignes  
                  bla bla ...fin du commentaire -> \*/

façon C++ :   // la fin de la ligne est ignorée ....



Nous utiliserons les commentaires // dans les programmes pour commenter des instructions précises (micro commentaires), ou pour commenter la structure de notre programme ( macro commentaires).

Nous utiliserons les commentaires /\* et \*/ pour mettre en commentaire des zones de programmes, par exemple en phase de recherche d'erreurs, ou de mise au point. Ceci permet alors de mettre des

commentaires en commentaire ( cette opération ne peut exister que lors d'une recherche pointue de problème. En aucun cas il ne peut subsister de commentaires en commentaires dans une version définitive de logiciel).

Si nous ne respectons pas cette consigne, nous serons gênés par les commentaires pour mettre en commentaire une zone de code.

## La déclaration des variables

- Une variable doit être déclarée avant d'être utilisée dans un programme : il faut donner son nom et son type.

Syntaxe : `nom_type nom_variable ;`

- Un type définit l'ensemble des valeurs que peut prendre la variable ainsi que l'ensemble des opérations que l'on peut effectuer dessus.
- Exemples :

```
char car ;           // variable de type caractère : UNICODE sur 2 octets
int entier ;         // entier signé sur 4 octets
long grandEntier ;   // entier signé sur 8 octets
float leReel ;        // réel en notation virgule flottante (mantisse, exposant)
double doublePrecision ; // idem en double précision
```

..etc...

Vous noterez ici que les variables commencent toutes par une minuscule, et ne comportent que des lettres, avec un changement de casse à chaque nouveau mot. Exemple :

ageCapitaine ( et pas age\_capitaine, ou AgeCapitaine )

Pour les fonctions la norme est la même. Pour des problèmes de compatibilité avec le livre de Anne TASSO, les fonctions de la boîte à outils Lire ne respectent pas cette convention. Nous jetterons donc un voile pudique sur les fonctions S, Attente, ... et nous respecterons la norme partout ailleurs.

Les constantes auront un nom en majuscule.

```
final int LIMITE = 45 ;
```

Les classes, et boîtes à outils commencent par une majuscule. ( Lire, BoîteVitesse, Essai, ... )

N'oubliez pas que les noms doivent être significatifs, c'est à dire que le nom de la variable, fonction ou constante doit nous permettre de savoir à quoi elle sert.

- Une variable peut être initialisée lors de sa déclaration :

```
char car = 'r' ;      // car reçoit le caractère 'r' dès sa déclaration
int relatif = - 456 ;
```



« Stocker une information » : pp. 25-31 ou pp. 33-41\*

- Pourquoi la représentation des caractères en Unicode est-elle un progrès par rapport à la notation ASCII utilisée dans les autres langages ?

- Comment préciser au compilateur que l'on veut stocker le réel 3.14159 en double précision ou en simple précision ? Quel type de constante le compilateur Java alloue-t-il par défaut ?

## Utilisation des constantes

- En Java, les constantes sont déclarées et typées comme des variables ; mais leur valeur est fixée définitivement lors de la déclaration (en réalité lors de la première affectation (et la seule) qui s'avère être souvent la déclaration).

Syntaxe : **final** *nom\_type* *nom\_constant* = *valeur*;

- constantes entières décimales : désignées par leur valeur, sans notation particulière  
1515, -10

- constantes caractères : symbole du caractère entouré par des apostrophes  
'A', '1', '?'

- suite de caractères (**String**):  
"Bonjour Régis"



Noter l'utilisation des guillemets pour les String et des apostrophes pour les caractères

- constantes virgules flottantes :  
notation décimale 123.90 , notation avec exposant 12E-10

## L'instruction d'affectation

Syntaxe : **variable1 = variable2; // lire : variable1 prend pour valeur variable2**  
**variable1 = constante1;**  
**variable1 = 2\*i+3 ;**

range le contenu de variable2 (ou la valeur de constante1, ou le résultat du calcul) dans variable1 : le contenu de variable2 n'est pas modifié

```
char cardeb, carfin ;           // déclaration de 2 variables
cardeb = 'A';                   // cardeb contient maintenant le caractère A
carfin = cardeb;                 // copie de cardeb dans carfin : carfin contient aussi A
```



On ne peut affecter des variables qu'avec des variables ou des constantes de même type :  
pas d'éléphants avec les souris !

```
double f;
int i;
i = f; // affectation incorrecte
```

- Lorsque le compilateur ne peut effectuer une « conversion implicite » d'un type dans un autre, il faut lui indiquer explicitement par une « conversion de type » ou « casting » :

`i = (int) f ;        // prend la partie entière du réel f, et la stocke dans l'entier i`

- Encore faut-il que cela ait un sens : en règle générale, toujours réfléchir après une erreur de compilation qui cache souvent une erreur d'analyse ou de raisonnement. Ne pas saupoudrer avec des « castings » pour faire plaisir au compilateur !



« Stocker une information » : pp. 31-34 ou pp.42-45\*,  
« casting » : pp. 37-39 ou pp. 49-53\*

| exemple   Base1.java

## *Les opérateurs*

A chaque type de variables ou constantes est associée une liste d'opérateurs arithmétiques, logiques, relationnels possibles.



Comme en mathématiques, les opérateurs ont une priorité (ou précedence) dont il faut tenir compte pour écrire des expressions complexes : différence entre `a * (b + c)` et `a * b + c`

Mais, pour ne pas être dépendant d'un langage donné, et éviter les erreurs d'inattention, le mieux est de toujours parenthéser, en ne présupposant rien sur la priorité : écrire `(a * b) + c`



Attention à l'utilisation de l'opérateur d'égalité (`==` en Java) sur des réels (**float** ou **double**). Les calculs en « virgule flottante » sur des réels représentés par une mantisse et un exposant sont des approximations. Donc deux expressions à priori égales peuvent avoir des résultats différents en fonctions des approximations qui ont été faites pour les évaluer. La différence est minime, mais suffisante pour qu'il n'y ait plus égalité.



« Stocker une information » : pp. 35-38 ou pp. 45-49\*

## *Affichage d'une ou plusieurs variables*

La bibliothèque **System** qui est fournie en standard avec le langage Java contient les fonctions d'affichage **print** et **println** pour chaque type prédéfini du langage. La fonction **println** va à la ligne après l'affichage (print line).



« Communiquer une information » : pp. 49-53 ou pp. 65-70\*

| exemple   Base2.java

### Quelques caractères spécifiques pour les affichages (séquences d'échappement)

tabulation	TAB	'\t'	
carriage return	CR	'\r'	
backslash	\	'\\'	
single quote	'	'\''	(avec deux apostrophes collés)
Bell		'\a'	

---

### *Lecture au clavier d'une ou plusieurs variables*

---

La bibliothèque **System** ne contient que des fonctions de lecture bas niveau (**read**) permettant de lire un ou plusieurs octets au clavier, sans effectuer de conversion. On utilisera donc une bibliothèque de fonctions qui réalisent ces conversions, pour faciliter l'écriture de nos programmes (listing de la bibliothèque fourni en Annexe). N'essayez pas de comprendre leur fonctionnement ( nous aurons tous les éléments plus tard ), mais seulement comment les utiliser.



« Communiquer une information » : pp. 54-56 ou pp. 71-75\*

Vous noterez dans les exemples qui suivent que les commentaires sont des commentaires techniques qui expliquent ce que fait une instruction. Dans les programmes ( comme vous le verrez dans les prochains exemples ) vous trouverez deux sortes de commentaires. Les commentaires fonctionnels qui expliquent ce que fait le programme, puis les grandes étapes pour expliquer comment nous arrivons à le faire. Nous trouvons également les commentaires techniques qui expliquent précisément une ou plusieurs instructions pour préciser le travail qui a été accompli.

## Exemples

```
/******
Programme      :  Basel.java
Auteur        :  Lécu Régis
Mise à jour   :  février 2001, maj nov 2003 jcc
Fonction      :  démo sur les variables, constantes, opérateurs
                  affectations et castings
******/

public class Basel
{
    public static void main (String [] argument)
    {
        // Partie déclarative
        long   chiffreAffaire;           // variable de type entier signé sur 8 octets
        int    factureHT ;               // variable de type entier signé sur 4 octets
        int    factureTTC;
        final float  tauxTVA = 1.33F; // constante de type réel, double précision
                                   // (=> mot clé final et emploi du F derrière la constante)

        // Partie exécutive
        System.out.println ("**** Demo variables, constantes, affectation ****");
        System.out.println ();

        chiffreAffaire = 100000000 ;
        // affectation de la valeur 100000000 dans chiffreAffaire
        // et affichage
        System.out.println ("Chiffre d'affaire initial : " + chiffreAffaire);

        factureHT = 10000;
        System.out.println ("Facture Hors Taxe          : " + factureHT);

        // Noter l'emploi du casting pour convertir le résultat en int
        factureTTC = (int) (factureHT * tauxTVA) ;

        chiffreAffaire = chiffreAffaire + factureTTC ;
        // Ajout de la facture au chiffre d'affaire
        System.out.println ("Chiffre d'affaire final   : " + chiffreAffaire);

        System.out.println();
        System.out.println("Tapez sur Entree pour Terminer ");
        Lire.c();
    }
}
```

```

/*****
Programme      : Base2.java
Auteur       : Lécu Régis
Mise à jour  : février 2001, maj nov 2003 jcc
Fonction     : démo sur les opérateurs, affichages
*****/

public class Base2
{
    public static void main (String [] argument)
    {
        System.out.println ("**** Demo operateurs, affichages ****");
        System.out.println ();

        // En java, on peut déclarer les variables locales n'importe où dans un
        // bloc, au plus près de leur utilisation
        int nombre = 10;
        int divEntier = 3;
        float divReel = 3.0F;

        // Utilisation de la division entière et réelle
        System.out.print ("Division entiere de " + nombre + " par ");
        System.out.println (divEntier);
        System.out.print ("Quotient = ");
        System.out.print (nombre / divEntier);
        System.out.print (" Reste = ");
        System.out.println (nombre % divEntier);
        System.out.println (".....");
        System.out.println ();

        System.out.println ("Division reelle de " + nombre + " par " + divReel);
        System.out.println ("Quotient = " + (nombre / divReel));
        System.out.println (".....");
        System.out.println ();

        // Affichage de quelques entiers en décimal, hexadécimal et octal
        long entier = 65535;
        short hexa = 0xFF;
        byte octal = 077;

        System.out.print ("Entier = ");
        System.out.println (entier);
        System.out.print ("Hexa = ");
        System.out.println (hexa);
        System.out.print ("Octal = ");
        System.out.println (octal);
        System.out.println (".....");
        System.out.println ();

        // Saisie d'un caractere et affichage de sa valeur ASCII ou UNICODE
        char car;
        System.out.print ("Entrez un caractere :");
        car = Lire.c ();

        // On réaffiche d'abord la valeur du caractère puis la valeur de son
        // code en "castant" le caractère en int
        System.out.print ("Le caractere " + car + " a pour valeur UNICODE :");
        System.out.println ((int) car);

        System.out.println();
        System.out.println("Tapez sur Entree pour Terminer ");
        Lire.c();
    }
}

```



## Exercices

- Ecrire un programme qui saisit un code Unicode en décimal et affiche le caractère correspondant. Exemple la saisie de l'entier 65 donne le caractère « A ».
- Afficher les caractères de contrôle de tabulation, beep, retour en début de ligne, entre deux étoiles, pour voir leur effet au niveau de l'affichage. Ces caractères vous seront utiles pour réaliser une présentation simple en mode texte.

## CHAPITRE 3

### Instructions conditionnelles et alternatives

#### Points clé

#### Structure algorithmique conditionnelle

Français : **S**'il se mettait à pleuvoir, je prendrais un parapluie

Syntaxe en pseudo-langage :

```
Si <condition> Alors  
  <instruction 1>  
  .....  
  <instruction N>  
Finsi  
  <instruction N+1>
```

- le mot clé **Alors** sépare la condition de la première action conditionnelle
- <instruction 1> ..... <instruction N> ne sont exécutées que si <condition> est vraie
- le mot clé **Finsi** marque la fin de la portée de la condition :  
<instruction N+1> sera toujours exécutée.



Respecter l'indentation, qui permet de voir facilement la portée de la condition

Exemple en pseudo-langage :

```
lire (entier)  
Si entier mod 2 = 0 Alors  
  écrire ("Entier pair")  
Finsi
```

Java :

```
if ( condition )  
{                               // début de portée de la condition  
  instruction1;  
  instruction2;  
  ...  
  instructionN ;  
}                               // fin de portée de la condition
```

exemple If1.java

## Conditions imbriquées

Français : S'il se mettait à faire beau, j'irais promener mon chien et alors si je rencontrais Pierre dans ma promenade, je lui poserais des questions sur Java.

Exemple en pseudo-langage :

```
lire (entier)
Si entier mod 2 = 0 Alors
    écrire ( "Entier pair")
    Si entier mod 4 = 0 Alors
        écrire ( "Multiple de 4")
    Finsi
Finsi
```

Java :

```
if ( condition1 )
{
    instruction1;
    if ( condition2 )
        instruction2;
    if ( condition3 )
    {
        instruction3;
        ...etc...
        instructionX;
    }
    instructionY;
}
// début de portée de condition1
// pas d'accolades obligatoires pour une seule instruction
// début de portée de condition3
// exécutées si condition1 et condition3 sont vraies
// dépend de condition1 mais pas de condition3
// fin de portée de condition1
```

exemple If2.java



Quand vous écrivez une condition qui comporte une seule instruction, nous vous conseillons de toujours mettre des accolades.

Ceci augmente la lisibilité de vos programmes.

Cela évite des problèmes en cas d'ajouts d'instructions dans la condition à posteriori.

Exemple :

// version 1 complètement illisible, mais correcte

```
if (entier % 2 == 0)
{System.out.println(" Entier Pair ") ;
if (entier % 4 == 0)
    System.out.println(" Multiple de 4 ") ;}
```

```
// version 2 lisible et correcte

if (entier % 2 == 0)
{
    System.out.println(" Entier Pair ") ;
    if (entier % 4 == 0)
        System.out.println(" Multiple de 4 ") ;
}

// version3 très lisible, correcte et conseillée

if (entier % 2 == 0)
{
    System.out.println(" Entier Pair ") ;
    if (entier % 4 == 0)
    {
        System.out.println(" Multiple de 4 ") ;
    }
}
```

### Structure alternative "Si ... alors .... sinon ..."

Français : S'il se mettait à faire beau, j'irais promener mon chien, sinon je resterais chez moi à lire un ouvrage de Java

Exemple en pseudo-langage :

```
lire (car)
Si car >= 'A' et car <= 'Z' Alors
    écrire ( "C'est une Majuscule" )
Sinon
    écrire ( "Ce n'est pas une Majuscule")
Finsi
```

Java :

```
if ( condition )
{
    instruction1;          // exécutées si condition est vraie
    ...etc....
    instructionX;
}
else
{
    // exécutées si condition est fausse
    instruction2;
    ...etc...
    instructionY;
}
```

exemple If3.java

## Choix

la condition porte sur la valeur d'une seule variable scalaire : entier ou caractère. Selon cette condition le programme doit traiter telle ou telle chose. Cette structure permet de voir au même niveau un certain nombre de traitements

Conseil : cette structure n'est à utiliser que lorsque les cas sont exclusifs entre eux.

Java :

```
switch ( variable_scalaire )
{
    case valeur1:  action1;
                  break; // fin du premier choix

    case valeur2:  action2;
                  ...etc..
                  actionN;
                  break; // fin du deuxième choix

    default       : actionX; // si différent de toutes les valeurs testées
}
```

exemple ChoixMul.java

## Alternatives multiples

Conseil : ce cas correspond à un choix, soit sur des plages de valeurs (  $3 < i < 17$  ) et non pas sur des valeurs discrètes ( 3 et 5 et 17 ), soit sur des valeurs qui ne sont pas scalaires ( chaînes, doubles, ... ). Dans tous les cas nous avons des valeurs, ou des plages de valeurs discriminantes.

Dans les cas de combinatoire classiques utilisez les « if » imbriqués

Pseudo langage :

```
lire (car)
Si car >= 'A' et car <= 'Z' Alors
    écrire ( "C'est une Majuscule")

sinon si car >= 'a' et car <= 'z' alors
    écrire ("C'est une minuscule")

sinon si car = '.' ou car = ',' alors
    écrire ( "C'est un caractère de ponctuation")
sinon
    écrire ( "Caractère d'un autre type")
finsi
```

Java :

```
if ( condition1 )
{
    instruction1 ;
}
else if ( condition2 )
{
    instruction2;           // exécutées si (non condition1) et condition2
    ...
    instructionX;
}
else
{
    instructionY;           // exécutées si toutes les conditions sont fausses
}
```

exemple If4.java

exemple Tempera.java



**A lire en complément : « Faire des choix » : pp. 61-80 ou pp.79-98\***

Dans les exemples qui suivent, notez le bloc de commentaires en entête de fichier. Il décrit ce que fait le programme. C'est bien sûr indispensable dès que l'on doit gérer quelques fichiers. ( Donc c'est indispensable tout court )

## Exemples

```
/******
Programme      : If1.java
Auteur        : Lécu Régis
Mise a jour   : février 2001, maj nov 2003 jcc
Fonction

- Programme de démonstration illustrant la structure algorithmique
CONDITIONNELLE : "si la condition est réalisée alors on fait l'action"

- Le programme lit un entier au clavier, et teste si celui-ci est un
multiple de 2, de 3, et s'il s'agit d'un nombre négatif. Lorsqu'une
condition est réalisée, le programme affiche un message d'information.
*****/

public class If1 // notez la majuscule au nom de la classe
{
    public static void main (String [] argument)
    {
        int entier;

        System.out.println ("*** If consecutifs ***" );
        System.out.println ();

        System.out.print( "Entier a tester ?" ) ;
        entier = Lire.i();

        if (entier % 2 == 0)
        {
            System.out.println ("C'est un entier pair") ;
        }

        if (entier % 3 == 0)
        {
            System.out.println ("C'est un multiple de trois" );
        }

        if (entier < 0)
        {
            System.out.println ("C'est un nombre negatif " );
        }

        System.out.println ();
        System.out.println ("Tapez sur Entree pour Terminer");
        Lire.c();
    }
}
```

```

/*****
Programme      : If2.java
Auteur       : Lécu Régis
Mise a jour  : février 2001, maj nov 2003 jcc
Fonction
- Illustre la Structure CONDITIONNELLE avec imbrication de if
- Ce programme reconnaît les nombres pairs, et les multiples
  de 4 : un nombre ne pouvant être multiple de 4 sans être pair, on ne teste
  la divisibilité par 4 que si le nombre est pair (-> if imbriqués )
*****/
public class If2
{
    public static void main (String [] argument)
    {
        // notez que la présentation met en évidence la structure du pg

        int entier;

        System.out.println ( "*** If imbriqués ***");
        System.out.println ();

        System.out.print( "Entier a tester ?") ;
        entier = Lire.i();

        if (entier % 2 == 0)          // début de la portée du premier if
        {
            System.out.println ("C'est un entier pair");

            if (entier % 4 == 0)      // if imbriqué
            {
                System.out.println ("C'est aussi un multiple de quatre") ;
            }

        }                          // fin de la portée du premier if

        System.out.println ();
        System.out.println ("Tapez sur Entree pour Terminer");
        Lire.c();
    }
}

```



```

/*****
Programme      : If3.java
Auteur        : Lécu Régis
Mise a jour   : février 2001, maj nov 2003 jcc
Fonction      :

- Illustre la Structure alternative if ... else
- Ce programme lit un caractère au clavier, et teste si ce caractère est
  une lettre majuscule, ou un autre type de caractère ...
*****/

public class If3
{
    public static void main (String [] argument)
    {
        char car;

        System.out.println ("**** If .. else ****" );
        System.out.println ();

        System.out.print ("Caractere a tester : " );
        car = Lire.c() ;

        if ( (car >= 'A')  &&  (car <= 'Z') )
        {
            System.out.println ("Ce caractere est une Majuscule !");
        }
        else
        {
            System.out.println ("Ce caractere n'est pas une Majuscule !");
        }

        System.out.println ();
        System.out.println ("Tapez sur Entree pour Terminer");
        Lire.c();
    }
}

```

```

/*****
Programme      : If4.java
Auteur        : Lécu Régis
Mise a jour   : février 2001, maj nov 2003 jcc
Fonction      :

- Illustre la Structure alternative if...else if...else if...else
- Ce programme lit un caractère au clavier, et teste si ce caractère est
  une lettre majuscule, une lettre minuscule, un chiffre, un signe de
  ponctuation ou un autre type de caractère ...
*****/

public class If4
{
    public static void main (String [] argument)
    {
        char car;

        System.out.println ("**** If...else if...else... ****" );
        System.out.println ();

        System.out.print ("Caractere a tester : " );
        car = Lire.c() ;

        if ( (car >= 'A') && (car <= 'Z') )
        {
            System.out.println ("Ce caractere est une Majuscule" ) ;
        }
        else if ( (car >= 'a') && (car <= 'z') )
        {
            System.out.println ("Ce caractere est une minuscule" ) ;
        }
        else if ( (car >= '0') && (car <= '9') )
        {
            System.out.println ("Ce caractere est un chiffre" ) ;
        }
        else if ( (car == '.') || (car == ',') || (car == ';') ||
                  (car == '!') || (car == ':'))
        {
            System.out.println ("Ce caractere est un signe de ponctuation" ) ;
        }
        else
        {
            System.out.println ("caractere d'un autre type !" ) ;
        }

        System.out.println ();
        System.out.println ("Tapez sur Entree pour Terminer");
        Lire.c();
    }
}

```

```

/*****
Programme      :  Tempera.java
Auteur        :  Lécu Régis
Mise a jour   :  février 2001
Fonction      :
Programme résumant les différents schémas conditionnels et alternatifs
*****/
public class Tempera
{
    public static void main (String [] argument)
    {
        char pluie, brouillard;

        System.out.println ("*** Meteo ***");
        System.out.println ();

        System.out.print ("Va-t-il pleuvoir ? (O pour oui) : " ) ;
        pluie = Lire.c();

        System.out.print ("Y aura-t-il du brouillard ? (O pour oui): " );
        brouillard = Lire.c();
        System.out.println();

        if (pluie == 'O')
        {
            if (brouillard == 'O')
            {
                System.out.println ("Bon je reste chez moi !");
            }
            else
            {
                System.out.println ("OK je prends mon parapluie ");
                System.out.println();

                short temperature;
                System.out.print ("Quelle temperature fera-t-il ? : ");
                temperature = Lire.s ();
                System.out.println();

                if (temperature < -20)
                {
                    System.out.println ("Je ne sors pas par " + temperature
                                         + " °C" );
                }
                else if (temperature < 5)
                {
                    System.out.println ("OK! Je viens avec un manteau " );
                }
                else if (temperature > 20)
                {
                    System.out.println ("Mais c'est le printemps !");
                }
                else
                {
                    System.out.println ("C'est moyen: je mets une veste");
                }
            }
        }
        else // il ne pleut pas
        {
            if (brouillard == 'O')
            {
                System.out.println ("Je viens avec une echarpe " );
            }
        }
    }
}

```

```

else
{
    // Noter l'utilisation de la variable locale soleil
    // (visible uniquement dans le bloc)
    char soleil ;
    System.out.print ("Fera-t-il du soleil ? (0 pour oui ) : " );
    soleil = Lire.c();
    System.out.println();

    if (soleil == '0')
    {
        System.out.println("Je prends mes lunettes noires!");
    }
    else
    {
        System.out.println("Beau temps pour les champignons!");
    }
}

System.out.println ();
System.out.println ("Tapez sur Entree pour Terminer");
Lire.c();
}
}

```

```

/*****
Programme      : ChoixMul.java
Auteur        : Lécu Régis
Mise a jour   : février 2001, maj nov 2003 jcc
Fonction      : Structure algorithmique "choix multiple"

Ce programme de démonstration lit un caractère au clavier,
et effectue plusieurs tests exclusifs par l'instruction "switch"
*****/

public class ChoixMul
{
    public static void main (String [] argument)
    {
        char car;    // caractère à tester

        System.out.println ( "*** Choix Multiples ***" ) ;
        System.out.println () ;

        System.out.print ( "Caractere a tester :" );
        car = Lire.c();

        switch (car)
        {
            case 'A':
            case 'F':    System.out.println ("C'est un A ou un F" ) ;
                        break;

            case 'Z':    System.out.println ("C'est un Z" ) ;
                        break;

            default :    System.out.println ("C'est un autre caractere" ) ;

        }

        System.out.println () ;
        System.out.println ("Tapez sur Entree pour Terminer");
        Lire.c();
    }
}

```

## Exercices

### **Exercice n° 1** : réalisation progressive d'une calculette

Faire la saisie puis l'affichage d'un nombre entier, en n'acceptant qu'un nombre compris entre -1000 et +1000.

Si l'utilisateur entre des valeurs hors norme, le programme forcera les valeurs à 0 et écrira un message d'erreur.

### **Exercice n° 2** : réalisation progressive d'une calculette

Faire la saisie de 2 nombres entiers, puis la saisie d'un opérateur '+', '-', '\*' ou '/'.

Si l'utilisateur entre un opérateur erroné, le programme affichera un message d'erreur.

Dans le cas contraire, le programme effectuera l'opération demandée (en prévoyant le cas d'erreur "division par 0"), puis affichera le résultat.

# CHAPITRE 4

## Les boucles

### Points clé

#### Utilité des boucles en algorithmique

- Nous avons vu différentes syntaxes permettant d'exécuter des actions, les unes après les autres ; ou de soumettre certaines actions à des conditions.
- Le langage décrit jusqu'à présent est trop pauvre, car il ne permet pas d'exprimer la répétition d'une même action.

*Exemple:* faire 10 fois "prendre un crayon et cocher une case".

Avec la grammaire que nous avons définie pour le moment, cela s'écrirait :

*Je prends mon crayon et je coche une case*

*.... codé 10 fois de suite ...*

*Je prends mon crayon et je coche une case*

#### Classification des boucles

- Il existe différents types de répétition, dans la vie courante. Dans le cas précédent, nous connaissions le nombre de cases à cocher.
- Comment exprimer notre algorithme dans le cas où le nombre de répétitions n'est pas connu à l'avance ? Par exemple : "cocher les cases d'une feuille, du haut en bas"

Deux catégories de boucles :

- **nombre de répétitions connu.**
- **nombre de répétitions inconnu :** la continuation dépend du résultat des actions effectuées dans la boucle.

## Boucle avec nombre de répétitions connu

Exemple de pseudo de principe pour des actions dépendant du nombre de passages :

- On veut cocher les cases paires et faire un rond dans les impaires, de la case numéro N à la case numéro M :

**Pour i de N à M faire**

Prendre le crayon

**Si i est pair alors**

Cocher la case i

**Sinon**

Dessiner un rond dans la case i

**Finsi**

**FinPour**

- Les actions comprises entre les mots clés **faire** et **FinPour** constituent le Corps de boucle : à chaque passage dans la boucle, ce corps de boucle s'exécute une nouvelle fois.
- La boucle fait parcourir à la variable entière i l'intervalle [N, M]
  - au premier passage dans la boucle,  $i = N$
  - à la fin de chaque boucle**, i est "incrémenté" (augmenté de 1)
  - la boucle s'arrête quand  $i = M+1$
  - la boucle s'exécute donc exactement  $M-N+1$  fois

Réalisation en Java :

```
for ( i = N ; i <= M ; i = i + 1 )      // i est incrémenté en fin de boucle
{                                       // début du corps de boucle
    ... suite d'instructions ....
}                                       // fin du corps de boucle
```



Surtout ne mettez pas de ; en fin de ligne du for ( avant { ). Cela signifie pour i parcourant l'intervalle [N,M] ne rien faire. Puis le programme exécute les instructions du bloc ( instructions comprises entre les accolades ) une seule fois.

```
for ( i = N ; i <= M ; i = i + 1 );    // attention c'est une erreur
{                                       // début du corps de boucle
    result = result + i ;
}                                       // fin du corps de boucle
```

à le même effet que :



```
i = M + 1 ;  
result = result + i ;
```

le point virgule en trop sur une boucle for peut donc avoir des effets surprenants sur les résultats attendus, voire conduire le programme à des débordements de tableau, ou autres catastrophes.

**exemple** For1.java

### Boucle avec nombre de répétitions inconnu

- La structure algorithmique précédente ne permet pas de cocher toutes les cases d'un questionnaire, si on ne connaît pas leur nombre à l'avance.
- On dirait en français :

```
je commence à la première ligne  
tant que je suis sur une ligne du questionnaire  
- je coche toutes les cases de la ligne en cours  
- je passe à la ligne suivante
```

que l'on peut décomposer en :

```
je commence à la première ligne  
tant que je suis sur une ligne du questionnaire  
- je commence à la première case de la ligne en cours  
- tant que je suis sur une case de la ligne en cours  
-- je coche la case en cours  
-- je passe à la case suivante  
- je passe à la ligne suivante
```

- La structure algorithmique utilisée est du type "Tant que ..." :

**TANT QUE la condition reste vraie, on refait la suite d'actions**

- Deux formulations en algorithmique et en Java :

1) la boucle peut s'effectuer de 0 à un nombre quelconque de fois : si la condition est fausse avant la boucle, on n'y rentre pas.

```
Tantque <condition> faire  
    <instruction1>  
    ....  
    <instructionN>  
fintantque
```

*On teste avant d'effectuer le corps de boucle*

Java :

```
while ( condition )
{
    ... suite d'actions .....
}
```



Ici aussi il faut faire attention à ne pas mettre un ; intempestif après la condition du while. Cela aurait la signification que tant que la condition est vraie, nous ne faisons rien. Dans le cas général cette condition n'évolue pas lors de son test, donc si la condition est vraie, nous ne faisons rien, mais comme la condition reste vraie, nous ne faisons toujours rien, et ainsi de suite. Cela s'appelle une boucle infinie, et cela mène à un programme qui reste bloqué et ne se termine jamais. Exemple :

```
bool = true ;
while ( bool == true) ;    // le programme restera bloqué sur l'exécution de cette ligne
{
    result = result + result
    if ( result > 10000 )
    {
        bool = false;
    }
}
```

**exemple** While1.java

2) Il faut boucler au moins une fois : même si la condition est fausse avant la boucle.

**répéter**

<instruction1>

....

<instructionN>

**jusqua** <condition de sortie>

*On teste ce que l'on vient de traiter dans le corps de boucle*

Java :

```
do
{
    ... suite d'actions .....
}
while ( condition_continuation );
```

**exemple** Dowhile1.java



Noter qu'en algorithmique, le **jusqua** est suivi de la condition de sortie de boucle. En Java, le **while** est suivi de la condition de continuation. Il est conseillé de toujours commencer par formuler la condition de sortie qui est plus facile à trouver, et de la mettre en commentaire de la boucle : si c'est une expression composée avec des **et/ou**, on appliquera les règles de De Morgan, pour trouver la condition contraire (=> condition de continuation)

Exemple de codage de boucle :

```
do
{
    i = fi(i) ;
    j = fj(j) ;
} while ((i <= 5) && (j > 3)) // arrêt quand i > 5 ou j <= 3
```

1) d'abord nous avons écrit le commentaire de condition d'arrêt

2) ensuite nous lui avons appliqué les règles de De Morgan  $\text{non} (a \text{ et } b) = \text{non } a \text{ ou non } b$   
 $\text{non} (a \text{ ou } b) = \text{non } a \text{ et non } b$

donc  $\text{non} (i > 5 \text{ ou } j <= 3) = \text{non} (i > 5) \text{ et non } (j <= 3)$   
 $= (i <= 5) \text{ et } (j > 3)$  // c'est la condition de continuation



**A lire en complément : « Faire des répétitions » : pp. 81-104 ou pp. 103-127\***

## Exemples

```
/******
Programme      : For1.java
Auteur        : Lécu Régis
Mise a jour   : février 2001, maj nov 2003 jcc
Fonction      : Illustre la boucle "for" en Java

Ce programme effectue la somme des N premiers nombres strictement
positifs, puis la somme des N plus grands nombres strictement négatifs
([-N,-1]), en affichant à l'écran la suite des nombres.
*****/

public class For1
{
    public static void main (String arg [])
    {
        int nombreEntiers;    // Nombre d'entiers que l'on doit sommer
        int somme;             // somme des entiers

        System.out.println ( "Somme des N Premiers entiers positifs" );
        System.out.print ( "Nombre d'entiers a sommer : " );
        nombreEntiers = Lire.i ();
        System.out.println ();

        // Etape 1 : affichage des N premiers entiers positifs
        // et calcul de leur somme

        somme = 0;
        for (int entier = 1; entier <= nombreEntiers; entier = entier + 1)
        {
            System.out.println ("J'ajoute " + entier);
            somme = somme + entier;
        }

        // Etape 2 : affichage de leur somme

        System.out.println ("Somme des " + nombreEntiers +
                             " premiers entiers positifs : " + somme);

        // Etape 3 : affichage des N plus grands entiers négatifs
        // et calcul de leur somme

        System.out.println ();
        System.out.println ( "Somme des N plus grands entiers negatifs");
        System.out.print ( "Nombre d'entiers a sommer: " );
        nombreEntiers = Lire.i ();
        somme = 0;
        System.out.println ();

        for (int entier = -1; entier >= -nombreEntiers; entier = entier - 1)
        {
            System.out.println ( "J'ajoute " + entier);
            somme = somme + entier;
        }

        // Etape 4 : affichage de leur somme
    }
}
```

```
System.out.println ();  
System.out.println ("Somme des " + nombreEntiers +  
                    " plus grands entiers negatifs : " + somme);  
  
System.out.println ();  
System.out.println ("*** Tapez Entree pour Terminer ***");  
Lire.c();  
}  
}
```

```

/*****
Nom du programme   : While1.java
Auteur            : Lécu Régis
Mise à jour       : février 2001, maj nov 2003 jcc
Fonction          : Illustre la structure algorithmique while (tant que)

    Tant que l'utilisateur répond 'O' à la question "Voulez vous continuer ",
    le programme boucle en affichant à l'écran le nombre de passages dans la
    boucle, et calcule à chaque passage une suite arithmétique de raison et
    de premier terme choisi par l'utilisateur.
    par exemple premier terme 3, raison 5, la suite est 3, 8, 13, 18, 23, ...
*****/

public class While1
{
    public static void main (String arg [])
    {

        char veutContinuer;        // egal à 'O' quand on veut continuer

        int nbrPassages;           // nombre de passages dans la boucle
        int suite;                 // suite arithmétique calculée dans la boucle
        int raison;                // raison de la suite arithmétique

        System.out.println ( "** While **" );
        System.out.println ();
        nbrPassages = 0;           // on n'a pas ENCORE exécuté la boucle !

        System.out.print ("Premier terme de la suite ? ");
        suite = Lire.i();          // U(0) premier terme de la suite

        System.out.print ("Raison de la suite ? ");
        raison = Lire.i();

        System.out.print ("Voulez-vous entrer dans la boucle (O pour oui)?");
        veutContinuer = Lire.c(); // si veutContinuer = 'O', on choisit
                                // d'exécuter la boucle au moins une fois!

        while (veutContinuer == 'O')
            // arrêt quand veutContinuer != 'O'
        {
            nbrPassages= nbrPassages + 1;
            // on compte le nouveau passage
            suite = suite + raison; // calcul du terme suivant de la suite

            System.out.print ( "Au " + nbrPassages + " passage, ");
            System.out.println ("la suite vaut: " + suite);

            System.out.print ("Tapez O pour continuer, une autre lettre "
                             + "pour arreter : ");
            veutContinuer = Lire.c();
        }

        System.out.println ();
        System.out.println ("*** Tapez Entree pour Terminer ***");
        Lire.c();
    }
}

```

/\*\*\*\*\*\*

Nom du programme : Dowhile1.java

Auteur : Lécu Régis

Mise a jour : février 2001, maj nov 2003 jcc

Fonction :

Ce programme lit un ensemble de caractères au clavier,  
(sur la même ligne ou non ), jusqu'au caractère "."

Parmi les caractères entrés, il compte :

- le nombre de lettres majuscules
- le nombre de lettres minuscules
- le nombre de caractères de ponctuation : ! ? , . ; :
- les caractères restant sont comptabilisés dans la catégorie "divers"

Dans cette version, on désire compter le point final comme un caractère de ponctuation.

\*\*\*\*\*/

```
public class Dowhile1
{
    public static void main (String arg [])
    {
        char carlu;
        // Déclarations et initialisations à 0 des différents compteurs
        // on n'a encore rien lu !
        int nbMajuscule = 0;
        int nbMinuscule = 0 ;
        int nbPonctuation = 0;
        int nbDivers = 0;

        // Dans cette version, on ne filtre pas les fins de ligne : tous
        // les caractères doivent être lus, y compris les caractères de contrôle

        Lire.Filtre (false);

        System.out.println ( "**** Boucle : do ... while (Version 1) ****" );
        System.out.println ();

        System.out.print ( "Texte termine par un point : ");

        do
        {
            carlu = Lire.c();

            if ( (carlu >= 'A') && (carlu <= 'Z') )
            {
                nbMajuscule = nbMajuscule + 1;
            }
            else if ((carlu >= 'a') && (carlu <= 'z') )
            {
                nbMinuscule = nbMinuscule + 1;
            }
        }
```

```

else if ((carlu == '.') || (carlu == ',') || (carlu == ';')
        || (carlu == '!') || (carlu == '?') || (carlu == ':'))
{
    nbPonctuation = nbPonctuation + 1;
}
else
{
    nbDivers = nbDivers + 1;
}
}
while (carlu != '.');

// affichage du nombre de caractères, dans chaque catégorie
System.out.println ( "Il y a ");

if (nbMajuscule != 0)
{
    System.out.println ( "\t" + nbMajuscule + " majuscules" );
}
if (nbMinuscule != 0)
{
    System.out.println ( "\t" + nbMinuscule + " minuscules" );
}
if (nbPonctuation != 0)
{
    System.out.println ( "\t" + nbPonctuation + " ponctuations" );
}
if (nbDivers != 0)
{
    System.out.println ( "\tet " + nbDivers + " caracteres d'un autre type" );
}

Lire.Purge();
Lire.Attente();

}
}

```



## Exercices

### **Exercice n° 1** : suite de la calculette

Faire la saisie de 2 nombres entiers, puis d'un opérateur + - \* ou / .

Tant que l'opérateur n'est pas correct, le programme recommencera sa saisie. Enfin le programme effectuera l'opération demandée (en prévoyant le cas d'erreur "division par 0"), puis affichera le résultat.

### **Exercice n° 2**

Ecrire un programme qui calcule les N premiers multiples d'un nombre entier X, N et X étant entrés au clavier.

Il est demandé de choisir la structure répétitive (for, while, do...while) la mieux appropriée au problème. On ne demande pas pour le moment de gérer les débordements (overflows) dus à des demandes de calcul dépassant la capacité de la machine.

### **Exercice n° 3** calculer le nombre de jeunes.

Il s'agit de dénombrer toutes les personnes d'âge inférieur strictement à vingt ans parmi un échantillon donné de vingt personnes. Les personnes saisissent leur âge sur le clavier.

Donnez le programme java correspondant.

### **Exercice n° 4** calculer le nombre de jeunes, de moyens et de vieux.

Il s'agit de dénombrer les personnes d'âge inférieur strictement à 20 ans, les personnes d'âge supérieur strictement à 40 ans et celles dont l'âge est compris entre 20 ans et 40 ans (20 ans et 40 ans y compris). Le comptage est arrêté dès la saisie d'un centenaire. Le centenaire est compté.

Donnez le programme java correspondant qui affiche les résultats.

# CHAPITRE 5

## FONCTIONS ET PARAMETRES

### Objectifs

- Définir et appeler des fonctions sans paramètre, avec des paramètres en entrée, et des valeurs de retour.
- Créer une bibliothèque de fonctions dans une classe indépendante.

### Points clé

#### Définition et utilisation d'une fonction

- Une fonction permet de regrouper un ensemble d'instructions qui réalisent une action cohérente.
- Les instructions (« corps » de la fonction) sont entourées par { et }
- La fonction a un nom qui sert à l'appeler une ou plusieurs fois, dans le programme.
- La fonction doit être définie, pour pouvoir être utilisée dans le programme principal (**main**)

```
public class Test
{
    //    définition de la fonction
    public static void afficheMessage ()
    {
        System.out.println ("Ceci est message");
    }

    //    programme principal
    public static void main(String arg [])
    {
        System.out.println ("Programme principal") ;
        .....
        afficheMessage ();
        .....
        afficheMessage ();
        .....
    }
}
```



- le terme **void** en début de définition
- les parenthèses () lors de la définition et lors de l'appel de la fonction
- le programme principal est lui même une fonction de nom réservé **main**

## Imbrication de fonctions

- Plusieurs fonctions peuvent être définies dans une classe Java.
- Elles peuvent être appelées par le programme principal ou par une autre fonction.
- Les appels peuvent être imbriqués sur plusieurs niveaux (une fonction en appelle une autre, qui en appelle une autre ....).

```
public class Test
{
    // Programme principal

    public static void main(String arg [])
    {
        .....
        fonctionDeux ();
        .....

        fonctionTrois ();

        .....
        fonctionDeux ();
        .....

        fonctionUn ();
        .....
    }

    // Définitions des fonctions

    public static void fonctionUn ()
    {
        .....
    }

    public static void fonctionDeux()
    {
        fonctionQuatre();
        ....
        fonctionTrois();
    }

    public static void fonctionTrois()
    {
        .....
        fonctionQuatre();
    }

    public static void fonctionQuatre()
    {
        fonctionUn();
        .....
    }
} // fin de la classe Test
```



« Fonctions, notions avancées » : pp. 129-138 ou pp. 161-172\*

## Paramétrage d'une fonction

- Telle quelle, une fonction se déroule toujours de façon identique.
- Il est intéressant d'adapter son comportement à chaque appel, et pour cela on va la paramétrer.
- Le rôle des paramètres est de modifier les valeurs de travail de la fonction (exemple d'une fonction de calcul, d'affichage...), voire de faire varier légèrement son déroulement (paramètre servant de condition de structure alternative ...).
- Il faut prévoir lors de la définition de la fonction les paramètres qui vont lui être fournis à chaque appel, en leur donnant un nom et un type.
- Le nom et le type des paramètres seront utilisés dans le corps de la fonction (ce sur quoi elle va travailler et ce qu'elle pourra en faire) : ces paramètres sont dits formels
- Lors de l'appel de la fonction dans le programme principal, il faudra donner des valeurs pour ces paramètres. Ces valeurs sont appelées paramètres effectifs : la fonction va effectivement travailler avec eux pour cet appel précis.

## Mode de passage des paramètres en Java

- Lorsqu'un paramètre est fourni à la fonction uniquement pour être "consulté" (sans être modifié), c'est un **paramètre en entrée**. L'appelant fournit alors une valeur à la fonction : passage de paramètres « **par valeur** ».
- Si un paramètre est fourni à la fonction pour être modifié, c'est un **paramètre de sortie**.
- Sur les types simples (char, int, float...), le langage Java ne fournit que la technique de passage de paramètres « par valeur ». Lorsque notre algorithme comportera un paramètre en sortie, nous le traduirons pour le moment par une **valeur de retour**. Nous examinerons plus tard, en fin de chapitre, les passages de paramètres en sortie, et en entrée/sortie.



« Fonctions, notions avancées » : pp. 138-140 ou pp. 172-176\*

Réfléchir sur l'exemple de la p. 139 pour éviter de tomber dans le piège du passage par valeur

| exemple Parametre.java

## Les fonctions qui retournent un résultat

- Une fonction peut renvoyer un résultat à l'appelant par l'instruction **return**.
- Le retour devra être typé dans l'interface de la fonction, au même titre que ses paramètres

- Les fonctions utilisées jusqu'à présent sont un cas particulier qui renvoie **void**, c'est-à-dire rien : dans le corps d'une fonction **void**, on ne doit pas renvoyer de valeur par **return**.

Exemple :

```
public class Test
{
    // le type int remplace void
    private static int addition (int op1, int op2)
    {
        int resultat ;

        resultat = op1 + op2;
        return resultat;
    }

    public static void main(String arg [])
    {
        int op1 ;
        int result ;

        op1 = 1515 ;

        result = addition (op1, 2001) ;    // paramètres effectifs : op1, constante 2001

        // op1 est inchangé ; result vaut 3516
    }
} // fin de la classe Test
```



A noter :

- Dans le corps de la fonction *addition*, la variable *resultat* est locale à la fonction, c'est à dire qu'elle n'a de sens que dans la fonction.
- Le résultat rendu par l'instruction **return** doit être récupéré dans une variable de l'appelant grâce à une affectation, ou testé, ou affiché directement comme le contenu d'une variable.
- L'instruction **return** a deux rôles :
  - elle arrête le déroulement de la fonction et renvoie à l'appelant
  - elle rend une valeur au programme principal
- Le compilateur Java accepte plusieurs **return** dans une fonction. Utilisé sans valeur de retour, **return** indique le point de sortie dans les fonctions de type **void**.
- Mais pour rester « structuré » et ne pas revenir à la « programmation Spaghetti », on se limitera toujours à un seul **return** juste avant **}**, pour renvoyer la valeur de retour dans les fonctions typées. **return** peut être évité dans les fonctions **void**

## Fonctions dans plusieurs classes

- Cette technique facilite le travail d'équipe : le programme principal se situe dans une classe, les fonctions utilisées dans une ou plusieurs autres classes.
- Les fonctions utilitaires doivent être regroupées par thème : fonctions de gestion d'écran, fonctions mathématiques couramment utilisées...
- Pour chaque groupe de fonctions, il faut faire une classe, contenue dans le fichier de même nom, avec l'extension .java



A titre d'exemple, une bibliothèque prédéfinie : « **La librairie Math** » : pp. 110-112 ou pp. 138-141\*.

- Rechercher des informations sur cette librairie dans l'aide en ligne de J++



Notre bibliothèque de lecture clavier **Lire.java** est fournie en annexe : jeter un coup d'œil à la structure de la classe, aux interfaces des fonctions, sans regarder leur contenu, pour le moment.

## Les paramètres en entrée sortie ou sortie

Un paramètre est en sortie si sa valeur est calculée par la fonction, et il est en entrée sortie, si sa valeur est modifiée par la fonction ( qui tient donc compte de la valeur initiale ).

Nous avons vu qu'il n'était pas possible de réaliser directement ces passages de paramètres. Pourtant il peut être nécessaire de le faire, si la fonction a plusieurs informations en sortie ( nous pouvons toujours privilégier un paramètre en sortie comme retour de la fonction, mais il faut traiter les autres ).

Pour cela nous avons défini des classes ( à considérer ici uniquement comme un artifice pour réaliser notre passage de paramètres ).

Pour chaque type simple une classe associée a été définie :

int	Entier
short	EntierCourt
long	EntierLong
byte	Octet
char	Caractere
double	Reel
float	ReelCourt
boolean	Booleen

Voici comment opérer pour réaliser un passage de paramètre en entrée sortie :

procédure convertir (entrée sortie reel valeur )  
Cette procédure convertit en euros des francs

Ici le type du paramètre est double. Donc la classe associée est Reel ( classe associée au type simple )

```
public void convertir ( Reel valeur )    // employer ici la classe correspondant au type désiré
{
    final double POIDS = 6.55957 ;
    double somme = valeur.getVal() ; // getVal permet de récupérer la valeur du type désiré
    somme = somme / POIDS;
    valeur.setVal(somme) ;           // setVal permet de modifier la valeur du paramètre
}
```

Comment un programme utilise cette procédure ?

```
Reel valeur = new Reel() ;           // définit une variable issue de la classe Reel
valeur.setVal(15000) ;               // initialise cette variable

convertir(valeur) ;
double paye = valeur.getVal() ;      // récupère la valeur contenue dans la variable
```

Pourquoi cela fonctionne-t'il ? Les variables définies à partir des classes sont toutes passées en entrée sortie en java. Il suffit donc de déguiser une variable de type simple en variable de classe pour que le passage se fasse en entrée sortie.

Toutes les classes associées aux types simples sont définies dans l'exemple de passage de paramètres, avec une fonction setVal pour modifier la valeur contenue dans la classe, et getVal pour récupérer cette valeur. La valeur est elle du type simple concerné.

exemple Main.java avec Calcul.java

## Exemples

```
/******  
Programme      : Parametre.java  
Auteur        : Lécu Régis  
Mise a jour   : février 2001, maj nov 2003 jcc  
Fonction      : idem fonction.java  
Utilisation   :  
    appels de fonctions avec passage de paramètres en entrée, et  
    valeurs de retour  
*****/  
  
public class Parametre  
{  
  
    public static void main(String arg [])  
    {  
        System.out.println ("*** Calcul de l'inverse d'un nombre (V2) *****");  
        System.out.println ();  
  
        char reponse;  
        String nom;  
  
        System.out.print ("Entrez votre nom :");  
        nom = Lire.S();  
  
        // passage de nom à la fonction afficheBonjour,  
        // en paramètre par valeur  
        afficheBonjour ( nom );  
  
        do  
        {  
            afficheInverse ();  
            System.out.print ("Tapez O si vous voulez continuer :");  
            reponse = Lire.c();  
            System.out.println ();  
        }  
        while (reponse == 'O');  
  
        afficheAuRevoir(nom);  
        Lire.Attente ();  
    }  
  
    public static void afficheBonjour (String unNom)  
    {  
        System.out.print ("Bonjour ");  
        afficheNom (unNom);  
        // transmet le paramètre reçu: le parametre formel de afficheBonjour  
        // devient paramètre effectif de afficheNom  
    }  
  
    private static void afficheAuRevoir(String unNom)  
    {  
        System.out.print ("Au revoir ");  
        afficheNom (unNom);  
    }  
  
    private static void afficheNom (String unNom)  
    {  
        // affiche le parametre formel  
        System.out.println ("\t Monsieur " + unNom);  
        System.out.println ();  
    }  
}
```



```

private static double saisieNombre ()
{
    // variable tampon recevant le reel qui est retourné à l'appelant
    double reel;

    System.out.print ("Entrez un nombre reel : " );
    // La fonction d() de la classe Lire fonctionne comme saisieNombre
    reel = Lire.d();

    return reel ;
}

private static void afficheInverse()
{
    // variables locales à la fonction AfficheInverse
    double inverse;
    double nombreLu;

    nombreLu = saisieNombre();
    if (nombreLu != 0)
    {
        inverse = 1 / nombreLu ;
        System.out.println("L'inverse de " + nombreLu + " est " + inverse);
    }
    else
    {
        System.out.println ("0 n'a pas d'inverse !");
    }

    System.out.println ();
}
}

```

```

/*****
Nom du programme   : Power.java
Auteur            : Lécu Régis
Mise a jour       : février 2001, maj nov 2003 jcc
Fonction          : Fonctions avec valeur de retour
*****/

public class Power
{
    public static void main (String arg [])
    {
        int  entier;
        int  exposant;
        int  resultat;

        System.out.println ("*** Elevation a la puissance ***");
        System.out.println ();

        System.out.print ("Nombre entier ? ");
        entier = Lire.i();

        System.out.print ("Exposant ? ");
        exposant = Lire.i();
        System.out.println ();

        System.out.print (entier + " puissance " + exposant);
        resultat = puissance (entier, exposant);

        if (resultat != 0)
        {
            System.out.println (" = " + resultat);
        }
        else
        {
            System.out.println (" : calcul impossible");
        }

        System.out.println ();
        Lire.Attente ();
    }

    // Fonction Puissance
    // Si le calcul est possible (exposant >= 0), cette fonction
    // retourne la puissance "exposant" de l'entier "operande", sinon
    // elle retourne 0

    public static int puissance (int operande, int exposant)
    {
        int result;

        if (exposant < 0)
        {
            result = 0;
        }
        else
        {
            result = 1;
            for (int nbfois = 1; nbfois <= exposant; nbfois ++ )
            {
                result = result * operande;
            }
        }
        return result;
    }
}

```

```

/*****
Nom du programme : Calculs.java
Auteur          : Lécu Regis
Mise a jour     : février 2001, maj nov 2003 jcc
Fonction        : bibliothèques de fonctions de calcul
*****/

```

```
public class Calculs
```

```
{
    // Fonction Puissance : Si le calcul est possible (exposant >= 0), cette fonction
    // retourne la puissance "exposant" de l'entier "operande", sinon elle retourne 0
    //
    // Entree :          operande : entier dont on veut la puissance
    //              exposant : exposant positif ou nul
    // Valeur de retour  operande à la puissance exposant

    public static int puissance (int operande, int exposant)
    {
        int result;

        if (exposant < 0)
        {
            result = 0;
        }
        else
        {
            result = 1;
            for (int nbfois = 1; nbfois <= exposant; nbfois ++ )
            {
                result = result * operande;
            }
        }
        return result;
    }
}
```

// Fonction Factorielle : Si le calcul est possible ( $n \geq 0$ ), cette fonction  
// retourne la factorielle de "n", sinon elle retourne 0

// Entree :                    n : entier dont on veut calculer la factorielle  
// Valeur de retour        n!

```
public static int factorielle (int n)
{
    int result;

    if (n < 0)
    {
        result = 0;
    }
    else
    {
        result = 1;
        for (int nbfois = 1; nbfois <= n ; nbfois ++ )
        {
            result = result * nbfois ;
        }
    }
    return result;
}
// fin classe Calculs
```

```

/*****
Programme   : TestCalculs.java
Auteur      : Lécu Regis
Mise a jour : février 2001, maj nov 2003 jcc
Fonction    : test de la bibliothèque de fonctions calculs
*****/

```

```

public class TestCalculs
{
    public static void main (String arg [])
    {
        int entier;
        int exposant;
        int resultat;

        System.out.println ("*** TestCalculs ***");
        System.out.println ();
        System.out.print ("Nombre entier ? ");
        entier = Lire.i();

        System.out.print ("Exposant ? ");
        exposant = Lire.i();
        System.out.println ();

        System.out.print (entier + " puissance " + exposant);
        resultat = Calculs.puissance (entier, exposant);
        if (resultat != 0)
        {
            System.out.println (" = " + resultat);
        }
        else
        {
            System.out.println (" : calcul impossible");
        }

        System.out.println ();
        System.out.print ("Factorielle " + entier);
        resultat = Calculs.factorielle (entier);
        if (resultat != 0)
        {
            System.out.println (" = " + resultat);
        }
        else
        {
            System.out.println (" : calcul impossible");
        }

        System.out.println ();
        Lire.Attente ();
    }
}

```

/\*\*\*\*\*\*

Programme : Main.java

Auteur : Corre Jean Christophe

Mise à jour : nov 2003 jcc

Fonction : Démo sur les passages de paramètres en entrée  
sortie et en sortie. Appel de la fonction calcul de la classe Calcul.

\*\*\*\*\*/

public class Main

{

public static void main (String[] args)

{

// test de la fonction calcul

**Entier valeur= new Entier();**

**EntierLong square = new EntierLong();**

**Reel root = new Reel();**

**valeur.setVal(45); // valeur en entrée du nombre**

System.out.println("valeur du nombre en entree " +  
**valeur.getVal());**

**boolean res = Calcul.calcul(valeur, square, root);**

System.out.println("valeur du nombre en sortie " +  
**valeur.getVal());**

System.out.println("valeur du carre " + **square.getVal());**

if ( res == true) // if ( res ) tout simplement

{

System.out.println("valeur de la racine carree "  
+ **root.getVal());**

}

else

{

System.out.println("pas de racine carree");

}

Lire.Attente();

}

}

/\*\*\*\*\*\*

Programme : Calcul.java

Auteur : Corre Jean Christophe

Mise à jour : nov 2003 jcc

Fonction : Démo sur les passages de paramètres en entrée sortie et en sortie.

Définition de la fonction calcul de la classe Calcul. Pour un entier, cette fonction calcule son carré, sa racine carrée, et nous dit si le nombre est positif pour que la racine carrée ait un sens. L'entier est incrémenté en sortie de fonction

\*\*\*\*\*/

```
public class Calcul
```

```
{
```

```
    // Fonction calcul: elle incrémente l'entier donné, après avoir
```

```
    // calculé son carré, sa racine, et vérifié qu'il est positif
```

```
    // Entrée Sortie: un entier en entrée un nombre ( int )
```

```
    //                               en sortie le nombre incrémenté
```

```
    // Sortie: le carré du nombre ( long )
```

```
    //          sa racine carré ( double )
```

```
    // En valeur de retour: un booléen vrai si le nombre est positif
```

```
    public static boolean calcul( Entier nombre, EntierLong carre, Reel racine)
```

```
    {
```

```
        int val = nombre.getVal ();                // valeur du nombre en entrée
```

```
        boolean positif;
```

```
        nombre.setVal (val+1);                    // incrémenter le nombre
```

```
        carre.setVal (val*val);                    // calcul du carré
```

```
        positif = (val>=0);
```

```
        if ( positif)
```

```
        {
```

```
            racine.setVal(Math.sqrt(val));        // calcul de la racine carrée
```

```
        }
```

```
        return positif;
```

```
    }
```

```
}
```

## Exercices

### Exercice n° 1

- Dans une nouvelle classe **Dialogue**, réaliser une fonction **veutContinuer** sans paramètre : pose la question "*Voulez-vous continuer ?*", filtre la réponse ((O ou o) pour oui, ou (N ou n) pour non) et retourne un booléen selon la réponse. Si la réponse n'est ni O, ni o, ni N, ni n, la question sera posée à nouveau, jusqu'à avoir un résultat satisfaisant.
- Grâce à cette nouvelle fonction, compléter le programme **Calcullette**, pour pouvoir effectuer plusieurs calculs à la suite.



## Petit projet de synthèse: CALCULETTE

### Objectifs

- Présentation sur un exemple des phases du développement logiciel
- Langage Java : synthèse des premiers chapitres.

### Points clés

#### Notions d'analyse fonctionnelle et d'analyse organique

##### 1° Analyse Fonctionnelle (ou Spécification)

- Elle exprime **ce que doit** faire le programme, tel qu'il est vu par le client
- Elle utilise du vocabulaire **non informatique**, pouvant être compris du client : texte libre en français, ou des représentations graphiques simples (cf. les « Use Case » dans la méthode UML)
- Elle aboutit à un découpage en **fonctionnalités** : principales actions réalisées par le programme, qui répondent à la demande exprimée par le cahier des charges
- C'est un **document contractuel** entre l'entreprise et le client

##### 2° Analyse Organique (ou Conception)

- Elle exprime **Comment** organiser, structurer le programme pour qu'il effectue les actions décrites
- Elle utilise du vocabulaire **informatique** : le dossier de conception est directement utilisable pour la programmation
- Elle aboutit à un découpage en « modules » : **classes** ou **fonctions**
- en programmation structurée, il existe deux points de vue sur un module :
  - \* pour son rédacteur, le module est lui-même un petit programme, qui réalise une suite d'actions et manipule des données, ou variables
  - \* pour le programme utilisateur, le module est une **boîte noire**, connue par sa **fonction** (définie en français), et son **interface** (ce qui entre et ce qui sort du module)

## *Avantages d'une décomposition modulaire*

---

- structuration de la pensée, clarté du programme
- localisation rapide des erreurs de programmation
- facilite les évolutions ultérieures du programme
- permet la réutilisation de certains modules ("briques" de base) dans d'autres projets
- facilite le découpage du travail d'équipe

## **Projet CALCULETTE**

## *Cahier des charges*

---

On veut réaliser une calculette élémentaire, qui effectue les 4 opérations de base  $+$   $-$   $*$   $/$  sur des nombres entiers compris entre -1000 et +1000.

Les opérandes et l'opérateur seront entrés comme suit :

- \* premier opérande <return>
- \* opérateur <return>
- \* deuxième opérande <return>

**Contrôles de cohérence** : le programme vérifie la validité de l'opérateur, affiche un message d'erreur et redemande l'opérateur si celui-ci est erroné. Il détecte les opérandes en dehors de la plage de calcul autorisée, affiche un message d'erreur et redemande l'opérande. Dans le cas de la division, le programme vérifie que le diviseur est non nul .

**Restriction d'utilisation** : dans cette version, on admettra exceptionnellement que certaines erreurs ne soient pas gérées : caractères non numériques à la place d'un nombre, overflow...

## *Analyse Fonctionnelle*

---

*Que doit faire le programme "calculette" ?*

- saisir un premier opérande valide
- saisir un opérateur valide
- saisir un deuxième opérande valide
- effectuer le calcul demandé, selon l'opérateur choisi
- afficher le résultat à l'écran

Il comporte 4 fonctionnalités principales :

### 1) Saisir des opérandes valides

Cette fonctionnalité demande les opérandes à l'utilisateur par les messages **Premier Opérande (-1000...1000]** puis **Deuxième Opérande (-1000...1000)**.

Si l'opérateur entre un nombre en dehors de cette plage, elle l'avertit par le message : **Erreur : opérande hors limite !**, et ressaisit l'opérande.

Elle ne gère pas les autres types d'erreurs .

### 2) Saisir un opérateur valide

Cette fonctionnalité demande l'opérateur à l'utilisateur par le message : **Opérateur (+ - \* / )** .

Si l'utilisateur entre un opérateur non valide, elle l'avertit par le message : **Ceci n'est pas un opérateur valide !** et ressaisit l'opérateur.

### 3) Effectuer le calcul demandé par l'utilisateur

Dans le cas d'une division, cette fonctionnalité vérifie que le diviseur n'est pas nul. Si l'opération demandée par l'opérateur est possible, elle effectue l'opération entre les deux opérandes.

### 4) Affichage du résultat du calcul

Affiche le résultat du calcul, si celui-ci est possible, par le message **Le résultat est : nnnn** , ou le message d'erreur **Calcul impossible :division par zéro**

---

## Conception

---

### Etapas communes (conception générale)

- 1) *écrire l'algorithme de principe, en langage algorithmique proche du français*
- 2) *définir les types des informations manipulées et spécifier les interfaces des fonctions de premier niveau*
- 3) *réécrire l'algorithme général et s'assurer que toutes les fonctions nécessaires ont été prévues*
- 4) *réaliser le squelette en Java : écrire la ou les classes, et leurs fonctions **public** (le corps des fonctions sera constitué d'une simple trace, c'est à dire `System.out.println(" nomfonction " ) ;`)*
- 5) *Compiler une première fois, vous avez mis en place le squelette de l'application. Cette application fonctionne, mais ne fait pas grand chose. Les fonctions seront créées, puis testées, une à une sur ce squelette ( nous allons habiller le squelette ).*

## Etapes pour chaque module (conception détaillée)

Chaque participant doit maintenant analyser ses classes et fonctions, et les réaliser. Il faut repartir des interfaces : à ce niveau on ne décide plus de la boîte, mais de son contenu.

- 1) écrire l'algorithme du module
- 2) définir les variables locales au module

### Algorithme de principe de Calculette

saisir un premier opérande valide

saisir un opérateur valide

saisir un deuxième opérande valide

// ici nous sommes sûr que l'opérateur et les opérandes sont corrects

effectuer le calcul

afficher résultat // affiche un message d'erreur si calcul impossible

### Variables

Signification	Nom variable	Type
premier opérande	! op1	! entier entre -1000 et 1000
deuxième opérande	! op2	! entier entre -1000 et 1000
opérateur	! opérateur	! caractère dans ( +,-,*,/)
résultat	! résultat	! entier
faisabilité du calcul		! réussite ! logique : vrai si calcul correct

### Spécifications des fonctions

Reprenons les actions de l'algorithme une par une :

1) "saisir un premier opérande valide" et "saisir un deuxième opérande valide" sont deux appels de la même fonction, pour saisir les variables op1 et op2

**fonction** saisirOperande (texte : chaîne de caractères) : **entier**

texte est une information pour constituer l'invite de saisie ( ici premier ou deuxième )

retourne un entier lu au clavier, compris entre -1000 et 1000

2) "saisir un opérateur valide"

**fonction** saisirOperateur () : **caractère**

retourne un caractère lu au clavier, appartenant à + - \* /

3) "effectuer le calcul "

*Sur quoi porte l'action ?*

- elle reçoit deux opérandes et un opérateur
- restitue un résultat et un compte\_rendu ( calcul possible ou non )

**fonction** effectuerCalcul ( **entrée** operande1 : **entier** ,  
                                  **entrée** operande2 : **entier** ,  
                                  **entrée** operateur : **caractère** ,  
                                  **sortie** résultat : entier) : **booléen**

Si l'opération représentée par « operateur » est faisable, effectue l'opération demandée sur « operande1 » et « operande2 », met la valeur calculée dans résultat, et retourne vrai.

Si l'opération n'est pas faisable (tentative de division par zéro), retourne faux.

4) " afficher les résultats ".

*Il faut savoir si l'action s'est bien exécutée ou non !*

**fonction afficher** ( **entrée** résultat : **entier** ;  
                          **entrée** réussite : **booléen** )

Si « réussite » est Vrai, affiche le résultat, sinon affiche un message d'erreur

## Algorithme définitif de Calculette

*Tous les appels de fonctions doivent figurer en clair, avec les noms des variables sur lesquelles elles portent.*

**Programme Calculette**

**Variables**

op1 : **entier**  
op2 : **entier**  
opérateur : **caractere**  
résultat : **entier**  
réussite : **booléen**

**debut**

op1 = **saisirOperande**("premier") // saisie premier opérande  
opérateur = **saisirOperateur**()  
op2 = **saisirOperande**("deuxième" ) // saisie deuxième opérande  
  
réussite = **effectuerCalcul** (op1, op2, opérateur, resultat)  
**afficher** ( résultat , réussite )

**Fin**

## Conception Détaillée de *effectuerCalcul*

**fonction** effectuerCalcul ( **entrée** operande1 : entier ;  
                                  **entrée** operande2 : entier ;  
                                  **entrée** operateur : caractère  
                                  **sortie** résultat : entier ) : booléen

**variables**

    succès : booléen

**début**

    succès := vrai

    résultat := 0

**Choix sur** operateur **faire**

        '+' :             résultat := operande1 + operande2

        '\*' :            résultat := operande1 \* operande2

        '-' :            résultat := operande1 - operande2

        '/' :            **Si** operande2 = 0 **alors**  
                          succes := faux

**Sinon**

                          resultat := operande1 / operande2

**Finsi**

**Autrecas** :   succes := faux // a priori impossible dans notre programme

**FinChoix**

**retourner** (succès)

**fin**

## TRAVAIL A EFFECTUER

- Finir la conception détaillée : fonctions *saisirOperande*, *saisirOperateur*, *afficher*
- Comme on ne dispose pas de paramètres en sortie sur les types simples en Java, on remplacera le type simple par une classe d’habillage ( Entier ), pour permettre aux fonctions *afficher* et *effectuerCalcul* de communiquer.
- Coder et mettre au point l’application.

# CHAPITRE 6

## Les tableaux

### Objectifs

- Organisation des données en tableaux.
- Utilisation en Java des notions essentielles d'algorithmique : parcours de tableaux, tris ...

### Points clés

#### Tableaux à une dimension

- Un tableau est une suite de variables de même type, indicée à partir de 0.
- Le type d'une case : n'importe quel type simple ou classe.
- Déclaration : **typeCase [] nomTableau ;** // nomTableau sera un accès à un tableau, ce que // nous appelons une référence à un tableau. Mais le // tableau lui n'a pas encore été créé.
- Exemples : **int [] entiers ; char [] texte ;**
- Avant d'être utilisé, le tableau doit être créé. La création alloue dynamiquement une zone mémoire correspondant au nombre de cases demandées. La commande **new** se charge de ce travail. New crée un emplacement mémoire chargé de contenir l'information que nous désirons y mettre.

```
texte = new char [100] ; // allocation de 100 caractères, numérotés de 0 à 99 (soit 200 octets)
                        // texte référence maintenant le tableau ( il permet d'y accéder )
entiers = new int [10] ; // allocation de 10 entiers (soit 40 octets)
                        // entiers référence le tableau d'entiers
```

- Accès à une case d'un tableau :

```
int x ;
entiers [i] = x;          // recopie le contenu de x dans la case numéro i du tableau entiers

x = entiers [7] ;         // recopie la case numéro 7 du tableau entiers dans la variable x

int [] tab ;
tab = new int [5] ;
tab[1] = entiers [6] ;    // recopie la case numéro 6 du tableau entiers dans la case numéro 1 de tab
```



Attention à ne pas "déborder" du tableau : erreur si indice  $< 0$  ou indice  $\geq N$

Dans l'exemple précédent (tableau de taille 10), l'instruction **entiers [10] = 3 ;** provoque l'exception Java : ***ArrayIndexOutOfBoundsException***.



Quand une exception ( une erreur ) se produit lors de l'exécution d'un programme l'intitulé de l'exception, ainsi que la ligne de code source où cette exception s'est produite sont affichés. Prenez la peine de lire ce code qui vous permettra de reconnaître le type d'erreur que vous avez commis, ainsi que l'endroit où cette erreur s'est produite. L'expérience montre que les développeurs croient savoir pourquoi « ça ne marche pas », mais qu'ils se trompent souvent car ils ne prennent pas le temps de lire les informations qui leurs sont données. L'expérience vous permettra de comprendre et d'enregistrer ces codes d'exception quelquefois abscons.



Le contenu d'un tableau ne peut être copié sur un autre par une affectation  
**texte1 = texte2 ;** // Compile mais ne veut pas dire ce que l'on croit en Java !

Lorsque l'on déclare un tableau, on ne réserve pas une zone mémoire, mais une **référence**, c'est à dire une case mémoire d'un type particulier, qui recevra l'adresse du tableau, à sa création.

Après l'affectation ci-dessus, **texte1** et **texte2** se réfèrent à la même zone mémoire.



Suivre le déroulement de ce programme sous debugger, pour comprendre le mécanisme de création dynamique de tableau

- Conclusion : il faut écrire des fonctions de lecture, d'affichage et de recopie, qui manipuleront une à une les cases de nos tableaux, en Java.

- En regardant cet exemple, on en déduit une conséquence pratique : les tableaux sont toujours manipulés par « référence » par le compilateur .
- Lorsque l'on passe un tableau en paramètre à une fonction, il est donc toujours passé en entrée-sortie. Vous l'aviez déjà compris, car la création d'un tableau ressemble à la création d'une variable de classe que nous utilisons pour permettre le passage des valeurs simples en entrée sortie ( utilisation de new ). Effectivement c'est le même mécanisme qui est utilisé, nous donnons en paramètre la référence sur l'information. La fonction a donc l'accès à l'information, et peut la modifier à son aise.



- Un tableau est donc toujours passé en entrée sortie. Que faire quand nous voulons qu'il ne soit qu'en entrée ( car l'algorithme modifie le tableau, et nous ne voulons pas qu'il soit modifié ) ? Il faut alors faire le travail que le compilateur n'a pas pu faire, c'est à dire recopier le tableau dans un autre, et faire le travail sur la copie ( nous simulons le passage de paramètre en entrée, avec copie de la valeur ). Attention toutefois de ne faire ce travail que si c'est nécessaire.



A lire : « **Collectionner un nombre fixe d'objets** » : pp. 203-208 ou pp. 245-251\*

## Exemples

```

/*****
Programme      :  Tableau.java
Auteur        :  Lécu Régis
Mise a jour   :  février 2001, maj nov 2003 jcc

Fonction      :  Utilisation d'un tableau à une dimension.

Le programme lit au clavier MAX entiers et les range dans un tableau.
Puis, il réaffiche les entiers dans l'ordre inverse de la saisie.
*****/

public class Tableau
{
    public static void main (String arg [] )
    {
        int [] tabEntier; // déclaration d'une référence sur un tableau d'entiers

        final int MAX = 5; // taille du tableau

        System.out.println ("*** Demo Tableaux ***");
        System.out.println ();

        // Etape 1 : allocation de MAX cases ( création du tableau )

        tabEntier = new int [MAX];

        // tabEntier référence le tableau nouvellement créé

        // Etape 2 : lecture des MAX entiers et rangement dans le tableau
        for (int i = 0 ; i < MAX ; i ++ )
        {
            System.out.println ("Entrez l'entier numero " + (i+1) + " : ");
            tabEntier[i] = Lire.i() ; // saisie de la case numéro i du tableau
        }

        // Etape 3 : réaffichage des MAX entiers dans l'ordre inverse
        System.out.println ("Vos entiers dans l'ordre inverse : ");

        for (int i= MAX - 1 ; i >= 0 ; i --)
        {
            System.out.println ("Case " + i + " : " + tabEntier[i] );
        }

        Lire.Attente ();
    }
}

```

```

/*****
Programme      : Erreurs.java
Auteur        : Lécu Régis
Mise a jour   : février 2001,maj nov 2003 jcc

Fonction       : Démonstration sur les erreurs les plus répandues sur
les tableaux : débordement, affectation d'un tableau à un autre...

Ce programme peut être suivi sous debugger, pour préciser les notions de
référence en Java
*****/
public class Erreurs
{
    public static void main (String arg [] )
    {
        System.out.println ("*** Erreurs frequentes sur les Tableaux ***");
        System.out.println ();

        // Etape 1 : création du tableau, avec un nombre de cases choisies par
        // l'utilisateur
        int [] entiers;          // déclaration du tableau d'entiers
        final int nbCases;      // taille du tableau (constant après sa lecture)

        System.out.print ("Taille de votre tableau ?");
        nbCases = Lire.i();
        entiers = new int [nbCases]; // attention si la taille est < 0

        // Etape 2 : rangement d'un entier dans le tableau, à un indice choisi
        // par l'utilisateur
        int iRang;
        do
        {
            System.out.print ("Indice de rangement ?");
            iRang = Lire.i(); // attention si l'indice est incorrect

            System.out.print ("Valeur ?");
            entiers [iRang] = Lire.i();
        }
        while (Lire.Question ("Une autre saisie"));

        // Remarquez l'utilisation directe de la case du tableau,
        // pour recevoir le retour de la fonction Lire

        // Etape 3 : réaffichage du tableau
        System.out.println ( "*** Contenu du tableau entiers ***");
        for (int i = 0 ; i < nbCases ; i ++)
        {
            System.out.println (i + ": " + entiers[i] );
        }

        // Etape 4 : un tableau "pointe", "se réfère" à un autre
        int clone [];
        // les deux tableaux désignent maintenant la même zone mémoire
        clone = entiers ;

        System.out.println ( "*** Contenu du tableau clone ***");
        for (int i = 0 ; i < nbCases ; i ++)
        {
            System.out.println (i + ": " + clone[i] );
        }
    }
}

```

```
// Etape 5 : si je modifie l'un, je modifie l'autre
clone[0] = 0; // je modifie la première case du tableau

System.out.println ( "** Contenu du tableau entiers **");
for (int i = 0 ; i < nbCases ; i ++)
{
    System.out.println (i + ": " + entiers[i] );
}

Lire.Attente ();
}
}
```

/\*\*\*\*\*\*

Programme : Tlettres.java  
Auteur : Lécu Régis  
Mise a jour : février 2001,maj nov 2003 jcc  
Fonction :

Utilisation d'un tableau de caractères en Java

Réalise un tri alphabétique sur un ensemble de maxLettres majuscules  
entrées au clavier, et réaffiche l'ensemble trié.

\*\*\*\*\*/

```
public class Tlettres
```

```
{  
    final static int MAXLETTRES = 10;  
  
    public static void main (String arg [])  
    {  
        char [] majuscules ;  
        int nbMaj ;  
        majuscules = new char [MAXLETTRES];  
  
        System.out.println ("Tri par ordre alphabetique d'un ensemble de lettres majuscules");  
        System.out.println ();  
  
        // étape 1 : lecture et stockage des lettres dans "majuscules"  
        nbMaj = lecture (majuscules);  
  
        // étape 2: les majuscules sont réaffichées dans l'ordre de la saisie  
        System.out.println ("Dans l'ordre de la saisie :");  
        affichage (majuscules, nbMaj);  
  
        // étape 3 : classement des lettres par ordre alphabétique  
        triAlphabetique (majuscules, nbMaj);  
  
        // étape 4 : réaffichage du tableau trié  
        System.out.println("Dans l'ordre alphabetique :" );  
        affichage (majuscules, nbMaj);  
  
        Lire.Attente ();  
    }  
}
```

```
// lecture : lit une suite de caractères terminée par un point, et
// stocke au plus maxLettres majuscules dans le tableau "maj".
// Renvoie le nombre de majuscules stockées
```

```
private static int lecture (char [] maj)
{
    int iRang = 0; // nombre de lettres majuscules déjà rangées, et indice de rangement de
                  // la prochaine lettre dans le tableau "maj"

    char carLu; // dernier caractère lu

    do
    {
        System.out.print ("Majuscule ou . pour arreter :");
        carLu = Lire.c();

        if ( (carLu >= 'A') && (carLu <= 'Z') )
        {
            // si le caractère est une lettre majuscule
            maj [iRang] = carLu; // il est rangé dans le tableau
            iRang ++;           // avance de l'indice de rangement
        }
        else if (carLu != '.')
        {
            System.out.println ("Ce n'est pas une majuscule ");
        }
    } while ( (iRang < maj.length) && (carLu != '.') );
    // arrêt quand iRang = taille du tableau ou quand un point est lu
    // maj.length est la taille du tableau

    System.out.println ();
    return iRang;
}
```

```
// affichage : affiche les "nbcar" caracteres du tableau "texte"
private static void affichage (char [] texte, int nbcar)
{
    for (int i = 0; i < nbcar ; i ++ )
    {
        System.out.print (texte [i] );
    }
    System.out.println ();
}
```

```

// permuter : inverse les caractères texte[iPermut] et texte[iPermut+1]
// utilisée pour le tri par remontée des bulles, dans TriAlphabetique
private static void permuter ( char [] texte, int iPermut)
{
    char sauvegarde = texte [iPermut];

    texte [iPermut] = texte [iPermut+1];
    texte [iPermut+1]= sauvegarde;
}

// triAlphabetique :
// met le tableau "texte" en ordre alphabétique, en utilisant le tri par "remontée des bulles"
private static void triAlphabetique (char [] texte, int nbText)
{
    boolean inversion; // vrai si on effectue au moins une inversion de deux
                       // éléments dans la petite boucle, c'est-à-dire si le tableau n'est
                       // pas encore complètement trié

    do
    {
        inversion = false;
        for (int i= 0; i < nbText - 1; i ++ ) // du premier à l'avant-dernier élément
        {
            if (texte [i] > texte [i+1] ) // si l'élément est plus grand que son successeur
            {
                inversion = true; // le tableau n'est pas encore ordonné
                permuter (texte, i);
            }
        }
    } while (inversion); // trier tant qu'il y a eu des inversions
}
}

```

## Exercices

### Exercice n° 1

Réaliser une procédure qui permet de remplir un tableau de taille fixe avec une chaîne de caractères donnée au clavier, et terminée par un point.

Cette procédure traitera avec soin les cas particuliers, et sera propre.

Si la chaîne saisie omet le point terminateur, vous le rajouterez.

Si la chaîne saisie est trop grande, vous la tronquerez, et rajouterez le caractère final.

Vous vérifierez qu'il ne reste pas de caractère dans le buffer de saisie, après l'appel à votre procédure, c'est à dire que l'appel à la méthode Lire.Attente() demande bien une saisie à l'utilisateur.

La saisie de la chaîne se fait bien en une seule saisie.

Rappel : la taille d'un tableau `tab` s'obtient en écrivant `tab.length`. Ceci vous donne le nombre maximum d'éléments que peut contenir le tableau.

Quand on frappe la touche entrée deux caractères sont générés : carriage return ( retour chariot ) '`\r`' et line feed ( ligne remplie ) '`\n`'. Cest deux caractères doivent être lus sinon ils restent dans le buffer de saisie, et seront consommés à la prochaine saisie ( par exemple par un Lire.Attente() qui n'a donc pas besoin d'une frappe au clavier pour s'exécuter car il y a un caractère disponible ).

Testez votre procédure avec soin

exemple de jeu d'essai : tableau de 6 éléments

saisie au clavier

contenu du tableau

<Entrée>	.
.<Entrée>	.
a.<Entrée>	a.
a.b<Entrée>	a.
abcde.<Entrée>	abcde.
abcdef.<Entrée>	abcde.
abcdefg<Entrée>	abcde.
abc<Entrée>	abc.

Votre procédure sera rangée dans une boîte à outils ( Cette procédure servira pour les prochaines procédures, pour lesquelles nous traiterons des tableaux de caractères ( et pas des String ) ).

### Exercice n° 2

Codez l'algorithme compter les occurrences d'une lettre dans une phrase terminée par un point.

### Exercice n° 3

Codez l'algorithme du calcul du nombre de lettre contenu dans une phrase.



#### **Exercice n° 4**

Codez l'algorithme de la recherche de 2 lettres successives dans une phrase.

#### **Exercice n° 5**

Codez l'algorithme du palindrome.

#### **Exercice n° 6**

Codez l'algorithme du tri par remontée des bulles sur un tableau d'entiers.

#### **Exercice n° 7**

Codez l'algorithme de la recherche par dichotomie d'un nombre strictement positif dans un tableau d'entiers triés. ( Nous traiterons les chaînes de caractères java plus tard ) Si le nombre est absent du tableau retournez 0.

#### **Exercice n° 8( optionnel)**

Codez l'algorithme de calcul de la valeur du  $x^{\text{ième}}$  bit d'un entier.

#### **Exercice n° 9 (optionnel)**

Codez l'algorithme du ET logique bit à bit entre deux entiers. Vous afficherez le résultat obtenu par l'opérateur & ( et logique ) sur ces deux nombres dans le programme de test pour vérifier votre algorithme.

## Exercices

Dans les exercices qui vont suivre nous allons également rester fidèles aux algorithmes qui ont été écrits. La méthode employée est la même que pour la calculatrice, mais appliqué à des tableaux de caractères.

### **Exercice n° 10**

Codez l'algorithme de recherche d'un mot dans une phrase.

### **Exercice n° 11**

Codez l'algorithme de recopie d'une phrase en inversant chaque mot.

### **Exercice n° 12**

Codez l'algorithme de justification d'une phrase sur 80 caractères.

### **Exercice n° 13**

Codez l'algorithme de calcul de la somme de nombres dans une base quelconque contenus dans une phrase.

# CHAPITRE 7

## LES TYPES STRUCTURES

### Objectif

- Tableaux à plusieurs dimensions
- Utilisation du mot clé **class** pour traduire la notion algorithmique d'enregistrement

### Points clés

#### Les chaînes de caractères java

En java nous ne sommes pas obligés de traiter les chaînes de caractères à l'aide de tableaux de caractères. Il existe une classe String qui nous permet de conserver des chaînes de caractères, et d'effectuer un certain nombre d'opérations dessus.

Une String est une chaîne de caractère constante. Si vous construisez une chaîne de caractères caractère à caractère, il est plus judicieux d'utiliser une StringBuffer ( chaîne de caractère pouvant évoluer ).

Exemple :

```
String nom = "dupond" ;
```

Si je change le dernier caractère pour un 't', je n'ai pas changé ma chaîne, j'ai construit une nouvelle chaîne de caractère. Ce travail est plus onéreux en ressources que de modifier un caractère.

Un exemple vous permettra de visualiser les différentes possibilités offertes par la classe String. Vous découvrirez à l'aide de la documentation les possibilités de la classe StringBuffer.

**exemple** : EssaiString.java

#### Tableaux à plusieurs dimensions

- Déclaration et création :

```
int [] [] matrice ;  
matrice = new int [5] [3] ;
```

*// déclare un tableau d'entiers à 2 dimensions  
// alloue 5 lignes, 3 colonnes*

- Accès à une case :

```
matrice [3] [2] = 1515;
```

**exemple** : Echec.java



A lire : « Collectionner un nombre fixe d'objets » : pp. 218-224 ou pp.263-270\*

## Les enregistrements en Java

<b>Tableau</b>	= stockage d'une suite de données de même type
<b>Enregistrement</b>	= stockage de données de types différents

- En Java, les enregistrements seront définis comme des **class**, avec tous leurs champs **public**, c'est-à-dire accessibles des autres classes.

```
class Couple
{
    public int a ;
    public char c;
}
```

- a et c sont les données membres de la classe Couple
- Ne pas faire précéder le mot clé **class** du mot clé **public**, qui désigne au compilateur la classe unique qui contient le **main**, si les deux classes sont dans le même fichier. Il ne peut y avoir qu'une classe publique dans un fichier, et la classe qui contient le main doit être publique.
- Si la structure est déclarée dans un fichier isolé, alors la classe peut être déclarée publique.

```
Couple ex1, ex2;           // déclare deux références ex1, ex2 à des objets de la classe Couple
```

```
ex1 = new Couple() ;       // crée l'objet et range sa référence (son adresse) dans ex1
```

```
Couple clone ;
clone = ex1;                // ex1 et clone se réfèrent au même objet en mémoire
```

```
ex2 = new Couple() ;       // ex2 se réfère à un nouvel objet
```

```
ex1.a = 1515 ;              // range 1515 dans le champ a de ex1
```

```
ex2.a = ex1.a;              // recopie le champ a de ex1 vers le champ a de ex2
```

**exemple** : Mesure.java



Lire : « **Les classes et les objets** » : pp. 160-165. ou pp. 194-201\*

- Se concentrer pour le moment sur la définition de l'objet, de ses données membres, sur la réservation mémoire par **new**, et la manipulation des champs par l'opérateur **.**
- L'utilisation des méthodes (ou fonctions membres) sera détaillée dans le cours objet.

## Récapitulatif sur l'organisation des données : analyse de données composées

- Partir du cahier des charges et décomposer les données, du général au particulier
- A chaque niveau de décomposition, se poser la question : de quoi X est-il composé ?
  - si X est une suite de x, le décrire par un tableau  
`x [] X ;`
  - si X se compose de x, y, z, le décrire par un enregistrement

```
class X  
{  
    public x cx;  
    public y cy;  
    public z cz;  
}
```

- Arrêter la décomposition quand on arrive à des types simples

exemple : TabMesures.java

## Exemples

```
/******
Programme      EssaiString.java
Auteur        jcc
Mise a jour    nov 2003
Fonction
Essai d'utilisation de différentes méthodes (fonctions) de la classe String
*****/

public class EssaiString
{
    public static void main(String [] arg)
    {
        String s;
        // lecture d'une chaîne au clavier
        System.out.print("frappez \"lapin\" :");
        s = Lire.S();

        //affichage de la String (entre des étoiles)
        System.out.println("votre chaine est :"+s+"");

        // testons les différentes comparaisons
        if (s == "lapin")
        // ici nous regardons si les références référencent bien les mêmes objets
        {
            System.out.println("votre chaine est reconnue");
        }
        else
        {
            // ce n'est pas la même chaîne, c'est une autre qui a même contenu
            System.out.println("votre chaine n'est pas reconnue");
        }

        if (s .equals( "lapin"))
        // ici nous regardons si les chaînes ont le même contenu
        // equalsIgnoreCase fait la même chose sans tenir compte de la casse
        {
            System.out.println("votre chaine est reconnue");
        }
        else
        {
            System.out.println("votre chaine n'est pas reconnue");
        }

        if (s.compareTo("lapin") == 0)
        // ici nous regardons l'ordre alphabétique des chaînes
        // 0 si même chaîne
        // >0 si s est plus grande que "lapin"
        // <0 si s est plus petite que "lapin"
        {
            System.out.println("votre chaine est reconnue");
        }
        else
        {
            System.out.println("votre chaine n'est pas reconnue");
        }

        // comment récupérer un caractère d'une chaîne:
        char c = s.charAt(0); // premier caractère de la chaîne si non vide
        System.out.println("le premier caractere de la chaine est :"+c);

        // calculer la longueur d'une chaîne
        System.out.println("la longueur de la chaine est :"+ s.length ());
    }
}
```

```
// récupérer la chaîne en majuscule
String smaj = s.toUpperCase();
System.out.println("la chaîne en majuscule est :"+ smaj);

// récupérer un morceau de la chaîne
// après le deuxième caractère jusqu'au cinquième compris
String sbout = s.substring (2,5);
System.out.println("la fin de la chaîne est :"+ sbout);

// vous constaterez que ces fonctions ne modifient pas la chaîne,
// mais construisent une autre chaîne à partir de s.
// en regardant l'aide vous trouverez d'autres fonctions encore.

Lire.Attente();
}
}
```

```

/*****
Nom du programme   : Echec.java
Auteur            : Lécu Régis
Mise à jour       : février 2001, maj nov 2003 jcc
Fonction          : utilisation des tableaux à deux dimensions, du type String
                   et de la surcharge de fonctions en Java

```

Ce programme affiche un échiquier, et bouge les pièces à la demande de l'utilisateur, sans aucun contrôle sur la cohérence des mouvements.

```

*****/

class Position      // décrit une position sur l'échiquier
{
    short ligne;     // numéro ligne sur l'échiquier
    short colonne;   // numéro colonne sur l'échiquier
}

public class Echec
{
    static final String TRAIT = "_|_|_|_|_|_|_|_|_|_|";

    public static void main (String arg [] )
    {
        // tableau 8 X 8 représentant l'échiquier
        String unJeu[][];
        unJeu = new String[8][8];

        // mémoriser les positions de départ et d'arrivée
        Position depart;
        depart = new Position();

        Position arrivee;
        arrivee = new Position();

        init (unJeu);           // met chaque pièce à sa place
        afficher (unJeu);       // réaffichage de l'échiquier
        do
        {
            get ("Position de depart  :", depart); // saisie position de départ
            get ("Position d'arrivee  :", arrivee); // saisie position d'arrivée

            bouger ( unJeu, depart, arrivee); // effectue le mouvement
            afficher (unJeu);                 // réaffichage de l'échiquier
        }
        while (Lire.Question ("Autre coup ") );
    }
}

```



```

// ..... Fonctions d'initialisation de l'échiquier.....

// Cette fonction recopie la string "piece" sur toutes les cases du
// tableau "ligne"
// utilisée pour l'initialisation des lignes vides, et des lignes de Pion

public static void init (String [] ligne, String piece)
{
    for (int iCol = 0; iCol <= 7; iCol++)
    {
        ligne [iCol] = piece;
    }
}

// Cette fonction initialise tout l'échiquier, en début de partie
// NOTER la surcharge avec la fonction précédente

public static void init (String [][] jeu)
{
    // Initialisation des lignes vides, au milieu de l'échiquier
    for (int iLig = 2; iLig <= 5 ; iLig ++)
    {
        init (jeu [iLig], " ");
    }

    // Initialisation des blancs : Tour, Cavalier, Fou...
    jeu [0][0] = "TO";
    jeu [0][1] = "CA";
    jeu [0][2] = "FO";
    jeu [0][3] = "RO";
    jeu [0][4] = "RE";

    // Recopie de l'autre moitié de la ligne, par symétrie
    for (int iCol = 0; iCol <= 2; iCol ++)
    {
        jeu [0] [7-iCol] = jeu [0] [iCol];
    }

    // La ligne de pions blancs
    init (jeu [1], "PI");

    // Idem avec les noirs
    jeu [7][0] = "to";
    jeu [7][1] = "ca";
    jeu [7][2] = "fo";
    jeu [7][3] = "ro";
    jeu [7][4] = "re";

    for (int iCol= 0; iCol <= 2; iCol++)
    {
        jeu [7] [7-iCol] = jeu [7] [iCol];
    }

    // La ligne de pions noirs
    init (jeu [6], "pi");
}

```

```

//..... Fonctions d'affichage de l'échiquier .....

// Affichage de la grille supérieure : A B C D ...

private static void afficherGrille ()
{
    System.out.println () ;
    for (char iCol = 'A'; iCol <= 'H' ; iCol ++)
    {
        System.out.print (" | " + iCol) ;
    }
    System.out.println (" |") ;
}

// Affichage complet d'une ligne de l'échiquier : cases et grille

private static void afficher (short iLig, String [] ligne)
{
    // Afficher le numéro de ligne et la grille
    System.out.println (TRAIT);
    System.out.print ((iLig + 1) + "| " );

    // Afficher chaque case de la ligne, avec sa pièce
    for (short iCol = 0; iCol <= 7; iCol++)
    {
        System.out.print ( " " + ligne [iCol] + " |");
    }

    System.out.println ();
}

// Affichage complet de l'échiquier

private static void afficher (String jeu [][] )
{
    afficherGrille ();

    for (short iLig = 0; iLig <= 7; iLig++)
    {
        afficher (iLig, jeu [iLig] );
    }

    System.out.println (TRAIT);
    System.out.println ();
}

```

```

//..... Saisie d'une position.....
// Entree : question
//          "Position de départ" ou "Position d'arrivée"
// Sortie : position de la pièce

private static void get ( String question, Position position)
{
    String pos ;
    char l='0', c='Z';

    do
    {
        System.out.print (question);
        pos = Lire.S();
        if (pos.length() == 2) // sinon on recommence
        {
            l = pos.charAt(0);
            c = pos.charAt(1);
        }
    }
    while ( (l < '1') || (l > '8') || (c < 'A') || (c > 'H'));
    // arrêt quand '8'>= l >= '1' et 'H' >= c >= 'A'

    position.ligne = (short) ( l - '1');
    position.colonne = (short) ( c - 'A');
}

//..... Effectuer le mouvement d'une piece dans l'échiquier ...
// Entree-Sortie : tableau jeu
//                  représente l'échiquier
// Entree          : depart
//                  ligne, colonne de depart
//                  : arrivee
//                  ligne, colonne d'arrivee

private static void bouger (String jeu [][], Position depart, Position arrivee)
{
    // recopie de la pièce sur la case d'arrivée
    jeu[arrivee.ligne][arrivee.colonne] = jeu [depart.ligne][depart.colonne];

    // la case de départ est maintenant vide (symbolisé par " " )
    jeu [depart.ligne] [depart.colonne] = " ";
}
}

```

```

/*****
Programme      Mesure.java
Auteur        Lécu Régis
Mise a jour    février 2001, maj nov 2003 jcc
Fonction

Utilisation des classes en Java comme "enregistrement", sans fonctions
membres.

On gère un ensemble de mesure. Une mesure a pour caractéristique :
1° un libellé
2° l'état du capteur : s'il est en panne, la mesure est sans signification
3° la valeur de la mesure.
*****/

class Capteur          // descripteur de mesure : class sans l'attribut public
{
    public String    lib;    // libellé de la mesure
    public boolean   etat;   // vrai si fonctionnement capteur OK
    public int       val;    // valeur mesurée
}

public class Mesure
{
    public static void main (String arg[])
    {
        System.out.println ("Demo sur les enregistrements," +
                             " suivre sur le listing");
        System.out.println ();

        // déclaration d'une variable qui référencera un "Descripteur"
        Capteur pression1 ;
        // création du Descripteur par new, référencé par pression1
        pression1 = new Capteur();
        // initialisation de ses champs (Notation avec le .)
        pression1.lib = "PRESSION";
        pression1.etat = true;
        pression1.val = 5;

        afficher ("Capteur Pression1", pression1);

        Capteur pression2 = pression1;
        afficher ("Capteur Pression2 (idem pression1)", pression2);
        // Les deux objets se réfèrent maintenant au même enregistrement
        // en mémoire

        // modification d'un champ
        pression2.etat = false;
        pression2.val = 1;
        pression2.lib = "Basse pression";
        afficher ("Capteur Pression1 (apres modif par pression2)", pression1);

        Lire.Attente ();

        // Création d'un nouvel enregistrement référencé par pression2
        pression2 = new Capteur();
        pression2.lib = "Haute Pression";
        pression2.val = 30;
        pression2.etat = true;

        afficher ("Capteur Pression1", pression1);
        afficher ("Capteur Pression2 (autre enregistrement)", pression2);
        Lire.Attente ();
    }
}

```

```

public static void afficher (String msg, Capteur des)
// ici msg et des sont en entrée sortie, mais
// msg est une chaîne constante, et ne peut pas être modifiée
// des n'est pas modifié par la fonction, donc est utilisée comme une entrée
{
    // Accès aux champs "public" d'un objet
    // par la notation :    nom_objet.nom_champ

    System.out.println (msg);
    System.out.println ();

    System.out.println (" Nom      : " + des.lib);
    System.out.print  (" Etat      : ");
    if (des.etat)
    {
        System.out.println ("** OK **"); // capteur en état de marche
    }
    else
    {
        System.out.println ("** HS **"); // capteur en panne
    }

    System.out.println (" Valeur : " + des.val);
    System.out.println ();
}
}

```

```

/*****
Programme          TabMesures.java
Auteur            Lécu Régis
Mise a jour       février 2001, maj nov 2003 jcc
Fonction
    Utilisation d'un tableau d'enregistrements en Java :
    Cet exemple reprend le type Descripteur appelé maintenant Capteur,
    présenté dans mesure.java et définit un tableau de ces structures.
*****/

class Capteur          // descripteur de mesure : class sans l'attribut public
{
    public String  lib;    // libellé de la mesure
    public boolean etat;   // vrai si fonctionnement capteur OK
    public int     val;    // valeur mesurée
}

public class TabMesures
{
    public static void main (String arg[])
    {
        Capteur [] tMesures ;    // déclaration du tableau de Capteurs
                                   // tMesures référencera un tableau de mesures
        System.out.println ("*** Demo sur les tableaux d'objets *** ");
        System.out.println ();

        int nbCapteurs;
        System.out.print ("Nombre de Capteurs ?");
        nbCapteurs = Lire.i();
        System.out.println ();

        //création du tableau de Capteur: attention, ne réserve que des
        //"références" il faut ensuite créer les Capteurs, case par case.
        tMesures = new Capteur [nbCapteurs];
        // tMesure référence maintenant un tableau
        // chaque élément du tableau référencera un Capteur

        // Etape 1 : saisie des mesures...
        int iVide = 0;           // indice de la première case vide dans tMesures
        do
        {
            tMesures [iVide] = new Capteur (); // création du Capteur
            // la case numéro iVide référence maintenant un Capteur

            lire (tMesures [iVide]); // initialisation du Capteur
            iVide ++;
        }
        while (iVide < nbCapteurs); // arrêt quand iVide == nbCapteurs

        // Etape 2 : réaffichage des mesures...
        System.out.println ("** Recapitulatif de vos mesures **" );

        for (int iDescr = 0 ; iDescr < nbCapteurs ; iDescr ++ )
        // iDescr est l'indice de la case à afficher
        {
            afficher (tMesures [iDescr]);
        }

        Lire.Attente();
    }
}

```

```

// Affichage complet d'un Capteur de mesure
public static void afficher (Capteur des)
// des est passé entrée sortie pour faire une entrée
{
    System.out.println ();
    System.out.println (" Nom : " + des.lib);
    System.out.print  (" Etat  : ");
    if (des.etat)
    {
        System.out.println  ("** OK **");
    }
    else
    {
        System.out.println  ("** HS **");
    }

    System.out.println (" Valeur : " + des.val);
    System.out.println ();
}

// Lecture clavier d'un Capteur de mesure
public static void lire (Capteur des)
// des est passé en entrée sortie pour faire une sortie
{
    System.out.println ("** Saisie d'un Capteur de mesure ** ") ;

    System.out.print (" Nom : ") ;
    des.lib = Lire.S();

    des.etat = Lire.Question (" Capteur en etat de marche");
    System.out.print (" Valeur : ");
    des.val = Lire.i();
}
}

```

## Exercices

### **Exercice n° 1 :** Manipulation des tableaux à plusieurs dimensions

Ce programme doit pouvoir transposer une matrice, représentée par un tableau à deux dimensions : la ligne I de la matrice transposée Mt est constituée de la colonne I de la matrice d'origine M.

Matrice M :

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

Matrice Mt :

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

### **Exercice n° 2**

Codez la structure de données permettant de conserver les informations sur les pièces fabriquées par une entreprise.

Testez cette structure de données en insérant quelques pièces et en les faisant afficher.

### **Exercice n° 3**

Codez la structure permettant de gérer une pile d'entiers, ainsi que les fonctions de manipulation associées.

### **Exercice n° 4( optionnel)**

Codez l'algorithme de manipulation d'une table de hashcode.

### **Exercice n° 5**

Codez la structure de données permettant de gérer une table de noms classés alphabétiquement ( codé grâce à un tableau ), ainsi que les fonctions d'initialisation, d'ajout d'un nom, et de retrait d'un nom de la liste.



## CHAPITRE 8

### COMPLEMENTS SUR LES FONCTIONS

#### Objectifs

- Règles de visibilité des variables dans une même classe et entre plusieurs classes d'une application Java
- Surcharge de fonctions
- Notions de récursivité

#### Points clés

##### Visibilité des variables en Java

- On appelle **bloc** une suite d'instructions entourées des { }
- Dans un bloc, on peut déclarer des variables qui ne seront visibles que jusqu'à la fin du bloc }.
- Les variables déclarées au premier niveau dans une fonction seront donc visibles jusqu'à la fin de la fonction }.
- Les variables déclarées dans la classe à l'extérieur des fonctions, sont visibles de toutes les fonctions de la classe.
- Une variable peut être « masquée » par une autre variable de même nom : variable de classe par une variable déclarée dans une fonction ou un bloc.

**exemple** : Visibilite.java

##### La surcharge des fonctions

- Plusieurs fonctions peuvent porter le même nom, (homonymie, surcharge ou "overloading") à condition de différer par leur nombre de paramètres, ou le type de leurs paramètres :

##### Déclarations :

```
int Add ( int i, int j);           // f1
int Add ( int i, int j, int k);    // f2
int Add ( int i, char car );       // f3
int Add ( char car , int i);       // f4
```

##### Appels :

```
x = Add ( 3, 8, 10 ); // appel de f2
```

```
x = Add ( 3, 'A' ); // appel de f3 ...etc...
```

Attention : pas d'overloading possible sur le type de la valeur de retour

```
int  ADD ();  
char ADD (); // provoque une erreur de compilation
```

## Réversivité simple

Une fonction qui s'appelle elle-même dans le déroulement de son code est dite réversive ; il faut bien sûr prévoir un moment où la fonction cesse de s'appeler pour se terminer.

Un grand nombre d'appels réversifs peu saturer la mémoire de l'ordinateur ( particulièrement la pile avec le message « stack overflow » ou débordement de pile). Une fonction sera réversive uniquement si la réversivité est la meilleure manière de traiter le problème. La fonction factorielle est à coder de manière itérative ( avec une boucle ) pas de manière réversive ( même si c'est l'exemple le plus répandu de programme réversif, car le plus simple ).

■ exemple : Recursive.java

## Réversivité croisée

- On a besoin de la réversivité pour traiter les problèmes d'analyse syntaxique : une expression contient des termes, un terme peut contenir une expression ...etc...
- En prenant la structure classique d'un programme d'analyse syntaxique (une fonction par élément de la grammaire), la fonction Expression doit donc appeler la fonction Terme et Terme peut appeler Expression...
- La fonction Expression s'appelle donc elle-même en passant par un appel à Terme. Terme s'appelle donc elle-même en passant par un appel à Expression. On est dans un cas d'appel "réversif croisé", à distinguer de la réversivité directe vue dans les exemples précédents.

■ exemple : Express1.java

## Pointeurs

La notion de pointeur n'existe pas directement en Java, mais les références sur les variables définies à partir des classes ( que nous appellerons bientôt objets ) nous rendent exactement le même service. La seule différence est qu'en java nous n'aurons pas de pointeur sur des types simples, seulement des références sur des enregistrements, et nous le verrons plus tard sur des objets.

Correspondance entre les pointeurs et les références :

créer                      new

détruire                  rien, car la destruction est automatique en java quand la structure n'est plus référencée. Les informations qui ne sont plus référencées seront collectées par le

ramasse miettes ( garbage collector ) qui est un outil du système, appelé automatiquement dans les moments de sous charge, ou en cas de manque de mémoire, ou encore appelé directement par l'appel **System.gc()** ;

null

null

Nous avons déjà utilisé des pointeurs en java. Chaque fois que nous définissons un tableau, nous manipulons une référence.

```
int [] tab ; // tab est une référence qui vaut null
```

```
tab = new int [ 6 ] ; // tab référence un tableau de 6 éléments
```

```
tab[3] = 67 ;
```

```
tab = new int [4] ; // tab référence maintenant un nouveau tableau de 4 éléments
```

```
// on note que le précédent tableau est perdu ( plus référencé par personne )
```

```
// le garbage collector s'occupera de son cas en temps et en heure
```

Notons une différence ( et simplification ) en java, l'élément pointé se confond avec la référence ( souvent dans le langage courant les programmeurs ne distinguent pas les deux ). Soit la définition suivante :

```
class Truc
```

```
{
```

```
    int i ;
```

```
    char c ;
```

```
    double d ;
```

```
}
```

```
Truc pt ; // pt ne référence rien
```

```
pt = new Truc() ; // pt maintenant référence un Truc
```

```
// on entend souvent dire pt est un Truc, mais c'est faux, dites bien pt référence un Truc
```

```
pt.i = 3 ; // ici pt. est l'élément référencé par pt ( l'équivalent p-> de l'algorithmie )
```

Nous utilisons donc les références en java sans nous rendre compte qu'il s'agit de pointeurs. L'exemple qui est donné vous montrera qu'il s'agit bien de pointeurs, dont la manipulation est simple si on veut bien s'en tenir au fait que l'on ne manipule pas les enregistrements, ( comme en algorithmie ), mais des références sur des enregistrements.

■ exemple : Pointeur.java

## Exemples

```

/*****
Programme          Visibilite.java
Auteur             Lécu Régis
Mise a jour        février 2001, maj nov 2003 jcc
Fonction
    visibilité des variables dans les classes Java. Données
    membres publiques et privées.
*****/

class Autre
{
    private static int x = 800; // x est privé, donc caché
    public static int y;        // y est public, donc visible de l'extérieur

    public static void Methode ()
    {
        System.out.println();
        System.out.println("\t\t** debut Autre.Methode **");
        System.out.println("\t\ttx (variable privée de la classe Autre)=" + x );
        System.out.println("\t\tty (variable publique de la classe Autre)=" + y );
        System.out.println("\t\t** fin Autre.Methode **");
        System.out.println();
    }
}

public class Visibilite
{
    static int x = 1959;    // x est visible dans toutes les méthodes de la classe

    public static void main (String arg [])
    {
        System.out.println("Visibilite des variables (suivre sur le listing)");
        System.out.println();

        System.out.println("x (variable de classe) = " + x );

        // x est visible de sa déclaration à la fin du main: il masque la
        // variable de classe de même nom ( évitez de faire cela !!! )

        int x = 1515;
        System.out.println("\tx (du main) = " + x );

        // appel d'une méthode de la classe "Visibilite"
        FoncLocale ();
        System.out.println("\tx (du main, bis) = " + x );

        Autre.y = 1600;
        Autre.Methode ();
        System.out.println("\tx (du main, ter) = " + x );

        Lire.Attente ();
    }
}

```

```

public static void FoncLocale ()
{
    System.out.println();
    System.out.println("\t\t** debut FoncLocale **");
    System.out.println("\t\ttx (variable de classe)  =" + x );

    // définition d'une variable locale à la fonction
    int x = 1492;
    System.out.println("\t\t\ttx (locale à la fonction)  =" + x );

    {
        // variable local au bloc ( délimité par { } )
        int y = 1789; // ici on ne peut pas redéfinir x
        System.out.println("\t\t\tty (dans le bloc1)  =" + y );
    }
    // ici y n'existe pas
    {
        // variable local au bloc ( délimité par { } )
        int y = 832; // ici on ne peut pas redéfinir x
        System.out.println("\t\t\tty (dans le bloc2)  =" + y );
    }
    // ici non plus y n'existe pas
    System.out.println("\t\ttx (locale à la fonction bis)  =" + x );
    System.out.println("\t\t** fin FoncLocale **");
    System.out.println();
}
}

```

```

/*****
Programme          Recursive.java
Auteur            Lécu Régis
Mise à jour       février 2001, maj nov 2003 jcc
Fonction

La fonction AppelRecuratif
- affiche le message "Vous entrez dans la fonction AppelRecuratif au niveau
d'appel I"
- demande à l'utilisateur s'il veut continuer
  et se rappelle elle-même si l'utilisateur répond oui , en passant comme
paramètre I + 1
- dans le cas contraire ,la procédure affiche le message
"Vous sortez de la procedure AppelRecuratif au niveau d'appel I " et se termine

Il est bien entendu que pour programmer ce problème, nous n'utiliserions pas
d'appels récuratifs, ( une boucle do...while suffit ), mais nous ne faisons que
visualiser le mécanisme de la récursivité.
*****/

public class Recursive
{
    public static void main (String arg[])
    {
        AppelRecuratif ( 1 ) ;
        Lire.Attente ();
    }

    public static void AppelRecuratif (int niveau)
    {
        System.out.println ("Vous entrez dans AppelRecuratif au niveau "
                             + niveau);

        if (Lire.Question ("Voulez-vous continuer"))
        {
            AppelRecuratif (niveau + 1); // avec le niveau d'appel + 1
        }

        System.out.println ("Vous sortez de AppelRecuratif au niveau " + niveau);
    }
}

```

/\*\*\*\*\*\*

Programme Express1.java  
Auteur Lécu Régis  
Mise à jour février 2001, maj nov 2003 jcc  
Fonction démo de récursivité croisée

Ce programme évalue des expressions parenthésées par une méthode récursive.

Exemple d'expressions :  $((100 + 900) * 5) + (((100 * 5) / 5) + 10)$   
10000 + 1

Exemples d'expressions interdites :  $(12 + A) * 4b$   
 $(100)$   
 $- 10000$   
10000  
 $(12 + 4)$

Signification des notations utilisées de la grammaire (BNF)

"x"	mot-clé : les mots-clés sont entourés de " "
x	une suite de caractères
x y	x et y se suivent et sont collés
x ::= y z	x se décompose en y et z
[x   y]	x ou y
{ x }	x est répété autant de fois que l'on veut (éventuellement 0)
N{ x }M	x est répété au minimum N et au maximum M fois
[x]	x est optionnel (peut être omis)

Définition de la grammaire des expressions parenthésées :

expression ::= terme opérateur terme  
(une expression se décompose en un premier terme , un opérateur  
et un deuxième terme : les opérateurs unaires + ou - sont interdits)

opérateur ::= "+" | "-" | "\*" | "/"  
(les opérateurs permis sont + - \* /)

terme ::= séparateur [ "(" expression ")" | terme\_simple ] séparateur  
(un terme est un terme simple (suite de chiffres) ou une expression parenthésée)

séparateur ::= { " " }  
séparateur est une suite d'un nombre quelconque d'espaces, voire 0

terme\_simple ::= 1{ chiffre }  
(un terme simple est une suite d'au moins un chiffre)

chiffre ::= "0" | "1" | ... etc ... | "9"

Le programme qui suit est construit en respectant la grammaire : une fonction par règle

On a besoin de la récursivité car une expression contient des termes ; un terme peut contenir une expression...

La fonction Expression doit donc appeler Terme, et Terme doit appeler Expression pour évaluer les formules parenthésées : la fonction Expression s'appelle donc elle-même en passant par un appel à Terme

\*/

```
public class Express1
{
    public static void main (String arg[])
    {
        // notez que l'indice est en entrée sortie des fonctions
        // l'erreur aussi, donc les classes d'habillage sont utilisées
        Booleen erreur = new Booleen();    // vrai si erreur de syntaxe détectée
        int  nbCar;                        // nombre de caractères de la formule
        char [] formule;                  // formule à calculer
        int valeur ;                      // valeur de l'expression parenthésée
        Entier iCour = new Entier();// indice du caractère à traiter

        erreur.setVal(false);            // pas d'erreur au départ
        iCour.setVal(0);                  // on part du début

        System.out.println("*** Calcul d'expression parenthesee ***" );
        System.out.println();

        // Saisie de la formule
        System.out.print("Formule :");
        String s = Lire.S();
        nbCar = s.length();
        formule = new char [nbCar];
        s.getChars (0, nbCar, formule, 0);

        // évaluation de la formule
        valeur = expression (iCour,formule,erreur);

        // réaffichage de la formule demandée
        System.out.print (" Votre formule " + formule);

        // Formule correcte : analysée jusqu'à la fin sans rencontrer d'erreur de syntaxe
        if ( iCour.getVal() == nbCar && !erreur.getVal())
        {
            System.out.println(" vaut " + valeur );
        }
        else
        {
            System.out.println(" est erronee ! ");
        }
        Lire.Attente ();
    }
}
```



// Separateur : suite d'un nombre quelconque d'espaces (y compris 0)

// saute les espaces à partir de la case iCour, jusqu'au premier

// caractère différent de espace, ou jusqu'à la fin de la formule

```
private static void separateur (Entier iCourant,char [] formu)
```

```
// iCourant est la valeur à partir de laquelle on saute les espaces
```

```
// en sortie le premier caractère différent d'un espace
```

```
// formu contient l'expression à analyser
```

```
{  
    int nbCar = formu.length ; // nombre de caracteres de la formule  
    int i = iCourant.getVal(); // indice courant de la formule  
    while ((i < nbCar) && (formu[i] == ' '))  
    { // arrêt quand plus d'espaces, ou fin de phrase  
        i++;  
    }  
    iCourant.setVal(i);  
}
```

//Operateur : vérifie que le caractère courant est + \* - ou /

```
private static char operateur (Entier iCourant,char [] formu,Booleen erreur)
```

```
// iCourant est la valeur à partir de laquelle on saute les espaces
```

```
// en sortie le premier caractère différent d'un espace
```

```
// formu contient l'expression à analyser
```

```
// erreur sera mis à vrai si il y a une erreur
```

```
{  
    char op = '?';  
    int i = iCourant.getVal();  
    if ( i == formu.length )  
    {  
        erreur.setVal(true); // fin d'expression anormale  
    }  
  
    // vérification de l'opérateur  
    else if ((formu[i] == '+' ) || (formu[i] == '-') ||  
            (formu[i] == '*') || (formu[i] == '/') )  
    {  
        op = formu [i];  
        iCourant.setVal(i+1);  
    }  
    else  
    {  
        erreur.setVal(true); // opérateur inconnu  
    }  
    return op;  
}
```

```

// Expression : une expression se décompose en un premier terme , un opérateur
//                  et un deuxième terme
//                  (les opérateurs unaires + ou - ne sont pas autorisés)
private static int expression (Entier iCourant,char [] formu,Booleen erreur)
// iCourant est la valeur à partir de laquelle on saute les espaces
//                  en sortie le premier caractère différent d'un espace
// formu contient l'expression à analyser
// erreur sera mis à vrai si il y a une erreur
{
    int valeur = 0;

    int t1 = terme (iCourant,formu,erreur);    // évaluation du premier terme
    if (! erreur.getVal())                    // test erreur de syntaxe dans le premier terme
    {
        char op = operateur (iCourant,formu,erreur);

        if (!erreur.getVal())
        {
            int t2 = terme (iCourant,formu,erreur);
            if (!erreur.getVal())// test erreur de syntaxe dans le 2ième terme
            {
                valeur = calculer (t1, op, t2,erreur);
                                // calcul de l'expression à partir des deux termes
            }
        }
    }
    return valeur;
}

```

```

// Terme : terme simple (suite de chiffres) ou expression parenthésée

// - saute le premier séparateur par Separateur
// - teste le mot-clé "(" pour distinguer si le terme est un terme SIMPLE, ou COMPOSÉ
// - Si le terme est un terme composé (commençant par "("),
//   - saute "("
//   - appelle Expression
//   - saute ")"
// - Sinon
//   - appelle TermeSimple
// - saute le dernier séparateur par Separateur
private static int terme (Entier iCourant,char [] formu,Booleen erreur)
// iCourant est la valeur à partir de laquelle on saute les espaces
//          en sortie le premier caractère différent d'un espace
// formu contient l'expression à analyser
// erreur sera mis à vrai si il y a une erreur
{
    int valeur = 0;
    if ( iCourant.getVal() == formu.length )
    {
        erreur.setVal(true); // fin rencontrée
    }
    else
    {
        separateur (iCourant,formu);
        if ( iCourant.getVal() == formu.length )
        {
            erreur.setVal(true); // fin rencontrée
        }
        else if (formu[iCourant.getVal()] == '(')
        {
            iCourant.setVal(iCourant.getVal()+1); // incrémente iCourant
            valeur = expression (iCourant,formu,erreur);
            if (formu[iCourant.getVal()] != ')')
            {
                erreur.setVal(true); // erreur syntaxe
            }
            else
            {
                iCourant.setVal(iCourant.getVal()+1); // incrémente iCourant
            }
        }
        else
        {
            valeur = termeSimple (iCourant,formu,erreur);
        }
        separateur (iCourant,formu);
    }
    return valeur;
}

```

```

// TermeSimple : suite de caractères numériques ( '0' à '9' )
// Un terme simple comporte AU MOINS un chiffre, donc la fonction détecte
// une erreur si le caractère courant n'est pas un chiffre
private static int termeSimple (Entier iCourant,char [] formu,Booleen erreur)
// iCourant est la valeur à partir de laquelle on saute les espaces
// en sortie le premier caractère différent d'un espace
// formu contient l'expression à analyser
// erreur sera mis à vrai si il y a une erreur
{
    String s = "";
    int valeur = 0;
    int iCour = iCourant.getVal();
    int nbCar = formu.length ;

    while ( (iCour < nbCar) && Character.isDigit(formu[iCour]) )
    // arrêt quand la chaine est finie, ou quand on a plus de chiffre
    {
        s = s + formu[iCour]; // concaténation pour avoir le nombre
        iCour ++;
    }
    // Si aucun caractère numérique, alors erreur
    if (s.length() == 0)
    {
        erreur.setVal( true ); // erreur syntaxe
    }
    else
    {
        valeur = Integer.parseInt (s); // conversion chaine en entier
    }
    iCourant.setVal(iCour);
    return valeur;
}

```

```

// EffectuerCalcul :
// effectue l'opération représentée par operateur sur les opérandes
// op1 et op2 et retourne le résultat. La seule erreur détectée est la
// division par 0
private static int calculer(int op1,char operateur,int op2,Booleen erreur)
{
    int valeur = 0;

    switch (operateur)
    {
        case '+':
            valeur = op1 + op2;
            break;

        case '*':
            valeur = op1 * op2;
            break;

        case '-':
            valeur = op1 - op2;
            break;

        case '/':
            if (op2 == 0)                // test de la division par zéro
            {
                erreur.setVal(true);    // calcul impossible !!
            }
            else
            {
                valeur = op1 / op2;
            }
            break;
    }
    return valeur;
}
}

```

```

/*****
Programme   : Pointeur.java
Auteur      : Corre Jean Christophe
Mise à jour : nov 2003 jcc
Fonction    : Démo sur l'utilisation des pointeurs en java à l'aide
               des références sur les enregistrements
*****/

class Element
{
    int donnée;          // information portée par une liste
    Element suivant;    // élément suivant de la liste
}

public class Pointeur
{
    public static void main (String[] args)
    {
        Element tete= null; // la liste est vide

        // remplissage de la liste

        for ( int i = 1; i < 6; i++)
        {
            Element e = new Element(); // creation d'un élément de la liste
            e.donnée = i*i;              // mise à jour de la donnée
            e.suivant = tete;           // chaînage des elements (ajout en tête)
            tete = e;                  // mise à jour de la nouvelle tête
        }

        // parcours et affichage de la liste

        Element courant= tete; // courant va parcourir la liste

        while (courant != null) // arrêt en fin de liste
        {
            System.out.println(courant.donnée);
            courant = courant.suivant; // courant référence maintenant l'élément
                                         // suivant dans la liste
        }
        Lire.Attente ();
    }
}

```

## Exercices

### Exercice n° 1

Faire l'algorithme des noms classés alphabétiquement à l'aide d'une liste chaînée dynamique ( donc en supprimant les tableaux, et en utilisant une référence sur l 'élément suivant dans la liste )

Vous aurez dans le main un petit menu qui propose d'ajouter, ou de retirer un nom de la liste, d'afficher le contenu de la liste, enfin de terminer le programme.

Vous appellerez alors les trois fonctions ajouter, retirer, et afficher.

## CHAPITRE 9

### LES FICHIERS

#### Objectifs

- Manipulation générale des fichiers et répertoires
- Utilisation des fichiers texte en Java, pour illustrer les traitements de fichier classiques (ruptures...)  
(La notion de « serialization » sera étudiée dans le cours spécialisé Java)

#### Points clés

##### *Manipulation des fichiers en Java par la classe **File***

- Cette classe a pour but d'homogénéiser les traitements de fichier, sur les différents environnements (Unix, Windows NT...).
- Elle fournit des méthodes standard de création de répertoire (**mkdir**), de parcours d'un répertoire (**list**), de renommage de fichiers (**renameTo**), de test du type d'un fichier (**isFile** et **isDirectory**)

exemple : ManipFic.java

##### *Accès en lecture-écriture sur des fichiers quelconques : classes **FileWriter** et **FileReader***

- Les fichiers sont considérés comme de simples suites de caractère.
- La classe **FileReader** permet de lire un fichier existant :
  - soit caractère par caractère : **int read()** ;  
(retourne le caractère lu ou -1, si la fin de fichier est atteinte)
  - soit par un tableau de caractères d'une taille donnée :  
**int read( char cbuf[], int off, int len ) ;**



- La classe **FileWriter** permet d'écrire dans un fichier :

<b><i>write(int)</i></b>	Ecrit un seul caractère
<b><i>write(String, int, int)</i></b>	Ecrit une partie d'une String
<b><i>write(char[], int, int)</i></b>	Ecrit une partie d'un tableau de caractères
<b><i>flush()</i></b>	Vide le buffer dans le fichier sur disque

exemple : Copie.java

---

## *Les fichiers texte*

---

- Les fichiers textes sont des fichiers directement visualisables ou imprimables, qui contiennent des représentations ASCII d'entiers, de réels...
- Ils se composent d'une suite de lignes.
- Avec l'utilisation actuelle des bases de données, leur utilité se limite à l'édition d'états imprimables, et à la récupération de données qui vont ensuite être réinjectées dans une autre base de données...
- En Java, les classes **BufferedWriter** et **BufferedReader** permettent de gérer les fichiers texte, ligne par ligne :

***void newLine()*** ;     ajoute une fin de ligne (méthode de **BufferedWriter**)

***String readLine()*** ;     lit une ligne de caractères (méthode de **BufferedReader**)

Ces classes permettent de faire des entrées sorties bufferisées, c'est à dire que l'on ne va pas gérer les entrées sorties caractère à caractère, mais ligne à ligne. Elles permettent donc d'avoir des outils plus évolués. Ces classes sont construites sur des objets permettant de gérer les entrées sorties en mode caractère.

- Les objets de ces deux classes sont construits à partir d'objets des classes **FileWriter** et **FileReader**

***BufferedWriter fw = new BufferedWriter (new FileWriter ( nom\_fichier ) ) ;***

exemple : Texte.java

## Exemples

```

/*****
Programme      ManipFic.java
Auteur        Lécu Régis
Mise a jour    avril 2001, maj nov 2003 JCC
Fonction       manipulation de fichiers et répertoires par la classe File

- Crée un répertoire, récupère son chemin absolu, et le supprime
- Ouvre un fichier ou un répertoire de nom choisi par l'utilisateur :
- S'il s'agit d'un répertoire liste son contenu.
- Termine en renommant le fichier ou répertoire sélectionné
*****/

import java.io.* ;
public class ManipFic
{
    public static void main (String[] args) throws IOException
    {
        System.out.println ("*** Manipulation des fichiers et repertoires ***");
        System.out.println ();

        System.out.print ("Nom du sous-repertoire a creer :");
        String rep = Lire.Chaine ();

        File fic = new File (rep);
        //création d'un répertoire
        fic.mkdir ();

        // affichage du chemin complet
        System.out.println ("Chemin complet " + fic.getAbsolutePath ());
        Lire.Suite ();

        // Suppression du répertoire
        fic.delete ();

        // Ouverture d'un fichier ou répertoire quelconque
        System.out.print ("Nom de fichier ou de repertoire a ouvrir :");
        rep = Lire.Chaine ();
        fic = new File (rep);

        // Si c'est un répertoire, on affiche son contenu
        if (fic.isDirectory())
        {
            System.out.println ("C'est un repertoire, de contenu : ");
            String tab [] = fic.list (); // liste des fichiers du répertoire
            for (int i = 0 ; i < tab.length ; i++)
            {
                System.out.println ( tab[i] );
            }
        }
        else if (fic.isFile())
        {
            System.out.println ("C'est un fichier de donnee");
        }
        else
        {
            System.out.println ("Fichier inexistant !");
            System.exit (1); // sortie sauvage du programme avec retour 1
        }
    }
}
```

```
// Renommage d'un fichier ou répertoire
System.out.print ("Nouveau nom du fichier :");
rep = Lire.Chaine ();
fic.renameTo ( new File (rep) );

Lire.Attente ();
}
}
```

```

/*****
Programme          Copie.java
Auteur            Lécu Régis
Mise a jour       avril 2001, maj nov 2003 JCC
Fonction
- Copie d'un fichier source vers un fichier cible
- Le fichier source et le fichier cible sont passés en paramètres
  du programme (args[0] et args[1])
- Ne fonctionne que pour les fichiers Texte (du fait de l'encodage
  effectué par les classes FileReader et FileWriter)
*****/
import java.io.* ;

public class Copie
{
    public static void main (String[] args) throws IOException
    {
        // Vérification du nombre de paramètres
        if (args.length != 2)
        {
            System.out.println ("** Syntaxe : Copie source cible" );
            System.exit (1); //arrêt programme avec code erreur
        }

        // test d'existence du fichier source
        File source = new File (args[0] );
        if (! source.exists())
        {
            System.out.println ("Fichier source inexistant ! ");
            System.exit (2); //arrêt programme avec code erreur
        }

        // ouverture en lecture du fichier source, en écriture du
        // fichier cible
        FileReader ficSource = new FileReader (source);
        FileWriter ficCible = new FileWriter (new File (args[1]));

        // relecture du fichier source octet par octet et recopie
        // dans le fichier cible
        // (à la fin du fichier, la méthode read renvoie -1)
        int octet = ficSource.read();
        while (octet != -1)
        {
            ficCible.write (octet);
            octet = ficSource.read ();
        }

        // fermeture des deux fichiers
        ficSource.close ();
        ficCible.close ();
    }
}

```

```

// réouverture des deux fichiers en lecture, et
// comparaison octet par octet pour vérification
ficSource = new FileReader (source);
FileReader ficCopie = new FileReader (args[1]);

int carSource;
int carCopie;
int nbcar = 0;
do
{
    carSource = ficSource.read ();
    carCopie = ficCopie.read ();
    nbcar = nbcar + 1;
}
while ( (carSource != -1) && (carCopie == carSource) );
// arrêt quand fin de lecture ou caractères différents

if (carSource != carCopie)
{
    System.out.println ("** Copie incorrecte ! **");
    System.out.println ("Octet numero " + nbcar + ": source= "
        + carSource + ", copie = " + carCopie);
}
else // carSource et carCopie égaux à -1
{
    System.out.println ("Copie achevee avec succes !");
}

ficSource.close ();
ficCopie.close ();
}
}

```

```

/*****
Programme      Texte.java
Auteur         Lécu Régis
Mise a jour    avril 2001, maj nov 2003 JCC
Fonction
- Saisie et réaffichage d'un fichier texte
*****/
import java.io.* ;
public class Texte
{

    public static void main (String[] args) throws IOException
    {
        System.out.println ("** Creation et reaffichage d'un fichier texte **");
        System.out.println ();

        // saisie du nom de fichier
        System.out.print ("Nom du fichier (sans extension) : ");
        String nom = Lire.Chaine ()+".txt";
        // Les entrées sorties bufferisées sont plus performantes que les
        // entrées sorties standards ( moins d'accès disque )
        // et possède en plus la gestion des lignes.
        BufferedWriter fw = new BufferedWriter (new FileWriter (nom)) ;

        // saisie du contenu du fichier
        System.out.println ("Entrez un texte termine par une ligne vide :)");
        String ligne = Lire.Chaine ();
        while (! ligne.equals ("") )
        {
            fw.write (ligne);
            fw.newLine ();
            ligne = Lire.Chaine ();
        }
        // fermeture du fichier
        fw.close ();

        // réaffichage du fichier
        System.out.println ();
        System.out.println ("Contenu du fichier " + nom);
        System.out.println (".....");
        BufferedReader fr = new BufferedReader (new FileReader (nom) );

        // la fonction readLine renvoie null en fin de fichier
        ligne = fr.readLine ();
        while ( ligne != null)
        {
            System.out.println (ligne);
            ligne = fr.readLine ();
        }

        Lire.Attente ();
    }
}

```

## Exercices

### Exercice n° 1( optionnel)

Ecrire un programme de « dump » qui affiche en hexadécimal et en ascii, le contenu d'un fichier choisi par l'utilisateur.

### Exercice n° 2

Client	N°Com	Code Produit	Quantité	Prix Total
Durant	98001	123	20	1002.50
Durant	98001	988	1	544.00
Durant	98001	344	20	22176.20
Durant	98002	872	5	552.00
Durant	98002	123	12	623.30
Durant	98002	222	20	66762.00
Durant	98003	100	1	727.50
Durant	98003	121	10	7798.00
Dupont	98005	665	13	4267.50
Dupont	98005	333	1	334.00
Dupont	98005	344	20	22176.20
Dupont	98007	223	33	21534.00
Dupont	98007	123	12	623.30
Dupont	98008	323	10	1862.00
Tutu	98009	345	3	22176.20
Tutu	98009	223	32	21534.00
Tutu	98010	920	11	623.30

Sur le fichier séquentiel ci-dessus (*Commandes.txt*), calculer les sous-totaux (prix) par numéro de commande, et les totaux par client, et afficher ces résultats sous la forme :

Client1	n° Commande 1	Sous Total
Client1	n° Commande 2	Sous Total
Client1		Total
Client2	n° Commande 3	Sous Total
Client2	n° Commande 4	Sous Total
Client2		Total
...etc ...		

### Exercice n° 3( optionnel)

Ecrire un programme permettant d'insérer un nombre dans un fichier texte, contenant des nombres par ordre croissant. Le fichier pourra être vide initialement. Bien sûr ce fichier reste classé par ordre croissant des nombres

#### **Exercice n° 4**

Gérer un fichier de salariés, où chaque fiche se compose d'un matricule (unique pour chaque salarié), un nom et un salaire.

L'application devra permettre de créer une nouvelle fiche, de détruire une fiche, de visualiser une fiche, et de lister l'ensemble des fiches, par numéro de matricule croissant.

Les matricules seront attribués automatiquement par l'application avec le format : MAT + numéro d'ordre (numéro d'ordre à partir de 1, croissant). Les numéros d'ordre des fiches supprimées ne seront pas récupérés.

#### *Choix techniques conseillés*

- on manipulera les fiches en mémoire dans un tableau.
- à la sortie de l'application, ce tableau sera sauvegardé dans un fichier texte.
- en début d'application, le tableau sera rechargé automatiquement à partir de ce fichier.

Pour clarifier la programmation, on réalisera une classe indépendante « Dialogue » qui regroupera les fonctions de dialogue opérateur : questions, affichage du menu et lecture du choix...



# ANNEXES

```

/*****
Nom du programme   : TestLire.java
Auteur            : Lécu Régis
Mise à jour       : février 2001, maj nov 2003 jcc
Fonction          : programme de test
                   de la bibliothèque de fonctions de lecture Lire.java
*****/
public class TestLire
{
    private static void ecrire (char car)
    {
        // Si le caractere est imprimable, on l'affiche
        if (car >= ' ')
        {
            System.out.println ("Caractere : " + car );
        }
        // Sinon si c'est un caractere de controle, on affiche son code ascci
        else
        {
            System.out.println ("Controle : code " + (int) car);
        }
    }

    public static void main (String[] args)
    {
        System.out.println ("**** Programme de test de la classe Lire ****");
        System.out.println ();

        String nom ;

        System.out.print ("Votre nom ? ");
        nom = Lire.S ();          // lecture d'une String

        int age ;
        System.out.print ("Votre age ? ");
        age = Lire.i ();          // lecture d'un entier

        float taille ;
        System.out.print ("Votre taille (en m) ? ");
        taille = Lire.f ();      // lecture d'un réel

        System.out.println ();
        System.out.println ("** Recapitulatif de votre fiche personnelle **");
        System.out.print ("Nom : ");
        System.out.println (nom);
        System.out.print ("Age : ");
        System.out.println (age+ " ans");
        System.out.print ("Taille : ");
        System.out.println (taille + " m");

        // Utilisation de la fonction booléenne Question
        System.out.println ();
        System.out.println ("***** Utilisation de la fonction Question *****");
        String trait = "";
        do
        {
            trait = trait + ".";
            System.out.println (trait);
        }
        while (Lire.Question ("Voulez-vous reboucler") );
        // arrêt quand on ne veut plus reboucler
    }
}

```

```

// Lecture d'un texte à la volée, sans aller à la ligne après chaque
// caractère frappé au clavier
System.out.println ();
System.out.println ("**** Saisie d'un texte sans filtrage ****");

Lire.Filtre (false);
System.out.println ("Entrez une suite de caracteres terminee par un .");
System.out.println ("Sur une ou plusieurs lignes. "
    + " Tous les caracteres sont lus");
char car = Lire.c();
while (car != '.')
{
    ecrire (car);
    car = Lire.c();
}

// remettons le filtre comme nous l'avons trouvé en arrivant
Lire.Filtre (true);

// Lecture d'un texte avec filtrage : seul le premier caractère
// de chaque ligne sera lu
System.out.println ();
System.out.println ("**** Saisie d'un caractere par ligne ****");

System.out.println ("Entrez une suite de caracteres terminee par un .");
System.out.println ("Seul le premier caractere de chaque ligne est lu");
car = Lire.c();
while (car != '.')
{
    ecrire (car);
    car = Lire.c();
}

// Lecture d'un texte sans filtrage : on élimine la fin de ligne
// à partir d'un caractère déterminé, par la fonction Purge

System.out.println ();
System.out.println ("**** Demo fonction Purge ****");
System.out.println ("Entrez une suite de caracteres terminee par un .");
System.out.println ("Sur chaque ligne, tous les caracteres apres "
    + "; seront ignores");

Lire.Filtre (false);
car = Lire.c();
while (car != '.')
{
    ecrire (car);
    if (car == ';')
    {
        Lire.Purge ();
    }
    car = Lire.c();
}
Lire.Purge ();
// il reste les caractères de fin de ligne '\r' et '\n' à traiter
// remettons le filtre comme nous l'avons trouvé en arrivant
Lire.Filtre (true);

Lire.Attente(); // Attente de la frappe de return
}
}

```

```

/*****
Nom du programme   : Lire.java
Auteur            : Lécu Régis
Mise à jour       : février 2001, maj nov 2003 jcc
Fonction          : bibliothèque de fonctions de lecture

Pour compatibilité dans la progression avec "Le livre de Java comme
premier langage", des alias de chaque fonction sont donnés en fin de la
classe.
Tous les noms de fonctions devraient commencer par une minuscule. En
dehors de cette boîte à outils de fonctions, cette convention sera
scrupuleusement respectée.
*****/
public class Lire
{
    // Par défaut, la bibliothèque ne lit que le premier caractère de chaque ligne
    private static boolean filtre = true;

    // Filtre :
    // si leFiltre = vrai, seul le premier caractère de chaque ligne
    // sera lu par la fonction Lire.c() (appel automatique de la fonction Purge)
    // sinon tous les caractères sont lus, y compris les caractères de contrôle
    public static void Filtre ( boolean leFiltre)
    {
        filtre = leFiltre;
    }

    // Purge : élimine tous les caractères jusqu'à la fin de la ligne
    public static void Purge()
    {
        try
        {
            char car ;
            do
            {
                car = (char) System.in.read ();
            }
            while (car != '\n'); // arrêt sur caractère fin de ligne
        }
        catch (java.io.IOException e)
        {
            System.out.println ("Erreur de saisie");
            System.exit (0);
        }
    }
}

```

```

// Lecture d'une chaine terminée par un "RETURN" : saute la fin de ligne
public static String Chaine()
{
    char car ;
    String result = "";

    try
    {
        car = (char) System.in.read ();
        //lecture de la ligne jusqu'au retour charriot (13, 0xD)
        while (car != '\r')
        {
            result = result + car;
            car = (char) System.in.read ();
        }

        // Saut du fin de ligne (10, 0xA, '\n'
        System.in.skip (1);
    }
    catch (java.io.IOException e)
    {
        System.out.println ("Erreur de saisie");
        System.exit (0);
    }
    return result;
}

// Lecture d'un caractère : uniquement le premier caractère de la ligne
// si filtrage, n'importe quel caractère sinon
public static char Caractere ()
{
    char result = 0;

    try
    {
        result = (char) System.in.read ();
    }

    catch (java.io.IOException e)
    {
        System.out.println ("Erreur de saisie");
        System.exit (0);
    }

    if (filtre) // filtre actif
    {
        Purge (); // vider la mémorisation des caractères frappés
    }

    return result;
}

```

```

public static int Entier ()
{
    int result = 0;

    try
    {
        result = Integer.parseInt ( Chaine () );
    }

    catch (NumberFormatException e)
    {
        System.out.println ("Format entier incorrect !");
        System.exit(0);
    }

    return result;
}

public static short EntierCourt ()
{
    short result = 0;

    try
    {
        result = Short.parseShort ( Chaine () );
    }

    catch (NumberFormatException e)
    {
        System.out.println ("Format entier incorrect !");
        System.exit(0);
    }

    return result;
}

public static long EntierLong ()
{
    long result = 0;

    try
    {
        result = Long.parseLong ( Chaine () );
    }

    catch (NumberFormatException e)
    {
        System.out.println ("Format entier incorrect !");
        System.exit(0);
    }

    return result;
}

```

```

public static float Reel ()
{
    float result = 0;

    try
    {
        result = Float.valueOf( Chaine() ).floatValue () ;
    }

    catch (NumberFormatException e)
    {
        System.out.println ("Format reel incorrect!");
        System.exit(0);
    }

    return result;
}

public static double ReelDouble ()
{
    double result = 0;

    try
    {
        result = Double.valueOf( Chaine() ).doubleValue () ;
    }

    catch (NumberFormatException e)
    {
        System.out.println ("Format reel incorrect!");
        System.exit(0);
    }

    return result;
}

// Attente : permet de visualiser les résultats avant la sortie
// de l'application.
public static void Attente()
{
    System.out.println ();
    System.out.println ("*** Tapez Entree pour Terminer ***");
    Lire.c();
}

// Attente : permet de visualiser les résultats avant la suite
// de l'application.
public static void Suite()
{
    System.out.println ();
    System.out.println ("*** Tapez Entree pour Continuer ***");
    Lire.c();
}

public static boolean Question(String msg)
{
    char reponse ;

    do
    {
        System.out.print (msg + " (O/N ) ?" );
        reponse = Lire.c();
    }while ((reponse!='O') && (reponse!='o') && (reponse!='n') && (reponse!='N'));
    // arrêt quand reponse est égal à O,o,N,n
    return (reponse == 'O') || (reponse == 'o') ;
}

// Alias des fonctions : pour compatibilité avec le livre

```

```
// "Le livre de Java comme premier langage"

public static String S ()
{
    return Chaine();
}

public static short s ()
{
    return EntierCourt();
}

public static long l ()
{
    return EntierLong();
}

public static int i ()
{
    return Entier();
}

public static char c ()
{
    return Caractere();
}

public static float f ()
{
    return Reel ();
}

public static double d ()
{
    return ReelDouble ();
}

}
```



## Classes d'habillage des types simples pour le passage des paramètres en entrée sortie

```
public class Booleen
{
    // cette classe permet d'encapsuler une valeur
    // en un objet pour le passage de paramètres
    private boolean val=false;
    public boolean getVal()
    {
        return val;
    }
    public void setVal(boolean valeur)
    {
        val = valeur;
    }
}
```

```
public class Caractere
{
    // cette classe permet d'encapsuler une valeur
    // en un objet pour le passage de paramètres
    private char val='\0';
    public char getVal()
    {
        return val;
    }
    public void setVal(char valeur)
    {
        val = valeur;
    }
}
```

```
public class Entier
{
    // cette classe permet d'encapsuler une valeur
    // en un objet pour le passage de paramètres
    private int val=0;
    public int getVal()
    {
        return val;
    }
    public void setVal(int valeur)
    {
        val = valeur;
    }
}
```

```
public class EntierCourt
{
    // cette classe permet d'encapsuler une valeur
    // en un objet pour le passage de paramètres
    private short val=0;
    public short getVal()
    {
        return val;
    }
    public void setVal(short valeur)
    {
        val = valeur;
    }
}
```

```
public class EntierLong
{
    // cette classe permet d'encapsuler une valeur
    // en un objet pour le passage de paramètres
    private long val=0;
    public long getVal()
    {
        return val;
    }
    public void setVal(long valeur)
    {
        val = valeur;
    }
}
```

```
public class Octet
{
    // cette classe permet d'encapsuler une valeur
    // en un objet pour le passage de paramètres
    private byte val=0;
    public byte getVal()
    {
        return val;
    }
    public void setVal(byte valeur)
    {
        val = valeur;
    }
}
```

```
public class Reel
{
    // cette classe permet d'encapsuler une valeur
    // en un objet pour le passage de paramètres
    private double val=0;
    public double getVal()
    {
        return val;
    }
    public void setVal(double valeur)
    {
        val = valeur;
    }
}
```

```
public class ReelCourt
{
    // cette classe permet d'encapsuler une valeur
    // en un objet pour le passage de paramètres
    private float val=0;
    public float getVal()
    {
        return val;
    }
    public void setVal(float valeur)
    {
        val = valeur;
    }
}
```

## Projet de développement d'une application

Nous nous proposons de développer une application en java, mettant en œuvre tout ce que vous avez appris dans le module d'algorithmie et programmation java.

Cette application se déroule en 5 temps. Les deux premières étapes sont importantes, les suivantes, un peu plus difficiles, seront mises en œuvre en fonction du temps dont vous disposez.

Nous nous proposons de réaliser un jeu de Mastermind ( marque déposée Hasbro Inc. ). Ici la partie graphique du jeu ne sera pas réalisée, car nous n'avons pas les éléments pour le faire. Il sera particulièrement intéressant de reprendre ce projet ultérieurement en développement objet avec une interface graphique. Les options étudiées ici vous permettront de réaliser une application bien plus complète que celles que vous trouverez sur le web...

### Le jeu de mastermind

Le but d'une partie de mastermind est de découvrir une combinaison cachée ( de couleurs dans le jeu du commerce, mais ici, avec une application console, nous prendrons une combinaison de chiffres ).

Pour cela nous avançons une proposition, le jeu nous donne alors le nombre d'éléments de la combinaison qui sont bien placés ( par rapport à la combinaison cachée ) et le nombre d'éléments mal placés ( un élément n'étant compté qu'une seule fois ).

Ces indications nous permettent de faire des propositions plus pointues, et par déduction de découvrir petit à petit la combinaison.

### Les règles du mastermind

La combinaison est composée de 4 chiffres de 0 à 7. Un chiffre peut se trouver plusieurs fois dans la combinaison cachée ( 7177, 4353, ... ).

Le joueur dispose de huit propositions pour trouver la solution. Si la solution n'est pas trouvée, elle sera montrée au joueur.

La proposition du joueur doit être conforme à ce que peut être la combinaison cachée. la proposition 8532 n'est donc pas correcte car le 8 n'appartient pas à la solution. De même une proposition partielle n'est pas possible ( .5.6 ).

### Déroulement d'une partie

combinaison cachée :        2 7 0 4

proposition n°1	1 2 3 4	* °	// * = 1 chiffre bien placé, ° = 1 chiffre mal placé
proposition n°2	1 5 2 6	°	// 1 chiffre mal placé
proposition n°3	3 2 7 0	°°°	// 3 chiffres mal placés
proposition n°4	2 0 3 2	*°	// 1 chiffre bien placé, 1 chiffre mal placé
proposition n°5	0 1 3 7	°°	// 2 chiffres mal placés
proposition n°6	2 7 0 4	****	// 4 chiffres bien placés, c'est gagné !

### Travail demandé

- 1) Programmer le jeu de mastermind. L'ordinateur génère une combinaison à trouver. Le candidat fait ses propositions, et à chaque proposition, l'ordinateur lui donne le nombre de chiffres bien placés et le nombre de chiffres mal placés. Le jeu s'arrête quand la combinaison a été trouvée, ou quand le joueur a fait 8 propositions.

Le jeu doit permettre d'enchaîner plusieurs parties sans relancer le programme et avoir accès à une page d'aide.

Il vous est conseillé de réafficher complètement les propositions du joueur à chaque nouvelle proposition ( ainsi que les réponses de l'ordinateur ). Cela évite les problèmes du travail en mode rouleau sur une console, et cela permet de garder un écran propre.

Vous pouvez prévoir une fonction qui traite une partie, paramétrée par le nombre de chiffres de la combinaison, le nombre de chiffres, et le nombre de coups autorisés.

Vous pouvez également prévoir une fonction de génération aléatoire de la combinaison cachée ( voir la bibliothèque de fonction Math, particulièrement la fonction random() ).

Vous pouvez prévoir une fonction de comparaison de la solution cachée avec une proposition ( son algorithme vous est donné en annexe ).

Vous pouvez prévoir une fonction d'affichage des « \* » et « ° » en réponse aux propositions.

Vous penserez aussi à définir la structure de donnée nécessaire pour conserver toutes les informations sur la partie en cours.

- 2) Votre jeu propose maintenant la possibilité de jouer soit au mastermind, soit au supermastermind. Le superMastermind est caractérisé par le fait de chercher une combinaison de 5 chiffres parmi 9 ( de 0 à 8 ) et cela en 10 coups maximum. Si le travail précédent est bien fait, cet exercice ne prend pas plus d'une demi heure. Le temps que vous passerez sera caractéristique de la qualité et du professionnalisme du travail réalisé dans la phase précédente.

- 3) (**optionnel**) Nous voulons mettre en place un fichier des records. Les dix premiers joueurs des différents jeux ( un fichier par type de partie ) seront répertoriés dans le fichier. Les joueurs seront classés par nombre de coups croissant, et pour un même nombre de coups, par le temps.

Le temps sera pris à chaque début de partie, et à chaque fin de partie. la différence des deux donnera le nombre de millisecondes de durée de la partie.( voir la fonction currentTimeMillis() de la boîte à outils System )

- 4) (**optionnel**) Constatant que certains joueurs perdent leur motivation en cours de partie, nous nous proposons de leur afficher, à coté de leur dernière proposition, le nombre de solutions possibles, compatibles avec l'ensemble des coups joués. A chaque nouvelle proposition ce nombre décroît rapidement, et donne au joueur l'illusion que le dénouement est proche. Ici, nous vous donnons le principe de ce travail.

Regardons une partie :

proposition	résultat	solutions possibles
1 2 3 4	°	976
2 5 6 7	°°	228
0 2 5 0		32
6 7 1 1	**°	2

Il ne reste plus que 2 solutions possibles ( sur les 4096 du départ (  $8^4$  ) ). Il devient excitant pour le joueur de trouver les deux combinaisons possibles ( 7 7 1 6 et 6 6 7 1 ) puis de parier sa chemise sur l'une des deux.

A n'importe quel stade du jeu nous pouvons calculer le nombre de coups possibles, pour cela il « suffit » d'essayer toutes les combinaisons possibles sur les propositions déjà jouées. La solution, quand nous la comparons à toutes les propositions, donne le résultat indiqué. Donc tous les essais qui réagissent comme la solution sont des solutions potentielles.

Il y a un léger inconvénient, c'est que c'est un peu long à faire. Mais un programme peut le faire relativement vite.

Comment générer toutes les solutions possibles ?

Il suffit de compter de 0 à 7777 (  $8^4 - 1$  ) dans la base 8. Pour chaque nombre on prend les restes successifs de la division dans la base pour reconstituer le nombre. Bien sûr il faut faire le même travail pour le supermastermind, de 0 à 8888 (  $9^5 - 1$  ) dans la base 9.

Pour chaque nombre il faut le comparer à chaque proposition, et le valider comme solution potentielle s'il réagit aux comparaisons comme la solution.

Ainsi nous avons calculé le nombre de solutions potentielles restantes.

- 5) **( optionnel )** Nous allons donner la possibilité aux joueurs qui sont coincés, non pas d'abandonner, mais de faire jouer l'ordinateur à leur place. Bien sûr le coup ne pourra pas être inscrit dans le fichier des records. Lors de la frappe de la combinaison, le joueur pourra rentrer un caractère spécial pour faire jouer l'ordinateur.

Lorsque nous avons calculé le nombre de solutions possibles, nous aurons conservé ces solutions, et il suffit de tirer une de ces solutions aléatoirement et de la jouer à la place du joueur.

## Procédure de comparaison d'une proposition avec la solution du mastermind

Définition de l'environnement de la procédure:

### constante

taille = 5 // nombre de pièces dans la combinaison cachée  
utilisé = '#' // marquage des pièces déjà comptabilisées

### type

code = **tableau** [ taille ] **de caractère**  
// type des codes cherchés et proposés

Définition de la procédure:

**procédure** mastermind ( **entrée** solution ,proposition : code;  
**sortie** juste, flou : **entier** )

// Cette procédure est une procédure du jeu de mastermind.  
//  
// Principes du jeu : soit une combinaison cachée de taille couleurs, cette  
// combinaison pouvant contenir plusieurs fois une ou plusieurs couleurs.  
// Le joueur doit essayer de deviner cette combinaison en faisant des propositions.  
// A chaque proposition le programme donne le nombre de couleurs placées au  
// bon endroit et le nombre de couleurs mal placées, une couleur n'étant comptée  
// qu'une fois. Donnons un exemple de déroulement du jeu.  
//

// Ici les couleurs sont représentées par des caractères de 'A' à 'J' et le nombre  
// taille vaut 5.

// combinaison cachée:      D      F      I      G      F

//

//                            propositions du joueur:                            justes                            floues

//

//                            A      B      C      D      E                            0                            1

//                            F      G      I      J      A                            1                            2

//                            H      I      J      I      A                            0                            1

//                            F      J      G      E      J                            0                            2

//                            I      G      F      C      G                            0                            3

//                            G      D      I      F      F                            2                            3

//                            D      F      I      G      F                            5                            0

//

// La procédure mastermind est la procédure qui à partir de la solution et d'une  
// proposition du joueur donne le nombre de pièces justes, et le nombre de pièces  
// mal placées( ou floues ).

//

// **solution** est la combinaison à découvrir

// **proposition** est la combinaison proposée par le joueur

// **juste** est le nombre de pièces justes dans la combinaison

// **flou** est le nombre de pièces mal placées dans la combinaison

//

// Cette procédure ne modifie ni la solution, ni la proposition du joueur.

#### variable

i : entier                    // indice de parcours de la solution

j : entier                    // indice de parcours de la proposition

#### début

// initialisation

juste := 0

flou := 0



// calcul du nombre de pièces justes

i := 1

**tantque** i <= taille **faire**

**si** solution[i] = proposition[i] **alors**

juste := juste + 1

// on a trouve une pièce juste

solution[i] := utilisé

// il ne faudrait pas la recompter

proposition[i] := utilisé

// celle-là non plus

**finsi**

i := i + 1

// on passe à la pièce suivante

**fintantque**

// calcul des pièces mal placées

// nous ne recomptons pas les pièces justes déjà marquées

i := 1

**tantque** i <= taille **faire**

// parcours de la solution

**si** solution[i] <> utilisé **alors** // la pièce n'a pas été comptée

// parcourons la proposition pour voir si la pièce est mal placée

j := 1

**tantque** ( j <= taille ) **et** ( solution[i] <> proposition[j] ) **faire**

j := j + 1

**fintantque**

// si j > taille alors la pièce n'appartient pas à la solution

**si** j <= taille **alors**

proposition[j] := utilisé

flou := flou + 1

// une pièce mal placée

**finsi**

**finsi**

i := i + 1

// pièce suivante dans la solution

**fintantque**

**fin**