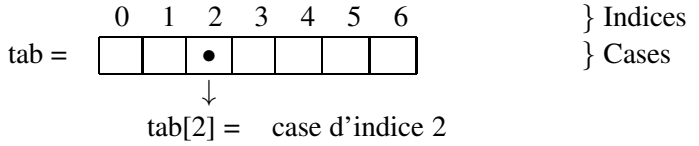
Chapitre 5

Tableaux

Jusqu'ici, nous avons employé les variables pour stocker les valeurs *individuelles* de types primitifs : une variable de type `int` pour stocker un entier, une variable de type `boolean` pour un booléen, etc.

Un **tableau** est *une structure regroupant plusieurs valeurs de même type*, chacune étant stockée dans un espace particulier appelé une *case* du tableau. On peut traiter un tableau comme un tout, ou case par case. Traité comme un tout, on pourra le stocker dans une variable, le passer en paramètre ou le donner en résultat d'un calcul. Chaque *case* est désignée individuellement via son *indice*, qui correspond à sa position dans le tableau, et peut être traitée comme *variable individuelle* : on pourra consulter sa valeur, la modifier, etc.



tab[0], tab[1], tab[2], tab[3], ..., tab[6] } 7 Variables (cases)

Les tableaux sont des structures des données présentes dans tous les langages de programmation.

5.1 Spécificités des tableaux

Un tableau n'existe pas tant qu'on ne l'a pas créé. Les valeurs des types primitifs de Java (int, double, boolean, char) existent sans qu'il soit nécessaire de les créer. Il n'est pas nécessaire de créer la valeur 17 ou la valeur true. Pour les tableaux, c'est différent : il faut les créer pour qu'ils existent et il y a un opérateur pour le faire : l'opérateur **new**.

Une tableau est une valeur d'un type. Ce type dépend de ce que l'on met dans le tableau. On a ainsi le type des tableaux qui contiennent des nombres entiers ou tableaux d'int, le type des tableaux qui contiennent des caractères char, etc. Pour chaque type qui existe en Java, on peut créer un tableau qui contient plusieurs valeurs de ce type, chacune étant stockée dans une case du tableau.

Le type d'un tableau d'entier se note int[], celui d'un tableau de char se note char[]. Plus généralement, si un type Java se note T, le type des tableaux contenant une valeur de ce type dans chaque case se note T[].

Un type tableau, comme int[] par exemple, peut servir à déclarer une variable.

```
int[] var;
```

Il n'y a rien de nouveau ici, c'est une déclaration de variable classique avec le type de la variable, puis son nom.

Cette déclaration ne crée pas de tableau. Elle crée un nom que l'on pourra utiliser pour désigner un tableau. Ce nom est associé à un espace dans la mémoire qui pourra contenir un tableau.

Un tableau a une taille fixée lors de sa création, c'est le nombre de cases qu'il contient. Un fois créé, le tableau ne peut plus jamais changer de taille : on ne peut ni lui ajouter ni lui enlever de cases.

Pour créer un tableau, on utilise new suivi du type des éléments et du nombre de cases entre crochets.

```
var = new int[10];
```

L'expression `new int[10]` crée un nouveau tableau de 10 cases, chaque case pouvant contenir un nombre entier, et elle renvoie ce tableau. Le reste de la ligne est une affectation normale qui permet de donner une valeur à la variable `var`.

A partir de ce moment, il est possible d'utiliser chaque case du tableau pour y stocker une valeur. Chaque case se comporte comme une variable de type int.

```
tab[0]=10;
tab[1]=tab[0]+1;
```

5.2 Déclaration et création

En Java, avant d'utiliser un tableau, il faut :

1. **Déclarer** une variable de type tableau (symbole `[]`), en indiquant le type T de ses futures cases ;

   ```
   T [] tab;          // tab est declare tableau de T
   ```

2. **Créer explicitement** la structure du tableau en mémoire (opération `new`), en donnant sa taille et le type T de ses éléments. Cette taille ne pourra plus changer : en Java les tableaux sont de *taille fixe*.

   ```
   tab = new T[taille];
   ```

3. L'initialisation des cases avec des valeurs par défaut, est réalisée implicitement par l'opération de création.

Étudions plus en détail ces étapes.

Déclaration

L'instruction :

```
T [] tab;
```

déclare une variable `tab` *destinée à contenir* un tableau, dont les cases seront de type T. Après déclaration, la variable `tab` existe, mais **n'est pas encore initialisée** à un tableau.

Exemples

:

```
int []  tabNum;      // tabNum est un tableau avec cases de type int
double [] t;         // t est un tableau avec cases de type double
String [] m;         // m est un tableau avec cases de type String
tabNum[0] = 5;       // provoque une erreur: le tableau n'existe pas
```

Après ces déclarations, les variables `tabNum`, `t` et `m` existent, mais pas encore la suite de cases que chacune d'entre elles pourra désigner. Par exemple, il est impossible de modifier la première case de `tabNum` (notée `tabNum[0]`) : elle n'existe pas encore. Le compilateur signale l'erreur :

```
Test.Java:7: variable tabNum might not have been initialized
    tabNum[0] = 5;
    ^
```

Création

L'opération de création :

```
new T[n];
```

réalise la création et l'initialisation d'un tableau de n cases de type T :

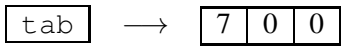
1. *Allocation en mémoire* d'un espace suffisant pour stocker n cases de type T.
2. *Initialisation* des cases du tableau avec des valeurs par défaut.

Les tableaux en Java sont de taille fixe. Une fois le tableau créé, l'espace qui lui est alloué en mémoire ne peut pas changer. Par exemple, il est impossible d'ajouter ou d'enlever des cases d'un tableau.

Exemple 1 : Déclaration, puis création du tableau `tab` avec trois entiers.

```
int []  tab;          // Declaration
tab = new int[3];     // Creation
tab[0] = 7;
```

Après l'instruction de création `new`, la variable `tab` est initialisée à un tableau contenant trois entiers. Après l'affectation `tab[0] = 7`, la structure du tableau en mémoire est :



Il est possible de réunir la déclaration et la création d'un tableau en une seule instruction. On pourra ainsi déclarer et créer le tableau de l'exemple 1 par :

Exemple 2 : Déclaration et création en une seule instruction.

```
int []  tab = new int[3];
```

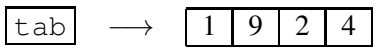
Initialisation par une liste de valeurs

Lorsqu'un tableau est de petite taille, il est possible de l'initialiser en donnant la liste des valeurs de chaque case. On utilise la notation $\{v_0, v_1, \ldots v_n\}$, où v_i est la valeur à donner à la case i du tableau. Nous reprendrons souvent cette notation pour **regrouper en une seule instruction** la déclaration, la création et l'initialisation d'un tableau.

Exemple 3 : Déclaration, création et initialisation d'un tableau en une seule instruction.

```
int []  tab = {1,9,2,4};
```

Il est alors **inutile de réaliser une création explicite** via `new` : elle se fait automatiquement à la taille nécessaire pour stocker le nombre des valeurs données. En mémoire on aura :



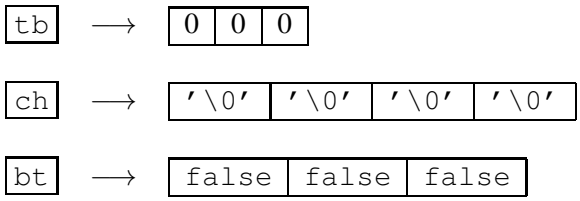
Valeurs par défaut à la création

Lors de la création, les cases d'un tableau sont initialisées avec des valeurs par défaut :
— les cases `boolean` sont initialisées à `false`.
— les cases numériques sont initialisées à 0.
— les cases `char` sont initialisées au caractère nul `'\0'`.
— les cases *référence* [1] sont initialisées à la valeur `null` (référence nulle).

Exemple 4 : Valeurs par défaut dans les cases

```
int []  tb = new int[3];
char [] ch = new char[4];
boolean [] bt = new boolean[3];
```

L'opération `new` initialise les cases de ces tableaux par :



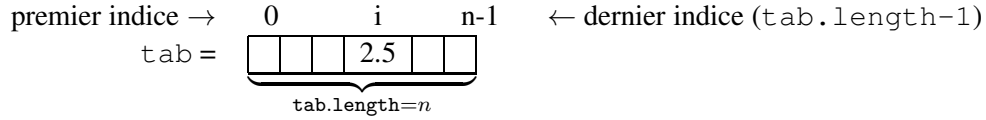
Longueur d'un tableau

La *taille* ou *longueur* d'un tableau est le nombre n des cases qu'il contient. Supposons que `tab` désigne un tableau de taille n. On peut obtenir sa longueur par la notation `tab.length`. Les indices du tableau `tab` sont alors compris entre 0 et `tab.length-1`. Cette notation sera souvent employée pour fixer la valeur maximale qui peut prendre l'indice d'un tableau (voir exemple 6).

1. Voir le chapitre dédié aux objets.

5.3 Accès aux cases

L'accès à une case de tableau permet de traiter cette case comme n'importe quelle variable individuelle : on peut modifier sa valeur dans le tableau, l'utiliser pour un calcul, un affichage, etc. L'accès d'une case se fait via son *indice ou position* dans le tableau. En Java, la première position a pour indice 0, et la dernière, a l'indice n-1 (taille du tableau moins un).



tab[i] vaut 2.5

L'accès aux cases de `tab` n'a de sens que pour les indices dans l'intervalle [0,. . ., tab.length-1]. Si i est un indice compris dans cet intervalle :

— `tab[i]` : est un **accès à la case de position i dans** `tab`. On peut consulter ou modifier cette valeur dans le tableau. *Exemple* : `tab[i] = tab[i] + 1;`

— **accès en dehors des bornes du tableau** `tab` : si `j` n'est pas compris entre 0 et tab.length-1, l'accès `tab[j]` provoque une erreur à l'exécution : l'indice j et la case associée n'existent pas dans le tableau. Java *lève l'exception* `ArrayIndexOutOfBoundsException`.

Exemple 5 : Modification d'une case, accès en dehors des bornes d'un tableau.

```
public class Test {
   public static void main (String args[]) {
      double [] tab = {1.0, 2.5, 7.2, 0.6}; // Creation
      // Affichage avant modification
      Terminal.ecrireString("tab[0] avant = ");
      Terminal.ecrireDoubleln(tab[0]);
      tab[0] = tab[0] + 4;
      // Affichage apr\'es modification
      Terminal.ecrireString("tab[0] apres = ");
      Terminal.ecrireDoubleln(tab[0]);
      // tab[5] = 17; // Erreur: indice en dehors des bornes
   }
}
```

Ce programme affiche la valeur de la première case de `tab`, avant et après modification.

```
Java/Essais> Java Test
tab[0] avant = 1.0
tab[0] apres = 5.0
```

Si nous enlevons le commentaire de la ligne 11, l'exécution se termine par une erreur : l'indice 5 est en dehors des bornes du tableau.

```
Java/Essais> Java Test
tab[0] avant = 1.0
tab[0] apres = 5.0
Exception in thread "main" Java.lang.ArrayIndexOutOfBoundsException: 5
        at Test.main(Test.Java:9)
```

5.3.1 Calcul du numéro de case

Le numéro de case dans une expression peut être le résultat du calcul d'une expression. Par exemple, il est possible d'utiliser le contenu d'une variable de type int comme numéro de case.

Exemple 5 bis : affichage d'une case de tableau

Cet exemple utilise la méthode `Math.random()` qui renvoie un nombre tiré au sort entre 0 et 1.

```
public class LeNumero{
   public static void main(String[] args){
  double[] values = new double[3];
  int lenumero;
  values[0]=Math.random();
  values[1]=Math.random();
  values[2]=Math.random();
  Terminal.ecrireString("Quel numéro de case? ");
  lenumero = Terminal.lireInt();
  Terminal.ecrireString("Contenu de la case: ");
  Terminal.ecrireDoubleln(values[lenumero]);
   }
}
```

Et voici deux exécutions de ce programme.

```
> Java LeNumero
Quel numéro de case? 2
Contenu de la case: 0.5789990719784663
> Java LeNumero
Quel numéro de case? 5
Contenu de la case: Exception in thread "main" Java.lang.ArrayIndexOutOfBoundsExc
        at LeNumero.main(LeNumero.Java:11)
```

Dans cet exemple, le calcul consiste à utiliser le contenu d'une variable, mais on peut tout aussi avoir un calcul avec opérateur ou avec un appel de méthode qui renvoie un nombre entier.

```
  int nb = 2;
  char[] tab = new char[15];
  tab[nb*2] = 57;
  tab[Math.min(nb,17)] = 8;
  tab[Math.min(nb,17)*2-1] = 5;
```

5.3.2 Affinité entre tableaux et boucles for

Le numéro de case peut être cherché dans une variable. Il arrive souvent qu'on utilise une boucle for pour effectuer une opération sur chacune des cases d'un tableau. On utilise alors la variable de la boucle for pour désigner un numéro de case. A chaque tour de boucle, l'opération est effectuée sur une des cases du tableau et la boucle est parcourue autant de fois qu'il y a de cases dans le tableau.

Exemple 6 : Parcours pour affichage d'un tableau

La boucle fait varier l'indice du tableau `tab` entre `i= 0` et `i = tab.length-1`.

Notons qu'on ne peut pas afficher un tableau directement avec System.out.print ni avec Terminal.ecrireXXX : ces méthodes ne fonctionnent que pour les types int, double, boolean, char et String. Pour afficher un tableau, il faut faire comme dans le programme qui suit un affichage case par case au moyen d'une boucle for.

```
public class AfficheTab {
   public static void main (String args[]) {
      int[] tab = {10,20,30,40};
      for (int i=0; i< tab.length; i++) {
          Terminal.ecrireStringln(tab[i]);
      }
   }
}
```

Ce programma affiche :

```
Java/Essais> Java AfficheTab
tab[0] =  10
tab[1] =  20
tab[2] =  30
tab[3] =  40
```

Une erreur commune dans une boucle, est de fixer le dernier indice à `i <= tab.length` plutôt qu'à `i <= tab.length -1`. Si nous changeons la condition de la boucle de cette manière, l'exécution produit une erreur : l'indice `tab.length`, égal à 4 ici, n'existe pas dans `tab`.

```
Java/Essais> Java AfficheTabErr
tab[0] =  10
tab[1] =  20
tab[2] =  30
tab[3] =  40
Exception in thread "main" Java.lang.ArrayIndexOutOfBoundsException: 4
        at AfficheTabErr.main(AfficheTabErr.Java:5)
```

Exemple 7 : Boucles d'initialisation et d'affichage.

On souhaite initialiser un tableau avec des notes d'élèves saisies au clavier. Le programme demande le nombre de notes à saisir, et crée un tableau `lesNotes` de cette taille. Une première boucle initialise le tableau ; la boucle suivante affiche son contenu. Les itérations se font dans l'intervalle de `i=0` jusqu'à `i <= lesNotes.length-1`.

```
public class Notes {
   public static void main (String args[]) {
      int nombreNotes;
      Terminal.ecrireString("Nombre de notes a saisir? ");
      nombreNotes = Terminal.lireInt();
      double [] lesNotes = new double[nombreNotes];
      // Initialisation
```

```
        for (int i=0; i<= lesNotes.length -1; i++) {
            Terminal.ecrireString("Note␣no.␣" + (i+1) + "?␣");
            lesNotes[i] = Terminal.lireDouble();
        }
        // Affichage
        Terminal.sautDeLigne();
        Terminal.ecrireStringln("Notes␣dans␣le␣tableau:");
        Terminal.ecrireStringln("**********************");
        for (int i=0; i<= lesNotes.length -1; i++) {
            Terminal.ecrireString("Note␣no.␣" + (i+1) + "␣=␣");
            Terminal.ecrireDoubleln(lesNotes[i]);
        }
    }
}
```

```
Java/Essais> Java Notes
Nombre de notes a saisir? 4
Note no. 1? 7.6
Note no. 2? 11
Note no. 3? 14
Note no. 4? 5

Notes dans le tableau:
**********************
Note no. 1 = 7.6
Note no. 2 = 11.0
Note no. 3 = 14.0
Note no. 4 = 5.0
```

5.4 Exemples d'utilisation de tableaux

Exemple 10 : Recherche du minimum et maximum dans un tableau d'entiers.

Il s'agit d'un exemple classique, correspondant à un type de problème courant.

Le cœur de l'algorithme consiste à conserver dans deux variables `min` et `max` le plus petit et le plus grand élément qu'on a vu dans la partie du tableau déjà visitée. Un tour de boucle consiste à regarder une case du tableau et mettre à jout les variables où cas où le contenu de la case serait plus petit que le contenu de `min` ou plus grand que le contenu de `max`.

Pour initialiser le processus, on considère le contenu de la première case : c'est nécessairement à la fois le plus petit et le plus grand vu jusqu'à ce point, vu que c'est le seul. Pour cette raison, on initialise les deux variables `min` et `max` avec le premier élément du tableau. La boucle doit ensuite parcourir une à une les autres cases. La comparaison se fait à partir du deuxième élément : c'est pourquoi l'indice i débute à `i=1`.

```
1 public class minMax {
2    public static void main (String args[]) {
3       int n;
4       Terminal.ecrireString("Combien␣des␣nombres␣a␣saisir?␣");
```

```
5         n = Terminal.lireInt();
6         int [] tab = new int[n];
7         // Initialisation
8         for (int i=0; i<= tab.length -1; i++) {
9             Terminal.ecrireString("Un entier? ");
10            tab[i] = Terminal.lireInt();
11        }
12        // Recherche de min et max
13        int min = tab[0];
14        int max = tab[0];
15        for (int i=1; i<= tab.length -1; i++) {
16          if (tab[i] < min) {
17            min = tab[i];
18          }
19          if (tab[i] > max) {
20            max = tab[i];
21          }
22        }
23        Terminal.ecrireStringln("Le minimum est: " + min);
24        Terminal.ecrireStringln("Le maximum est: " + max);
25    }
26 }
```

Notons que dans ce programme, l'algorithme proprement dit occupe les lignes 12 à 18. Il est précédé par une suite d'instructions servant à créer et initisaliser le tableau et suivi d'un affichage des résultats. Ce découpage en trois phases – lecture, calculs, affichage – est très courant et souvent recommandé car il permet de séparer le cœur de l'algorithme des opérations d'interface avec l'utilisateur.

Voici une exécution du programme :

```
Java/Essais> Java minMax
Combien des nombres a saisir? 5
Un entier? 5
Un entier? 2
Un entier? -6
Un entier? 45
Un entier? 3
Le minimum est: -6
Le maximum est: 45
```

Exemple 11 : Gestion de notes.

Nous modifions le programme de l'exemple 7 afin de calculer la moyenne des notes, la note minimale et maximale, et le nombre de notes supérieures ou égales à 10. Nous reprenons la boucle de recherche du minimum et maximum de l'exemple 10.

```
public class Notes {
   public static void main (String args[]) {
```

```
      int nombreNotes;
      Terminal.ecrireString("Nombre de notes a saisir? ");
      nombreNotes = Terminal.lireInt();
      double [] lesNotes = new double[nombreNotes];
      // Initialisation
      for (int i=0; i<= lesNotes.length -1; i++) {
          Terminal.ecrireString("Note no. " + (i+1) + "? ");
          lesNotes[i] = Terminal.lireDouble();
      }
      double min = lesNotes[0];
      double max = lesNotes[0];
      double somme = 0;
      int sup10 = 0;
      for (int i=0; i<= lesNotes.length -1; i++) {
         if (lesNotes[i] < min) { min = lesNotes[i];}
         if (lesNotes[i] > max) { max = lesNotes[i];}
         if (lesNotes[i] >= 10) { sup10++;}
         somme = somme + lesNotes[i];
      }
      Terminal.ecrireString("La moyenne des notes est: ");
      Terminal.ecrireDoubleln(somme/nombreNotes);
      Terminal.ecrireStringln("Le nombre de notes >= 10 est: " + sup10);
      Terminal.ecrireStringln("La note minimum est: " + min);
      Terminal.ecrireStringln("La note  maximum est: " + max);
   }}
```

```
Java/Essais> Java Notes
Nombre de notes a` saisir? 4
Note no. 1? 5
Note no. 2? 8
Note no. 3? 10
Note no. 4? 15
La moyenne des notes est: 9.5
Le nombre de notes >= 10 est: 2
La note minimum est: 5.0
La note  maximum est: 15.0
```

Exemple 12 : Inversion d'un tableau de caractères.

La boucle d'inversion utilise deux variables d'itération i et j, initialisées avec le premier et le dernier élément du tableau. A chaque tour de boucle, les éléments dans i et j sont échangés, puis i est incrémenté et j décrémenté. Il y a deux cas d'arrêt possibles selon la taille du tableau : s'il est de taille impair, alors l'arrêt se produit lorsque i=j ; s'il est de taille pair, alors l'arrêt se fait lorsque `j < i`. En conclusion, la boucle doit terminer si `i >= j`.

```
public class Inversion {
   public static void main (String[] args) {
```

```
        int n;
        char [] t;
        Terminal.ecrireString("Combien␣de␣caracteres␣a␣saisir?␣");
        n = Terminal.lireInt();
        t = new char[n];
        // Initialisation
        for (int i=0; i<= t.length-1; i++) {
             Terminal.ecrireString("Un␣caractere:␣␣");
             t[i] = Terminal.lireChar();
        }
        // Affichage avant inversion
        Terminal.ecrireString("Le␣tableau␣saisi:␣");
        for (int i=0; i<= t.length-1; i++) {
             Terminal.ecrireChar(t[i]);
        }
        Terminal.sautDeLigne();
        // Inversion: arret si (i >= j)
        int i,j;
        char tampon;
        for (i=0, j= t.length-1 ; i < j; i++, j--) {
             tampon = t[i];
             t[i] = t[j];
             t[j] = tampon;
        }
        // Affichage final
        Terminal.ecrireString("Le␣tableau␣inverse:␣");
        for (int k=0; k<= t.length-1; k++) {
             Terminal.ecrireChar(t[k]);
        }
        Terminal.sautDeLigne();
    }
}
```

Pour échanger les valeurs de deux cases du tableau, on est obligé d'utiliser une variable pour stocker temporairement la valeur d'une des deux cases. Si l'on écrivait le code suivant :

```
t[i]=t[j];
t[j]=t[i];
```

on aurait juste recopié la valeur de `t[j]` dans `t[i]`. Quant à la valeur quavait initialement `t[i]`, elle est perdue après la première affectation. C'est donc la nouvelle valeur donnée à `t[i]` qui serait copiée dans `t[j]` par la deuxième affectation. Celle-ci ne sert donc à rien. Le code correct consiste à sauvegarder la valeur initiale de `t[i]` dans `tampon` :

```
tampon = t[i];
t[i] = t[j];
t[j] = tampon;
```

Un exemple d'exécution du programme `Inversion` :

```
Java/Essais> Java Inversion
Combien de caracteres a saisir? 5
Un caractere:  s
Un caractere:  a
Un caractere:  l
Un caractere:  u
Un caractere:  t
Le tableau saisit: salut
Le tableau inverse: tulas
```

5.5 Vérification d'une propriété

Lorsqu'on veut vérifier une propriété du tableau, c'est-à-dire lorsqu'on veut faire un programme qui calcule un résultat booléen, il y a une assymétrie entre les deux réponses.

Généralement, ce type de propriétés peut se résumer par une question dont la réponse est oui ou non. Voici des exemples de telles propriétés :

— Est-ce que tel élément appartient à tel tableau ?
— Est-ce que tous les nombres du tableau sont positifs ?
— Est-ce que tous les caractères d'un tableau de char sont des lettres ?
— Est-ce qu'il y a au moins une lettre parmi les éléments d'un tableau de char ?

Il arrive souvent que l'on ait à écrire un programme qui réponde à ce type de questions.

Développons l'exemple de la première question : est-ce que tel élément appartient à tel tableau ? Pour répondre oui à la question, il peut suffire de regarder une case, la bonne, celle qui contient l'élément en question. Pour répondre non, il est nécessaire d'avoir regardé toutes les cases et de n'avoir trouvé l'élément dans aucune d'entre elles.

Pour répondre oui, il faut être sûr qu'**il existe** une case contenant l'élément, alors que pour répondre non, il faut que **pour toute case**, l'élément n'est pas dans cette case. Donc la réponse oui (true) peut être déterminée dans le corps de la boucle, au moment où on ne regarde qu'une case, alors que la réponse non (false) ne peut en aucun cas être déterminée à l'intérieur de la boucle. Ce n'est qu'après la fin de la boucle que cette réponse peut être choisie.

Voyons une première version du programme qui parcourt toujours toutes les cases au moyen d'une boucle for.

Exemple 13 : recherche d'un élément avec un for

On utilise une variable booléenne `appartient` qui contiendra `false` tant qu'on n'a pas trouvé l'élément dans le tableau et `true` à partir du moment où on l'a trouvé. Cette variable est initialisée à `false` parce qu'au début du programme, aucune case n'a été vue et le nombre n'a pas encore été trouvé.

```
1 public class RechercheFor{
2    public static void main(String[] args){
3   int[] tab = {10,20,30,40};
4   int elem;
5   boolean appartient = false;
6   System.out.print("Entrez␣un␣nombre:␣");
7   elem = Terminal.lireInt();
8   for (int numcase=0; numcase < 4; numcase = numcase+1){
```

```
9        if (tab[numcase]==elem){
10       appartient = true;
11        }
12    }
13    if (appartient){
14       System.out.println(elem + " appartient au tableau");
15    }else{
16       System.out.println(elem + " n'appartient pas au tableau");
17    }
18     }
19 }
```

Ce serait une grave erreur de changer la boucle avec un else dans le cas où tab[numcase] est différent de `elem`. Le fait que l'élément ne soit pas celui qu'on cherche ne permet pas du tout de dire que l'élément n'appartient pas au tableau.

Exemple 14 : recherche d'un élément avec un while

On peut reprocher au programme qui parcourt toutes les cases du tableau de faire un travail inutile : à partir du moment où l'élément a été vu dans une case, on peut arrêter de parcourir le tableau. On peut alors préférer écrire une boucle while qui s'arrête lorsque l'élément est trouvé.

```
public class RechercheWhile{
   public static void main(String[] args){
   int[] tab = {10,20,30,40};
   int elem;
   boolean appartient = false;
   System.out.print("Entrez un nombre: ");
   elem = Terminal.lireInt();
   int numcase=0;
   while (numcase < 4 && appartient == false){
      if (tab[numcase]==elem){
      appartient = true;
       }
       numcase = numcase+1;
   }
   if (appartient){
      System.out.println(elem + " appartient au tableau");
   }else{
      System.out.println(elem + " n'appartient pas au tableau");
   }
    }
}
```

Lequel des deux programmes RechercheFor et RechercheWhile est meilleur ? Si le tableau est très grand, il peut y avoir un petit avantage à utiliser RechercheWhile en terme de temps d'exécution. Dans la plupart des cas, la différence ne sera pas perceptible et on peut employer indifféremment l'un ou l'autre. RechercheFor est plus simple à lire et RechercheWhile théoriquement plus efficace.