# Chapitre 11

# Représenter des données avec des tableaux

Nous allons voir dans ce chapitre comment utiliser des tableaux pour représenter des données, notamment quand le nombre d'éléments varie au fil de l'exécution et quand il faut enregistrer des données de types différents pour représenter une certaine donnée.

## 11.1 Un problème de taille

Il arrive couramment que l'on doive représenter une collection de données dont la taille varie au fil du temps, avec des opérations d'ajout et de suppression. Par exemple, le nombre d'employés d'une entreprise augmente avec les embauches et diminue avec les démissions, départs en retraite et licenciements. Dans un répertoire téléphonique, des contacts peuvent être ajoutés et supprimés. Des produits peuvent être ajoutés ou retirés du catalogue.

Lorsqu'on utilise un tableau pour stocker une telle collection, il se pose le problème d'adapter le tableau à la taille des données. En Java, les tableaux ne changent jamais de taille, mais les variables peuvent désigner successivement des tableaux de différentes tailles.

Dans la suite du chapitre, nous allons prendre l'exemple d'un tableau de nombres entiers et les deux opérations d'ajout d'un nombre ou de retrait d'un nombre.

### 11.1.1 Adapter la taille du tableau

Il est possible de mettre la liste dans une variable et de créer un nouveau tableau à chaque ajout et chaque retrait et affecter ce nouveau tableau à la variable. Il faut ensuite recopier les nombres de l'ancien au nouveau tableau.

```
public class SDD1{
    public static void main(String[] args){
        int[] liste = {474, 317, 215, 700, 895, 999};
        // retirer un nombre
        int indice = 0, nb;
        for (int i=0; i<liste.length; i=i+1){
            System.out.println(liste[i]+ " ");
        }
        System.out.println();
```

```
        System.out.println("Quel nombre voulez-vous retirer?");
        nb = Terminal.lireInt();
        // recherche du numéro de case du nombre
        while(indice<liste.length && liste[indice]!=nb){
            indice=indice+1;
        }
        if (indice == liste.length){ // on n'a pas trouvé le nombre
            System.out.println("Ce nombre n'est pas dans le tableau");
        }else{
            int[] nouveau = new int[liste.length-1];
            // recopier les noms avant le nombre à supprimer: même indice
            for (int i=0; i<indice; i++){
                nouveau[i]=liste[i];
            }
            // recopier les noms après nb: une case plus à gauche dans le nouveau
            for (int i=indice+1; i<liste.length; i++){
                nouveau[i-1]=liste[i];
            }
            // le nouveau tableau est prêt, on l'affecte à liste
            liste = nouveau;
            System.out.println("nombre supprimé");
        }
        // ajouter un nombre
        System.out.println("Quel nombre ajouter?");
        nb = Terminal.lireInt();
        int[] nouveau = new int[liste.length+1];
        // recopier tous les noms dans le nouveau tableau
        for (int i=0; i<liste.length; i=i+1){
            nouveau[i]=liste[i];
        }
        // ajouter nb dans la dernière case
        nouveau[nouveau.length-1]=nb;
        // le nouveau tableau est prêt, on l'affecte à liste
        liste = nouveau;
        // affichage de la liste
        for (int i=0; i<liste.length; i=i+1){
            System.out.println(liste[i]+ " ");
        }
        System.out.println();
    }
}
```

Cette technique d'adaptation de la taille du tableau aux besoins est intéressante dans son principe mais ne doit pas être utilisée à chaque opération d'ajout ou de suppression parce que les opérations à réaliser sont trop coûteuses. D'une part l'opération de création du tableau (new) est de loin l'opération qui prend le plus de temps. D'autre part, devoir recopier tous les noms d'un tableau à l'autre est également long, surtout si le tableau est grand.

On préférera utiliser un tableau plus grand que nécessaire avec certaines cases vides. Les opérations de changement de tableau seront réservées au cas où il n'y a plus de cases vides et on rajoutera d'un coup non pas une case, mais plusieurs cases (par exemple cent cases), dont certaines seront vides.

### 11.1.2 Gérer des cases vides : valeur spécifique

Une case d'un tableau ne peut pas vraiment être vide : elle contient toujours une valeur. En effet, une case de tableau est toujours une suite de bits en mémoire, chaque bit étant à 0 ou à 1 et une telle suite est toujours une valeur du type java correspondant. Dans notre exemple, dans un tableau d'entier il y a toujours un entier.

Une première façon de gérer les cases vides est d'utiliser une certaine valeur entière pour représenter le fait que la case est vide. On peut être tenté d'utiliser 0, parce que c'est la valeur par défaut des cases de tableaux d'entiers. Mais dans beaucoup de cas, 0 est une valeur que l'on peut vouloir mettre dans le tableau. Par exemple, s'il s'agit de notes, 0 est une note possible (bien que peu souhaitable !). On ne pourra donc pas utiliser 0 pour les cases vides. Dans ce cas, on pourra utiliser -1 car -1 n'est pas une note.

Dans d'autres applications où -1 peut être une valeur à stocker (par exemple s'il s'agit de température de l'air ambiant), il faudra utiliser une autre valeur. Par exemple 1000 car l'air ambiant ne peut pas être à 1000 degrés. Une solution qui marche souvent consiste à utiliser le plus petit de tous les entiers du type int. Il se note Integer.MIN_VALUE et il vaut -2147483648.

Pour ajouter un élément, on va parcourir le tableau jusqu'à la première case libre (qui contient Integer.MIN_VALUE) et y mettre le nombre à ajouter. Pour supprimer un élément, on va le remplacer dans le tableau par Integer.MIN_VALUE.

```
public class SDD2{
    public static void main(String[] args){
        // créer une liste avec 100 cases
        int[] init = {474, 317, 215, 700, 895, 999};
        int[] liste = new int[100];
        // mettre les nombres en début de liste
        for (int i=0; i<init.length; i=i+1){
            liste[i]=init[i];
        }
        // Remplir les autres cases (qui sont vides) avec Integer.MIN_VALUE
        for (int i=init.length; i<liste.length; i=i+1){
            liste[i]=Integer.MIN_VALUE;
        }
        // retirer un nombre
        int indice = 0, nb;
        for (int i=0; i<liste.length; i=i+1){
            // ne pas afficher les cases vides
            if (liste[i]!=Integer.MIN_VALUE){
                System.out.print(liste[i]+ " ");
            }
        }
        System.out.println();
        System.out.println("Quel nombre voulez-vous retirer?");
```

```
        nb = Terminal.lireInt();
        // recherche du numéro de case du nombre
        while(indice<liste.length && liste[indice]!=nb){
            indice=indice+1;
        }
        if (indice == liste.length){ // on n'a pas trouvé le nombre
            System.out.println("Ce nombre n'est pas dans le tableau");
        }else{ // le nombre a été trouvé en case indice
            liste[indice]=Integer.MIN_VALUE;
            System.out.println("nombre supprimé");
        }
        // ajouter un nombre
        System.out.println("Quel nombre ajouter?");
        nb = Terminal.lireInt();
        // chercher la première case libre
        indice = 0;
        while(liste[indice] != Integer.MIN_VALUE){
            indice = indice+1;
        }
        // remplir la case vide avec le nombre
        liste[indice]=nb;
        // affichage de la liste
        for (int i=0; i<liste.length; i=i+1){
            // ne pas afficher les cases vides
            if (liste[i]!=Integer.MIN_VALUE){
                System.out.print(liste[i]+ " ");
            }
        }
        System.out.println();
    }
}
```

### 11.1.3 Gérer des cases vides : regrouper les cases occupées

Une autre façon courante de gérer un tableau avec des cases vides consiste à occuper les premières cases du tableau et laisser libre celles de la fin du tableau. Entre les deux, une frontière qui correspond au nombre de cases occupées.

| 0 | 1 | 2 | | n-1 | n | | k-1 |
|---|---|---|---|---|---|---|---|
| X | X | X | ... | X | _ | ... | _ |

Ce schéma représente un tableau de taille $k$ avec $n$ cases occupées (figurées avec un X dedans) et des cases libres (figurées avec un souligné).

S'il y a n cases occupées dans un tel tableau, ce sont les cases numérotées de 0 à n-1 et la case numéro n est la première case libre. Si l'on doit ajouter un nouvel élément, ce sera dans cette nième case. Si l'on doit supprimer un élément, il faudra peut-être et même probablement ré-aménager le tableau car il faudra libérer la n-1ième case alors que l'élément supprimé n'était peut-être pas là. Il faut donc procéder en deux temps : recopier l'élément qui était dans la n-1ième case dans la case occupée par l'élément à supprimer. Puis libérer la n-1ième case.

La structure de donnée sera composée de deux variables : le tableau et le nombre de cases occupées dans le tableau (ce nombre sera dans une variable de type int).

```
public class SDD3{
    public static void main(String[] args){
        // créer une liste avec 100 cases
        int[] init = {474, 317, 215, 700, 895, 999};
        int[] liste = new int[100];
        int nbelem = init.length;
        // mettre les nombres en début de liste
        for (int i=0; i<init.length; i=i+1){
            liste[i]=init[i];
        }
        // retirer un nombre
        int indice = 0, nb;
        for (int i=0; i<nbelem; i=i+1){
            // on n'affiche que les cases occupées
            System.out.print(liste[i]+ " ");
        }
        System.out.println();
        System.out.println("Quel nombre voulez-vous retirer?");
        nb = Terminal.lireInt();
        // recherche du numéro de case du nombre
        while(indice<nbelem && liste[indice]!=nb){
            indice=indice+1;
        }
        if (indice == nbelem){ // on n'a pas trouvé le nombre
            System.out.println("Ce nombre n'est pas dans le tableau");
        }else{ // le nombre a été trouvé en case indice
            liste[indice]=liste[nbelem-1];
            nbelem=nbelem-1;
            System.out.println("nombre supprimé");
        }
        // ajouter un nombre
        System.out.println("Quel nombre ajouter?");
        nb = Terminal.lireInt();
        // remplir la première case vide avec le nombre
        liste[nbelem]=nb;
        nbelem = nbelem+1;
        // affichage de la liste
        for (int i=0; i<nbelem; i=i+1){
            System.out.print(liste[i]+ " ");
        }
        System.out.println();
    }
}
```

Notons que la façon de supprimer un élément ne préserve pas l'ordre d'origine des éléments

restants : c'est le dernier élément du tableau qui prend la place de l'élément supprimé. Si l'ordre des éléments dans la tableau est important, il faut procéder autrement : il faut décaler d'un cran à gauche tous les éléments situés à droite de l'élément supprimé. Cela peut se faire facilement avec une boucle for.

### 11.1.4 Structure de données et méthodes

La structure de données utilisée pour la gestion des cases libres comporte deux variables indissociables : le tableau et le nombre d'élément (`liste` et `nbelem` dans le code de l'exemple). Si on n'a que le nombre d'éléments, on ne connaît pas les nombres ; si on n'a que le tableau, on ne sait pas quelles cases sont libres ou occupées. Les deux informations sont nécessaires et leur cohérence doit être respectée par le code du programme.

On voudrait pouvoir faire des opérations sur les structures de données au moyen de méthodes : l'ajout, la suppression, l'affichage de la structure, etc. Les opérations qui modifient le nombre d'éléments posent un problème : une méthode peut modifier le contenu d'un tableau, elle ne peut pas modifier une variable de type int comme nbelem.

Voyons cela avec le code suivant.

```
public class SDD3Bis{
    public static void ajouter(int[] liste, int nb, int elem){
        liste[nb]=elem;
        nb=nb+1;
    }
    public static void afficher(int[] liste, int nb){
        if (nb == 0){
            System.out.print("La liste est vide");
        }
        for (int i=0; i<nb; i=i+1){
            System.out.print(liste[i]+" ");
        }
        System.out.println();
    }
    public static void main(String[] args){
        int[] liste = new int[100];
        int nbelem = 0;
        ajouter(liste,nbelem,474);
        afficher(liste,nbelem);
    }
}
```

Lorsqu'on exécute ce code, cela donne :

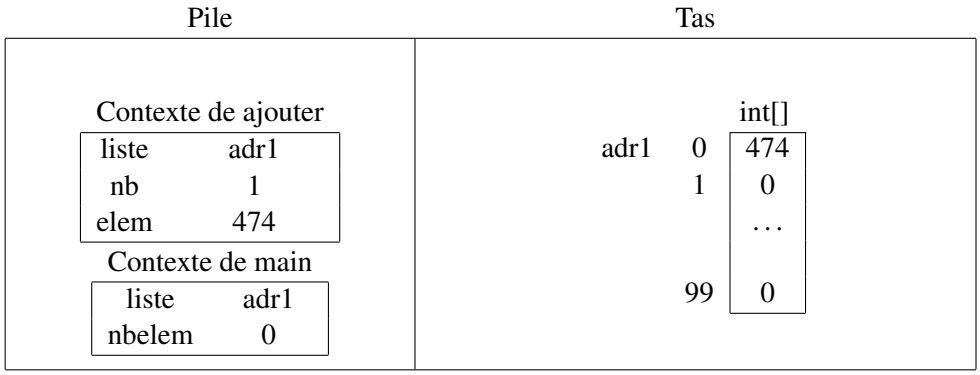
```
> java SDD3Bis
La liste est vide
```

L'ajout de l'élément 474 par la méthode `ajouter` ne s'est pas effectué correctement. Il y a deux affectations dans cette méthode. La première modifie le contenu de la case nb (dans le premier appel, la case 0) du tableau liste. Cette case est la même pour le paramètre liste d'ajouter et la variable liste de

main. Cette affectation change donc le contenu de la variable liste de main. La deuxième affectation change le paramètre nb d'ajouter, mais pas la variable nbelem de main qui reste à 0.

La case du tableau, en mémoire, est dans le tas alors que le paramètre nb et la variable nbelem sont dans la pile et donc locales à leurs méthodes respectives.

Le dession suivant illustre l'état de la mémoire à la fin de l'exécution de la méthode `ajouter`.




Comment faire en sorte que ajouter et main puisse partager un nombre d'éléments dans un espace mémoire commun ? Il faut que cet espace mémoire soit dans le tas. Une façon de procéder consiste à utiliser un tableau d'entier à une case pour enregistrer le nombre d'éléments de la liste. Cette case de tableau est partagée entre les différentes méthodes (ici, main et ajouter).

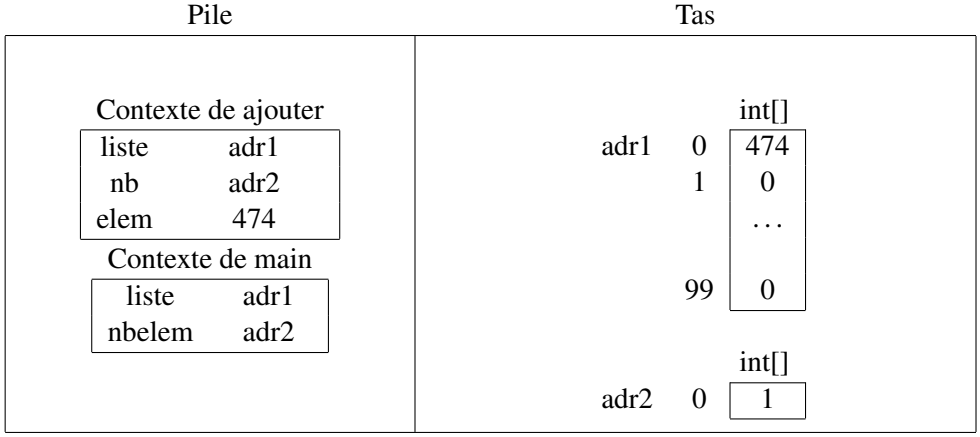
Voici le code que cela donne.

```
public class SDD3Ter{
    public static void ajouter(int[] liste, int[] nb, int elem){
        liste[nb[0]]=elem;
        nb[0]=nb[0]+1;
    }
    public static void afficher(int[] liste, int[] nb){
        if (nb[0] == 0){
            System.out.print("La liste est vide");
        }
        for (int i=0; i<nb[0]; i=i+1){
            System.out.print(liste[i]+" ");
        }
        System.out.println();
    }
    public static void main(String[] args){
        int[] liste = new int[100];
        int[] nbelem = {0};
        ajouter(liste,nbelem,474);
        afficher(liste,nbelem);
    }
}
```

L'exécution du programme est correcte.

```
> java SDD3Ter
474
```

Le dessin de la mémoire en fin d'exécution de la méthode ajouter est le suivant.




## 11.2 Avoir un tableau trié

On peut maintenir un certain ordre dans un tableau sans avoir besoin de le trier : il suffit de partir d'un tableau vide et de tenir compte de l'ordre lorsqu'on insère un nouvel élément.

Cette façon de procéder est intéressante parce que l'opération de tri est complexe et coûteuse.

Lorsqu'on veut insérer un élément dans un tableau trié, il faut trouver sa place dans le tableau et pour ce faire, le comparer aux éléments déjà présents.

### 11.2.1 Procédure d'insertion qui assure l'ordre des éléments

Cette recherche est simple : on compare l'élément à insérer à tous les éléments successifs du tableau jusqu'à trouver le premier élément qui est plus grand. C'est à la place de ce premier élément plus grand qu'il faut insérer le nouvel arrivant. Mais avant cela, il faut décaler tous les éléments plus grand d'un cran vers la droite pour lui faire une place.

Dans cet exemple, nous utilisons la gestion de cases libres avec le compteur de nombres d'éléments.

```
public class SDD4{
    // procédure qui insère elem dans tab en respectant l'ordre croissant
    public static void inserer(int[] tab, int[] nbelem, int elem){
        int indice = 0;
        while(indice<nbelem[0] && elem>tab[indice]){
            indice = indice+1;
        }
        // indice est le numéro où insérer elem
        // il faut d'abord décaler les éléments plus grands
        // vers la droite
        for (int i=nbelem[0]; i>indice; i=i-1){
            tab[i]=tab[i-1];
        }
        tab[indice]=elem;
        nbelem[0]=nbelem[0]+1;
```

```
    }
    public static void afficher(int[] liste, int[] nbelem){
        for (int i=0; i<nbelem[0]; i=i+1){
            System.out.print(liste[i]+ " ");
        }
        System.out.println();
    }
    public static void main(String[] args){
        // créer une liste avec 100 cases
        int[] liste = new int[100];
        int[] nbelem = {0};
        int nb;
        // mettre les nombres en début de liste
        inserer(liste,nbelem,474);
        inserer(liste,nbelem,317);
        inserer(liste,nbelem,215);
        inserer(liste,nbelem,700);
        inserer(liste,nbelem,999);
        inserer(liste,nbelem,895);
        // affichage de la liste
        afficher(liste,nbelem);
        // supprimer un nombre
        System.out.println("Quel nombre ajouter?");
        nb = Terminal.lireInt();
        inserer(liste,nbelem,nb);
        // affichage de la liste
        afficher(liste,nbelem);
    }
}
```

Exécution du programme :

```
> java SDD4
Quel nombre ajouter?
330
215 317 330 474 700 895 999
```

On peut améliorer l'efficacité de la recherche de la case où l'élément doit être inséré en faisant une recherche dichotomique à la place de la recherche séquentielle. Cette optimisation n'est utile que pour des gros tableaux et elle rend l'opération d'insertion plus complexe. C'est pourquoi nous ne la donnons pas ici.

### 11.2.2 Retirer un élément en conservant le tableau trié

Lorsqu'on retire un élément du tableau, il faut remplir la case qu'il occupait avec un autre élément sans perturber l'ordre des éléments du tableau. Il faut donc décaler tous les éléments plus grands que celui qu'on supprime d'un cran vers la gauche.

```
public class SDD4Bis{
```

```
// procédure qui insère elem dans tab en respectant l'ordre croissant
public static void inserer(int[] tab, int[] nbelem, int elem){
    int indice = 0;
    while(indice<nbelem[0] && elem>tab[indice]){
        indice = indice+1;
    }
    // indice est le numéro où insérer elem
    // il faut d'abord décaler les éléments plus grands
    // vers la droite
    for (int i=nbelem[0]; i>indice; i=i-1){
        tab[i]=tab[i-1];
    }
    tab[indice]=elem;
    nbelem[0]=nbelem[0]+1;
}
// procédure qui supprime un élément en maintenant l'ordre
// le résultat renvoyé dit si la suppression a été faite
public static boolean supprimer(int[] tab, int[] nbelem, int elem){
    int indice = 0;
    while(indice<nbelem[0] && elem>tab[indice]){
        indice = indice+1;
    }
    if (indice==nbelem[0] || elem!=tab[indice]){
        // l'élément n'a pas été trouvé dans le tableau
        return false;
    }else{
        // décaler les éléments plus grands
        for (int i =indice; i<nbelem[0]; i=i+1){
            tab[i]=tab[i+1];
        }
        // diminuer le nombre d'éléments
        nbelem[0]=nbelem[0]-1;
        return true;
    }
}
public static void afficher(int[] liste, int[] nbelem){
    for (int i=0; i<nbelem[0]; i=i+1){
        System.out.print(liste[i]+ " ");
    }
    System.out.println();
}
public static void main(String[] args){
    // créer une liste avec 100 cases
    int[] liste = new int[100];
    int[] nbelem = {0};
    int nb;
    // mettre les nombres en début de liste
```

```
        inserer(liste,nbelem,474);
        inserer(liste,nbelem,317);
        inserer(liste,nbelem,215);
        inserer(liste,nbelem,700);
        inserer(liste,nbelem,999);
        inserer(liste,nbelem,895);
        // affichage de la liste
        afficher(liste,nbelem);
        // supprimer un nombre
        System.out.println("Quel nombre supprimer?");
        nb = Terminal.lireInt();
        if (supprimer(liste,nbelem,nb)){
            System.out.println("Elément supprimé");
        }else{
            System.out.println(nb + " non trouvé dans le tableau.");
        }
        // affichage de la liste
        afficher(liste,nbelem);
    }
}
```

## 11.3 Coordonner plusieurs tableaux

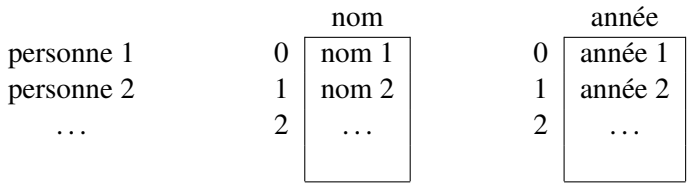
Si l'on a plusieurs informations du même type Java pour caractériser une certaine entité, on peut utiliser un tableau à deux dimensions. Par exemple, si l'on veut représenter le nom et le prénom d'une liste de personne, les deux informations sont de type String. On peut utiliser un tableau avec une colonne pour les noms, une autre colonne pour les prénoms et une ligne pour chaque personne.

| | | nom | prénom |
|---|---|---|---|
| personne 1 | 0 | nom 1 | prénom 1 |
| personne 2 | 1 | nom 2 | prénom 2 |
| ... | 2 | ... | ... |

Si maintenant les deux informations sont de types différents, on ne peut pas utiliser un tableau à deux dimensions car toutes les cases d'un tableau à deux dimensions contiennent des informations du même type.

Supposons que l'on veuille caractériser une personne avec son nom et son année de naissance. Le nom est un String et l'année est un int. On peut utiliser deux tableaux distincts au sens de Java, mais avec le même type de gestion que si ces deux tableaux différents étaient les deux colonnes d'un même tableau. C'est-à-dire qu'on utilisera le numéro de case (numéro de ligne) pour relier le nom et l'année de naissance d'une personne données.

Si l'on trouve le nom d'une personne dans la case numéro n du tableau des noms, alors on sait que l'année de naissance de la personne dont c'est le nom est dans la case numéro n du tableau des années de naissance.




```
public class SDD6{
    public static void ajouter(String[] nom, int[] annee, int[] nb,
                               String n, int a){
        nom[nb[0]]=n;
        annee[nb[0]]=a;
        nb[0]=nb[0]+1;
    }
    public static boolean supprimer(String[] nom, int[] annee, int[] nb,
                               String n, int a){
        int indice = 0;
        while(indice<nb[0] && (n!=nom[indice] || a!=annee[indice])){
            indice = indice+1;
        }
        if (indice<nb[0] && n==nom[indice] && a==annee[indice]){
            for (int i=indice; i<nb[0]; i=i+1){
                nom[i]=nom[i+1];
                annee[i]=annee[i+1];
            }
            nb[0]=nb[0]-1;
            return true;
        }else{
            return false;
        }
    }
    public static void afficher(String[] nom, int[] annee, int[] nb){
        for (int i=0; i<nb[0]; i=i+1){
            System.out.println(nom[i]+" "+annee[i]);
        }
    }
    public static void main(String[] args){
        String[] lesnoms = new String[30];
        int[] lesannees = new int[30];
        int[] nbpers = {0};
        ajouter(lesnoms,lesannees,nbpers,"riri",1960);
        ajouter(lesnoms,lesannees,nbpers,"fifi",1960);
        ajouter(lesnoms,lesannees,nbpers,"loulou",1960);
        ajouter(lesnoms,lesannees,nbpers,"donald",1939);
        afficher(lesnoms,lesannees,nbpers);
        System.out.println("-----------------------------");
        supprimer(lesnoms,lesannees,nbpers,"fifi",1960);
        afficher(lesnoms,lesannees,nbpers);
```

```
    }
}
```

On peut utiliser le même principe pour gérer plus de deux tableaux ainsi que des tableaux avec des dimensions différentes. On peut par exemple coordonner un tableau à deux dimensions pour les noms et prénoms avec un tableau à une dimension pour les années de naissance.

| | | nom | prénom | | année |
|---|---|---|---|---|---|
| personne 1 | 0 | nom 1 | prénom 1 | 0 | année 1 |
| personne 2 | 1 | nom 2 | prénom 2 | 1 | année 2 |
| ... | 2 | ... | ... | 2 | ... |