Tableaux

Maria-Virginia Aponte, François Barthélémy

NFA031

1 Motivation : pourquoi avons-nous besoin de tableaux ?

Jusqu'à présent, nos programmes de calcul d'âge traitaient une personne à la fois, ou utilisaient des boucles pour traiter plusieurs personnes séquentiellement. Mais que se passerait-il si nous voulions conserver les âges de toutes ces personnes pour faire des comparaisons ou des calculs statistiques ?

1.1 Limites des variables individuelles

Supposons que nous voulions calculer l'âge de 5 personnes et ensuite déterminer qui est le plus jeune et le plus âgé. Avec nos connaissances actuelles, nous devrions écrire :

Listing 1 – Approche fastidieuse avec variables séparées

```
package nfa031;

public class CinqAgesIndividuels {
   public static void main(String[] args) {
      int anneeNaissance1, anneeNaissance2, anneeNaissance3,
         anneeNaissance4, anneeNaissance5;
      int anneeActuelle;
      int age1, age2, age3, age4, age5;

      System.out.println("Année actuelle :");
      anneeActuelle = Terminal.lireInt();

      // Saisie de 5 personnes
      System.out.println("Année de naissance personne 1 :");
      anneeNaissance1 = Terminal.lireInt();
      age1 = anneeActuelle - anneeNaissance1;

      System.out.println("Année de naissance personne 2 :");
      anneeNaissance2 = Terminal.lireInt();
      age2 = anneeActuelle - anneeNaissance2;

      // ... et ainsi de suite pour les 3 autres

      // Pour trouver le plus jeune : comparaisons multiples
      int plusJeune = age1;
      if (age2 < plusJeune) plusJeune = age2;
      if (age3 < plusJeune) plusJeune = age3;
      if (age4 < plusJeune) plusJeune = age4;
      if (age5 < plusJeune) plusJeune = age5;
```

```
        System.out.println("La personne la plus jeune a " + plusJeune
                        + " ans.");
    }
}
```

1.2 Problèmes de cette approche

Cette méthode devient rapidement ingérable :

— **Code répétitif** : Nous devons dupliquer le même traitement pour chaque personne
— **Nombres fixes** : Difficile de traiter 10, 50 ou 100 personnes
— **Maintenance complexe** : Modifier le traitement nécessite de changer de nombreux endroits
— **Pas de généralisation** : Impossible de traiter un nombre variable de personnes

Et si nous voulions traiter 100 personnes ? Nous aurions besoin de 100 variables différentes ! C'est là qu'interviennent les **tableaux**.

1.3 La solution : les tableaux

Un tableau nous permet de stocker plusieurs valeurs du même type dans une structure unique. Au lieu d'avoir :

```
int age1, age2, age3, age4, age5;
```

Nous aurons :

```
int[] ages; // Un tableau pouvant contenir plusieurs âges
```

Chaque valeur est stockée dans une **case** du tableau, accessible par son **indice** (position) :

Indice :	0	1	2	3	4
Âge :	25	32	18	45	29

Si une variable (par exemple `ages`) contient un tableau, on peut accéder au contenu d'une case du tableau en précisant l'indice de cette case entre crochets. Par exemple, `System.out.println(ages[2]);` permet d'afficher l'entier contenu dans la case d'indice 2, c'est-à-dire 18.

2 Déclaration et création d'un tableau

En Java, utiliser un tableau nécessite trois étapes :

1. **Déclaration** : Annoncer qu'une variable contiendra un tableau
2. **Création** : Allouer l'espace mémoire pour le tableau
3. **Utilisation** : Accéder aux cases pour lire ou modifier les valeurs

2.1 Déclaration

Listing 2 – Déclaration d'une variable tableau

```
int[] ages; // ages peut contenir un tableau d'entiers
```

Cette ligne déclare une variable `ages` capable de référencer un tableau d'entiers. Le tableau n'existe pas encore en mémoire.

2.2 Création

Listing 3 – Création du tableau en mémoire

```
ages = new int[5]; // Crée un tableau de 5 cases d'entiers
```

L'opérateur `new` crée effectivement le tableau en mémoire avec 5 cases, toutes initialisées à 0. Les indices de tableaux commencent toujours à 0. Les cases du tableau ont les indices numérotés 0 à 4.

2.3 Déclaration et création combinées

Listing 4 – Déclaration et création en une instruction

```
int[] ages = new int[5]; // Plus concis
```

2.4 Initialisation avec valeurs prédéfinies

Pour de petits tableaux, on peut initialiser directement avec des valeurs :

Listing 5 – Initialisation directe d'un tableau

```
int[] agesFamille = {45, 42, 18, 16, 12}; // 5 membres
```

Cette syntaxe `{45, 42, 18, 16, 12}` crée automatiquement un tableau de la bonne taille et l'initialise avec les valeurs données.

2.5 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 `String` sont initialisées à la valeur `null`

2.6 Longueur d'un tableau

La **taille** ou **longueur** d'un tableau est le nombre de cases qu'il contient. Si `ages` désigne un tableau, on peut obtenir sa longueur par la notation `ages.length`. Les indices du tableau sont alors compris entre 0 et `ages.length-1`.

3 Premier tableau d'âges

Créons maintenant notre premier programme utilisant un tableau pour stocker les âges de plusieurs personnes.

Listing 6 – Calcul d'âges avec tableau

```
package nfa031;

public class CalculAgesTableau {
   public static void main(String[] args) {
      int[] ages = new int[5]; // Tableau pour 5 âges
      int anneeActuelle;
      int anneeNaissance;

      System.out.println("Année actuelle :");
```

```
        anneeActuelle = Terminal.lireInt();

        // Saisie et calcul des âges
        for (int i = 0; i < 5; i++) {
            System.out.println("Année de naissance personne " + (i + 1)
                               + " :");
            anneeNaissance = Terminal.lireInt();
            // Stockage dans le tableau
            ages[i] = anneeActuelle - anneeNaissance;
        }

        // Affichage des résultats
        System.out.println("\n=== RÉSULTATS ===");
        for (int i = 0; i < 5; i++) {
            System.out.println("Personne " + (i + 1) + " : " + ages[i]
                               + " ans");
        }
    }
}
```

3.1 Analyse du programme

3.1.1 Stockage dans le tableau

La ligne `ages[i] = anneeActuelle - anneeNaissance;` stocke l'âge calculé dans la case `i` du tableau. Pendant l'exécution :

- Tour 1 : `i = 0`, stockage dans `ages[0]`
- Tour 2 : `i = 1`, stockage dans `ages[1]`
- Tour 3 : `i = 2`, stockage dans `ages[2]`
- etc.

3.1.2 Accès aux cases

La notation `ages[i]` permet d'accéder à la case d'indice `i`. Cette case se comporte exactement comme une variable normale :

- `ages[i] = valeur` : écriture dans la case
- `variable = ages[i]` : lecture de la case
- `ages[i] = ages[i] + 1` : modification de la case

3.1.3 Indices et longueur

Important : Les indices commencent à 0 en Java. Pour un tableau de 5 cases :

- Premier indice : 0
- Dernier indice : 4
- Indices valides : 0, 1, 2, 3, 4
- Longueur du tableau : 5 (accessible avec `ages.length`)

3.2 Version améliorée avec nombre variable

Listing 7 – Tableau de taille variable

```
package nfa031;

public class CalculAgesVariable {
```

```
    public static void main(String[] args) {
        int nombrePersonnes;
        int anneeActuelle;
        int anneeNaissance;

        System.out.println("Combien de personnes ?");
        nombrePersonnes = Terminal.lireInt();

        int[] ages = new int[nombrePersonnes]; // Taille variable !

        System.out.println("Année actuelle :");
        anneeActuelle = Terminal.lireInt();

        // Saisie et calcul
        // Utilisation de ages.length dans l'entête de boucle
        for (int i = 0; i < ages.length; i++) {
            System.out.println("Année de naissance personne " + (i + 1)
                               + " :");
            anneeNaissance = Terminal.lireInt();
            ages[i] = anneeActuelle - anneeNaissance;
        }

        // Affichage avec classification
        System.out.println("\n=== RÉSULTATS ===");
        for (int i = 0; i < ages.length; i++) {
            System.out.print("Personne " + (i + 1) + " : " + ages[i]
                             + " ans");

            if (ages[i] >= 18) {
                System.out.println(" (majeur)");
            } else {
                System.out.println(" (mineur)");
            }
        }
    }
}
```

À noter que la taille du tableau varie d'une exécution à l'autre du programme, mais pour une exécution donnée, la taille du tableau est fixe et ne varie pas. C'est une caractéristique importante des tableaux : leur taille est fixée à la création et ne varie jamais par la suite.

3.3 Utilisation de ages.length

La propriété `ages.length` donne la taille du tableau. Cette approche présente plusieurs avantages :

- **Flexible** : Fonctionne quelle que soit la taille du tableau
- **Sûre** : Évite les erreurs d'indices hors limites
- **Lisible** : Le code exprime clairement l'intention

3.4 Exemple avec valeurs prédéfinies

Listing 8 – Initialisation directe d'un tableau

```
package nfa031;
```

```java
public class AgesPredefinis {
   public static void main(String[] args) {
      // Initialisation directe avec les âges d'une famille
      int[] agesFamille = {45, 42, 18, 16, 12}; // 5 membres

      System.out.println("=== ÂGES DE LA FAMILLE ===");
      for (int i = 0; i < agesFamille.length; i++) {
         System.out.print("Membre " + (i + 1) + " : " + agesFamille[i]
                          + " ans");

         if (agesFamille[i] >= 18) {
            System.out.println(" (adulte)");
         } else {
            System.out.println(" (enfant)");
         }
      }

      // Calcul de l'âge moyen
      int somme = 0;
      for (int i = 0; i < agesFamille.length; i++) {
         somme += agesFamille[i];
      }

      double moyenneAge = (double) somme / agesFamille.length;
      System.out.println("\nÂge moyen de la famille : " + moyenneAge
                         + " ans");
   }
}
```

3.5 Erreurs courantes à éviter

3.5.1 Indice hors limites

Listing 9 – Erreur d'indice

```java
int[] ages = new int[5]; // Indices valides : 0, 1, 2, 3, 4
ages[5] = 30; // ERREUR ! L'indice 5 n'existe pas
```

Cette erreur provoque une `ArrayIndexOutOfBoundsException` à l'exécution.

3.5.2 Confusion sur la longueur

Listing 10 – Boucle incorrecte et correcte

```java
// INCORRECT
for (int i = 0; i <= ages.length; i++) { // <= au lieu de <
   System.out.println(ages[i]); // Erreur au dernier tour !
}

// CORRECT
for (int i = 0; i < ages.length; i++) {
   System.out.println(ages[i]);
}
```

4 Parcours et traitement : calculer des statistiques

Maintenant que nous savons stocker les âges de plusieurs personnes dans un tableau, exploitons cette capacité pour effectuer des calculs que nous ne pouvions pas faire facilement auparavant : statistiques, comparaisons, analyses de groupe.

4.1 Calculs statistiques de base

Listing 11 – Statistiques complètes sur un groupe

```
package nfa031;

public class StatistiquesAges {
   public static void main(String[] args) {
      int nombrePersonnes;
      int anneeActuelle;
      int anneeNaissance;

      System.out.println("Combien de personnes dans le groupe ?");
      nombrePersonnes = Terminal.lireInt();

      int[] ages = new int[nombrePersonnes];

      System.out.println("Année actuelle :");
      anneeActuelle = Terminal.lireInt();

      // Saisie des âges
      for (int i = 0; i < ages.length; i++) {
         System.out.println("Année de naissance personne " + (i + 1)
                        + " :");
         anneeNaissance = Terminal.lireInt();
         ages[i] = anneeActuelle - anneeNaissance;
      }

      // Calcul de la somme
      int sommeAges = 0;
      for (int i = 0; i < ages.length; i++) {
         sommeAges += ages[i];
      }

      // Recherche du minimum et maximum
      int ageMin = ages[0]; // Initialisation avec le premier élément
      int ageMax = ages[0];

      for (int i = 1; i < ages.length; i++) { // Commencer à1
         if (ages[i] < ageMin) {
            ageMin = ages[i];
         }
         if (ages[i] > ageMax) {
            ageMax = ages[i];
         }
      }

      // Calcul de la moyenne
```

```
        double moyenneAge = (double) sommeAges / ages.length;

        // Affichage des résultats
        System.out.println("\n=== STATISTIQUES DU GROUPE ===");
        System.out.println("Nombre de personnes : " + ages.length);
        System.out.println("Âge moyen : " + moyenneAge + " ans");
        System.out.println("Plus jeune : " + ageMin + " ans");
        System.out.println("Plus âgé : " + ageMax + " ans");
        System.out.println("Écart d'âge : " + (ageMax - ageMin)
                    + " ans");
    }
}
```

4.2 Analyse du calcul de statistiques

4.2.1 Calcul de la somme par accumulation

Listing 12 – Pattern d'accumulation

```
int sommeAges = 0; // Initialisation à0
for (int i = 0; i < ages.length; i++) {
    sommeAges += ages[i]; // Ajout de chaque âge
}
```

Ce pattern est fondamental : on parcourt tout le tableau en accumulant les valeurs dans une variable.

4.2.2 Recherche du minimum et maximum

Listing 13 – Pattern de recherche d'extremums

```
int ageMin = ages[0]; // Hypothèse : le premier est le minimum
for (int i = 1; i < ages.length; i++) { // Vérifier les autres
    if (ages[i] < ageMin) {
        ageMin = ages[i]; // Nouveau minimum trouvé
    }
}
```

Points clés :

- Initialisation avec le premier élément (pas avec 0 !)
- Boucle commence à l'indice 1
- Mise à jour seulement si une meilleure valeur est trouvée

4.3 Comptage et classification

Listing 14 – Classification par tranches d'âge

```
package nfa031;

public class ClassificationAges {
    public static void main(String[] args) {
        int[] agesFamille = {8, 15, 23, 45, 52, 67, 12, 34};

        // Compteurs pour différentes catégories
        int enfants = 0; // < 13 ans
```

```
      int adolescents = 0; // 13-17 ans
      int adultes = 0; // 18-64 ans
      int seniors = 0; // >= 65 ans

      // Classification
      for (int i = 0; i < agesFamille.length; i++) {
         if (agesFamille[i] < 13) {
            enfants++;
         } else if (agesFamille[i] < 18) {
            adolescents++;
         } else if (agesFamille[i] < 65) {
            adultes++;
         } else {
            seniors++;
         }
      }

      // Affichage détaillé
      System.out.println("=== COMPOSITION DU GROUPE ===");
      System.out.println("Enfants (< 13 ans) : " + enfants);
      System.out.println("Adolescents (13-17 ans) : " + adolescents);
      System.out.println("Adultes (18-64 ans) : " + adultes);
      System.out.println("Seniors (65 ans et plus) : " + seniors);

      // Calculs de pourcentages
      double totalPersonnes = agesFamille.length;
      System.out.println("\n=== RÉPARTITION (%) ===");
      System.out.println("Enfants : " +
                  (enfants * 100.0 / totalPersonnes) + "%");
      System.out.println("Adolescents : " +
                  (adolescents * 100.0 / totalPersonnes) + "%");
      System.out.println("Adultes : " +
                  (adultes * 100.0 / totalPersonnes) + "%");
      System.out.println("Seniors : " +
                  (seniors * 100.0 / totalPersonnes) + "%");
   }
}
```

5 Recherche et filtrage dans les tableaux

Apprenons maintenant à rechercher des informations spécifiques dans nos tableaux d'âges.

5.1 Recherche d'un âge spécifique

Listing 15 – Rechercher si un âge existe dans le groupe

```
package nfa031;

public class RechercheAge {
   public static void main(String[] args) {
      int[] agesGroupe = {25, 32, 18, 45, 29, 18, 38, 25};
      int ageRecherche;
      boolean trouve;
```

```
        int position;

        System.out.println("=== RECHERCHE D'ÂGE ===");
        System.out.println("Quel âge recherchez-vous ?");
        ageRecherche = Terminal.lireInt();

        // Recherche simple : présence/absence
        trouve = false;
        for (int i = 0; i < agesGroupe.length; i++) {
           if (agesGroupe[i] == ageRecherche) {
              trouve = true;
           }
        }

        if (trouve) {
           System.out.println("L'âge " + ageRecherche +
                         " ans est présent dans le groupe.");
        } else {
           System.out.println("L'âge " + ageRecherche +
                         " ans n'est pas présent dans le groupe.");
        }

        // Recherche avec position de la première occurrence
        position = -1; // -1 indique "non trouvé"
        for (int i = 0; i < agesGroupe.length && position == -1; i++) {
           if (agesGroupe[i] == ageRecherche) {
              position = i;
           }
        }

        if (position != -1) {
           System.out.println("Première occurrence trouvée : personne "
                         + (position + 1));
        }

        // Recherche de toutes les occurrences
        System.out.println("\n=== TOUTES LES OCCURRENCES ===");
        int compteur = 0;
        for (int i = 0; i < agesGroupe.length; i++) {
           if (agesGroupe[i] == ageRecherche) {
              compteur++;
              System.out.println("Personne " + (i + 1) + " a "
                            + ageRecherche + " ans");
           }
        }

        if (compteur > 1) {
           System.out.println("Total : " + compteur + " personnes ont "
                         + ageRecherche + " ans");
        }
     }
}
```

5.2 Filtrage par critères

Pour la conditionnelle, je propose de mettre le cas else en premier, ce sera plus facile à lire.

Listing 16 – Filtrer selon différents critères d'âge

```
package nfa031;
public class FiltrageAges {
   public static void main(String[] args) {
      int[] ages = {15, 22, 17, 35, 19, 16, 45, 20, 14, 38};
      int ageMin, ageMax;

      System.out.println("=== FILTRAGE PAR TRANCHE D'ÂGE ===");

      // Saisie et validation
      System.out.println("Âge minimum de la tranche :");
      ageMin = Terminal.lireInt();
      System.out.println("Âge maximum de la tranche :");
      ageMax = Terminal.lireInt();

      if (ageMin > ageMax) {
         System.out.println("Erreur : l'âge minimum ne peut pas être"
                        + "supérieur au maximum.");
         return;
      }

      // Recherche dans la tranche
      System.out.println("\nPersonnes dans la tranche " + ageMin + "-"
                     + ageMax + " ans :");
      int compteur = 0;
      int sommeTranche = 0;

      for (int i = 0; i < ages.length; i++) {
         if (ages[i] >= ageMin && ages[i] <= ageMax) {
            compteur++;
            sommeTranche += ages[i];
            System.out.println("Personne " + (i + 1) + " : "
                           + ages[i] + " ans");
         }
      }

      // Statistiques de la tranche
      if (compteur > 0) {
         double moyenneTranche = (double) sommeTranche / compteur;
         System.out.println("\n=== STATISTIQUES DE LA TRANCHE ===");
         System.out.println("Nombre de personnes : " + compteur);
         System.out.println("Âge moyen : " + moyenneTranche + " ans");
         System.out.println("Pourcentage du groupe : " +
                        (compteur * 100.0 / ages.length) + "%");
      } else {
         System.out.println("Personne dans cette tranche d'âge.");
      }
   }
}
```

5.3 Recherche des extremums avec informations détaillées

Listing 17 – Trouver les personnes les plus jeunes et les plus âgées

```
package nfa031;

public class ExtremumsDetailles {
   public static void main(String[] args) {
      int[] ages = {25, 32, 15, 45, 29, 15, 45, 38};
      int ageMin, ageMax;

      // Recherche des extremums
      ageMin = ages[0];
      ageMax = ages[0];

      for (int i = 1; i < ages.length; i++) {
         if (ages[i] < ageMin) {
            ageMin = ages[i];
         }
         if (ages[i] > ageMax) {
            ageMax = ages[i];
         }
      }

      // Recherche de toutes les personnes ayant l'âge minimum
      System.out.println("=== PERSONNE(S) LA/LES PLUS JEUNE(S) ===");
      System.out.println("Âge minimum : " + ageMin + " ans");
      for (int i = 0; i < ages.length; i++) {
         if (ages[i] == ageMin) {
            System.out.println("Personne " + (i + 1) + " : " + ages[i]
                  + " ans");
         }
      }

      // Recherche de toutes les personnes ayant l'âge maximum
      System.out.println("\n=== PERSONNE(S) LA/LES PLUS ÂGÉE(S) ===");
      System.out.println("Âge maximum : " + ageMax + " ans");
      for (int i = 0; i < ages.length; i++) {
         if (ages[i] == ageMax) {
            System.out.println("Personne " + (i + 1) + " : " + ages[i]
                  + " ans");
         }
      }

      // Analyse de la répartition
      System.out.println("\n=== ANALYSE ===");
      System.out.println("Écart d'âge dans le groupe : " +
               (ageMax - ageMin) + " ans");

      if (ageMax - ageMin > 30) {
         System.out.println("Groupe très hétérogène (écart>30 ans)");
      } else if (ageMax - ageMin > 15) {
         System.out.println("Groupe moyennement hétérogène " +
                  "(écart > 15 ans)");
```

```
      } else {
         System.out.println("Groupe homogène (écart <= 15 ans)");
      }
   }
}
```

5.4 Recherche avec boucles while optimisées

Pour certaines recherches, nous pouvons utiliser des boucles `while` qui s'arrêtent dès que le résultat est trouvé :

Listing 18 – Recherche optimisée d'un élément

```
package nfa031;

public class RechercheOptimisee {
   public static void main(String[] args) {
      int[] ages = {25, 32, 18, 45, 29};
      int ageRecherche = 18;

      // Recherche d'existence avec while optimisé
      boolean trouve = false;
      int i = 0;
      while (i < ages.length && !trouve) {
         if (ages[i] == ageRecherche) {
            trouve = true;
         } else {
            i++;
         }
      }

      if (trouve) {
         System.out.println("Âge " + ageRecherche +
                        " trouvé àla position " + (i + 1));
      } else {
         System.out.println("Âge " + ageRecherche +
                        " non trouvé dans le groupe");
      }
   }
}
```

5.5 Patterns de recherche essentiels

5.5.1 Recherche d'existence

Listing 19 – Pattern pour vérifier si un élément existe

```
boolean trouve = false;
for (int i = 0; i < tableau.length; i++) {
   if (tableau[i] == valeurRecherchee) {
      trouve = true;
   }
}
```

5.5.2 Recherche de position

Listing 20 – Pattern pour trouver la position d'un élément

```
int position = -1; // -1 = "non trouvé"
for (int i = 0; i < tableau.length && position == -1; i++) {
   if (tableau[i] == valeurRecherchee) {
      position = i;
   }
}
```

5.5.3 Comptage d'occurrences

Listing 21 – Pattern pour compter les occurrences

```
int compteur = 0;
for (int i = 0; i < tableau.length; i++) {
   if (tableau[i] == valeurRecherchee) {
      compteur++;
      // Pas d'arrêt : on veut toutes les occurrences
   }
}
```

6 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 asymétrie 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.

6.1 Recherche d'un élément avec boucle for

Listing 22 – Recherche d'un élément avec un for

```
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 < tab.length; numcase++) {
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  }
```

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.

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 la case qu'on regarde à une position donnée du tableau ne corresponde pas à `elem` ne permet pas du tout de dire que `elem` ne se trouve pas par exemple quelques cases plus loin dans le tableau.

6.2 Recherche d'un élément avec boucle 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.

Listing 23 – Recherche d'un élément avec un while

```
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 < tab.length && !appartient) {
         if (tab[numcase] == elem) {
            appartient = true;
         }
         numcase++;
      }
      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.

7 Exemples d'applications pratiques

7.1 Inversion d'un tableau de caractères

Listing 24 – Inversion d'un tableau

```
package nfa031;

public class Inversion {
   public static void main(String[] args) {
      int n;
      char[] t;
      System.out.println("Combien de caractères àsaisir ?");
      n = Terminal.lireInt();
      t = new char[n];

      // Initialisation
      for (int i = 0; i < t.length; i++) {
         System.out.print("Un caractère: ");
         t[i] = Terminal.lireChar();
      }

      // Affichage avant inversion
      System.out.print("Le tableau saisi: ");
      for (int i = 0; i < t.length; i++) {
         System.out.print(t[i]);
      }
      System.out.println();

      // Inversion: arrêt 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
      System.out.print("Le tableau inversé: ");
      for (int k = 0; k < t.length; k++) {
         System.out.print(t[k]);
      }
      System.out.println();
   }
}
```

La boucle d'inversion utilise deux variables d'itération i et j, initialisées avec le premier et le dernier élément du tableau. À chaque tour de boucle, les éléments aux positions 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 impaire, alors l'arrêt se produit lorsque i=j ; s'il est de taille paire, alors l'arrêt se fait lorsque j < i. En conclusion, la boucle doit terminer si i >= j.

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.

7.2 Gestion complète de notes

Listing 25 – Gestion de notes avec statistiques

```
package nfa031;

public class GestionNotes {
   public static void main(String[] args) {
      int nombreNotes;
      System.out.println("Nombre de notes àsaisir ?");
      nombreNotes = Terminal.lireInt();
      double[] lesNotes = new double[nombreNotes];

      // Initialisation
      for (int i = 0; i < lesNotes.length; i++) {
         System.out.print("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; i++) {
         if (lesNotes[i] < min) {
            min = lesNotes[i];
         }
         if (lesNotes[i] > max) {
            max = lesNotes[i];
         }
         if (lesNotes[i] >= 10) {
            sup10++;
         }
         somme = somme + lesNotes[i];
      }

      System.out.println("La moyenne des notes est: " +
                    (somme / nombreNotes));
      System.out.println("Le nombre de notes >= 10 est: " + sup10);
      System.out.println("La note minimum est: " + min);
      System.out.println("La note maximum est: " + max);
   }
}
```

8 Conclusion : maîtriser les collections de données

Ce chapitre nous a fait découvrir les tableaux, nos premiers outils pour manipuler des collections de données homogènes. Nous avons transformé notre approche de la programmation : de programmes traitant des valeurs individuelles, nous sommes passés à des programmes capables de gérer des groupes entiers.

8.1 Patterns fondamentaux maîtrisés

Les tableaux nous ont permis d'apprendre des patterns de programmation essentiels :
Accumulation : Calculer des sommes, moyennes, totaux

```
int somme = 0;
for (int i = 0; i < tableau.length; i++) {
   somme += tableau[i];
}
```

Recherche d'extremums : Trouver minimum, maximum

```
int min = tableau[0];
for (int i = 1; i < tableau.length; i++) {
   if (tableau[i] < min) min = tableau[i];
}
```

Recherche et filtrage : Localiser des éléments spécifiques

```
boolean trouve = false;
for (int i = 0; i < tableau.length; i++) {
   if (tableau[i] == valeurRecherchee) {
      trouve = true;
   }
}
```

Comptage conditionnel : Dénombrer selon des critères

```
int compteur = 0;
for (int i = 0; i < tableau.length; i++) {
   if (condition) {
      compteur++;
   }
}
```

8.2 Synergie entre tableaux et structures de contrôle

Les tableaux révèlent toute leur puissance quand ils sont combinés avec les structures de contrôle vues précédemment :

— **Boucles for** : Parcours systématique des éléments
— **Conditions if** : Traitement sélectif des données
— **Boucles while** : Recherches avec arrêt conditionnel optimisé

8.3 Du calcul d'âge aux analyses de groupe

Notre exemple du calcul d'âge s'est enrichi considérablement :

— **Version initiale** : Un âge pour une personne
— **Avec boucles** : Traitement séquentiel de plusieurs personnes
— **Avec tableaux** : Conservation et analyse de groupes entiers
— **Applications** : Statistiques, classifications, recherches multicritères

8.4 Vers des structures de données plus sophistiquées

Les tableaux constituent notre première approche des structures de données. Dans les chapitres suivants, nous découvrirons :

— **Tableaux multidimensionnels** : Pour des données plus complexes
— **Collections dynamiques** : Structures de taille variable
— **Objets** : Regroupement de données et traitements
— **Méthodes** : Organisation et réutilisation du code

8.5 Principes de programmation acquis

Au-delà de la technique, ce chapitre nous a enseigné des principes durables :

— **Réutilisabilité** : Patterns applicables à différents contextes
— **Lisibilité** : Code clair et structures de contrôle prévisibles
— **Robustesse** : Gestion des indices et validation des données

Les tableaux transforment notre façon de concevoir les programmes : au lieu de traiter des éléments isolés, nous pensons désormais en termes de collections, d'analyses de groupe et de traitement de masse. Cette évolution conceptuelle prépare les défis plus complexes de la programmation orientée objet et des algorithmes avancés.

L'exemple du calcul d'âge, parti d'un besoin simple, nous a menés vers la maîtrise d'outils puissants pour l'analyse de données. Cette approche progressive restera notre méthode : partir du concret et familier pour construire vers l'abstrait et le sophistiqué.

Ce chapitre a été rédigé avec l'assistance d'une intelligence artificielle, Claude 4 Sonnet de la société Anthropic