

Instruction conditionnelle et boucles

Maria-Virginia Aponte, François Barthélémy

NFA031

1 Introduction

Les instructions que nous avons vu jusqu'ici modifient la mémoire ou l'écran. Dans ce chapitre, nous allons voir plusieurs instructions qui ne modifient directement ni la mémoire ni l'écran mais qui déterminent quelles instructions du programme vont s'exécuter.

On les appelle des instructions de contrôle et elles ont la particularité de contenir d'autres instructions. On va avoir des instructions imbriquées dans d'autres instructions. Ces instructions de contrôle ont un rôle déterminant dans l'exécution des programmes et font que ceux-ci font plus que d'exécuter une suite d'instructions invariable.

2 La conditionnelle : faire des choix dans nos programmes

Jusqu'à présent, nos programmes exécutaient toujours les mêmes instructions dans le même ordre. Mais dans la réalité, nous avons souvent besoin de faire des choix selon les circonstances. Reprenons notre programme de calcul d'âge et enrichissons-le pour qu'il puisse adapter son comportement selon l'âge calculé.

2.1 Un besoin concret : déterminer si une personne est majeure

Améliorons notre programme `CalculAge` pour qu'il indique en plus si la personne est majeure ou mineure :

Objectif : Afficher "Vous êtes majeur(e)" si l'âge est supérieur ou égal à 18 ans, sinon afficher "Vous êtes mineur(e)".

Avec nos connaissances actuelles, nous ne pouvons pas exprimer ce choix conditionnel. Nous avons besoin d'une nouvelle instruction : **la conditionnelle**.

2.2 L'instruction if-else

La conditionnelle a la forme suivante :

Listing 1 – Syntaxe de la conditionnelle

```
if (condition) {  
    // Instructions à exécuter si la condition est vraie  
} else {  
    // Instructions à exécuter si la condition est fausse  
}
```

La **condition** doit être une expression booléenne, c'est-à-dire une expression qui vaut **true** ou **false**.

2.3 Premier exemple : majeur ou mineur

Voici notre programme enrichi :

Listing 2 – CalculAge avec test de majorité

```
package nfa031;

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

        System.out.println("En quelle année êtes-vous né(e) ?");
        anneeNaissance = Terminal.lireInt();

        System.out.println("En quelle année sommes-nous ?");
        anneeActuelle = Terminal.lireInt();

        age = anneeActuelle - anneeNaissance;

        System.out.println("Vous avez " + age + " ans.");

        // Nouvelle partie : test de majorité
        if (age >= 18) {
            System.out.println("Vous êtes majeur(e).");
        } else {
            System.out.println("Vous êtes mineur(e).");
        }
    }
}
```

2.4 Comment fonctionne la conditionnelle

Analysons l'exécution de la condition `age >= 18` :

1. Java évalue l'expression `age >= 18`
2. Si `age` contient 25, alors `25 >= 18` vaut `true`
3. Java exécute alors le bloc après `if` : "Vous êtes majeur(e)."
4. Le bloc après `else` est ignoré

Si `age` contient 16 :

1. `16 >= 18` vaut `false`
2. Java ignore le bloc après `if`
3. Java exécute le bloc après `else` : "Vous êtes mineur(e)."

2.5 Exemple plus complexe : tranches d'âge

Enrichissons encore notre programme pour distinguer plusieurs tranches d'âge :

Listing 3 – Classification par tranches d'âge

```
package nfa031;

public class CalculAgeDetaille {
```

```

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

    System.out.println("En quelle année êtes-vous né(e) ?");
    anneeNaissance = Terminal.lireInt();

    System.out.println("En quelle année sommes-nous ?");
    anneeActuelle = Terminal.lireInt();

    age = anneeActuelle - anneeNaissance;

    System.out.println("Vous avez " + age + " ans.");

    // Classification détaillée
    if (age < 13) {
        System.out.println("Vous êtes un(e) enfant.");
    } else if (age < 18) {
        System.out.println("Vous êtes un(e) adolescent(e).");
    } else if (age < 65) {
        System.out.println("Vous êtes un(e) adulte.");
    } else {
        System.out.println("Vous êtes un(e) senior.");
    }
}

```

2.6 La forme else if

Quand nous avons plus de deux cas à distinguer, nous utilisons **else if** :

Listing 4 – Structure else if

```

if (condition1) {
    // Cas 1
} else if (condition2) {
    // Cas 2
} else if (condition3) {
    // Cas 3
} else {
    // Cas par défaut
}

```

Points importants :

- Les conditions sont testées dans l'ordre
- Dès qu'une condition est vraie, son bloc s'exécute et les autres sont ignorés
- Le **else** final est optionnel et s'exécute si aucune condition n'est vraie

2.7 Conditions sans else

Parfois, nous voulons faire quelque chose seulement si une condition est vraie, sans action alternative :

Listing 5 – if sans else - Avertissement pour les mineurs

```

package nfa031;

```

```

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

        System.out.println("En quelle année êtes-vous né(e) ?");
        anneeNaissance = Terminal.lireInt();

        System.out.println("En quelle année sommes-nous ?");
        anneeActuelle = Terminal.lireInt();

        age = anneeActuelle - anneeNaissance;

        System.out.println("Vous avez " + age + " ans.");

        // Avertissement uniquement pour les mineurs
        if (age < 18) {
            System.out.println("ATTENTION : Vous êtes mineur(e).");
            System.out.println("Certaines restrictions peuvent s'appliquer.");
        }

        System.out.println("Calcul terminé.");
    }
}

```

2.8 Validation des données d'entrée

Un usage fréquent des conditionnelles est la validation des données saisies par l'utilisateur :

Listing 6 – Validation de l'année de naissance

```

package nfa031;

```

```

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

        System.out.println("En quelle année êtes-vous né(e) ?");
        anneeNaissance = Terminal.lireInt();

        System.out.println("En quelle année sommes-nous ?");
        anneeActuelle = Terminal.lireInt();

        // Validation des données
        if (anneeNaissance > anneeActuelle) {
            System.out.println("ERREUR : Vous ne pouvez pas être né(e) dans le futur !")
        } else if (anneeNaissance < 1900) {
            System.out.println("ERREUR : Année de naissance peu plausible.");
        } else {
            // Calcul normal

```

```

        age = anneeActuelle - anneeNaissance;
        System.out.println("Vous avez " + age + " ans.");

        if (age >= 18) {
            System.out.println("Vous êtes majeur(e).");
        } else {
            System.out.println("Vous êtes mineur(e).");
        }
    }
}
}
}

```

2.9 Conditions composées

Nous pouvons combiner plusieurs conditions avec les opérateurs logiques :

Listing 7 – Conditions composées avec opérateurs logiques

```
package nfa031;
```

```

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

        System.out.println("En quelle année êtes-vous né(e) ?");
        anneeNaissance = Terminal.lireInt();

        System.out.println("En quelle année sommes-nous ?");
        anneeActuelle = Terminal.lireInt();

        // Validation avec condition composée
        if (anneeNaissance >= 1900 && anneeNaissance <= anneeActuelle) {
            age = anneeActuelle - anneeNaissance;
            System.out.println("Vous avez " + age + " ans.");

            // Analyse générationnelle
            if (age >= 18 && age <= 25) {
                System.out.println("Vous appartenez à la génération Z.");
            } else if (age >= 26 && age <= 41) {
                System.out.println("Vous êtes un(e) millennial.");
            } else if (age >= 42 && age <= 57) {
                System.out.println("Vous appartenez à la génération X.");
            } else if (age >= 58) {
                System.out.println("Vous êtes un(e) baby-boomer.");
            } else {
                System.out.println("Vous êtes très jeune !");
            }
        } else {
            System.out.println("Données invalides. Vérifiez vos saisies.");
        }
    }
}

```

Rappel des opérateurs logiques :

- **&&** : ET logique (les deux conditions doivent être vraies)
- **||** : OU logique (au moins une condition doit être vraie)
- **!** : NON logique (inverse la condition)

2.10 Bonnes pratiques avec les conditionnelles

2.10.1 Utiliser des parenthèses pour la lisibilité

```
// Moins lisible
if (age >= 18 && age <= 65 || statut == 1)

// Plus lisible
if ((age >= 18 && age <= 65) || (statut == 1))
```

2.10.2 Éviter les conditions trop complexes

```
// Difficile à comprendre
if (age >= 18 && age <= 65 && salaire > 30000 && experience > 2)

// Plus clair avec variables intermédiaires
boolean ageValide = (age >= 18) && (age <= 65);
boolean profilAccepte = (salaire > 30000) && (experience > 2);

if (ageValide && profilAccepte) {
    // ...
}
```

2.10.3 Ordonner les conditions du plus probable au moins probable

```
// Si la plupart des utilisateurs sont adultes
if (age >= 18 && age < 65) {
    System.out.println("Adulte");
} else if (age < 18) {
    System.out.println("Mineur");
} else {
    System.out.println("Senior");
}
```

3 La boucle for : répéter des actions

Maintenant que nous savons faire des choix avec les conditionnelles, nous allons apprendre à répéter des actions. Imaginons que nous voulions calculer l'âge de plusieurs personnes avec notre programme. Actuellement, nous devons relancer le programme à chaque fois. Les boucles vont nous permettre de répéter des tâches automatiquement.

3.1 Un besoin concret : calculer l'âge de plusieurs personnes

Supposons que nous voulions calculer l'âge de 5 personnes différentes. Sans boucle, nous devrions copier-coller notre code 5 fois :

Listing 8 – Solution répétitive sans boucle

```
package nfa031;

public class CalculCinqAges {
    public static void main(String[] args) {
        int anneeNaissance, anneeActuelle, age;

        // Personne 1
        System.out.println("=== Personne 1 ===");
        System.out.println("Votre année de naissance :");
        anneeNaissance = Terminal.lireInt();
        System.out.println("Année actuelle :");
        anneeActuelle = Terminal.lireInt();
        age = anneeActuelle - anneeNaissance;
        System.out.println("Vous avez " + age + " ans.");

        // Personne 2
        System.out.println("=== Personne 2 ===");
        System.out.println("Votre année de naissance :");
        anneeNaissance = Terminal.lireInt();
        // ... et ainsi de suite pour les 3 autres personnes
    }
}
```

Cette approche est fastidieuse et peu élégante. La boucle **for** va nous permettre de faire mieux.

3.2 La boucle for : syntaxe et fonctionnement

La boucle **for** a la syntaxe suivante :

Listing 9 – Syntaxe de la boucle for

```
for (initialisation; condition; mise_à_jour) {
    // Instructions à répéter
}
```

Les trois parties :

- **Initialisation** : Déclaration et valeur initiale du compteur
- **Condition** : Tant qu'elle est vraie, la boucle continue
- **Mise à jour** : Comment le compteur évolue à chaque tour

3.3 Premier exemple : calcul pour plusieurs personnes

Listing 10 – Calcul d'âge pour 3 personnes avec boucle for

```
package nfa031;

public class CalculAgesMultiples {
    public static void main(String[] args) {
        int anneeNaissance, anneeActuelle, age;

        for (int i = 1; i <= 3; i++) {
            System.out.println("=== Personne " + i + " ===");

            System.out.println("Votre année de naissance :");
            anneeNaissance = Terminal.lireInt();
```

```

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

        age = anneeActuelle - anneeNaissance;
        System.out.println("Vous avez " + age + " ans.");

        if (age >= 18) {
            System.out.println("Vous êtes majeur(e).");
        } else {
            System.out.println("Vous êtes mineur(e).");
        }

        System.out.println(); // Ligne vide pour séparer
    }

    System.out.println("Calculs terminés pour toutes les personnes.");
}
}

```

3.4 Analyse du fonctionnement

Analysons `for (int i = 1; i <= 3; i++)` :

1. **Initialisation** : `int i = 1` - Création d'un compteur `i` qui vaut 1
2. **Condition** : `i <= 3` - Tant que `i` est inférieur ou égal à 3
3. **Mise à jour** : `i++` - À chaque tour, `i` augmente de 1

Déroulement :

- Tour 1 : `i = 1`, condition vraie → exécution du bloc
- Tour 2 : `i = 2`, condition vraie → exécution du bloc
- Tour 3 : `i = 3`, condition vraie → exécution du bloc
- Tour 4 : `i = 4`, condition fausse → fin de la boucle

3.5 Boucle for avec nombre variable de répétitions

Améliorons notre programme pour demander combien de personnes traiter :

Listing 11 – Nombre de personnes choisi par l'utilisateur

```

package nfa031;

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

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

        for (int i = 1; i <= nombrePersonnes; i++) {
            System.out.println("=== Personne " + i + " ===");

            System.out.println("Année de naissance :");
            anneeNaissance = Terminal.lireInt();

```

```

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

        age = anneeActuelle - anneeNaissance;
        System.out.println("Vous avez " + age + " ans.");

        System.out.println(); // Ligne vide
    }

    System.out.println("Traitement terminé pour " + nombrePersonnes + " personnes.");
}
}

```

3.6 Utiliser le compteur dans les calculs

Le compteur de boucle peut servir dans nos calculs. Créons un programme qui calcule les âges dans 10 ans :

Listing 12 – Projection d'âge sur plusieurs années

```

package nfa031;

public class ProjectionAge {
    public static void main(String[] args) {
        int anneeNaissance, anneeActuelle, age;

        System.out.println("Votre année de naissance :");
        anneeNaissance = Terminal.lireInt();

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

        age = anneeActuelle - anneeNaissance;
        System.out.println("Actuellement, vous avez " + age + " ans.");
        System.out.println();
        System.out.println("Projection sur les 10 prochaines années :");

        for (int i = 1; i <= 10; i++) {
            int ageProjet = age + i;
            int anneeFuture = anneeActuelle + i;
            System.out.println("En " + anneeFuture + ", vous aurez " + ageProjet + " ans");
        }
    }
}

```

3.7 Variations sur la boucle for

3.7.1 Compter à rebours

Listing 13 – Compte à rebours avant calcul

```

package nfa031;

public class CalculAvecCompteRebours {

```

```

public static void main(String[] args) {
    System.out.println("Démarrage du calcul d'âge dans :");

    for (int i = 5; i >= 1; i--) {
        System.out.println(i + "...");
    }

    System.out.println("C'est parti !");
    System.out.println();

    // Calcul normal
    int anneeNaissance, anneeActuelle, age;

    System.out.println("Votre année de naissance :");
    anneeNaissance = Terminal.lireInt();

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

    age = anneeActuelle - anneeNaissance;
    System.out.println("Vous avez " + age + " ans.");
}

```

3.7.2 Pas d'incrémentations différents

Listing 14 – Affichage des âges pairs uniquement

```

package nfa031;

public class AgesPairs {
    public static void main(String[] args) {
        System.out.println("Les âges pairs entre 0 et 100 ans :");

        for (int age = 0; age <= 100; age += 2) {
            System.out.print(age + " ");

            // Passage à la ligne tous les 10 affichages pour la lisibilité
            if ((age / 2 + 1) % 10 == 0) {
                System.out.println();
            }
        }
    }
}

```

3.8 Statistiques avec boucles

Créons un programme qui calcule la moyenne d'âge d'un groupe :

Listing 15 – Calcul de la moyenne d'âge

```

package nfa031;

public class MoyenneAge {

```

```

public static void main(String[] args) {
    int nombrePersonnes;
    int sommeAges = 0;
    int anneeNaissance, anneeActuelle, age;

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

    for (int i = 1; i <= nombrePersonnes; i++) {
        System.out.println("--- Personne " + i + " ---");

        System.out.println("Année de naissance :");
        anneeNaissance = Terminal.lireInt();

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

        age = anneeActuelle - anneeNaissance;
        System.out.println("Âge : " + age + " ans");

        sommeAges = sommeAges + age; // Accumulation
        System.out.println();
    }

    double moyenneAge = (double) sommeAges / nombrePersonnes;

    System.out.println("=== STATISTIQUES ===");
    System.out.println("Nombre de personnes : " + nombrePersonnes);
    System.out.println("Somme des âges : " + sommeAges + " ans");
    System.out.println("Âge moyen : " + moyenneAge + " ans");
}

```

3.9 Boucles imbriquées

Nous pouvons imbriquer des boucles pour traiter plusieurs groupes :

Listing 16 – Calcul pour plusieurs groupes de personnes

```

package nfa031;

public class PlusieursGroupes {
    public static void main(String[] args) {
        int nombreGroupes;
        int nombrePersonnes;
        int anneeNaissance, anneeActuelle, age;
        double sommeAgesGroupe;
        double moyenneGroupe;

        System.out.println("Combien de groupes à traiter ?");
        nombreGroupes = Terminal.lireInt();

        for (int groupe = 1; groupe <= nombreGroupes; groupe++) {
            System.out.println("##### GROUPE " + groupe + " #####");

```

```

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

        sommeAgesGroupe = 0.0;

        for (int personne = 1; personne <= nombrePersonnes; personne++) {
            System.out.println("-- Personne "+personne + " du groupe "+groupe+ "--");

            System.out.println("Année de naissance :");
            anneeNaissance = Terminal.lireInt();

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

            age = anneeActuelle - anneeNaissance;
            System.out.println("Âge : "+ age+ " ans");

            sommeAgesGroupe += age;
        }

        moyenneGroupe = sommeAgesGroupe / nombrePersonnes;
        System.out.println("Moyenne d'âge groupe "+ groupe + " : "+moyenneGroupe+ " a");
        System.out.println();
    }

    System.out.println("Traitement terminé pour tous les groupes.");
}
}

```

3.10 Bonnes pratiques avec les boucles for

3.10.1 Noms de variables explicites

```

// Moins clair
for (int i = 0; i < n; i++)

// Plus clair dans le contexte
for (int numeroPersonne = 1; numeroPersonne <= nombrePersonnes; numeroPersonne++)

```

3.10.2 Éviter de modifier le compteur dans la boucle

```

// À éviter - comportement imprévisible
for (int i = 1; i <= 10; i++) {
    if (condition) {
        i += 2; // Modifie le compteur !
    }
}

// Préférer des conditions claires
for (int i = 1; i <= 10; i++) {
    if (i % 3 != 0) { // Traiter seulement si i n'est pas multiple de 3
        // traitement
    }
}

```

```
}
}
```

3.10.3 Portée des variables de boucle

Une variable déclarée directement dans l'en-tête d'une boucle **for** (comme `int i = 1`) a une **portée limitée au bloc de la boucle**. Cela signifie que cette variable n'existe que pendant l'exécution de la boucle et devient inaccessible dès que la boucle se termine.

```
for (int i = 1; i <= 5; i++) {
    // i existe ici
    System.out.println(i);
}
// i n'existe plus ici - erreur de compilation
// System.out.println(i);
```

Cette limitation de portée est une bonne pratique car elle :

- Empêche l'utilisation accidentelle de la variable en dehors de la boucle
- Permet de réutiliser le même nom de variable `i` dans d'autres boucles du même bloc
- Rend le code plus clair en indiquant que la variable n'est utile que dans la boucle

Si vous avez besoin d'accéder à la valeur de la variable après la boucle, vous devez la déclarer **avant** la boucle :

```
int i; // Déclaration avant la boucle
for (i = 1; i <= 5; i++) {
    System.out.println(i);
}
// i existe toujours ici et vaut 6
System.out.println("Valeur finale : " + i);
```

La boucle **for** est idéale quand nous connaissons à l'avance le nombre de répétitions à effectuer. Dans la section suivante, nous découvrirons la boucle **while** pour les cas où le nombre de répétitions dépend d'une condition testée pendant les itérations.

4 La boucle while : répéter tant qu'une condition est vraie

La boucle **for** convient parfaitement quand nous savons combien de fois répéter une action. Mais parfois, nous voulons répéter une tâche jusqu'à ce qu'un événement se produise, sans savoir à l'avance combien de tentatives seront nécessaires. C'est là qu'intervient la boucle **while**.

4.1 Un besoin concret : valider les saisies utilisateur

Reprenons notre programme de calcul d'âge. Que se passe-t-il si l'utilisateur saisit une année de naissance invalide, comme 2050? Notre programme actuel accepte n'importe quelle valeur. Nous voulons redemander la saisie tant que l'utilisateur n'entre pas une année valide.

Problème : Nous ne savons pas combien de fois l'utilisateur va se tromper. Une boucle **for** ne convient pas car nous ne connaissons pas le nombre d'itérations nécessaires.

4.2 La boucle while : syntaxe et fonctionnement

La boucle **while** a la syntaxe suivante :

Listing 17 – Syntaxe de la boucle while

```
while (condition) {
    // Instructions à répéter tant que la condition est vraie
}
```

Fonctionnement :

1. Java évalue la condition
2. Si elle est vraie, le bloc d'instructions s'exécute
3. On retourne au point 1 (réévaluation de la condition)
4. Si elle est fausse, on sort de la boucle

4.3 Premier exemple : validation de l'année de naissance

Listing 18 – Validation de saisie avec while

```
package nfa031;

public class CalculAgeValide {
    public static void main(String[] args) {
        int anneeNaissance, anneeActuelle, age;
        boolean saisieValide;

        System.out.println("En quelle année sommes-nous ?");
        anneeActuelle = Terminal.lireInt();

        // Saisie avec validation
        saisieValide = false;
        while (!saisieValide) {
            System.out.println("Votre année de naissance :");
            anneeNaissance = Terminal.lireInt();

            if (anneeNaissance >= 1900 && anneeNaissance <= anneeActuelle) {
                saisieValide = true;
                System.out.println("Année valide acceptée.");
            } else {
                System.out.println("ERREUR : Année invalide.");
                System.out.println("L'année doit être entre 1900 et " + anneeActuelle);
                System.out.println("Veuillez recommencer.");
            }
        }

        // Une fois ici, nous sommes sûrs que anneeNaissance est valide
        age = anneeActuelle - anneeNaissance;
        System.out.println("Vous avez " + age + " ans.");

        if (age >= 18) {
            System.out.println("Vous êtes majeur(e).");
        } else {
            System.out.println("Vous êtes mineur(e).");
        }
    }
}
```

4.4 Analyse du fonctionnement

Dans l'exemple précédent :

1. **Initialisation** : `saisieValide = false` garantit qu'on entre au moins une fois dans la boucle

2. **Condition** : `!saisieValide` signifie "tant que la saisie n'est pas valide"
3. **Corps de boucle** : Demande une saisie et la valide
4. **Mise à jour** : `saisieValide = true` permet de sortir de la boucle au prochain test de sa condition
5. **Sortie** : Quand `saisieValide` devient `true`, la condition devient `false`

4.5 Version simplifiée avec condition directe

Nous pouvons simplifier en testant directement les valeurs :

Listing 19 – Validation simplifiée

```
package nfa031;

public class CalculAgeValideSimple {
    public static void main(String[] args) {
        int anneeNaissance, anneeActuelle, age;

        System.out.println("En quelle année sommes-nous ?");
        anneeActuelle = Terminal.lireInt();

        // Première saisie
        System.out.println("Votre année de naissance :");
        anneeNaissance = Terminal.lireInt();

        // Redemander tant que la saisie est invalide
        while (anneeNaissance < 1900 || anneeNaissance > anneeActuelle) {
            System.out.println("ERREUR : Année invalide.");
            System.out.println("L'année doit être entre 1900 et " + anneeActuelle);
            System.out.println("Votre année de naissance :");
            anneeNaissance = Terminal.lireInt();
        }

        System.out.println("Année valide acceptée.");

        age = anneeActuelle - anneeNaissance;
        System.out.println("Vous avez " + age + " ans.");

        if (age >= 18) {
            System.out.println("Vous êtes majeur(e).");
        } else {
            System.out.println("Vous êtes mineur(e).");
        }
    }
}
```

4.6 Exemple : menu avec choix utilisateur

La boucle `while` est parfaite pour créer des menus interactifs :

Listing 20 – Menu interactif avec `while`

```
package nfa031;

public class MenuCalculAge {
```

```

public static void main(String[] args) {
    int choix;
    int anneeNaissance, anneeActuelle, age;
    boolean continuer;

    continuer = true;

    while (continuer) {
        System.out.println("=== MENU CALCUL D'ÂGE ===");
        System.out.println("1. Calculer votre âge");
        System.out.println("2. Calculer votre âge dans 10 ans");
        System.out.println("3. Vérifier si vous êtes majeur");
        System.out.println("4. Quitter");
        System.out.println("Votre choix (1-4) :");

        choix = Terminal.lireInt();

        if (choix >= 1 && choix <= 3) {
            // Saisie commune pour les calculs
            System.out.println("Votre année de naissance :");
            anneeNaissance = Terminal.lireInt();

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

            age = anneeActuelle - anneeNaissance;
        }

        if (choix == 1) {
            System.out.println("Vous avez " + age + " ans.");
        } else if (choix == 2) {
            int ageFutur = age + 10;
            int anneeFuture = anneeActuelle + 10;
            System.out.println("En " + anneeFuture + ", vous aurez " + ageFutur + " a");
        } else if (choix == 3) {
            if (age >= 18) {
                System.out.println("Vous êtes majeur(e) (" + age + " ans).");
            } else {
                int anneesRestantes = 18 - age;
                System.out.println("Vous êtes mineur(e) (" + age + " ans).");
                System.out.println("Il vous reste " + anneesRestantes + " ans avant la");
            }
        } else if (choix == 4) {
            System.out.println("Au revoir !");
            continuer = false;
        } else {
            System.out.println("Choix invalide. Veuillez choisir entre 1 et 4.");
        }
    }

    if (continuer) {

```

```

        System.out.println("\nAppuyez sur Entrée pour continuer...");
        Terminal.lireString(); // Pause
        System.out.println();
    }
}
}
}
}

```

4.7 Recherche avec boucle while

Créons un programme qui trouve tous les multiples d'âge jusqu'à une limite :

Listing 21 – Recherche de multiples d'âge

```

package nfa031;

public class MultiplesAge {
    public static void main(String[] args) {
        int anneeNaissance, anneeActuelle, age;
        int limite;
        int multiple;
        int compteur;

        System.out.println("Votre année de naissance :");
        anneeNaissance = Terminal.lireInt();

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

        age = anneeActuelle - anneeNaissance;
        System.out.println("Vous avez " + age + " ans.");

        if (age <= 0) {
            System.out.println("Âge invalide pour ce calcul.");
            return;
        }

        System.out.println("Chercher multiples de " + age + " jusqu'à la limite ?");
        limite = Terminal.lireInt();

        System.out.println("Multiples de " + age + " jusqu'à " + limite + " :");

        multiple = age;
        compteur = 1;

        while (multiple <= limite) {
            System.out.println(age + " * " + compteur + " = " + multiple);
            compteur++;
            multiple = age * compteur;
        }

        System.out.println("Recherche terminée.");
    }
}

```

4.8 Accumulation de données avec while

Programme qui calcule la moyenne d'âge d'un nombre variable de personnes :

Explication sur
do-while arrive
plus tard !

Listing 22 – Calcul de moyenne avec condition d'arrêt

```
package nfa031;

public class MoyenneAgeVariable {
    public static void main(String[] args) {
        int anneeNaissance, anneeActuelle, age;
        int nombrePersonnes, sommeAges;
        String continuer;
        double moyenne;

        nombrePersonnes = 0;
        sommeAges = 0;

        System.out.println("=== CALCUL DE MOYENNE D'ÂGE ===");

        do {
            nombrePersonnes++;
            System.out.println("--- Personne " + nombrePersonnes + " ---");

            System.out.println("Année de naissance :");
            anneeNaissance = Terminal.lireInt();

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

            age = anneeActuelle - anneeNaissance;
            System.out.println("Âge : " + age + " ans");

            sommeAges += age;

            System.out.println("Ajouter une autre personne ? (oui/non) :");
            continuer = Terminal.lireString();

        } while (continuer.equals("oui"));

        moyenne = (double) sommeAges / nombrePersonnes;

        System.out.println("=== RÉSULTAT ===");
        System.out.println("Nombre de personnes : " + nombrePersonnes);
        System.out.println("Âge moyen : " + moyenne + " ans");
    }
}
```

4.9 Attention aux boucles infinies

Une boucle `while` peut tourner indéfiniment si la condition ne devient jamais fausse :

Listing 23 – Exemple de boucle infinie à éviter

```
// ATTENTION : Boucle infinie !
int compteur = 1;
```

```
while (compteur > 0) {
    System.out.println(compteur);
    compteur++; // compteur augmente toujours, donc toujours > 0
}
```

Pour éviter les boucles infinies :

- Vérifiez que la condition peut devenir fausse
- Assurez-vous que le corps de la boucle modifie les variables de la condition
- Testez avec des exemples simples
- Ajoutez un compteur de sécurité si nécessaire

4.10 Choix entre for et while

Utilisez for quand :

- Vous connaissez le nombre exact d'itérations
- Vous avez un compteur qui évolue de façon prévisible
- Exemples : traiter N personnes, compter de 1 à 10

Utilisez while quand :

- Le nombre d'itérations dépend d'une condition
- Vous attendez un événement ou une saisie valide
- Exemples : validation de saisie, menus interactifs, recherche

Listing 24 – Comparaison for vs while

```
// FOR - nombre connu d'itérations
for (int i = 1; i <= 5; i++) {
    // Exactement 5 répétitions
}

// WHILE - condition d'arrêt
while (!saisieValide) {
    // Nombre d'itérations inconnu
}
```

4.11 Équivalence entre for et while

Toute boucle **for** peut être transformée en **while** :

Listing 25 – Transformation for vers while

```
// Boucle for
for (int i = 1; i <= 5; i++) {
    System.out.println("Tour " + i);
}

// Équivalent en while
int i = 1; // Initialisation
while (i <= 5) { // Condition
    System.out.println("Tour " + i);
    i++; // Mise à jour
}
```

Maintenant que nous maîtrisons les conditionnelles et les boucles, nous avons tous les outils pour créer des programmes interactifs sophistiqués qui s'adaptent aux besoins des utilisateurs et traitent des volumes de données variables.

Corrigé problème avec end listing ici.

5 La boucle do-while : exécuter au moins une fois

Il existe une troisième forme de boucle en Java : la boucle **do-while**. Elle est similaire à **while**, mais avec une différence importante : elle exécute toujours le corps de la boucle au moins une fois avant de tester la condition.

5.1 Syntaxe et différence avec while

Listing 26 – Comparaison while vs do-while

```
// Boucle while classique
while (condition) {
    // Peut ne jamais s'exécuter si condition fausse dès le début
}

// Boucle do-while
do {
    // S'exécute toujours au moins une fois
} while (condition);
```

Différence clé : Avec **do-while**, la condition est testée **après** l'exécution du corps, garantissant au moins un passage.

5.2 Exemple pratique : saisie garantie

La boucle **do-while** est particulièrement utile quand nous devons faire une action au moins une fois :

Listing 27 – Saisie d'âge avec do-while

```
package nfa031;

public class SaisieAgeDoWhile {
    public static void main(String[] args) {
        int age;
        String continuer;

        System.out.println("=== VÉRIFICATION D'ÂGES ===");

        do {
            System.out.println("Votre âge :");
            age = Terminal.lireInt();

            if (age >= 18) {
                System.out.println("Vous êtes majeur(e).");
            } else if (age > 0) {
                System.out.println("Vous êtes mineur(e).");
            } else {
                System.out.println("Âge invalide.");
            }

            System.out.println("Vérifier un autre âge ? (oui/non) :");
            continuer = Terminal.lireString();
        } while (continuer.equals("oui"));
```

```

        System.out.println("Vérifications terminées.");
    }
}

```

5.3 Quand utiliser do-while

Utilisez **do-while** quand :

- L'action doit se faire au moins une fois
- La condition d'arrêt se détermine après la première exécution
- Exemples : menus, saisies interactives, validations

Comparaison avec **while** :

Listing 28 – Comparaison while et do-while

```

// Avec while - nécessite une initialisation
String reponse = "oui";
while (reponse.equals("oui")) {
    // traitement
    System.out.println("Continuer ? (oui/non)");
    reponse = Terminal.lireString();
}

// Avec do-while - plus naturel
do {
    // traitement
    System.out.println("Continuer ? (oui/non)");
    reponse = Terminal.lireString();
} while (reponse.equals("oui"));

```

La boucle **do-while** est moins fréquente que **for** et **while**, mais elle rend le code plus clair dans certaines situations spécifiques.

6 Conclusion : maîtriser les structures de contrôle

Ce chapitre nous a fait franchir une étape décisive dans l'apprentissage de la programmation. Nous sommes passés de programmes linéaires qui exécutent toujours les mêmes instructions dans le même ordre, à des programmes dynamiques capables de s'adapter et de répéter des actions.

6.1 Ce que nous avons appris

Les conditionnelles (**if-else**) nous permettent de créer des programmes qui prennent des décisions :

- Tests simples (majeur/mineur)
- Classifications complexes (tranches d'âge, générations)
- Validation de données utilisateur
- Gestion d'erreurs et de cas particuliers

La boucle **for** nous permet de répéter des actions un nombre connu de fois :

- Traitement de plusieurs personnes
- Calculs statistiques (moyennes, totaux)
- Projections temporelles
- Génération de séquences

La boucle **while** nous permet de répéter des actions jusqu'à ce qu'une condition soit remplie :

- Validation de saisies utilisateur
- Menus interactifs

- Collecte de données à volume variable
- Recherche avec critères d'arrêt
- La boucle `do-while`** garantit l'exécution d'au moins une itération :
- Saisies interactives obligatoires
- Menus qui doivent apparaître au moins une fois

6.2 L'évolution de notre programme de calcul d'âge

Notre simple programme de calcul d'âge s'est transformé au fil du chapitre :

1. **Version de base** : Calcul simple sans vérification
2. **Avec conditions** : Détection majorité/minorité, validation des données
3. **Avec boucles `for`** : Traitement de plusieurs personnes, calculs statistiques
4. **Avec boucles `while`** : Validation robuste, menus interactifs

Cette progression illustre comment les structures de contrôle transforment un programme simple en application interactive et robuste.

6.3 Principes de choix entre les structures

Pour les conditions :

- `if` simple pour deux alternatives
- `if-else if-else` pour plusieurs cas mutuellement exclusifs
- Conditions composées avec `&&`, `||`, `!` pour la logique complexe

Pour les répétitions :

- `for` quand le nombre d'itérations est connu à l'avance
- `while` quand la répétition dépend d'une condition
- `do-while` quand au moins une exécution est garantie

6.4 Vers des programmes plus sophistiqués

Avec ces outils, nous pouvons maintenant créer des programmes qui :

- S'adaptent aux données qu'ils reçoivent
- Valident et corrigent les erreurs de saisie
- Traitent des volumes de données variables
- Proposent des interfaces utilisateur interactives
- Effectuent des calculs complexes avec accumulation

Ces structures de contrôle sont les fondations de tous les algorithmes. Dans les chapitres suivants, nous découvrirons comment organiser ces instructions en méthodes réutilisables, puis comment structurer nos données avec les tableaux et les objets.

L'exemple du calcul d'âge, simple au départ, nous a menés vers des concepts fondamentaux de la programmation. Cette approche progressive sera notre méthode pour aborder les prochains défis : partir du concret et familier pour construire vers l'abstrait et le puissant.