Types, Expressions et programmes

Maria-Virginia Aponte, François Barthélémy

NFA031

Dans cette section, nous allons revenir en détail sur des notions abordées dans le premier chapitre : les expressions, les types en Java.

Avant de rentrer dans le cœur du sujet, nous allons présenter les commentaires que l'on peut insérer dans le code d'un programme.

1 Commentaires en Java

Avant d'analyser les expressions, introduisons un élément important pour la lisibilité du code : les **commentaires**. Les commentaires permettent d'ajouter des explications dans le code sans affecter son exécution.

Ce ne sont pas des instructions : ils sont présents dans le code source mais ne sont pas traduits ni exécutés par la machine Java. Ils peuvent apparaître partout dans le fichier, même à des endroits où l'on ne pourrait pas mettre d'instructions, par exemple avant la classe ou avant la méthode `main`. Ce sont des aides à la lecture pour les humains. Ils peuvent aider le programmeur à relire son code. Ils peuvent également aider une autre personne un code qu'elle n'a pas écrit, par exemple pour en assurer la maintenance.

Faire les bons commentaires est un art : il n'en faut ni trop, ni trop peu. Il faut surtout qu'ils ne paraphrasent pas le code Java mais apportent des éléments que l'on ne voit pas facilement dans ce code.

Java propose deux types de commentaires :

1.1 Commentaires de fin de ligne

Ils commencent par `//` et s'étendent jusqu'à la fin de la ligne :

Listing 1 – Commentaires de fin de ligne

```
int anneeNaissance; // Année de naissance de l'utilisateur
int age = 2025 - 1995; // Calcul simple de l'âge
// Cette ligne entière est un commentaire
```

1.2 Commentaires multi-lignes

Ils sont délimités par `/*` et `*/` et peuvent s'étendre sur plusieurs lignes :

Listing 2 – Commentaires multi-lignes

```
package nfa031;
/*
 * Ce programme calcule l'âge d'une personne
 * en fonction de son année de naissance
 */
public class CalculAge {
   /* Méthode principale */
```

```
    public static void main(String[] args) {
        int age = 30; /* âge en années */
    }
}
```

Bonnes pratiques :
— Utilisez les commentaires pour expliquer **pourquoi** vous faites quelque chose, pas **ce que** vous faites
— Exemple correct : `// Division par 12.0 pour convertir en années`
— Exemple inutile : `// Déclare une variable age`
— Gardez les commentaires concis et à jour
Nous allons utiliser les commentaires dans certains exemples de ce chapitre.

2 Analyser les expressions dans notre programme

Reprenons le programme `CalculAge` du chapitre précédent et analysons-le sous l'angle des **expressions**. Une expression est un morceau de code Java qui, lors de son exécution, calcule une valeur d'un certain type.

Listing 3 – Notre programme CalculAge

```
1  package nfa031;
2
3  public class CalculAge {
4      public static void main(String[] args) {
5          int anneeNaissance;
6          int anneeActuelle;
7          int age;
8
9          System.out.println("En quelle année êtes-vous né(e) ?");
10         anneeNaissance = Terminal.lireInt();
11         System.out.println("En quelle année sommes-nous ?");
12         anneeActuelle = Terminal.lireInt();
13         age = anneeActuelle - anneeNaissance;
14         System.out.println("Vous avez " + age + " ans.");
15     }
16 }
```

2.1 Identifier les expressions dans le programme

Si nous listons toutes les expressions du programme, voici ce que nous obtenons :
— **Ligne 9 :** `"En quelle année êtes-vous né(e) ?"` - Expression de type `String` (chaîne de caractères)
— **Ligne 10 :** `Terminal.lireInt()` - Expression de type `int`
— **Ligne 11 :** `"En quelle année sommes-nous ?"` - Expression de type `String`
— **Ligne 12 :** `Terminal.lireInt()` - Expression de type `int`
— **Ligne 13 :** `anneeActuelle - anneeNaissance` - Expression de type `int`
— **Ligne 14 :** `"Vous avez " + age + " ans."` - Expression de type `String`
Chaque expression a un **type** qui correspond au type de la valeur qu'elle calcule.

2.2 Les quatre sortes d'expressions

En analysant notre programme, nous découvrons qu'il existe quatre sortes d'expressions en Java :

1. **Les valeurs littérales**
   Dans notre programme, `"En quelle année êtes-vous né(e) ?"` est une valeur littérale de type `String`. Le calcul de cette expression est trivial : sa valeur est la chaîne elle-même.
2. **Les variables**
   Dans l'expression `anneeActuelle - anneeNaissance`, les mots `anneeActuelle` et `anneeNaissance` sont des expressions de type `int`. Leur calcul consiste à aller lire la valeur stockée dans la variable correspondante.
3. **Les opérateurs appliqués à des expressions**
   L'expression `anneeActuelle - anneeNaissance` utilise l'opérateur `-` (soustraction) appliqué à deux expressions de type `int`. Le résultat est également de type `int`.
   De même, `"Vous avez " + age + " ans."` utilise l'opérateur `+` (concaténation) pour assembler du texte. Cela va créer une nouvelle chaîne comportant à la suite la première chaîne, l'âge et la seconde chaîne. Par exemple, si la variable `age` contient la valeur 30, la chaîne calculée par l'expression sera `"vous avez 30 ans"`.
4. **Les appels de méthodes qui retournent une valeur**
   `Terminal.lireInt()` est un appel de méthode qui retourne une valeur de type `int`. Cette expression calcule sa valeur en lisant ce que l'utilisateur tape au clavier.
   Notez que certaines méthodes ne renvoient pas de valeur comme par exemple `System.out.println`. Les appels de ces méthodes ne constituent pas des expressions, ce sont des instructions.

3 Les types de données à travers des exemples concrets

Explorons maintenant les différents types de données Java en étendant notre programme avec des exemples concrets.

3.1 Type int : les nombres entiers

Dans notre programme, nous utilisons le type `int` pour les années et l'âge. Ce type représente les nombres entiers compris entre -2 147 483 648 et +2 147 483 647.

Listing 4 – Utilisation des entiers dans le calcul d'âge

```
int anneeNaissance = 1995; // Valeur littérale
int anneeActuelle = 2025; // Valeur littérale
int age = anneeActuelle - anneeNaissance; // Expression avec opérateur
```

Les opérateurs disponibles pour les `int` sont :
— `+` : addition (`2025 + 1` donne `2026`)
— `-` : soustraction (`2025 - 1995` donne `30`)
— `*` : multiplication (`30 * 12` donne `360` mois)
— `/` : division entière (`30 / 10` donne `3`)
— `%` : reste de la division (`30 % 7` donne `2`)
La division entre deux entiers donne un résultat entier. Par exemple 32/10 donne 3. Le reste de la division, ici 2, peut être calculé avec l'opérateur % au moyen de l'expression 32%10.

3.2 Type double : les nombres à virgule

Améliorons notre programme pour calculer l'âge plus précisément en tenant compte des mois :

Listing 5 – Calcul d'âge précis avec des nombres à virgule

```
int anneeNaissance = 1995;
int moisNaissance = 6; // Juin
```

```java
int anneeActuelle = 2025;
int moisActuel = 3; // Mars

double ageExact = (anneeActuelle - anneeNaissance) +
              (moisActuel - moisNaissance) / 12.0;
System.out.println("Votre âge exact : " + ageExact + " ans");
```

Le type `double` représente les nombres à virgule avec une notation utilisant le point (`12.0`, `3.14`, `-2.5`).

Important : Les calculs avec `double` sont approximatifs. Par exemple :

Listing 6 – Démonstration de l'approximation des calculs

```java
package nfa031;
public class TestPrecision {
   public static void main(String[] args) {
      double resultat = 0.7 * 0.4;
      System.out.println("0.7 * 0.4 = " + resultat);
      // Affiche : 0.27999999999999997 au lieu de 0.28
   }
}
```

3.3 Type String : les chaînes de caractères

Dans notre programme, nous utilisons plusieurs chaînes de caractères pour l'affichage. Le type `String` représente des séquences de caractères notées entre guillemets.

Listing 7 – Utilisation des chaînes dans notre programme

```java
String question = "En quelle année êtes-vous né(e) ?";
String reponse = "Vous avez " + age + " ans.";
String chaineVide = ""; // Chaîne sans aucun caractère
```

L'opérateur principal des `String` est la **concaténation** (`+`) :

```java
int age = 30;
String debut = "Vous avez ";
String fin = " ans.";
String phrase = debut + age + fin; // "Vous avez 30 ans."
```

3.4 Type boolean : les valeurs de vérité

Le type `boolean` qui encode en Java la notion de vérité (vrai ou faux) est utilisé pour des tests qui permettent de faire certaines instructions seulement si une condition est vraie. Cela sera présenté en détail dans le prochain chapitre.

Nous allons dans l'exemple suivant tester si l'année de naissance entrée par l'utilisateur semble plausible ou pas et afficher ce qu'il en est. Une année est considérée comme plausible si elle est inférieure à l'annee actuelle.

Listing 8 – Utilisation des booléens pour les vérifications

```java
package nfa031;
public class CalculAgeAvecVerification {
  public static void main(String[] args) {
    int anneeNaissance;
    int anneeActuelle;
    boolean anneePlausible;
```

```
    System.out.println("Votre année de naissance :");
    anneeNaissance = Terminal.lireInt();
    System.out.println("Année actuelle :");
    anneeActuelle = Terminal.lireInt();
    anneePlausible = anneeNaissance < anneeActuelle;
    System.out.println("L'année est-celle plausible ? " + anneePlausible);
  }
}
```

Dans cet exemple, l'expression `anneeNaissance < anneeActuelle` utilise l'opérateur d'ordre $<$ qui permet de vérifier notamment si un entier est plus petit qu'un autre. On y reviendra ci-dessous.

Le type `boolean` ne peut prendre que deux valeurs : `true` (vrai) ou `false` (faux).

Les opérateurs logiques sont :

— `&&` : ET logique (`true && false` donne `false`)

— `||` : OU logique (`true || false` donne `true`)

— `!` : NON logique (`!true` donne `false`)

Ces opérateurs prennent des opérandes de type `boolean` et calculent un résultat de type `boolean`. Il sont pour le type boolean l'équivalent des opérateurs arithmétiques pour le type `int`.

3.5 Type char : les caractères

Le type `char` représente un seul caractère, noté entre apostrophes :

Listing 9 – Exemple d'utilisation des caractères

```
char initiale = 'M';
char chiffre = '5';
char espace = ' '; // caractère espace
char arobase = '@';
```

Les chaînes de caractères sont des structures comportant plusieurs caractères, chacun d'entre eux étant un élément du type `char`.

4 Les opérateurs de comparaison

Les opérateurs de comparaison sont des opérateurs qui permettent de calculer un résultat de type `boolean`.

— Les opérateurs d'égalité/différence.

  — Opérateur d'égalité : ==

  — Opérateur de différence : !=

  Ils permettent de vérifier si deux valeurs sont égales (==) ou si elles sont différentes (!=). Ces opérateurs sont disponibles pour les types `int, double, char` et `boolean`. Ils ne fonctionnent pas pour le type `String`.

— Les opérateurs d'ordre

  — $<$ plus petit strict

  — $>$ plus grand strict

  — $<=$ plus petit ou égal

  — $>=$ plus grand ou égal

  Ces opérateurs permettent de vérifier si une condition d'ordre entre les deux opérandes est remplie. Ils sont disponibles pour les nombres (`int`, `double`) et les caractères (`char`). L'ordre utilisé pour les caractères est un ordre arbitraire, celui du codage unicode.

Revenons à notre vérification d'âge pour comprendre les opérateurs de comparaison :

Listing 10 – Exemples d'expressions de comparaison

```
int age = 22;
boolean estMajeur = age >= 18;
System.out.println(estMajeur); // affiche true
boolean a25ans = age == 25;
System.out.println(a25ans); // affiche false
boolean pas62ans = age != 62;
System.out.println(pas62ans); // affiche true
boolean estAdo = (age > 12) && (age<20);
System.out.println(estAdo); // affiche false
```

5 Priorité des opérateurs

Analysons une expression plus complexe pour notre calcul d'âge :

Listing 11 – Expression complexe nécessitant la compréhension des priorités

```
boolean peutVoter = anneeActuelle - anneeNaissance >= 18 && anneeNaissance > 1900;
```

Cette expression se calcule dans l'ordre suivant :

1. `anneeActuelle - anneeNaissance` (la soustraction d'abord)
2. `... >= 18` (puis la comparaison)
3. `anneeNaissance > 1900` (autre comparaison)
4. `... && ...` (enfin l'opérateur logique)

Ordre de priorité des opérateurs (du plus prioritaire au moins prioritaire) :

1. Opérateurs unaires : `-x`, `!condition`
2. Multiplication, division, modulo : `*`, `/`, `%`
3. Addition, soustraction : `+`, `-`
4. Comparaisons : `<`, `<=`, `>`, `>=`
5. Égalité : `==`, `!=`
6. ET logique : `&&`
7. OU logique : `||`

En cas de doute, utilisez des parenthèses : `(anneeActuelle - anneeNaissance) >= 18`

6 Méthodologie : concevoir un programme étape par étape

La méthodologie de conception d'applications est un art qui fait l'objet de cours spécifiques. Nous allons ici vous proposer une démarche adaptée aux tout petits programmes que nous allons réaliser en début de ce cours. Cette démarche comporte quatre étapes : exprimer le but de programme, identifier les données, planifier la structure, enfin écrire le code. Ensuite, il reste encore à tester le code et corriger d'éentuelles erreurs de comportement.

6.1 Étape 1 : Exprimer le but du programme

But initial : "Créer un programme qui calcule l'âge d'une personne en fonction de son année de naissance."

Reformulation précise : "Le programme demande à l'utilisateur son année de naissance et l'année actuelle, puis calcule et affiche son âge en années."

6.2 Étape 2 : Identifier les données

Données nécessaires :
— Année de naissance (entrée utilisateur), nombre entier, type `int`
— Année actuelle (entrée utilisateur), nombre entier, type `int`
— Âge calculé (résultat), nombre entier, type `int`
Schéma entrées/sorties :

ENTRÉES	SORTIE
`anneeNaissance : int` `anneeActuelle : int`	`age : int`

6.3 Étape 3 : Planifier la structure

Structure de base :

1. Saisir l'année de naissance
2. Saisir l'année actuelle
3. Calculer l'âge = année actuelle - année naissance
4. Afficher le résultat

6.4 Étape 4 : Programmer et améliorer

Version de base : Notre programme `CalculAge`
Améliorations possibles :
— Ajouter des vérifications (années valides)
— Calculer l'âge en tenant compte du mois

7 Validation des expressions

Pour vérifier qu'une expression est correcte, il faut s'assurer d'un certain nombre de propriété.

1. **Syntaxe correcte :** Chaque opérateur a le bon nombre d'opérandes
2. **Types compatibles :** Les types des opérandes correspondent à ce qu'attend l'opérateur
3. **Variables déclarées :** Toute variable utilisée a été déclarée

Exemples corrects dans notre contexte :

```
age >= 18 // Comparaison : int >= int -> boolean
"Âge : " + age // Concaténation : String + int -> String
Terminal.lireInt() // Appel sans paramètre -> int
```

Exemples incorrects :

```
age + // Manque un opérande
age == "18" // Types incompatibles : int == String
Terminal.lireInt(age) // Trop de paramètres
```

Le compilateur vérifie automatiquement que les expressions sont correctes et si ce n'est pas le cas, il produit des messages vous permettant d'identifier et résoudre les problèmes.

8 Erreurs de Java et bugs de programmes

On appelle bug (en français bogue) une erreur d'un programme qui fait qu'il calcule un résultat faux ou qu'il plante sans réussir à calculer un résultat ou encore ou encore qu'il parte dans un calcul interminable et ne parvienne pas à calculer un résultat attendu.

C'est différent par nature des erreurs de langage qui font qu'un programme n'est pas écrit en bon Java, en respectant les règles de ce langage. Ces erreurs-là sont détectées par le compilateur et un programme qui comporte ces erreurs n'étant pas compilé, il ne peut pas s'exécuter.

Le compilateur peut détecter par exemple qu'une variable est utilisée sans avoir été déclarée auparavant et cette erreur apparaît dans des messages d'erreur et dans Eclipse avec une croix rouge dans la marge de la ligne où l'erreur a été détectée.

Le compilateur ne peut pas signaler les bugs parce qu'il ne comprend pas ce que le programme est censé faire et qu'il ne sait pas non plus comment résoudre les problèmes, quels méthodes permettent d'arriver au résultat. Il ne suffit donc pas de compiler un programme pour être sûr qu'un programme fonctionne. Il faut aussi le tester de la façon la plus complète possible.

9 Récapitulatif : maîtriser les fondamentaux

Ce chapitre nous a permis de consolider les bases de la programmation Java à travers l'analyse et l'amélioration de notre programme de calcul d'âge.

9.1 Concepts maîtrisés

Expressions : Nous savons maintenant identifier et construire les quatre types d'expressions Java :

— Valeurs littérales (`1995`, `"Bonjour"`)
— Variables (`age`, `anneeNaissance`)
— Opérations (`age >= 18`, `"Vous avez " + age`)
— Appels de méthodes (`Terminal.lireInt()`)

Types de données : Nous maîtrisons les cinq types fondamentaux et savons quand les utiliser dans des contextes concrets.

Méthodologie : Nous avons appliqué une démarche structurée pour concevoir et améliorer un programme.

9.2 Prochaines étapes

Les concepts vus ici constituent les fondations pour aborder les structures de contrôle (conditions et boucles) qui permettront de créer des programmes plus sophistiqués et interactifs.

L'exemple du calcul d'âge, simple en apparence, nous a permis d'explorer la richesse des expressions Java tout en gardant un ancrage pratique. Cette approche progressive sera notre fil conducteur pour les chapitres suivants.

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