Méthodes statiques

Maria-Virginia Aponte, François Barthélémy

NFA031

Introduction : Pourquoi les méthodes ?

Reprenons notre programme de calcul d'âge du chapitre précédent. Imaginons maintenant que nous voulions calculer les âges de plusieurs personnes dans une famille et effectuer différentes vérifications sur ces âges.

0.1 Le problème : répétition de code

Voici ce que pourrait donner notre programme sans utilisation de méthodes :

Listing 1 – Programme répétitif sans méthodes

```
package nfa031;

public class CalculAgesFamille {
   public static void main(String[] args) {
      int anneeActuelle = 2025;

      // Calcul pour le père
      System.out.println("Année de naissance du père ?");
      int anneeNaissancePere = Terminal.lireInt();
      int agePere = anneeActuelle - anneeNaissancePere;
      System.out.println("Le père a " + agePere + " ans");

      if (agePere >= 18) {
         System.out.println("Le père est majeur");
      } else {
         System.out.println("Le père est mineur");
      }

      // Calcul pour la mère
      System.out.println("Année de naissance de la mère ?");
      int anneeNaissanceMere = Terminal.lireInt();
      int ageMere = anneeActuelle - anneeNaissanceMere;
      System.out.println("La mère a " + ageMere + " ans");

      if (ageMere >= 18) {
         System.out.println("La mère est majeure");
      } else {
         System.out.println("La mère est mineure");
      }

      // Calcul pour l'enfant
      System.out.println("Année de naissance de l'enfant ?");
```

```
        int anneeNaissanceEnfant = Terminal.lireInt();
        int ageEnfant = anneeActuelle - anneeNaissanceEnfant;
        System.out.println("L'enfant a " + ageEnfant + " ans");

        if (ageEnfant >= 18) {
           System.out.println("L'enfant est majeur");
        } else {
           System.out.println("L'enfant est mineur");
        }

        // Calculs familiaux
        int ageMoyen = (agePere + ageMere + ageEnfant) / 3;
        System.out.println("Âge moyen de la famille : " + ageMoyen
                     + " ans");
    }
}
```

0.2 Problèmes identifiés

Ce programme présente plusieurs défauts majeurs :

— **Code dupliqué** : Le calcul d'âge et la vérification de majorité sont répétés trois fois identiquement

— **Maintenance difficile** : Si nous voulons changer la formule de calcul (par exemple, tenir compte du mois), nous devons modifier le code à trois endroits

— **Risque d'erreurs** : Plus de répétitions signifie plus de risques d'oublier une modification ou de faire une erreur de copie

— **Lisibilité réduite** : Le code devient long et difficile à comprendre rapidement

— **Pas de réutilisabilité** : Impossible d'utiliser ces calculs dans d'autres programmes

0.3 La solution : les méthodes

Les **méthodes** permettent de regrouper des instructions qui réalisent une tâche précise sous un nom, puis de les réutiliser autant de fois que nécessaire. C'est le principe de la **factorisation** du code.

Une méthode possède :

— Un **nom** qui décrit sa fonction

— Des **paramètres** (données d'entrée)

— Un **type de retour** (donnée de sortie, ou `void` s'il n'y a pas de données de sortie)

— Un **corps** (les instructions à exécuter)

Avec les méthodes, notre programme précédent deviendrait beaucoup plus clair :

Listing 2 – Aperçu avec méthodes (détail dans la section suivante)

```
// Calcul pour chaque membre de la famille
int agePere = calculerAge(anneeNaissancePere, anneeActuelle);
int ageMere = calculerAge(anneeNaissanceMere, anneeActuelle);
int ageEnfant = calculerAge(anneeNaissanceEnfant, anneeActuelle);

// Vérifications
afficherStatutMajorite(agePere, "père");
afficherStatutMajorite(ageMere, "mère");
afficherStatutMajorite(ageEnfant, "enfant");
```

Dans ce code, `calculerAge` et `afficherStatutMajorite` sont deux méthodes utilisées dans la méthode `main`. Pour que ces méthodes puissent être utilisées de la sorte, il faut les définir. Nous allons voir comment procéder à cette définition.

1 Première méthode : calculer un âge

Créons maintenant notre première méthode pour résoudre le problème de répétition identifié précédemment.

1.1 Définition d'une méthode avec paramètres et avec résultat

Voici notre première méthode qui calcule l'âge d'une personne :

Listing 3 – Première méthode pour calculer un âge

```
package nfa031;

public class CalculAgeAvecMethode {

   public static int calculerAge(int anneeNaissance,
                          int anneeActuelle) {
      int age = anneeActuelle - anneeNaissance;
      return age;
   }

   public static void main(String[] args) {
      int anneeActuelle = 2025;

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

      int age = calculerAge(anneeNaissance, anneeActuelle);

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

1.2 Analyse de la méthode

Décomposons la ligne de définition de notre méthode :

```
public static int calculerAge(int anneeNaissance, int anneeActuelle)
```

- `public static` : Mots-clés obligatoires que nous utiliserons systématiquement (explication détaillée dans un cours ultérieur)
- `int` : Type de retour - notre méthode renvoie un entier (l'âge calculé)
- `calculerAge` : Nom de la méthode - décrit clairement ce qu'elle fait
- `(int anneeNaissance, int anneeActuelle)` : Liste des paramètres - les données dont la méthode a besoin pour fonctionner, les entrées de la méthode.

1.3 Signature d'une méthode

La **signature** d'une méthode comprend son nom, ses paramètres et son type de retour. Elle peut se noter graphiquement :

```
calculerAge : int, int → int
```

Cela signifie : "La méthode calculerAge prend deux entiers en paramètres et retourne un entier".

1.4 Utilisation de la méthode

Pour utiliser (on dit `appeler` ou `invoquer`) une méthode, on écrit son nom suivi des valeurs des paramètres entre parenthèses :

Listing 4 – Différentes façons d'appeler notre méthode

```
// Avec des variables
int age1 = calculerAge(anneeNaissance, anneeActuelle);

// Avec des valeurs littérales
int age2 = calculerAge(1995, 2025);

// Avec des expressions
int age3 = calculerAge(anneeNaissance, anneeActuelle + 1);

// Dans un affichage
System.out.println("Âge : " + calculerAge(1990, 2025));
```

1.5 L'instruction return

L'instruction `return` a deux rôles essentiels :

1. Elle **renvoie une valeur** à la méthode qui a appelé notre méthode
2. Elle **termine immédiatement** l'exécution de la méthode

Listing 5 – Utilisation de return

```
public static int calculerAge(int anneeNaissance, int anneeActuelle) {
   int age = anneeActuelle - anneeNaissance;
   return age; // Renvoie la valeur et termine la méthode

   // Ces lignes ne seraient jamais exécutées !
   // System.out.println("Cette ligne est inaccessible");
}
```

On peut aussi simplifier en retournant directement le résultat du calcul :

Listing 6 – Return direct du calcul

```
public static int calculerAge(int anneeNaissance, int anneeActuelle) {
   return anneeActuelle - anneeNaissance;
}
```

1.6 Exemple complet avec plusieurs appels

Voici un programme complet qui utilise notre méthode pour calculer l'âge de plusieurs personnes :

Listing 7 – Programme familial avec notre méthode

```
package nfa031;

public class FamilleAvecMethode {

   public static int calculerAge(int anneeNaissance,
                                 int anneeActuelle) {
      return anneeActuelle - anneeNaissance;
```

```
    }

    public static void main(String[] args) {
        int anneeActuelle = 2025;

        // Saisie des années de naissance
        System.out.println("Année de naissance du père ?");
        int anneeNaissancePere = Terminal.lireInt();

        System.out.println("Année de naissance de la mère ?");
        int anneeNaissanceMere = Terminal.lireInt();

        System.out.println("Année de naissance de l'enfant ?");
        int anneeNaissanceEnfant = Terminal.lireInt();

        // Calculs des âges avec notre méthode
        int agePere = calculerAge(anneeNaissancePere, anneeActuelle);
        int ageMere = calculerAge(anneeNaissanceMere, anneeActuelle);
        int ageEnfant = calculerAge(anneeNaissanceEnfant,
                                    anneeActuelle);

        // Affichage des résultats
        System.out.println("=== ÂGES DE LA FAMILLE ===");
        System.out.println("Père : " + agePere + " ans");
        System.out.println("Mère : " + ageMere + " ans");
        System.out.println("Enfant : " + ageEnfant + " ans");

        // Calcul de l'âge moyen
        int ageMoyen = (agePere + ageMere + ageEnfant) / 3;
        System.out.println("Âge moyen : " + ageMoyen + " ans");
    }
}
```

1.7 Enrichissement : méthodes de validation booléennes

Maintenant que nous maîtrisons les méthodes à retour numérique, créons des méthodes qui retournent des valeurs booléennes pour valider nos données :

Listing 8 – Méthodes de validation booléennes

```
public static boolean estAnneeValide(int anneeNaissance,
                                     int anneeActuelle) {
    return anneeNaissance >= 1900 &&
           anneeNaissance <= anneeActuelle &&
           (anneeActuelle - anneeNaissance) <= 150;
}

public static boolean estMajeur(int age) {
    return age >= 18;
}

public static boolean estSenior(int age) {
    return age >= 65;
}
```

```
public static boolean estEnfant(int age) {
   return age < 13;
}
```

Ces méthodes illustrent parfaitement le principe "une méthode = une responsabilité = un résultat". Chacune répond à une question précise et retourne une réponse claire.

1.8 Combinaison de méthodes

Les méthodes peuvent se combiner naturellement pour créer des logiques plus complexes :

Listing 9 – Combinaison de méthodes de validation

```
public static String determinerCategorie(int age) {
   if (estEnfant(age)) {
      return "enfant";
   } else if (!estMajeur(age)) { // entre 13 et 17 ans
      return "adolescent";
   } else if (estSenior(age)) {
      return "senior";
   } else {
      return "adulte";
   }
}
```

1.9 Exemple complet avec validation

Voici un programme complet qui utilise toutes nos méthodes pour traiter les données de manière robuste :

Listing 10 – Programme familial avec validation

```
package nfa031;

public class FamilleAvecValidation {

   public static int calculerAge(int anneeNaissance,
                          int anneeActuelle) {
      return anneeActuelle - anneeNaissance;
   }

   public static boolean estAnneeValide(int anneeNaissance,
                                int anneeActuelle) {
      return anneeNaissance >= 1900 &&
           anneeNaissance <= anneeActuelle &&
           (anneeActuelle - anneeNaissance) <= 150;
   }

   public static boolean estMajeur(int age) {
      return age >= 18;
   }

   public static String determinerCategorie(int age) {
      if (age < 13) {
         return "enfant";
```

```
    } else if (age < 18) {
       return "adolescent";
    } else if (age < 65) {
       return "adulte";
    } else {
       return "senior";
    }
}

public static void main(String[] args) {
    int anneeActuelle = 2025;

    // Saisie avec validation du père
    System.out.println("Année de naissance du père ?");
    int anneeNaissancePere = Terminal.lireInt();

    if (estAnneeValide(anneeNaissancePere, anneeActuelle)) {
       int agePere = calculerAge(anneeNaissancePere, anneeActuelle);
       System.out.println("Père : " + agePere + " ans (" +
                   determinerCategorie(agePere) + ")");

       if (estMajeur(agePere)) {
          System.out.println("Le père peut voter");
       }
    } else {
       System.out.println("Année de naissance du père invalide");
    }

    // Saisie avec validation de la mère
    System.out.println("Année de naissance de la mère ?");
    int anneeNaissanceMere = Terminal.lireInt();

    if (estAnneeValide(anneeNaissanceMere, anneeActuelle)) {
       int ageMere = calculerAge(anneeNaissanceMere, anneeActuelle);
       System.out.println("Mère : " + ageMere + " ans (" +
                   determinerCategorie(ageMere) + ")");
    } else {
       System.out.println("Année de naissance de la mère invalide");
    }

    // Saisie avec validation de l'enfant
    System.out.println("Année de naissance de l'enfant ?");
    int anneeNaissanceEnfant = Terminal.lireInt();

    if (estAnneeValide(anneeNaissanceEnfant, anneeActuelle)) {
       int ageEnfant = calculerAge(anneeNaissanceEnfant,
                              anneeActuelle);
       System.out.println("Enfant : " + ageEnfant + " ans (" +
                   determinerCategorie(ageEnfant) + ")");

       if (!estMajeur(ageEnfant)) {
          int anneesRestantes = 18 - ageEnfant;
          System.out.println("Majorité dans " + anneesRestantes
                        + " ans");
```

```
        }
    } else {
        System.out.println("Année de naissance de l'enfant invalide");
    }
  }
}
```

1.10 Avantages obtenus

Comparé au programme initial, nous avons gagné :
— **Clarté** : Le code exprime mieux l'intention (calculer un âge, valider une donnée)
— **Réutilisabilité** : Les méthodes peuvent être utilisées autant de fois que nécessaire
— **Maintenabilité** : Pour modifier un calcul ou une validation, un seul endroit à changer
— **Testabilité** : On peut facilement tester chaque méthode avec différentes valeurs
— **Lisibilité** : Le programme principal se concentre sur la logique métier
— **Robustesse** : La validation empêche les erreurs et données aberrantes

Cette première expérience avec les méthodes illustre le principe fondamental : une méthode doit avoir une responsabilité claire et bien définie.

2 Méthodes sans retour (void)

Maintenant que nous savons créer des méthodes qui calculent et retournent des valeurs, explorons les méthodes qui effectuent des actions sans renvoyer de résultat. Ces méthodes sont déclarées avec le mot-clé `void`.

2.1 Première méthode void : afficher le résultat

Reprenons notre programme familial et créons une méthode pour afficher les informations sur l'âge d'une personne :

Listing 11 – Méthode void pour afficher un âge

```
package nfa031;

public class AffichageAge {

   public static int calculerAge(int anneeNaissance,
                          int anneeActuelle) {
      return anneeActuelle - anneeNaissance;
   }

   public static void afficherAge(String personne, int age) {
      System.out.println(personne + " : " + age + " ans");

      if (age >= 18) {
         System.out.println(personne + " est majeur(e)");
      } else {
         System.out.println(personne + " est mineur(e)");
      }
   }

   public static void main(String[] args) {
      int anneeActuelle = 2025;
```

```
        // Saisie et calculs
        System.out.println("Année de naissance du père ?");
        int agePere = calculerAge(Terminal.lireInt(), anneeActuelle);

        System.out.println("Année de naissance de la mère ?");
        int ageMere = calculerAge(Terminal.lireInt(), anneeActuelle);

        System.out.println("Année de naissance de l'enfant ?");
        int ageEnfant = calculerAge(Terminal.lireInt(), anneeActuelle);

        // Affichage avec notre méthode void
        System.out.println("\n=== RÉSULTATS ===");
        afficherAge("Père", agePere);
        afficherAge("Mère", ageMere);
        afficherAge("Enfant", ageEnfant);
    }
}
```

2.2 Différences entre méthodes void et méthodes avec retour

2.2.1 Déclaration

Listing 12 – Comparaison des déclarations

```
// Méthode avec retour
public static int calculerAge(int anneeNaissance, int anneeActuelle) {
    return anneeActuelle - anneeNaissance;
}

// Méthode void (sans retour)
public static void afficherAge(String personne, int age) {
    System.out.println(personne + " : " + age + " ans");
    // Pas d'instruction return nécessaire
}
```

2.2.2 Utilisation

Listing 13 – Différences d'utilisation

```
// Méthode avec retour : on utilise le résultat
int age = calculerAge(1995, 2025);
System.out.println("Âge calculé : " + calculerAge(1995, 2025));

// Méthode void : instruction autonome
afficherAge("Pierre", 30);
// On ne peut PAS écrire : int x = afficherAge("Pierre", 30);
```

2.3 Méthodes void utilitaires

Les méthodes void sont particulièrement utiles pour des tâches répétitives d'affichage ou de formatage :

Listing 14 – Méthodes utilitaires d'affichage

```
public static void afficherSeparateur() {
```

```
    System.out.println("================================");
}

public static void afficherTitre(String titre) {
    afficherSeparateur();
    System.out.println(" " + titre);
    afficherSeparateur();
}

public static void afficherPersonne(String nom, int age, String
                            statut) {
    System.out.println(nom + " (" + age + " ans) - " + statut);
}

// Utilisation dans main
public static void main(String[] args) {
    afficherTitre("CALCUL DES ÂGES");

    // ... calculs des âges ...

    afficherTitre("RÉSULTATS");
    afficherPersonne("Pierre", 45, "majeur");
    afficherPersonne("Marie", 42, "majeure");
    afficherPersonne("Tom", 16, "mineur");
}
```

3 Concepts fondamentaux

Maintenant que nous maîtrisons la création de méthodes avec et sans retour, approfondissons les concepts essentiels qui régissent leur fonctionnement.

3.1 Variables locales et portée

Les variables déclarées dans une méthode sont **locales** à cette méthode. Elles n'existent que pendant l'exécution de la méthode et ne sont pas accessibles depuis l'extérieur.

Listing 15 – Illustration des variables locales

```
package nfa031;

public class VariablesLocales {

    public static int calculerAge(int anneeNaissance,
                            int anneeActuelle) {
        int age = anneeActuelle - anneeNaissance; // Variable locale
        int decennies = age / 10; // Variable locale

        System.out.println("Calcul en cours : " + decennies
                        + " décennies");
        return age;
        // age et decennies disparaissent ici
    }

    public static void analyserAge(int age) {
```

```
        String categorie; // Variable locale

        if (age < 13) {
            categorie = "enfant";
        } else if (age < 18) {
            categorie = "adolescent";
        } else if (age < 65) {
            categorie = "adulte";
        } else {
            categorie = "senior";
        }

        System.out.println("Catégorie : " + categorie);
        // categorie disparaît ici
    }

    public static void main(String[] args) {
        int anneeActuelle = 2025; // Variable locale au main

        int monAge = calculerAge(1990, anneeActuelle);
        analyserAge(monAge);

        // ERREUR : ces variables n'existent pas ici
        // System.out.println(decennies); // Erreur de compilation
        // System.out.println(categorie); // Erreur de compilation
    }
}
```

3.1.1 Isolement des méthodes

Chaque méthode a son propre espace de variables. Deux méthodes peuvent avoir des variables de même nom sans conflit :

Listing 16 – Variables de même nom dans différentes méthodes

```
public static int calculerAge(int anneeNaissance, int anneeActuelle) {
    int resultat = anneeActuelle - anneeNaissance; // resultat local
    return resultat;
}

public static double calculerMoyenne(int age1, int age2, int age3) {
    int resultat = (age1 + age2 + age3)/3; // resultat local différent
    return resultat;
}
```

3.2 Passage par valeur

En Java, les paramètres sont transmis **par valeur**. Cela signifie que la méthode reçoit une copie de la valeur, pas la variable elle-même.

Listing 17 – Démonstration du passage par valeur

```
package nfa031;

public class PassageParValeur {
```

```
    public static void tenterModification(int parametre) {
        System.out.println("Paramètre reçu : " + parametre);
        parametre = parametre + 10; // Modifie seulement la copie locale
        System.out.println("Paramètre modifié : " + parametre);
    }

    public static int ajouterDixAns(int age) {
        age = age + 10; // Modifie la copie locale
        return age; // Retourne la nouvelle valeur
    }

    public static void main(String[] args) {
        int monAge = 25;

        System.out.println("Avant appel : " + monAge);
        tenterModification(monAge);
        System.out.println("Après appel : " + monAge); // monAge inchangé

        // Pour modifier, il faut utiliser la valeur de retour
        monAge = ajouterDixAns(monAge);
        System.out.println("Avec retour : " + monAge); // monAge modifié
    }
}
```

Sortie du programme :

```
Avant appel : 25
Paramètre reçu : 25
Paramètre modifié : 35
Après appel : 25
Avec retour : 35
```

3.3 Signature et surcharge de méthodes

La **signature** d'une méthode comprend son nom et la liste des types de ses paramètres. Deux méthodes peuvent avoir le même nom si leurs signatures sont différentes (surcharge).

Listing 18 – Surcharge de méthodes

```
package nfa031;

public class SurchargeAge {

    // Calcul d'âge simple
    public static int calculerAge(int anneeNaissance,
                                  int anneeActuelle) {
        return anneeActuelle - anneeNaissance;
    }

    // Calcul d'âge avec mois (surcharge)
    public static double calculerAge(int anneeNaissance,
                                     int moisNaissance,
                                     int anneeActuelle, int moisActuel) {
        double ageAnnees = anneeActuelle - anneeNaissance;
        double ageMois = (moisActuel - moisNaissance) / 12.0;
```

```
        return ageAnnees + ageMois;
    }

    // Affichage simple
    public static void afficherAge(int age) {
        System.out.println("Âge : " + age + " ans");
    }

    // Affichage détaillé (surcharge)
    public static void afficherAge(String nom, int age) {
        System.out.println(nom + " a " + age + " ans");
    }

    public static void main(String[] args) {
        // Utilisation des différentes versions
        int ageSimple = calculerAge(1995, 2025);
        double agePrecis = calculerAge(1995, 6, 2025, 3);

        afficherAge(ageSimple);
        afficherAge("Pierre", ageSimple);
    }
}
```

3.4 Bonnes pratiques de conception

3.4.1 Une responsabilité par méthode

Chaque méthode doit avoir une seule responsabilité clairement définie :

Listing 19 – Séparation des responsabilités

```
// BIEN : une responsabilité par méthode
public static int calculerAge(int anneeNaissance, int anneeActuelle) {
    return anneeActuelle - anneeNaissance;
}

public static boolean estMajeur(int age) {
    return age >= 18;
}

public static void afficherStatutMajorite(String nom, int age) {
    if (estMajeur(age)) {
        System.out.println(nom + " est majeur(e)");
    } else {
        System.out.println(nom + " est mineur(e)");
    }
}

// MOINS BIEN : méthode qui fait trop de choses
public static void calculerEtAfficherTout(int anneeNaissance,
                                          String nom) {
    int age = 2025 - anneeNaissance; // calcul
    System.out.println(nom + " : " + age + " ans"); // affichage
    if (age >= 18) { // validation et affichage
        System.out.println("Majeur");
```

```
    }
    // Mélange de responsabilités
}
```

3.4.2 Noms expressifs

Le nom d'une méthode doit clairement indiquer ce qu'elle fait :

```
// BIEN : noms expressifs
public static int calculerAge(int anneeNaissance, int anneeActuelle)
public static boolean estMajeur(int age)
public static void afficherResultats(int[] ages)

// MOINS BIEN : noms vagues
public static int faire(int a, int b)
public static boolean tester(int x)
public static void montrer(int[] tab)
```

Conclusion

Ce chapitre nous a permis de maîtriser les méthodes statiques, un outil fondamental pour organiser et structurer nos programmes.

3.5 Concepts abordés

3.5.1 Types de méthodes

Nous savons maintenant créer et utiliser :
- **Méthodes avec retour** : qui calculent et renvoient une valeur (`calculerAge`, `estMajeur`)
- **Méthodes void** : qui effectuent des actions sans retour (`afficherAge`, `afficherComparaison`)
- **Méthodes de validation** : qui vérifient des conditions et retournent des booléens
- **Méthodes utilitaires** : qui encapsulent des tâches répétitives

3.5.2 Principes de conception

- **Une responsabilité par méthode** : Chaque méthode a un rôle précis et bien défini
- **Noms expressifs** : Le nom de la méthode indique clairement sa fonction
- **Paramètres appropriés** : Les données nécessaires sont passées en paramètres
- **Réutilisabilité** : Les méthodes peuvent être utilisées dans différents contextes

3.5.3 Mécanismes techniques

- **Passage par valeur** : Les paramètres sont des copies locales
- **Variables locales** : Chaque méthode a son espace de variables
- **Signature** : Identification unique d'une méthode par son nom et ses paramètres
- **Surcharge** : Possibilité d'avoir plusieurs méthodes de même nom avec des paramètres différents

3.6 Avantages acquis

Grâce aux méthodes, nos programmes sont maintenant :
- **Plus lisibles** : Le code exprime clairement les intentions

— **Plus maintenables** : Modifier un calcul ne nécessite qu'un changement à un endroit
— **Plus fiables** : Les méthodes peuvent être testées individuellement
— **Plus réutilisables** : Le même code peut servir dans plusieurs contextes
— **Mieux organisés** : Séparation claire entre calculs, validations et affichages

3.7 Bonnes pratiques retenues

— Donner des noms expressifs aux méthodes et paramètres
— Une méthode = une responsabilité
— Valider les données avant de les traiter
— Séparer les calculs de l'affichage
— Créer des méthodes réutilisables plutôt que du code spécialisé
— Tester les méthodes avec différentes valeurs

L'exemple du calcul d'âge, parti d'un besoin simple de factorisation, nous a menés vers la maîtrise d'un outil puissant pour l'organisation du code. Cette approche méthodique et modulaire sera notre base pour aborder les structures de données plus complexes comme les objets, où la combinaison de méthodes bien conçues et de données révélera toute sa puissance.