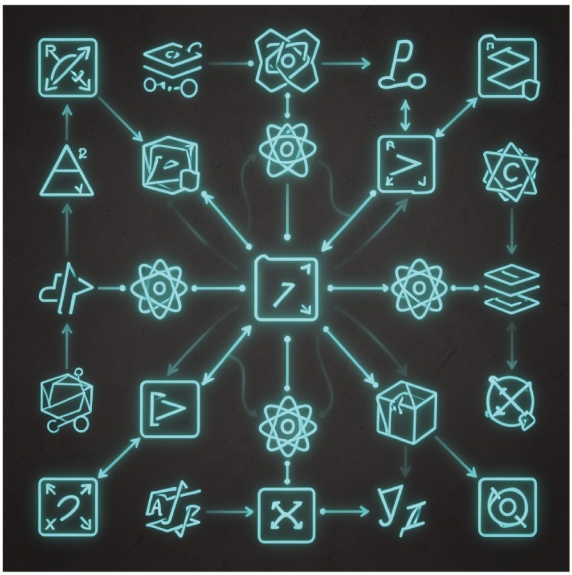
Programmation orientée objet en Java



Lucien Sina

Programmation orientée objet en Java

Lucien Sina

3.6.2026

Table des matières

Préface

Ce livre a été écrit avec un objectif clair en tête : vous guider à travers les concepts fondamentaux de la programmation et vous amener progressivement aux principes de la conception orientée objet, un paradigme essentiel pour le développement de logiciels modernes.

La programmation ne se résume pas à écrire du code ; il s'agit de résoudre des problèmes, de créer des solutions efficaces et d'apprendre à penser de manière logique et systématique. Ce livre commence par les bases : il présente ce qu'est la programmation, comment se forment les algorithmes et comment Java, un langage polyvalent et largement utilisé, peut être votre outil pour mettre en œuvre des solutions.

Au fil des chapitres, vous découvrirez les éléments essentiels de la programmation, tels que les boucles, les types de données et les méthodes. Le livre se penche ensuite sur les principes de la conception orientée objet, en abordant des concepts tels que l'encapsulation, l'héritage, le polymorphisme et la classification. Tout au long

du livre, des exercices, des exemples et des solutions vous aideront à renforcer votre compréhension et vous encourageront à appliquer ce que vous avez appris.

Ce livre est destiné aux débutants, mais il ne s'arrête pas là. Pour ceux qui sont prêts à passer à l'étape suivante, il offre une transition en douceur vers des sujets plus avancés. De plus, j'ai inclus un aperçu des futurs livres, comme un consacré aux compétences avancées en programmation, un autre aux algorithmes et aux structures de données, et une exploration plus approfondie de sujets théoriques comme la calculabilité et la complexité. Ces livres sont destinés à compléter celui-ci, vous aidant à évoluer en tant que programmeur et résolveur de problèmes.

J'espère que ce livre vous sera utile tout au long de votre parcours dans la programmation. Si vous avez des commentaires, des questions ou des suggestions, n'hésitez pas à nous contacter.

Bon codage !

Chapitre 1

Problèmes, algorithmes et programmes

1.1 Introduction

Chaque programme commence par un problème. En programmation, un problème est une tâche ou une question qui nécessite une solution, et un programme est l'outil qui fournit cette solution à travers une série d'étapes logiques. L'objectif de ce chapitre est de vous présenter les concepts, problèmes, algorithmes et programmes fondamentaux et de vous donner un aperçu des langages et paradigmes de programmation qui influencent la manière dont les solutions sont élaborées et mises en œuvre.

1.2 Qu'est-ce qu'un problème ?

En programmation, un problème est une tâche que nous devons résoudre ou une question qui nécessite une réponse. Par exemple : Comment trier une liste de nombres dans l'ordre ? ou : Comment puis-je créer un logiciel qui aide à gérer les stocks dans un magasin ?

Un problème de programmation comporte généralement :

Entrées : Données ou informations fournies pour la solution (par exemple, une liste de nombres). Sorties : Le résultat souhaité (par exemple, la liste triée par ordre croissant). Contraintes : Règles ou limites dans lesquelles la solution doit fonctionner (par exemple, la liste ne doit pas dépasser une certaine longueur ou le processus de tri doit être efficace).

Exercice 1.1 : Identifiez un problème réel auquel vous êtes confronté, comme trouver le chemin le plus court pour vous rendre au travail. Identifiez :

Les entrées Les sorties souhaitées Les contraintes éventuelles Solution à l'exercice 1.1 : Exemple : Planifier l'itinéraire le plus court pour se rendre au travail.

Entrées : Lieu de départ, destination, données de trafic. Sorties : L'itinéraire le plus court. Contraintes : Minimiser le temps de trajet, éviter les péages ou les routes fermées.

1.3 Étapes pour transformer un problème du monde réel en programme

Le passage d'un problème du monde réel à un programme fonctionnel implique une série de transformations :

Problème du monde réel : identifier la tâche à résoudre. Abstraction : simplifier le problème du monde réel en se concentrant sur les détails pertinents tout en ignorant ceux qui ne sont pas nécessaires. Description du problème : décrire le problème en termes généraux. Préciser et formaliser : affiner la description pour spécifier les détails et les exigences clés. Spécification du problème : définir clairement les entrées, les sorties et les contraintes. Déclarer l'espace de solution possible : explorer les moyens potentiels de résoudre le problème. Algorithme : créer un plan étape par étape pour la solution. Programmation dans un langage d'ordre supérieur : traduire l'algorithme en langage de programmation. Traduction en code machine : convertir le code en instructions que le matériel d'un ordinateur peut exécuter. Entrée : fournir les données au programme. Sortie/Résultat : le programme traite l'entrée pour produire le résultat souhaité.

Exercice 1.2 : Pensez à une tâche simple, comme la préparation d'un gâteau. Décrivez chaque étape de la transformation de la tâche en une série d'instructions qu'une personne pourrait suivre, similaires aux étapes ci-dessus.

Solution à l'exercice 1.2 : Exemple : Faire cuire un gâteau.

Problème du monde réel : Faire un gâteau. Abstraction : Se concentrer uniquement sur les ingrédients et les étapes nécessaires. Description du problème : Comment faire cuire un gâteau ? Précision : Spécifier le type de gâteau, les ingrédients et la méthode de cuisson. Spécification : Recette exacte avec ingrédients et étapes. Solution : Identifier toutes les étapes nécessaires (mélange, cuisson). Algorithme : Instructions étape par étape. Langage d'ordre supérieur : Fiche de recette ou instructions de cuisson. Code machine : Actions effectuées par le boulanger. Entrée : Ingrédients. Sortie : Un gâteau fraîchement cuit.

1.4 Qu'est-ce qu'un algorithme ?

Un algorithme est un ensemble d'instructions claires, étape par étape, conçues pour résoudre un problème. Considérez un algorithme comme une recette : il s'agit d'une séquence d'étapes à suivre pour obtenir un résultat spécifique.

Par exemple, si votre problème consiste à trier une liste de nombres, un algorithme fournit les étapes permettant d'organiser les nombres dans l'ordre. Les algorithmes doivent être :

Clair : Chaque étape doit être sans ambiguïté et facile à comprendre. Précis : Chaque détail est défini sans ambiguïté. Efficace : Idéalement, les étapes doivent

conduire à une solution de la manière la plus efficace possible.

Exercice 1.3 : Écrivez un algorithme pour vous brosser les dents. Chaque étape doit être claire et précise.

Solution à l'exercice 1.3 :

Prenez la brosse à dents. Appliquez du dentifrice sur la brosse. Placez la brosse à dents dans votre bouche. Déplacez la brosse sur toutes les dents pendant deux minutes. Rincez-vous la bouche et la brosse à dents.

1.5 Qu'est-ce qu'un programme ?

Un programme est l'implémentation d'un algorithme de manière à ce qu'un ordinateur puisse l'exécuter. Les programmes sont écrits dans des langages de programmation, qui nous permettent de traduire les algorithmes en instructions compréhensibles par un ordinateur. Par exemple, un programme qui trie des nombres suivrait les étapes d'un algorithme de tri, exprimé dans un langage que l'ordinateur peut exécuter.

Les programmes comprennent :

Code : Instructions écrites dans un langage de programmation. Données : Informations traitées par le programme. Logique : Prise de décision qui détermine les actions du programme.

Les langages de programmation sont des outils que nous utilisons pour écrire des programmes, chacun avec sa propre syntaxe (règles d'écriture des instructions) et ses propres fonctionnalités. Par exemple :

Python : connu pour sa simplicité, ce qui le rend idéal pour les débutants. Java (que nous utiliserons dans ce livre) : connu pour sa fiabilité et sa capacité à fonctionner sur plusieurs plates-formes.

Paradigmes de programmation : Un paradigme de programmation est une façon d'organiser et de penser le code. Différents paradigmes conviennent à différents types de problèmes, et la compréhension de ces paradigmes nous aide à choisir la meilleure approche pour une tâche donnée. Examinons quelques paradigmes courants :

- Programmation impérative : Écriture de séquences d'instructions qui indiquent à l'ordinateur comment obtenir un résultat. Exemples : - Programmation procédurale : Se concentrer sur des fonctions qui effectuent des opérations dans une séquence (par exemple, des étapes pour résoudre un problème mathématique).

Programmation orientée objet (POO) : organisation du code en objets qui représentent des choses du monde réel, comme la création d'un système de bibliothèque avec des objets livres et emprunteurs. Programmation déclarative : se concentrer sur ce que devrait être le résultat, laissant le comment au langage ou à l'environnement de programmation. Exemples :

Programmation fonctionnelle : Définition d'opérations comme une série de fonctions, souvent sans modification d'état ou de données. Courante dans les applications mathématiques, elle met l'accent sur des relations claires entre les entrées et les sorties. Programmation logique : Utilisation de la logique et des règles pour spécifier

les résultats souhaités. Un exemple est Prolog, où les relations et les faits sont utilisés pour déduire de nouvelles informations. Programmation orientée aspect (AOP) : Séparation des préoccupations transversales (par exemple, la journalisation ou la gestion des erreurs) pour rendre le code plus modulaire et plus facile à maintenir. L'AOP permet d'isoler les fonctionnalités qui affectent plusieurs domaines d'un programme.

Exercice 1.4 : Identifiez trois problèmes différents et réfléchissez au paradigme de programmation qui conviendrait le mieux à chacun d'eux. Justifiez vos choix.

Solution à l'exercice 1.4 :

Trier une liste de nombres : procédural, car il s'agit d'une séquence simple d'étapes. Système de gestion des stocks : orienté objet, car les articles et les opérations d'inventaire peuvent être représentés comme des objets. Système expert pour le diagnostic médical : programmation logique, car il peut utiliser des règles et des faits pour déduire des diagnostics possibles.

1.6 Résumé

Dans ce chapitre, nous avons défini les éléments fondamentaux de la programmation : les problèmes, les algorithmes et les programmes. Nous avons exploré comment un problème du monde réel est transformé en une solution programmée à travers une série d'étapes. Nous avons également présenté les langages et les paradigmes de programmation, fournissant un contexte sur la façon dont

les approches de programmation varient et pourquoi elles sont adaptées à différentes tâches. Maintenant, avec ces concepts de base à l'esprit, vous êtes prêt à plonger plus profondément dans la programmation et à commencer à créer des solutions !

Chapitre 2

Installation JDK

2.1 Introduction

Avant de vous lancer dans la programmation Java, il est essentiel de configurer un environnement de développement pour écrire, compiler et exécuter du code Java de manière efficace. Dans ce chapitre, nous vous guiderons dans l'installation du kit de développement Java (JDK) et la configuration d'un environnement de développement intégré (IDE). Nous aborderons deux options populaires, Eclipse et Android Studio, afin que vous puissiez choisir celle qui correspond le mieux à vos besoins. À la fin de ce chapitre, vous disposerez d'un environnement complet prêt pour la programmation.

Le JDK est nécessaire pour compiler et exécuter des programmes Java, et il fournit les bibliothèques de base nécessaires pour travailler avec Java.

Guide étape par étape pour l'installation du JDK : Téléchargez le JDK :

Visitez la page de téléchargement officielle d'Oracle JDK ou installez OpenJDK à partir du site Web d'OpenJDK. Choisissez la dernière version du JDK compatible avec votre système d'exploitation (Windows, macOS ou Linux). Installez le JDK :

Suivez les instructions d'installation sur la page de téléchargement. La plupart des programmes d'installation proposent un chemin par défaut, que vous pouvez conserver, ou vous pouvez choisir un chemin personnalisé. Définir les variables d'environnement (Windows uniquement) :

Ouvrez Propriétés système > Paramètres système avancés > Variables d'environnement. Sous "Variables système", recherchez Chemin, sélectionnez-le et cliquez sur "Modifier". Ajoutez un nouveau chemin vers le répertoire bin de votre JDK, par exemple

```
1 C:\Program Files\Java\jdk-xx\bin
```

Vérifiez l'installation :

Ouvrez un terminal ou une invite de commande et saisissez java -version. Si le JDK est correctement installé, la version du JDK s'affiche. Exercice 2.1 : ouvrez votre terminal ou votre invite de commande et exécutez java -version et javac -version. Vérifiez que les deux commandes reconnaissent l'installation de Java. Si vous voyez les informations de version, votre JDK est correctement installé.

Solution à l'exercice 2.1 : si votre installation a réussi, vous devriez voir un résultat similaire à : java version "17.0.1" 2021-10-19 LTS Java(TM) SE Runtime Environment (build 17.0.1+12-39) Java HotSpot(TM) 64-Bit Server VM (build 17.0.1+12-39, mode mixte, partage)

2.2 Choisir un IDE :

Un environnement de développement intégré (IDE) est essentiel pour la programmation, car il fournit des outils pour éditer, compiler et déboguer du code efficacement. Comparons brièvement Eclipse et Android Studio pour vous aider à choisir celui qui répond le mieux à vos besoins.

Éclipse :

Avantages : hautement personnalisable, léger, populaire pour une large gamme d'applications Java. Inconvénients : peut nécessiter des plugins pour certaines fonctionnalités, moins optimisé pour le développement Android. Idéal pour : le développement Java général et les applications plus volumineuses.

Android Studio :

Avantages : Excellent pour le développement Android, comprend des bibliothèques et des outils spécifiques à Android. Inconvénients : Nécessite plus de ressources système, mieux adapté aux projets axés sur Android. Idéal pour : Les débutants intéressés par le développement d'applications mobiles avec Java. Choisissez l'IDE qui correspond à vos objectifs. Si vous débutez en programma-

tion, vous trouverez peut-être plus facile de commencer avec l'interface plus simple d'Eclipse. Si le développement d'applications mobiles vous intéresse, Android Studio vous fournira une base solide.

2.3 Configuration d'Eclipse

Guide étape par étape pour l'installation et la configuration d'Eclipse Télécharger Eclipse :

Accédez à la page de téléchargement d'Eclipse. Sélectionnez « Eclipse IDE for Java Developers » et téléchargez le programme d'installation correspondant à votre système d'exploitation. Installez Eclipse :

Exécutez le programme d'installation téléchargé et sélectionnez les options d'installation par défaut. Une fois installé, lancez Eclipse. Créez un nouveau projet Java :

Ouvrez Eclipse et sélectionnez Fichier > Nouveau projet Java > . Nommez le projet (par exemple, « HelloWorld ») et cliquez sur « Terminer ». Écrivez votre premier programme :

Cliquez avec le bouton droit sur le dossier src dans l'explorateur de projets. Sélectionnez Nouvelle classe > , nommez-la HelloWorld et cochez « public static void main(String[] args) » pour inclure la méthode principale. Dans la méthode principale, saisissez :

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Hello, World!");
4     }
```

```
5 }
```

Enregistrez le fichier et exécutez-le en cliquant avec le bouton droit de la souris et en sélectionnant Exécuter en tant que > Application Java. Exercice 2.2 : Écrivez et exécutez votre premier programme « Hello, World ! » dans Eclipse, en suivant les étapes ci-dessus. Vérifiez que la sortie s'affiche dans la console.

2.4 Configuration d'Android Studio

Guide étape par étape pour l'installation et la configuration d'Android Studio Téléchargez Android Studio :

Visitez la page de téléchargement d'Android Studio et téléchargez le programme d'installation correspondant à votre système d'exploitation. Installez Android Studio :

Exécutez le programme d'installation et suivez les instructions pour terminer l'installation. Android Studio téléchargera également les SDK et les outils nécessaires au développement Android. Créer un nouveau projet :

Ouvrez Android Studio et sélectionnez Nouveau projet. Choisissez un modèle de projet, comme « Aucune activité » pour un programme Java simple ou « Activité vide » si vous êtes intéressé par les applications Android. Nommez votre projet (par exemple, « HelloWorld ») et choisissez « Java » comme langage. Écrivez votre premier programme :

Dans le fichier Java principal du projet, créez une classe HelloWorld (ou modifiez l'activité principale pour

les projets Android). Dans la méthode principale, saisissez :

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Hello, World!");
4     }
5 }
```

S'il s'agit d'une application Android, vous configurerez plutôt un TextView dans la mise en page XML pour afficher « Hello, World ! » sur l'écran. Exécutez votre programme :

Pour les projets Java non Android, exécutez le programme en cliquant avec le bouton droit sur le fichier Java principal et en sélectionnant Exécuter. Pour les applications Android, sélectionnez Exécuter > Exécuter « app » pour tester l'application sur un émulateur ou un appareil connecté. Exercice 2.3 : créez et exécutez votre programme « Hello, World ! » dans Android Studio. Si vous travaillez avec une application Android, affichez le message à l'écran à l'aide d'un widget TextView.

La configuration d'un IDE peut parfois entraîner des problèmes inattendus. Voici quelques problèmes courants et leurs solutions :

JDK non reconnu : assurez-vous que le JDK est correctement installé et que les variables d'environnement sont définies (utilisateurs Windows). L'IDE plante ou ne s'ouvre pas : vérifiez si votre système répond aux exigences minimales de l'IDE. Le programme ne s'exécute

pas : vérifiez que la méthode principale est correctement écrite sous la forme public static void main(String[] args). Assurez-vous que vous exécutez le bon fichier. Astuce : si les problèmes persistent, consultez la documentation officielle ou les forums en ligne pour obtenir des conseils de dépannage. Eclipse et Android Studio ont tous deux de grandes communautés, de sorte que la plupart des problèmes ont des solutions publiées en ligne.

2.5 Résumé

Dans ce chapitre, nous avons mis en place les outils essentiels nécessaires à la programmation Java. Que vous ayez choisi Eclipse ou Android Studio, vous disposez désormais d'un environnement dans lequel vous pouvez écrire, exécuter et dépanner du code Java. Cette configuration constitue la base de votre parcours dans la programmation.

Dans le prochain chapitre, nous explorerons la structure du programme Java, en apprenant les diagrammes de syntaxe, les en-têtes de programme et d'autres aspects fondamentaux du code Java. Félicitations pour votre installation et l'écriture de votre premier programme !

Chapitre 3

Structure du programme Java

3.1 Introduction

Un programme Java possède une structure spécifique qui lui permet d'être compilé et exécuté. La compréhension de cette structure vous aidera à lire, écrire et déboguer du code efficacement. Dans ce chapitre, nous allons décomposer les parties d'un programme Java, explorer comment représenter visuellement la syntaxe du programme et introduire des concepts clés tels que les entêtes de programme, les blocs, les parties de déclaration et les parties d'instruction.

3.2 Diagrammes de syntaxe :

Les diagrammes de syntaxe (également appelés diagrammes de chemin de fer) sont un moyen utile de visualiser la structure des langages de programmation. Ils illustrent les règles de combinaison des éléments en Java et aident à clarifier la manière dont les différents composants du code s'articulent.

Concepts clés : Terminaux : éléments fixes du langage, comme les mots-clés (class , public , static) et les symboles (, , ;), qui doivent apparaître exactement comme écrits. Non-terminaux : éléments variables représentant des espaces réservés ou des modèles qui doivent être remplis avec des valeurs spécifiques, comme des identifiants (par exemple, des noms de variables) ou des expressions. Exemple : un diagramme de syntaxe pour une définition de classe Java pourrait montrer qu'une classe commence par le mot-clé class , suivi d'un identifiant (le nom de la classe), puis du bloc de code ... .

3.3 Structure de base d'un programme Java

La structure d'un programme Java se compose généralement des éléments clés suivants :

1. Titre du programme (définition de classe) :

Chaque programme Java commence par une définition de classe. L'en-tête du programme comprend des mots-clés tels que public et class , suivis du nom de la classe,

et constitue la structure de niveau supérieur. Exemple :

```
1 public class HelloWorld {
2     // bloc de code
3 }
```

Diagramme de syntaxe pour la définition de classe

public (facultatif) -> classe -> ClassName -> -> Bloc de code ->

Explication :

public (optionnel) : Le modificateur d'accès, qui est facultatif ici. class : Le mot-clé pour définir une classe. ClassName : Espace réservé pour le nom de la classe (un non-terminal). : Des accolades entourent le bloc de code, qui contient le corps principal de la classe.

2. Méthode principale (point d'entrée du programme) :

La méthode principale, public static void main(String[] args), est le point d'entrée des programmes Java. Cette méthode définit ce que le programme exécutera lors de son exécution. Exemple :

```
1 public static void main(String[] args) {
2 //Déclarations
3 }
```

3. Bloc de code :

Un bloc de code est une section de code incluse dans , regroupant des instructions ou des déclarations. Les blocs définissent la portée des sections de code, ce qui

signifie que les variables et les instructions d'un bloc ne sont accessibles qu'à l'intérieur de ce bloc. Exemple :

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Hello, World!");
4     }
5 }
```

3.4 Partie Déclaration

La partie déclaration est l'endroit où vous définissez les variables, les constantes et les autres éléments nécessaires au programme. Les déclarations apparaissent généralement au début d'une méthode ou d'une classe, ce qui vous permet d'organiser les informations sur les données avant de les utiliser dans des instructions.

Variables : les variables stockent des données qui peuvent changer tout au long de l'exécution du programme. Constantes : les constantes contiennent des valeurs qui ne changent pas. Exemple :

```
1 int age = 25;
2 //déclaration de variable
3 final double PI = 3.14159;
4 // déclaration de constantes
```

Diagramme de syntaxe pour la déclaration :

Une déclaration de variable pourrait ressembler à : [type] [nom de la variable] [valeur initiale (facultatif)] ;