Chapitre 3
Programmation objet

1. Introduction à la POO

Avec Java, la notion d'objet est omniprésente et nécessite un minimum d'apprentissage. Nous allons donc voir dans un premier temps les principes de la programmation objet et le vocabulaire associé, puis nous verrons comment mettre cela en application avec Java.

Dans un langage procédural classique, le fonctionnement d'une application est réglé par une succession d'appels aux différentes procédures et fonctions disponibles dans le code. Ces procédures et fonctions sont chargées de manipuler les données de l'application qui sont représentées par les variables de l'application. Il n'y a aucun lien entre les données et le code qui les manipule. Dans un langage objet, on va au contraire essayer de regrouper le code. Ce regroupement est appelé une classe. Une application développée avec un langage objet est donc constituée de nombreuses classes représentant les différents éléments manipulés par l'application. Les classes vont décrire les caractéristiques de chacun des éléments. C'est ensuite l'assemblage de ces éléments qui va permettre le fonctionnement de l'application.

Ce principe est largement utilisé dans d'autres domaines que l'informatique. Dans l'industrie automobile, par exemple, il n'existe certainement pas, chez aucun constructeur, un plan complet décrivant les milliers de pièces constituant un véhicule. Cependant, chaque sous-ensemble d'un véhicule peut être représenté par un plan spécifique (le châssis, la boîte de vitesses, le moteur...). Chaque sous-ensemble est également décomposé jusqu'à la pièce élémentaire (un boulon, un piston, un pignon...). C'est l'assemblage de tous ces éléments qui permet la fabrication d'un véhicule.

En fait, ce n'est pas l'assemblage des plans qui permet la construction du véhicule, mais l'assemblage des pièces fabriquées à partir de ces plans. Dans une application informatique, c'est l'assemblage des objets créés à partir des classes qui va permettre le fonctionnement de l'application. Les deux termes classe et objet sont souvent confondus, mais ils représentent des notions bien distinctes. La classe décrit la structure d'un élément alors que l'objet représente un exemplaire créé sur le modèle de cette structure. Après sa création, un objet est indépendant des autres objets construits à partir de la même classe. Par exemple, une portière de voiture pourra après fabrication être peinte d'une couleur différente des autres portières fabriquées selon le même plan.

Les classes sont constituées de champs et de méthodes. Les champs représentent les caractéristiques des objets. Ils sont représentés par des variables et il est donc possible de lire leur contenu ou de leur affecter une valeur directement. Le robot qui va peindre une portière va modifier le champ couleur de cette portière. Les méthodes représentent les actions qu'un objet peut effectuer. Elles sont mises en œuvre par la création de procédures ou de fonctions dans une classe.

Ceci n'est qu'une facette de la programmation orientée objet. Trois autres concepts sont également fondamentaux :

– l'encapsulation,

– l'héritage,

– le polymorphisme.

L'encapsulation consiste à cacher les éléments qui ne sont pas nécessaires pour l'utilisation d'un objet. Cette technique permet de garantir que l'objet sera correctement utilisé. C'est un principe qui est aussi largement utilisé dans d'autres domaines que l'informatique. Pour reprendre l'exemple de l'industrie automobile, savez-vous comment fonctionne la boîte de vitesses de votre voiture ?

Pour changer de vitesse allez-vous directement modifier la position des différents engrenages ? Heureusement que non. Les constructeurs ont en fait prévu des solutions plus pratiques pour la manipulation de la boîte de vitesse.

Les éléments d'une classe visibles de l'extérieur de la classe sont appelés l'interface de la classe. Dans le cas de notre voiture, le levier de vitesses constitue l'interface de la boîte de vitesses. C'est par son intermédiaire que l'on peut agir sans risque sur les mécanismes internes de la boîte de vitesses.

L'héritage permet la création d'une nouvelle classe à partir d'une classe existante. La classe servant de modèle est appelée classe de base, classe mère ou super-classe. La classe ainsi créée hérite des caractéristiques de sa classe de base. Elle peut aussi être personnalisée en y ajoutant des caractéristiques supplémentaires. Les classes créées à partir d'une classe de base sont appelées classes dérivées, classes filles ou sous-classes. Ce principe est bien sûr aussi utilisé dans le monde industriel. La boîte de vitesses de votre voiture comporte peut-être cinq rapports. Les ingénieurs qui ont conçu cette pièce ne sont certainement pas repartis de zéro. Ils ont repris le plan de la génération précédente (quatre rapports) et y ont ajouté des éléments. De même que les ingénieurs qui ont réfléchi à la boîte de vitesses à six rapports sont repartis de la version précédente.

Le polymorphisme est une autre notion importante de la programmation orientée objet. Par son intermédiaire, il est possible d'utiliser plusieurs classes de manière interchangeable même si le fonctionnement interne de ces classes est différent. Si vous savez changer de vitesse sur une voiture Peugeot, vous savez également comment le faire sur une voiture Renault et pourtant les deux types de boîtes de vitesses ne sont pas conçus de la même façon.

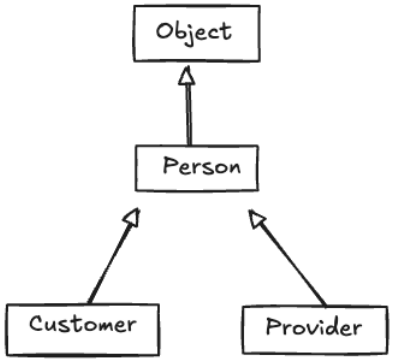
Deux autres concepts sont également associés au polymorphisme. La surcharge et la substitution de méthodes. La surcharge est utilisée pour concevoir dans une classe des méthodes ayant le même nom, mais ayant un nombre de paramètres différent ou des types de paramètres différents. La substitution est utilisée lorsque dans une classe dérivée, on souhaite modifier le fonctionnement d'une méthode dont on a hérité. Le nombre et le type des paramètres restent identiques à ceux définis dans la classe de base.

Voici mis en place les quelques principes de base de la programmation objet. Le sujet de cet ouvrage n'étant pas la mécanique automobile nous allons tout de suite voir la mise en œuvre de ces principes avec le langage Java.

2. Mise en œuvre de la POO avec Java

2.1 Contexte

Dans le reste de ce chapitre, nous allons travailler sur la classe `Person` et ses sous-classes. Cette classe `Person` représentera une personne au sens courant du terme en français. Notez qu'il est généralement recommandé de nommer ses classes en anglais, et nous respecterons cette règle. Vous trouverez ci-dessous une représentation simplifiée en UML (*Unified Modeling Language*) de cette hiérarchie de classes.



UML est un langage graphique dédié à la représentation des concepts de programmation orientée objet. Pour plus d'informations sur ce langage, vous pouvez consulter l'ouvrage UML 2.5 dans la collection Ressources Informatiques des Éditions ENI.

En UML, une flèche triangulaire vide décrit une dérivation. La classe dérivée est la classe de base, mais avec des propriétés additionnelles ou modifiées, comme le concept d'héritage présenté précédemment.

Nous avons la classe `Person` qui hérite de la classe `Object` (comme toute classe en Java). Cette classe `Person` possédera deux classes filles : `Customer` (un client) et `Provider` (un fournisseur).

2.2 Création d'une classe

La création d'une classe passe par la déclaration de la classe elle-même et de tous les éléments la constituant.

2.2.1 Déclaration de la classe

La déclaration d'une classe se fait en utilisant le mot-clé `class` suivi du nom de la classe puis d'un bloc de code délimité par les caractères `{` et `}`. Dans ce bloc de code, on trouve des déclarations de variables qui seront les champs de la classe et des fonctions qui seront les méthodes de la classe. Plusieurs mots-clés peuvent être ajoutés pour modifier les caractéristiques de la classe. La syntaxe générale de déclaration d'une classe est donc la suivante :

```
[modificateurs] class NomDeLaClasse
                [extends NomDeLaClasseDeBase]
                [implements NomDeInterface1,NomDeInterface2, ...] {
{
    Code de la classe
}
```

Les signes `[` et `]` sont utilisés ici pour indiquer le caractère optionnel de l'élément. Ils ne doivent pas être utilisés dans le code de déclaration d'une classe.

Les modificateurs permettent de déterminer la visibilité de la classe et la manière de l'utiliser. Voici la liste des modificateurs disponibles :

`public` : indique que la classe peut être utilisée par toutes les autres classes. Sans ce modificateur, la classe ne sera utilisable que par les autres classes faisant partie du même `package`. Pour plus d'informations sur les packages, veuillez vous référer à la section éponyme un peu plus loin dans le chapitre.

`abstract` : indique que la classe est abstraite et ne peut donc pas être instanciée. Elle ne peut être utilisée que comme classe de base dans une relation d'héritage. En général, dans ce genre de classe, seules les déclarations de méthodes sont définies, et il faudra écrire le contenu des méthodes dans les classes dérivées.

`final` : la classe ne peut pas être utilisée comme classe de base dans une relation d'héritage. Autrement dit, il n'est pas possible de la dériver. Elle peut uniquement être instanciée.

La signification des mots-clés `abstract` et `final` étant contradictoire, leur utilisation simultanée est bien sûr interdite.

Pour indiquer que votre classe récupère les caractéristiques d'une autre classe par une relation d'héritage, vous devez utiliser le mot-clé `extends` suivi du nom de la classe de base. Vous pouvez également implémenter dans votre classe une ou plusieurs interfaces en utilisant le mot `implements` suivi de la liste des interfaces implémentées. Ces deux notions seront vues en détail plus loin dans ce chapitre.

Le début de la déclaration de la classe `Person` est donc le suivant :

```
public class Person {

}
```

Ce code doit obligatoirement être saisi dans un fichier ayant le même nom que la classe (`Person`) et l'extension `.java`.

2.2.2 Création des champs

Intéressons-nous maintenant au contenu de la classe. Nous devons créer les différents champs (ou variables membres) de la classe. Pour cela, il suffit de déclarer des variables à l'intérieur du bloc de code de la classe en indiquant la visibilité de la variable, son type et son nom.

```
[private | protected | public] typeDeLaVariable nomDeLaVariable;
```

Il est possible de définir des variables de classe ou des constantes en utilisant les mots-clés `static` et `final`. Veuillez vous référer au chapitre sur les bases du langage pour plus d'informations.

La visibilité de la variable répond aux règles suivantes :

`private` : la variable n'est accessible que dans la classe où elle est déclarée.

`protected` : la variable est accessible dans la classe où elle est déclarée, dans les autres classes faisant partie du même package et dans les classes héritant de la classe où elle est déclarée.

`public` : la variable est accessible à partir de n'importe où.

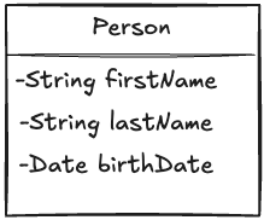
Si aucune information n'est fournie concernant la visibilité, la variable est accessible à partir de la classe où elle est déclarée et des autres classes faisant partie du même package.

Lorsque vous choisissez la visibilité d'une variable, vous devez autant que possible respecter le principe d'encapsulation et limiter au maximum la visibilité des variables. L'idéal étant de toujours avoir des variables `private` ou `protected` mais jamais `public`.

La variable doit également avoir un type. Il n'y a pas de limitation concernant le type d'une variable et vous pouvez donc utiliser aussi bien les **types primitifs** du langage Java, tels que `int`, `float`, `char`, que des **types références** (comme `String`, `LocalDate` ou des objets que vous avez vous-même créés).

Le nom de la variable doit quant à lui simplement respecter les règles de nommage (pas d'utilisation de mot-clé du langage). Veuillez vous référer au chapitre sur les conventions de nommage (cf. chapitre Comprendre un programme - Les conventions de nommage) pour plus d'informations.

On peut représenter la classe `Person` avec ses variables membres en UML de la façon suivante :



La classe est composée de trois variables privées (le `-` devant les noms de variables indique ce caractère privé). Les variables `firstName` et `lastName` (nom et prénom) sont de type `String` et la variable `birthDate` (date de naissance) est de type `Date`. Le type `Date` est un type UML standard. Lors de la transposition en code, il est possible d'utiliser un type différent adapté au langage utilisé.

La classe `Person` prend donc maintenant la forme suivante :

```
public class Person {
    private String firstName;
    private String lastName;
    private LocalDate birthDate;
}
```

2.2.3 Création des méthodes

Les méthodes sont simplement des fonctions définies à l'intérieur d'une classe. Elles sont en général utilisées pour manipuler les champs de la classe. La syntaxe générale de déclaration d'une méthode est décrite ci-dessous.

```
[modificateurs] typeRetour nomMethode ([listeParametres])
                                        [throws listeException] {


}
```

Les modificateurs de visibilité suivants sont disponibles :

`private` : indique que la méthode ne peut être utilisée que dans la classe où elle est définie.

`protected` : indique que la méthode peut être utilisée dans la classe où elle est définie, dans les sous-classes de cette classe et dans les classes faisant partie du même package.

`public` : indique que la méthode peut être utilisée depuis n'importe quelle autre classe.

Si aucun de ces mots-clés n'est utilisé, alors la visibilité sera limitée au package dans lequel la classe est définie.

Des modificateurs complémentaires sont disponibles.

`static` : indique que la méthode est une méthode de classe.

`abstract` : indique que la méthode est abstraite et qu'elle ne contient pas de code. La classe où elle est définie doit elle aussi être abstraite.

`final` : indique que la méthode ne peut pas être substituée dans une sous-classe.

`native` : indique que le code de la méthode se trouve dans un fichier externe écrit dans un autre langage.

`synchronized` : indique que la méthode ne peut être exécutée que par un seul thread à la fois.

Le type de retour peut être n'importe quel type de données, type valeur ou type référence. Si la méthode n'a pas d'information à renvoyer, vous devez utiliser le mot-clé `void` en remplacement du type de retour.

La liste des paramètres est identique à une liste de déclaration de variables. Il faut spécifier le type du paramètre et le nom du paramètre. Si plusieurs paramètres sont attendus, il faut séparer leur déclaration par une virgule. Même si aucun paramètre n'est attendu, les parenthèses sont tout de même obligatoires.

Chapitre 3
Introduction à la plate-forme Java

1. Introduction

Au cœur des ordinateurs, des listes d'instructions sont exécutées par les microprocesseurs. Ces listes contiennent des codes binaires représentant des calculs et des actions à faire sur des registres, de la mémoire ou encore sur des périphériques. Sauf cas très rare, lorsqu'un développeur écrit un programme, il ne construit pas directement cette liste d'instructions aussi appelée code de bas niveau ou langage assembleur. Heureusement pour lui, il passe par un langage de programmation avec lequel il va décrire un enchaînement d'actions de « haut niveau ». Par exemple, si le programme a besoin d'ouvrir le fichier config.txt alors le développeur utilisera la fonction `fopen("config.txt")` de son langage favori pour le faire. Derrière cette ligne se cachera toute une liste d'instructions « machine » que le microprocesseur exécutera quand l'application sera exécutée. Les langages de hauts niveaux simplifient donc grandement l'écriture mais, au final, ce sont toujours des instructions machine qui seront déroulées.

Le programme source – dans notre exemple, celui qui contient `fopen("config.txt")` – est un fichier texte que le développeur construit avec un éditeur de texte basique comme Notepad, Notepad++ ou bien encore l'éditeur intégré à son environnement de développement (IDE), présenté plus loin. Ce fichier texte tout seul ne peut pas être exécuté par la machine. Il doit être converti directement ou indirectement en langage « machine » pour devenir exécutable.

Cette translation peut s'opérer de trois façons différentes :

– **La compilation directe** : le fichier est converti en un contenu binaire directement exécutable par le système d'exploitation. C'est le cas de la plupart des programmes écrits en C et C++ (la plupart, car il est possible d'écrire des programmes .NET en C++, mais c'est une autre histoire).

– **L'interprétation** : le fichier est converti par un interpréteur « à la volée ». C'est le cas des scripts PHP dans les pages web. Le navigateur internet du client demande une page à un serveur web. Le serveur web détecte que la page demandée contient un script PHP car son contenu est dynamique (exemple : liste de clients, commandes en cours, etc.). Il la passe alors à son analyseur qui va l'interpréter et composer dynamiquement la page HTML. Cette méthode est très pratique dans le cas du Web car les scripts sont écrits au format texte. En cas de modifications, pas besoin de recompiler quoi que ce soit, il suffit de mettre à jour le fichier dans l'arborescence du serveur et le changement est immédiatement actif. Bien sûr, l'ensemble (décodage du PHP, son exécution et réencodage en HTML) peut s'avérer lent.

– **La compilation indirecte** : c'est le cas de Java et de C# et c'est un mélange des deux précédents... En fait, le fichier source est compilé dans un format binaire intermédiaire qui va être exécuté très rapidement par une machine virtuelle. Pourquoi cette machine virtuelle entre le programme et le système d'exploitation ? Cette machine virtuelle sert à plusieurs choses : tout d'abord elle va vérifier que le programme source ne fait pas de « bêtises » pendant son exécution en allant écrire, par exemple, dans la mémoire des autres applications, et ainsi rendre le système d'exploitation instable. Elle va également s'occuper de la gestion de la mémoire, de la sécurité, des droits, etc. La machine virtuelle est l'amie des systèmes d'exploitation et l'amie des développeurs. D'un côté, elle assure une exécution machine la plus fiable possible, et de l'autre, elle fédère un code source (pratiquement) unique pour toutes les plates-formes.

Microsoft Windows propose .NET et sa machine virtuelle. .NET se programme à partir de nombreux langages (même si le C# a été conçu pour lui). Ce puissant framework était au départ très lié aux systèmes d'exploitation Microsoft (PC, serveurs, Web, tablettes, téléphones) mais est maintenant multiplate-forme et open source. Cet effort d'ouverture de Microsoft est relativement récent alors que, depuis le départ, Java a pu s'exécuter sur la plupart des systèmes d'exploitation dès qu'ils proposaient une machine virtuelle appropriée.

Remarque

Une plate-forme Java désigne donc un environnement d'exécution pour un système d'exploitation sur une machine donnée et l'environnement de développement associé.

2. Environnement d'exécution

Les applications écrites en Java ne communiquent jamais directement avec le système d'exploitation. D'ailleurs, le résultat d'une compilation Java (fichiers d'extension *.class*) n'est pas directement exécutable par le système d'exploitation. Cette première mouture, appelée *Byte Code*, contient des instructions pour la machine virtuelle Java qui va « convertir à la volée » le code intermédiaire en instructions compatibles avec l'environnement réel d'exécution. On parle de *Just In Time Compiler* (ou *JIT Compiler*).

Chaque environnement réel d'exécution (Windows, Linux, macOS...) a sa propre machine virtuelle Java, mais le *Byte Code* généré après la compilation du programme source est le même et est relativement portable entre les différentes machines virtuelles Java. L'adverbe « relativement » est ajouté ici pour préciser que la conception d'un programme « portable » requiert quelques précautions. Imaginons que vous souhaitiez réaliser un programme qui devra être fonctionnel à la fois sur PC, tablette et téléphone. Il est évident que les caractéristiques de ces trois dispositifs diffèrent totalement, et il faudra donc que le programme s'adapte automatiquement à l'environnement d'exécution. Il pourra, par exemple, lui demander la résolution du dispositif d'affichage pour adapter sa présentation en conséquence.

Remarque

Revers de la médaille : le résultat de la compilation peut être facilement désassemblé, c'est-à-dire converti du Byte Code en lignes de programmes source Java.

Cela est problématique quand, par exemple, vos concurrents parviennent à reconstituer votre code source à partir du produit compilé que vous commercialisez... Il est possible de rendre le résultat de cette action moins lisible en effectuant un traitement d'obscurcissement (obfuscation) avec des outils spécialisés tels que ProGuard (open source). Sachez que le problème est le même pour nos collègues développeurs C#...

3. Une librairie très complète

La plate-forme Java propose une très large collection de classes sur lesquelles s'appuient les applications. On parle d'API Java. Ces classes simplifient considérablement la gestion d'objets courants (chaînes de caractères, valeurs décimales, collections, etc.), mais aussi la gestion de fichiers, des interfaces graphiques classiques, les API web, l'accès aux bases de données, les communications réseau, la sécurité, les diagnostics, etc. La liste des classes est très conséquente et, même si elles sont proposées de façon hiérarchique, le problème est souvent de savoir s'y repérer !

Ces classes sont, la plupart du temps, extensibles. Cela veut dire qu'il est possible de les étendre pour profiter de leurs comportements de base puis d'y ajouter des spécificités métier.

Pour organiser ces classes, la plate-forme utilise le concept des packages qui regroupent les classes par finalités (services de même nature). Par exemple, le package `java.io` contient une « boîte à outils » pour gérer les fichiers.

Il est naturellement possible de créer ses propres packages. Le package Java correspond au package de l'UML. Le découpage effectué veille à réduire les dépendances entre les packages. Pour qu'un programme puisse utiliser un package, ce dernier doit être référencé dans le projet et son utilisation doit être déclarée en haut du fichier source concerné grâce à la directive `import`, par exemple : `import java.io.*;`.

4. Des outils de développement performants

La plate-forme Java propose un compilateur (`javac.exe`) que l'on peut utiliser directement en ligne de commandes après avoir saisi le code du programme dans un éditeur de texte. La procédure est tout à fait possible mais pas très productive... Évidemment, le développeur recherche l'utilisation d'un environnement de développement totalement intégré, l'assistant pendant la rédaction du code, la réalisation de l'interface graphique, la mise au point et le déploiement de son application. Ce programme est un IDE (*Integrated Development Environment*). Il intègre au moins un éditeur de code source, des outils automatisant les compilations et un débogueur.

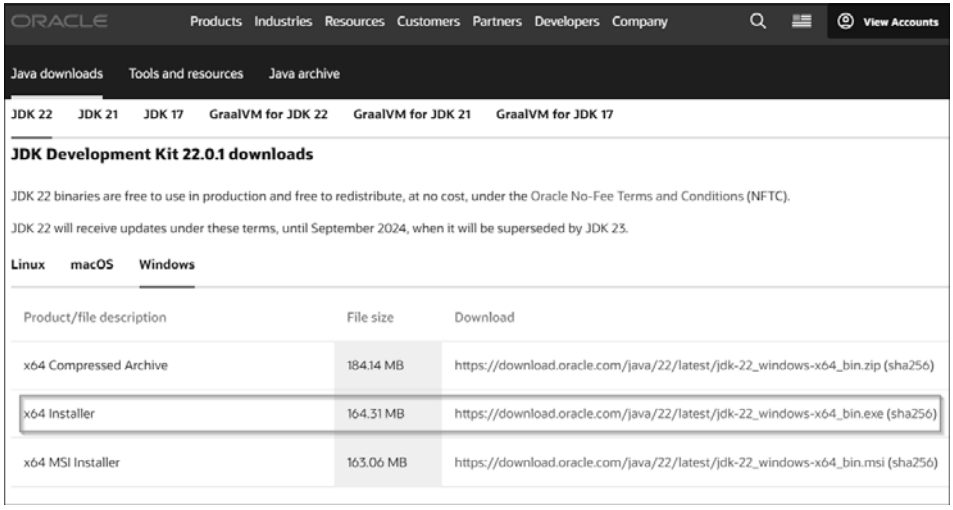
Il existe plusieurs IDE pour Java avec parmi eux trois grandes références : IntelliJ IDEA de JetBrains, Eclipse de la fondation Eclipse et NetBeans créé à l'initiative de Sun Microsystems et maintenu et distribué par Oracle et par Apache Software Foundation pour sa version Apache NetBeans. Les démonstrations et exercices de ce livre sont basés sur l'utilisation d'IntelliJ IDEA car c'est l'IDE qui nous a semblé le plus simple pour commencer à développer.

5. Téléchargement et installation du JDK

- Pour développer en Java nous aurons besoin d'un « kit de développement » appelé JDK sur lequel nous reviendrons très vite… Pour l'instant, rendez-vous sur le site Oracle à cette adresse : https://www.oracle.com/java/technologies/downloads/#jdk22-windows pour télécharger la version du JDK 22 avec son programme d'installation.

Remarque

Cette version 22 ne sera peut-être pas la dernière version lorsque vous lirez ces lignes.

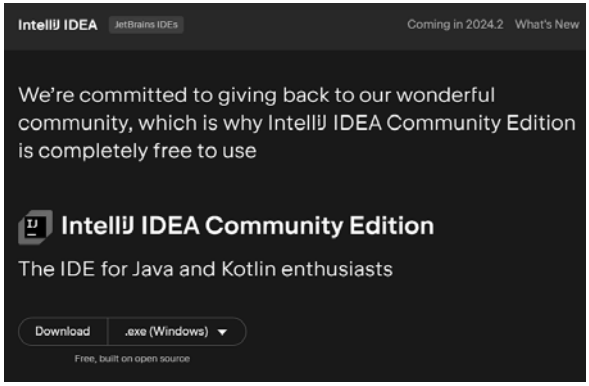


- Exécutez le programme téléchargé : jdk-22_windows-x64_bin.exe en gardant les options par défaut proposées dans les différentes boîtes de dialogue.

6. Téléchargement et installation d'IntelliJ IDEA

- Rendez-vous maintenant sur le site officiel de JETBRAINS (https://www.jetbrains.com/idea/download/?section=windows) pour télécharger la version Community Edition de cet IDE.

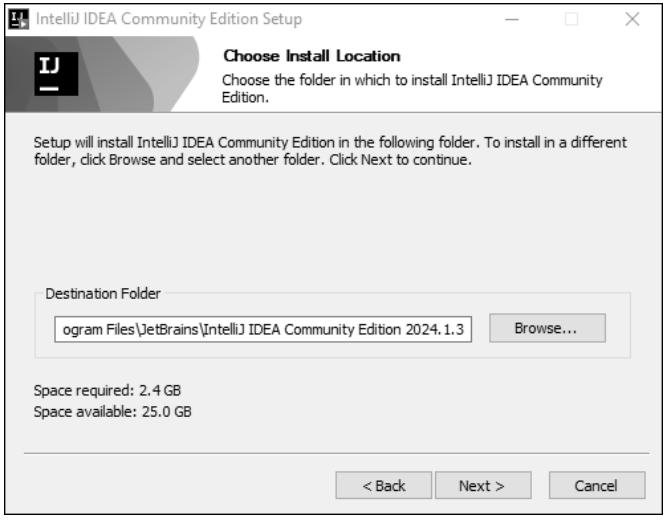
- Faites défiler l'écran jusqu'à l'affichage suivant :



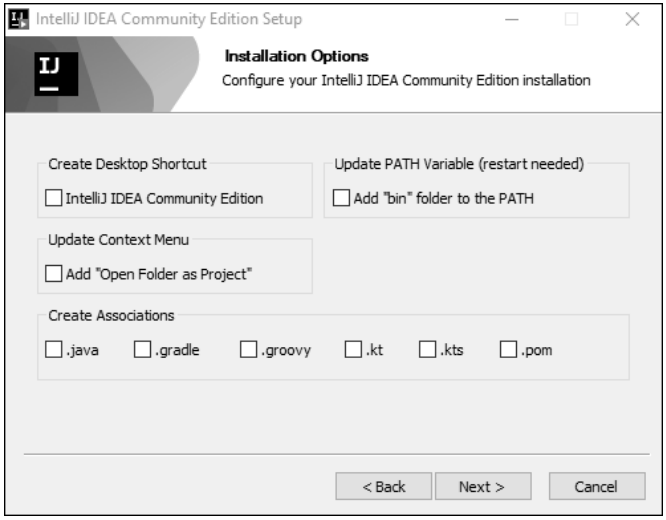
- Cliquez sur le bouton **Download**.

- Une fois le téléchargement terminé, double cliquez sur le fichier reçu (ideaIC-2024.1.3.exe à l'écriture de cet ouvrage).
- Autorisez l'application à apporter des modifications.

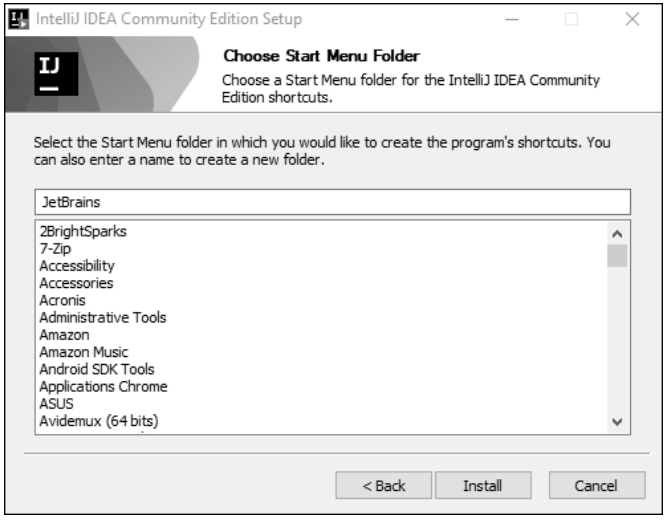
▶Cliquez sur le bouton **Next**.



▶Conservez les valeurs proposées par défaut et cliquez sur le bouton **Next**.



▶Conservez les valeurs proposées par défaut et cliquez sur le bouton **Next**.



▶Conservez les valeurs proposées par défaut et cliquez sur le bouton **Install**.