

Capítulo 3

Programación orientada a objetos

1. Introducción a la POO

Con Java, la noción de objeto es omnipresente y precisa un mínimo de aprendizaje. En primer lugar, nos centraremos en los principios de la programación orientada a objetos y en su vocabulario asociado, para pasar luego a ponerlo todo en práctica con Java.

En un lenguaje procedural clásico, el funcionamiento de una aplicación está regido por una sucesión de llamadas a diferentes procedimientos y funciones disponibles en el código. Estos procedimientos y funciones son los encargados de procesar los datos de la aplicación representados por las variables de esta última. No existe relación alguna entre los datos y el código que los manipula. Por el contrario, en un lenguaje orientado a objetos, lo que se intenta es agrupar el código. A esta agrupación se le llama clase. Por lo tanto, una aplicación desarrollada con un lenguaje orientado a objetos está formada por numerosas clases que representan los diferentes elementos procesados por la aplicación. Las clases describirán las características de cada uno de los elementos. El ensamblado de estos elementos va a permitir el funcionamiento de la aplicación.

Este principio se utiliza ampliamente en otras disciplinas, además de la informática. En la industria automovilística, por ejemplo, seguramente no existe ningún constructor que disponga de un plano que contenga todas y cada una de las piezas que constituyen un vehículo. Sin embargo, cada subconjunto de un vehículo puede estar representado por un plano específico (el chasis, la caja de cambios, el motor...). Cada subconjunto se va descomponiendo a su vez hasta la pieza elemental (un perno, un pistón, una rueda dentada...). Es el ensamblado de todos estos elementos lo que permite la fabricación de un vehículo.

De hecho, no es el ensamblado de los planos lo que permite la construcción del vehículo, sino el de las piezas fabricadas a partir de ellos. En una aplicación informática, será el ensamblado de los objetos creados a partir de las clases lo que permitirá el funcionamiento de la aplicación. Los dos términos, clase y objeto, suelen confundirse, aunque representan nociones bien distintas. La clase describe la estructura de un elemento, mientras que el objeto representa un ejemplar creado a partir del modelo de dicha estructura. Tras su creación, un objeto es independiente de otros objetos construidos a partir de la misma clase. Por ejemplo, después de la fabricación, la puerta de un coche podrá pintarse de un color distinto al de las otras puertas fabricadas con el mismo plano.

Las clases están constituidas por campos y métodos. Los campos representan las características de los objetos. Están identificadas por variables y es posible leer su contenido o asignarles un valor directamente. El robot que va a pintar una puerta cambiará el campo color de dicha puerta. Los métodos representan acciones que un objeto puede efectuar. Se ejecutan mediante la creación de procedimientos o de funciones en una clase.

Esto solo es una faceta de la programación orientada a objetos. Del mismo modo, existen otros tres conceptos fundamentales:

- la encapsulación
- la herencia
- el polimorfismo

La encapsulación consiste en esconder elementos no necesarios para la utilización de un objeto. Esta técnica permite garantizar que el objeto se utilice correctamente. Se trata de un principio muy utilizado en otras disciplinas, aparte de la informática. Regresando al ejemplo de la industria del automóvil, ¿conoce usted cómo funciona la caja de cambios de su vehículo?

Para cambiar de marcha, ¿modificará directamente la posición de los engranajes? Afortunadamente, no. Los constructores tienen previstas soluciones más prácticas para la manipulación de la caja de cambios.

Los elementos visibles de una clase desde su exterior forman la interfaz de clase. En el caso de nuestro coche, la palanca de cambios constituye la interfaz de la caja de cambios. Es a través de ella como se podrá actuar sin riesgo sobre los mecanismos internos de la caja de cambios.

La herencia permite la creación de una nueva clase a partir de otra ya existente. La clase que sirve de modelo se llama clase base, clase madre o superclase. La clase así creada hereda las características de su clase base. También es posible personalizarla añadiendo características adicionales. Las clases creadas a partir de una clase base se denominan clases derivadas, clases hijas o subclasses. Este principio se utiliza también en el mundo industrial. Seguramente, la caja de cambios de su coche incluye cinco marchas. A ciencia cierta, los ingenieros que concibieron esta pieza no partieron de cero. Retomaron el plano de la generación anterior (cuatro marchas) y le añadieron elementos. De la misma manera, los ingenieros que han reflexionado sobre la caja de cambios de seis marchas han tomado como base la versión precedente.

El polimorfismo es otra noción importante en la programación orientada a objetos. Gracias a él, es posible utilizar varias clases de manera intercambiable incluso si el funcionamiento interno de estas clases muestra diferencias. Si usted sabe cambiar la marcha en un coche Peugeot, también sabrá hacerlo en un Renault, y sin embargo los dos tipos de caja de cambios no fueron concebidos de la misma manera.

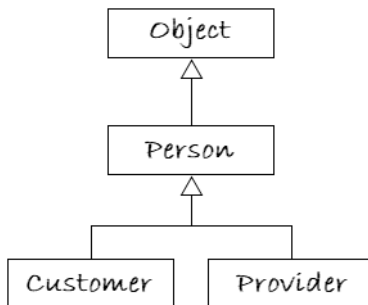
Asociados al polimorfismo existen otros dos conceptos: la sobrecarga y la sustitución de métodos. La sobrecarga se utiliza para diseñar en una clase métodos que compartan el mismo nombre, pero que tengan un número o tipos de parámetros distintos. Se utiliza la sustitución cuando, en una clase derivada, se quiere modificar el funcionamiento de uno de los métodos heredados. El número y el tipo de los parámetros se mantienen idénticos a los definidos en la clase base.

Veamos la puesta en práctica de algunos de los principios fundamentales de la programación orientada a objetos. Dado que el tema de este manual no versa sobre la mecánica automovilística, aplicaremos los conceptos de la orientación a objetos sobre el lenguaje Java.

2. Puesta en práctica de la POO con Java

2.1 Contexto

En el resto del capítulo, vamos a trabajar con la clase `Person` y sus subclases. Esta clase `Person` representará a una persona en el sentido común del término en español. Tenga en cuenta que generalmente se recomienda nombrar tus clases en inglés, y respetaremos esta norma. A continuación, se muestra una representación simplificada en UML (*Unified Modeling Language*) de esta jerarquía de clases.



UML es un lenguaje gráfico destinado a la representación de los conceptos de programación orientada a objetos. Para obtener más información sobre este lenguaje, puede consultar el manual UML 2.5 en la colección Recursos Informáticos de Ediciones ENI.

En UML, una flecha triangular vacía describe una derivación. La clase derivada es la clase base, pero con propiedades adicionales o modificadas, como el concepto de herencia presentado anteriormente.

Tenemos la clase `Person`, que hereda de clase `Object` (como cualquier clase en Java). Esta clase `Person` tendrá dos clases hijas: `Customer` (un cliente) y `Provider` (un proveedor).

2.2 Creación de una clase

La creación de una clase consiste en su declaración y la de todos los elementos que la componen.

2.2.1 Declaración de la clase

La declaración de una clase se lleva a cabo utilizando la palabra clave `class` seguida del nombre de la clase y de un bloque de código delimitado por los caracteres `{` y `}` (llaves). En este bloque de código se encuentran las declaraciones de variables, que serán los campos de la clase, y las funciones, que serán los métodos de la clase. Se pueden añadir varias palabras clave para modificar las características de la clase. Por lo tanto, la sintaxis de la declaración de una clase es la siguiente.

```
[modificadores] class NombreDeLaClase
    [extends NombreDeLaClaseBasica]
    [implements NombreDeInterfaz1,NombreDeInterfaz2,...]
{
    Código de la clase
}
```

Los signos `[` y `]` (corchetes) se utilizan para indicar qué elemento es opcional. No se deben utilizar en el código de declaración de una clase.

Los modificadores permiten determinar la visibilidad de la clase y la manera de utilizarla. A continuación presentamos la lista de los modificadores disponibles:

`public`: indica que la clase puede ser utilizada por cualquier otra clase. Sin este modificador, la clase solo podrán utilizarla clases que formen parte del mismo *package*. Para más información sobre los paquetes, vaya a la sección correspondiente, un poco más adelante en este capítulo.

`abstract`: indica que la clase es abstracta y no puede ser instanciada. Solo se la puede utilizar como clase básica en una relación de herencia. En este tipo de clase solo se suelen definir las declaraciones de métodos y habrá que escribir el contenido de los métodos en las clases derivadas.

`final`: la clase no puede utilizarse como clase de base en una relación de herencia. Dicho de otra manera, no es posible derivarla. Solo puede ser instanciada.

Al ser contradictorio el significado de las palabras clave `abstract` y `final`, su uso simultáneo está prohibido.

Para indicar que su clase recupera las características de otra clase por una relación de herencia, debe utilizar la palabra clave `extends` seguida del nombre de la clase base. También puede implementar en su clase una o varias interfaces utilizando la palabra `implements` seguida de la lista de las interfaces implementadas. Se detallarán estas dos nociones más adelante en este capítulo.

El inicio de la declaración de la clase `Person` es, por lo tanto, el siguiente:

```
public class Person
{
}
```

Se debe introducir obligatoriamente este código en un archivo con el mismo nombre que la clase (`Person`) y la extensión `.java`.

2.2.2 Creación de los campos

A continuación, nos vamos a interesar en el contenido de la clase. Debemos crear los diferentes campos (o variables miembro) de la clase. Para ello, basta con declarar variables en el interior del bloque de código de la clase e indicar la visibilidad de la variable, su tipo y su nombre.

```
■ [private | protected | public] tipoDeLaVariable nombreDeLaVariable;
```

Es posible definir variables de clase o constantes usando las palabras clave `static` y `final`. Vaya al capítulo que habla de las bases del lenguaje para más información.

La visibilidad de la variable responde a las reglas siguientes:

`private`: la variable solo es accesible en la clase donde está declarada.

`protected`: la variable es accesible en la clase donde está declarada, en las demás clases que forman parte del mismo paquete y en las clases que heredan de la clase donde esa misma variable está declarada.

`public`: la variable es accesible desde cualquier ubicación.

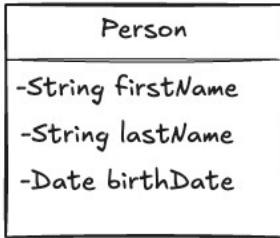
Si no se proporciona ninguna información relativa a la visibilidad, la variable es accesible desde la clase donde está declarada y desde las demás clases que forman parte del mismo paquete.

Cuando definimos la visibilidad de una variable, debemos respetar en lo posible el principio de encapsulación y limitar al máximo la visibilidad de las variables. Lo ideal sería tener siempre variables `private` o `protected`, pero nunca `public`.

La variable debe tener también un tipo. No existe ningún límite en cuanto al tipo de una variable y por lo tanto podemos utilizar tanto los **tipos primitivos** del lenguaje Java tales como `int`, `float`, `char`... como **tipos por referencia** (como `String`, `LocalDate` u objetos que usted mismo haya creado).

En cuanto al nombre de la variable, debe respetar sencillamente las reglas de nomenclatura (no utilizar palabras clave del lenguaje). Vaya al capítulo sobre las convenciones de nomenclatura (capítulo Entender un programa - Convenciones de nomenclatura) para más información.

Podemos representar la clase `Person` con sus variables miembro en UML de la siguiente manera:



La clase se compone de tres variables privadas (el `-` delante de los nombres de las variables indica este carácter privado). Las variables `firstName` y `lastName` (nombre y apellido) son de tipo `String` y la variable `birthDate` (fecha de nacimiento) es de tipo `Date`. El tipo `Date` es un tipo UML estándar. Cuando se codifica, es posible usar un tipo diferente adaptado al lenguaje utilizado.

Vemos, a continuación, que la clase `Persona` tiene la forma siguiente:

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

2.2.3 Creación de métodos

Los métodos son simplemente funciones definidas en el interior de una clase. Se suelen utilizar para manipular los campos de la clase. A continuación se describe la sintaxis general de declaración de un método.

```
[modificadores] tipoDeRetorno nombreMetodo ([listaParametros])
                                     [throws listaExcepciones]
{
}
}
```

Java cuenta con los siguientes modificadores de visibilidad:

`private`: indica que el método solo puede utilizarse en la clase donde está definido.

`protected`: indica que solo se puede utilizar el método en la clase donde está definido, en las subclases de esta clase y en las demás clases que forman parte del mismo paquete.

`public`: indica que se puede utilizar el método desde cualquier otra clase.

Si no se utiliza ninguna de estas palabras, entonces la visibilidad se limitará al paquete donde está definida la clase.

Hay disponibles modificadores adicionales.

`static`: indica que el método es un método de clase.

`abstract`: indica que el método es abstracto y que no contiene código. La clase donde está definido también debe ser abstracta.

`final`: indica que el método no puede ser sobrescrito en una subclase.

`native`: indica que el código del método se encuentra en un archivo externo escrito en otro lenguaje.

`synchronized`: indica que el método solo puede ser ejecutado por un único hilo a la vez.

El tipo de retorno puede ser cualquier tipo de dato, tipo por valor o tipo por referencia. Si el método no tiene información para devolver, deberemos usar la palabra clave `void` en sustitución del tipo de retorno.

La lista de los parámetros es idéntica a una lista de declaración de variables. Hay que especificar el tipo y el nombre del parámetro. Si se esperan varios parámetros, hay que separar sus declaraciones con una coma. Incluso si no se espera ningún parámetro, los paréntesis son obligatorios.

La palabra clave `throws` indica la lista de excepciones que este método puede lanzar durante su ejecución. La gestión de las excepciones se aborda más adelante en este capítulo.

Capítulo 3

Introducción a la plataforma Java

1. Introducción

En el corazón de los ordenadores, los microprocesadores ejecutan las listas de instrucciones. Estas listas contienen códigos binarios que representan cálculos y acciones que se tienen que hacer sobre los registros, la memoria o incluso los periféricos. Salvo en casos excepcionales, cuando un desarrollador escribe un programa, no construye directamente esta lista de instrucciones también llamada código de bajo nivel o lenguaje ensamblador. Por fortuna para él, utiliza un lenguaje de programación para describir una sucesión de acciones de «alto nivel». Por ejemplo, si el programa necesita abrir el archivo `config.txt`, entonces el desarrollador utilizará la función `fopen(«config.txt»)` de su lenguaje favorito para hacerlo. Después de esta línea, se ocultará toda una lista de instrucciones «de máquina» que el microprocesador ejecutará cuando la aplicación se ejecute. Por lo tanto, los lenguajes de alto nivel simplifican mucho la escritura, pero, al final, siempre son instrucciones de máquina las que se ejecutan.

El programa «fuente» –en nuestro ejemplo el que contiene `fopen(«config.txt»)`– es un archivo de texto que el desarrollador escribe con un editor de texto básico, como Notepad, Notepad++ o incluso el editor integrado en su entorno de desarrollo (IDE), que se presentará más adelante. La máquina no puede ejecutar este archivo de texto. Se debe convertir directa o indirectamente en lenguaje «máquina» para que sea ejecutable.

Esta translación se puede realizar de tres maneras diferentes:

- La compilación directa: el archivo se convierte en un contenido binario directamente ejecutable por el sistema operativo. Es el caso de la mayor parte de los programas escritos en C y C++ (aunque también es posible escribir programas .NET en C++... pero es otra historia).
- La interpretación: un intérprete convierte el archivo «sobre la marcha». Es el caso de los scripts PHP en las páginas web. El navegador de Internet del cliente solicita una página a un servidor web. El servidor web detecta que la página solicitada contiene un script PHP, porque su contenido es dinámico (ejemplo: lista de clientes, pedidos actuales, etc). Entonces la pasa a su analizador, que va a interpretarla y componer dinámicamente la página HTML. Este método es muy práctico en el caso de la Web porque los scripts se escriben en formato texto. En caso de modificaciones, no es necesario recompilar nada. Es suficiente con actualizar el archivo en la arborescencia del servidor y el cambio es efectivo inmediatamente. Por supuesto, el conjunto (decodificación del PHP, su ejecución y recodificación en HTML) puede ser lento.
- La compilación indirecta: es el caso de Java y de C# y es una mezcla de los dos anteriores... En efecto, el archivo fuente se compila en un formato binario intermedio, que se va a ejecutar muy rápidamente por una máquina virtual. ¿Para qué situar esta máquina virtual entre el programa y el sistema operativo? Esta máquina virtual sirve para varias cosas: en primer lugar, va a comprobar que el programa fuente no haga «tonterías» durante su ejecución, escribiendo por ejemplo en la memoria de otras aplicaciones y, de esta manera, lo que puede hacer que el sistema operativo se vuelva inestable. También se va a ocupar de la gestión de la memoria, la seguridad, los permisos, etc. La máquina virtual es la mejor amiga de los sistemas operativos y los desarrolladores. Por un lado, asegura una ejecución de máquina lo más fiable posible y, por otro, genera un código fuente (prácticamente) único para todas las plataformas.

Microsoft Windows ofrece .NET y su máquina virtual. .NET se programa con muchos lenguajes, incluso C# se diseñó para él. Sin embargo, este potente framework que inicialmente estaba muy relacionado con los sistemas operativos de Microsoft (PC, servidores, Web, tabletas, teléfonos) pero ahora es multiplataforma y open source. Este esfuerzo de apertura de Microsoft es relativamente reciente, mientras que, desde el inicio, Java se ha podido ejecutar en la mayor parte de los sistemas operativos que ofrecen una máquina virtual apropiada.

■ Observación

Por lo tanto, una plataforma Java designa un entorno de ejecución para un sistema operativo en una máquina dada y el entorno de desarrollo asociado.

2. Entorno de ejecución

Las aplicaciones escritas en Java no se comunican nunca directamente con el sistema operativo. Además, el resultado de una compilación Java (archivos con extensión *.class*) no es directamente ejecutable por el sistema operativo. Esta primera compilación, llamada *Byte Code*, contiene las instrucciones para la máquina virtual de Java, que va a «convertir sobre la marcha» el código intermedio en instrucciones compatibles con el entorno real de ejecución. Se habla de *Just In Time Compiler* (o *JIT Compiler*).

Cada entorno real de ejecución (Windows, Linux, macOS...) tiene su propia máquina virtual de Java, pero el *Byte Code* generado después de la compilación del programa fuente es el mismo y relativamente portable entre diferentes máquinas virtuales de Java. El adverbio «relativamente» se añade aquí para precisar que el diseño de un programa «portable» requiere algunas precauciones. Imaginemos que desea realizar un programa que deberá funcionar al mismo tiempo en el PC, la tableta y un teléfono. Es evidente que las características de estos tres dispositivos son totalmente diferentes y, por lo tanto, será necesario que el programa se adapte automáticamente al entorno de ejecución. Por ejemplo, podrá solicitar la resolución del dispositivo de visualización para adaptar su presentación en consecuencia.

■ Observación

La otra cara de la moneda: el resultado de la compilación se puede descompilar fácilmente, es decir, convertir el Byte Code en líneas de programa fuente Java.

Esto es un problema cuando, por ejemplo, su competencia consigue reconstruir su código fuente a partir del producto «compilado» que usted comercializa... Es posible que el resultado de esta acción sea menos legible, realizando una operación de ofuscación con herramientas especializadas, como ProGuard (open source). Sepa que el problema es el mismo para nuestros colegas que desarrollan en C#...

3. Una librería muy completa

La plataforma Java ofrece una amplia colección de clases sobre las que se basan las aplicaciones. Se habla de API Java. Estas clases simplifican considerablemente la gestión de los objetos habituales (cadenas de caracteres, valores decimales, colecciones, etc.), así como la gestión de archivos, las interfaces gráficas clásicas, los API web, el acceso a las bases de datos, las comunicaciones de red, la seguridad, los diagnósticos, etc. La lista de las clases es muy amplia y, aunque se ofrezcan de manera jerárquica, el problema normalmente es saber identificarlas.

La mayor parte de las veces, estas clases son extensibles. Esto quiere decir que es posible extenderlas para mejorar su comportamiento básico y, a continuación, añadir los aspectos específicos del negocio.

Para organizar estas clases, Java utiliza el concepto de paquetes, que agrupan las clases por objetivos (servicios de la misma naturaleza). Por ejemplo, el paquete `java.io` contiene una «caja de herramientas» para administrar los archivos.

Naturalmente, es posible crear sus propios paquetes. El paquete Java se corresponde con el paquete del UML. El desacoplamiento realizado intenta reducir las dependencias entre los paquetes. Para que un programa pueda utilizar un paquete, este último se debe referenciar en el proyecto y su utilización se debe declarar en la parte superior del archivo fuente implicado mediante la directiva `import`, por ejemplo: `import java.io.*;`

4. Las herramientas de desarrollo con buen rendimiento

La plataforma Java ofrece un compilador (`javac.exe`), que se puede utilizar directamente en línea de comandos, después de haber introducido el código del programa en un editor de texto. Por supuesto, el procedimiento es posible pero no muy productivo. Evidentemente, el desarrollador persigue la utilización de un entorno de desarrollo totalmente integrado, que le ayude durante la redacción del código, la realización de la interfaz gráfica, la puesta a punto y el despliegue de su aplicación. Este programa se denomina IDE (*Integrated Development Environment*) o Entorno de Desarrollo Integrado. Como mínimo, integra un editor de código fuente, las herramientas que automatizan las compilaciones y un depurador.

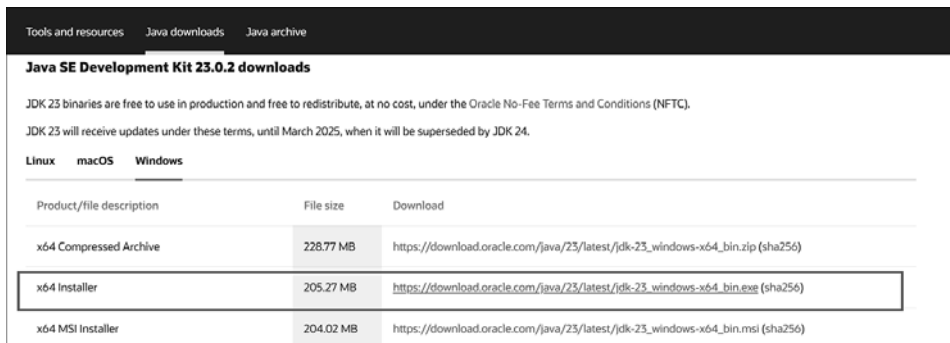
Existen varios IDE para Java. Entre ellos hay tres principales: IntelliJ IDEA, de JetBrains; Eclipse, de la fundación Eclipse, y NetBeans, creado por iniciativa de Sun Microsystems y actualmente mantenido y distribuido por Oracle y por Apache Software Foundation para su versión Apache NetBeans. Las demostraciones y ejercicios de este libro se basan en la utilización de IntelliJ IDEA porque es el IDE que nos ha parecido más sencillo para empezar a desarrollar.

5. Descarga e instalación de JDK

■ Para desarrollar en Java necesitaremos un «kit de desarrollo» llamado JDK al que volveremos muy pronto. Por ahora, vaya al sitio web de Oracle en <https://www.oracle.com/java/technologies/downloads/#jdk22-windows> para descargar la versión de JDK 23 con su instalador.

Observación

Esta versión 23 puede que no sea la última cuando lea estas líneas.



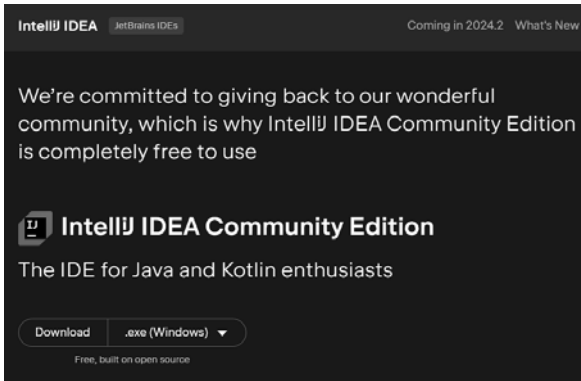
Product/file description	File size	Download
x64 Compressed Archive	228.77 MB	https://download.oracle.com/java/23/latest/jdk-23_windows-x64_bin.zip (sha256)
x64 Installer	205.27 MB	https://download.oracle.com/java/23/latest/jdk-23_windows-x64_bin.exe (sha256)
x64 MSI Installer	204.02 MB	https://download.oracle.com/java/23/latest/jdk-23_windows-x64_bin.msi (sha256)

- ▶ Ejecute el programa descargado: `jdk-23_windows-x64_bin.exe` conservando todas las opciones por defeto, propuestas en los diferentes cuadros de diálogo.

6. Descarga e instalación de IntelliJ IDEA

- ▶ Vaya ahora al sitio web oficial JETBRAINS (<https://www.jetbrains.com/idea/download/?section=windows>) para descargar la versión Community Edition de este IDE.

▶ Avance por la pantalla hasta mostrar lo siguiente:



▶ Haga clic en el botón **Download**.

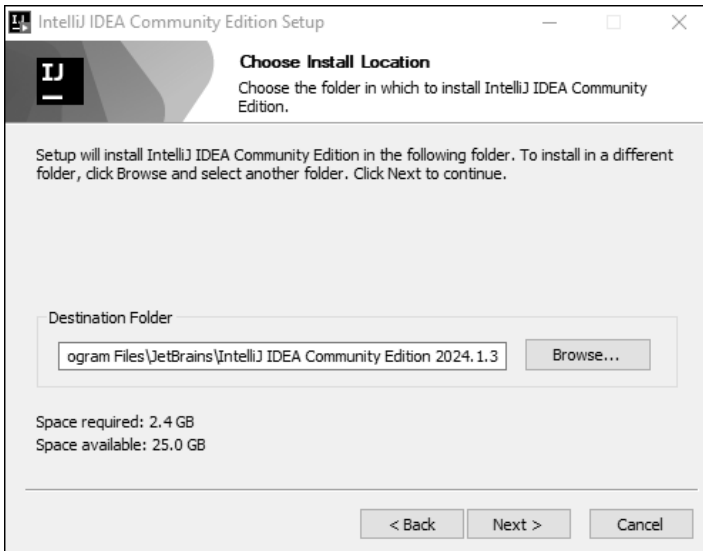


▶ Cuando haya terminado la descarga, haga doble clic en el archivo descargado (ideaIC-2024.1.3.exe en el momento en el que se escriben estas líneas).

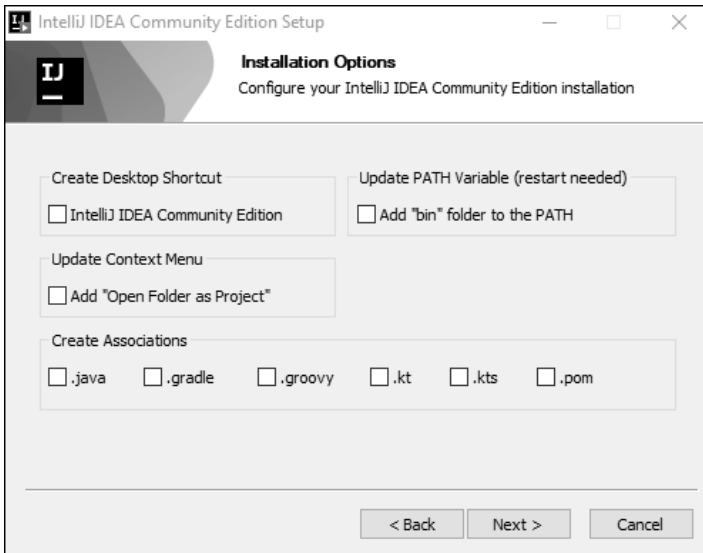
▶ Permita a la aplicación que añada modificaciones.



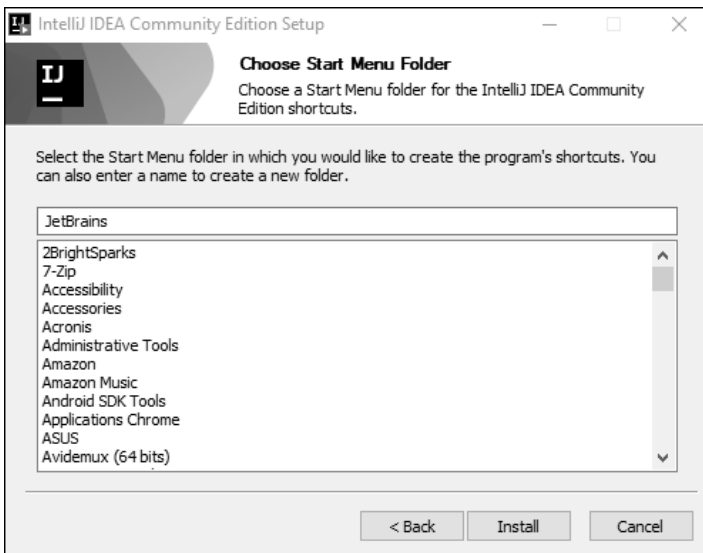
► Pulse en el botón **Next**.



► Conserve los valores propuestos por defecto y pulse el botón **Next**.



► Conserve los valores propuestos por defecto y pulse el botón **Next**.



► Conserve los valores propuestos por defecto y pulse el botón **Install**.