**INTRODUCCIÓN A LA PROGRAMACIÓN**

MANUAL DEL LENGUAJE JAVA

Autor: Hugo Pelayo Aseko

Fecha: 1 de abril de 2023

Asignatura: Programación

Este manual cubre aspectos básicos y generales relacionados con la programación, con un enfoque alrededor del lenguaje de programación Java. Se empieza por introducir la programación en general abstrayendo de los detalles de los lenguajes de programación.

Al final del manual se agrega un apéndice donde se pueden consultar las definiciones de términos relevantes al manual que sean de difícil comprensión.

Índice

[Introducción 4](#_Toc133089667)

[Desarrollo de software 5](#_Toc133089668)

[El programa 6](#_Toc133089669)

[Programación en Java 8](#_Toc133089670)

[Utilidades necesarias para empezar a programar en Java 10](#_Toc133089671)

[Configuración de entorno de desarrollo para Java en Visual Studio Code 10](#_Toc133089672)

[IntelliJ IDEA para la programación en Java 13](#_Toc133089673)

[Estructura de un programa 14](#_Toc133089674)

[Elementos básicos de Java 14](#_Toc133089675)

[Entrada y salida de datos 14](#_Toc133089676)

[Salida de datos 15](#_Toc133089677)

[Entrada de datos 18](#_Toc133089678)

[Mediante del BufferedReader 18](#_Toc133089679)

[Mediante Scanner 20](#_Toc133089680)

[Estructuras de control 22](#_Toc133089681)

[Arrays 22](#_Toc133089682)

[Apéndice 25](#_Toc133089683)

[Bibliografía 27](#_Toc133089684)

# Introducción

Se entiende por software a aquel sistema dotado de componentes lógicos que posibilitan la realización de tareas bien definidas, constituye entonces el conjunto de instrucciones que debe ser ejecutado por el **hardware**.

El software por lo general se desarrolla utilizando lenguajes de programación de alto nivel. Son lenguajes que utilizan una estructura semántica más próxima a los lenguajes humanos y, por tanto, más fáciles de entender para los programadores. Debido a que el hardware entiende únicamente el lenguaje máquina, que está formado exclusivamente por cadenas de unos y ceros, el software escrito en lenguaje de alto nivel debe ser compilado primero antes de ser ejecutado por nuestro hardware, interpretado o, en el caso de algunos lenguajes de alto nivel, ambos a la vez.

Aunque sea poco común, para el desarrollo de software se utiliza también el lenguaje ensamblador, que es un lenguaje que constituye una capa inmediatamente superior al lenguaje máquina. Una de las diferencias principales entre estos dos lenguajes (que también motiva su uso) y los de alto nivel se basa en que estos últimos abstraen al programador de la arquitectura para la cual se desarrolla la aplicación, de tal modo que podemos desarrollar aplicaciones en lenguajes de alto nivel sin tener mucho conocimiento sobre la arquitectura destino. Sin embargo, este no es el caso para el lenguaje ensamblador que al igual que el lenguaje máquina, va de la mano con la arquitectura.

El software por lo general se clasifica en tres tipos acorde a las funciones las cuales este ha de realizar, tenemos entonces el software de sistema, software de programación y el software de aplicación.

El software de sistema está destinado a administrar el sistema informático, esto es, proveer al programador de una interfaz de alto nivel o conjunto de herramientas para su diagnóstico y mantenimiento aislando al mismo tiempo al programador de los detalles internos del sistema.

El software de programación es el conjunto de aplicaciones que permiten al programador desarrollar programas informáticos, es decir, desarrollar software, en este grupo se incluyen los editores de texto, los compiladores, entornos de desarrollo integrados (IDE), entre otros. Este software dota al programador de herramientas prácticas para el desarrollo de aplicaciones permitiendo el uso de lenguajes de programación tanto de alto nivel como de bajo nivel mencionados con anterioridad.

Por último, tenemos el software de aplicación. Se trata de un conjunto de software destino a un usuario final para la realización de tareas específicas que pueden ser automatizadas o realizadas mediante la asistencia del software. Se incluye dentro de este grupo las aplicaciones ofimáticas como el Microsoft Word, aplicaciones para la manipulación de bases de datos como MySQL Workbench, entre otros.

Para el desarrollo de software se sigue normalmente un conjunto de pasos para llegar al producto final. Es un proceso que llega a ser muy complejo y durar mucho tiempo dependiendo de la complejidad del producto, de hecho, una aplicación mala de esta metodología puede llevar el desarrollo de un producto al fracaso o imponer retrasos prolongados sobre este.

## Desarrollo de software

El desarrollo de software acostumbra a involucrar varias fases: análisis de requisitos, diseño, implementación o codificación, pruebas unitarias y de integración, implantación del producto en el entorno de uso y finalmente el mantenimiento, etapa que acostumbra a ser más longeva.

La fase de análisis de requisitos es la primera que se realiza siempre en un proyecto de desarrollo de software (en realidad esto se aplica a cualquier proyecto). En esta fase se especifican las características funcionales (qué debe hacer nuestro software) y las no funcionales (requisitos de nuestro sistema, problemas de escalabilidad, confiabilidad, entre otros). En esta fase normalmente participa un analista junto a varios programadores para analizar los detalles del producto que se desea obtener. Acostumbra a ser una fase muy difícil ya que sienta las bases del proyecto, una fase de análisis cadente puede dificultar el avance del proyecto.

En el diseño es una fase posterior a la fase de análisis donde se define cómo se pretende cumplir con los requisitos especificados en la fase de análisis. Conviene entonces, identificar una variedad de soluciones posibles para el problema que se está intentando solucionar, evaluarlas y determinar de estas soluciones cuál es la que más nos conviene. Habiendo optado por una solución, se procede a optar por las herramientas de desarrollo necesarias para el proyecto.

En la implementación se realizan las tareas de programación, en esencia pasamos a implementar los esquemas resultantes de la fase de diseño. En esta fase participan mayoritariamente los programadores.

El período de pruebas se puede separar en dos partes, pruebas unitarias o pruebas de integración. Esta fase básicamente se encarga de poner a prueba el producto y determinar si cumple con las especificaciones establecidas anteriormente. En el caso de pruebas unitarias, probamos las piezas pequeñas de nuestro software que pueden ser procedimientos, módulos, funciones, clases, entre otros; las pruebas de integración se realizan una vez se han realizado de manera exitosa las pruebas unitarias, en este caso nos aseguramos que el sistema completo funciona correctamente.

Finalmente pasamos a la producción, donde el producto es transferido al entorno de uso del usuario final (o el cliente que lo haya solicitado). Después de esta fase tenemos el mantenimiento que es un proceso de control, mejora y optimización de nuestro producto que ya se encuentra en los equipos de los usuarios finales. Este período acostumbra a ser el más longevo del proceso de desarrollo de software. Durante esta fase, se pueden realizar revisiones e incluso mejoras sobre el producto si hubiese la necesidad.

## El programa

Como ya se había mencionado anteriormente todo software consta de programas. Existe cierta confusión entre programa y el software en sí, un programa es una pieza del software. Se considera programa a una secuencia de instrucciones u órdenes escritas en un lenguaje de programación cualquiera que han de ser interpretadas por la computadora para realizar una tarea específica, esta secuencia de pasos ha de ser finita, bien definida y concisa, se conocen generalmente con el nombre de algoritmo.

El conjunto de expresiones que componen un programa se conoce como el código del programa. Este se escribe en lenguajes de programación acorde a un paradigma de programación que se adapte a las necesidades del programador. Los paradigmas de programación más comunes se clasifican en dos grandes grupos: el imperativo y el declarativo. En el paradigma imperativo el programador instruye a la máquina a través del código a cómo cambiar su estado; en el paradigma declarativo se instruye a la máquina sobre las propiedades del resultado esperado de ciertas operaciones y no a cómo realizarlas.

Entre las herramientas más comunes para el diseño de algoritmos destacan los diagramas de flujo y el pseudocódigo:

El pseudocódigo, también conocido con el nombre de lenguaje algorítmico, es una forma de describir con lenguaje natural el flujo de ejecución de un algoritmo o programa de forma compacta. Normalmente utiliza las construcciones de un lenguaje de programación real, pero está diseñado para la interpretación por parte de humanos ya que abstrae las particularidades de los lenguajes de programación facilitando el enfoque en la lógica y funcionamiento del programa. Por lo general se utiliza el pseudocódigo en libros, textos científicos que exponen algoritmos y también es usado para la planificación de algoritmos para programas complejos. Esto permite, por ejemplo, descubrir posibles errores de lógica en nuestros algoritmos antes de proceder a su implementación. Como se ha mencionado previamente, el pseudocódigo sigue una sintaxis arbitraria y no existe cierta convención ya que el objetivo principal es simplemente exponer la solución a un problema de la forma más sencilla posible. Sin embargo, cabe recalcar que ciertos IDE como PSeInt utilizan pseudocódigo con sintaxis propia para el desarrollo de algoritmos.

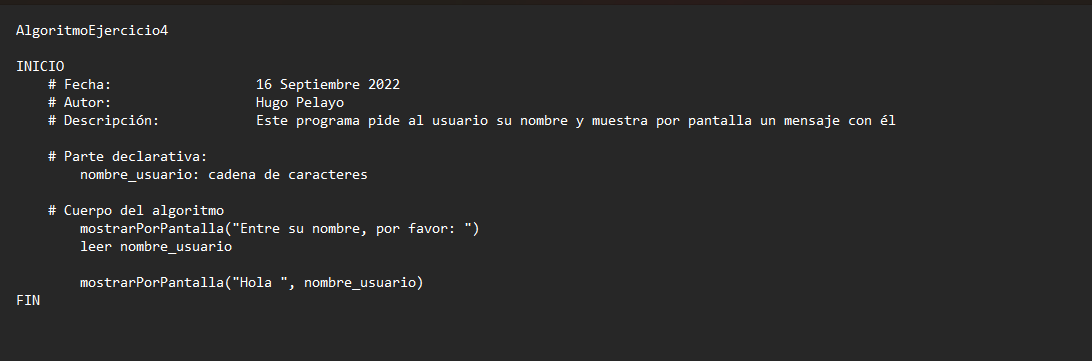


Figura 1. Ejemplo de pseudocódigo. Fuente: creación propia

El cuerpo del pseudocódigo está constituido normalmente por estructuras de control y estructuras e iterativas o bucles y declaraciones de variables, entre otros aspectos. Las estructuras de control se dividen principalmente en dos grupos, estructuras secuenciales donde tenemos un flujo de ejecución de sentencias continua o, pueden ser selectivas, en cuyo caso el flujo de ejecución de las sentencias varía acorde al resultado de evaluar ciertas expresiones lógicas.

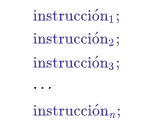


Figura 2 Estructura secuencial. Fuente: Wikipedia

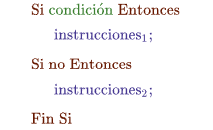


Figura 3 Estructura selectiva con alternativa. Fuente: Wikipedia

El otro grupo de grupo elementos está constituido por las estructuras iterativas, en las cuales se ejecuta un cierto grupo de instrucciones mientras se cumpla cierta condición, tenemos entonces: el **bucle mientras** donde ejecutamos su cuerpo mientras una condición se satisfaga; tenemos el **bucle hacer** donde, a diferencia del bucle anterior, se ejecuta el cuerpo al menos una vez, cabe destacar que en el caso del bucle anterior si la condición especificada en la cabecera del bucle no se cumple ni en la primera iteración entonces nunca se entra al cuerpo de este; para finalizar tenemos el **bucle repetir** donde ejecutamos un grupo de sentencias un número finito de veces.

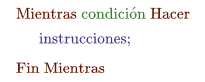


Figura 4. Formato del bucle mientras. Fuente: Wikipedia

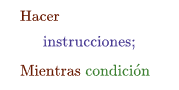


Figura 5. Formato del bucle hacer. Fuente: Wikipedia

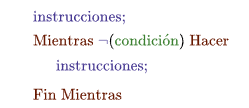


Figura 6. Formato del bucle repetir. Fuente: Wikipedia

# Programación en Java

Java es un lenguaje de programación muy utilizado en el desarrollo de aplicaciones web, es multiplataforma y soporta principalmente el paradigma de la programación orientada objetos. Se utiliza también para el desarrollo de aplicaciones móviles y de escritorio. Java en sí es considerado una plataforma por ser un sistema que sirve como base para hacer funcionar ciertos módulos de hardware y software con los que es compatible. Fue comercializado por primera vez en 1995 por Sun Microsystems y desarrollado originalmente por James Gosling. Su sintaxis deriva en gran parte de C y C++, pero ofrece menos soporte para utilidades de bajo nivel que ambos. Las aplicaciones en Java se compilan a bytecode que puede interpretar cualquier máquina virtual de Java indistintamente de la arquitectura del computador en cuestión.

Sun definió por primera vez la implementación de referencia original para los compiladores de Java, máquinas virtuales y librerías, trabajo que se publicó en 1995.

Originalmente Java se creó para utilizar en un proyecto sobre un decodificador de señales de televisión en la operación The Green Project en el año 1991. En un principio se llamó Oak, se pasó a llamar Green tras darse cuenta de que Oak ya existía y pertenecía a una marca comercial de adaptadores de tarjetas gráficas, entonces se acabó llamando Java. Hay muchas teorías sobre la elección de este último nombre, sin embargo, destaca más el hecho de que el nombre tiene origen en un tipo de café de una cafetería que frecuentaban sus desarrolladores. El objetivo en un principio era diseñar un lenguaje de programación con una estructura similar a la de C++, aún así en 1994, sus desarrolladores acabaron orientándolo al entorno Web, porque creyeron que el navegador web Mosaic haría del internet un medio más interactivo, concepto que se asimilaba mucho a la televisión por cable, idea inicial del proyecto de Java. Naughton, uno de los desarrolladores de Java, creó entonces un prototipo de navegador, WebRunner, que más tarde se conocería con el nombre de HotJava.

La promesa inicial de Gosling era “Write once, Run Anywhere” (Escríbelo una vez, ejecútalo en cualquier plataforma), un concepto que promovía la transportabilidad, es decir, la capacidad de ejecutar los mismos códigos en cualquier plataforma que ofreciese soporte para el entorno de ejecución de Java, pudiendo compartir de esta forma los mismos binarios si tener que reescribir el código fuente.

Desde la versión 1.0 del JDK (Java Development Kit), lanzada en enero de 1996, Java ha experimentado números cambios sobre todo en las clases que ofrece su librería estándar. En diciembre de 1998 se lanzó el JDK 1.2 bajo los nombres de Java 2 y J2SE (Java 2 Platform, Standard Edition), esto con el objetivo de distinguir la plataforma base de las ediciones J2EE (Java 2 Platform, Enterprise Edition) y J2ME (Java 2 Platform, Micro Edition).

Entre las características de Java podemos destacar que es fuertemente orientado a objetos, en efecto, el programa principal es un objeto que viene encapsulado en forma de clase. Aquí los objetos son generalmente clases que constan de atributos y los métodos que nos sirven para poder operar sobre estos atributos. Otro aspecto importante a destacar en este aspecto es el soporte para objetos genéricos que se pueden reaprovechar para otras aplicaciones, cosa que agiliza el desarrollo de estas.

Java ofrece independencia de plataforma, por tanto, programas escritos en este lenguaje se pueden ejecutar en cualquier plataforma indistintamente del hardware, siempre y cuando haya soporte para la máquina virtual de Java en la plataforma destino. Para ello los compiladores de Java, traducen los archivos de código fuente de Java a binarios con extensión **.class**, los cuáles contienen bytecode, un tipo de instrucciones interpretables por la máquina virtual de Java.

A diferencia de C++, lenguaje del cuál Java hereda muchos aspectos sobre el paradigma de la programación orientada a objetos, Java proporciona un recolector de basura para solucionar el problema de las fugas de memoria, error muy común entre programadores de C++. Las fugas de memoria suceden cuando alojamos memoria de forma dinámica (en tiempo de ejecución) y no la desalojamos acabado su uso, esto se convierte entonces en memoria ocupada pero no aprovechable. El *garbage collector* (recolector de basura) de Java se ocupa de este problema.

## Utilidades necesarias para empezar a programar en Java

Para empezar a desarrollar aplicaciones en Java lo primero que debemos hacer es decidirnos por una edición, la que más se ajuste a nuestras necesidades (mencionadas con anterioridad). Tenemos entonces a elegir: J2ME, J2SE o J2EE. Esta última se reserva para aplicaciones web más complejas con accesos a bases de datos, entre otros aspectos. Para aplicaciones menos complejas y más o menos completas que se puedan ejecutar en PC, la edición J2SE sería suficiente.

Ya escogido J2SE, debemos descargarnos el SDK (Software Development Kit) de Java que incluye herramientas para desarrollo de aplicaciones Java como el compilador, el depurador, entre otras.

## Configuración de entorno de desarrollo para Java en Visual Studio Code

Para configurar un entorno de desarrollo para Java en el editor de código Visual Studio Code deberemos primero instalar el editor en cuestión, seguido de un JDK y finalmente las extensiones que el editor ofrece para la programación con Java. Nos dirigimos entonces al enlace de descarga de VS Code: [Visual Studio Code](https://code.visualstudio.com/docs/java/java-tutorial).

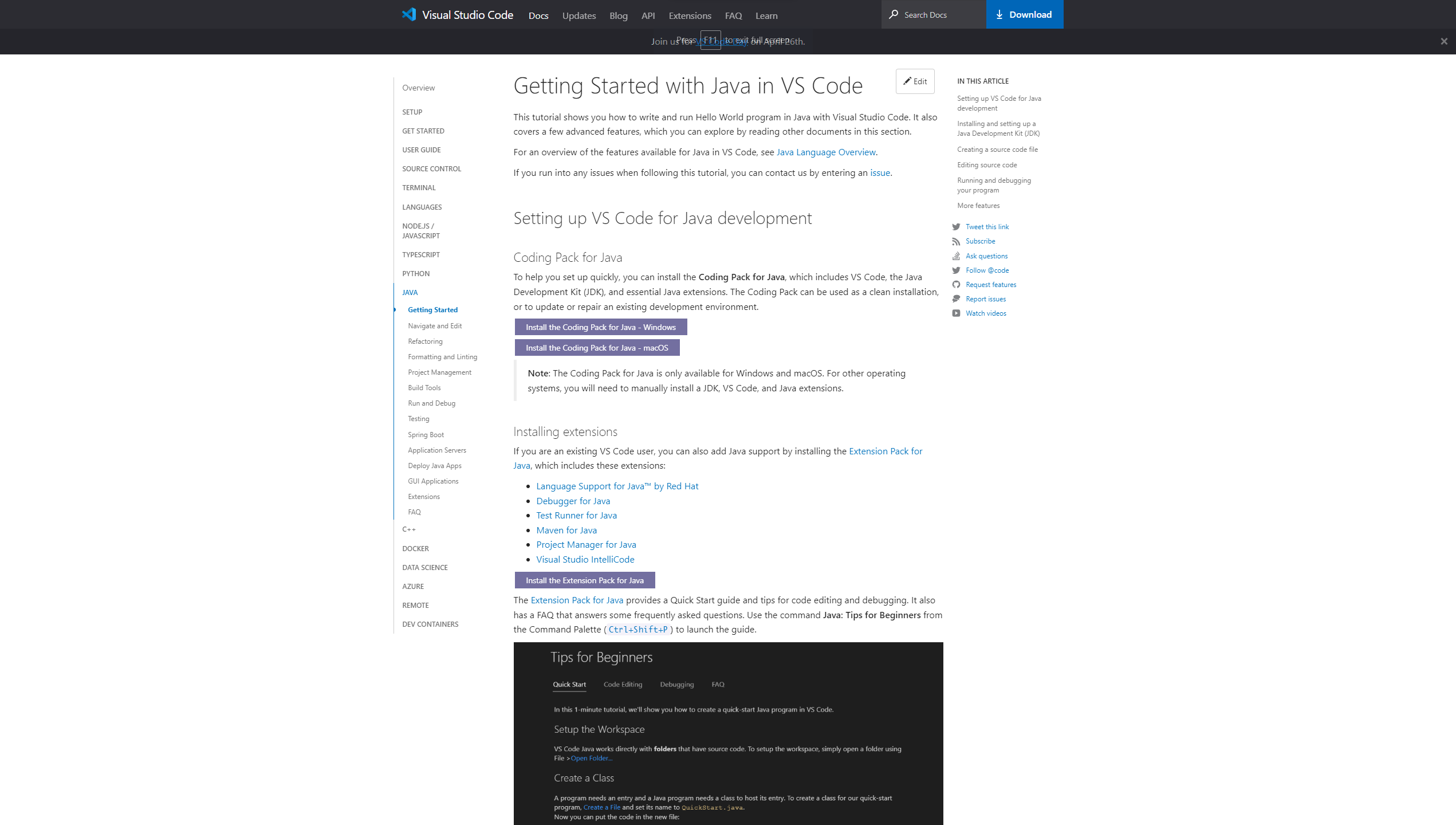


Figura 7. Tutorial Java VS Code. Fuente: Elaboración propia

Debemos asegurarnos primero de que tenemos el JDK instalado, para ello abrimos una terminal (Tecla Windows y escribimos “cmd”), ahora escribimos el siguiente comando sin las comillas: java --version, nos debería salir un mensaje como el siguiente:

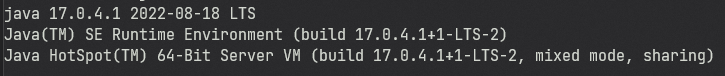


Figura 8. Captura terminal versión de JDK. Fuente: Elaboración propia

Si no nos sale el mensaje, muy probablemente no tengamos el JDK instalado. Podemos conseguir uno de forma gratuita desde la página mencionada anteriormente. Deslizando un poco hacia abajo, encontraremos un listado de JDKs que podemos descargar, hacemos clic sobre el enlace que está señalado con la marca verde en la imagen de abajo, nos redirigirá a la página a través de la cuál podremos descargar e instalar nuestro JDK.



Figura 9. Listados JDK para VSCode. Fuente: Elaboración propia

Una vez seguros de que tenemos nuestro JDK instalado, tendremos dos vías de instalación para el editor y las herramientas necesarias para desarrollar aplicaciones con Java: podemos descargarnos el ejecutable *Coding Pack for Java* (disponible sólo para Windows y MacOS), a través de los botones en magenta: *Install the Coding Pack for Java - Windows* y *Install the Coding Pack for Java - macOS*, y proceder con la instalación mediante esta aplicación; o podemos realizarla manualmente siguiendo los siguientes pasos:

Hacemos clic sobre el botón azul en la esquina superior derecha de la pantalla que dice “Download” y descargamos el instalador de Visual Studio Code:

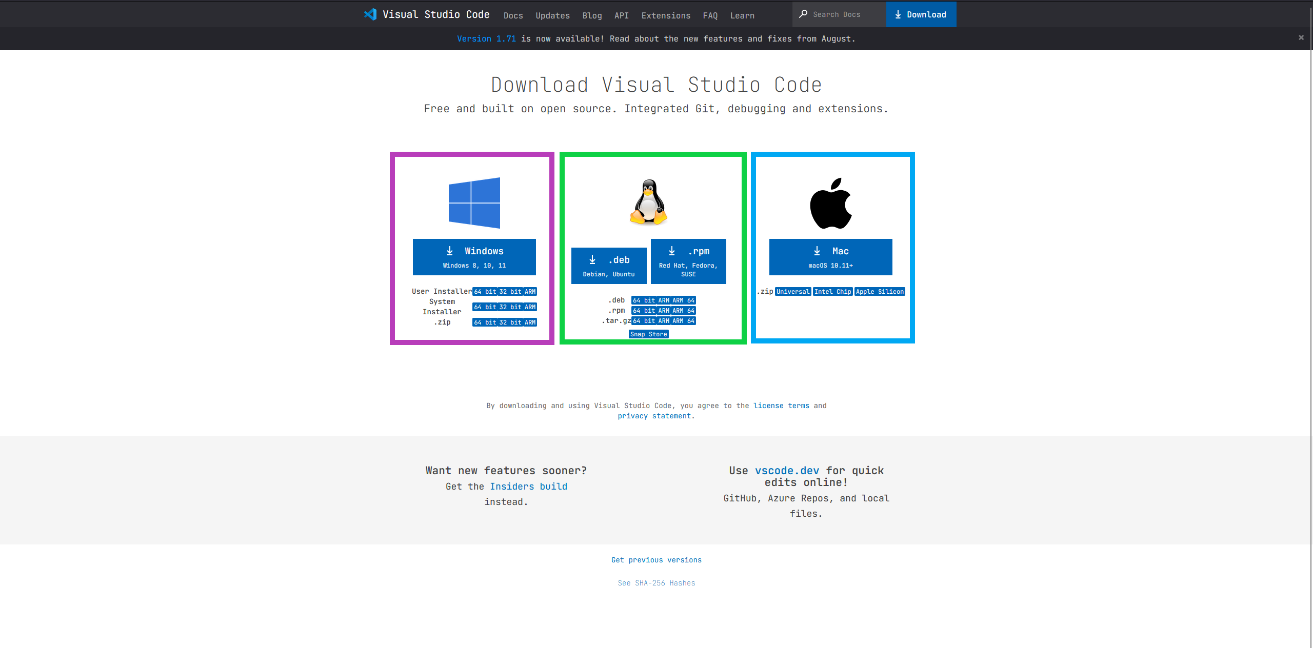


Figura 10. Plataformas para VSCode. Fuente: Elaboración propia

Aquí escogemos el instalador dependiendo de nuestro sistema operativo: el primero es para sistemas operativos Windows, el segundo para sistemas operativos Linux y el tercero para Mac, hacemos clic sobre los recuadros en azul grandes para descargar nuestro instalador e instalamos Visual Studio Code.

Una vez instalado Visual Studio Code, volvemos a nuestra página [VS Code Tutorial](https://code.visualstudio.com/docs/java/java-tutorial), e instalamos las extensiones de Java para Visual Studio Code, para ello hacemos clic sobre e botón de color magenta que dice “*Install the Extension Pack for Java*”.



Figura 11. Extensiones de Java para VSCode. Fuente: Elaboración propia

Nos abrirá una ventana de Visual Studio Code que contiene información similar a la siguiente:

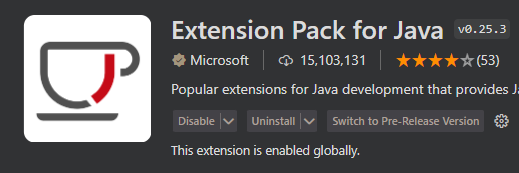


Figura 12. Paquete de extensiones de Java para VSCode. Fuente: Elaboración propia

Cuando se realice una instalación por primera vez, nos saldrá un botón diciendo “**Instalar**” en esta ventana, hacemos clic sobre él y esperamos a que se acaben de instalar las extensiones de Java.

Finalizado el paso anterior ya estaremos listos para crear nuestro programa en Java y ejecutarlo.

## IntelliJ IDEA para la programación en Java

Para el desarrollo de aplicaciones Java, se recomienda IDEs como IntelliJ IDEA, NetBeans o Eclipse. IntelliJ IDEA es utilizado incluso por Google como la estructura base para Android Studio, IDE de desarrollo de aplicaciones para Android. IntelliJ IDEA ofrece una versión gratuita mantenida por cierta comunidad de usuarios del producto que se llama IntelliJ IDEA Community, y tenemos la versión de pago que es IntelliJ IDEA Ultimate que ofrece varias tarifas dependiendo del uso que le vayamos a dar y que tipo de entidad formen los usuarios, ya sea uso personal, empresa, etcétera. Estas dos versiones se pueden obtener de su [página oficial](https://www.jetbrains.com/es-es/idea/download/#section=windows). Abajo se recoge un listado de diferencias principales entre ambas versiones:

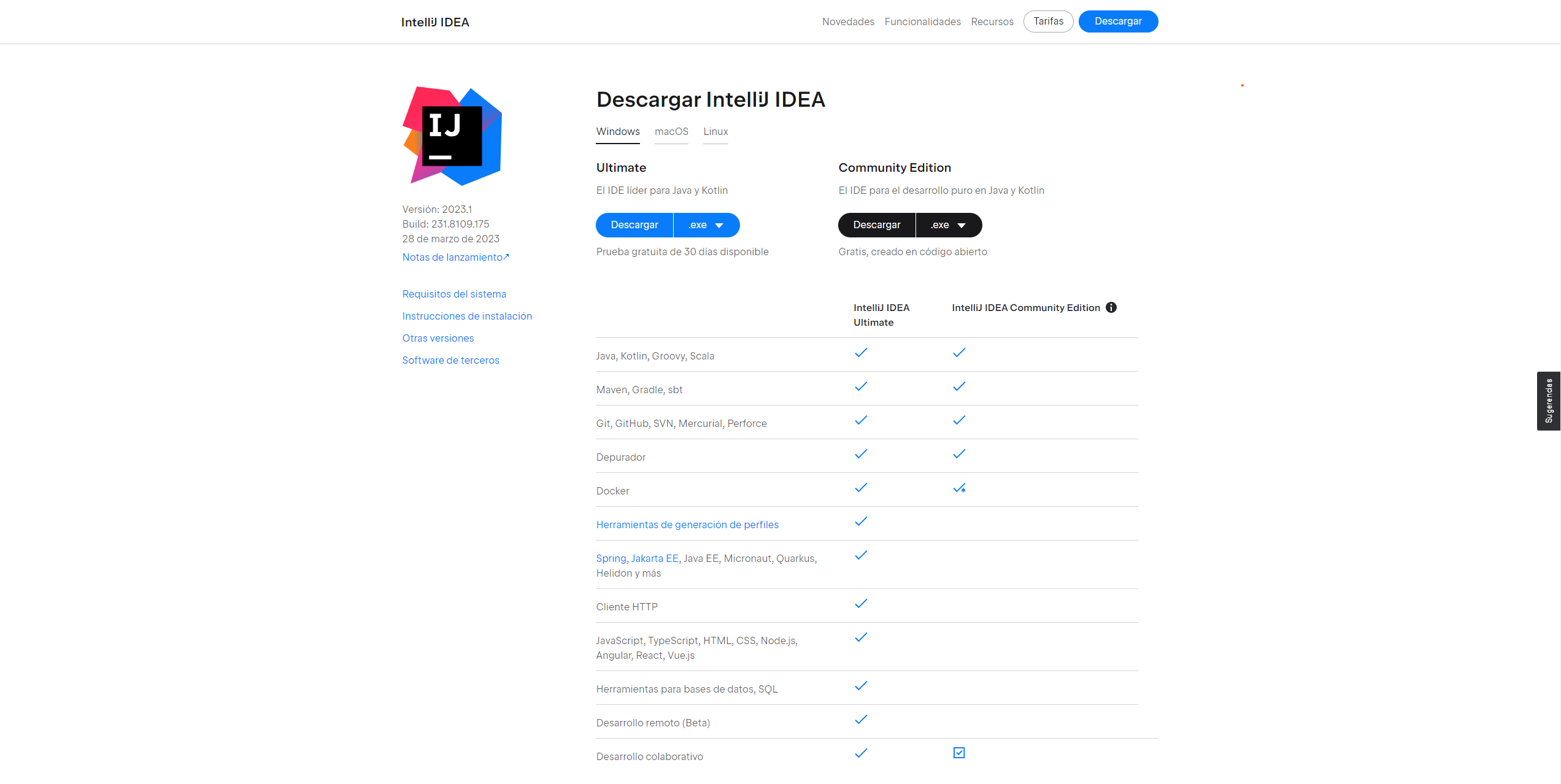


Figura 13. Comparativa Ultimate y Community. Fuente: IntelliJ IDEA sitio web oficial

# Estructura de un programa

Java es un lenguaje que sigue estrictamente el paradigma de la programación orientado a objetos, en el por tanto, todo programa consta de una colección de clases. La estructura de un programa en Java es muy similar a la de un programa en C++ o C. Como se explicó previamente, las clases son agrupaciones de datos sobre los cuales se puede operar a través de un conjunto de métodos. Todo programa Java debe contener un sólo método main() entre todas las clases que lo forma, aunque pueda estar constituido de varias clases. En el método principal main() se inicia la ejecución de nuestro programa.

La estructura estándar de los ficheros de nuestro programa Java siempre empieza por indicar el paquete de la clase que define dicho fichero. A continuación, se importan todas las clases que vaya a utilizar la clase que está definiendo el fichero. Para importar clases de otros paquetes o proyectos java utilizamos la directiva import. Y finalmente, tenemos la definición de nuestra clase.

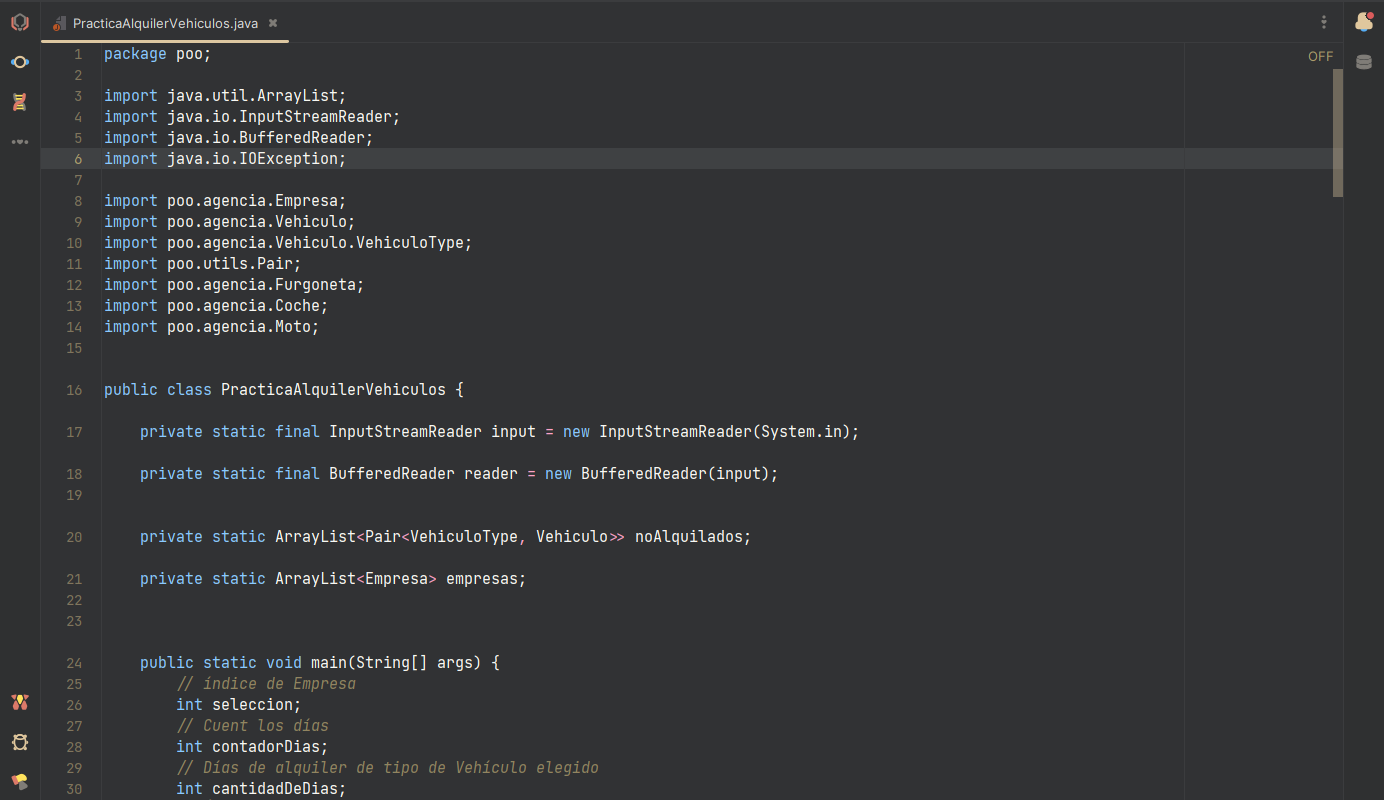


Figura 14. Ejemplo estructura programa en Java. Fuente: elaboración propia

# Elementos básicos de Java

# Entrada y salida de datos

Una de las operaciones más habituales que tiene que realizar un programa en Java es intercambiar datos con otros sistemas o con el mismo en el cual se está ejecutando. Para este fin, el SE de Java ofrece el paquete java.io que contiene una serie de clases que nos permiten realizar operaciones de lectura y escritura sobre todo tipo de dispositivos abstrayendo al programador de estos mediante el uso de una interfaz común para ellos. Las principales clases para la entrada de datos son *InputStreamReader* y *BufferedReader*, ambas especializaciones de *InputStream*; y para la salida estándar de datos tenemos la clase PrintStream de la cual se especializan muchas otras, también tenemos *BufferedWritter*, *InputStreadWritter*, entre otros.

La salida estándar está asociada a la de un proceso en cuestión, es decir, la salida de datos por defecto del proceso sobre el que se ejecuta nuestro programa, en Linux, por ejemplo, la salida estándar de todos los procesos se asocia al terminal, sin embargo, este comportamiento se puede cambiar redireccionando la salida del proceso en cuestión a dispositivos como sockets, ficheros, etcétera.

## Salida de datos

Para enviar datos por la salida estándar debemos crear uno objeto de tipo *PrintStream*, la clase *System* ya ofrece un atributo llamado *out* para mostrar datos por la salida por defecto. Para el envío de datos esta clase nos ofrece los métodos print(String) o println(String) que envían cadenas de caracteres por la salida de datos, la principal diferencia entre ambos es que el segundo añade un salto de línea al final (carácter de control '\n') tras haber enviado el contenido que recibe como parámetro. Cabe destacar que estos métodos ofrecen varias sobrecargas que reciben muchos otros tipos de parámetros, a parte la clase madre de todas, *Object*, ofrece el método toString() el cual puede ser redefinido por todas las clases de Java para ofrecer una mejor representación de sí.

Métodos de la clase PrintStream:

|  |  |
| --- | --- |
| **Firma** | **Descripción** |
| PrintStream append(...) | Concatena un carácter o secuencia de caracteres a este PrintStream |
| boolean checkError() | Comprueba el estado de este PrintStream |
| void clearError() | Restaura el estado de errores de este PrintStream |
| void close() | Cierra el flujo de datos de este PrintStream |
| void flush() | Limpia el buffer de datos de este PrintStream una vez que los caracteres se han enviado por la salida de datos |
| PrintStream format(...) | Formatea los datos que se pasa como parámetro antes de enviarlos por la salida de datos |
| void print(...) | Envía la cadena de caracteres que se pasa como parámetro por la salida de datos |
| void printf(...) | Envía la cadena de caracteres que se pasa como parámetro por la salida de datos y permite formatear la cadena |
| void println(...) | Envía la cadena de caracteres que se pasa como parámetro por la salida de datos y al final añade un salto de línea |
| void setError(...) | Activa el estado de error de este PrintStream |
| void write(...) | Escribe bytes por la salida de datos |

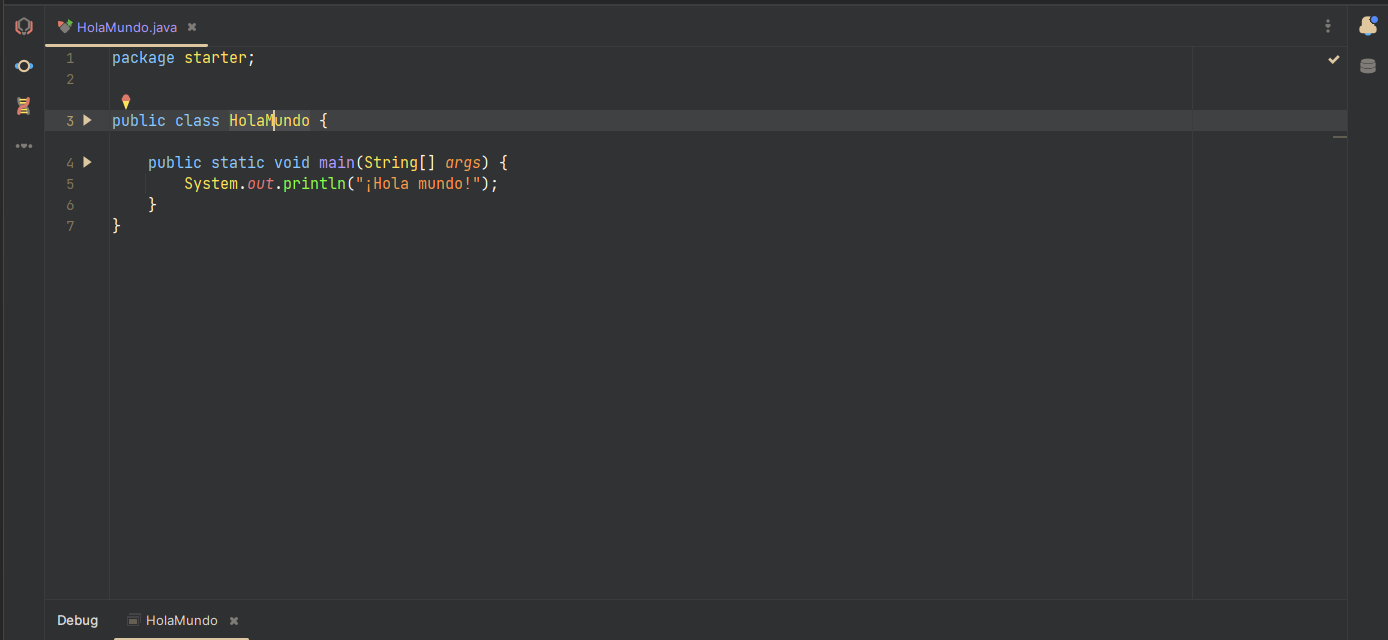


Figura 15. Ejemplo de salida de mensaje. Fuente: elaboración propia

La clase *PrintStream* también ofrece los métodos printf() y format() desde la versión 5 de Java, estos nos permiten formatear nuestra cadena de caracteres antes de mostrarla por la salida de datos. La cadena que nosotros enviamos a estos métodos está formada por nuestro texto acompañado de una serie de especificadores de formateo en aquellas posiciones donde queramos un formato de muestreo específico, la sintaxis para los especificadores de formateo es estricta y la mínima equivocación en esta puede llevar a Java interpretarlo como texto normal o lanzar una excepción.

EL formato para los especificadores:

% [posición\_argumento$] [indicador] [mínimo] [.num\_decimales] conversión

El signo de porcentaje al principio indica que el contenido a continuación representa un especificador de formato, si queremos el signo de porcentaje en sí en nuestra salida utilizamos entonces el "%%". posicion\_argumento es un valor entero positivo que indica cuál de los datos que pasamos como parámetro a nuestro método se asocia al especificador en cuestión, al igual que el resto de especificadores que están entre corchetes, este es opcional. El indicador especifica un formato de salida. mínimo representa un número mínimo de caracteres a ser representados. num\_decimales es un entero positivo que indica el número de decimales con que mostrar un valor en coma flotante, importante destacar que este valor debe venir precedido de un punto (ejemplo %.3f). Para finalizar tenemos la conversión que indica el tipo de argumento (String, char, double, int, etcétera). A continuación, se recogen los especificadores de conversión más comunes:

|  |  |  |
| --- | --- | --- |
| **Especificador** | **Categoría** | **Descripción** |
| ‘h’ o ‘H’ | general | Muestra valor retornado por  Integer.toHexString(arg.hashCode())  Donde “arg” es nuestro dato |
| ‘s’ o ‘S’ | general | Muestra valor de un String |
| ‘c’ o ‘C’ | carácter | Muestra el resultado de un carácter en Unicode |
| ‘x’ o ‘X’ | entero | Muestra valor entero en hexadecimal |
| ‘e’ o ‘E’ | coma flotante | Decimal en notación científica |
| ‘g’ o ‘G’ | coma flotante | Decimal en notación científica con precisión específica |
| ‘a’ o ‘A’ | coma flotante | Decimal en notación científica con un dígito significante y el exponente correspondiente |
| ‘d’ | entero | Muestra el valor de un entero en decimal |
| ‘f’ | coma flotante | Muestra un valor en coma flotante con |
| ‘o’ | entero | Muestra el valor de un entero en octal |



Figura 16. Ejemplo de formateo con especificadores. Fuente: elaboración propia

El ejemplo anterior muestra el resultado siguiente:



Figura 17. Resultado ejemplo Figura 14. Fuente: elaboración ropia

A parte de la salida estándar de datos existe la salida estándar de error de un proceso, que tenemos disponible a través del atributo *err* de la clase *System*. Al igual que el atributo *out*, *err* es un objeto de tipo *PrintStream*. La ventaja de esta diferenciación es que podemos redireccionar la salida estándar de un proceso y la salida estándar de error a ficheros diferentes, por ejemplo, podríamos tener nuestra salida estándar vinculada a la terminal o un fichero de texto con nombre "salida.txt" y en nuestra salida estándar de error a otro fichero con nombre "errores.log" o a la misma terminal. En algunos IDEs como NetBeans, la salida estándar de error se suele mostrar con un color rojo llamativo ya que se va más alarmante, sin embargo, la salida estándar suele estar en un color negro.

Desde java podemos cambiar la salida estándar de nuestro programa o su salida estándar de error utilizando los métodos setOut() o setErr() que nos ofrece la clase *System*, debemos primero abrir un flujo de datos o stream al fichero que queremos asociar cierta salida mediante la clase FileOutputStream(File).

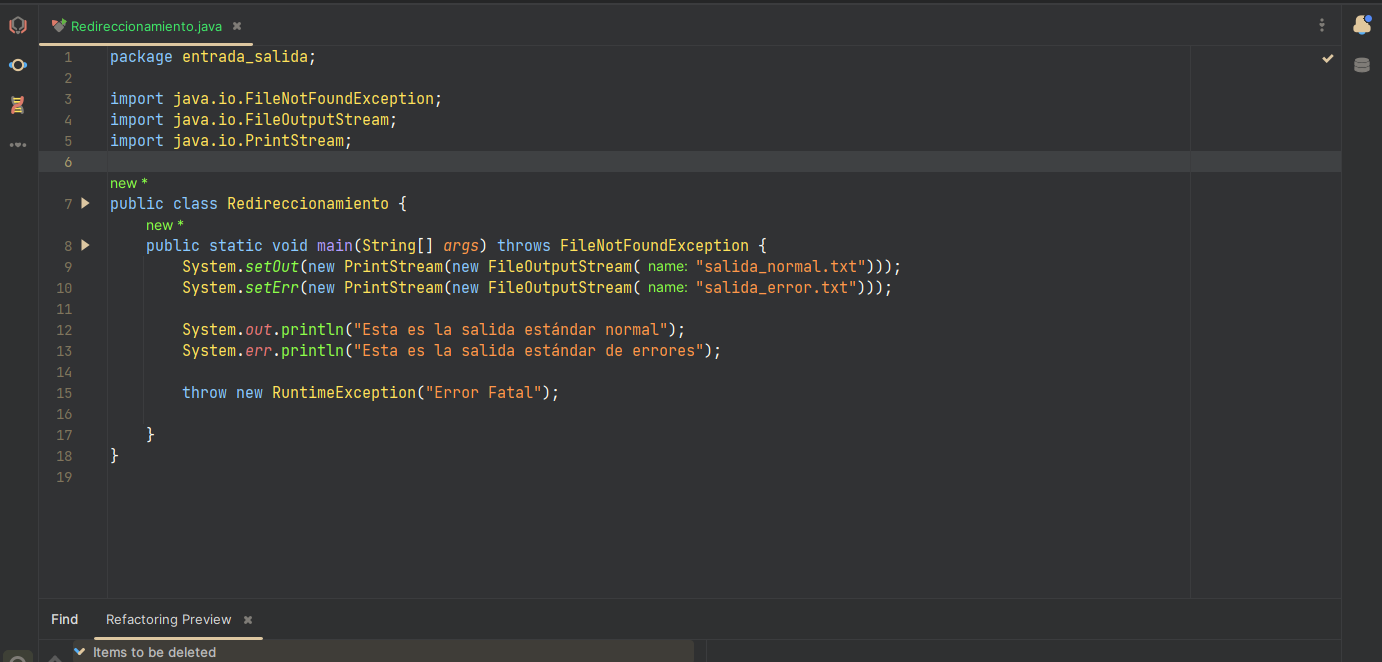


Figura 18. Ejemplo redireccionamiento salida. Fuente: elaboración propia

## Entrada de datos

### Mediante del BufferedReader

Para la lectura de datos de disponemos de la claseInputStream. Este objeto está asociado a un fichero. La clase System nos ofrece un atributo público y estático al cual podemos acceder llamado *in*. A este atributo está asociado por defecto el teclado.

|  |  |
| --- | --- |
| **Firma** | **Descripción** |
| int available() | Retorna el número estimado de bytes que pueden ser leídos sin ser bloquearse en la lectura de datos |
| void close() | Cierra el flujo de datos de este InputStream |
| void mark() | Marca la posición actual de este InputStream |
| boolean markSupported() | Prueba si este InputStream soporta los métodos read() o reset() |
| void read() | Lee el siguiente byte en el fichero que tiene asociado este InputStream |
| void reset() | Reposiciona el punto lector de datos de este InputStream a la posición donde estaba en el momento de última llamada a este método |
| long skip(...) | Salata una cantidad determinada de bytes del flujo de datos de este InputStream |

La clase InputStream nos ofrece el método read() que sólo devuelve el último carácter leído del fichero que está asociado al objeto, por lo tanto, consecutivas llamadas a este método son necesarias para leer varias cadenas de caracteres, lo cual es ineficiente. Para agilizar esta tarea disponemos de la clase BufferedReader, cuyo constructor recibe un InputStream y nos aporta métodos con los cuales leer cadenas de caracteres del fichero asociado al InpuStream que recibe en el momento de construcción.

Para leer una cadena de caracteres de la entrada de datos, la clase BufferedReader nos ofrece el método readLine(), el cual lee una secuencia de caracteres hasta el primer salto de línea y la devuelve en formato String. Un punto muy importante de este método es que lanza una excepción de tipo IOException que está marcada y por tanto es necesario capturar o volver a lanzar desde el método que se invoque.

Es importante destacar que el método readLine() devuelve una cadena de caracteres en un objeto String, si necesitamos recuperar un valor entero o decimal de la entrada de datos, tendremos que formatear la cadena devuelta por readLine() utilizando los métodos estáticos que ofrecen ciertas clases envoltorio. Los nombres de estos métodos empiezan por parseXxx() donde Xxx equivale al nombre de la clase envoltorio, tenemos como ejemplos: Integer.parseInt(cadena), Double.parseDouble(cadena), Long.parseLong(cadena), etcétera.

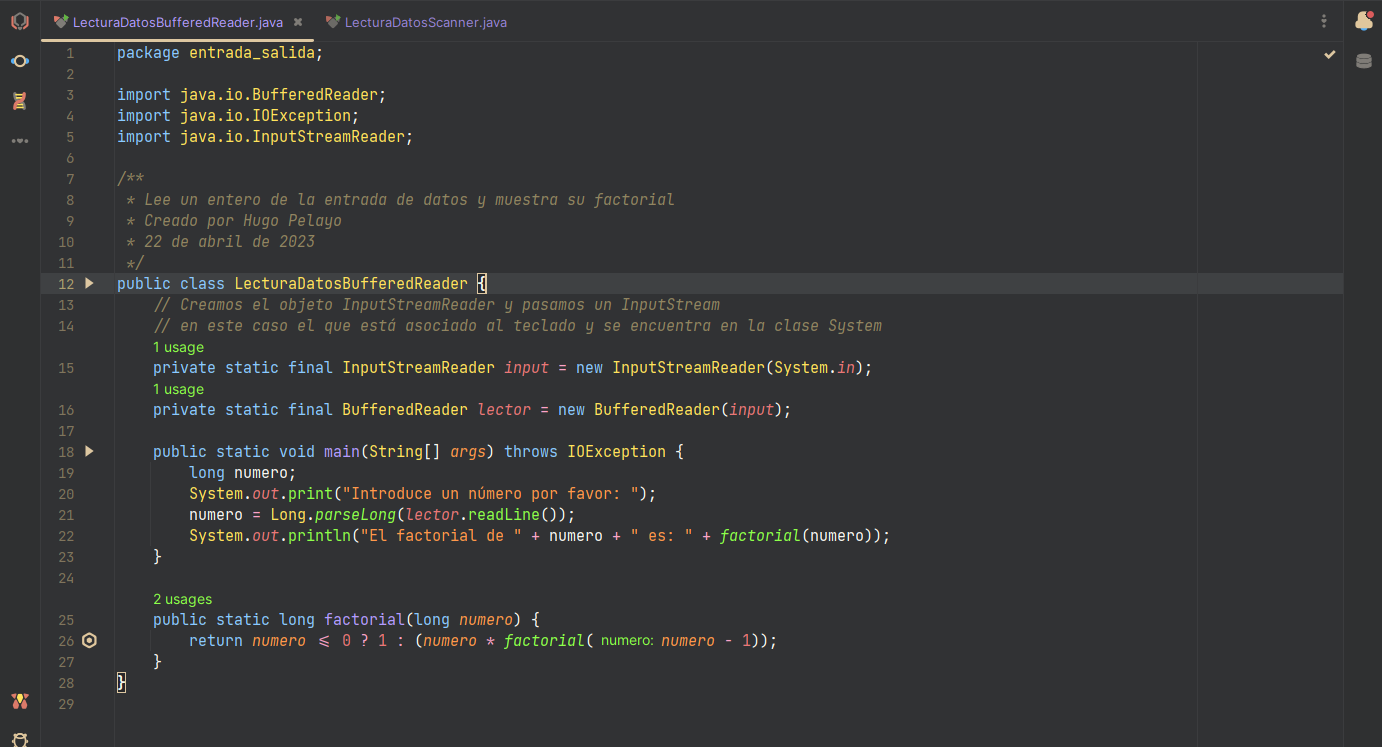


Figura 19. Ejemplo lectura de datos. Fuente: elaboración propia

### Mediante Scanner

La clase Scanner funciona de manera similar a la clase BufferedReader. La clase Scanner ofrece métodos como nextLine() para leer una cadena de caracteres, nextInt() o nextDouble() para leer un valores enteros o decimales respectivamente. Cabe destacar que estos dos últimos lanzan una excepción si la cadena leída de la entrada de datos no se válida, es decir, no se puede formatear a un valor decimal correctamente.



Figura 20. Lectura de datos con Scanner. Fuente: elaboración propia

Cierto problema que presenta la lectura con un objeto Scanner es que al cuando lee una cadena de caracteres de la entrada de datos, cuando se encuentra un salto de línea, este es abandonado en el buffer interno que el mismo objeto monitoriza, de tal manera que próximas llamadas a métodos como nextInt() o nextLine() devolverían una cadena vacía. Una forma de solucionar este problema es añadir una llamada intermedia a nextLine() que no nos sea necesaria. De todos modos, existe variedad de soluciones para este inconveniente.



Figura 21. Problema lectura con Scanner. Fuente: elaboración propia

# Estructuras de control

En la programación con Java se emplea principalmente tres tipos de estructuras para el control del flujo de ejecución de un programa. Con la introducción de estructuras de datos y ciertas características (no únicamente en Java) estas muestran formas sintácticas ligeramente variadas pero la idea principal es la misma. Podemos agruparlas en tres campos diferentes: estructuras secuenciales, estructuras selectivas y estructuras iterativas:

## Estructuras secuenciales

Están compuestas por un grupo de instrucciones o ninguna que se ejecutan en el orden en que aparecen una detrás de otra. Es la estructura más básica de ejecución de un programa y de la cual se nutren el resto. Como ejemplo tenemos básicamente cualquiera de los programas que se han mostrado en las figuras anteriores. Las estructuras secuenciales en esencia representan un paquete de instrucciones que se ejecutan como una unidad.

Las estructuras secuenciales vienen agrupadas en los llamados bloques. Estos bloques normalmente viene delimitados por llaves “{ }” para poder separarlos de otros, aunque también hay bloques sin llaves, esto pasa cuando este está compuesto por una única instrucción.

## Estructuras selectivas

En estas estructuras evaluamos una expresión cuyo resultado debe ser del tipo buleano o convertible a dicho tipo de dato. Las estructuras selectivas se dividen en estructuras de selección simples, estructuras de selección compuestas, estructuras con el operador condicional y estructuras con el operador switch.

Las estructuras selectivas también se suelen denominar estructuras de decisión, básicamente nos permite bifurcar el flujo de ejecución de un programa mediante la evaluación de una condición cuyo resultado debe ser un valor buleano, es decir, puede ser cierto o falso. En otras palabras, estas estructuras nos permiten ejecutar un grupo de instrucciones u otro en base a si una condición se da o no.

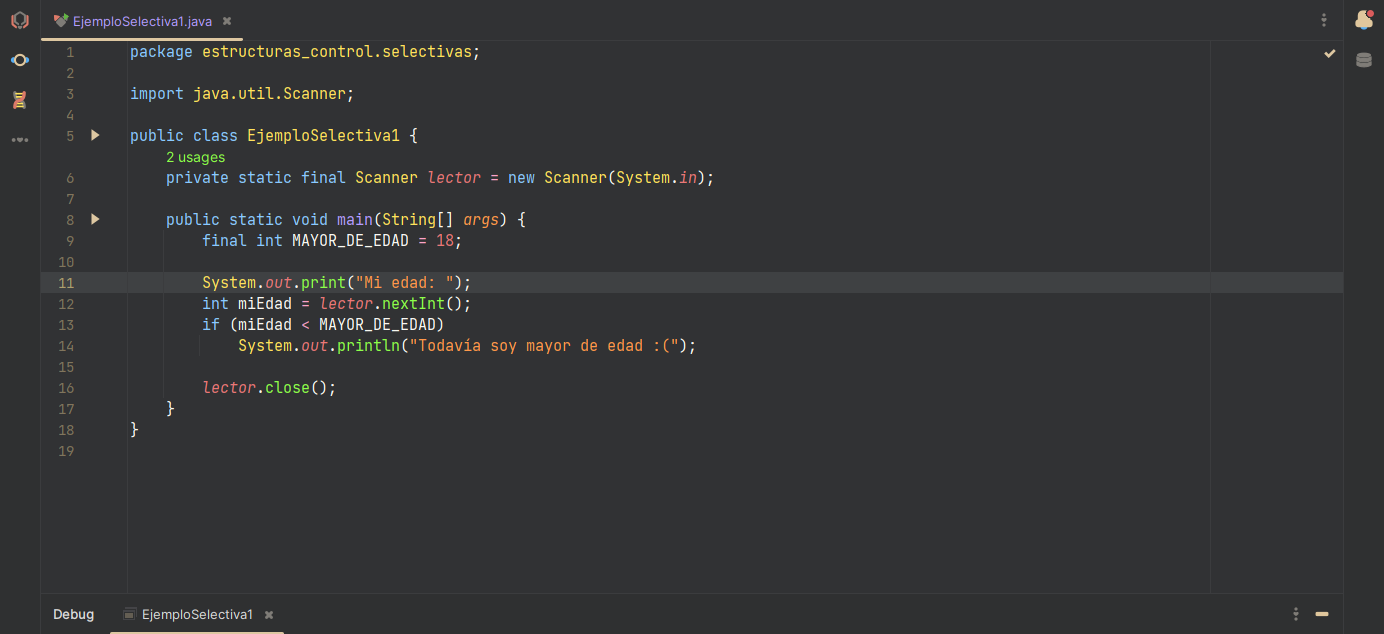


Figura 22. Ejemplo sentencia selectiva simple. Fuente: elaboración propia

La estructura selectiva *if* ofrece una variante con un bloque que de instrucciones que se puede ejecutar en caso de que la condición sea falsa, o simplemente evaluar otras condicionas en caso de que la primera resulte ser falsa. Para indicar que queremos un bloque alternativo utilizamos la cláusula *else*. Está clausula no es obligatoria, pero es recomendable si se sabe de antemano que es necesario realizar ciertas acciones en caso de no ser cierta la condición inicial.

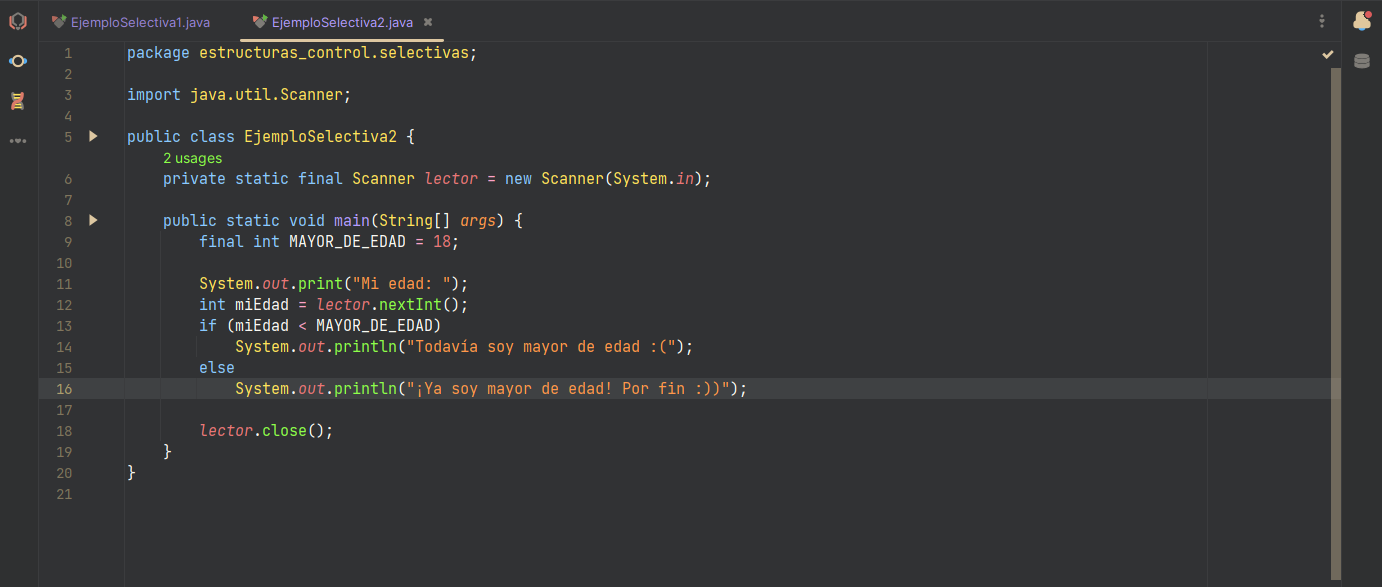


Figura 23. Estructura selectiva con bloque alternativo. Fuente: elaboración propia

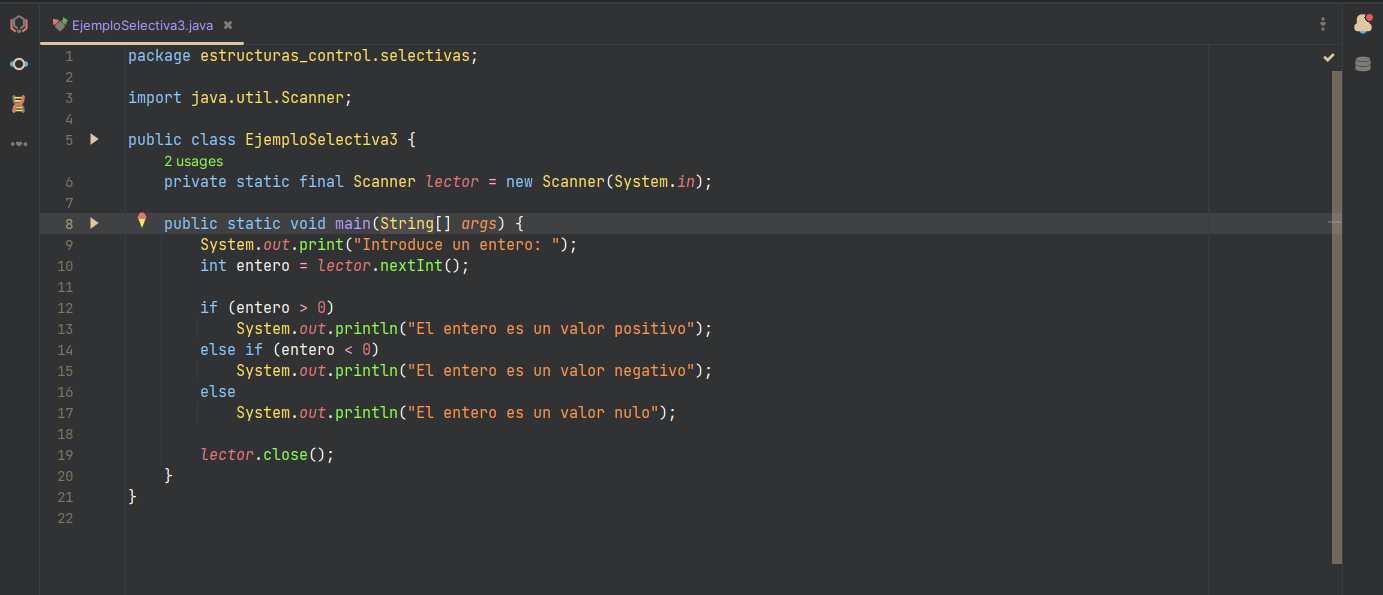


Figura 24. Estructura selectiva con evaluación múltiple. Fuente: elaboración propia

En ciertas ocasiones, queremos ejecutar un bloque de instrucciones, pero tenemos un seguido de instrucciones que se basa en evaluar si una variable es igual a cierto valor entero, esta operación acaba siendo innecesaria ya que realizamos la misma evaluación varias veces sobre la misma variable, para solucionar este inconveniente, tenemos las sentencias switch. Esta sentencia tiene dos variantes, una que se puede considerar normal y que hereda de otros lenguajes de programación y otra mejora también denominada *Enhanced Switch Statement* en inglés, que es una versión con mejora sintáctica que se hizo pública a partir de la versión 14 de Java, aunque en versiones previos, más concretamente en la 12 y 13, esta característica ya era se estaba poniendo a prueba.

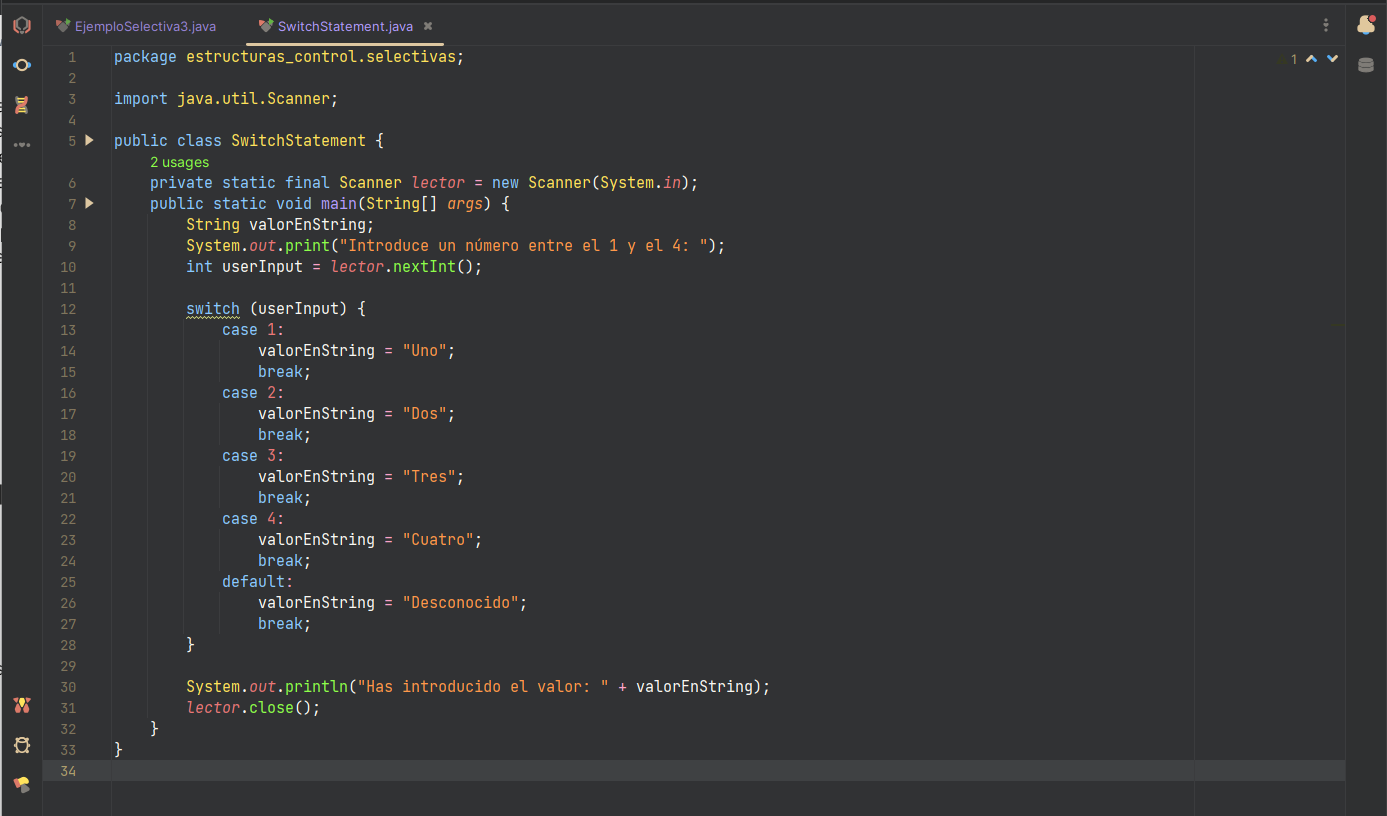


Figura 25. Ejemplo sentencia switch. Fuente: elaboración propia



Figura 26. Sentencia switch mejorada. Fuente: elaboración propia

Para acabar tenemos el operador ternario, es una forma simplificada de redactar una sentencia selectiva compuesta (un if con bloque else) la sintaxis es la siguiente:

expresion ? valor1 : valor2

Donde *expresion* es la expresión a ser evaluada, si esta es cierta el resultado global del operador ternario sería el *valor1* y *valor2* en caso contrario.

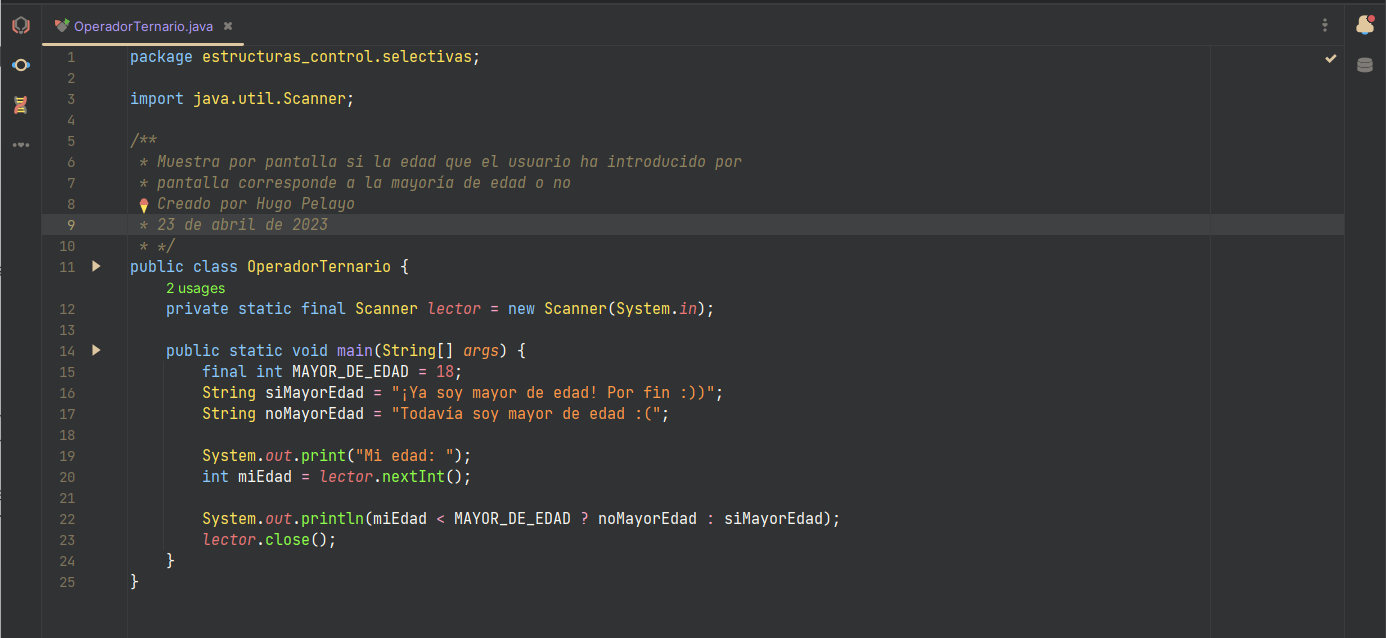


Figura 27. Ejemplo operador ternario. Fuente: elaboración propia

## Estructuras iterativas

También llamadas bucles, son una combinación de las anteriores, estas estructuras nos permiten ejecutar un grupo de instrucciones de manera reiterada mientras se cumpla una condición.

La primera estructura iterativa es el bucle while. Esta estructura nos permite repetir una serie de instrucciones mientras se cumpla una condición. Esta condición es evaluado incluso la primera vez que se intenta iterar sobre el cuerpo del bucle, de modo que, si la condición no se cumple el cuerpo no llega a ejecutarse nunca. Es imprescindible que el bucle while controlemos la condición de fin, de lo contrario se convertiría en un bucle infinito.

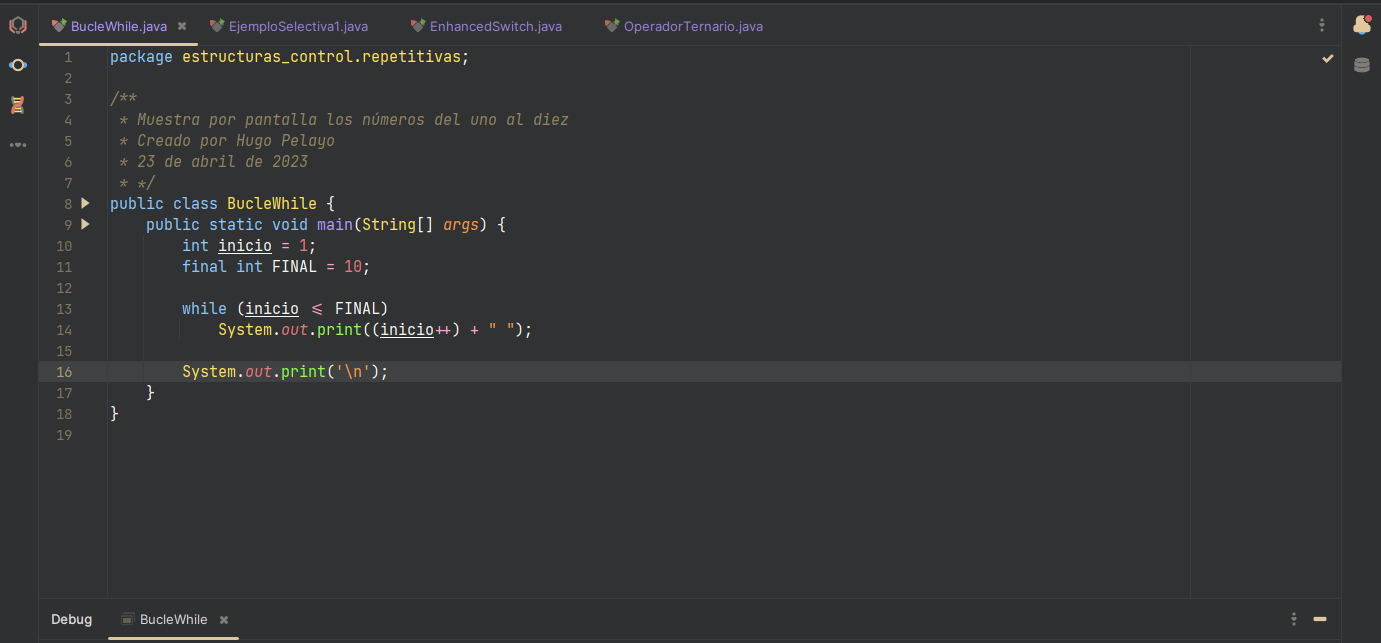


Figura 28. Bucle while. Fuente: elaboración propia

La segunda estructura repetitiva es el bucle do-while. Este bucle es similar al anterior, sin embargo, evalúa la condición después de ejecutar el cuerpo del bucle, de modo, se garantiza que nuestras instrucciones se ejecutarán al menos una vez.

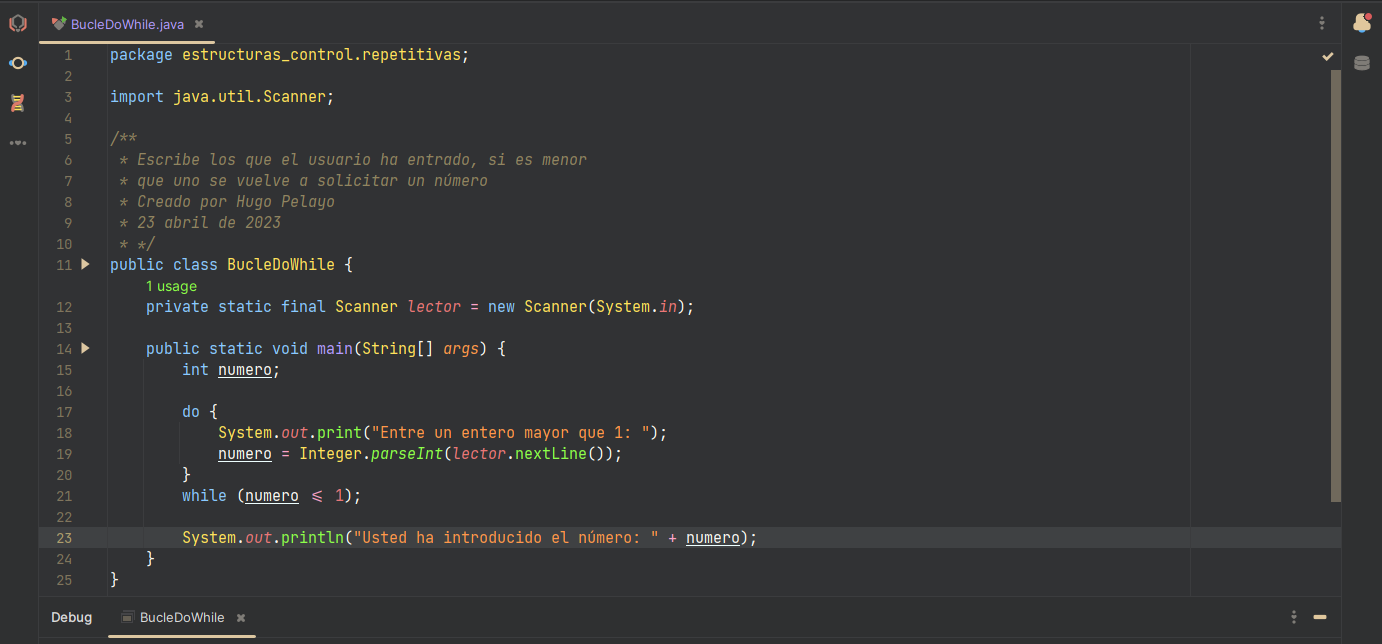


Figura 29. Bucle do-while. Fuente: elaboración propia

Por último, tenemos el bucle for, este nos permite ejecutar un conjunto de instrucciones un número conocido y fijo de veces. Es ideal cuando sabemos exactamente cuántas veces queremos iterar sobre un bloque de sentencias.

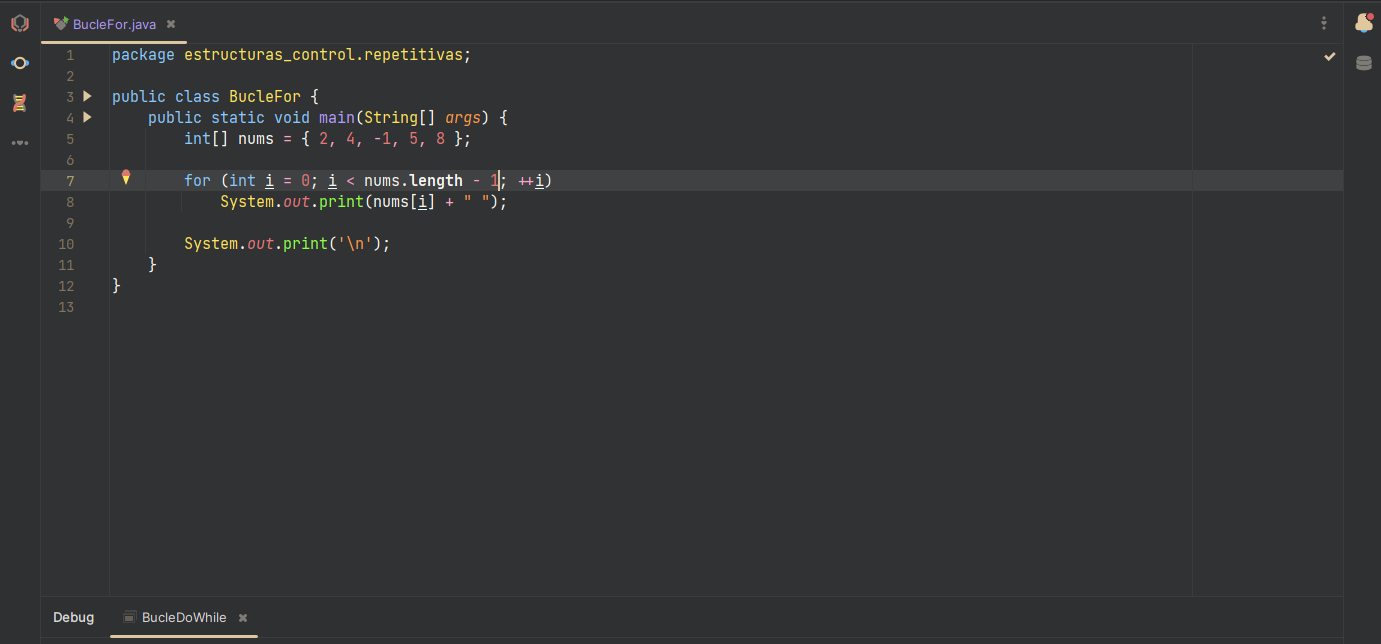


Figura 30. Bucle for. Fuente: elaboración propia

## Estructuras de salto

Las estructuras de salto son estructuras que nos sirven para romper el flujo *normal* de nuestro código o simplemente retorna de funciones o procedimientos, estás compuestas por las sentencias break, continue, return, Java no ofrece soporte para la sentencia goto. El uso de las sentencias break (en estructuras iterativas), continue acostumbra a ser desaconsejado por muchos programadores por la dificultad que estos generan a la hora de seguir el flujo de ejecución y a lógica de nuestro programa.

La sentencia break nos permite terminar inmediatamente la ejecución de un bucle, es decir, cualquier instrucción que haya después de la sentencia break no se ejecuta. La sentencia continue nos permite omitir la iteración actual, funciona de manera similar a la sentencia break, pero nos permite volver a evaluar la condición del bucle y si esta se cumple entonces se vuelve a iterar. Cuando estas sentencias se encuentran en bucles anidados, afectan al bucle más interno.



Figura 31. Sentencia break. Fuente: elaboración propia



Figura 32. Sentencia continue. Fuente: elaboración propia

Por último, tenemos la sentencia return. La primera nos permite retornar de un procedimiento y la segunda nos permite saltar a una etiqueta que indicamos en nuestro código, esta funcionalidad también es admitida por la sentencia break. La sentencia return también sirve para devolver valores de funciones:



Figura 33. Ejemplo sentencia return. Fuente: elaboración propia

# Arrays

Un array o arreglo en castellano es un objeto en el cual podemos almacenar una serie de datos del mismo tipo. Es un tipo de contenedor de elementos secuencial ya que almacena los datos de manera secuencial. Cada elemento ocupa una sola posición o índice que indica donde se sitúa un dentro del array, siendo 0 el primer índice y la máxima posición siendo el total de elementos menos uno. Para declarar un array utilizamos la siguiente sintaxis:

tipo\_dato[] identificador\_array;

tipo\_dato identificador\_array[];

Los arrays se declaran en los mismos bloques que las variables de tipos de datos básicos, es decir, los podemos declarar como atributos de una clase o como variables locales de un método. Al declarar un array, las posiciones de este se inicializan al valor por defecto del tipo de dato, es decir, en el caso de tipos básicos como enteros o decimales, se inicializan con el valor 0, si se trata de un array de valores buleanos, todas las posiciones se inicializan a falso, y en el caso de referencias, estas se inicializan todas con el valor nulo.



Figura 34. Ejemplo declaración arrays. Fuente: elaboración propia

Para asignar valores a las posiciones de un array utilizamos la sintaxis:

identificador\_array[indice] = valor;

También es posible darle valor inicial a nuestro array en el momento de la creación con la siguiente sintaxis.

tipo\_dato[] identificador\_array = { valor1, valor2, valor3 .. valorN };

tipo\_dato identificador\_array[] = { valor1, valor2, valor3 .. valorN };

tipo\_dato[] identificador\_array = new tipo\_dato[]{ valor1, valor2, valor3 .. valorN };

tipo\_dato identificador\_array[] = new tipo\_dato[]{ valor1, valor2, valor3 .. valorN };

Cabe destacar que es incorrecto declarar el array con valores iniciales y con tamaño incial a la vez indiferentemente del tamaño inicial que le demos y los valores que asignemos, es decir, las siguientes sintaxis son errónea:

Los arrays son objetos de tipo referencia (no son tipos básicos). Como todos los objetos, podemos pasar los arrays como argumentos a funciones. Al pasar un array a un método, este crea una copia local que es una referencia que apunta a la dirección de memoria donde se localiza el array original, de tal modo que podemos modificar nuestro array desde cualquier método que lo reciba como parámetro.

tipo\_dato[] identificador\_array = new tipo\_dato[tamanio]{ valor1, valor2, valor3 .. valorN };

tipo\_dato identificador\_array[] = new tipo\_dato[tamanio]{ valor1, valor2, valor3 .. valorN };



Figura 35. Pasando arrays como argumentos. Fuente: elaboración propia

Para recorrer los elementos de un array podemos utilizar los bucles vistos con anterioridad, de tal modo que podemos acceder a todos sus elementos o a unos específicos. A parte, tenemos el bucle *for-each* que está disponible desde la versión 5 de Java y nos permite recorrer todos los elementos de un objeto que esté marcado como iterable, tal es el caso del array y otros contenedores de la librería de Java. La sintaxis para el bucle *for-each* es la siguiente:

for (tipo\_dato identificador\_variable : contenedor)

sentencias...

Contenedor es el identificador del contenedor o iterable que queremos recorrer:



Figura 36. Ejemplo recorridos array. Fuente: elaboración propia

Los arrays también pueden ser multidimensionales, de tal manera que tenemos estructuras que se asemejan a las matrices. La sintaxis de declaración es similar a la de los arrays unidimensionales:

tipo\_dato[] identificador\_array = new tipo\_dato[tamanio1][tamanio2];

tipo\_dato identificador\_array[] = new tipo\_dato[tamanio1][tamanio2];

La estructura en memoria se asemeja al siguiente esquema:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | ... | N |
| 0 | identificador[0][0] | identificador[0][1] | identificador[0][...] | identificador[0][N] |
| 1 | identificador[1][0] | identificador[1][1] | identificador[1][...] | identificador[1][N] |
| ... | identificador[...][0] | identificador[...][1] | identificador[...][...] | identificador[...][N] |
| N | identificador[N][0] | identificador[N][1] | identificador[N][...] | identificador[N][N] |

La idea es tener un array de arrays, de tal modo que, con el primer índice, accedemos un array en concreto y con el segundo accedemos a un elemento específico. También es posible realizar el mismo recorrido con el bucle *for-each*. Cabe destacar que no necesariamente debe construirse un array bidimensional con la estructura de una matriz. La estructura puede ser irregular, para esto tendríamos que manualmente indicar el tamaño de los subarrays.

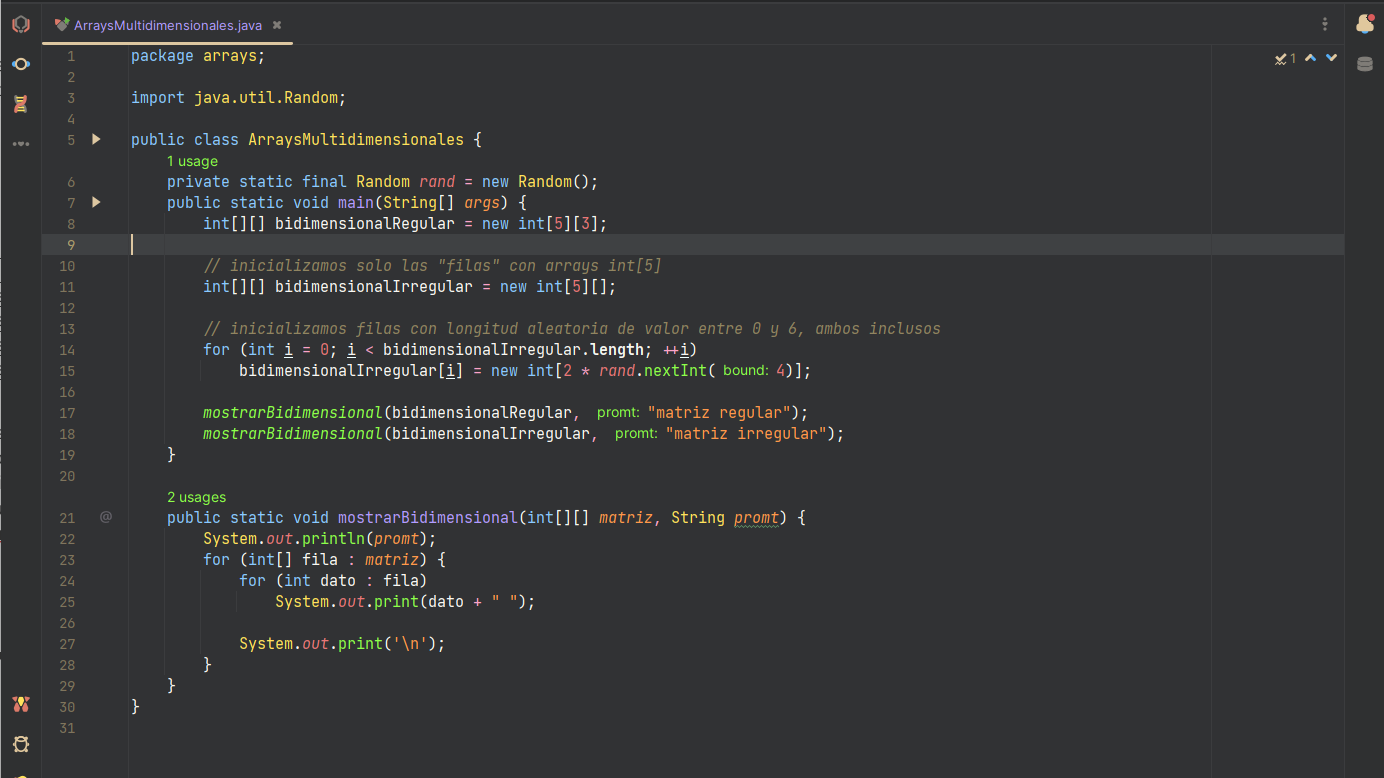


Figura 37. Inicialización de arrays multidimensionales. Fuente: elaboración propia

# Apéndice

A continuación, se lista una serie de defunciones de aquellos términos que se introdujeron en el manual y que puedan haber resultados confusos o desconocidos para el lectora o lectora.

* Algoritmo:
* Bytecode: También llamado *bytecode Java*, es el tipo de instrucciones capaces de ser interpretadas por la máquina virtual de Java (JVM). Se encuentra en los ficheros con extensión .class que puede generar un compilador de java. Estas instrucciones son interpretadas por un traductor JIT (Just-In-Time) en tiempo de ejecución el cual las traduce al lenguaje máquina de la plataforma subyacente.
* Clase: plantilla que representa atributos de una entidad junto con operaciones sobre como operar sobre los mismos, abstrayéndonos de detalles irrelevantes de estos.
* Expresión: combinación de valores constantes o literales, funciones y variables que se interpretan de acuerdo a las normas de precedencia y asociación de un lenguaje.
* Expresión lógica: expresión cuya evaluación retorna un valor que puede ser cierto o verdadero.
* Firma (Function Signature)
* Función:
* Hardware: Es el conjunto de elementos físicos y palpables que constituyen una computadora o sistema informático. Destacan entre ellos los dispositivos de entrada como el teclado o ratón y los de salida como el monitor.
* IDE:
* Iterable:
* Módulo:
* Paradigma:
* Procedimiento:
* Proceso: en un sistema operativo, representa un programa en ejecución. Los procesos normalmente son las entidades que mantienen en funcionamiento nuestro sistema ofreciendo servicios como lectura de datos, o control sobre nuestro hardware, de ellos el sistema operativo generalmente guarda información como el identificador, el consumo de memoria, tiempo en ejecución, entre otros.

# Bibliografía

|  |  |
| --- | --- |
| [1] | Wikipedia, «Software,» Wikipedia, 2023 Marzo 31. [En línea]. Available: https://es.wikipedia.org/wiki/Software. [Último acceso: 2023 Abril 3]. |
| [2] | Wikipedia, «Programa informático,» Wikipedia, 24 Marzo 2023. [En línea]. Available: https://es.wikipedia.org/wiki/Programa\_inform%C3%A1tico. [Último acceso: 6 Abril 2023]. |
| [3] | Wikipedia, «Pseudocódigo,» Wikipedia, 20 Febrero 2023. [En línea]. Available: https://es.wikipedia.org/wiki/Pseudoc%C3%B3digo. [Último acceso: 8 Abril 2023]. |
| [4] | F. H. Chowdhury, «The Java Handbook - Learn Java,» FreeCdeChamp, 7 Septiembre 2022. [En línea]. Available: https://www.freecodecamp.org/news/the-java-handbook/. [Último acceso: 10 Abril 2023]. |
| [5] | visure, «Qué son los requisitos no funcionales: ejemplos, definición, guía completa,» VISURE, [En línea]. Available: https://visuresolutions.com/es/blog/requerimientos-no-funcionales/#:~:text=Los%20requisitos%20no%20funcionales%20son%20las%20restricciones%20o%20los%20requisitos,la%20confiabilidad%20y%20muchos%20m%C3%A1s.. [Último acceso: 10 Abril 2023]. |
| [6] | Oracle, «Formatter,» [En línea]. Available: https://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html#syntax. [Último acceso: 2023 abril 22]. |