**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](#_Toc132053316)

[Desarrollo de software 5](#_Toc132053317)

[El programa 6](#_Toc132053318)

[Apéndice 8](#_Toc132053319)

[Bibliografía 10](#_Toc132053320)

# 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.

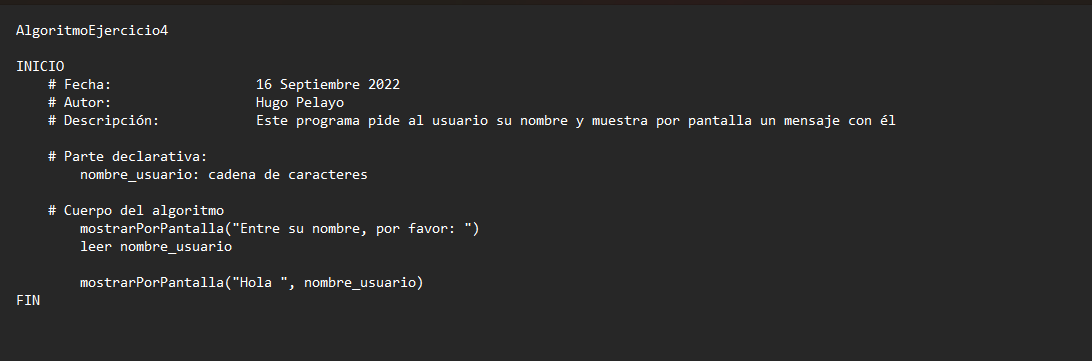


Figura 1. Ejemplo de pseudocódigo. Fuente: creación propia

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.

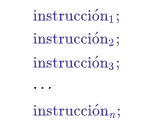


Figura 2 Estructura secuencial. Fuente: Wikipedia

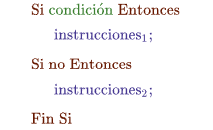


Figura 3 Estructura selectiva con alternativa. Fuente: Wikipedia

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.

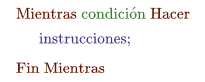


Figura 4. Formato del bucle mientras. Fuente: Wikipedia

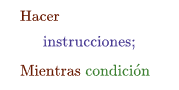


Figura 5. Formato del bucle hacer. Fuente: Wikipedia

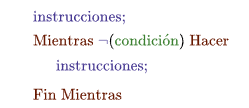


Figura 6. Formato del bucle repetir. Fuente: Wikipedia

# 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).

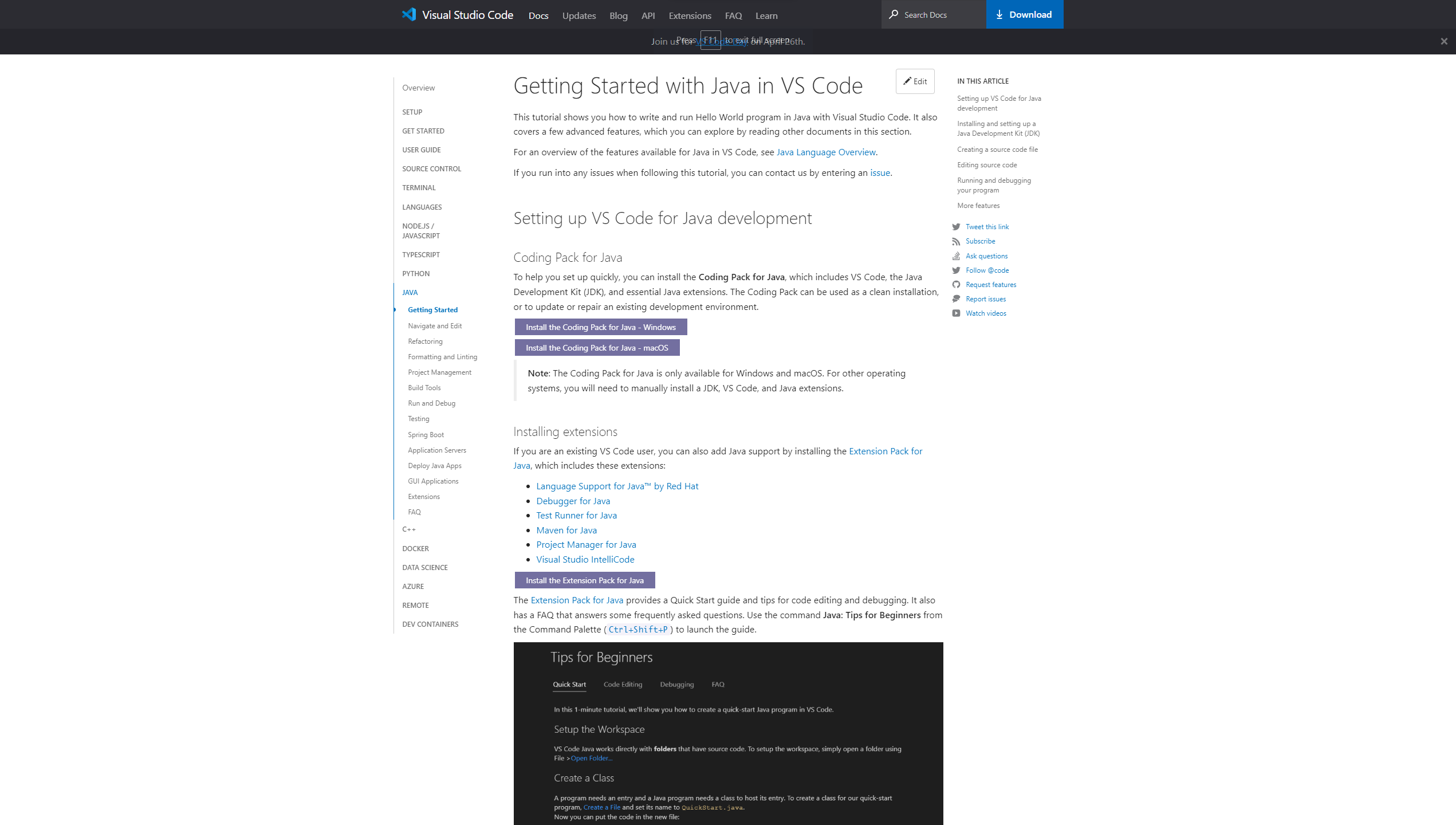


Figura 7. Tutorial Java VS Code. Fuente: Elaboración propia

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:

## IntelliJ IDEA para la programación en Java

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 estándard de datos tenemos la clase PrintStream de la cual se especializan muchas otras, tmbién tenemos BufferedWritter, InputStreadWritter, entre otros.

La salida estándard 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 estándard de todos los procesos se asocia al terminal, sin embargo, este comportamiento se puede cambiar redirecionando la salida del proceso en cuestión a dispositivos como sockets, ficheros, etcétera.

Para enviar datos por la salida estándard 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 envian cadenas de caracteres por la salida de datos, la principal diferencia entre ambos es que el segundo añade un salto de línea al final (caracter 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.

[]Insertar ejemplo de escritura con system.out

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 mostrala por la salida de datos. La cadena que nosotros enviamos a estos métodos está formada por nuestro texto acompañado de una serie de flags de formateo en aquellas posiciones donde queramos un formato de muestreo específico, la sintaxis para los flags 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 cual 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 diferentes especificadores de conversión:

poner flags de formateo: https://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html#syntax

A parte de salida estándard de datos existe la salida estándard 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 difrerenciación es que podemos redirecciónar la salida estánmdard de un proceso y la salida estándard de error a ficheros diferentes, por ejemplo podriamos tener nuestra salida estándard vinculada a la terminal o un fichero de texto con nombre "salida.txt" y un nuestra salida estándard de error a otro fichero con nombre "errores.log" o a la misma terminal. En algunos IDEs como NetBeans, la salida estándard de error se suele mostrar con un color rojo llamativo ya que se va más alarmante, sin embargo la salida estándard suele estar en un color negro.

Desde java podemos cambiar la salida estándard de nuestro programa o su salida estándard 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)

System.setOut(new PrintStream(new FileOutputStream("salida\_normal.txt")));

System.setErr(new PrintStream(new FileOutputStream("salida\_error.txt")));

System.out.println("Esta es la salida estandar normal");

System.err.println("Esta es la salida estandar de errores");

throw new RuntimeException("Error Fatal");

Apendice:

(1)proceso:

# Apéndice

* Algoritmo:
* Bytecode:
* Clase:
* Expresión:
* Expresión lógica:
* Función:
* Hardware:
* IDE:
* Módulo:
* Paradigma:
* Procedimiento:

# Bibliografía

|  |  |
| --- | --- |
| [1] | Wikipedia, «Software,» Wikipedia, 2023 Marzo 31. [En línea]. Available: https://es.wikipedia.org/wiki/Software. [Último acceso: 2023 Abril 3]. |