

# UNIDAD 1.

## Introducción a la programación y elementos básicos de los lenguajes de programación.



# Índice

1. Ficha unidad didáctica.....	1
2. Contenidos.....	2
2.1. Introducción.....	2
2.2. Lenguajes de programación.....	3
2.3. Compiladores, intérpretes, IDE, lenguajes, ensamblador.....	6
2.3.1 . Compiladores e intérpretes.....	6
2.4. Utilización de los entornos integrados de desarrollo.....	9
2.5. Estructura y bloques fundamentales.....	14
2.6. Java.....	16
2.7. Variables.....	19
2.7.1 Tipos de datos.....	21
2.7.2 Operadores y expresiones.....	23
2.7.2.1 Operadores de asignación, aritméticos y unarios.....	23
2.7.2.2 Precedencia de operadores.....	28
2.7.3 Conversiones de tipo.....	29
2.8. Constantes.....	31
2.9. Comentarios.....	31
3. Actividades y ejercicios.....	32
4. Bibliografía.....	36

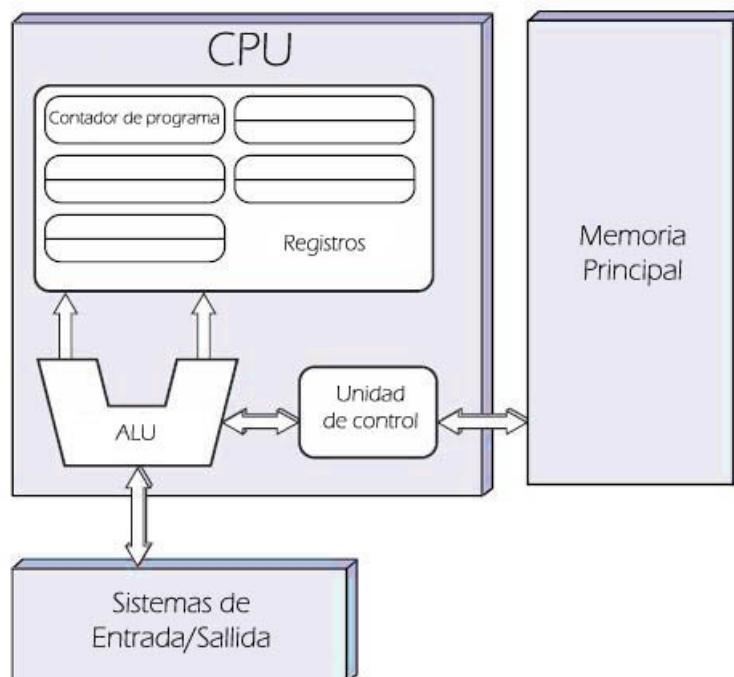
# 1. Ficha unidad didáctica.

OBJETIVOS DIDÁCTICOS	
<p>OD1: Describir el proceso de generación de código máquina/ ejecutables.  OD2: Exponer las diferencias entre lenguajes compilados e interpretados.  OD3: Caracterizar las fases de traducción de código fuente a máquina.  OD4: Crear y modificar proyectos en entornos de desarrollo.  OD5: Exponer la estructura básica de un programa simple.  OD6: Enunciar el concepto de algoritmo.  OD7: Usar los tipos de datos adecuados en función de las necesidades .  OD8: Utilizar operadores y variables correctamente.  OD9: Interpretar código con variables, constantes y operadores.  OD10: Prever y conocer el funcionamiento de la conversión de tipos.  EV1. Reconocer el derecho a propiedad intelectual de programas, código y librerías, no haciendo un uso indebido o ilícito de estos.  EV2. Concienciar de la importancia de emplear hábitos respetuosos con el medio ambiente y control del gasto energético de las instalaciones informáticas.  RL1. Utilizar de forma adecuada y ergonómica el mobiliario de oficina, evitando posturas incorrectas que conlleven lesiones.  TIC2. Usar Internet como forma de actualización en nuevas tecnologías relacionadas con el ámbito laboral de la informática.  ID1: Habituar a la lectura y comprensión de documentación técnica en inglés.</p>	
<b>RESULTADOS DE APRENDIZAJE</b>	RA1
<b>OBJETIVOS GENERALES</b>	j, q
<b>COMPETENCIAS</b>	a, e, f, i, j, t
CONTENIDOS	
<p>Lenguajes de programación.  Compiladores, intérpretes, IDE, lenguajes, ensamblador.  Utilización de los entornos integrados de desarrollo.  Soluciones y proyectos.  Estructura y bloques fundamentales.  Java  Variables.  Tipos de datos.  Operadores y expresiones.  Conversiones de tipo.  Constantes.  Comentarios.</p>	
ORIENTACIONES METODOLÓGICAS	
<p>Se relaciona las actividades y proceso de enseñanza con el futuro laboral del alumnado.  Se utilizan actividades de investigación que permiten al alumno tener una visión global del módulo.  Las clases y actividades se orientan a despertar el interés por el módulo.  Las actividades son incrementales en duración y dificultad.  Se plantean retos entre los alumnos para despertar el interés.  Se desarrollan ejemplos y asesora para resolver las actividades de forma metódica.</p>	
<b>CRITERIO DE EVALUACIÓN</b>	1a, 1b, 1c,1d, 1e, 1f, 1g, 1h, 1i.

## 2. Contenidos.

### 2.1. Introducción.

El funcionamiento de un equipo informático es relativamente sencillo, en un área de la memoria principal se tiene un conjunto de bits agrupados en unidades denominadas instrucciones, que son potencias de 2 y cuyo tamaño depende de la arquitectura del equipo: 8 bits, 16 bits, 32 bits, 64 bits, 128 bits. El procesador obtiene la siguiente instrucción a ejecutar, la interpreta y manda "órdenes" a los diferentes elementos del equipo como registros, memoria principal, ALU o periféricos, una vez finalizada la instrucción pasa a la siguiente instrucción de la memoria principal.



Reflexionar: ¿Es fácil para un humano crear, entender y modificar programas en lenguaje binario?

Para facilitar la tarea de crear y modificar programas se han desarrollado diferentes lenguajes de programación cuyo principal objetivo es facilitar la creación de programas a los humanos. A pesar de existir una gran variedad de lenguajes y paradigmas, la

ejecución se realiza en un procesador con programas binarios, por lo que de una u otra forma los fundamentos para escribir programas.

En esta unidad se tratan en primer lugar y de forma superficial el proceso de creación de programas desde su codificación en lenguajes de alto nivel hasta su transformación en lenguaje máquina, explicando la estructura básica de un programa junto con las herramientas utilizadas en dicho proceso. En la segunda parte de la unidad se tratan algunos de los elementos básicos de los lenguajes de programación como las constantes y las variables.



[Vídeo de la UPV sobre el proceso de creación de un programa.](#)

## 2.2. Lenguajes de programación.

Inicialmente el desarrollo del software se realizaba conectando directamente conectores entre los diferentes elementos del equipo, esto era una tarea laboriosa, muy dada a fallos y lo que es peor difícil de detectar.

La siguiente evolución fue el desarrollo de las tarjetas perforadas en las cuales se tenía en binario el programa, es decir los códigos que la CPU entiende junto con los operadores (registros, literales, direcciones de memoria),



[Vídeo sobre el ordenador IBM 1401 y el lenguaje FORTRAN](#)

Este sistema seguía siendo en binario, pero los desarrolladores escribían nemotécnicos para crear el programa y no binario ya que **facilita el desarrollo pensar en ADD, MOVE...**que en 01010101, no tardaron mucho en crear programas que traducían de esos códigos a lenguaje binario, naciendo el ENSAMBLADOR. **El lenguaje binario/máquina y ensamblador se conocen como lenguajes de bajo nivel.**

La siguiente evolución conocida como lenguajes de alto nivel intentan definir lenguajes lo más próximos al lenguaje natural.



¿Por qué se evoluciona hacia lenguajes cada vez más parecidos al lenguaje humano?

Existen otras clasificaciones en cuanto a los lenguajes de programación por ejemplo el **tipo de paradigma** (una teoría o modelo explicativo de las realidades, se modela el software en función de una “forma de pensar”):

- **Imperativos.** Se basa en una lista de sentencias explícitas. El más característico es C.
- **Orientados a objetos.** Se modela la realidad en base a clases y objetos, los lenguajes más usados en la actualidad posee características o son cien por cien orientados a objetos, los más usados C++/ Java, C# y con características aunque no 100% OO Python o Javascript (depende de la versión)
- **Funcionales.** Ha vuelto a utilizarse después de un tiempo, los lenguajes han incluido en los últimos años características funcionales, y su base es el uso de la función lambda, permite entre otros realizar mapeos y filtros de forma sencilla. El clásico es Lisp. Un ejemplo de filtrado en Javascript.

```
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction',  
'present'];  
  
const result = words.filter(word => word.length > 6);  
console.log(result);  
// expected output: Array ["exuberant", "destruction", "present"]
```

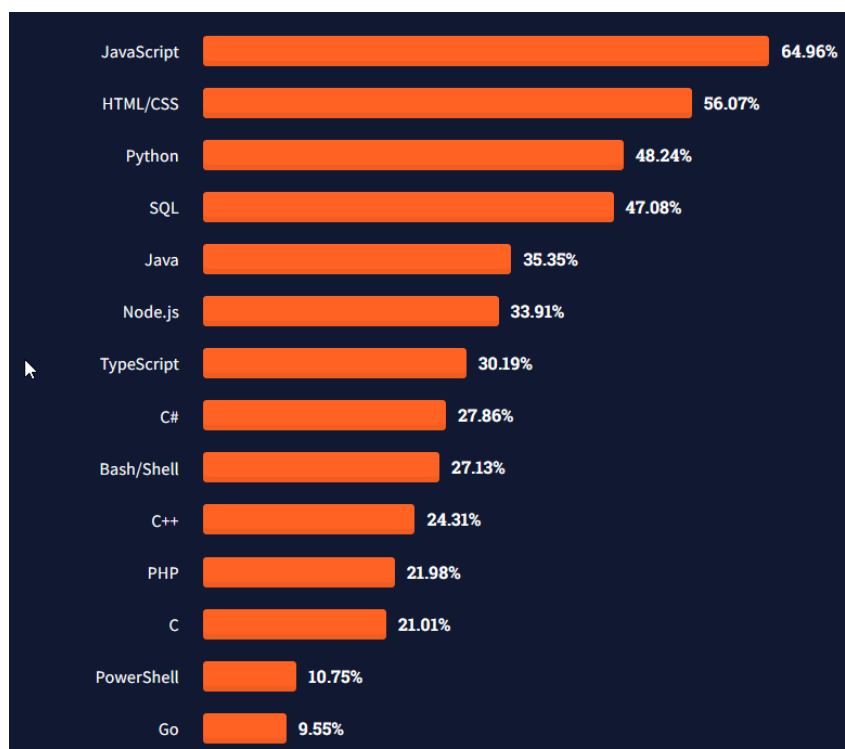
- **Lógicos.** Basados en la lógica de predicados, se tiene una serie de reglas y usando la lógica a partir de una entrada (hechos) es capaz de inferir una solución. El más conocido es Prolog. Usado en inteligencia artificial y entornos de investigación y universitarios.

Los lenguajes actuales son mayoritariamente OO, normalmente añaden programación funcional y se puede desarrollar de modo imperativo, y con el uso de librerías externas se puede realizar programación lógica con lo que los lenguajes que solo están enfocados a un paradigma cada vez son menos comunes.

Existen muchas otras clasificaciones como compilados/interpretados, evolución histórica, estáticos/dinámicos/basados en pila... En la siguiente imagen se puede ver el índice tiobe que clasifica a los lenguajes de programación más usado: <https://www.tiobe.com/tiobe-index/>

Aug 2021	Aug 2020	Change	Programming Language	Ratings	Change
1	1		C	12.57%	-4.41%
2	3	▲	Python	11.86%	+2.17%
3	2	▼	Java	10.43%	-4.00%
4	4		C++	7.36%	+0.52%
5	5		C#	5.14%	+0.46%
6	6		Visual Basic	4.67%	+0.01%
7	7		JavaScript	2.95%	+0.07%
8	9	▲	PHP	2.19%	-0.05%
9	14	▲▲	Assembly language	2.03%	+0.99%
10	10		SQL	1.47%	+0.02%
11	18	▲	Groovy	1.36%	+0.59%
12	17	▲	Classic Visual Basic	1.23%	+0.41%
13	42	▲	Fortran	1.14%	+0.83%
14	8	▼	R	1.05%	-1.75%
15	15		Ruby	1.01%	-0.03%
16	12	▼	Swift	0.98%	-0.44%
17	16	▼	MATLAB	0.98%	+0.11%
18	11	▼	Go	0.90%	-0.52%
19	36	▲	Prolog	0.80%	+0.41%

El sitio web [stackoverflow.com](https://stackoverflow.com) (foro de desarrolladores a nivel mundial) realiza encuestas entre sus usuarios sobre diferentes aspectos del desarrollo, siendo las tecnologías más usadas en el 2021 las siguientes:



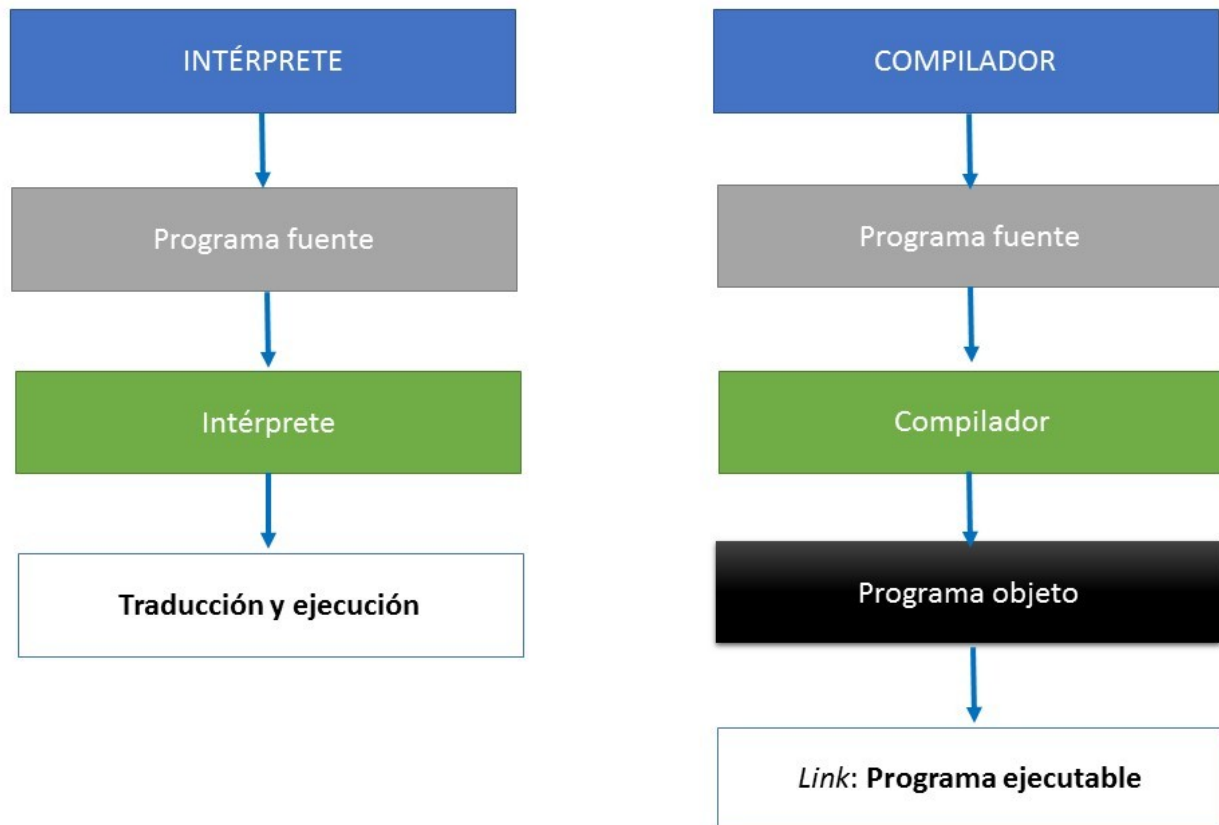
## 2.3. Compiladores, intérpretes, IDE, lenguajes, ensamblador

### 2.3.1 . Compiladores e intérpretes.

Los traductores son programas cuya finalidad es traducir lenguajes de alto nivel (en los que se programa) a lenguajes de bajo nivel como ensamblador código máquina. Existen dos grandes grupos de tipos de traductores: los compiladores y los intérpretes:

- **Un compilador** traduce el un programa entero de un lenguaje de programación (llamado código fuente) a otro denominado lenguaje objeto. Usualmente, el lenguaje objetivo es código máquina, aunque también puede ser traducido a un código intermedio (bytecode) o a texto. El compilador únicamente estará instalado en la máquina de desarrollo. El código generado para un sistema sólo funcionará para una arquitectura hardware y software determinadas. Si se desea ejecutar en un sistema con un hardware o software diferente habrá que volver a recompilarlo. Ejemplos de lenguajes compilados típicos son: C/C++ y Pascal, entre otros.
- **Un intérprete** traduce el código fuente línea a línea como se describe a continuación: primero traduce la primera línea, detiene la traducción y, seguidamente la ejecuta; lee la siguiente línea, detiene la traducción y la ejecuta, y así sucesivamente. El intérprete tiene que estar cargado en memoria ejecutándose para poder ejecutar el programa. Al igual que el intérprete, el código fuente tiene que estar también en la memoria. En caso de detectar un error durante el proceso de traducción el intérprete detiene la ejecución del programa. Los siguientes enlaces llevan a las páginas oficiales de los principales lenguajes interpretados, en las que se puede encontrar documentación de los mismos, librerías, así como cursos y materiales de aprendizaje:
  - PHP.
  - Perl.
  - Python.
  - Ruby.





Al observar las diferencias entre compilador e intérprete, se ve claramente los puntos fuertes y débiles de cada solución para traducir el código fuente: con el intérprete, los programas se pueden ejecutar de inmediato y, por lo tanto, se inician mucho más rápido. Además, el desarrollo es mucho más fácil que con un compilador, porque el proceso de depuración (es decir, la corrección de errores) se lleva a cabo igual que la traducción, línea por línea. En el caso del compilador, primero debe traducirse todo el código antes de poder resolver los errores o iniciar la aplicación. Sin embargo, una vez que se ejecuta el programa, los servicios del compilador ya no son necesarios, mientras que el intérprete continúa utilizando los recursos informáticos.

Algunas ventajas de compilar frente a interpretar son:

Se compila una vez; se ejecuta muchas veces.

- La ejecución del programa objeto es mucho más rápida que si se interpreta el programa fuente.
- El compilador tiene una visión global del programa, por lo que la información de los mensajes de error es más detallada.

Por otro lado, algunas ventajas de interpretar frente a compilar son:

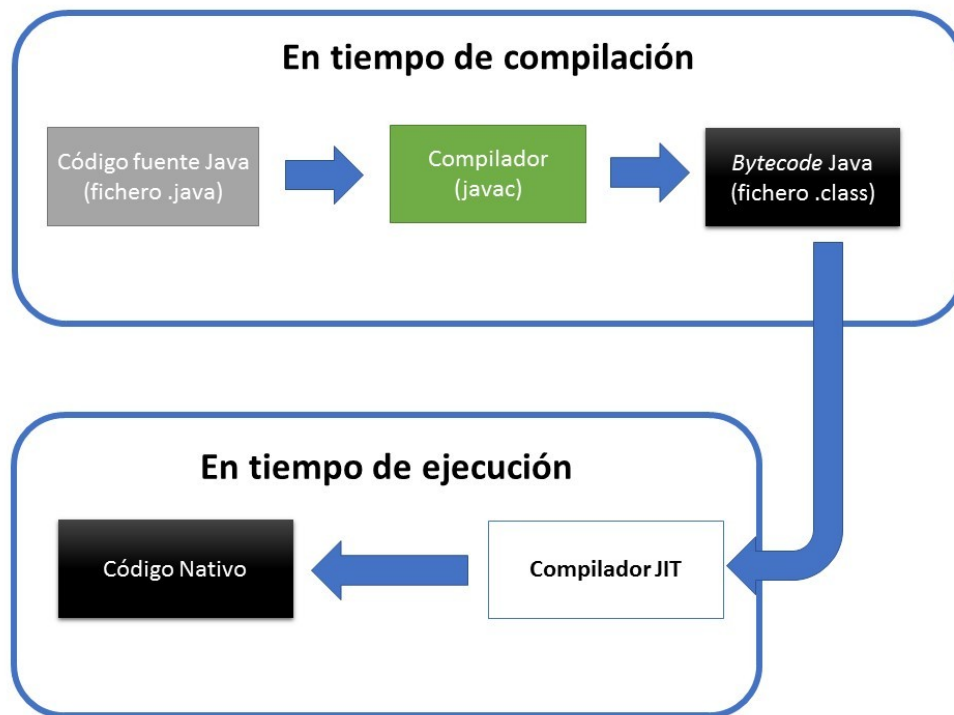
- Un interprete necesita menos memoria que un compilador.
- Permite una mayor interactividad con el código en tiempo de desarrollo (e incluso, en algunos casos, mientras se ejecuta el código).

Para compensar los puntos débiles de ambas soluciones, también existe el llamado modelo de compilación en tiempo de ejecución (en inglés, Just-in-time-compiler, o “compilador justo a tiempo”). Este tipo de compilador, que a veces también se conoce por el término inglés *compreter* (acrónimo de *compiler* e *interpreter*), rompe con el modelo habitual de compilación y traduce el código del programa durante el tiempo de ejecución, al igual que el intérprete. De esta forma, la alta velocidad de ejecución típica de los compiladores se complementa con la simplificación del proceso de desarrollo.

En un entorno de compilación en tiempo de ejecución, la compilación a bytecode es el primer paso, reduciendo el código fuente a una representación intermedia portable y optimizable. El bytecode se despliega en el sistema de destino. Cuando dicho código se ejecuta, el compilador en tiempo de ejecución lo traduce a código máquina nativo. Esto puede realizarse a nivel de fichero (programa) o de funciones, compilándose en este último caso el código correspondiente a una función justo cuando va a ejecutarse (de aquí el nombre de just-in-time, «justo a tiempo»).

El objetivo es combinar muchas de las ventajas de la compilación a código nativo y a bytecode: la mayoría del «trabajo pesado» de procesar el código fuente original y realizar optimizaciones básicas se realiza en el momento de compilar a bytecode, mucho antes del despliegue: así, la compilación a código máquina del programa resulta mucho más rápida que partiendo del código fuente. El bytecode desplegado es portable, a diferencia del código máquina para cualquier arquitectura concreta. Los compiladores dinámicos son más fáciles de escribir, pues el compilador a bytecode ya realiza buena parte del trabajo.

Java es uno de los ejemplos más conocidos de lenguaje basado en compilación en tiempo de ejecución: el compilador JIT, que figura entre los componentes del Java Runtime Environment (JRE), mejora el rendimiento de las aplicaciones Java traduciendo el código de bytes en código máquina de manera dinámica.



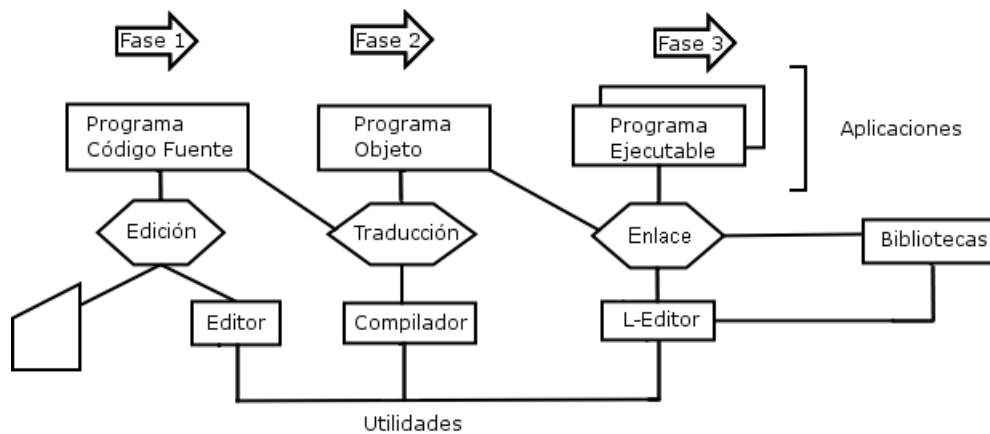
## 2.4. Utilización de los entornos integrados de desarrollo.


Tal y como se ha comentado en puntos anteriores los sistemas informáticos ejecutan instrucciones en lenguaje binario o en lenguajes próximos como lenguajes intermedios usados en algunos lenguajes sobre máquinas virtuales como Java con ByteCodes o C# con CLI. Para la creación del software no se escribe en estos lenguajes, ya que su creación y especialmente su modificación es prácticamente imposible por pequeño que sea el programa a realizar, utilizándose lenguajes de programación de alto nivel que permiten la creación y modificación del software.



**Al conjunto de líneas de texto escritas en un lenguaje de programación que contienen la lógica de la aplicación se le denomina código fuente.**

Al introducir el código fuente en un compilador se obtiene el código objeto concreto para una arquitectura, por último se realiza el "linkado" que une diferentes ficheros de código objeto como el generado por el código fuente y librerías externas necesarias para obtener el ejecutable.



 [Explicación de los elementos y las fases de generación de un sencillo programa en C: escritura del código fuente, compilación y "linkado".](#)

El código fuente se crea utilizando editores de texto, que **no se han de confundir con los procesadores de texto**, los primeros simplemente trabajan con texto sin formato como negrita, tipos de letra... y los segundos almacenan además del texto, el formato, imágenes y otros elementos, por ejemplo Writer de OpenOffice/LibreOffice es un procesador de texto y no se ha de utilizar para gestionar código fuente. Si además de poder editar texto plano incluyen funcionalidades relacionadas con la programación se denominan editores de código.

Entre los editores de texto y/o código destacan:

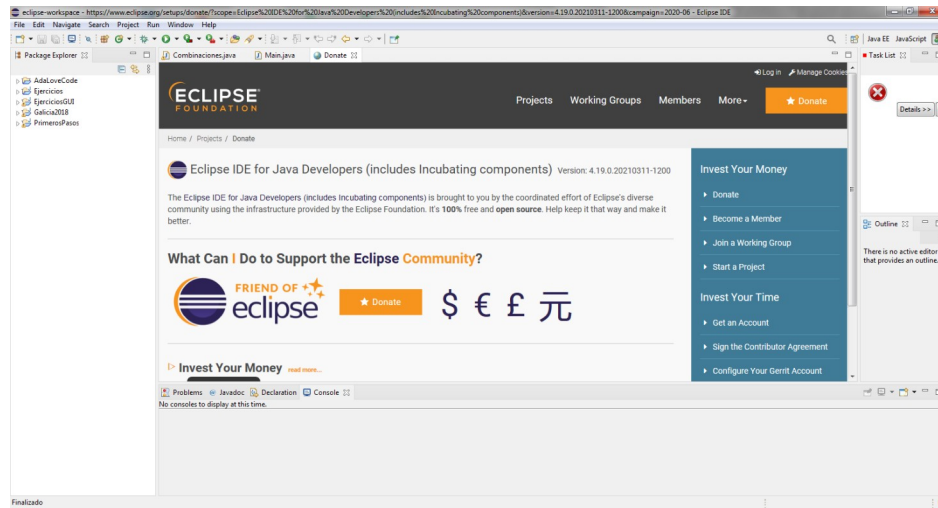
- Notepad: Editor de texto básico disponible en la familia de sistemas operativos Windows. Suele ser útil para resolver problemas puntuales como guiones bat o para pequeñas modificaciones en lenguajes interpretados.
- Notepad++: Más avanzado que el anterior, es un editor de código fuente gratuito, también disponible para Windows, posee funcionalidades para trabajar con multitud de lenguajes de programación, extensiones e integración con otras herramientas. Destaca por los bajos requerimientos de procesador y memoria. Se recomienda su uso en pequeños proyectos. Su uso se rige por la Licencia Pública General GNU y es posible descargarlo desde su web oficial <https://notepad-plus-plus.org/>
- Nano: Editor de texto texto minimalista y amigable, disponible para los sistemas operativos UNIX/Linux. Sin embargo, no solo nos permite editar texto, sino que

además tiene otras características muy interesantes que lo hacen especialmente útil para modificar archivos de configuración en la terminal o crear lanzadores.

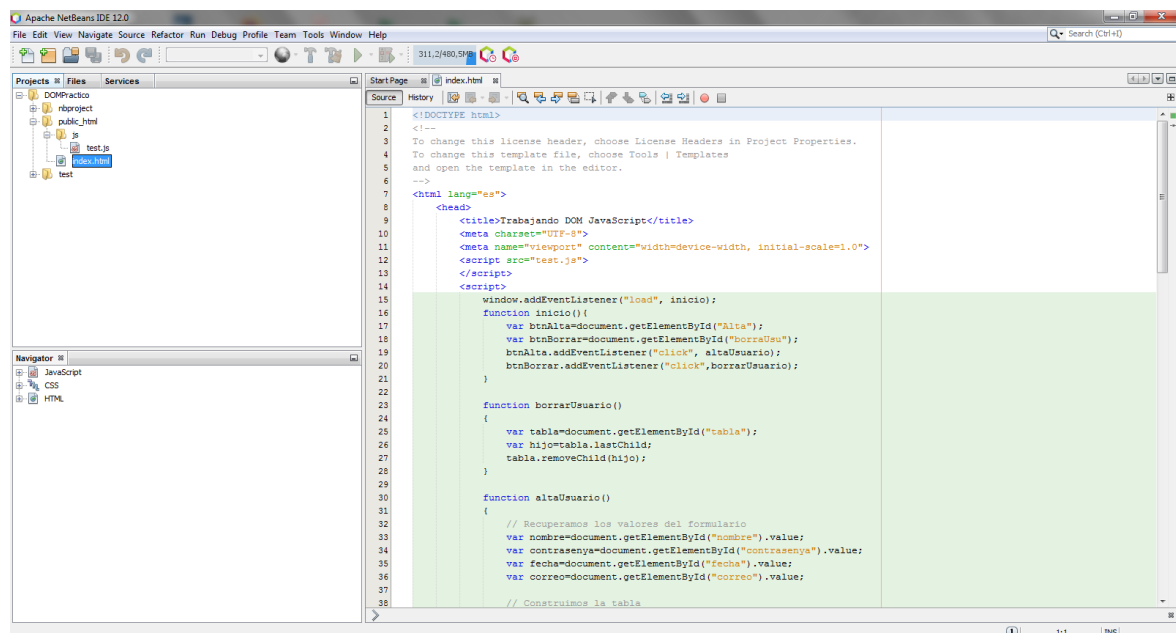
- Vi/Vim: Editor que forma parte del software estándar de los sistemas UNIX y Apple OS X, utilizándose desde el terminal. Su curva de aprendizaje es lenta, posee multitud de comandos que permiten su manejo a través del teclado. Posee un extenso número de plugins para aumentar su funcionalidad. Soporta cientos de lenguajes de programación y formatos de archivo. Y además, se integra fácilmente con muchas herramientas.
- Emacs: Competidor de Vim, forma parte del proyecto GNU. Incluye herramientas pensadas para el desarrollo de software como compilación y depuración desde el mismo entorno.

La evolución de los editores de texto son los entornos de desarrollo, que añaden nuevas funcionalidades e integran el proceso del desarrollo del software en una única herramienta, además de poder proporcionar instrumentos para el trabajo en el equipo o integración en metodologías de desarrollo, inicialmente desarrollados para un lenguaje concreto aunque en la actualidad utilizando plugins y/o extensiones se puede trabajar en prácticamente cualquier lenguaje, los más conocidos son:

- IntelliJ IDEA. El más utilizado en entornos profesionales, posee versión gratuita y de pago. Trabaja con múltiples lenguajes y entornos de trabajo, integración de herramientas como control de versiones o pruebas entre otros,. Es posible añadir extensiones. Los requisitos mínimos del equipo son altos, mínimo 2 GB y recomendado 8GB de memoria principal.
- Eclipse. El siguiente en uso, utilizado en grandes entornos como consultorías y entidades bancarias, de código abierto. Inicialmente desarrollado para Java actualmente soporta muchos lenguajes, posee sistema de plugins. Es posible descargar versiones adaptadas a las necesidades del desarrollo como proyectos Java y desarrollo Web, exclusivo para Java, C/C++ o PHP entre otros.



- **Netbeans.** Entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo. NetBeans IDE es un producto libre y gratuito sin restricciones de uso. Posee características similares a los anteriores, aunque su uso es menor. Se ha integrado como proyecto en la fundación Apache.

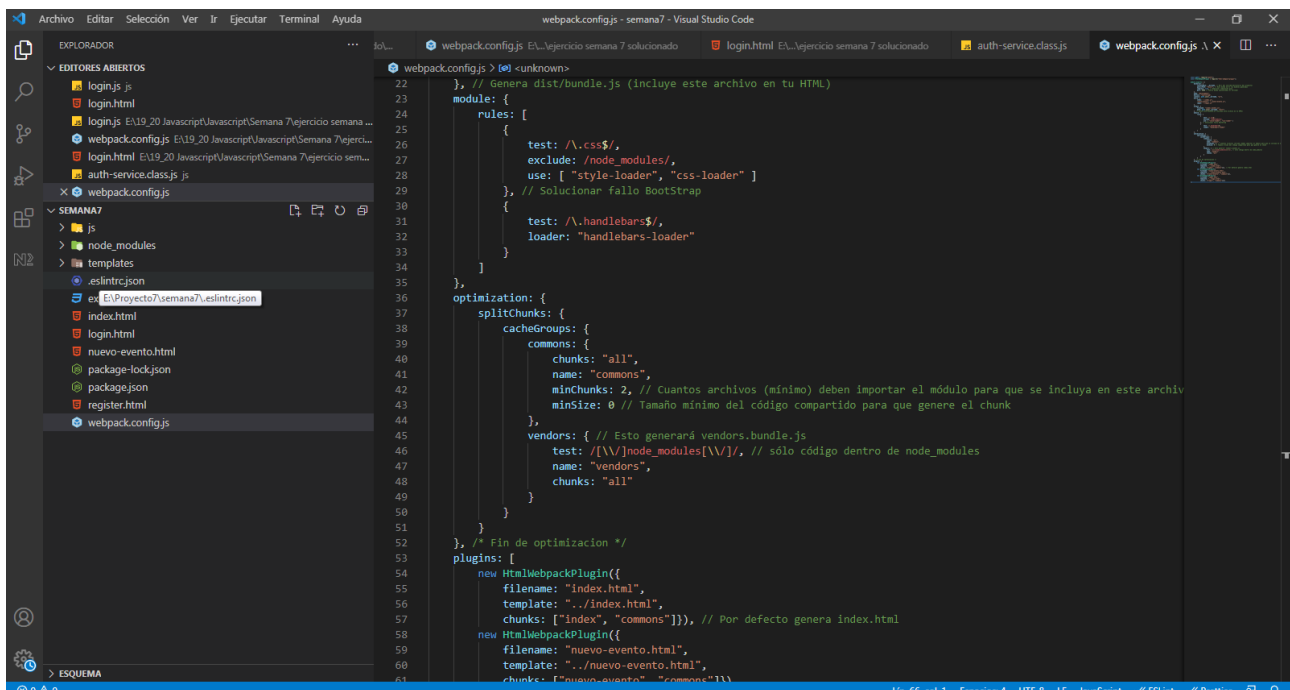


- **Visual Studio.** Entorno de desarrollo de Microsoft centrado en sus lenguajes y tecnologías, en especial C# aunque puede trabajar con otros como Python o Javascript. Entre las ventajas destaca la integración con otras herramientas como el ecosistema Azure.
- **Android Studio.** Basado en IntelliJ IDEA, su uso se centra en el desarrollo de aplicaciones para Android. Se distribuye de forma gratuita. Si bien inicialmente se desarrollaba en Java actualmente se puede desarrollar en C++ y Kotlin, el nuevo

lenguaje preferente seleccionado por Google para el desarrollo de aplicaciones para Android.

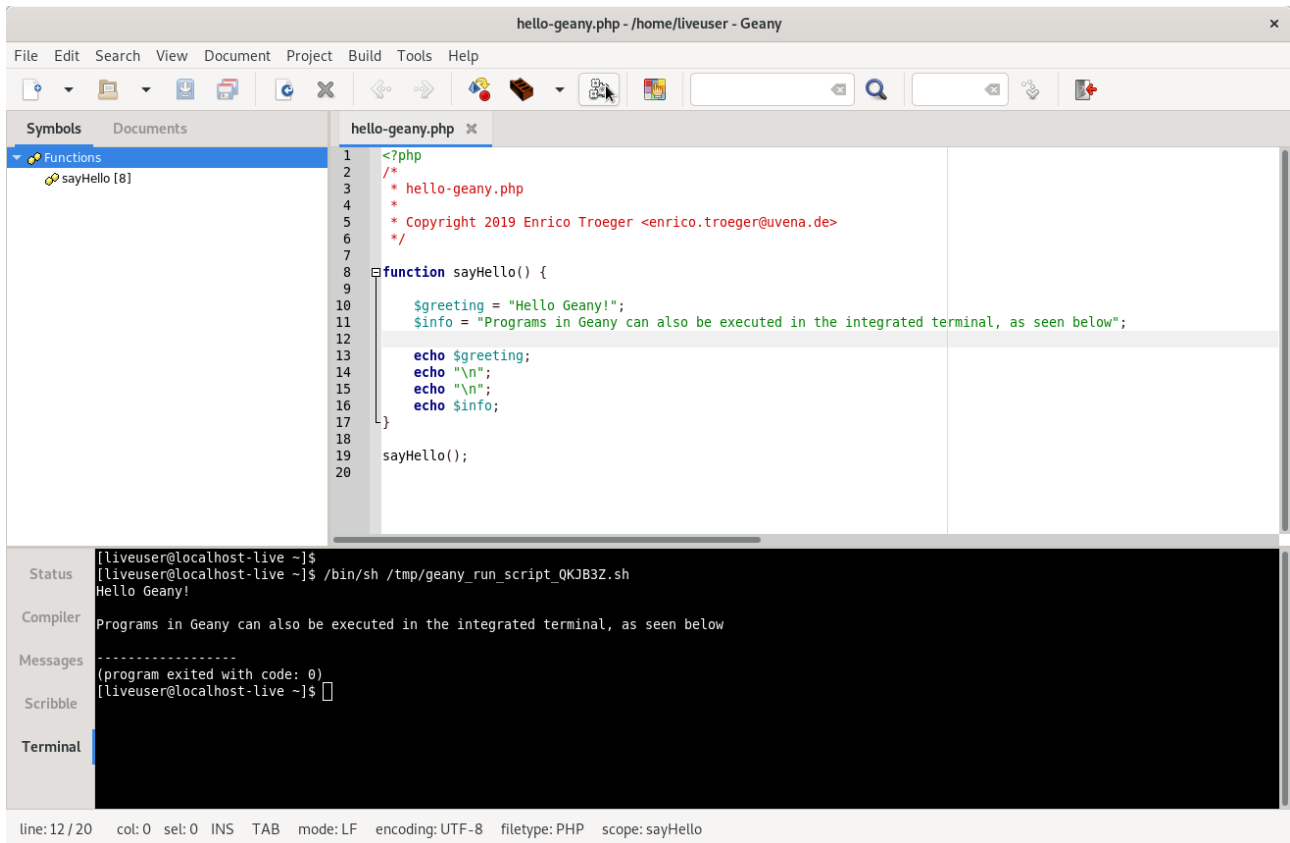
Si bien los entornos de desarrollo son muy potentes, las necesidades de procesador y memoria son altas, necesitando equipos muy potentes para poder desarrollar con comodidad. Ante esta problemática en los últimos años han aparecido herramientas que vuelven a los orígenes de los editores de texto/código con bajos requerimientos de hardware pero que poseen herramientas, funcionalidades y extensiones pensadas para el desarrollo en diferentes lenguajes, destacando:

- Visual Studio Code. Editor de código gratuito. Posee sistema de plugins, integración con gestor de código Git o depurador entre otras características. Es multiplataforma y los requisitos no son altos. Se centra en el desarrollo front-end.



- Atom. Desarrollado con Node.js, se trata de un editor de código abierto multiplataforma ideal para el desarrollo web, posee control de versiones mediante Git y ha sido desarrollado por GitHub. Posee también sistema de plugins.
- Geany. Geany es un editor de texto pequeño, potente y ligero de interfaz gráfica de usuario basado en Scintilla y bibliotecas gráficas GTK, Incluye características básica de IDE. Está diseñado para tener tiempos de carga reducidos. Se ejecuta en Linux, Windows y MacOS, está traducido a más de 40 idiomas y tiene soporte integrado para más de 50 lenguajes de programación.





```
hello-geany.php - /home/liveuser - Geany
File Edit Search View Document Project Build Tools Help
Symbols Documents
Functions
sayHello [8]
1 <?php
2 /*
3  * hello-geany.php
4  * Copyright 2019 Enrico Troeger <enrico.troeger@uvena.de>
5  */
6
7
8 function sayHello() {
9
10     $greeting = "Hello Geany!";
11     $info = "Programs in Geany can also be executed in the integrated terminal, as seen below";
12
13     echo $greeting;
14     echo "\n";
15     echo "\n";
16     echo $info;
17 }
18
19 sayHello();
20
Status [liveuser@localhost-live ~]$
[liveuser@localhost-live ~]$ /bin/sh /tmp/geany_run_script_QKJB3Z.sh
Hello Geany!
Compiler Programs in Geany can also be executed in the integrated terminal, as seen below
Messages -----
(program exited with code: 0)
[liveuser@localhost-live ~]$
Scribble
Terminal
line: 12 / 20 col: 0 sel: 0 INS TAB mode: LF encoding: UTF-8 filetype: PHP scope: sayHello
```

- Brackets. Es un editor de código fuente moderno, ligero y potente cuyo principal enfoque es el desarrollo y programación web. Creado por Adobe Systems, se trata de un software gratuito y de código abierto con licencia MIT, y actualmente se mantiene en GitHub por Adobe y otros desarrolladores. Está escrito en JavaScript, HTML y CSS.

Su interfaz clara y directa que dispone de detección automática del código y ayudas en la escritura, sangrado y coloración para identificar segmentos. Especialmente diseñado para evitar distracciones y mejorar la productividad; facilita la obtención de asistencia y ayuda cuando es necesario sin interferir con su proceso creativo.



[La elección de editor de código o entorno de desarrollo despierta pasiones entre los desarrolladores, en el siguiente vídeo se trata con un toque de humor los debates entre usuarios de vi/vim y Emacs.](#)

## 2.5. Estructura y bloques fundamentales.

Los elementos principales de un programa informático clásico se divide en varios grupos dependiendo del paradigma, del lenguaje y de la finalidad, por ejemplo en C es necesaria



la existencia de un punto de entrada llamado main, o en los primeros lenguajes de servidor como PHP se ejecutaba el fichero completo. A pesar de la variedad de elementos de un programa, prácticamente todos poseen los siguientes bloques:

- Bloque de declaraciones. Incluye la declaración y normalmente la instanciación de todos los objetos y elementos a procesar como constantes o variables.
- Bloque de instrucciones. Acciones sobre los elementos definidos en el bloque de declaración que permiten lograr el objetivo del programa. Dentro de este se puede diferenciar a su vez 3 grupos de instrucciones:
  - Entrada: Su función es obtener la información aportada desde el exterior necesaria para realizar el procesamiento, esta información se almacena en los elementos definidos en el bloque de declaraciones.
  - Proceso: Instrucciones cuya finalidad es alcanzar el objetivo del programa a partir de la información proporcionada en la entrada.
  - Salida. Una vez realizado los cálculos este bloque se encarga de almacenar, por ejemplo en un fichero o en una base de datos o mostrarlo en algún dispositivo de salida como un monitor o impresora.

En la mayoría de los lenguajes actuales, además de los bloques anteriores se tienen también:

- Espacio de nombre o paquete: Indica el ámbito o alcance del código de forma que se puedan encapsular para su posterior uso.
- Bloque de uso de elementos externos. Indica las clases o funciones externas que se van a utilizar en el programa, denominados librerías o paquetes dependiendo del lenguaje, por ejemplo C o Java y que posteriormente serán enlazados, por ejemplo en C se utiliza la palabra reservada `#include` o Java la palabra `import`.
- Bloque de definición del fichero/clase en el que se incluyen comentarios como el autor, el tipo de licencia, el uso o función del código o clase.



## 2.6. Java.

**Java como lenguaje** es uno de los más utilizados en los últimos 20 años junto a C y C++, inicialmente desarrollado para la creación de programas embebidos en dispositivos con poca capacidad de almacenamiento y cálculo y con gran variedad de arquitecturas y hardware. Fue desarrollado por la desaparecida empresa Sun Microsystem, adquirida posteriormente por Oracle.

Es un lenguaje orientado a objetos, con tipado fuerte de tipos y multiplataforma. A lo largo de su historia ha añadido otras características, como estándares diversos por ejemplo trabajar con documentos XML, los genéricos, programación concurrente, destacando Java 8 con la inclusión de la programación funcional.



En este curso se trabajará con Java SE 8, SE es de Standard Edition, existiendo otras versiones como EE de Enterprise Edition, o ME de Micro Edition para dispositivos con poca capacidad (En el sistema operativo Symbian se utilizaba

esta versión para desarrollar aplicaciones), actualmente se encuentra en la versión 16 pero no se incluyen cambios muy significativos desde la 8 y la mayor parte de las empresas continúan con esta.



Se ha indicado que una de las características de Java es la de ser multiplataforma, pero ¿Cómo lo consigue?

**Java también es la Java Virtual Machine (JVM)** una máquina virtual que se instala en cada uno de los dispositivos en los que se quiere ejecutar los programas, de forma que no se ejecutan instrucciones binarias nativas, sino que se ejecuta un **lenguaje intermedio** sobre la máquina virtual que **traduce** al lenguaje máquina del anfitrión.



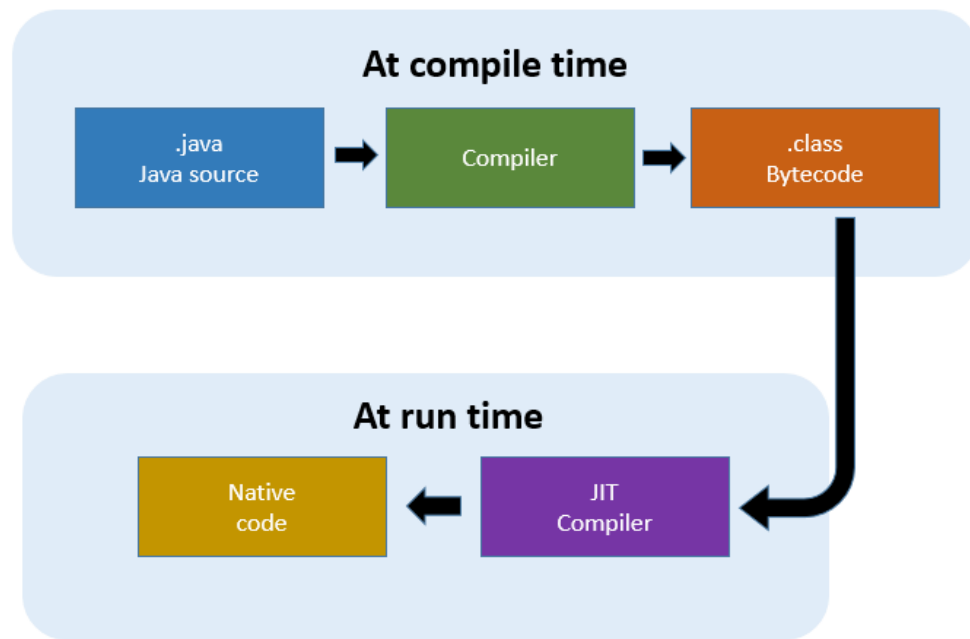
Debatir los pros y contras de esta solución en cuanto a espacio, velocidad de ejecución, portabilidad...

El lenguaje intermedio entre el de alto nivel(Java) y el binario en este caso se denomina Bytecode que se puede ejecutar en cualquier dispositivo. Java siempre ha acusado de ser lento en la ejecución ya que por ejemplo puede darse el caso de no utilizar registros de ciertas arquitecturas para trabajar con números en coma flotante (vídeo, gráficos...).



[Características Java UPV.](#)

Para paliar este problema (ha mejorado mucho desde sus inicios en velocidad de ejecución gracias al uso de el compilador JIT (Just-In-Time) que mejora el rendimiento de aplicaciones Java compilando códigos de bytes en código de máquina nativo **en tiempo de ejecución**.



**El compilador** es otro de los elementos que realiza diferentes funciones para transformar el programa escrito en Java a Bytecode, denominado Javac, que puede ser llamado desde la línea de comandos con:

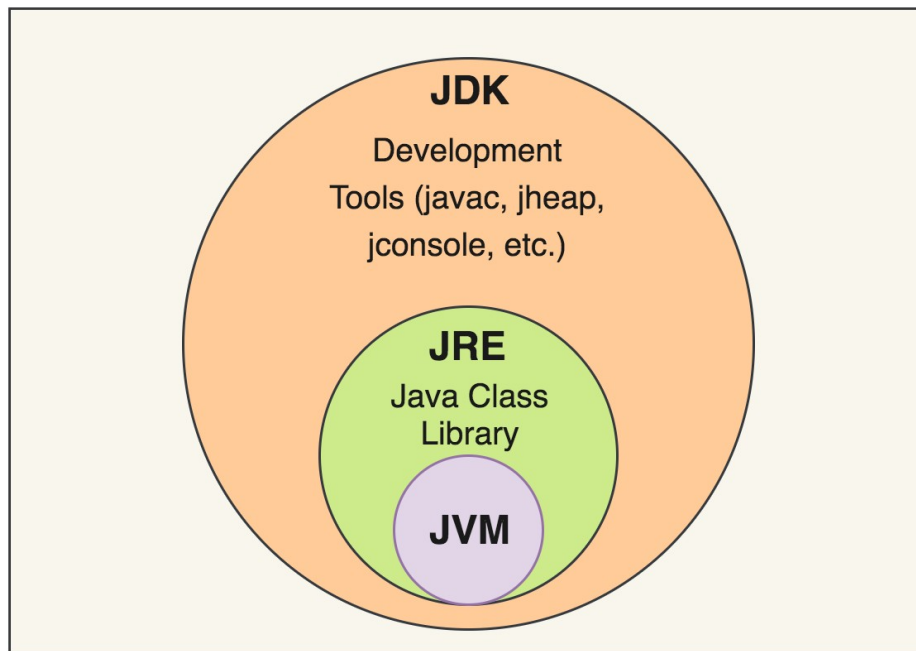
```
javac [opciones] archivo.java
```



Las opciones de Javac se pueden consultar en la página oficial de [Oracle](https://www.oracle.com/es/technetwork/java/javase-downloads-1421228.html).

El software anterior se engloba a su vez en:

- **Java Runtime Environment (JRE).** Compuesto con un conjunto de programas y utilidades que **permite la ejecución** de programas Java, principalmente la JVM y el conjunto de librerías de código que forman parte del estándar.
- **Java Development Kit (JDK).** Software necesario para en **desarrollo** de aplicaciones en Java, entre las utilidades que posee destacan:
  - Javac. Compilador.
  - Javadoc. Generación automática de documentación a partir del código (comentarios).



Gracias a la compilación a ByteCode se han desarrollado otros lenguajes además de Java que pueden utilizar la máquina virtual y el JRE como son

- Groovy.
- Scala.
- Kotlin. Muy usado en desarrollo de aplicaciones Android, puede compilar también a Javascript
- Clojure. Orientado al paradigma funcional y concurrente.
- ....



## 2.7. Variables.

Asociación de un nombre simbólico a un espacio de memoria, de forma que se puede referenciar por el nombre y no por la dirección siendo posible consultar y modificar su valor. En los lenguajes de tipado fuerte es necesario indicar el tipo de dato como C, Java o C++, en otros denominados no tipados no es necesario indicarlo como PHP o Javascript, también se ha de distinguir entre los lenguajes de

tipado dinámico y estático que indican en qué momento se realiza la comprobación de tipos, en la ejecución o en la compilación.



Razonar si la siguiente afirmación es cierta o falsa: El espacio en memoria para almacenar la edad es igual al espacio necesario para almacenar el nombre y apellidos de una persona.

Para usar una variable es necesario en primer lugar definirla, el comportamiento de la variable dependerá si va a referenciar o almacenar datos sencillos como (números, valores lógicos o caracteres) en datos referenciados, **en el primer caso al mismo tiempo que se define se instancia( se asocia un espacio de memoria y ya es posible su uso), en el segundo caso (en Java) se reserva el nombre, pero no tiene espacio de memoria asignado con lo cual no se puede usar, siendo necesario reservar y asociar el espacio a posteriori (puede ser en la misma instrucción).**

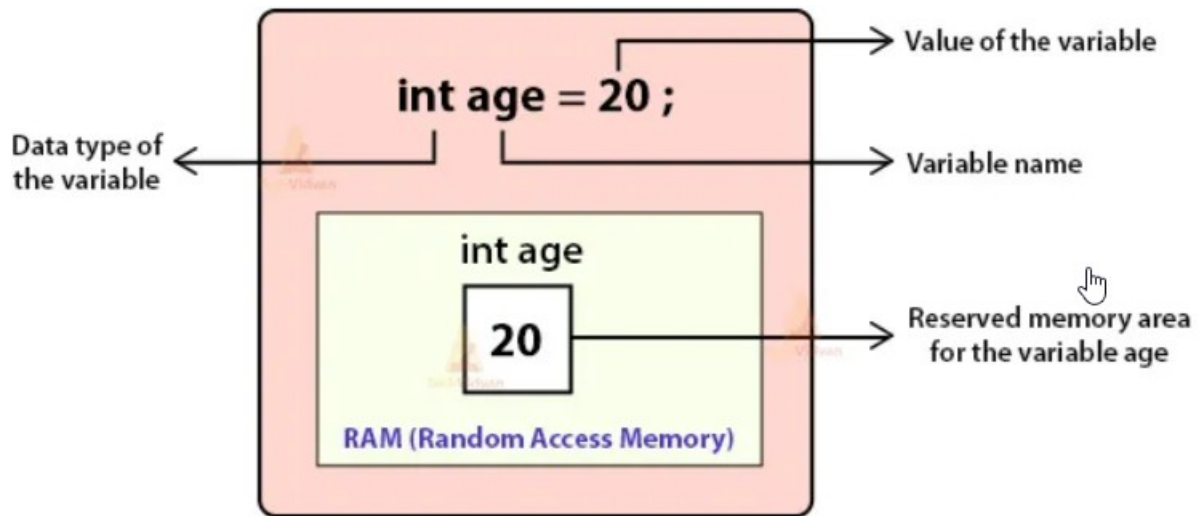
En el caso de primitivos junto con la declaración es posible la inicialización de la misma, es decir el valor que contendrá la variable por defecto (aunque posteriormente puede cambiar).

Para definir una variable se emplea la siguiente sintaxis:

```
tipo_de_dato nombre_variable;
```

Si se desea inicializar al mismo tiempo que se crea:

```
tipo_de_dato nombre_variable=valor_inicial;
```



En el caso de variables referenciadas se tiene que indicar explícitamente que se desea reservar espacio de memoria, en el caso de Java, se utiliza el operador `new`.

Para definir se sigue declarando de la misma forma:

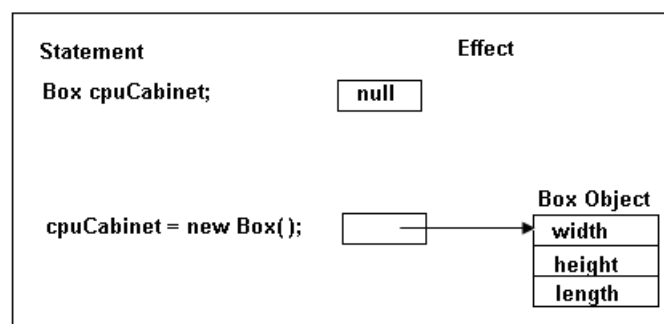
```
tipo_de_dato nombre_variable;
```

Se puede a continuación reservar el espacio:

```
nombre_variable= new tipo_de_dato();
```

O realizarlo en la misma línea de código:

```
tipo_de_dato nombre_variable= new tipo_de_dato();
```



## 2.7.1 Tipos de datos.

Cuando se define una variable y se instancia (en el caso de tipos primitivos se realiza al mismo tiempo) se reserva espacio en la memoria RAM, los lenguajes fuertemente tipado reservan un espacio concreto dependiendo del tipo de variable, siendo los tipos primitivos en Java y sus tamaños los siguientes:

- **byte**: 8 bits almacenados en complemento a 2, puede almacenar valores desde el -127 al 128.
- **short**: 16 bits en complemento a 2, valores entre el -32768 y el 32767.
- **int**: Almacenar un valor entero, posee 32 bits y se almacena en complemento a 2.
- **long**: Para números enteros de 64 bits mayores o iguales que 0.
- **float**: Se reserva una memoria de 32 bits almacenando el número real en formato IEEE 754
- **double**: Ampliación a 64 bits del tipo float, con las mismas características.
- **boolean**: Almacena valores ciertos o falsos, solo puede tener dos valores 0/1, cierto/falso.
- **char**: Representa un carácter en formato unicode de 16 bits.

En caso de no indicarse valor inicial, la variable contiene los siguientes valores:

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
boolean	false

En los tipos numéricos se pueden utilizar diferentes sistemas de numeración para realizar operaciones o inicializar, por ejemplo en caso de enteros:

```
// The number 26, in decimal
int decVal = 26;

// The number 26, in hexadecimal
int hexVal = 0x1a;

// The number 26, in binary
int binVal = 0b11010;
```

O en el caso de reales:

```
double d1 = 123.4;
```



```
// same value as d1, but in scientific notation  
double d2 = 1.234e2;  
float f1 = 123.4f;
```

Para las cadenas, usar las “, y para los booleanos true, false u operaciones lógicas (se tratan posteriormente).

## 2.7.2 Operadores y expresiones.

Ya se tienen variables, ahora es necesario realizar cálculos y operaciones con las mismas usando los nombres simbólicos que referencias a las direcciones de memoria, usando las expresiones, definiéndose como:



**La combinación de constantes(se verá más adelante), variables y operadores que devuelve un valor.**

Los operadores que se pueden utilizar y la operación que realizan dependen de los tipos de datos implicados, por ejemplo el símbolo + con enteros realiza la suma y con cadenas (string, se tratarán posteriormente) concatena(une “trozos” de texto).

### 2.7.2.1 Operadores de asignación, aritméticos y unarios.

#### Operación asignación. =

En los tipos básicos permite asignar a una variable la evaluación de un literal (expresión), en el caso de tipos compuestos/clases se trata de referencias y se tratará en posteriores temas. Algunos ejemplos:

```
int cadence = 0;  
int numero = 5;  
int resultado = 5 + numero;  
boolean correcto = TRUE;
```

El operador asignación se puede usar combinado con operaciones aritméticas como +, -, \*, / con tipos numéricos, por ejemplo:

```
a += b;  
a = a + b;
```

#### Operadores aritméticos.

Se utilizan con tipos de datos int, short, long, float y double, necesita dos operandos. Son la multiplicación, división, suma, resta y resto de división entera. El resto de la división entera devuelve el resto de dividir un número entre otro, por ejemplo:

```
int dividendo=5, divisor=3, resultado;  
resultado=dividendo % divisor;
```

Da como resultado 2. Un ejemplo extraído de la documentación oficial de Java:

```
class ArithmeticDemo {  
    public static void main (String[] args) {  
        int result = 1 + 2;  
        // result is now 3  
        System.out.println("1 + 2 = " + result);  
        int original_result = result;  
  
        result = result - 1;  
        // result is now 2  
        System.out.println(original_result + " - 1 = " + result);  
        original_result = result;  
  
        result = result * 2;  
        // result is now 4  
        System.out.println(original_result + " * 2 = " + result);  
        original_result = result;  
  
        result = result / 2;  
        // result is now 2  
        System.out.println(original_result + " / 2 = " + result);  
        original_result = result;  
  
        result = result + 8;  
        // result is now 10  
        System.out.println(original_result + " + 8 = " + result);  
        original_result = result;  
  
        result = result % 7;
```

```
// result is now 3  
  
System.out.println(original_result + " % 7 = " + result);  
  
}  
  
}
```

El resultado del código anterior es:

```
1 + 2 = 3  
3 - 1 = 2  
2 * 2 = 4  
4 / 2 = 2  
2 + 8 = 10  
10 % 7 = 3
```

### Operadores unarios.

Se aplica a un operando de tipo numérico (+,-,++,--) o lógico/booleano. Los operadores + y - cambian el símbolo a positivo y negativo respectivamente, ++ y -- producen el incremento o decremento en una unidad del valor de la variable, y el operador ! Sobre una variable booleana la niega, es decir si es falso en esa operación es cierta y viceversa.

Operator	Description
+	Unary plus operator; indicates positive value (numbers are positive without this, however)
-	Unary minus operator; negates an expression
++	Increment operator; increments a value by 1
--	Decrement operator; decrements a value by 1
!	Logical complement operator; inverts the value of a boolean


Resolver en clase: ¿Qué valor tiene la variable result al realizar cada una de las instrucciones? ¿Y la variable success?

```
int result = +1;  
result--;  
result++;  
result = -result;  
boolean success = false;  
sucess!=sucess;
```

En realidad los operadores ++ y – se dividen cada uno de ellos en dos preincremento y predecremento y postincremento y postdecremento, dependiendo de si se sitúan a la izquierda o a la derecha. Ejemplo:

```
int resultado, operador=5;  
resultado= ++operador1;  
operador=5;  
resultado=operador1++;
```

En el primer caso en primer lugar se realiza el incremento y a continuación la asignación, siendo el valor de resultado 6 y operador1 también 6, pero en el segundo caso el incremento se realiza después de la asignación con lo que resultado tendrá un valor de 5 y operador1 de 6 al finalizar el literal.

 [Operadores aritméticos y unarios en C \(prácticamente iguales en Java\) de la UPV.](#)

## Operadores de igualdad y relacionales.

Estos operadores permiten realizar comparaciones entre valores y variables numéricas, devolviendo un booleano (cierto/falso). Los operadores son:

==	equal to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

 [Operaciones relacionales en C de la UPV.](#)

Se utilizan mayoritariamente en instrucciones condicionales (siguiente unidad) aunque también se pueden usar para asignar valor a una variable de tipo booleano.



**Destacar que el operador = indica asignación y el operador == comparación, es un error común que al no usarlo correctamente se produzcan errores.**

```
int var1=5, var2=10;  
boolean logico= (var1 <=var2);
```

En el código anterior logico valdrá cierto o 0.

## Operaciones condicionales o lógicos.

Este tipo de operadores evalúan expresiones lógicas o variables de tipo booleano devolviendo a su vez un valor booleano. Los operadores son.

- &&. Operador AND, todas las variables o expresiones han de ser ciertas para que el operador devuelva cierta, con que solo una sea falta, el operador devuelve falso.
- ||. Operador OR, al menos una de las variables o expresiones han de ser ciertos, solo devuelve falso si todos son falsos.

Estos operadores funcionan en versión cortocircuito, es decir en el momento que se cumple una condición que hace que el operador no pueda cambiar de valor se **deja de evaluar el resto de expresiones**. Por ejemplo en el operador AND en el momento que una expresión sea falta, se devolverá falta independientemente del resto de expresión, en el caso de OR se deja de evaluar en el momento que una de los elementos que lo componen es cierto, ya que no tiene sentido continuar evaluando.

 [Operadores lógicos en C de la UPV.](#)



Evaluar las siguientes expresiones:

```
x < 5 && x < 10  
x < 5 || x < 4  
!(x < 5 && x < 10)
```

## Operadores de bits.

Al fin y al cabo independientemente del tipo de dato su valor es almacenado en bits y bytes. Se dispone de una serie de operadores que permiten trabajar con dichos bits, estos operadores realizan operaciones lógicas a nivel de bit, siendo estos.

- &. AND realiza la operación Y a nivel de bit.
- ^. OR exclusiva o XOR
- |. OR realizado la operación O a nivel de bit.

INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

En matemáticas se sabe que las potencias se realizan antes que las multiplicaciones y divisiones y estas antes que las sumas y las restas, siendo necesario usar paréntesis para romper dicha precedencia.

En los lenguajes de programación y por supuesto en Java también existe la precedencia de operadores que indica que operaciones se realizan antes y que operaciones se realizan después, el orden es el siguiente, siendo el de mayor precedencia el que se encuentra primero en la tabla:

Operator Precedence	
Operators	Precedence
postfix	<code>expr++ expr--</code>
unary	<code>++expr --expr +expr -expr ~ !</code>
multiplicative	<code>* / %</code>
additive	<code>+ -</code>
shift	<code>&lt;&lt; &gt;&gt; &gt;&gt;&gt;</code>
relational	<code>&lt; &gt; &lt;= &gt;= instanceof</code>
equality	<code>== !=</code>
bitwise AND	<code>&amp;</code>
bitwise exclusive OR	<code>^</code>
bitwise inclusive OR	<code> </code>
logical AND	<code>&amp;&amp;</code>
logical OR	<code>  </code>
ternary	<code>? :</code>
assignment	<code>= += -= *= /= %= &amp;= ^=  = &lt;&lt;= &gt;&gt;= &gt;&gt;&gt;=</code>

En caso de desear romper la precedencia de operadores se utilizan los paréntesis, que pueden estar anidados.

### 2.7.3 Conversiones de tipo.

Es posible asignar una variable de un tipo a otra de otro tipo, pero dependiendo de los tipos implicados la conversión tendrá unas implicaciones. Dependiendo del lenguaje se permitirán o no ciertas conversiones dependiendo del tipo, aunque por ejemplo no tiene mucho sentido la conversión de booleano a otro tipo de dato primitivo, y Java en particular no lo permite.

Existen 2 tipos de conversiones:

**Ensanchamiento o promoción.** Se convierte un tipo de dato con un tamaño menor o igual a otro con un tamaño mayor, por ejemplo de short a int. La implicación que tiene es que se ocupa un mayor tamaño, no perdiéndose los datos originales.

TIPO ORIGEN	TIPO DESTINO
byte	double, float, long, int, char, short
short	double, float, long, int
char	double, float, long, int
int	double, float, long
long	double, float
float	Double

**byte -> short -> int -> long -> float -> double**

**Estrechamiento o conversión.** En este caso se pasa de una variable con un tipo de dato que ocupa **más espacio en memoria** que al que se quiere convertir, esto implica **pérdida de información y se ha de realizar de forma implícita**, colocando el tipo al que se desea convertir entre paréntesis

```
double myDouble = 9.78d;  
int myInt = (int) myDouble;
```

Observar que se ha de indicar entre paréntesis el tipo al que se quiere convertir, en este caso int.

Las conversiones o “casting” se puede realizar en diferentes lugares de la expresión, pero se ha de tener en cuenta que dependiendo del lugar en el que se realice puede dar resultados diferentes.

**double -> float -> long -> int -> short -> byte**



**Al realizar una operación entre variables de diferente tipo el resultado se devuelve con el tipo del de mayor tamaño.**



Ambos operandos enteros → División entera

Algún operando real → División real

Ejemplo:

- $500 / 3$  resultado 166
- $500.0 / 3$  resultado 166.667
- $500 / 3.0$  resultado 166.667
- $500.0 / 3.0$  resultado 166.667

Las conversiones entre diferentes tipos ya sean de promoción o de contracción se realizan en los siguientes casos:

- **Al realizar una asignación.** Por defecto se utiliza la promoción entre asignación del valor un tipo de dato de tamaño menor a otro mayor. En caso contrario se tiene que realizar una conversión explícita, punto 3.
- **Al realizar una operación aritmética.** Se realiza una conversión de promoción del tipo menor al tipo mayor, por ejemplo  $\text{float}/\text{int}$  el resultado es un  $\text{float}$ .
- **Explícitamente.** Se realiza una conversión de tipos de forma explícita, anteponiendo entre paréntesis el tipo de dato al que se quiere convertir.

Por ejemplo:

```
int i; double j = 2.3; i = (int) j;
```

## 2.8. Constantes.

Al igual que las variables se asocia un nombre simbólico a un área de memoria, pero en este caso solo es posible la lectura, no la modificación.

Las constantes se utilizan cuando **no se desea que sea posible la modificación del valor** del nombre simbólico, por ejemplo el número  $\pi$  es constante en cualquier aplicación.

En Java la definición de constantes es un tanto peculiar, por ejemplo en C se utiliza la palabra reservada

```
#define valor
```

Y en C++, además de la forma anterior se utiliza:

```
const double pi = 3.1415926;
```

En Java se puede utilizar el modificador **static**, que permite ser accedido de forma directa (esto se entenderá mejor al tratar objetos), puede ser utilizada en cualquier parte del programa (denominadas variables globales). Pero solo con **static** no se consigue una constante, ya que se puede modificar. Para evitar que se pueda modificar (entre otras funciones) se utiliza la palabra clave **final** que implica que esa variable no se puede modificar. Un ejemplo de definición de constante en Java:

```
static final double PI = 3.141592653589793;
```

## 2.9. Comentarios.

El desarrollo de código y especialmente la comprensión y mantenimiento del mismo puede ser una tarea compleja en pequeños proyectos pero especialmente en medianos y grandes. Todo programa debe contar con una documentación que explique los principales elementos del mismo, su instalación, configuración, ficheros...pero no menos importante (prácticamente imprescindible) es añadir comentarios en el código que indique de forma resumida que hace el código, así como fragmentos que se consideren de especial importancia o dificultad.



**El compilador/interprete o programa encargado de leer el fichero no lee los comentarios, para ellos no existen.**

Su uso más comunes son los siguientes:

- Indicaciones de modificaciones realizadas (histórico).
- Descripción del fragmento/algoritmo.
- Depuración y pruebas.
- Generación de documentación y/o elementos externos. (Javadoc)

Dependiendo del lenguaje y/o fichero.

- En los ficheros de configuración se suele comentar usando el símbolo #.
- En ensamblador se utiliza ;

- En la mayoría de los lenguajes de programación (C, C++, C#, Java o Javascript) se utilizan las dos barras para comentarios de una línea // y para varias líneas se comienza con \* (barra asterisco) y se finaliza con \*/ (asterisco barra).
- En XML se utiliza la combinación <!-- para abrir comentario y --> para finalizar.

Es posible la generación de documentación de forma automática usando la herramienta Javadoc. La documentación se puede encontrar en:

<https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html>

Un ejemplo en Netbeans:

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.mycompany.operaciones_conversiones;

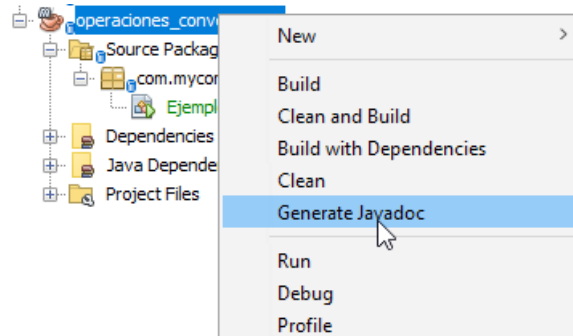
/**
 * Ejemplo de Javadoc.
 *
 * @author Pedrod
 * @version 1.0
 */
public class Ejemplo {

    /**
     * Método estático main.
     *
     */
    public static void main(String args[]) {

    }

}
```

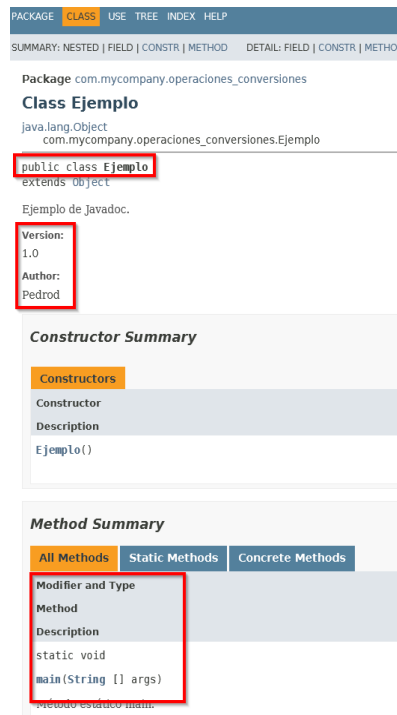
Observar el formato de los comentarios. Ahora botón derecho sobre el proyecto y Generar documentación.



Ver el lugar en que se ha generado:

View Generated javadoc at C:\Users\Pedro\Documents\NetBeansProjects\operaciones\_conversiones\target\site\apidocs\  
BUILD SUCCESS

Generando una página web con los comentarios:



### 3. Actividades y ejercicios.



Para cada uno de los lenguajes siguientes completar la tabla:

	OO	Programación funcional	Concurrencia	Tipado(débil/ fuerte)	Compilado/ Interpretado.
C					

C++					
Javascript					
C#					
Lisp					
Prolog					
Visual Basic					
PASCAL					

? Muchas veces al realizar una instalación de algún programa en Windows aparece el error de que no se encuentra instalado la versión XX del Framework .NET, a partir del artículo <https://docs.microsoft.com/es-es/dotnet/core/introduction> explicar:

- ¿En que sistemas se puede desarrollar?
- ¿En qué arquitecturas?
- ¿Cómo piensas que es posible ejecutar en tantos sistemas y arquitecturas? Relacionar con Java.
- ¿Qué lenguajes soporta de forma nativa?
- ¿Qué es el SDK para .NET?
- ¿Qué es el CLR? ¿Cuál es su equivalente en Java?
- ¿Utiliza lenguaje intermedio? ¿Una vez compilado a lenguaje intermedio, se optimiza en el sistema/arquitectura concreta?

? Buscar el espacio que ocupan los tipos básicos/primitivos en C++.

? Explicar las diferencias entre código interpretado y código compilado.

? ¿Qué solución emplea Java? Interprete o compilación.

? ¿Qué hace Java para conseguir mayor velocidad de ejecución?

? Explicar qué es JRE.

? Nos han contratado para realizar una aplicación en Java ¿Qué debe tener instalado el cliente en el que se ejecute el programa para funcionar? ¿Y nosotros como desarrolladores?

? A partir de la web [https://en.wikipedia.org/wiki/Java\\_version\\_history](https://en.wikipedia.org/wiki/Java_version_history) indicar:

- En que versión se añadió la librería New file I/O también conocida como NIO.
- En que versión se han añadido certificados raíz o Root Certificates
- En que versión de Java se añadieron los Enumerations.

? ¿Qué es una variable?

? ¿Cómo se define una variable en Java?

? ¿En qué elemento del ordenador se almacena los valores de la memoria? ¿Qué sucede si se declaran demasiadas variables?

? ¿Por qué se tienen diferentes tipos de datos? ¿Qué se consigue con ellos?

? Indicar qué tipos de datos usar para almacenar las siguientes magnitudes.

- Temperatura para experimento científico.
- Temperatura para aire acondicionado.
- Edad.
- Saldo bancario.
- Sexo (Hombre/Mujer).
- Área de un rectángulo.
- Latitud.
- Un octeto de una dirección IPv4.

? Resolver de forma manual las siguientes expresiones:

- $5 + 4/6$
- $2^4 * 6$
- $5 \% 5 * 3$

? Sean las variables `int a=4, b=6, c=3`. Y tomando en cada expresión los valores iniciales, indicar el estado de las variables al ejecutar la operación. Por ejemplo al ejecutar la instrucción `c=a++`; se tiene que `a=5, b=6` y `c=3`.

- `a++`;
- `c=a++`;
- `c=++a`;
- `c= a % c`;
- `b= --b*a++ % b`;

? Se han definido las siguientes variables `int a=4, b=7, c=3`. Indicar el resultado de las siguientes operaciones.

- `a==b`
- `b<=c`
- `(a<b || c==3) && (a>c)`
- `!(a!=b)`

? ¿Qué significa que los operadores lógicos se evalúan en modo cortocircuito?. Explicarlo en los siguientes ejemplos con las variables `int a=4, b=7, c=3`; `boolean d=TRUE, e=FALSE`.

- `a==4 && !d && c<a`
- `a==4 || !d || c<a`

? Se han definido las siguientes variables `int a=4, b=7, c=3`. Realizar de forma manual las siguientes operaciones.

- `a & b`
- `~b`
- `a | c`
- `a ^ b`

? Indicar el resultado de las siguientes expresiones, con las siguientes variables `int a=3`; `float b=5.66`; `long c=20`; `double d=2.346`;

- `int resultado1= a/b;`
- `float resultado2= a/b;`
- `btype resultado3=(byte)d;`
- `float resultado4=c;`
- `int resultado5= (float)b;`

? ¿Por qué algunas operaciones llevan el tipo delante y otras no?

? ¿Qué es una constante? ¿Cómo se definen en C?

? Explicar qué sucede cuando se añade la palabra reservada `static` a la definición de una variable.

? Explicar qué sucede cuando se añade la palabra reservada `final` a la definición de una variable.

## 4. Bibliografía.

Tutorial básico de Java de Oracle.  
<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html>

Configuración de JVM de oracle.  
[https://docs.oracle.com/cd/E22289\\_01/html/821-1274/configuring-the-default-jvm-and-java-arguments.html](https://docs.oracle.com/cd/E22289_01/html/821-1274/configuring-the-default-jvm-and-java-arguments.html)