# Tema1: Introducción al lenguaje JAVA.

## 7.- El lenguaje de programación Java.

## 7.1.- ¿Qué y cómo es Java?

Java es un lenguaje sencillo de aprender, con una sintaxis parecida a la de C++, pero en la que se han eliminado elementos complicados y que pueden originar errores. Java es orientado a objetos, con lo que elimina muchas preocupaciones al programador y permite la utilización de gran cantidad de bibliotecas ya definidas, evitando reescribir código que ya existe. Es un lenguaje de programación creado para satisfacer nuevas necesidades que los lenguajes existentes hasta el momento no eran capaces de solventar.

Una de las principales virtudes de Java es su independencia del hardware, ya que el código que se genera es válido para cualquier plataforma. Este código será ejecutado sobre una máquina virtual denominada **Maquina Virtual Java** (MVJ o JVM – Java Virtual Machine), que interpretará el código convirtiéndolo a código específico de la plataforma que lo soporta. De este modo el programa se escribe una única vez y puede hacerse funcionar en cualquier lugar. Lema del lenguaje:***"Write once, run everywhere".***

Antes de que apareciera Java, el lenguaje C era uno de los más extendidos por su versatilidad. Pero cuando los programas escritos en C aumentaban de volumen, su manejo comenzaba a complicarse. Mediante las técnicas de programación estructurada y programación modular se conseguían reducir estas complicaciones, pero no era suficiente.

Fue entonces cuando la Programación Orientada a Objetos (POO) entra en escena, aproximando notablemente la construcción de programas al pensamiento humano y haciendo más sencillo todo el proceso. Los problemas se dividen en objetos que tienen propiedades e interactúan con otros objetos, de este modo, el programador puede centrarse en cada objeto para programar internamente los elementos y funciones que lo componen.

Las características principales de lenguaje Java se resumen a continuación:

- El código generado por el compilador Java es independiente de la arquitectura.
- Está totalmente orientado a objetos.
- Su sintaxis es similar a C y C++.
- Es distribuido, preparado para aplicaciones TCP/IP.
- Dispone de un amplio conjunto de bibliotecas.
- Es robusto, realizando comprobaciones del código en tiempo de compilación y de ejecución.
- La seguridad está garantizada, ya que las aplicaciones Java no acceden a zonas delicadas de memoria o de sistema.

## 7.2.- Breve historia.

Javasurgió en 1991 cuando un grupo de ingenieros de Sun Microsystems trataron de diseñar un nuevo lenguaje de programación destinado a programar pequeños dispositivos electrónicos. La dificultad de estos dispositivos es que cambian continuamente y para que un programa funcione en el siguiente dispositivo aparecido, hay que rescribir el código. Por eso la empresa Sun quería crear un lenguaje **independiente del dispositivo.**

Pero no fue hasta 1995 cuando pasó a llamarse **Java**, dándose a conocer al público como lenguaje de programación para computadores. Java pasa a ser un lenguaje totalmente independiente de la plataforma y a la vez potente y orientado a objetos. Esa filosofía y su facilidad para crear aplicaciones para redes TCP/IP ha hecho que sea uno de los lenguajes más utilizados en la actualidad.

El factor determinante para su expansión fue la incorporación de un intérprete Java en la versión 2.0 del navegador Web Netscape Navigator, lo que supuso una gran revuelo en Internet. A principios de 1997 apareció **Java 1.1**que proporcionó sustanciales mejoras al lenguaje. **Java 1.2**, más tarde rebautizado como **Java 2,** nació a finales de 1998.

# Tema1: Introducción al lenguaje JAVA.

El principal objetivo del lenguaje Java es llegar a ser el **nexo universal** que conecte a los usuarios con la información, esté ésta situada en el ordenador local, en un servidor Web, en una base de datos o en cualquier otro lugar.

Para el desarrollo de programas en lenguaje Java es necesario utilizar un entorno de desarrollo denominado **JDK** (Java Development Kit), que provee de un compilador y un entorno de ejecución (**JRE** – Java Run Environment) para los bytecodes generados a partir del código fuente. Al igual que las diferentes versiones del lenguaje han incorporado mejoras, el entorno de desarrollo y ejecución también ha sido mejorado sucesivamente.

**Java 2** es la tercera versión del lenguaje, pero es algo más que un lenguaje de programación, incluye los siguientes elementos:

- Un lenguaje de programación: Java.
- Un conjunto de bibliotecas estándar que vienen incluidas en la plataforma y que son necesarias en todo entorno Java. Es el Java Core.
- Un conjunto de herramientas para el desarrollo de programas, como es el compilador de bytecodes, el generador de documentación, un depurador, etc.
- Un entorno de ejecución que en definitiva es una máquina virtual que ejecuta los programas traducidos a bytecodes.

El siguiente esquema muestra los elementos fundamentales de la plataforma de desarrollo Java 2.

Actualmente hay tres ediciones de la plataforma Java 2:

- **J2SE**: Entorno de Sun relacionado con la creación de aplicaciones en lenguaje Java.
- **J2EE**: Pensada para la creación de aplicaciones Java empresariales y del lado del servidor.
- **J2ME**: Pensada para la creación de aplicaciones Java para dispositivos móviles.

## 7.3.- La POO y Java.

En Java, los datos y el código (funciones o métodos) se combinan en entidades llamadas **objetos**. El objeto tendrá un comportamiento (su código interno) y un estado (los datos). Los objetos permiten la reutilización del código y pueden considerarse, en sí mismos, como piezas reutilizables en múltiples proyectos distintos. Esta característica permite reducir el tiempo de desarrollo de software.

Por simplificar un poco las cosas, un programa en Java será como una representación teatral en la que debemos preparar primero cada personaje, definir sus características y qué va a saber hacer. Cuando esta fase esté terminada, la obra se desarrollará sacando personajes a escena y haciéndoles interactuar.

Al emplear los conceptos de la Programación Orientada a Objetos (POO), Java incorpora las tres características propias de este paradigma: **encapsulación**, **herencia** y **polimorfismo**. Los patrones o tipos de objetos se denominan **clases** y los objetos que utilizan estos patrones o pertenecen a dichos tipos, se identifican con el nombre de **instancias**. Pero, no hay que alarmarse, estos conceptos se verán más adelante en sucesivas unidades.

Otro ejemplo para seguir aclarando ideas, piensa en los bloques de juegos de construcción. Suponemos que conoces los cubos de plástico en varios colores y tamaños. Por una de sus caras disponen de pequeños conectores circulares y en otra de sus caras pequeños orificios en los que pueden conectarse otros bloques, con el objetivo principal de permitir construir formas más grandes. Si usas diferentes piezas del lego puedes construir aviones, coches, edificios, etc. Si te fijas bien, cada pieza es un objeto pequeño que puede unirse con otros objetos para crear objetos más grandes.

Pues bien, aproximadamente así es como funciona la programación dirigida a objetos: unimos elementos pequeños para construir otros más grandes. Nuestros programas estarán formados por muchos componentes (objetos) independientes y diferentes; cada uno con una función determinada en nuestro software y que podrá comunicarse con los demás de una manera predefinida.

## 7.4.- Independencia de la plataforma y trabajo en red.

Existen dos características que distinguen a **Java** de otros lenguajes, como son la **independencia de la plataforma** y la posibilidad de trabajar en red o, mejor, la posibilidad de **crear aplicaciones que trabajan en red**.

Estas características las vamos a explicar a continuación:

a. **Independencia:** Los programas escritos en Java pueden ser ejecutados en cualquier tipo de hardware. El código fuente es compilado, generándose el código conocido como **Java Bytecode** (instrucciones máquina simplificadas que son específicas de la plataforma Java), el bytecode será interpretado y ejecutado en la **Máquina Virtual Java (MVJ o JVM – Java Virtual Machine)** que es un programa escrito en código nativo de la plataforma destino entendible por el hardware. Con esto se evita tener que realizar un programa diferente para cada CPU o plataforma.

   Por tanto, la parte que realmente es dependiente del sistema es la Máquina Virtual Java, así como las librerías o bibliotecas básicas que permiten acceder directamente al hardware de la máquina.

b. **Trabajo en red:** Esta capacidad del lenguaje ofrece múltiples posibilidades para la comunicación vía TCP/IP. Para poder hacerlo existen librerías que permiten el acceso y la interacción con protocolos como http, ftp, etc., facilitando al programador las tareas del tratamiento de la información a través de redes.

## 7.5.- Seguridad y simplicidad.

Junto a las características diferenciadoras del lenguaje Java relacionadas con la independencia y el trabajo en red, han de destacarse dos virtudes que hacen a este lenguaje uno de los más extendidos entre la comunidad de programadores: su seguridad y su simplicidad.

a. **Seguridad:** En primer lugar, los posibles accesos a zonas de memoria “sensibles” que en otros lenguajes como C y C++ podían suponer peligros importantes, se han eliminado en Java.

   En segundo lugar, el código Java es comprobado y verificado para evitar que determinadas secciones del código produzcan efectos no deseados. Los test que se aplican garantizan que las operaciones, operandos, conversiones, uso de clases y demás acciones son seguras.

   Y en tercer lugar, Java no permite la apertura de ficheros en la máquina local, tampoco permite ejecutar ninguna aplicación nativa de una plataforma e impide que se utilicen otros ordenadores como puente, es decir, nadie puede utilizar nuestra máquina para hacer peticiones o realizar operaciones con otra.

   En definitiva, podemos afirmar que Java es un lenguaje seguro.

b. **Simplicidad:** Aunque Java es tan potente como C o C++, es bastante más sencillo. Posee una curva de aprendizaje muy rápida y, para alguien que comienza a programar en este lenguaje, le resulta relativamente fácil comenzar a escribir aplicaciones interesantes.

   Si has programado alguna vez en C o C++ encontrarás que Java te pone las cosas más fáciles, ya que se han eliminado: la aritmética de punteros, los registros, la definición de tipos, la gestión de memoria, etc. Con esta simplificación se reduce bastante la posibilidad de cometer errores

comunes en los programas. Un programador experimentado en C o C++ puede cambiar a este lenguaje rápidamente y obtener resultados en muy poco espacio de tiempo.

Muy relacionado con la simplicidad que aporta Java está la incorporación de un elemento muy útil como es el **Recolector de Basura (Garbage collector)**. Permite al programador liberarse de la gestión de la memoria y hace que ciertos bloques de memoria puedan reaprovecharse, disminuyendo el número de huecos libres (fragmentación de memoria).

Cuando realicemos programas, crearemos objetos, haremos que éstos interaccionen, etc. Todas estas operaciones requieren de uso de memoria del sistema, pero la gestión de ésta será realizada de manera transparente al programador. Todo lo contrario que ocurría en otros lenguajes. Podremos crear tantos objetos como solicitemos, pero nunca tendremos que destruirlos. El entorno de Java borrará los objetos cuando determine que no se van a utilizar más. Este proceso es conocido como recolección de basura.

## 7.6.- Java y los Bytecodes.

Un programa escrito en Java no es directamente ejecutable, es necesario que el código fuente sea interpretado por la Maquina Virtual Java. ¿Cuáles son los pasos que se siguen desde que se genera el código fuente hasta que se ejecuta? A continuación se detallan cada uno de ellos.

Una vez escrito el código fuente (archivos con extensión **`.Java`**), éste es precompilado generándose los códigos de bytes, Bytecodes o Java Bytecodes (archivos con extensión **`.class`**) que serán interpretados directamente por la Maquina Virtual Java y traducidos a código nativo de la plataforma sobre la que se esté ejecutando el programa.

**Bytecode:** Son un conjunto de instrucciones en lenguaje máquina que no son específicas a ningún procesador o sistema de cómputo. Un intérprete de código de bytes (bytecodes) para una plataforma específica será quien los ejecute. A estos intérpretes también se les conoce como Máquinas Virtuales Java o intérpretes Java de tiempo de ejecución.

En el proceso de precompilación, existe un verificador de códigos de bytes que se asegurará de que se cumplen las siguientes condiciones:

- El código satisface las especificaciones de la Máquina Virtual Java.
- No existe amenaza contra la integridad del sistema.
- No se producen desbordamientos de memoria.
- Los parámetros y sus tipos son adecuados.
- No existen conversiones de datos no permitidas.

Para que un bytecode pueda ser ejecutado en cualquier plataforma, es imprescindible que dicha plataforma cuente con el intérprete adecuado, es decir, la máquina virtual específica para esa plataforma. En general, la Máquina Virtual Java es un programa de reducido tamaño y gratuito para todos los sistemas operativos.

## 8.- Programas en Java.

Hasta ahora, hemos descrito el lenguaje de programación Java, hemos hecho un recorrido por su historia y nos hemos instruido sobre su filosofía de trabajo, pero te preguntarás ¿Cuándo empezamos a desarrollar programas? ¿Qué elementos forman parte de un programa en Java? ¿Qué se necesita para programar en este lenguaje? ¿Podemos crear programas de diferente tipo?

No te impacientes, cada vez estamos más cerca de comenzar la experiencia con el lenguaje de programación Java. Iniciaremos nuestro camino conociendo cuales son los elementos básicos de un programa Java, la forma en que debemos escribir el código y los tipos de aplicaciones que pueden crearse en este lenguaje.

# Tema1: Introducción al lenguaje JAVA.

## 8.1.- Estructura de un programa.

En el gráfico al que puedes acceder a continuación, se presenta la estructura general de un programa realizado en un lenguaje orientado a objetos como es Java.

Vamos a analizar cada uno de los elementos que aparecen en dicho gráfico:

**`public class Clase_Principal:`** Todos los programas han de incluir una clase como esta. Es una clase general en la que se incluyen todos los demás elementos del programa. Entre otras cosas, contiene el método o función **`main()`** que representa al programa principal, desde el que se llevará a cabo la ejecución del programa. Esta clase puede contener a su vez otras clases del usuario, pero sólo una puede ser **`public`**. El nombre del fichero **`.Java`** que contiene el código fuente de nuestro programa, coincidirá con el nombre de la clase que estamos describiendo en estas líneas.

**Recomendación**

Ten en cuenta que **Java distingue entre mayúsculas y minúsculas**. Si le das a la clase principal el nombre PrimerPrograma, el archivo **`.Java`** tendrá como identificador **PrimerPrograma.Java**, que es totalmente diferente a primerprograma.Java. Además, para Java los elementos PrimerPrograma y primerprograma serían considerados dos clases diferentes dentro del código fuente.

- **`public static void main (String[] args)`**: Es el método que representa al programa principal, en él se podrán incluir las instrucciones que estimemos oportunas para la ejecución del programa. Desde él se podrá hacer uso del resto de clases creadas. Todos los programas Java tienen un método **`main`**.

- **Comentarios**: Los comentarios se suelen incluir en el código fuente para realizar aclaraciones, anotaciones o cualquier otra indicación que el programador estime oportuna. Estos comentarios pueden introducirse de dos formas, **con `//`** y **con `/* */`**. Con la primera forma estaríamos estableciendo una única línea completa de comentario y, con la segunda, con **`/*`** comenzaríamos el comentario y éste no terminaría hasta que no insertáramos **`*/`**.

- **Bloques de código**: son conjuntos de instrucciones que se marcan mediante la apertura y cierre de llaves **`{ }`**. El código así marcado es considerado interno al bloque.

- **Punto y coma**: aunque en el ejemplo no hemos incluido ninguna línea de código que termine con punto y coma, hay que hacer hincapié en que cada línea de código ha de terminar con punto y coma (**`;`**). En caso de no hacerlo, tendremos errores sintácticos.

## 8.2.- El entorno básico de desarrollo Java.

Ya conoces cómo es la estructura de un programa en Java, pero, ¿qué necesitamos para llevarlo a la práctica? La herramienta básica para empezar a desarrollar aplicaciones en Java es el **JDK (Java Development Kit o Kit de Desarrollo Java)**, que incluye un compilador y un intérprete para línea de comandos. Estos dos programas son los empleados en la precompilación e interpretación del código.

Como veremos, existen diferentes entornos para la creación de programas en Java que incluyen multitud de herramientas, pero por ahora nos centraremos en el entorno más básico, extendido y gratuito, el Java Development Kit (JDK). Según se indica en la propia página web de Oracle, JDK es un entorno de desarrollo para construir aplicaciones, applets y componentes utilizando el lenguaje de programación Java. Incluye herramientas útiles para el desarrollo y prueba de programas escritos en Java y ejecutados en la Plataforma Java.

Así mismo, junto a JDK se incluye una implementación del entorno de ejecución Java, el **JRE (Java Runtime Environment)** para ser utilizado por el JDK. El JRE incluye la Máquina Virtual de Java (MVJ ó JVM – Java Virtual Machine), bibliotecas de clases y otros ficheros que soportan la ejecución de programas escritos en el lenguaje de programación Java.

## 8.3.- La API de Java.

Junto con el kit de desarrollo que hemos descargado e instalado anteriormente, vienen incluidas gratuitamente todas las bibliotecas de la API (Aplication Programming Interface – Interfaz de programación de aplicaciones) de Java, es lo que se conoce como Bibliotecas de Clases Java. Este conjunto de bibliotecas proporciona al programador paquetes de clases útiles para la realización de múltiples tareas dentro de un programa. Está organizada en paquetes lógicos, donde cada paquete contiene un conjunto de clases relacionadas semánticamente.

En décadas pasadas una biblioteca era un conjunto de programas que contenían cientos de rutinas (una rutina es un procedimiento o función bien verificados, en determinado lenguaje de programación). Las rutinas de biblioteca manejaban las tareas que todos o casi todos los programas necesitaban. El programador podía recurrir a esta biblioteca para desarrollar programas con rapidez.

Una biblioteca de clases es un conjunto de clases de programación orientada a objetos. Esas clases contienen métodos que son útiles para los programadores. En el caso de Java cuando descargamos el JDK obtenemos la biblioteca de clases API. Utilizar las clases y métodos de las APIs de Java reduce el tiempo de desarrollo de los programas. También, existen diversas bibliotecas de clases desarrolladas por terceros que contienen componentes reutilizables de software, y están disponibles a través de la Web.

## 8.4.- Afinando la configuración.

Para que podamos compilar y ejecutar ficheros Java es necesario que realicemos unos pequeños ajustes en la configuración del sistema. Vamos a indicarle dónde encontrar los ficheros necesarios para realizar las labores de compilación y ejecución, en este caso **`Javac.exe`** y **`Java.exe`**, así como las librerías contenidas en la API de Java y las clases del usuario.

**La variable PATH:** Como aún no disponemos de un IDE (Integrated Development Environment - Entono Integrado de Desarrollo) la única forma de ejecutar programas es a través de línea de comandos. Pero sólo podremos ejecutar programas directamente si la ruta hacia ellos está indicada en la variable PATH del ordenador. Es necesario que incluyamos la ruta hacia estos programas en nuestra variable PATH. Esta ruta será el lugar donde se instaló el JDK hasta su directorio **`bin`**.

**La variable `CLASSPATH`**: esta variable de entorno establece dónde buscar las clases o bibliotecas de la API de Java, así como las clases creadas por el usuario. Es decir, los ficheros **`.class`** que se obtienen una vez compilado el código fuente de un programa escrito en Java. Es posible que en dicha ruta existan directorios y ficheros comprimidos en los formatos **`zip`** o **`jar`**que pueden ser utilizados directamente por el JDK, conteniendo en su interior archivos con extensión **`class`**.

(Por ejemplo: **`C:\Program Files\Java\jdk1.6.0_25\bin`**)

Si no existe la variable **CLASSPATH** debes crearla, para modificar su contenido sigue el mismo método que hemos empleado para la modificación del valor de la variable **`PATH`**, anteriormente descrito. Ten en cuenta que la ruta que debes incluir será el lugar donde se instaló el JDK hasta su directorio **`lib`**.

(Por ejemplo: **`C:\Program Files\Java\jdk1.6.0_25\lib`**)

# Tema1: Introducción al lenguaje JAVA.

## 8.5.- Codificación, compilación y ejecución de aplicaciones.

Una vez que la configuración del entorno Java está completada y tenemos el código fuente de nuestro programa escrito en un archivo con extensión, **`Java`**, la compilación de aplicaciones se realiza mediante el programa **Javac** incluido en el software de desarrollo de Java.

Para llevar a cabo la compilación desde la línea de comandos, escribiremos:

```
Javac archivo.Java
```

Donde **`Javac`** es el compilador de Java y **`archivo.Java`** es nuestro código fuente.

El resultado de la compilación será un archivo con el mismo nombre que el archivo Java pero con la **extensión `class`**. Esto ya es el archivo con el código en forma de bytecode. Es decir con el código precompilado. Si en el código fuente de nuestro programa figuraran más de una clase, veremos como al realizar la compilación se generarán tantos archivos con extensión **`.class`** como clases tengamos. Además, si estas clases tenían método**`main`** podremos ejecutar dichos archivos por separado para ver el funcionamiento de dichas clases.

Para que el programa pueda ser ejecutado, siempre y cuando esté incluido en su interior el método main, **podremos utilizar el interprete incluido en el kit de desarrollo.**

**La ejecución de nuestro programa desde la línea de comandos podremos hacerla escribiendo:**

```
Java archivo.class
```

Donde **`Java`** es el intérprete y **`archivo.class`** es el archivo con el código precompilado.

## 8.6.- Tipos de aplicaciones en Java.

La versatilidad del lenguaje de programación Java permite al programador crear distintos tipos de aplicaciones. A continuación, describiremos las características más relevantes de cada uno de ellos:

- **Aplicaciones de consola:**
  - Son programas independientes al igual que los creados con los lenguajes tradicionales.
  - Se componen como mínimo de un archivo `.class` que debe contar necesariamente con el método `main`.
  - No necesitan un navegador web y se ejecutan cuando invocamos el comando Java para iniciar la Máquina Virtual de Java (JVM). De no encontrarse el método `main` la aplicación no podrá ejecutarse.
  - Las aplicaciones de consola leen y escriben hacia y desde la entrada y salida estándar, sin ninguna interfaz gráfica de usuario.

- **Aplicaciones gráficas:**
  - Aquellas que utilizan las clases con capacidades gráficas, como Swing que es la biblioteca para la interfaz gráfica de usuario avanzada de la plataforma Java SE.
  - Incluyen las instrucciones **`import`**, que indican al compilador de Java que las clases de Swing o JAVAFX se incluyan en la compilación.

- **Applets:**
  - Son programas incrustados en otras aplicaciones, normalmente una página web que se muestra en un navegador. Cuando el navegador carga una web que contiene unapplet, éste se descarga en el navegador web y comienza a ejecutarse. Esto nos permite crear programas que cualquier usuario puede ejecutar con tan solo cargar la página web en su navegador.
  - Se pueden descargar de Internet y se observan en un navegador. Los applets se descargan junto con una página HTML desde un servidor web y se ejecutan en la máquina cliente.

  - No tienen acceso a partes sensibles (por ejemplo: no pueden escribir archivos), a menos que uno mismo le dé los permisos necesarios en el sistema.
  - No tienen un método principal.
  - Son multiplataforma y pueden ejecutarse en cualquier navegador que soporte Java.

- **Servlets:**
  - Son componentes de la parte del servidor de Java EE, encargados de generar respuestas a las peticiones recibidas de los clientes.
  - Los servlets, al contrario de los applets, son programas que están pensados para trabajar en el lado del servidor y desarrollar aplicaciones Web que interactúen con los clientes.

- **Midlets:**
  - Son aplicaciones creadas en Java para su ejecución en sistemas de propósito simple o dispositivos móviles. Los juegos Java creados para teléfonos móviles son midlets.
  - Son programas creados para dispositivos embebidos (se dedican a una sola actividad), más específicamente para la máquina virtual Java MicroEdition (Java ME).
  - Generalmente son juegos y aplicaciones que se ejecutan en teléfonos móviles.

## 9.- Entornos Integrados de Desarrollo (IDE).

En los comienzos de Java la utilización de la línea de comandos era algo habitual. El programador escribía el código fuente empleando un editor de texto básico, seguidamente, pasaba a utilizar un compilador y con él obtenía el código compilado. En un paso posterior, necesitaba emplear una tercera herramienta para el ensamblado del programa. Por último, podía probar a través de la línea de comandos el archivo ejecutable. El problema surgía cuando se producía algún error, lo que provocaba tener que volver a iniciar el proceso completo.

Estas circunstancias hacían que el desarrollo de software no estuviera optimizado. Con el paso del tiempo, se fueron desarrollando aplicaciones que incluían las herramientas necesarias para realizar todo el proceso de programación de forma más sencilla, fiable y rápida. Para cada lenguaje de programación existen múltiples entornos de desarrollo, cada uno con sus ventajas e inconvenientes. Dependiendo de las necesidades de la persona que va a programar, la facilidad de uso o lo agradable que le resulte trabajar con él, se elegirá entre unos u otros entornos.

Para el lenguaje de programación Java existen múltiples alternativas, siendo los principales entornos de desarrollo **NetBeans** (que cuenta con el apoyo de la empresa Sun), **Eclipse y JCreator**. Los dos primeros son gratuitos, con soporte de idiomas y multiplataforma (Windows, Linux, MacOS).

¿Y cuál será con el que vamos a trabajar? El entorno que hemos seleccionado llevar a cabo nuestros desarrollos de software en este módulo profesional será **NetBeans,** al haber sido construido por la misma compañía que creó Java, ser de código abierto y ofrecer capacidades profesionales. Aunque, no te preocupes, también haremos un recorrido por otros entornos destacables.

### 9.1.- ¿Qué son?

Son aplicaciones que ofrecen la posibilidad de llevar a cabo el proceso completo de desarrollo de software a través de un único programa. Podremos realizar las labores de edición, compilación, depuración, detección de errores, corrección y ejecución de programas escritos en Java o en otros lenguajes de programación, bajo un entorno gráfico (no mediante línea de comandos). Junto a las capacidades descritas, cada entorno añade otras que ayudan a realizar el proceso de programación, como por ejemplo: código fuente coloreado, plantillas para diferentes tipos de aplicaciones, creación de proyectos, etc.

Hay que tener en cuenta que un entorno de desarrollo no es más que una fachada para el proceso de compilación y ejecución de un programa. ¿Qué quiere decir eso? Pues que si tenemos instalado un IDE y no tenemos instalado el compilador, no tenemos nada.

# Tema1: Introducción al lenguaje JAVA.

## 9.2.- Algunos de los IDE's actuales.

Existen en el mercado multitud de entornos de desarrollo para el lenguaje Java, los hay de libre distribución, de pago, para principiantes, para profesionales, que consumen más recursos, que son más ligeros, más amigables, más complejos que otros, etc.

Entre los que son gratuitos o de libre distribución tenemos:

- **NetBeans**
- **Eclipse**
- **BlueJ**
- **Jgrasp**
- **Jcreator LE**

Entre los que son propietarios o de pago tenemos:

- **IntelliJ IDEA**
- **Jbuilder**
- **Jcreator**
- **JDeveloper**

.
Pero, ¿cuál o cuáles son los más utilizados por la comunidad de programadores Java? El puesto de honor se lo disputan entre **Eclipse, IntelliJ IDEA** y**NetBeans**. En los siguientes epígrafes haremos una descripción de NetBeans y Eclipse, para posteriormente desarrollar los puntos claves del entorno NetBeans.

## 9.3.- El entorno NetBeans.

Como se ha indicado anteriormente, el entorno de desarrollo que vamos a utilizar a lo largo de los contenidos del módulo profesional será **NetBeans**. Por lo que vamos primero a analizar sus características y destacar las ventajas que puede aportar su utilización.

Se trata de un entorno de desarrollo orientado principalmente al lenguaje Java, aunque puede servir para otros lenguajes de programación. Es un producto libre y gratuito sin restricciones de uso. Es un proyecto de código abierto de gran éxito, con una comunidad de usuarios numerosa, en continuo crecimiento y apoyado por varias empresas.

El origen de este entorno hay que buscarlo en un proyecto realizado por estudiantes de la República Checa. Fue el primer IDE creado en lenguaje Java. Un tiempo más tarde, se formó una compañía que sería comprada en 1999 por Sun Microsystems (quien había creado el lenguaje Java). Poco después, Sun decidió que el producto sería libre y de código abierto y nació Netbeans como IDE de código abierto para crear aplicaciones Java.

NetBeans lleva tiempo pugnando con Eclipse por convertirse en la plataforma más importante para crear aplicaciones en Java. Hoy en día es un producto en el que participan decenas de empresas con Sun a la cabeza. Sigue siendo software libre y ofrece las siguientes posibilidades:

- Escribir código en C, C++, Ruby, Groovy, Javascript, CSS y PHP además de Java.
- Permitir crear aplicaciones J2EE gracias a que incorpora servidores de aplicaciones Java (actualmente Glassfish y Tomcat)
- Crear aplicaciones gráficas JavaFX o Swing de forma sencilla, al estilo del Visual Studio de Microsoft.
- Crear aplicaciones **JME** para dispositivos móviles.

# Tema1: Introducción al lenguaje JAVA.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de Java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo.

Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en esta plataforma pueden ser extendidas fácilmente por cualquiera que desarrolle también software.

Cada módulo provee una función bien definida, tales como el soporte de Java, edición, o soporte para el sistema de control de versiones. NetBeans contiene todos los módulos necesarios para el desarrollo de aplicaciones Java en una sola descarga, permitiendo a la persona que va a realizar el programa comenzar a trabajar inmediatamente.

Encuentra más información sobre esta plataforma en los enlaces que te proponemos a continuación:

Información oficial sobre NetBeans

Versiones del entorno NetBeans