

INTRODUCCIÓN A LA PROGRAMACIÓN	2
LENGUAJE DE PROGRAMACIÓN JAVA.	6
LA MÁQUINA VIRTUAL DE JAVA - JVM	10
CARACTERÍSTICAS DEL LENGUAJE JAVA	12
VERSIONES JAVA	13
EL KIT DE HERRAMIENTAS PARA PROGRAMAR EN JAVA:	13
APPLICATION PROGRAM INTERFACE (API) DE JAVA	13
INSTALAR JAVA EN MI EQUIPO	14
IDE (INTEGRATED DEVELOPMENT ENVIRONMENT – ENTORNOS DE DESARROLLO).	15
¿CÓMO SON LOS PROGRAMAS JAVA?	16
¿POR DÓNDE ARRANCA UN PROGRAMA JAVA? – MAIN CLASS	17
HABLAR CON JAVA	18

INTRODUCCIÓN A LA PROGRAMACIÓN

Según Wikipedia un **programa informático** es una secuencia de instrucciones, escritas para realizar una tarea específica en un computador. Cualquier dispositivo electrónico programable requiere programas para funcionar y los ejecuta cargándolos en algún tipo de memoria, y enviándoselos de forma organizada a uno o varios procesadores para su ejecución.

La **programación** es la base para el desarrollo de SOFTWARE, componente básico de todo sistema informático. Actualmente ya podríamos decir que la programación es la base de todo dispositivo electrónico, pues se programan no sólo ordenadores, sino toda clase de dispositivos móviles y también firmware “empotrado” en toda clase de dispositivos domésticos, en el mundo de la automoción, industria ... incluso las tarjetas de débito/crédito, dni electrónico, etc.

La programación en Asturias

Antes de hablar de aspectos concretos de programación vamos a contextualizar brevemente lo que significa el desarrollo de aplicaciones en nuestra región.

Empresas

En Asturias hay un importante tejido empresarial relacionado con la programación y el desarrollo de aplicaciones:

- CapGemini en Valnalón - Langreo
- DXC Technology (antes CSC) en Avilés



Sólo entre ambas se acercan actualmente a los 2000 empleos. Además, hay un importante número de Pymes que se dedican al desarrollo de aplicaciones de toda índole, con importante presencia en el **parque Científico-Tecnológico de Gijón** y el **parque tecnológico de Asturias en Llanera**.

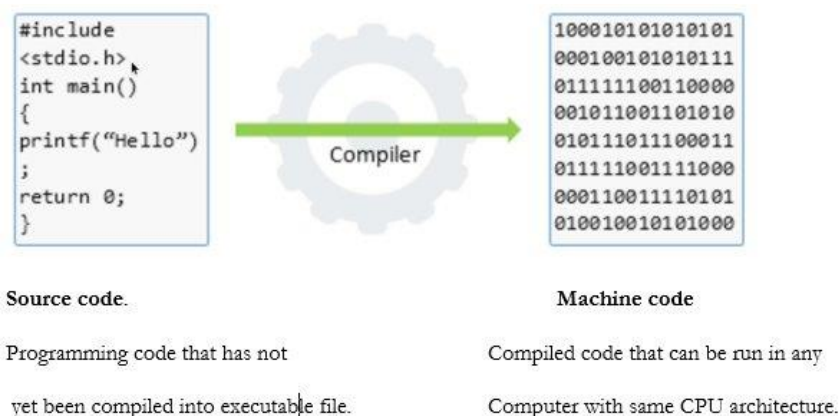


Todo programa pasa por 2 fases:

Código Fuente: Puede estar escrito (editado) en diferentes lenguajes de programación, pero en cualquier caso en esta fase el programa es **editado por y legible para** humanos expertos en el lenguaje en particular en el que está escrito. Hay multitud de lenguajes de programación que nos van a permitir desarrollar programas. Ninguno de esos lenguajes es entendible directamente por la máquina, pues han sido desarrollados precisamente para facilitar al elemento humano la tarea profesional del desarrollo de aplicaciones.

Código Ejecutable: El objetivo de todo programa. En esta fase el programa tiene un formato ejecutable que un computador o similar puede utilizar directamente para ejecutar las instrucciones: **código máquina o código intermedio**. En esta fase, el programa no es editable ni legible para humanos. Si hubiera que hacer modificaciones en el programa por mal funcionamiento, o para ampliar su funcionalidad, siempre habrá que realizar estos cambios sobre el **código fuente**.

Para que los programas desarrollados en alguno de los múltiples lenguajes disponibles puedan ser ejecutados en algún dispositivo programable, el **código fuente** ha de ser previamente **compilado/interpretado** directamente a código máquina, o **compilado a un código intermedio** y después interpretado y ejecutado como en el caso de JAVA



En el caso de los lenguajes compilados/interpretados a código máquina tenemos una fuerte dependencia del hardware. En el caso de los lenguajes compilados a código intermedio como JAVA hay un nuevo componente, llamado **máquina virtual**, que nos proporciona un nuevo nivel de abstracción.

Hay que tener en cuenta que la creación de Software no consiste únicamente en la tarea de programar en un lenguaje. Hay que Analizar los requerimientos y especificaciones de cada problema a resolver, diseñar soluciones algorítmicas y finalmente codificarlas correctamente de forma organizada. Esto es lo que se entiende normalmente como **desarrollo de aplicaciones**, y es una tarea bastante complicada pues requiere de capacidad de abstracción, lógica, y conocimiento de un buen número de herramientas y técnicas.

Siempre se ha considerado la programación como un trabajo a mitad de camino entre la técnica y la creatividad, aunque lo ideal sería atraerlo lo más posible a la pura técnica, por imperativos de efectividad y perdurabilidad del producto. Se busca que el programador sea un técnico que elabore un material casi mecánico, perfectamente evaluable y medible según criterios industriales de calidad. Para ayudar en la tarea siempre difícil de creación de un algoritmo, se idearon dos herramientas de uso muy extendido, a la vez que discutido entre la comunidad de programadores:

- Flujogramas.
- Pseudocódigo.

Sin entrar en polémicas sobre la conveniencia o no de utilizar estas técnicas para muchos desfasadas, es comúnmente aceptado el hecho de que su uso en las etapas iniciales del aprendizaje de la programación es interesante, por su alto valor pedagógico y auto-explicativo.

Algoritmo: Conjunto ordenado de pasos que permite hallar la solución de un problema dado.

Un **algoritmo** debe ser independiente del lenguaje de programación, y tiene que funcionar en diferentes dispositivos. Vale como ejemplo una receta de cocina, ya que se deben de seguir unos pasos y usar unos ingredientes, independientemente de si lo hace un cocinero inglés, francés o español, e independiente de la cocina donde se haga. Los pasos a seguir deben estar muy claros, y a partir de ahí se **implementará** la receta en el lenguaje y con los utensilios adecuados en cada caso.

Programa: Consiste en **implementar** el algoritmo en un determinado **lenguaje de programación**, para que pueda ser ejecutado en un dispositivo dotado de un sistema operativo concreto.






Los diferentes lenguajes de programación no son más que una **forma concreta** de codificar un algoritmo, compilarlo/interpretarlo y empaquetarlo para convertirlo en un **programa ejecutable** para un dispositivo concreto. El algoritmo será la solución propuesta después de un análisis previo del problema (pseudocódigo, diagrama de flujo u otros) y el **programa** será la **implementación** del **algoritmo** en un lenguaje de programación determinado.

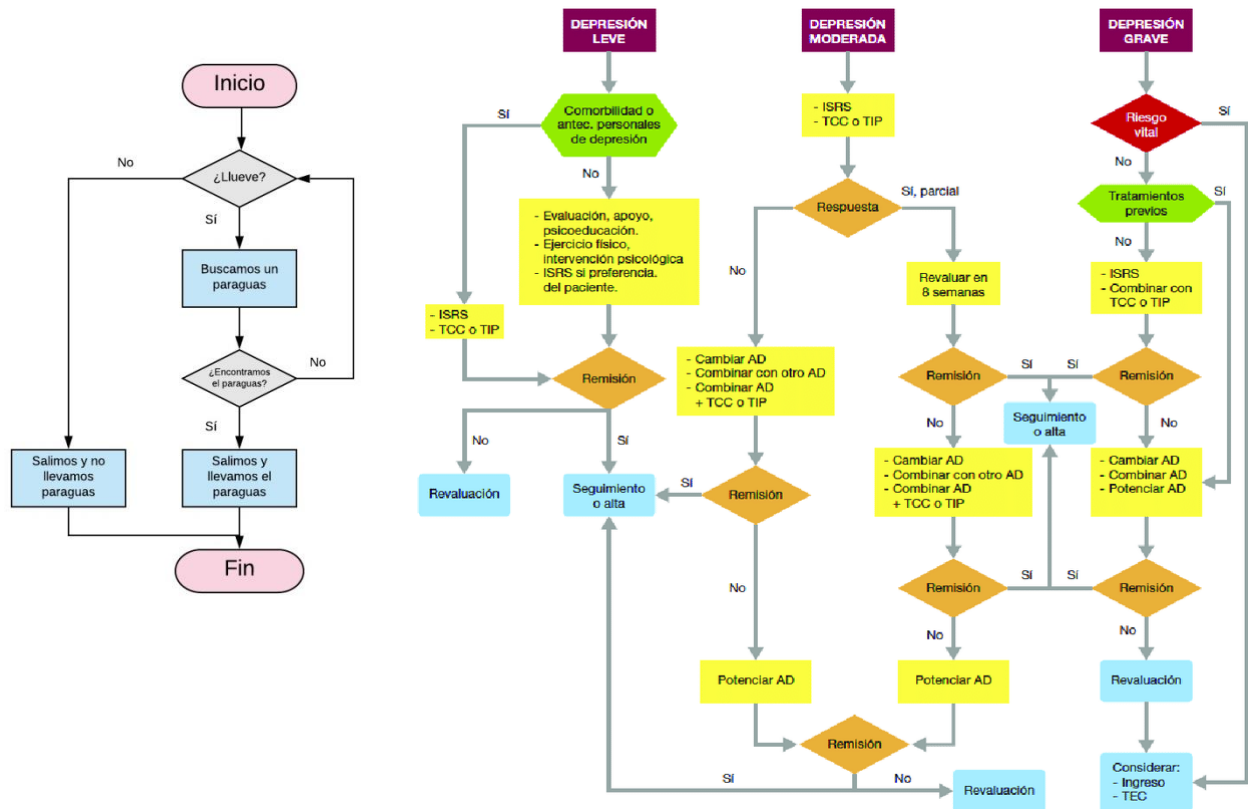
Hoy en día la palabra “**algoritmo**” se usa mucho al hablar de las estrategias de selección, decisión, posicionamiento, marketing ... empleadas por las grandes corporaciones tecnológicas. Efectivamente, estas grandes corporaciones han diseñado algoritmos muy complejos, siguiendo técnicas de Inteligencia artificial, que luego son codificados en algún **lenguaje de programación** para ejecutarlos en un equipo en concreto.

Así pues, el **algoritmo** se puede considerar como el prototipo de una idea, la fase “embrionaria” de todo programa, que puede ser para un programa “pequeñito”, o para el complejísimo programa que decide el posicionamiento de los vídeos en YouTube.

FLUJOGRAMAS. Se denominan Flujogramas, o diagramas de flujo, a la representación gráfica de la solución de un problema. Se utilizan no sólo en informática, sino también en el ámbito empresarial para representar procesos administrativos o diagramas de empresa.

Se utilizan como herramienta que hace más fácil la visualización de todos los recorridos posibles que puede seguir un programa. Los símbolos con los cuales se construye un organigrama son los siguientes:

Símbolo	Nombre	Función
	Inicio / Final	Representa el inicio y el final de un proceso
	Línea de Flujo	Indica el orden de la ejecución de las operaciones. La flecha indica la siguiente instrucción.
	Entrada / Salida	Representa la lectura de datos en la entrada y la impresión de datos en la salida
	Proceso	Representa cualquier tipo de operación
	Decisión	Nos permite analizar una situación, con base en los valores verdadero y falso



PSEUDOCÓDIGO. Se denomina pseudocódigo (falso código) a una especie de lenguaje utilizado para describir algoritmos, a medio camino entre el lenguaje y un lenguaje de programación de alto nivel. Cualquier algoritmo bien descrito en pseudocódigo es convertible a cualquier lenguaje de programación:

- Operaciones de Procesado interno y Entrada / Salida
- Bifurcaciones o tomas de decisión condicionales
- Bucles
- Llamadas a subprogramas
- Descripciones y Estructuras de datos

La sintaxis del pseudocódigo no es estándar, y existen diferentes variantes.

```

Procedimiento Ordenar (L)
//Comentario: L = (L1, L2, ..., Ln) es una lista con n elementos//
k ← 0;
Repetir
    intercambio ← Falso;
    k ← k + 1;
    Para i ← 1 Hasta n - k Con Paso 1 Hacer
        Si Li > Li+1 Entonces
            intercambiar (Li, Li+1)
            intercambio ← Verdadero;
        Fin Si
    Fin Para
Hasta Que intercambio = Falso;
Fin Procedimiento
  
```

Proceso Adivina_Numero

```

intentos ← 10
num_secreto ← azar(100)+1

Escribir "Adivine el numero (de 1 a 100):"
Leer num_ingresado
Mientras num_secreto ≠ num_ingresado Y intentos > 1 Hacer
    Si num_secreto > num_ingresado Entonces
        Escribir "Muy bajo"
    Sino
        Escribir "Muy alto"
    FinSi
    intentos ← intentos - 1
    Escribir "Le quedan ", intentos, " intentos:"
    Leer num_ingresado
FinMientras

Si num_secreto = num_ingresado Entonces
    Escribir "Exacto! Usted adivino en ", 11 - intentos, " intentos."
Sino
    Escribir "El numero era: ", num_secreto
FinSi

FinProceso
  
```

Se puede decir que con los flujogramas y el pseudocódigo **empezó todo...** pero a partir de ahí se han desarrollado numerosas técnicas de **análisis y diseño** previo a la codificación de programas, mejor dicho, **al desarrollo de aplicaciones informáticas**.

Los flujogramas y el pseudocódigo pueden ayudarnos a entender la lógica de la programación en los períodos formativos iniciales, pero una vez superada esa fase de aprendizaje, veremos que la complejidad de una aplicación informática hoy en día nos obliga a conocer y utilizar otras técnicas más complejas de **análisis y diseño orientado a objetos**. Mucho más allá de lo que representan los flujogramas y el pseudocódigo, hay diversas metodologías de análisis y diseño de aplicaciones a las que se presta atención y se las dota de contenido en el módulo de primer curso **“Entornos de Desarrollo”**.

Diagrama objetos UML

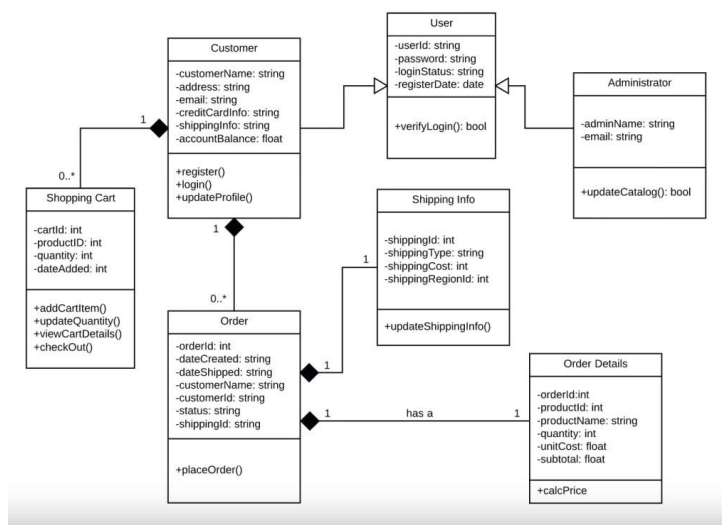
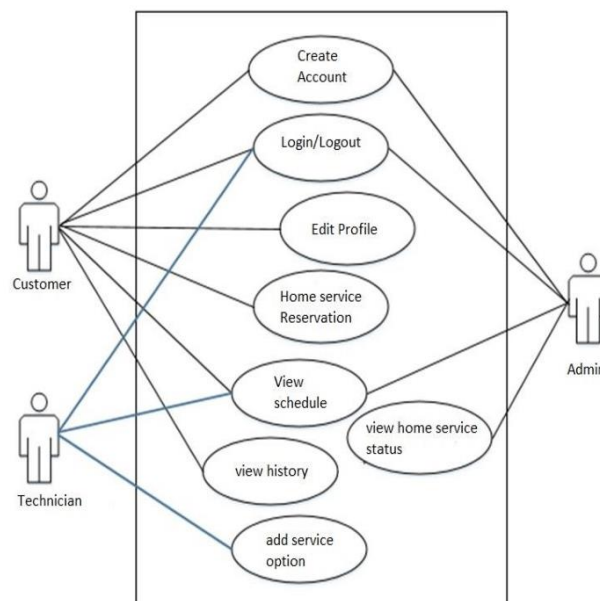


Diagrama de casos de uso



LENGUAJE DE PROGRAMACIÓN JAVA.



¿Por qué estudiamos Java?

Por una combinación de varios aspectos: el diseño del lenguaje, su popularidad y su uso masivo en proyectos **web y multiplataforma** de toda índole. El alto porcentaje de uso de JAVA como lenguaje es incuestionable. En cuanto a su diseño como lenguaje de programación, Java proporciona una implementación muy limpia de la mayor parte de los conceptos de orientación a objetos más importantes, y sirve muy bien como lenguaje introductorio de enseñanza.

Además, la popularidad y uso masivo de Java nos garantiza una inmensa variedad de recursos de soporte: libros, tutoriales, ejercicios, entornos de desarrollo y tests de muchos tipos y en muchos estilos diferentes. Muchos de ellos son recursos en línea y la gran mayoría disponibles de manera gratuita. La gran cantidad y la buena calidad de los materiales de soporte hacen de Java una elección excelente como **introducción a la programación orientada a objetos**.

Historia de Java

Su origen se sitúa a principios de los años 90 en el seno de la empresa Sun Microsystems, dentro del ya legendario **Green Project**.

El objetivo era crear un único Software que se pudiera ejecutar en dispositivos de diferente naturaleza, sin necesidad de modificar y recompilar el código para cada diferente plataforma Hardware, como se venía haciendo hasta la fecha con los demás lenguajes de programación. (C++ era el más utilizado entonces).

Se aspiraba a lograr, **y se consiguió**, un lenguaje de programación multiplataforma, independiente del dispositivo, para implementar programas que se pudieran ejecutar en cualquier dispositivo (Hardware) independientemente de su arquitectura o Sistema Operativo.

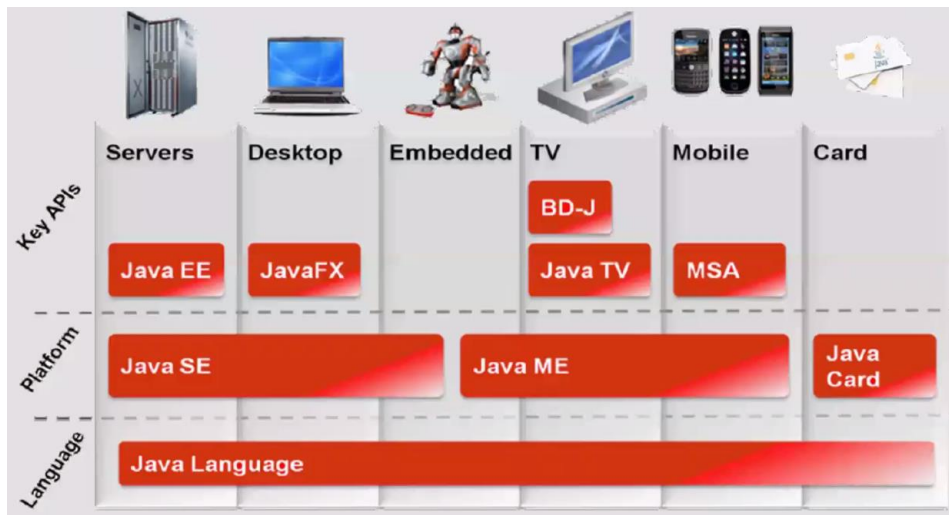
C++ no servía para ese propósito. Un programa sólo podía ser ejecutado en el dispositivo concreto para el que había sido compilado, y debía ser modificado y recompilado de nuevo para que se pueda ejecutar en un dispositivo diferente.

En **1996** surgió la primera versión de **Java**.

El eslogan que Sun popularizó sobre el lenguaje Java fue **Write Once, Run Anywhere**. También conocido con sus iniciales **WORA**. Esto quiere decir que un programa Java se escribe una vez y se puede ejecutar en cualquier plataforma sin tener que ser modificado ni recompilado.

Java logró convertirse en el **lenguaje multiplataforma** al que se aspiraba de inicio.

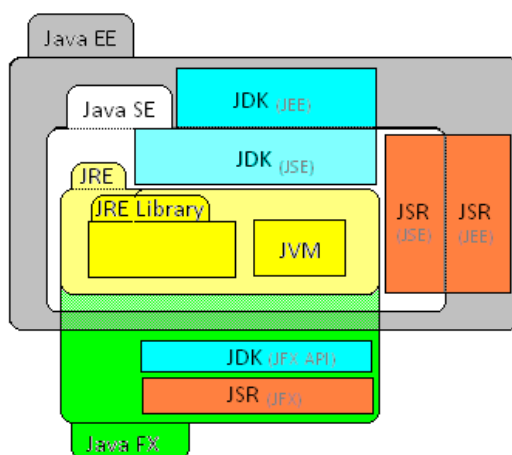
En 2010 **Oracle** compró Sun Microsystems y Java ha continuado evolucionando. Uno de los cambios recientes más determinantes ha sido el giro hacia la arquitectura de **organización modular de proyectos** a partir de la versión JAVA 9. Los paquetes se agrupan ahora en módulos y el alcance de la palabra “public” cambia notablemente, a la vez que se consigue “trocear” en módulos la API del lenguaje para hacerla más manejable y ligera a la hora de desarrollar aplicaciones Java para dispositivos con recursos hardware limitados. **Todo esto se entenderá mucho mejor a medida que avance el curso** y vayamos conociendo mejor las características del lenguaje JAVA, el papel de los paquetes, y la estructura de la API de JAVA (Aplication program Interface).



Existen tres distribuciones principales de Java:

- **Java SE** (Java Standard Edition). Versión estándar de Java. Contiene los fundamentos básicos del lenguaje y está orientada al desarrollo de aplicaciones de escritorio.
- **Java EE** (Java Enterprise Edition). Utiliza muchos de los componentes de JSE y añade nuevas funcionalidades. Está orientada al desarrollo de servicios web, networking, aplicaciones en el lado del servidor y aplicaciones basadas en la web. Es uno de los estándares para el desarrollo de aplicaciones web.
- **Java ME** (Java Platform Micro Edition). Esta distribución de Java está orientada a la programación de dispositivos móviles, dispositivos inalámbricos y pequeños dispositivos.

Lo importante es que por debajo de estas 3 distribuciones orientadas a diferentes funcionalidades lo que está es el **lenguaje Java** en sí mismo, de ahí que sea el lenguaje ideal para que los/as no iniciados/as puedan empezar a abrirse camino en el mundo del desarrollo de aplicaciones.



JAVA RUNTIME ENVIROMENT (JRE)

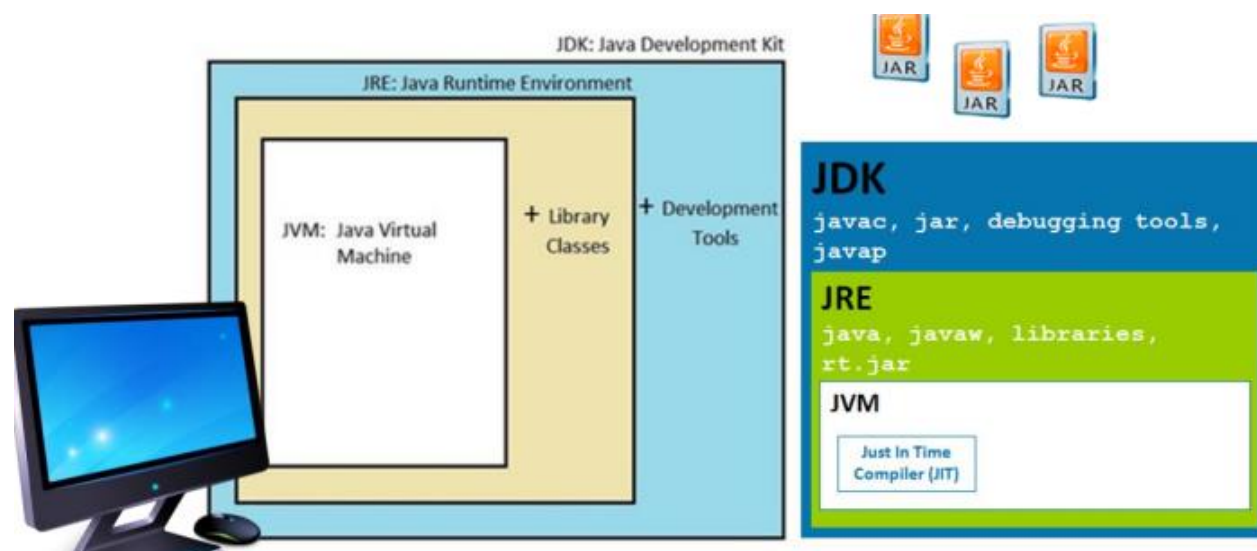
Es lo mínimo que debemos instalar en una máquina para poder ejecutar aplicaciones JAVA en alguna de sus variantes. Incluye la máquina virtual de JAVA (JVM) + el API – CORE de JAVA que es la funcionalidad básica que el propio lenguaje nos ofrece.

JAVA DEVELOPMENT KIT (JDK)

El kit completo de Java para un desarrollador. Incluye el comentado JRE + herramientas de compilación, ejecución, empaquetado de aplicaciones y más. Siendo realistas estas herramientas no son utilizadas en las empresas de desarrollo de forma masiva, sino que la tendencia se inclina más a la utilización de entornos integrados de desarrollo (IDES) cómo NetBeans, Eclipse o IntelliJ. Hasta Java 8 se podía descargar e instalar sólo el núcleo de JAVA (JRE). Desde las versiones modulares de Java (Java 9 en adelante), JAVA sólo permite descargar e instalar el JDK completo.

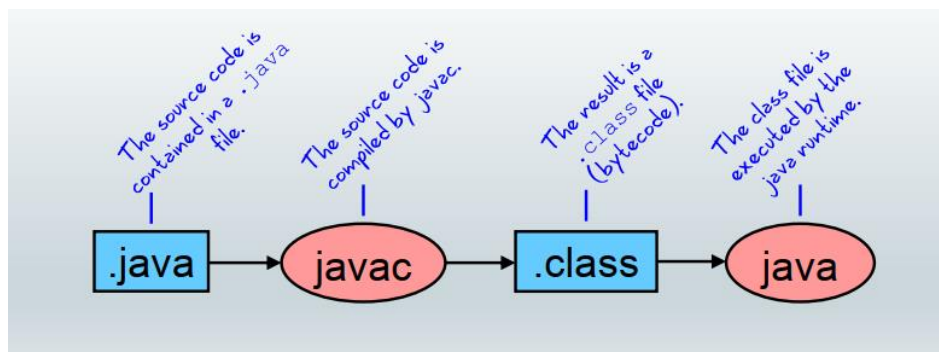
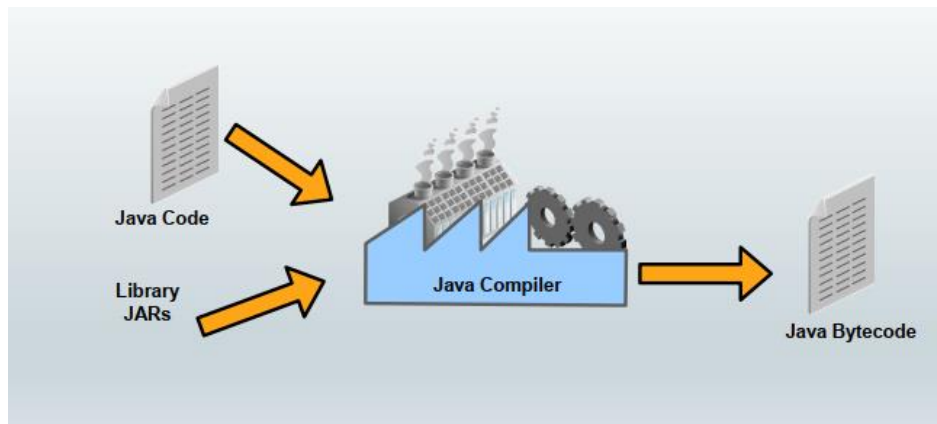
JAVA SPECIFICATION REQUEST (JSR)

Documentación formal de cada distribución JAVA.

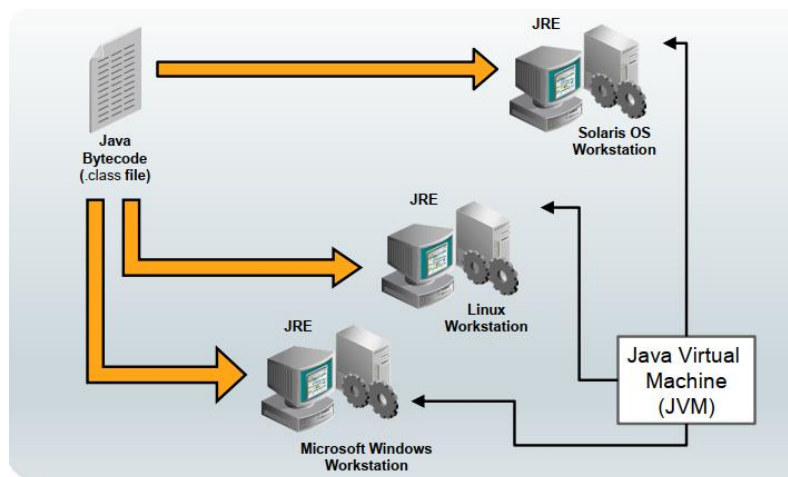


En este módulo utilizaremos entornos integrados de desarrollo o Integrated development Enviroment (**IDES**) como **BlueJ**, **NetBeans** o **Eclipse** para facilitar al alumnado la tarea de programar en JAVA, y además lo haremos así porque los IDE (sobre todo **Netbeans** y **Eclipse**) se utilizan masivamente en el sector del Desarrollo de Aplicaciones a nivel profesional.

De todas formas, es importante destacar que los **programas de formación oficial de JAVA por parte de ORACLE se basan en el manejo del JDK original, sin recurrir a ningún IDE**. Así mismo, las **certificaciones oficiales de JAVA-ORACLE** exigen a los candidatos el manejo de las herramientas del JDK a bajo nivel, por lo que si un programador (en formación o en activo) tiene el objetivo de superar algunos de los exámenes de certificación de JAVA-ORACLE, deberá conocer y utilizar las herramientas del JDK, aunque desarrolle gran parte de su trabajo diario con algún IDE. **En exámenes de certificación de JAVA-ORACLE no se permite la utilización de Entornos Integrados de Desarrollo (IDES)**. Afortunadamente, en toda vuestra formación de DAW si, por supuesto, y esto os hará la vida como desarrolladores/as mucho más fácil.



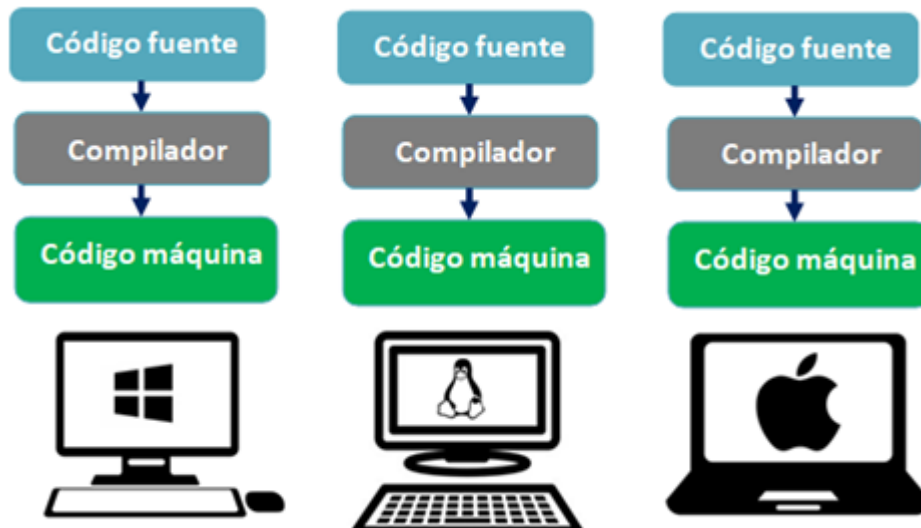
LA MÁQUINA VIRTUAL DE JAVA - JVM



Fue quizás (inicialmente) la mayor gran aportación que introdujo Java respecto a los demás lenguajes de programación preexistentes, e hizo posible que un programa funcionara en cualquier plataforma. (Java Multiplataforma)

Hasta ese momento un programa escrito en un lenguaje de programación (**código fuente**) se debía traducir a un lenguaje ejecutable por la máquina (**código máquina** o **ejecutable**). Esa “traducción” la realizaba el compilador/interprete correspondiente, de ahí que se hablara de **lenguajes compilados** o **lenguajes interpretados**.

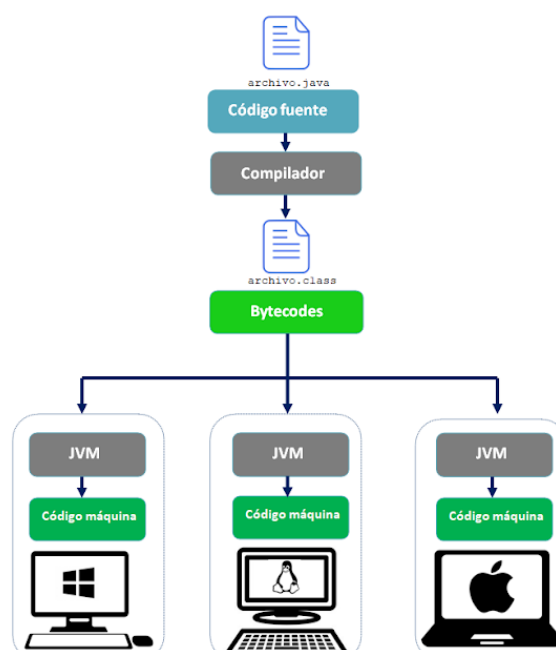
El código máquina es distinto para cada plataforma o arquitectura HARDWARE. Ordenadores con sistemas operativos y/o hardware distintos tienen distinto código máquina pues el código máquina es muy dependiente del microprocesador y de la arquitectura interna del hardware.



Era necesario cambiar el código fuente y recompilar los programas para cada tipo de arquitectura S.O/HARDWARE. Sin embargo, **JAVA es multiplataforma**:

- En Java un programa no se traduce directamente a código ejecutable.
- Un programa Java se compila y se obtiene un **código intermedio** llamado **bytecode**.
- El **bytecode** lo interpreta la **Máquina Virtual de Java (JVM)** y obtiene el código ejecutable.

La **JVM (Java Virtual Machine)** se **distribuye gratuitamente** dentro del correspondiente JRE o JDK para prácticamente todos los sistemas operativos actuales. Un archivo **.class (bytecode)** se puede ejecutar en cualquier ordenador que tenga instalada la máquina virtual java JVM.



CARACTERÍSTICAS DEL LENGUAJE JAVA

Orientado a Objetos

De sintaxis similar a C++, lo que en principio facilitó el aprendizaje de Java a los programadores de C++. Además, eliminó algunas de las características más conflictivas de C++

- En JAVA no hay punteros.
- No permite la herencia múltiple, a no ser a través de Interfaces.
- No hay necesidad de liberar memoria manualmente. La gestión de memoria dinámica se hace automáticamente (recolector de basura en el HEAP).

Compilado + interpretado

Multiplataforma

Robusto

- Es capaz de manejar errores en tiempo de ejecución mediante el controlador de **excepciones**.
- Utiliza un recolector de basura para eliminar de la memoria todos los objetos que ya no se usan quitando esa responsabilidad al programador/a.
- Realiza la comprobación de tipos en cualquier operación, avisando si intervienen tipos incompatibles lo que hace que se eviten errores de cálculo.

Multitarea

Permite crear programas con varios hilos (threads) de ejecución, lo que a su vez permite aprovechar las características y potencia de los modernos procesadores con múltiples núcleos.

VERSIONES JAVA

Java ha experimentado numerosos cambios desde la primera versión 1.0, así como un enorme incremento en el número de clases y paquetes que componen la biblioteca estándar o **API**.

- A partir de la versión 1.6 se deja de utilizar la nomenclatura J2SE para llamarse Java SE. Java SE 8...
- Cada versión tiene varias revisiones.
- A partir de Java 9 (2017) aparece una nueva versión de Java cada 6 meses.

A pesar de la aparición de versiones más nuevas, **Java 8** sigue siendo muy utilizado, pues es la última versión de JAVA antes del cambio a la filosofía de empaquetado modular. El cambio a **JAVA modular a partir de JAVA 9** no es un cambio obligatorio para la mayoría de los programadores y por ello es razonable la “fidelidad” a JAVA 8. El cambio a JAVA 9 y posteriores es una nueva forma de empaquetar y organizar las aplicaciones para adaptarlas mejor al diseño en capas y a la diversidad de Hardware.

También es importante el cambio de la estructura del API del lenguaje a partir de JAVA 9, que pasa de ser una librería monolítica para convertirse también en un API modular de la que los programadores pueden usar sólo aquellas partes que necesitan para cada proyecto. Se puedan así aligerar los requerimientos hardware y permitir ejecutar con mayor agilidad aplicaciones JAVA en dispositivos con pocos recursos.

Además, a partir de Java 11 se produce un cambio radical en la licencia de uso. Antes de Java 11 se podía descargar el JDK, programar y poner las aplicaciones en producción sin tener que pagar nada. A partir de Java 11 hay que pagar una licencia a Oracle si queremos utilizarlo para poner aplicaciones en producción. Esto no afecta a versiones anteriores. Las versiones 8, 9 o 10 siguen siendo gratuitas.

Las versiones que más nos interesan de JAVA son las LTS (Long time support). Actualmente:

LTS No Modular - JAVA 8 – Soporte hasta Marzo 2025 (permite instalar sólo el entorno básico de Java JRE)
LTS Modular - JAVA 11 – Soporte hasta Septiembre 2026 (Obliga a descargar e instalar el JDK completo)

EL KIT DE HERRAMIENTAS PARA PROGRAMAR EN JAVA:

Cómo hemos comentado, todo programador que quiera trabajar con JAVA necesita disponer de un kit básico de herramientas sin el cual no podrá ni ejecutar ni compilar aplicaciones.

JAVA RUNTIME ENVIROMENT (JRE)

Es lo mínimo que debemos instalar en una máquina para poder ejecutar aplicaciones JAVA en alguna de sus variantes. Incluye la máquina virtual de JAVA (JVM) + el **API de JAVA**.

JAVA DEVELOPMENT KIT (JDK)

Lo habitual es instalar el kit completo, que incluye el comentado JRE + las herramientas de compilación, ejecución, empaquetado, documentación, debug, etc.

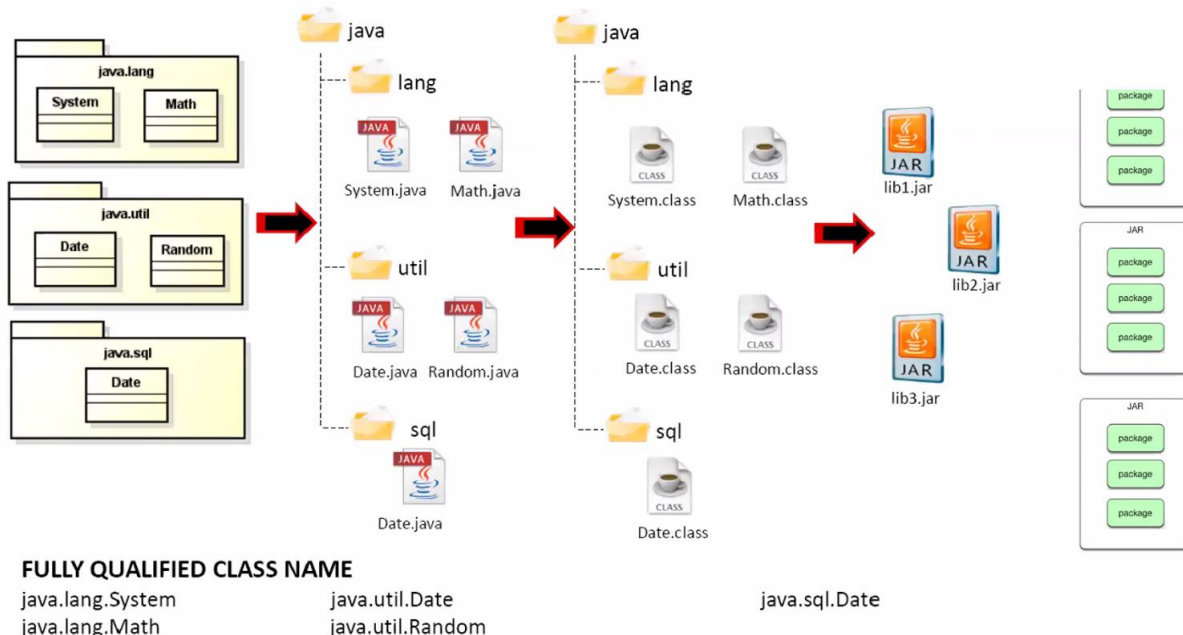
Tanto el JRE como el JDK de las diferentes versiones de JAVA se pueden descargar de forma gratuita desde la sección correspondiente de la web de **ORACLE**: <https://www.oracle.com/java/technologies/downloads/>

APPLICATION PROGRAM INTERFACE (API) DE JAVA

En la web de ORACLE está toda la documentación técnica de las diferentes versiones JAVA. Es un recurso muy potente para toda la comunidad de desarrolladores. Muy importante para nosotros/as es la documentación del **API** de Java. En ella se describe, paquete por paquete, toda la funcionalidad que el lenguaje ofrece a los desarrolladores. Muchas de las funciones específicas que vamos a necesitar en nuestras aplicaciones ya están disponibles, y lo que tenemos que hacer es encontrarlas dentro del **API**.

<https://docs.oracle.com/en/java/javase/index.html>

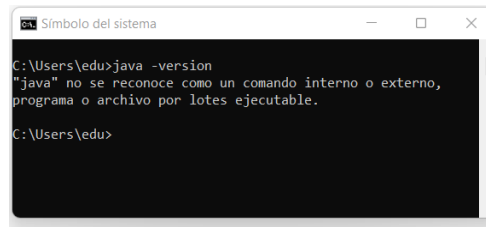
Hasta JAVA 9, casi toda la API está incluida en un gran paquete, el conocido **rt.jar**. **A partir de Java 9**, el rt.jar se “trocea” en unos **90 paquetes .jmod** para hacer que la API sea más manejable y no sea necesario tenerla disponible al completo en dispositivos con pocos recursos hardware.



COMO INSTALAR JAVA EN MI EQUIPO

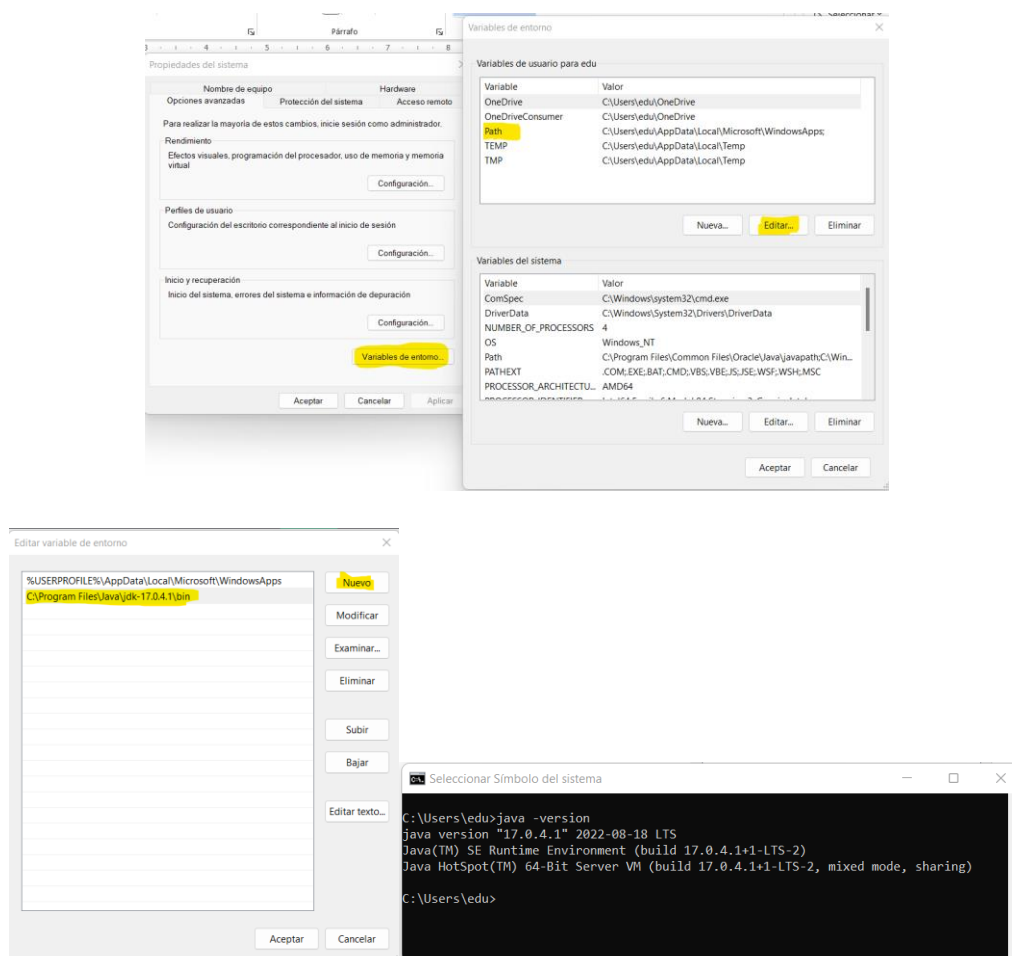
Como se ha comentado, para poder trabajar con Java en un equipo, lo 1º es instalar el JRE o el JDK de la versión de Java que decidamos instalar. <https://www.oracle.com/java/technologies/downloads/>

La comprobación rutinaria que debemos hacer para comprobar que Java está correctamente instalado en nuestro sistema es ejecutar el comando `java -version` abriendo una ventana del CMD (Windows)



```
C:\Users\edu>java -version
"java" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.
C:\Users\edu>
```

No basta con descargar e instalar el JDK de Java. Es necesario modificar la variable `PATH` para que Java funcione correctamente. Para ello hay que acceder a la configuración avanzada del sistema, y debido a los cambios que van surgiendo con las sucesivas versiones Windows, en vez de hacer una guía de “pantallazos”, que probablemente sirva para algunas versiones pero no para otras, lo más práctico es recomendar que cada uno/a realice una búsqueda en su sistema de la palabra “`PATH`” y que deje que Windows le guíe a la sección de configuración para variables de entorno. Una vez allí:



Es conveniente reiniciar el sistema después de editar el `PATH` y ejecutar de nuevo `Java -version`

IDE (INTEGRATED DEVELOPMENT ENVIRONMENT – ENTORNOS DE DESARROLLO).

Aunque podemos escribir programas desde cualquier editor de texto y guardarlos en archivos .java para después compilarlos y ejecutarlos con las herramientas incluidas en el JDK, la opción inicial de la mayoría de programadores es instalar el JRE/JDK de la versión de JAVA elegida + un IDE. Los IDE o entornos de desarrollo son herramientas Software que facilitan al programador el desarrollo de aplicaciones.

Algunas ventajas que ofrecen los IDE son:

- Facilidades para escribir código, corregir errores sintácticos y re-factorizar el código.
- Facilidades de depuración.
- Facilidad de configuración del sistema.
- Facilidades para organizar los archivos de código.
- Facilidad para exportar e importar proyectos.

Mediante un IDE el desarrollador de aplicaciones puede escribir, compilar y ejecutar programas de forma sencilla. Algunos de los IDEs más utilizados para programar en Java son: **Eclipse, NetBeans e IntelliJ Idea.**

Los diferentes IDE son descargables de forma gratuita desde las WEBS correspondientes.

¡IMPORTANTE! Aunque algunas de las últimas versiones de los IDEs incluyen la instalación del JDK de JAVA, facilitando también la instalación de Java, hay que tener precaución con esto pues no son todos, ni todas las versiones. Lo más recomendable es instalar JAVA previamente y editar la variable PATH como hemos explicado en el apartado anterior. Después ya podremos instalar el IDE que queramos y este IDE reconocerá automáticamente la/s versión/es de Java que tengamos instalada/s (podemos tener varias)



NetBeans - Puede descargarse desde <http://netbeans.org/>

Contiene todo lo que se suele pedir a un entorno de desarrollo: editor avanzado de código, depurador, extensiones, etc.

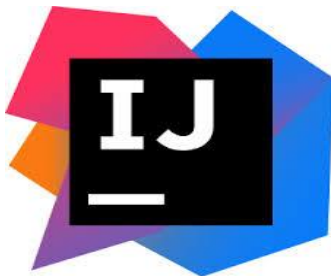
Incluye además un servidor de aplicaciones (Tomcat) para probar aplicaciones de servidor y WEB.

Tiene una arquitectura extensible, con módulos específicos para desarrollo web, aplicaciones móviles, diseño UML, etc.



ECLIPSE - Puede descargarse en <http://www.eclipse.org/>

Junto a NetBeans, el entorno de desarrollo más utilizado. Es un IDE de código abierto que al igual que NetBeans permite el desarrollo de aplicaciones en varios lenguajes de programación. Es el que vamos a usar preferentemente para este módulo.



IntelliJ IDEA - <https://www.jetbrains.com/idea/>

Entorno de desarrollo completo creado por JetBrains para la creación de software. Está disponible en dos versiones, una versión para la comunidad y otra versión comercial.

¿CÓMO SON LOS PROGRAMAS JAVA?

Una Aplicación **JAVA** estará estructurada en paquetes (**como mínimo 1 PAQUETE**), y dentro de esos paquetes se organizan las **CLASES**. El código fuente de las **clases** se guarda individualmente en archivos con extensión **.java**, y los paquetes serán los que crean la estructura (carpetas) para organizar los **.java** de las clases.

- **Lo habitual** es guardar en un archivo cada clase y darle al archivo el **mismo** nombre de la clase .java
- Puede darse el caso de guardar en un archivo varias clases, pero entonces sólo 1 será pública (**public**) y dará nombre al archivo. El resto serán clases privadas.

Se suele decir que (a nivel organizativo) un programa JAVA es como una biblioteca, dónde los **paquetes** son las **estanterías**, y las **clases** son los **libros**. La misión de los paquetes es actuar como “contenedores” para organizar bien las clases, y a la vez dotarlas de un **nombre único y exclusivo a nivel global**.

Una **recomendación profesional** es utilizar el nombre de dominio de nuestra empresa **INVERTIDO** como **prefijo para nuestros paquetes**. De esta forma garantizamos que todas las clases en ellos contenidas tendrán un nombre único a nivel mundial, y que podemos crear una clase llamada **Empleado**, por ejemplo, que no se confunda con otra clase **Empleado** que alguien más haya desarrollado en cualquier otra parte del mundo.

Siguiendo esta recomendación, durante este curso usaremos el prefijo **es.cifplaboral.nombreusuarioeducastur** como prefijo para nuestros paquetes.

Se puede decir que los paquetes organizan las clases y (si lo hacemos bien) les garantizan un espacio de nombres único. Lo **1º** que debemos hacer en el código de una clase es **identificar** a que paquete pertenece esa clase (**package**). De esa forma estamos organizando esa clase en el paquete que le corresponde.

Archivo HolaMundo.java

```
package
es.cifplalaboral.eduardocl.prueba
```

```
Class HolaMundo{
```

```
    Atributos y métodos de la clase
```

llamase igual que la nuestra, pues ninguna otra pertenecerá al paquete es.cifplalaboral.eduardocl.prueba. **Las clases se identifican con su FQDN.**

El archivo HolaMundo.java contiene el código JAVA de la clase **HolaMundo**. Al compilar ese código, se creará la estructura de carpetas adecuada para contener el proyecto

Si esa aplicación se empaqueta en un archivo **.jar** y se distribuye, la clase HolaMundo se describe como **es.cifplalaboral.eduardocl.prueba.HolaMundo**, y a nivel mundial será inconfundible con ninguna otra, aunque se

Las aplicaciones complejas se suelen organizar en varios **packages (paquetes)** o **a partir de Java 9 también se pueden organizar en módulos**. En uno u otro se van agrupando las clases según su funcionalidad, para estructurar correctamente la aplicación en capas, y también para facilitar la reutilización de código. Lo iremos entendiendo a medida que avance el curso.

No nos asustemos. Si no usamos un IDE, resulta complicado compilar, ejecutar y empaquetar aplicaciones Java, **tal y cómo se exige en los programas de certificación oficial en JAVA de ORACLE**. Sin embargo, cuando usamos un IDE como **NetBeans** o **Eclipse** (lo habitual en empresa) veremos lo sencillo que resulta todo, pues el IDE nos facilitará muchísimo la compilación, ejecución y empaquetado de proyectos complejos y nos generará automáticamente las estructuras de carpetas correspondientes a los paquetes, así como el posterior empaquetado completo del proyecto en **.jar** o **.jmod**

De todas formas, por el hecho de usar IDEs no hay que mal acostumbrarse, o perder la perspectiva de cómo se organizan los proyectos JAVA. Al menos hay que conocer como encajan las piezas del “puzzle” Java, y al menos diferenciar los 2 formas diferentes de empaquetar aplicaciones (paquetes **.jar** o módulos **.jmod**)

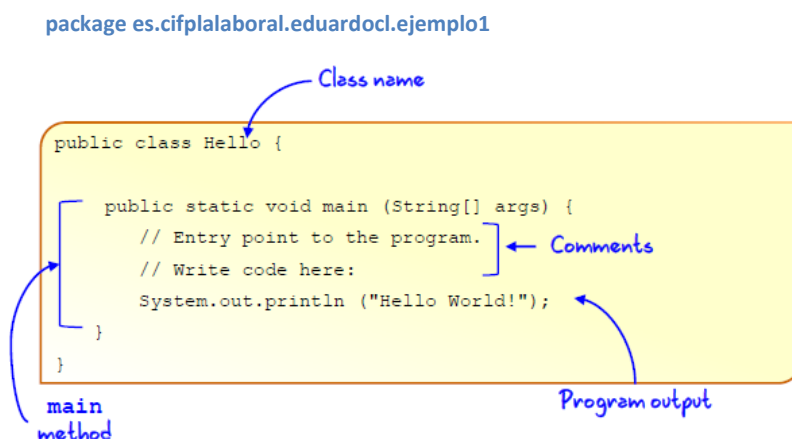
¿POR DÓNDE ARRANCA UN PROGRAMA JAVA? – MAIN CLASS

Método main() → “Programa principal”

Un programa JAVA tiene que arrancar por alguna parte. De esa misión se encarga una clase “especial” que es aquella en la que escribiremos el llamado método **main()**. Sólo puede haber 1 método **main()**, y la clase en la que decidamos incluirlo será llamada formalmente la **Main Class**.

Cuidado!, esa clase debe tener su propio nombre ... *HolaMundo*, *Hello*, *MiProgramaJava* ... o como queramos llamarla. Por el hecho de ser la “portadora” del método main, se la considera **formalmente Main-Class**, coloquialmente clase driver - “conductora”, pero para que arranque el programa, La máquina virtual de JAVA (JVM) debe saber cuál es la Main-Class de nuestro proyecto, y establecer en ella el punto de arranque de la aplicación. Esto se le indica “marcando” dónde está la Main_Class creando un archivo de texto **MANIFEST.MF** incluido en el archivo .jar en el que irá empaquetada la aplicación. En el MANIFEST.MF se escribe el nombre de la clase (***Hello en este ejemplo***) que contiene el método main()

Parece todo tan difícil !! ... pero no lo es, gracias a la ayuda de Eclipse, NetBeans o el IDE que empleamos para desarrollar aplicaciones. El IDE automatiza la creación y edición del MANIFEST.MF y hace que nos podamos abstraer de muchos aspectos relacionados con el empaquetado y distribución de aplicaciones. Pero ojo con estas cosas si algún día queremos obtener alguna certificación oficial de Java ORACLE ...



En el **MANIFEST.MF** del archivo .jar se incluirá el atributo **Main-Class: Hello**

Repasemos. Sobre la estructura de un programa JAVA:

- Estará formado por uno o varios archivos fuente con extensión **.java**
- Cada uno de estos archivos se corresponderá con una o varias clases (Class). Las clases se organizarán en paquetes (packages) o módulos (modules)
- El nombre del archivo .java debe coincidir con el nombre de la clase pública (public) que contenga.
- Si el archivo fuente contiene varias clases, solamente una de ellas podrá ser pública.
- Si no se cumplen estas condiciones el programa no compila. (*El uso de un IDE nos facilitará mucho las cosas en este sentido*).
- Para que un programa Java se pueda ejecutar debe contener una clase que llamaremos Main Class o *Clase Principal*, y que ha de incluir un método llamado **main**, con la siguiente declaración:

```
public static void main (String [] args) {
    //Aquí se escribe el código del método main
}
```

public: indica que el método es público y, por tanto, puede ser llamado desde otras clases.

static: indica que no es necesario crear ningún objeto de la clase para poder utilizar el método. También indica que el método es el mismo para todas las instancias que pudieran crearse de la clase.

void: indica que la función main no devuelve ningún valor.

El método main debe aceptar siempre, como parámetro, un Array de Strings, que contendrá los posibles argumentos que se le pasen al programa en la línea de comandos, aunque apenas usaremos esa posibilidad. Todos estos conceptos se estudian en profundidad a lo largo del curso y se irán entendiendo poco a poco, pero es importante ir viendo cómo las cosas “funcionan”.

Algunos aspectos a tener en cuenta a la hora de escribir código:

- Java **diferencia** entre MAYÚSCULAS y minúsculas.
- La mayoría de líneas de código terminan con `;` salvo excepciones que veremos durante el curso.
- Una instrucción puede ocupar más de una línea.

HABLAR CON JAVA.

A lo largo de este curso necesitaremos que los programas que realicemos muestren información por pantalla. También necesitaremos poder introducir información a través del teclado.

- Cuando necesitemos mostrar algo por pantalla utilizaremos la clase **System** del API de JAVA.
`System.out.println("mensaje que queremos mostrar");`
- Cuando necesitemos introducir información utilizaremos la clase **Scanner** del API de JAVA.
`Scanner teclado=new Scanner(System.in);`
`String texto=teclado.nextLine();`

package es.cifplalaboral.eduardocl.ejemplo1;

import java.util.Scanner; (La clase **Scanner** – **java.util** es necesario importarla , la clase **System** – **java.Lang** no)

```
public class EjemEntradaSalida {
    public static void main(String[] args) {
        Scanner teclado=new Scanner(System.in);
        System.out.println("Hola.");
        System.out.println("¿Cómo te llamas?");

        String miNombre=teclado.nextLine();
        System.out.println("Así que te llamas "+miNombre);
        teclado.close();
    }
}
```

Comentarios de código

A lo largo del curso, el profesor o tus compañeros/as tendrán que entender qué habéis querido hacer con el código que habéis escrito. Y por supuesto, a lo largo de vuestra vida profesional, usaréis código de terceros, participaréis en proyectos con otros/as desarrolladores/as, que también reutilizarán/retocarán/ampliarán vuestro código. **Todo esto será imposible de lograr sin comentar el código.**

Por eso son necesarios los **comentarios** en el código fuente. Se trata de unos “apuntes” en lenguaje natural que ayudan a quien lee vuestro código a comprenderlo mejor. Ahora puede parecer absurdo, pero llega un momento en el que ni uno mismo recuerda lo que quería hacer con el código sino lo ha comentado convenientemente.

En Java hay dos tipos de comentarios: de línea y de bloque. Los comentarios de línea ocupan, como era de esperar, una única línea. Se marcan incluyendo dos barras (‘//’) antes del comentario.

```
//Esto es un comentario de línea.
```

Los comentarios de bloque ocupan varias líneas. Se delimitan con los caracteres /* y */.

```
/*  
    Este es un comentario multilínea o de bloque.  
*/
```

Hay un tercer tipo de comentarios que son los **comentarios de JAVADOC**, que servirán para generar **documentación automática** en formato HTML y son muy importantes para la reutilización de código. Comienzan con /** y terminan con */ pero en cada línea hay que poner también un *.

```
/**  
 *  
 * @author Eduardo  
 * @version 2.0  
 *  
 */
```