QUE ES JAVA

Es una tecnología que incluye un lenguaje de programación y una plataforma de desarrollo

La plataforma Java

Por definición, una plataforma es el ambiente hardware o software sobre el cual se puede ejecutar un programa. La mayoría de las plataformas actuales se pueden describir como una combinación de un sistema operativo y su hardware subyacente; por ejemplo, un pc con su sistema operativo Windows se denomina comúnmente plataforma Windows, una notebook con una distribución Linux se denomina plataforma Linux y un celular con Android se conoce como plataforma Android.

La plataforma Java difiere respecto de las plataformas mencionadas en el párrafo anterior debido a que **es una plataforma software que se ejecuta sobre las prestaciones** que brinda otra plataforma **que gestiona el hardware** (denominada plataforma base) (observe la figura 1):

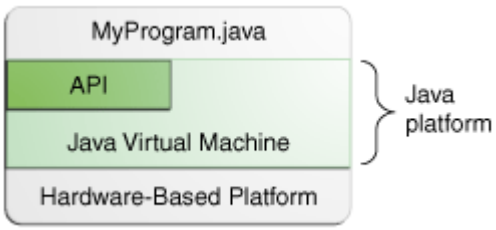


Figura 1. Plataforma Java

Esto significa que se puede montar la plataforma Java sobre cualquier plataforma base.

De la Figura 1 se puede concluir que el objetivo de la plataforma Java consiste en poder ejecutar sobre la plataforma base los programas realizados con el lenguaje Java.

La figura 1 muestra los componentes principales de la plataforma Java:

1. La máquina virtual de Java (Java Virtual Machine, JVM)
2. La Interfaz de Programación de Aplicaciones Java (Java Application Programming Interfaces, Java API)

La JVM es un intérprete. Los intérpretes son programas que permiten traducir el código creado por un programador en un lenguaje de alto nivel a código máquina.

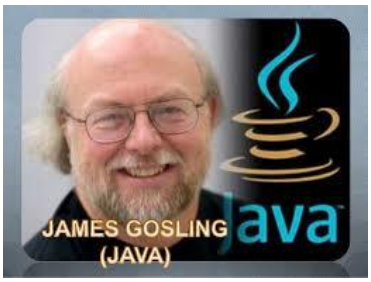
Pero, la JVM es particularmente especial. En lugar de traducir código fuente, traduce un tipo de código denominado **bytecode** al código máquina de la plataforma base al momento de ejecutar la aplicación. El funcionamiento de la JVM y su asociación con el concepto multiplataforma se describirán con mayor profundidad más adelante.

La API Java es una gran colección de componentes software que proveen muchas capacidades útiles para el desarrollador. Está agrupada dentro de bibliotecas (libraries) conformadas por clases e interfaces (conceptos que se desarrollarán en detalle más adelante). Estas bibliotecas se conocen con el nombre de paquetes (packages)

El lenguaje de programación Java

Es un lenguaje de **programación de alto nivel**. Su creador James Gosling lo define en base a sus características: sencillo, orientado a objetos, distribuido, interpretado, robusto, seguro, independiente de las arquitecturas, portable, eficaz, multitarea y dinámico. Antes de poder describir en detalle cada una de estas características es conveniente indicar como funciona el proceso de desarrollo de una aplicación Java.

En el lenguaje de programación Java, todo el código fuente se escribe en archivos de texto plano con la extensión *.java*. Estos archivos fuente son entonces compilados por una herramienta de la plataforma denominada **compilador *javac*** guardando el resultado en archivos con el mismo nombre, pero con extensión *.class*. Un archivo *.class* no contiene código nativo para el procesador de la plataforma base, sino que contiene un tipo de código denominado ***bytecode*** (lenguaje binario intermedio). Los bytecode se pueden considerar el lenguaje máquina de la JVM, debido a que la JVM es la encargada de traducir los bytecode en el código máquina del ordenador, tal como se puede ver en la figura 2.



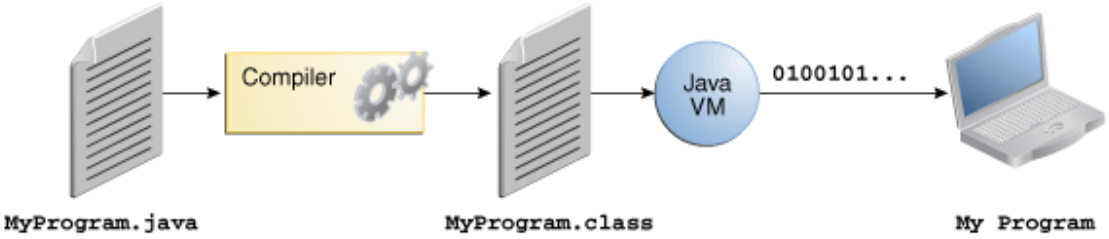


Figura 2. Proceso de ejecución de una aplicación Java

Para realizar esta operación, la plataforma cuenta con una herramienta denominada ***java launcher***, la cual es la encargada de crear una instancia de la JVM y ejecutar la aplicación en esa instancia.

El proceso de ejecución de una aplicación java se denomina pseudo interpretado, debido a que existe una primera etapa de compilación a código intermedio (bytecode) y luego una segunda etapa donde se interpreta el código intermedio mediante la JVM.

Esta forma de ejecución permite utilizar una misma aplicación (los bytecode) en diferentes sistemas operativos; siempre y cuando ese sistema operativo disponga de la JVM tal como lo expresa la figura 3.

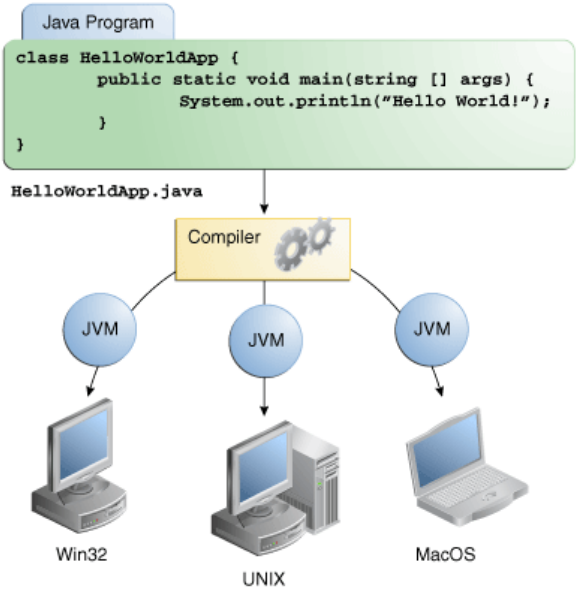


Figura 3. A través de la JVM, la misma aplicación es capaz de ser ejecutada en diferentes plataformas ⟶ Java es multiplataforma

Algunas JVM, tales como **Java SE HotSpot** (la versión creada por Oracle) incorporan tareas adicionales en tiempo de ejecución que aumentan el rendimiento de la aplicación. Entre estas tareas se incluyen encontrar cuellos de botella en el rendimiento y volver a recompilar (en código nativo) secciones de código frecuente.

Características de Java

- **Sencillo**: La sintaxis de Java es similar a la de los lenguajes C y C++, pero evita características semánticas que los vuelven complejos, confusos y poco seguros:
  - ✓ En Java sólo existen tres tipos primitivos: los numéricos (enteros y reales), el tipo carácter y el tipo booleano. Todos los tipos numéricos están firmados (esto significa que a pesar de no ser objetos representan el mismo tipo y alcance en cualquier plataforma que se ejecute)
  - ✓ En Java, los arreglos, las matrices y las cadenas de caracteres son objetos, lo que facilita su creación y manipulación.
  - ✓ En Java, **el programador no tiene que preocuparse de la gestión de la memoria**. Un sistema denominado "el recolector de basura" (garbage collector) se encarga de asignar la memoria necesaria a la hora de crear objetos y de liberarla cuando estos ya no se referencian en el dominio del programa (cuando ninguna variable referencia al objeto).
  - ✓ En Java, no existen preprocesadores ni archivos de encabezamiento. Las instrucciones "*define*" de C se sustituyen por constantes en Java y las instrucciones "*typedef*" de C se reemplazan por clases.
  - ✓ En C y C++, se definen estructuras y uniones para representar tipos de datos complejos, mientras que, en Java, se crean instancias de clases para representar tipos de datos complejos.
  - ✓ En C++, una clase puede heredar varias clases, lo que puede generar problemas de ambigüedad. Con el fin de evitar estos problemas, **Java sólo autoriza la herencia simple** pero aporta un mecanismo de simulación de herencia múltiple mediante la implementación de una o varias interfaces.
  - ✓ En Java **no existe la famosa instrucción *goto***, simplemente porque aporta una complejidad a la lectura de los programas y porque a menudo se puede prescindir de esta instrucción escribiendo un código más limpio. Además, en C y C++ se suele utilizar el *goto* para salir de bucles anidados. En Java, se utilizan las instrucciones *break* y *continue*, que permiten salir de uno o varios niveles de anidamiento.
  - ✓ En Java, **no es posible sobrecargar los operadores**, para evitar problemas de incomprensión del programa. Se preferirá crear clases con métodos y variables de instancia.

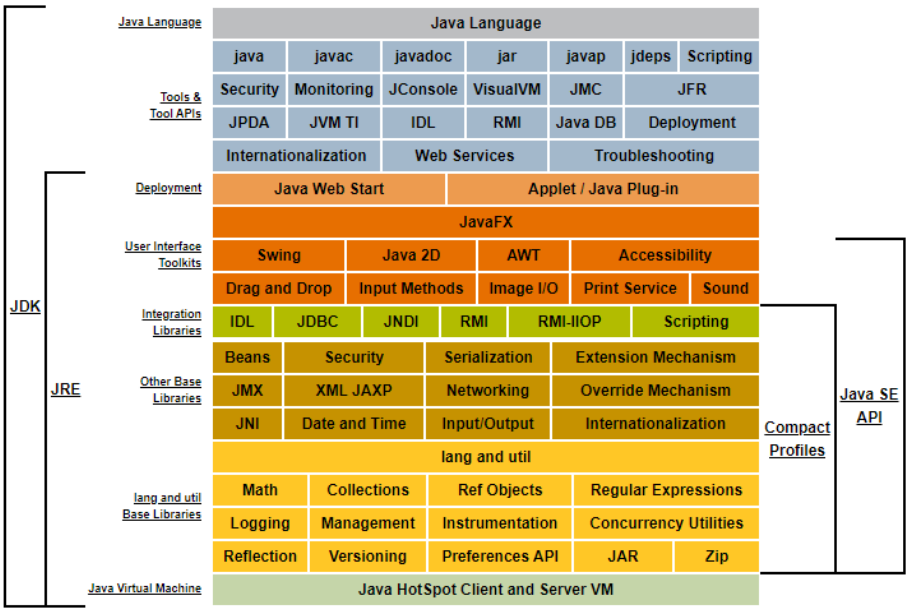
  Y, para terminar**, en Java, no hay punteros sino referencias a objetos** o celdas de un arreglo o matriz (referenciadas por su índice), simplemente porque la gestión de punteros es fuente de muchos errores en los programas C y C++.
- **Orientado a objetos**: Salvo los tipos de datos primitivos, todo en Java es concebido como un objeto. Y, además, Java se ha provisto de clases incorporadas que encapsulan los tipos primitivos (los wrappers). Por lo tanto, Java es un lenguaje de programación orientado a objetos y diseñado según el modelo de otros lenguajes (C++, Eiffel, SmallTalk, Objective C, Cedar/Mesa, Ada, Perl), pero sin sus defectos. Las ventajas de la programación orientada a objetos de forma resumida son: un mejor dominio de la complejidad (dividir un problema complejo en una serie de pequeños problemas), una reutilización más sencilla, y una mayor facilidad de corrección y de evolución. Java

estándar está dotado de un conjunto de clases que permiten crear y manipular todo tipo de objetos (interfaz gráfica, acceso a la red, gestión de entradas/salidas...).

- **Distribuido**: Java implementa los protocolos de red estándar, lo que **permite desarrollar aplicaciones cliente/servidor en arquitecturas distribuidas**, con el fin de invocar tratamientos y/o recuperar datos de máquinas remotas. Con este fin, Java estándar cuenta con API´s que permiten crear aplicaciones cliente/servidor distribuidas.
- **Interpretado**: La ejecución de un programa Java se realiza mediante el intérprete JVM. Esto hace que sea más lento. Sin embargo, conlleva también sus ventajas, en particular el hecho de no tener que recompilar un programa Java de un sistema a otro porque basta, para cada uno de los sistemas, con tener su propia máquina virtual.
  Debido a que Java es un lenguaje interpretado, no es necesario editar los enlaces (obligatorio en C++) antes de ejecutar un programa. En Java, por lo tanto, sólo hay dos etapas, la compilación y la ejecución. La máquina virtual se encarga de la operación de edición de enlaces en tiempo de ejecución del programa.
- **Robusto**: **Java es un lenguaje fuertemente tipado y estricto**. Por ejemplo, la declaración de las variables debe ser obligatoriamente explícita en Java. Se verifica el código (sintaxis, tipos) en el momento de la compilación y también en el momento de la ejecución (gestión de excepciones), lo que permite reducir los errores y los problemas de incompatibilidad de versiones.
  Además, **Java se encarga totalmente de la gestión de las referencias de los objetos y el programador no tiene manera de acceder a ellos**, lo que evita la sobreescritura accidental de datos en memoria y la manipulación de datos corruptos.
- **Seguro**: Dados los campos de aplicación de Java, es muy importante que haya un mecanismo que vigile la seguridad de las aplicaciones y los sistemas. El motor de ejecución de Java (Java Runtime Environment, JRE) es el encargado de esta tarea.
  El JRE se apoya en particular en el archivo de texto java.policy, que contiene información relativa a la configuración de la seguridad. Si bien el JRE se describirá en mayor detalle cuando se detallen los componentes de la plataforma Java, se puede adelantar que es el encargado de gestionar el consumo de memoria de los objetos, y no el compilador, como es el caso en C++.
  Puesto que en Java no hay punteros sino referencias a objetos, el código compilado contiene identificadores sobre los objetos que luego el JRE traduce en direcciones de memoria: esta parte es totalmente opaca para los desarrolladores con el objetivo de controlar que no se generen ni manipulen punteros en el espacio de memoria donde se ejecuta la aplicación, y que tampoco se presenten violaciones de acceso.
  Para lograr esto, en el momento de la ejecución de un programa Java, el JRE utiliza un proceso denominado ***ClassLoader*** que realiza la carga del *bytecode* contenido en las clases Java y a continuación, se analiza el bytecode.
  Como Java es un lenguaje distribuido, se implementan los principales protocolos de acceso a la red (FTP, HTTP, Telnet...). Se puede, pues, configurar el JRE con el fin de controlar el acceso a la red de sus aplicaciones:
  - Prohibir todos los accesos.
  - Autorizar el acceso solamente a la máquina anfitriona de donde procede el código de aplicación. Es la configuración por defecto para los applets Java.
  - Autorizar el acceso a máquinas en la red externa (más allá del firewall), en el caso de que el código de la aplicación también proceda de una máquina anfitriona de la red externa.
  - Autorizar todos los accesos. Es la configuración por defecto para las aplicaciones de tipo cliente pesado.

- **Independiente de las arquitecturas**: El compilador Java no produce un código específico para un tipo de arquitectura. De hecho, **el compilador genera un bytecode** que es independiente de cualquier arquitectura, de todo sistema operativo y de todo dispositivo de gestión de la interfaz gráfica de usuario (GUI). La ventaja de este bytecode reside en su fácil interpretación o transformación dinámica en código nativo para aumentar el rendimiento. Basta con disponer de la máquina virtual específica de la plataforma base para hacer funcionar un programa Java. Esta última se encarga de traducir el bytecode a código nativo.
- **Portable**: Java es portable gracias a que se trata de un lenguaje interpretado (haciendo referencia al proceso de interpretar bytecode). Además, a diferencia del lenguaje C y C++, los tipos de datos primitivos (numéricos, carácter y booleano) de Java tienen el mismo tamaño, sea cual sea la plataforma en la cual se ejecuta el código. Las bibliotecas de clases estándar de Java facilitan la escritura de código fuente que, a continuación, se puede desplegar en diferentes plataformas sin adaptación.
- **Eficaz**: Incluso si un programa Java es interpretado, lo cual es más lento que un programa nativo, Java pone en marcha un proceso de optimización de la interpretación del código, llamado ***JIT*** (Just In Time) o ***HotSpot***. Este proceso compila el bytecode Java en código nativo en tiempo de ejecución, lo que permite alcanzar el mismo rendimiento que un programa escrito en lenguaje C o C++.
- **Multitarea**: Java permite desarrollar aplicaciones que ponen en marcha la ejecución simultánea de varios hilos (o procesos ligeros). Esto permite efectuar simultáneamente varias tareas, con el fin de aumentar la velocidad de las aplicaciones, ya sea compartiendo el tiempo del CPU o repartiendo las tareas entre varios procesadores.
- **Dinámico**: En Java, el programador no tiene que editar los vínculos (obligatorio en C y C++). Por lo tanto, es posible modificar una o varias clases sin tener que efectuar una actualización de estas modificaciones para el conjunto del programa. La comprobación de la existencia de las clases se realiza en tiempo de compilación y la llamada al código de estas clases sólo se hace en el momento de la ejecución del programa. Este proceso permite disponer de aplicaciones más ligeras de tamaño en memoria.

Componentes de la Plataforma Java



El esquema precedente representa al lenguaje Java Stándar Edición versión 8 y muestra los diferentes componentes de la plataforma Java.

Lo más destacable de este esquema radica en que clasifica la funcionalidad de la plataforma:

- El **JRE** (Java Runtime Environment) permite ejecutar aplicaciones Java en el ordenador. Básicamente lo que hace es instalar la JVM, la API Java (repartida en tres categorías: API básicas, API de acceso a los datos y de integración con lo existente y API de gestión de la interfaz de las aplicaciones con el usuario) y las herramientas de despliegue de aplicaciones necesarias para asegurar que cualquier aplicación Java se pueda ejecutar en la plataforma base. A partir de la versión 8 se incorporan los perfiles compactos (compact profiles) los cuales permiten el uso reducido de memoria para las aplicaciones que no requieren toda la plataforma Java.
- El **JDK** (Java Development Kit) incluye todos los componentes del JRE y agrega el lenguaje de programación Java y un conjunto de herramientas de ayuda que permiten al desarrollador crear aplicaciones Java.

Ediciones de la Plataforma Java

La configuración de una plataforma Java afecta fundamentalmente los servicios que esta ofrece, las herramientas adicionales que incorpora y también aspectos menos visibles, tales como el modo de funcionamiento de la JVM. De acuerdo con la documentación actual de Oracle se puede distinguir las siguientes cuatro ediciones de Java:

- **Java SE**: conocida como *Standard Edition* es la edición más difundida de la plataforma Java. Incorpora el JDK, necesario para crear **aplicaciones de escritorio** con o sin interfaz gráfica de usuario, acceso al sistema de archivos, comunicación a través de redes, concurrencia y otros servicios básicos.
- **JavaFX**: originalmente JavaFX era una alternativa a Java SE para el **desarrollo de proyectos de tipo RIA** (*Rich Internet Applications*), con un núcleo más ligero y fácil de distribuir, capacidad de **aceleración 3D** aprovechando la GPU, servicios avanzados para producción de gráficos y animaciones, y un mecanismo simplificado para el diseño de interfaces de usuario. JavaFX forma parte de Java SE desde la versión 7 de dicha edición de la plataforma (y se eliminó de la misma a partir de la versión 13)
- **Java EE**: denominada *Enterprise Edition* de la plataforma Java, es la versión dirigida al desarrollo de soluciones software que se ejecutarán en un **servidor de aplicaciones**. A las capacidades de Java SE, la edición EE agrega servicios para gestionar la persistencia de objetos en bases de datos, hacer posible la invocación remota de métodos, crear aplicaciones con interfaz de usuario web, etc.
- **Java ME**: *Micro Edition*, está enfocada a la creación de programas que se ejecutarán en **sistemas con recursos limitados**, tales como teléfonos móviles, electrodomésticos y dispositivos de domótica o equipos para entornos empotrados como la Rasperry Pi y similares.

Las ediciones de Java son en realidad especificaciones abstractas de los servicios y modos de funcionamiento de los distintos elementos de la plataforma. Existen múltiples implementaciones de dichas especificaciones. Por ejemplo, se puede volver a destacar HotSpot, la implementación de Oracle de la JVM, existiendo implementaciones alternativas de esa misma especificación como por ejemplo Open JDK o la de IBM. Estos temas se retomarán más adelante.

BIBLIOGRAFÍA

- Plataforma Java: https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html
- Componentes de la Plataforma Java: https://docs.oracle.com/javase/8/docs/index.html
- Características del Lenguaje Java: https://www.oracle.com/technetwork/java/langenv-140151.html
- Como aprender a programar en Java 9na Edición. Deitel-Deitel. Pearson Educación, México (2012).