Notas técnicas de JAVA Nro. 4 – White Paper

(Lo nuevo, lo escondido, o simplemente lo de siempre pero bien explicado)

JAVA "Basics": Entendiendo la Java Virtual Machine (JVM)

Tema: Java, JVM, objetos, introducción, compilador

Descripción: Este artículo es el segundo de una serie de tips, destinados a cubrir los conceptos introductorios del lenguaje. Como parte de la serie, se publicarán también: Java y applets, JAVA y Servlets, JAVA y la red, Mitos y Verdades de JAVA, Preguntas Frecuentes

Nivel: Básico

Fecha pub: 2004

---

*"Notas Técnicas de JAVA" se envía con frecuencia variable y absolutamente **sin cargo** como un servicio a nuestros clientes. Contiene notas/recursos/artículos técnicos desarrollados en forma totalmente objetiva e independiente. Teknoda es una organización de servicios de tecnología informática y **NO comercializa hardware, software ni otros productos**. Si desea suscribir otra dirección de e-mail para que comience a recibir los tips envíe un mensaje desde esa dirección a develop@teknoda.com, indicando su nombre, empresa a la que pertenece, cargo y país.*

Lista de Tips publicados hasta la fecha:

1. JAVA Basics: Cómo conformar un entorno de programación JAVA (serie de varios tips). Parte I: Selección e instalación de un IDE gratuito.
2. Una introducción a JDBC (Java Database Connectivity) (Acceso a bases de datos desde JAVA)
3. Manejo del error "Bad Magic Number"
4. Java Basics: Entendiendo la Java Virtual Machine

Próximos Tips:

Nivel Técnico avanzado

- Organización de memoria en JAVA Vs. Modelo Tradicional
- JAVA Vs. C++

Nivel Básico

- JAVA Basics: Entendiendo los applets
- JAVA Basics: Entendiendo los servlets
- JAVA Basics: Mitos y Verdades sobre JAVA

Tabla de contenido

El presente White Paper forma parte de una serie de artículos destinados a clarificar conceptos básicos sobre el entorno JAVA. Si bien están destinados a individuos que se inician con el lenguaje, aún los programadores de cierta experiencia pueden beneficiarse con estos artículos, ordenando sus conocimientos.

I. La singularidad de JAVA

Java surgió en 1991 cuando un grupo de ingenieros de **Sun Microsystems** trataron de diseñar un nuevo **lenguaje de programación** destinado a electrodomésticos y dispositivos electrónicos. La reducida potencia de cálculo y memoria de estos aparatos llevó a desarrollar un lenguaje sencillo capaz de generar código de tamaño muy reducido. Además se necesitaba desterrar la metodología de trabajo que se basaba en aplicar un nuevo lenguaje de programación para cada nueva CPU o chip utilizado. La clave consistió en desarrollar un código "neutro" el cual estuviera preparado para ser ejecutado sobre una "máquina hipotética o virtual", denominada **Java Virtual Machine (JVM) .** La JVM o maquina virtual pasó a ser una marca registrada de la plataforma JAVA**.** A pesar de los esfuerzos realizados por sus creadores, ninguna empresa de electrodomésticos se interesó por el nuevo lenguaje.

Lejos de su aplicación original, JAVA apareció como una opción interesante a partir del desarrollo de Internet, donde justamente se requería distribuir aplicaciones (applets en un inicio) destinadas a ejecutar en una plataforma no controlable por el desarrollador. A finales de 1995, JAVA pisa el terreno de la programación gracias a la incorporación de un intérprete Java en el programa Netscape Navigator, produciendo una verdadera revolución en Internet. Luego, a principios de 1997, Java 1.1 apareció mejorando sustancialmente la primera versión del lenguaje.

La genuina portabilidad, es la primera característica singular de JAVA, que la diferencia del cualquier otro lenguaje. Pero además, JAVA es el primer lenguaje que implenta completamente el paradigma de objetos. Al programar en Java no se parte de cero. Cualquier aplicación que se desarrolle "cuelga" (o se apoya, según cómo se quiera ver) en un gran número de **clases preexistentes**. Algunas de ellas "confeccionadas" por el propio usuario, otras comerciales, pero siempre hay un número muy importante de clases que forman parte del propio lenguaje (el **API** o Application Programming Interface de Java).

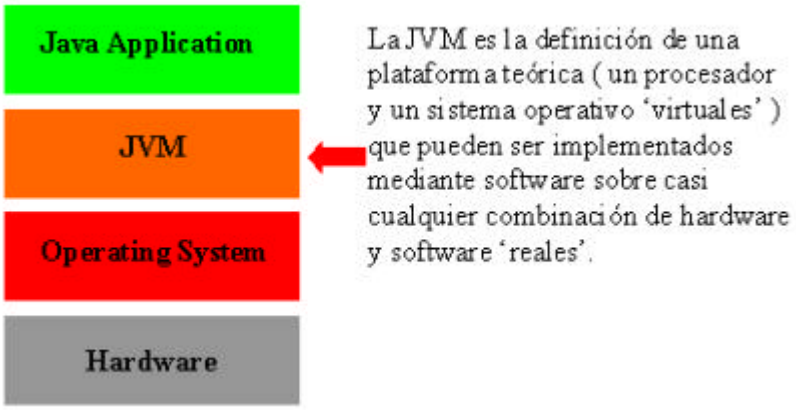
Java incorpora, de un modo estándar, mucho más sencillo y claro que otros lenguajes, muchos aspectos que en cualquier otro lenguaje son extensiones propiedad de empresas de software o fabricantes de ordenadores (threads, ejecución remota, componentes, seguridad, acceso a bases de datos, etc.). Esto es consecuencia de haber sido diseñado más recientemente y por un único equipo.

JAVA hoy ha madurado y crecido para ser considerado en aplicaciones importantes, de misión crítica, y debe verse más como una plataforma que como un lenguaje, debido a la extensión de su aplicación y portabilidad.

II. Conceptos de JVM

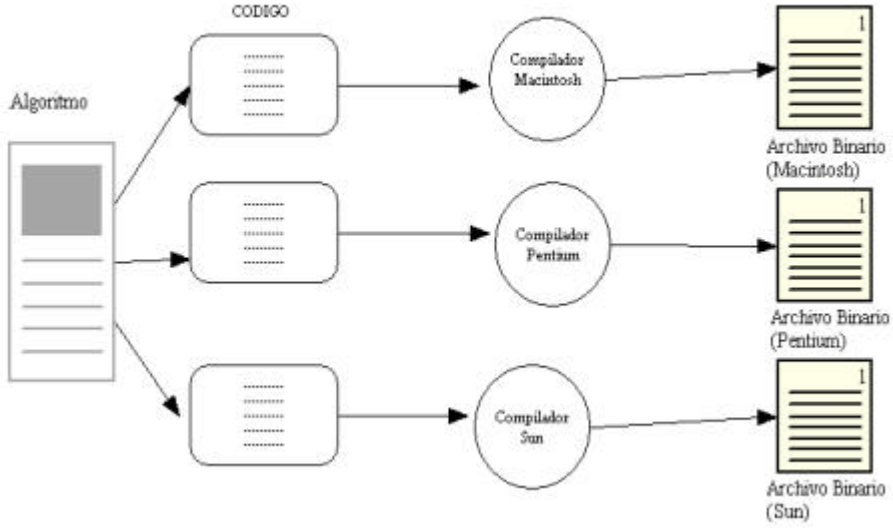
Java es un lenguaje independiente del entorno operativo. Al compilar el código JAVA, lo que se genera es un código intermedio denominado **ByteCode (80% de las instrucciones de la aplicación)**. Este código intermedio es de muy bajo nivel, pero no alcanza las instrucciones de máquina.

Ese mismo código es el que se puede ejecutar sobre cualquier plataforma. Siempre que se corre un programa Java, las instrucciones que lo **componen no son ejecutadas directamente por el hardware** sobre el que subyace, sino que son pasadas a un elemento de software intermedio, que es el encargado de que las instrucciones sean ejecutadas por el hardware. Es decir, el código Java no se ejecuta directamente sobre un procesador físico, sino sobre un **procesador virtual Java**. Para ello hace falta un *runtime*, que sí es completamente dependiente de la máquina y del sistema operativo que interpreta dinámicamente el ByteCode y añade el 20% de instrucciones que faltaban para su ejecución. Esta es la función que provee la **Java Virtual Machine JVM**: proveer el *runtim y el procesador virtual JAVA* correspondiente al sistema operativo utilizado.

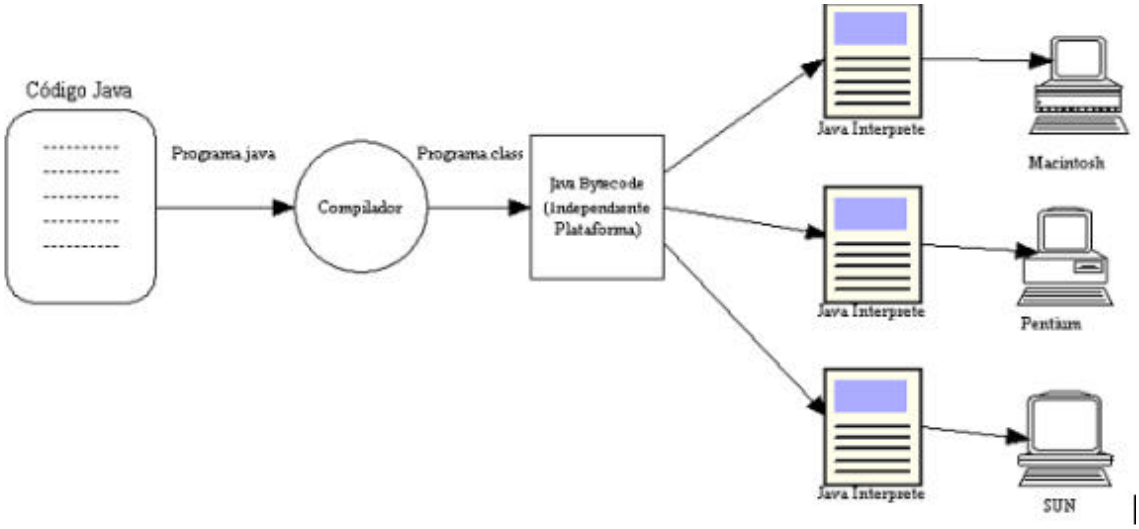


La portabilidad de JAVA reside, entonces, en la JVM, la cuál es el núcleo del lenguaje de programación Java. De hecho, es imposible ejecutar un programa Java si no se cuenta con la Java Virtual Machine (JVM) instalada. En ésta se encuentra el motor que en realidad ejecuta el programa Java y es la clave de muchas de las características principales de Java, como la portabilidad, la eficiencia y la seguridad. Las JAVA APIs son una gran colección de componentes de software listos para ser usados. Están agrupadas en librerías (paquetes) de componentes relacionados. Las JAVA APIs y la Máquina Virtual aislan el programa JAVA de las dependencias de Hardware y Sistema Operativo.

El siguiente esquema muestra la forma de programar tradicionalmente y cómo se resuelve la portabilidad en los distintos entornos:



En cambio, en el esquema de más abajo, se observa cómo se trabaja bajo la plataforma Java y cómo se maneja con respecto a la portabilidad. Observemos que un mismo programa Java codificado en una máquina con procesador Pentium puede correr (sin hacer ningún cambio en las lineas de código) en una máquina Macintosh.



III. Funcionalidad de la Java Virtual Machine

La Java Virtual Machine (JVM) es un programa que simula los componentes primarios de una computadora. Administra su propia memoria y ejecuta su propio juego de instrucciones de bajo nivel. Debido a esto, cualquier plataforma que pueda ejecutar una JVM, podrá a su vez ejecutar aplicaciones hechas en Java, ya que

la JVM hace de intermediaria entre ambas. En teoria, una JVM ejecutándose en un teléfono celular podría ejecutar una misma aplicación que en una JVM funcionando en un servidor de última generación.

Es importante mencionar que la JVM no responde a una única implementación. Sun Microsystems ha definido únicamente una especificación, que en el tiempo recibe algunas actualizaciones, pero que impone muy pocas restricciones en cuanto a la implementación interna de cada JVM. Por lo tanto, en el presente tip existirán ciertos puntos que representan mas bien una de las posibles opciones existentes, no necesariamente a alguna puntual ni mucho menos a la mayoria.

Las principales funcionalidades implementadas por la JVM son:

- Motor de ejecución. El procesador virtual que se encarga de ejecutar el código (bytecode), generado por algún compilador de Java o por algún ensamblador del procesador virtual Java.
- Manejador de memoria. Encargado de obtener memoria para las nuevas instancias de objetos, arreglos, etc., y realizar tareas de recolección de basura.
- Manejador de errores y excepciones. Encargado de generar, lanzar y atrapar excepciones.
- Soporte de métodos nativos. Encargado de llamar métodos de C++ o funciones de C, desde métodos Java y viceversa.
- Interfaz multihilos. Encargada de proporcionar el soporte para hilos y monitores.
- Cargador de clases. Su función es cargar dinámicamente las clases Java a partir de los archivos de clase (.class).
- Administrador de seguridad. Se encarga de asegurar que las clases cargadas sean seguras, así como controlar el acceso a los recursos del sistema.

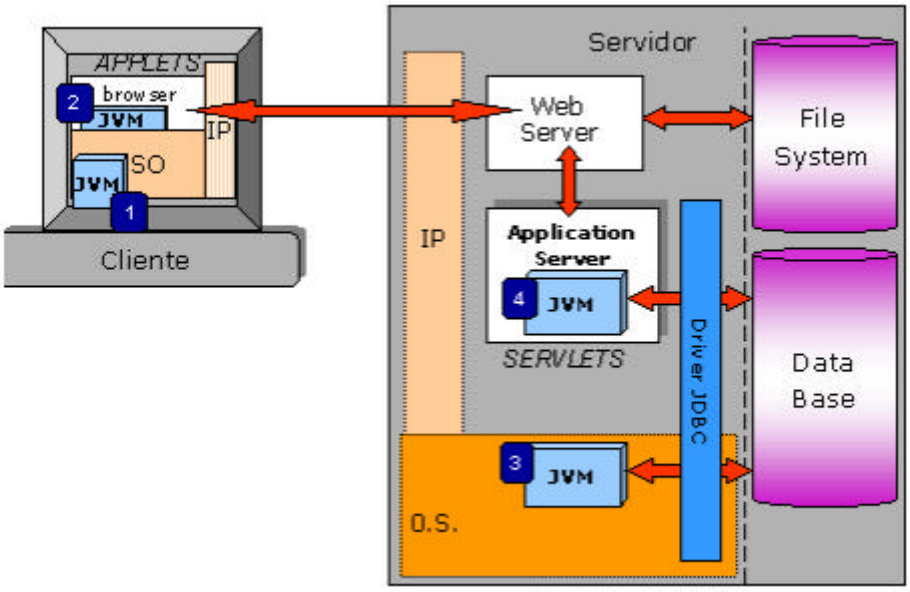
Java tiene la inteligencia del *"make"* en el propio lenguaje para simplificar la compilación de byte-codes. Esto significa que el compilador Java "se da cuenta" qué archivos necesita compilar por dependencia de los mismos. Además no recompila innecesariamente archivos ya compilados a menos que los mismos fueran modificados y se estuviera su uso en demanda.

Un paso adelante en el rendimiento del código Java lo han representado los **compiladores Just-In-Time**, que compilan el código convirtiéndolo a código de máquina antes de ejecutarlo. Es decir, un compilador JIT va trasladando los ByteCodes al código máquina de la plataforma según los va leyendo, realizando un cierto grado de optimización. El resultado es que cuando el programa se ejecute, habrá partes que no se ejecuten y que no serán compiladas, y el compilador JIT no perderá el tiempo en optimizar código que nunca se va a ejecutar. En síntesis: "los compiladores JIT siempre son capaces de optimizar la parte de código de inicialización de un programa. Debido a que hay código que ellos no ven, existen partes del programa que deben ser optimizadas según se van cargando, con lo cual, hay una cierta cantidad de tiempo que inevitablemente ha de perderse y también existe un cierto ahorro de tiempo al no compilar las partes de código que no se ejecutarán".

La última tendencia en lo que a compilación e intérpretes se refiere es **HotSpot** de Sun, una herramienta que incluye un compilador dinámico y una máquina virtual para interpretar los ByteCodes. Cuando se cargan los ByteCodes producidos por el compilador por primera vez, éstos son interpretados en la máquina virtual. Cuando ya están en ejecución, el *profiler* mantiene información sobre el rendimiento y selecciona el método sobre el que se va a realizar la compilación. Los métodos ya compilados se almacenan en un caché en código de máquina nativo. Cuando un método es invocado, esta versión en código de máquina nativo es la que se utiliza, en caso de que exista; en caso contrario, los ByteCodes son reinterpretados.

IV. Puntos o contextos de ejecución de Java

Justamente, en virtud de lo expuesto sobre el rol de la JAVA Virtual Machine, es importante entender que aún en un mismo escenario Cliente-Servidor es posible ejecutar código JAVA en distintos contextos o ubicaciones del cliente y/o del servidor, según la JVM que se encargue de ejecutarlas. Estas diferentes aplicaciones de JAVA reciben distintas designaciones, y tienen distintas características. (Ej.: JAVA Servlets, JAVA Applets, JAVA Applications)



1. Aplicaciones Java en el cliente (JVM del OS)

Comunmente, son aplicaciones de escritorio que corren en el cliente, utilizan la **JVM "local"**. Las mismas pueden ser ejecutadas mediante un archivo del tipo .bat o un archivo .exe. La JVM se instala ad-hoc, o está provista como parte del sistema operativo.

2. Applets Java (JVM del Browser)

Las Applets son aplicaciones web que se ejecutan en el **JVM local del Web Browser** . Son pequeñas ventanas que se despliegan del lado de cliente. Por ejemplo, Internet Explorer, o Netscape Navigator tienen incorporada una JVM como parte de su funcionalidad. Las applets son comunes en aplicaciones acotadas, distribuidas generalmente a través de Internet.

3.Aplicaciones Java en el Server (JVM del OS)

Estas " Aplicaciones Web " corren gracias a la JVM del lado del Servidor, es decir el cliente va poder utilizar esta " Aplicación Web " independientemente que posea o no una máquina virtual de Java. Es habitual que este tipo de aplicaciones estén conectadas a una base de datos a través de una conexión JDBC

Para más información (ver tip *Una introducción a JDBC (Java Database Connectivity) Acceso a bases de datos desde JAVA*)

4.Servlets Java (JVM del Application Server)

Las Servlets se ejecutan también en la JVM del Application Server del lado del Servidor, pero son gestionadas por requerimientos emanados del cliente, a través de una URL. **Requieren de un software específico, denominado JAVA WEB APPLICATION SERVER**. Nuevamente se repite la situación de independencia de la JVM del lado del cliente. El cliente puede trabajar con la aplicación desarrollada con servlets sin necesidad de tener instalada la maquina virtual de Java .

V. Dónde Obtener Información Adicional

www.java.sun.com

www.java.sun.com/docs/books/tutorial/jdbc/

www.sun.com/developer-products/java/

wwws.sun.com/software/sundev/jde/index.html

www.java.sun.com/products/plugin/

www.borland.com

www.symantec.com

Dudas o consultas ***develop@teknoda.com***