LA MÁQUINA VIRTUAL DE JAVA: La Portabilidad del Software

CARLOS ALBERTO VANEGAS*
cvanegas.udtecnologica.edu.co
SONIA ALEXANDRA PINZÓN
spinzon@udistrital.edu.co

1. Introducción

Los lenguajes de programación se utilizan para desarrollar programas, y en general todas las aplicaciones que cotidianamenente son utilizadas por miles de usuarios: graficadores, hojas de cálculo, procesadores de texto, juegos, etc. Sin embargo, estas aplicaciones requieren de una plataforma específica para poder ser ejecutados; por ejemplo las aplicaciones Windows sólo pueden ser ejecutadas en un sistema Windows, no en sistemas Macintosh, y viceversa. Esto no sucede con los programas realizados en Java, ya que ellos se pueden ejecutar en cualquier tipo de computador (con sistema Windows, Macintosh, Unix, etc.), gracias a su Máquina Virtual.

2. Lenguaje Java

Java es un lenguaje de programación que tiene una gran ventaja sobre los lenguajes tradicionales, al permitir crear programas para una plataforma universal[1]: *la plataforma Java.* La independencia de la plataforma es un factor importante de éxito; sus programas se compilan en código de bytes[2] (bytecodes) y pueden ejecutarse en cualquier computador con un intérprete o Máquina Virtual de Java (MVJ), la cual proporciona un ambiente en tiempo de ejecución para ejecutar programas de Java[3].

* Ingeniero de Sistemas Universidad Incca de Colombia, Especialista en Ingeniería de Software Universidad Distrital F.J.C., Estudios de Maestría en Ingeniería de Sistemas Universidad Nacional de Colombia. Profesor adscrito a la Facultad Tecnológica de la Universidad Distrital F.J.C.
** Ingeniera de Sistemas y Especialista en Multimedia Educativa Universidad Antonio Nariño, Estudios de Especialización en Educación en Tecnología Universidad Distrital F.J.C., Profesora adscrita a la Facultad Tecnológica de la Universidad Distrital

1 Una plataforma es una combinación de hardware y software; por ejemplo, los computadores con procesadores Intel (como el Pentium) y sistema operativo Windows son una plataforma; los computadores Macintosh y el sistema operativo Mac son otra plataforma

2 Es el código de máquina que traduce la Máquina Virtual de Java.

3 WANG, Paul S. Java con programación Orientada a Objetos y Aplicaciones en la WWW. 2000, p.1.

PALABRAS CLAVES

JAVA, MÁQUINA VIRTUAL JAVA, LENGUAJE INTERPRETADO, LENGUAJE COMPILADO, BYTECODE

El lenguaje Java es a la vez compilado e interpretado. La Figura 1 muestra cómo con el compilador convierte el código fuente almacenado en archivos de extensión ***java***, a un conjunto de instrucciones que recibe el nombre de *bytecodes* que se guardan en un archivo de extensión ***class***. Luego, mediante un proceso de intérprete, este programa puede ser ejecutado en cualquier plataforma.

De esta manera, Java integra dos tecnologías, al poseer la flexibilidad de los lenguajes interpretados y la eficiencia de los compilados, para garantizar la portabilidad de los programas hechos en este lenguaje.

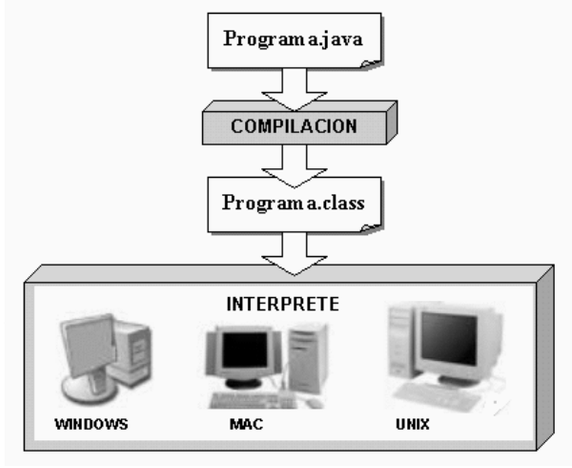


Figura 1. Portabilidad de Java

2.1 Lenguajes Compilados

Son los que requieren de un programa especial llamado compilador, el cual se encarga de traducir el programa fuente (el grupo de instrucciones elaborado en el lenguaje), a su equivalente en código de máquina, también denominado *programa objeto*, para de esta forma poder ser ejecutados.

Los lenguajes compilados pasan por cuatro procesos, como se observa en la Figura 2.

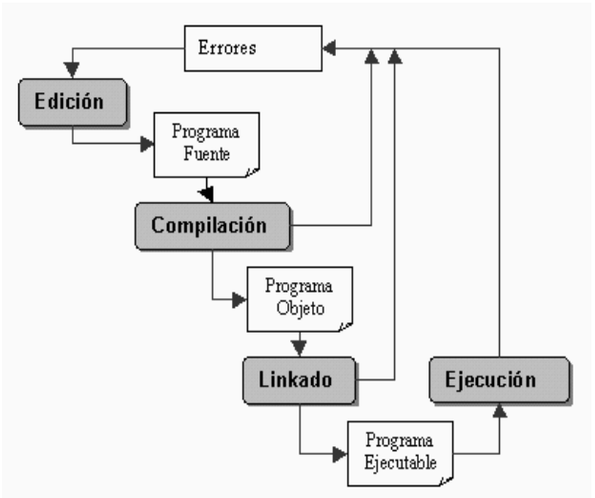


Figura 2. Proceso de Compilación de un Programa

- **Edición**: escribir las instrucciones del programa en el lenguaje de programación. El resultado de este proceso es el código fuente.
- **Compilación**: traduce el programa fuente a código de máquina, generando el programa objeto
- **Enlace**: une el programa objeto con las rutinas básicas del lenguaje de programación para generar el programa ejecutable
- **Ejecución**: es el proceso que muestra el resultado del programa cuando éste es llamado a través del sistema.

2.2 Lenguajes Interpretados

Son aquellos que, por medio de un programa llamado *intérprete*, se convierten en lenguaje de máquina y se ejecutan, independiente del sistema operativo en que se hayan elaborado; el intérprete toma cada instrucción del programa, verifica la existencia de errores de sintaxis, las traduce y las ejecuta, como se puede observar en la Figura 3. Los procesos de traducción y ejecución se realizan simultáneamente, lo cual hace que los lenguajes interpre-

tados sean más lentos en su ejecución. En otras palabras, la *portabilidad* que adquieren les disminuye su velocidad de ejecución.

3. La Máquina Virtual de Java (MVJ)

La MVJ, *"conocida también como el intérprete de Java o el ambiente de tiempo de ejecución de Java"*[4], es un módulo de software que funciona como intérprete de la representación binaria intermedia. Ella examina cada uno de los códigos binarios (bytecodes) que se han compilado y los ejecuta en el computador donde reside el interprete. El código Java puede ser ejecutado en cualquier plataforma que disponga de la MVJ; en otras palabras, un computador necesita solamente de este intérprete para ejecutar cualquier programa hecho bajo este lenguaje.

Debido a que en Java se pueden crear programas de tipo *applet* y *programas autónomos*[5], la MVJ puede encontrarse en dos formas. Si se desean ejecutar applets, ella está implícita en el Navegador que se use para visualizar, por ejemplo Netscape Navigator o Internet Explorer; para el segundo caso se requiere instalar la MVJ que se encuentra en el Kit de Desarrollo de Java JDK.

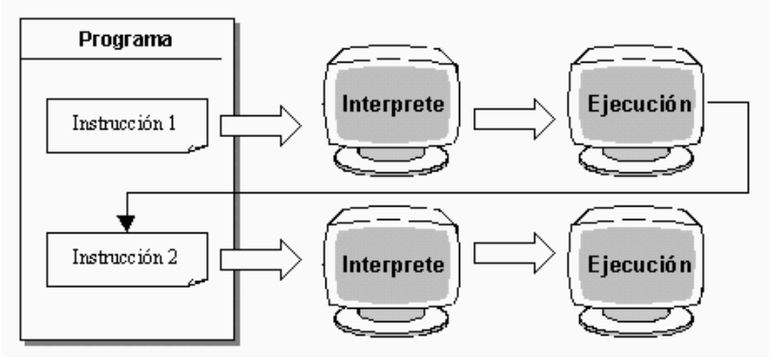


Figura 3. Proceso de Interpretación de un Programa

Las tareas principales de la MVJ son las siguientes:

- Reservar espacio en memoria para los objetos creados
- Liberar la memoria no usada (garbage collection)
- Asignar variables a registros y pilas
- Cargar y enlazar los archivos de clases que sean necesarios para la ejecución del programa
- Vigilar el cumplimiento de las normas de seguridad de las aplicaciones Java

3.1 ¿Cómo trabaja la MVJ?

Cuando el programador genera su archivo *java*, éste se compila y se convierte en un archivo *class*, que consiste en códigos de bytes, esto es, un conjunto de instrucciones para la MVJ[6].

[4] LEMAY, Laura, CADENHEAD, Rogers. Aprendiendo Java 2 en 21 Días. Ed. Prentice Hall, Mexico, 1999, p. 17

[5] Los applets son programas hechos en lenguaje Java que se pueden ejecutar dentro de una página de Internet y los programas autónomos son aquellos programas que se ejecutan por línea de comando, con una estructura similar a la de los programas hechos en lenguaje C y C++

[6] HOLZNER, Steven. La Biblia de Java. Ed. Anaya Multimedia. México, 2000.p. 635

En la Figura 4 se muestra la interacción de los componentes de la MVJ para ejecutar un programa. El proceso comienza al cargar los archivos *.class* desde la red o desde un computador local. Luego, el *verificador de código* se encarga de revisar los códigos de bytes que son cargados en la MVJ. Si estos no pasan la verificación, la ejecución se para con un mensaje de error; de lo contrario el intérprete leerá los códigos y los convertirá en instrucciones que serán ejecutadas en la plataforma[7].

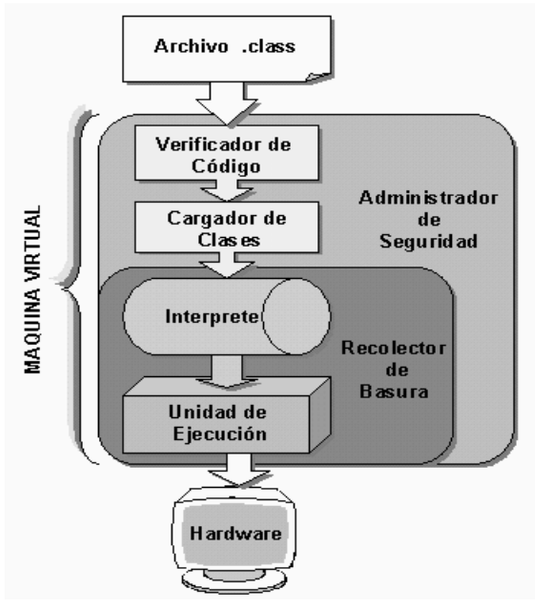


Figura 4. Funcionamiento de la Máquina Virtual de Java

El *cargador de clases* importa y combina los archivos *.class* que son invocados desde el archivo que se desea ejecutar, ya que muchos de los archivos *java* pueden referirse a otros archivos *class*, debido a la generación de objetos y clases que se puede definir en la programación Java.

En cuanto al *administrador de seguridad*, el sistema sólo permite cargar las clases que se encuentren en la misma ubicación en las que está el archivo *class* que se interpreta.

De otra parte, la MVJ permite hacer una revisión de espacio de memoria no utilizado mediante el *recolector de basura*, el cual hace una limpieza de las variables y/u objetos no usados por el programa en el momento de su ejecución.

Java es, por tanto, algo más que un lenguaje. El término se refiere a dos cosas inseparables: el lenguaje que nos sirve para crear programas, y la MVJ que sirve para ejecutarlos. Como se observa en la Figura 5, el API de Java y la MVJ forman una capa intermedia que aísla el programa Java de las especificaciones del hardware.

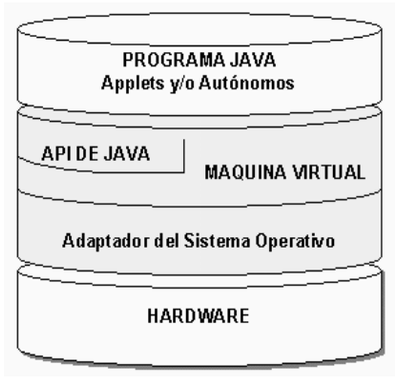


Figura 5. Capa de la Máquina Virtual Java

3.2 Componentes de la Máquina Virtual

Sun utiliza el término MVJ para referirse a la especificación abstracta de una máquina de software para ejecutar programas Java. La especificación de esta máquina virtual define elementos como el formato de los archivos de clases de Java *(class)*, así como la semántica de cada instrucción que compone el conjunto de instrucciones de la MVJ. A las implantaciones de esta especificación se les conocen como "*Sistemas en Tiempo de Ejecución Java*", por ejemplo el Navega-

[7] Los fabricantes de hardware y sistemas operativos implementan la MVJ en su combinación hardware - sistema operativo

dor de Nestcape, el Explorador de Microsoft y el programa Java (incluido en el JDK), entre otros.

Un sistema de tiempo de ejecución incluye típicamente los siguientes componentes:

- **Motor de Ejecución.** Es la entidad de hardware o software que ejecuta las instrucciones contenidas en los códigos de operación (*bytecodes*) que implementan los métodos Java. Se utiliza la tecnología de "*generación de código justo en el momento*" (*Just-in-Time code generation*), en donde las instrucciones que implementan los métodos se convierten en código nativo que se ejecuta directamente en la máquina sobre la que se programa. El código nativo se genera únicamente la primera vez que se ejecuta el código de operación Java, por lo que se logra un aumento considerable en el rendimiento de los programas[8].
- **Conjunto de Instrucciones del Procesador Virtual.** Muchas de estas instrucciones son muy similares a las que se pueden encontrar para los procesadores comunes, como los Intel; esto significa que se incluyen los grupos de instrucciones típicos (aritméticos, de control de flujo, de acceso a memoria, a la pila, etc.). Una de las características más significativas de este conjunto de instrucciones es que están basadas en la pila y utilizan *posiciones de memoria* numeradas en lugar de registros. Esto es hasta cierto punto lógico, debido a que la MVJ está pensada para correr sobre sistemas con procesadores sustancialmente diferentes.
- **El Verificador de Java.** Se ejecuta sobre un archivo *class* y tiene la función de asegurar que los códigos de bytes Java se ajustan a la especificación de la MVJ. Fundamentalmente es uno de los niveles de seguridad incorporado en la plataforma Java para chequear que virus u otros programas corruptos no realicen daños sobre el sistema local. Además permite comprobar la corrección sintáctica y semántica de los archivos de clases, así como que la versión se ajuste al API.

Por otra parte, el verificador protege contra errores en el compilador o el añadido intencionado de códigos de bytes maliciosos. Los desbordamientos por exceso y defecto de los valores de las operaciones aritméticas se identifican durante la ejecución del verificador. También lleva a cabo otros chequeos en tiempo de ejecución, lo que hace que el intérprete de Java sea más rápido[9]. Este verificador realiza su función en cuatro fases, como se describe a continuación y se representa en la Figura 6.

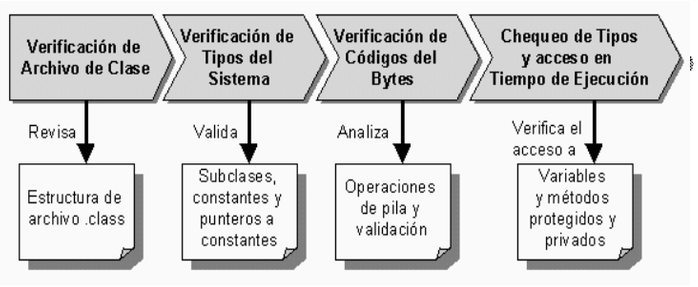
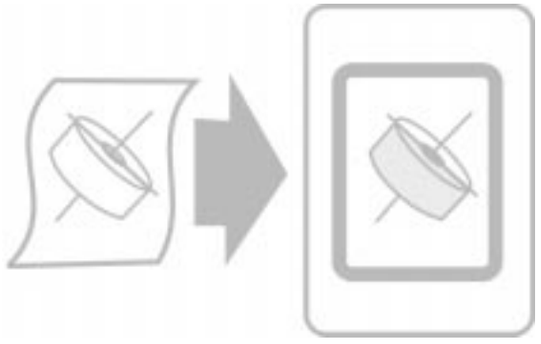


Figura 6. Fases del Verificador de Java

Lo primero que se realiza es la *verificación del archivo de clase*, en donde se comprueba la estructura del archivo *.class*, el cual debe comenzar por el código 0xCAFEBABE, Además se verifica que no existan bytes adicionales al final de él. La segunda fase comprende la *verificación de tipos del sistema*, es decir, la comprobación de subclases, constantes y punteros a constantes, definiendo la validez de los mismos para que no se generen acciones no permitidas.

[8] **MENCHACA M. Rolando, GARCÍA C., Félix**. Arquitectura de la Máquina Virtual de Java. En: Revista Digital Universitaria. UNAM de México. www.revista.unam.mx/vol.1/num2/art4/

[9] HOLZNER, Steven, op. cit., p. 650



La tercera fase es la *verificación de los códigos de bytes*, donde se examina exhaustivamente cada método analizando las operaciones de pila, validación de argumentos e inicialización de variables. Finalmente se hace *un chequeo de tipos de acceso en tiempo de ejecución.* De esta forma se observa si los niveles de acceso a las variables y los métodos no se han sobrepasado, es decir, si los códigos de bytes no han alterado variables, métodos protegidos o privados.

- **Administrador de Memoria.** Java utiliza un modelo de memoria conocido como "*administración automática del almacenamiento*" (*automatic storage management*), en el que el sistema en tiempo de ejecución mantiene un seguimiento de los objetos. En el momento que no están siendo referenciados por alguien, automáticamente se libera la memoria asociada con ellos. El proceso es ejecutado por el programa llamado *recolector de basura* (garbage collection). Existen muchas maneras de implementar recolectores de basura, entre ellas están:
  - ***Contabilizar referencias.*** La MVJ asocia un contador a cada instancia de un objeto, donde se refleja el número de referencias hacia él. Cuando este contador es 0, la memoria asociada al objeto es susceptible de ser liberada
  - ***Marcar e intercambiar (Mark-and-Sweep).*** Este es el esquema más común para implementar el manejo de almacenamiento automático. Consiste en almacenar los objetos en un montículo (*heap*) de un tamaño considerable y marcar periódicamente (generalmente mediante un bit en un campo que se utiliza para este fin) los objetos que no tengan ninguna referencia hacia ellos. Adicionalmente existe un montón alterno, en el cual los objetos que no han sido marcados son movidos periódicamente. Una vez en el montículo alterno, el recolector de basura se encarga de actualizar las referencias de los objetos a sus nuevas localidades. De esta manera se genera un nuevo montículo, que contiene únicamente objetos que están siendo utilizados[10].
- **Administrador de Errores y Excepciones.** Las excepciones son la manera como Java percibe alguna anomalía durante la ejecución de un programa Java. Comúnmente son generadas y lanzadas por el sistema, cuando uno de estos eventos ocurre. Así mismo, los métodos tienen la capacidad de lanzar excepciones, utilizando la instrucción de la MVJ *throw*.

  Todas las excepciones en Java son instancias de la clase *java.lang.Throwable,* o de alguna otra que la especialice. Las clases *java.lang.Exception* y *java.lang.Error* heredan directamente de java.lang.Throwable. La primera se utiliza para mostrar eventos de los cuales es posible recuperarse, como la lectura del fin de archivo o la falla de la red, mientras que la segunda se utiliza para indicar situaciones de las cuales no es posible recuperarse, como un acceso indebido a la memoria. Cuando se genera una excepción, el sistema de tiempo de ejecución de Java, y en particular el manejador (*handler*) de errores y excepciones, busca un manejador para esa excepción, comen-

[10] **MENCHACA R., GARCÍA C.,** op. cit., p. 1.
[11] Ibid, p. 1

zando por el método que la originó y luego hacia abajo en la pila de llamadas. Cuando se encuentra uno, éste atrapa la excepción y se ejecuta el código asociado con él. Lo que ocurre después depende del código del manejador[11]; en el caso que no se encuentre uno para alguna excepción previamente lanzada, se ejecuta el manejador del sistema, cuya acción típica es imprimir un mensaje de error y terminar la ejecución del programa.

Como se mencionó anteriormente, para generar una excepción se utiliza la instrucción *throw*, que toma un elemento de la pila. Dicho elemento debe ser la referencia a un objeto que herede de la clase java.lang.Throwable.

- **Soporte para Métodos Nativos.** Las clases en Java pueden contener métodos que no estén implementados por códigos de operación (*bytecode*) Java, sino por algún otro lenguaje compilado en código nativo y almacenado en bibliotecas de enlace dinámico, como las DLL de Windows o las bibliotecas compartidas SO de Solaris. El sistema de tiempo de ejecución incluye el código necesario para cargar dinámicamente y ejecutar el código nativo que implementa estos métodos. Una vez que se enlaza el módulo que contiene el código que implementa el método, el procesador virtual atrapa las llamadas a éste y se encarga de invocarlo. Este proceso incluye la modificación de los argumentos de llamada para adecuarlos al formato que requiere el código nativo, así como transferirle el control de la ejecución. Cuando el código nativo termina, el módulo de soporte para métodos nativos se encarga de recuperar lo resultados y adecuarlos al formato de la MVJ. De manera análoga, el módulo de soporte para código nativo se encarga de canalizar una llamada a un método escrito en Java, hecha desde un procedimiento o método nativo.

- **Interfaz de Hilos.** Java es un lenguaje que permite la ejecución concurrente de varios hilos de ejecución. El sistema de tiempo de ejecución de Java tiene entonces la posibilidad de crear más de un procesador virtual para ejecutar diferentes flujos de instrucciones, cada uno con su propia pila y su propio estado local. Ellos pueden ser simulados por software o implementados mediante llamadas al sistema operativo. En el conjunto de instrucciones de la MVJ sólo existen dos directamente relacionadas con los hilos, *monitorenter* y *monitorexit*, que sirven para definir secciones de código que deben ejecutarse en exclusión mutua. El resto del soporte de los hilos se realiza atrapando llamadas a los métodos pertenecientes a la clase java.lang.Thread.

- **Cargador de Clases.** Los programas Java están completamente estructurados en clases. Por lo tanto una función muy importante del sistema en tiempo de ejecución es cargar, enlazar e inicializar clases dinámicamente, de forma que sea posible instalar componentes de software en tiempo de ejecución. El proceso de cargado de las clases se realiza sobre demanda, hasta el último momento posible. La MVJ utiliza dos mecanismos para cargar las clases. El primero consiste en un cargador de clases del sistema, cuya función es cargar las clases estándar de Java, así como la clase cuyo nombre es entrado a través de la línea de comandos. Existe también un segundo mecanismo para cargar clases, utilizando una instancia de la clase *java.lang.ClassLoader* o alguna otra definida por el usuario, que especialice a la anterior.

4. Desventajas de las Máquinas Virtuales

Cuando la MVJ interpreta cada instrucción para luego ejecutarla el sistema se hace lento; esto significa que la portabilidad del sistema sacrifica la velo-

[12] HOLZNER, steven, op . cit., p. 639

cidad de ejecución. De otra parte, el proceso de verificación lleva su tiempo, ya que ella se hace en tiempo de ejecución. Otra desventaja es la disminución en el rendimiento del sistema, debido a que el tamaño de cada instrucción de Java es de un byte[12], sin tener en cuenta que los objetos pueden tener un tamaño considerable y tienen que generarse varios códigos de bytes para obtener los diferentes operadores y sus parámetros.

Así, la principal desventaja de los lenguajes basados en MVJ es que, efectivamente, son más lentos que los lenguajes completamente compilados, debido a la sobrecarga que genera tener una capa de software intermedia entre la aplicación y el hardware del computador. Esta desventaja no es demasiado crítica, ya que se obtiene un programa que puede ejecutarse en diferentes tipos de plataformas.

5. Conclusiones

- Los programas de Java pueden ser ejecutados en cualquier plataforma que disponga de la MVJ. Eso quiere decir que un computador necesita solamente de ella para ejecutar cualquier programa hecho bajo este lenguaje. Por eso se dice que los programas de Java son *portables*
- Dentro de las principales tareas que realiza la MVJ se encuentran: reserva de espacio en memoria para los objetos creados, liberación de memoria no usada, asignación de variables a registros y pilas, cargar y enlazar los archivos de clases que sean necesarios para la ejecución del programa
- Como la MVJ interpreta cada instrucción para luego ejecutarla el sistema se hace lento; la portabilidad del sistema sacrifica entonces la velocidad de ejecución.

REFERENCIAS BIBLIOGRÁFICAS

- HOLZNER, steven. La Biblia de Java. Ed. Anaya Multimedia. México, 2000.
- LEMAY, Laura, CADENHEAD, Rogers. Aprendiendo Java 2 en 21 Días. Ed. Prentice Hall, Mexico, 1999
- MENCHACA M. Rolando. GARCÍA C. Félix. Arquitectura de la Máquina Virtual de Java. En: Revista Digital Universitaria. UNAM de México. http:// www.revista.unam.mx/ vol.1/ num2/ art4/
- VENNERS, Bill. Inside the Java Virtual Machine. Ed. McGraw-Hill
- WANG, Paul S. Java con Programación Orientada a Objetos y Aplicaciones en la WWW. Ed. Thomson, México, 2000

INFOGRAFÍA

http://www.javasoft.com/docs/books/vmspec.

http://java.sun.com/products/jdk/1.3/docs/guide/security.

http://www.kaffe.org.

http://java.sun.com/products/jdk/1.3/docs/guide/jni.