UNIVERSIDAD CATOLICA DE COLOMBIA
FACULTAD DE INGENIERIA DE SISTEMAS

CURSO: JAVA BASICO
PROFESOR: EMERSON CASTAÑEDA SANABRIA

TEMA: Introducción a Java

OBJETIVOS:

- Conocer como se origino el lenguaje de programación java.
- Establecer las características de Java como lenguaje de programación.
- Proporcionar una información del conjunto de herramientas de desarrollo del lenguaje y de la interfaz de programación desde el punto de vista de un creador de software.
- Establecer unas pautas que faciliten la lectura y el mantenimiento de los programas.

CONTENIDO:

1. Historia
2. Características
3. El entorno de desarrollo
4. Nomenclatura

DESARROLLO:

1. Historia

Java surgió en 1991 cuando un grupo de ingenieros de Sun Microsystems trataron de diseñar un nuevo lenguaje de programación destinado a electrodomésticos. La reducida potencia de cálculo y memoria de los electrodomésticos llevó a desarrollar un lenguaje sencillo capaz de generar código de tamaño muy reducido.

Java fue concebido por James Gosling, Patrick Naughton, Chis Warth, Ed Frank y Mike Sherindan en Sun Microsystem Inc. En su primera versión que duro unos 18 meses. Entre 1.992 y 1.995 colaboraron en la madurez del prototipo inicial Bill Joy, Arthur Van Hoff, Jonathan Payne, Frank Yellia, y Tim Lindolm.

Debido a la existencia de distintos tipos de CPU y a los continuos cambios, era importante conseguir una herramienta independiente del tipo de CPU utilizada. Desarrollaron un código "neutro" que no dependía del tipo de electrodoméstico, el cual se ejecutaba sobre una "máquina hipotética o virtual" denominada Java Virtual Machine (JVM). Era la JVM quien interpretaba el código neutro convirtiéndolo a código particular de la CPU utilizada. Esto permitía lo que luego se ha convertido en el principal lema del lenguaje: "Write Once, Run Everywhere". A pesar de los esfuerzos realizados por sus creadores, ninguna empresa de electrodomésticos se interesó por el nuevo lenguaje.

Como lenguaje de programación para computadores, Java se introdujo a finales de 1995. La clave fue la incorporación de un intérprete Java en la versión 2.0 del programa Netscape Navigator, produciendo una verdadera revolución en Internet. Java 1.1 apareció a principios de 1997, mejorando sustancialmente la primera versión del lenguaje. Java 1.2, más tarde rebautizado como Java 2, nació a finales de 1998.

Al programar en Java no se parte de cero. Cualquier aplicación que se desarrolle depende (o se apoya, según como se quiera ver) en un gran número de clases preexistentes. Algunas de ellas las ha podido hacer el propio usuario, otras pueden ser 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 en el propio lenguaje 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.). Por eso muchos expertos opinan que Java es el lenguaje ideal para aprender la

informática moderna, porque incorpora todos estos conceptos de un modo estándar, mucho más sencillo y claro que con las citadas extensiones de otros lenguajes. Esto es consecuencia de haber sido diseñado más recientemente y por un único equipo.

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

Java es un lenguaje muy completo (de hecho se está convirtiendo en un macro-lenguaje: Java 1.0 tenía 12 packages; Java 1.1 tenía 23 , Java 1.2 tiene 59 y Java 1.3 tiene 70). En cierta forma todo depende de todo. Por ello, conviene aprenderlo de modo iterativo: primero una visión muy general, que se va refinando en sucesivas iteraciones. Una forma de hacerlo es empezar con un ejemplo completo en el que ya aparecen algunas de las características más importantes.

La compañía Sun describe el lenguaje Java como "simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico". Además de una serie de halagos por parte de Sun hacia su propia criatura, el hecho es que todo ello describe bastante bien el lenguaje Java, aunque en algunas de esas características el lenguaje sea todavía bastante mejorable. Algunas de las anteriores ideas se irán explicando a lo largo de la guía.

QUÉ ES JAVA 2

Java 2 (antes llamado Java 1.2 o JDK 1.2) es la tercera versión importante del lenguaje de programación Java. No hay cambios conceptuales importantes respecto a Java 1.1 (en Java 1.1 sí los hubo respecto a Java 1.0), sino extensiones y ampliaciones, lo cual hace que a muchos efectos –por ejemplo, para esta introducción- sea casi lo mismo trabajar con Java 1.1 o con Java 1.2. Los programas desarrollados en Java presentan diversas ventajas frente a los desarrollados en otros lenguajes como C/C++. La ejecución de programas en Java tiene muchas posibilidades: ejecución como aplicación independiente (Stand-alone Application), ejecución como applet, ejecución como servlet, etc. Un applet es una aplicación especial que se ejecuta dentro de un navegador o browser (por ejemplo Netscape Navigator o Internet Explorer) al cargar una página HTML desde un servidor Web. El applet se descarga desde el servidor y no requiere instalación en el computador donde se encuentra el browser. Un servlet es una aplicación sin interface gráfica que se ejecuta en un servidor de Internet. La ejecución como aplicación independiente es análoga a los programas desarrollados con otros lenguajes.

Además de incorporar la ejecución como Applet, Java permite fácilmente el desarrollo tanto de arquitecturas cliente-servidor como de aplicaciones distribuidas, consistentes en crear aplicaciones capaces de conectarse a otros computadores y ejecutar tareas en varios computadores simultáneamente, repartiendo por lo tanto el trabajo. Aunque también otros lenguajes de programación permiten crear aplicaciones de este tipo, Java incorpora en su propio API estas funcionalidades.

2. Características

Antes de seguir, habría que preguntarse ¿qué es Java?, y como respuesta podemos tener (como en el manual del JDK indica) que son dos cosas, por un lado un lenguaje de programación y por otro una plataforma.

Como lenguaje de programación Java es un lenguaje de alto nivel con las siguientes características:

Es simple: En Java hay un número reducido de formas claras para abordar una tarea dada. Ofrece toda la funcionalidad de un lenguaje potente pero sin las características menos usadas y más confusas de estas. Hereda la sintaxis de C/C++ y muchas de las características orientadas a objetos de C++. Todos los programadores que conozcan C/C++ no tendrán ningún problema para aprender Java. Elimina algunas características de estos lenguajes, entre los que destacan:

- Aritmética de punteros
- Registros (struct)
- Definición de tipos (typedef)
- Macros (#define)
- Necesidad de liberar memoria (free)
- No hay herencia múltiple
- No hay sobrecarga de operadores
- No hay estructura un uniones

Es orientado a objetos: Java es un lenguaje diseñado partiendo de cero, como resultado de esto se realiza una aproximación limpia, útil y pragmática a los objetos. El modelo de objeto de Java es simple y fácil de ampliar.

Es distribuido: Java fue diseñado con extensas capacidades de interconexión TCP/IP. De hecho permite a los programadores acceder a la información a través de la red con tanta facilidad como a los Archivos locales.

Es robusto: Java es fuertemente tipado, por lo que permite comprobar el código en tiempo de compilación. Además también comprueba el código en tiempo de ejecución. La liberación de memoria se realiza de forma automática, ya que se proporciona un recolector de basura automático para los objetos que no se utilizan. Java asimismo proporciona una gestión de excepciones orientada a objetos. Un programa escrito correctamente todos los errores de ejecución pueden y deben ser gestionados por el programa.

Es de arquitectura neutral: El principal objetivo de los diseñadores de Java era "escribir una vez, ejecuta en cualquier sitio, en cualquier momento y para siempre". El código Java se compila a un código de bytes de alto nivel independiente de la maquina. Este código esta diseñado para ejecutarse en cualquier máquina con un sistema run-time (interprete) el cual si es dependiente de esta.

Es seguro: Las necesidades de la informática distribuida exigen los mayores niveles de seguridad para los sistemas operativos clientes. Java proporciona seguridad a través de varias características de su entorno en tiempo de ejecución:

- Un verificador de bytecodes
- La disposición de memoria en tiempo de ejecución
- Restricciones de acceso a los archivos.

Aunque el compilador solo genere código correcto el interprete lo vuelve a verificar para asegurarse que el código no ha sido cambiado (intencionadamente o no) entre el momento de la compilación y la ejecución. Además el interprete Java determina la disposición de la memoria para las clases. Se puede considerar a Java uno de las aplicaciones más seguras para cualquier sistema.

Es portable: Además de la portabilidad básica para ser de arquitectura neutral Java también implementa unos estándares de portabilidad, los enteros son siempre enteros, el sistema de interfaces de usuario lo constituye un sistema abstracto de ventanas, por lo que es independiente de la arquitectura en la que se implemente (UNIX, PC, Mac).

Es interpretado: Para poder conseguir uno de los objetivos básicos de Java, que es la independencia de plataforma, el compilador de Java genera un código intermedio (bytecodes). Este código puede ser ejecutado en cualquier sistema

que posea un interprete. Esta característica puede llevarnos a pensar en los posibles problemas de rendimiento. Aunque por este mismo motivo los desarrolladores del lenguaje diseñaron con mucho cuidado el código intermedio de manera que fuese sencillo traducirlo directamente a código máquina nativo para conseguir un rendimiento alto.

Es multihilo: En Java se puede escribir programas que hagan varias cosas a la vez y de una forma fácil y robusta.

Es dinámico : Java no intenta enlazar todos los módulos que componen una aplicación hasta el tiempo de ejecución. Esto permite enlazar dinámicamente el código de una forma segura y conveniente.

3. El entorno de desarrollo

Existen distintos programas comerciales que permiten desarrollar código ***Java***. La Compañía ***Sun***, creadora de ***Java***, distribuye gratuitamente el *Java Development Kit* (***JDK***). Se trata de un conjunto de programas y librerías que permiten desarrollar, compilar y ejecutar programas en ***Java***.

Incorpora además la posibilidad de ejecutar parcialmente el programa, deteniendo la ejecución en el punto deseado y estudiando en cada momento el valor de cada una de las variables (con el denominado ***Debugger***). Cualquier programador con un mínimo de experiencia sabe que una parte muy importante (muchas veces la mayor parte) del tiempo destinado a la elaboración de un programa se destina a la ***detección y corrección de errores***. Existe también una versión reducida del ***JDK***, denominada ***JRE*** (*Java Runtime Environment*) destinada únicamente a ejecutar código ***Java*** (no permite compilar).

Los ***IDE***s (*Integrated Development Environment*), tal y como su nombre indica, son entornos de desarrollo integrados. En un mismo programa es posible escribir el código ***Java***, compilarlo y ejecutarlo sin tener que cambiar de aplicación. Algunos incluyen una herramienta para realizar ***Debug*** gráficamente, frente a la versión que incorpora el ***JDK*** basada en la utilización de una consola (denominada habitualmente ventana de comandos de MS-DOS, en ***Windows NT/95/98***) bastante difícil y pesada de utilizar. Estos entornos integrados permiten desarrollar las aplicaciones de forma mucho más rápida, incorporando en muchos casos librerías con ***componentes*** ya desarrollados, los cuales se incorporan al proyecto o programa. Como inconvenientes se pueden señalar algunos fallos de compatibilidad entre plataformas, y Archivos resultantes de mayor tamaño que los basados en clases estándar.

El compilador de Java

Se trata de una de las herramientas de desarrollo incluidas en el ***JDK***. Realiza un análisis de sintaxis del código escrito en los Archivos fuente de ***Java*** (con extensión ****.java***). Si no encuentra errores en el código genera los Archivos compilados (con extensión ****.class***). En otro caso muestra la línea o líneas erróneas. En el ***JDK*** de ***Sun*** dicho compilador se llama ***javac***. Tiene numerosas opciones, algunas de las cuales varían de una versión a otra. Se aconseja consultar la documentación de la versión del ***JDK*** utilizada para obtener una información detallada de las distintas posibilidades.

La Java Virtual Machina

Tal y como se ha comentado al comienzo del capítulo, la existencia de distintos tipos de procesadores y ordenadores llevó a los ingenieros de ***Sun*** a la conclusión de que era muy importante conseguir un software que no dependiera del tipo de procesador utilizado. Se planteó la necesidad de conseguir un código capaz de ejecutarse en cualquier tipo de máquina. Una vez compilado no debería ser necesaria ninguna modificación por el hecho de cambiar de procesador o de ejecutarlo en otra máquina. La clave consistió en desarrollar un código "neutro" el cual estuviera preparado para ser ejecutado sobre una "*máquina hipotética o virtual*", denominada ***Java Virtual Machina*** (***JVM***). Es esta ***JVM*** quien ***interpreta*** este código neutro convirtiéndolo a código particular de la CPU utilizada. Se evita tener que realizar un programa diferente para cada CPU o plataforma.

La ***JVM*** es el intérprete de ***Java***. Ejecuta los "***bytecodes***" (Archivos compilados con extensión ****.class***) creados por el compilador de ***Java*** (***javac***). Tiene numerosas opciones entre las que destaca la posibilidad de utilizar el denominado ***JIT*** (*Just-In-Time Compiler*), que puede mejorar entre 10 y 20 veces la velocidad de ejecución de un programa.

Las variables PATH y CLASSPATH

El desarrollo y ejecución de aplicaciones en ***Java*** exige que las herramientas para compilar (***javac***) y ejecutar (***java***) se encuentren accesibles. El ordenador, desde una ventana de comandos, sólo es capaz de ejecutar los programas que se encuentran en los directorios indicados en la variable de ambiente ***PATH*** del computador (o en el directorio activo). Si se desea compilar o ejecutar código en ***Java***, el directorio donde se encuentran estos programas (***java*** y ***javac***) deberá encontrarse en el ***PATH***. Tecleando ***PATH*** en una ventana de comandos se muestran los nombres de directorios incluidos en dicha variable de entorno.

Java utiliza además una nueva variable de entorno denominada ***CLASSPATH***, la cual determina dónde buscar tanto las clases o librerías de ***Java*** (el ***API*** de ***Java***) como otras clases de usuario. A partir de la versión 1.1.4 del JDK no es necesario indicar esta variable, salvo que se desee añadir conjuntos de clases de usuario que no vengan con dicho ***JDK***. La variable ***CLASSPATH*** puede incluir la ruta de directorios o Archivos ****.zip*** o ****.jar*** en los que se encuentren los Archivos ****.class***. En el caso de los Archivos ****.zip*** hay que observar que los Archivos en él incluidos no deben estar comprimidos. En el caso de archivos ****.jar*** existe una herramienta (***jar.exe***), incorporada en el ***JDK***, que permite generar estos Archivos a partir de los archivos compilados ****.class***. Los Archivos ****.jar*** son archivos comprimidos y por lo tanto ocupan menos espacio que los archivos ****.class*** por separado o que el Archivo ****.zip*** equivalente.

Una forma general de indicar estas dos variables es crear un archivo ***batch*** (****.bat o sh***) donde se indiquen los valores de dichas variables. Cada vez que se abra una ventana de comando será necesario ejecutar este archivo ****.bat*** para asignar adecuadamente estos valores. Un posible Archivo llamado ***jdk117.bat***, podría ser como sigue:

```
set JAVAPATH=C:\jdk1.1.7
set PATH=.;%JAVAPATH%\bin;%PATH%
set CLASSPATH=.\;%JAVAPATH%\lib\classes.zip;%CLASSPATH%
```

lo cual sería válido en el caso de que el ***JDK*** estuviera situado en el directorio ***C:\jdk1.1.7***. Si no se desea tener que ejecutar este Archivo cada vez que se abre ventana de consola es necesario indicar estos cambios de forma "permanente". La forma de hacerlo difiere ***entre Windows 95/98*** y ***Windows NT***. En ***Windows 95/98*** es necesario modificar el Archivo ***Autoexec.bat*** situado en C:\, añadiendo las líneas antes mencionadas. Una vez rearrancado el ordenador estarán presentes en cualquier consola de MS-DOS que se cree. La modificación al archivo ***Autoexec.bat*** en ***Windows 95/98*** será la siguiente:

```
set JAVAPATH=C:\jdk1.1.7
set PATH=.;%JAVAPATH%\bin;%PATH%
set CLASSPATH=
```

donde en la tercera línea debe incluir la ruta de los archivos donde están las clases de ***Java***. En el caso de utilizar **Windows NT** se añadirá la variable **PATH** en el cuadro de diálogo que se abre con *Start -> Settings -> Control Panel -> System -> Environment -> User Variables for NombreUsuario*. También es posible utilizar la opción ***–classpath*** en el momento de llamar al compilador ***javac*** o al intérprete ***java***. En este caso los Archivos ****.jar*** deben ponerse con el nombre completo en el ***CLASSPATH***: no basta poner el ***PATH*** o directorio en el que se encuentra. Por ejemplo, si se desea compilar y ejecutar el Archivo ***ContieneMain.java***, y éste necesitara la librería de clases ***G:\MyProject\OtherClasses.jar***, además de las incluidas en el ***CLASSPATH***, la forma de compilar y ejecutar sería:

```
javac -classpath .\;G:\MyProject\OtherClasses.jar ContieneMain.java
```

java -classpath .\;G:\MyProject\OtherClasses.jar ContieneMain

Se aconseja consultar la ayuda correspondiente a la versión que se esté utilizando, debido a que existen pequeñas variaciones entre las distintas versiones del JDK. Cuando un archivo ***filename.java*** se compila y en ese directorio existe ya un archivo ***filename.class***, se comparan las fechas de los dos archivos. Si el archivo ***filename.java*** es más antiguo que el ***filename.class*** no se produce un nuevo archivo ***filename.class***. Esto sólo es válido para archivos ****.class*** que se corresponden con una clase ***public***.

4. Nomenclatura

Los nombres de ***Java*** son sensibles a las letras mayúsculas y minúsculas. Así, las variables ***masa***, ***Masa*** y ***MASA*** son consideradas variables completamente diferentes. Las reglas del lenguaje respecto a los nombres de variables son muy amplias y permiten mucha libertad al programador, pero es habitual seguir ciertas normas que facilitan la lectura y el mantenimiento de los programas de ordenador. Se recomienda seguir las siguientes instrucciones:

En ***Java*** es habitual utilizar nombres con minúsculas, con las excepciones que se indican en los puntos siguientes.

Cuando un nombre consta de varias palabras es habitual poner una a continuación de otra, poniendo con mayúscula la primera letra de la palabra que sigue a otra (Ejemplos: *elNuevo()*,*VentanaProglamable*, *RectanguloGrafico*, *addWindowListener()*).

Los nombres de ***clases*** e ***interfaces*** comienzan siempre por mayúscula (Ejemplos: *Geometria*, *Rectangulo*, *Dibujable*, *Graphics*, *ArrayList*, *Iterator*).

Los nombres de ***objetos***, los nombres de ***métodos*** y ***variables miembro***, y los nombres de las ***variables locales*** de los métodos, comienzan siempre por minúscula (Ejemplos: *main()*, *dibujar()*, *numRectangulos*, *x*, *y*, *r*).

Los nombres de las ***variables finales***, es decir de las constantes, se definen siempre con mayúsculas (Ejemplo: *PI*).