ADSI

Introducción a Java - POO

Instructor: Gustavo Adolfo Rodríguez Q.

garodriguez335@misena.edu.co

ADSI

INTRODUCCION A AL LENGUAJE JAVA

En el mundo de la informática existe una gran variedad de lenguajes de programación que se utilizan para dar instrucciones precisas y predecibles a un elemento computacional.

Existen algunos lenguajes de programación que están más fuertemente ligados a un paradigma de programación particular como es el caso del lenguaje C, que es un lenguaje intrínsecamente ligado al paradigma de programación orientada a procedimientos, o el caso del lenguaje Java es un lenguaje totalmente orientado a objetos.

Decir que un lenguaje es orientado a objetos quiere decir que ese lenguaje tiene soporte para los conceptos básicos del paradigma de orientación a objetos tales como clases, objetos, herencia simple y múltiple, polimorfismo, sobrecarga, tipos de acceso, interfaces, etc.

Java es un lenguaje de programación orientado a objetos desarrollado por la multinacional Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de los lenguajes C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Las aplicaciones Java están típicamente compiladas en un bytecode, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el bytecode es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del bytecode por un procesador Java también es posible.

La implementación original y de referencia del compilador, la máquina virtual y las bibliotecas de clases de Java fueron desarrolladas por Sun Microsystems en 1995. Desde entonces, Sun ha controlado las especificaciones, el desarrollo y evolución del lenguaje a través del *Java Community Process* (**JCP**), si bien otros han desarrollado también implementaciones alternativas de estas tecnologías de Sun, algunas incluso bajo licencias de software libre.

Entre noviembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de acuerdo con las especificaciones del *Java Community Process*, de tal forma que prácticamente todo el Java de Sun es ahora software libre (aunque la biblioteca de clases de Sun que se requiere para ejecutar los programas Java todavía no es software libre).

DESCRIPCION GENERAL DEL LENGUAJE JAVA

El lenguaje Java fue creado teniendo como meta principal cumplir con los siguientes objetivos.

- Utilizar el paradigma de programación orientada a objetos.
- Permitir la ejecución del mismo programa en múltiples sistemas operativos.
- Incluir por defecto soporte para el trabajo en red.
- Permitir la ejecución de código en máquinas remotas de forma segura.

- Fácil de utilizar conservando los mejores aspectos de lenguajes como C y C++

Cada uno de estos objetivos fue logrado con el lenguaje Java y son estos objetivos de diseño los que se han convertidos en las características más importantes del lenguaje Java convirtiéndolo en uno de los más utilizados en el mundo.

CARACTERISTICAS DEL LENGUAJE JAVA

1. ORIENTACION A OBJETOS

Esta característica hace que el lenguaje tenga soporte nativo para el paradigma de programación orientada a objetos, es decir, el lenguaje Java permite la definición de clases, permite la creación de objetos instancia de esa clase, permite el intercambio de mensajes entre objetos respetando los tipos de acceso, permite la herencia, permite el polimorfismo, permite el encapsulamiento y todos los otros conceptos relacionados con el paradigma.

La meta buscada con orientación a objetos del lenguaje Java es hacer que los grandes proyectos de software sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos. Otra de las grandes promesas de la programación orientada a objetos es la creación de entidades más genéricas (objetos) que permitan la reutilización del software entre proyectos, una de las premisas fundamentales de la Ingeniería del Software. Un objeto genérico “cliente”, por ejemplo, debería en teoría tener el mismo conjunto de comportamiento en diferentes proyectos, sobre todo cuando estos coinciden en cierta medida, algo que suele suceder en las grandes organizaciones. En este sentido, los objetos podrían verse como piezas reutilizables que pueden emplearse en múltiples proyectos distintos, posibilitando así a la industria del software a construir proyectos de envergadura empleando componentes ya existentes y de comprobada calidad; conduciendo esto finalmente a una reducción drástica del tiempo de desarrollo. Podemos usar como ejemplo de objeto el aluminio. Una vez definidos datos (peso, maleabilidad, etc.), y su “comportamiento” (soldar dos piezas, etc.), el objeto “aluminio” puede ser reutilizado en el campo de la construcción, del automóvil, de la aviación, etc.

La reutilización del software ha experimentado resultados dispares, encontrando dos dificultades principales: el diseño de objetos realmente genéricos es pobremente comprendido, y falta una metodología para la amplia comunicación de oportunidades de reutilización. Algunas comunidades de “código abierto” (open source) quieren ayudar en este problema dando medios a los desarrolladores para diseminar la información sobre el uso y versatilidad de objetos reutilizables y bibliotecas de objetos.

2. INDEPENDENCIA DE LA PLATAFORMA

La segunda característica, la independencia de la plataforma, significa que los programas escritos en el lenguaje Java pueden ejecutarse igualmente en cualquier tipo de hardware. Este es el significado de ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo, tal como reza el axioma de Java, ‘’’write once, run everywhere’’’.

Para ello, se compila el código fuente escrito en lenguaje Java, para generar un código conocido como “bytecode” (específicamente Java bytecode)—instrucciones máquina simplificadas específicas de la plataforma Java. Esta pieza está “a medio camino” entre el código fuente y el código máquina que entiende el dispositivo destino. El bytecode es ejecutado entonces en la máquina virtual (JVM), un programa escrito en código nativo de la plataforma destino (que es el que entiende su hardware), que interpreta y ejecuta el código. Además, se suministran bibliotecas adicionales para acceder a las características de cada dispositivo (como los gráficos, ejecución mediante hilos o threads, la interfaz de red) de forma unificada.

Hay implementaciones del compilador de Java que convierten el código fuente directamente en código objeto nativo, como **GCJ**. Esto elimina la etapa intermedia donde se genera el bytecode, pero la salida de este tipo de compiladores sólo puede ejecutarse en un tipo de arquitectura específica.

La licencia sobre Java de Sun insiste que todas las implementaciones sean “compatibles”. Esto dio lugar a una disputa legal entre Microsoft y Sun, cuando éste último alegó que la implementación de Microsoft no daba soporte a las interfaces RMI y JNI además de haber añadido características ‘’dependientes’’ de su plataforma. Sun demandó a Microsoft y ganó por daños y perjuicios (unos 20 millones de dólares) así como una orden judicial forzando la acatación de la licencia de Sun. Como respuesta, Microsoft no ofrece Java con su versión de sistema operativo, y en recientes versiones de Windows, su navegador Internet Explorer no admite la ejecución de applets sin un conector (o plugin) aparte. Sin embargo, Sun y otras fuentes ofrecen versiones gratuitas para distintas versiones de Windows.

Las primeras implementaciones del lenguaje usaban una máquina virtual interpretada para conseguir la portabilidad. Sin embargo, el resultado eran programas que se ejecutaban comparativamente más lentos que aquellos escritos en C o C++. Esto hizo que Java se ganase una reputación de lento en rendimiento. Las implementaciones recientes de la JVM dan lugar a programas que se ejecutan considerablemente más rápido que las versiones antiguas, empleando diversas técnicas, aunque sigue siendo mucho más lento que otros lenguajes.

La primera de estas técnicas es simplemente compilar directamente en código nativo como hacen los compiladores tradicionales, eliminando la etapa del bytecode. Esto da lugar a un gran rendimiento en la ejecución, pero tapa el camino a la portabilidad. Otra técnica, conocida como compilación JIT (Just In Time, o ‘’’compilación al vuelo’’’), convierte el bytecode a código nativo cuando se ejecuta la aplicación. Otras máquinas virtuales más sofisticadas usan una ‘’’recompilación dinámica’’’ en la que la VM es capaz de analizar el comportamiento del programa en ejecución y recompila y optimiza las partes críticas. La recompilación dinámica

puede lograr mayor grado de optimización que la compilación tradicional (o estática), ya que puede basar su trabajo en el conocimiento que de primera mano tiene sobre el entorno de ejecución y el conjunto de clases cargadas en memoria. La compilación JIT y la recompilación dinámica permiten a los programas Java aprovechar la velocidad de ejecución del código nativo sin por ello perder la ventaja de la portabilidad en ambos.

La portabilidad es técnicamente difícil de lograr, y el éxito de Java en ese campo ha sido dispar. Aunque es de hecho posible escribir programas para la plataforma Java que actúen de forma correcta en múltiples plataformas de distinta arquitectura.

El concepto de independencia de la plataforma de Java cuenta, sin embargo, con un gran éxito en las aplicaciones en el entorno del servidor, como los Servicios Web, los Servlets, los Java Beans, así como en sistemas desarrollados para dispositivos móviles.

3. SOPORTE DE RED Y PROCEDIMIENTOS REMOTOS

La tercera característica del lenguaje Java permite la posibilidad de creación de programas software que realizan llamados a procedimientos remotos. La llamada a procedimientos remotos mediante RMI (Remote Method Invocation) ha sido la base para la computación distribuida. La computación distribuida permite que los sistema de información puedan ser divididos en secciones o subsistemas donde cada subsistema se ejecuta en una máquina o conjunto de máquinas y la interacción de los subsistemas crea un sistema más grande y complejo pero totalmente funcional. Gracias al éxito de algunos protocolo para el trabajo orientado a objetos y computación distribuida como IIOP, y a la gran aceptación de RMI, es posible trabajar RMI-IIOP para crear sistemas distribuidos que hacen uso de la red para el intercambio de datos y la llamada a procedimientos remotos. RMI-IIOP es la base para la invocación a Servicios Web en Java.

4. RECOLECTOR DE BASURA

Un argumento en contra de lenguajes como **C++** es que los programadores tienen la responsabilidad de administrar la memoria solicitada dinámicamente de forma manual.

En C++, el desarrollador puede asignar memoria en una zona conocida como *heap* para crear cualquier objeto, y posteriormente desalojar el espacio asignado cuando desea borrarlo. Un olvido a la hora de desalojar memoria previamente solicitada puede llevar a una *fuga de memoria*, ya que el sistema operativo seguirá pensando que esa zona de memoria está siendo usada por una aplicación cuando en realidad no es así. Así, un programa mal diseñado podría consumir una cantidad desproporcionada de memoria. Además, si una misma región de memoria es desalojada dos veces el programa puede volverse inestable y llevar a un eventual *bloqueo*.

En Java, este problema potencial es evitado en gran medida por el recolector automático de basura (o *automatic garbage collector*). El programador determina cuándo se crean los objetos y el entorno en tiempo de ejecución de Java (Java runtime) es el responsable de gestionar el ciclo de vida de los objetos. El programa, u otros objetos pueden tener localizado un objeto mediante una referencia a éste (que, desde un punto de vista de bajo nivel es una dirección de memoria). Cuando no quedan referencias a un objeto, el recolector de basura de Java borra el objeto, liberando así la memoria que ocupaba previniendo posibles fugas (ejemplo: un objeto creado y únicamente usado dentro de un método sólo tiene entidad dentro de éste; al salir del método el objeto es eliminado). Aun así, es posible que se produzcan fugas de memoria si el código almacena referencias a objetos que ya no son necesarios—es decir, pueden aún ocurrir, pero en un nivel conceptual superior. En definitiva, el recolector de basura de Java permite una fácil creación y eliminación de objetos, mayor seguridad y puede que más rápida que en C++.

La recolección de basura de Java es un proceso prácticamente invisible al desarrollador. Es decir, el programador no tiene conciencia de cuándo la recolección de basura tendrá lugar, ya que ésta no tiene necesariamente que guardar relación con las acciones que realiza el código fuente.

Debe tenerse en cuenta que la memoria es sólo uno de los muchos recursos que deben ser gestionados

ENTORNOS DE FUNCIONAMIENTO

El diseño de Java, su robustez, el respaldo de la industria y su fácil portabilidad han hecho de Java uno de los lenguajes con un mayor crecimiento y amplitud de uso en distintos ámbitos de la industria de la informática. Estos son los escenarios en los cuales los programas desarrollados con el lenguaje Java pueden ejecutarse:

1. TARJETAS INTELINGENTES

Es posible encontrar dispositivos muy pequeños con un hardware de características muy discretas capaces de ejecutar bytecodes de Java como es el caso de los decodificadores de televisión digital, buscapersonas, e incluso tarjetas inteligentes (Tarjetas Bancarias o SIM Cards).

La especificación de Java para este tipo de dispositivos se denomina JavaCard . JavaCard es un conjunto de herramientas y librerías de programación Java que permiten la creación de aplicaciones utilizando el lenguaje Java y la orientación a objetos en estos ambientes tan especiales.

2. DISPOSITIVOS MOVILES

En el mundo tecnológico existen una gran variedad de dispositivos móviles, estos dispositivos móviles pueden ser clasificados en las siguientes categorías:

- Dispositovos de entretenimiento
- Asistentes Digitales Personales (PDA PALM, PDA POCKET PC)
- Teléfonos celulares
- Teléfonos Inteligentes.
- Dispositivos de entretenimiento.

Si pensamos en la cantidad y gran variedad de dispositivos móviles que se encuentran en el mercado donde cada uno de ellos tiene características propias y muy dispares vemos que no existe una plataforma común para la ejecución de programas software. Este hecho ha impulsado la amplia utilización de Java en estos dispositivos por sus características de independencia de la plataforma

La especificación de Java que permite el desarrollo de aplicaciones sobre dispositivos móviles se llama Java Micro Edition (JavaME)

La utilización más popular de la plataforma Java en dispositivos móviles es para juegos. Las aplicaciones desarrolladas con JavaME para teléfonos celulares se llaman MIDlets.

3. NAVEGADORES WEB

El lenguaje Java gracias a su soporte nativo para trabajar en red permite la creación de navegadores Web. Los navegadores Web que han sido escritos utilizando el lenguaje Java se apoyan en motores de *renderizado* para poder desplegar adecuadamente las páginas visitadas.

Existen también algunos proyectos de software libre desarrollados con el lenguaje Java orientados al intercambio de archivos en rede *peer to peer*.

Sin embargo, aun cuando ya es una tecnología en desuso, los applets han sido principalmente aplicaciones que se ejecutan embebidas en el navegador web.

4. SISTEMAS DE ESCRITORIO

En los ambientes de escritorio existen una variedad de sistemas operativos considerable, a pesar de que el mercado de los sistemas operativos para equipos de escritorio está dominado por Microsoft y sus sistemas Windows, existen también otros sistemas operativos de amplia utilización tales como MAC OS, y las múltiples distribuciones de Linux.

Puesto que las aplicaciones desarrolladas con el lenguaje Java gozan de la independencia de la plataforma, es posible utiliza las aplicaciones desarrolladas indistintamente en un sistema Windows como en un sistema MAC.

La especificación de Java para el desarrollo de aplicaciones sobre sistemas de escritorio se denomina Java SE (Java Standard Edition).

Hoy en día existen multitud de aplicaciones gráficas de usuario basadas en Java. El entorno de ejecución Java (JRE) se ha convertido en un componente habitual en los PC de usuario de los

sistemas operativos más usados en el mundo. Además, muchas aplicaciones Java lo incluyen dentro del propio paquete de la aplicación de modo que se ejecuten en cualquier PC.

En las primeras versiones de la plataforma Java existían importantes limitaciones en las APIs de desarrollo gráfico (AWT). Desde la aparición de la biblioteca Swing la situación mejoró substancialmente y posteriormente con la aparición de bibliotecas como SWT hacen que el desarrollo de aplicaciones de escritorio complejas y con gran dinamismo, usabilidad, etc. sea relativamente sencillo.

5. SISTEMAS SERVIDORES

En la parte del servidor, Java es más popular que nunca, desde la aparición de la especificación de Servlets y JSP (Java Server Pages).

Los **servlets**, son objetos que corren dentro del contexto de un contenedor en un servidor de aplicaciones

Hasta entonces, las aplicaciones web dinámicas de servidor que existían se basaban fundamentalmente en componentes CGI y lenguajes interpretados. Ambos tenían diversos inconvenientes (fundamentalmente lentitud, elevada carga computacional o de memoria y propensión a errores por su interpretación dinámica).

Los servlets y las JSPs supusieron un importante avance ya que:

- El API de programación es muy sencilla, flexible y extensible.
- Los servlets no son procesos independientes (como los CGIs) y por tanto se ejecutan dentro del mismo proceso que la JVM mejorando notablemente el rendimiento y reduciendo la carga computacional y de memoria requeridas.
- Las JSPs son páginas que se compilan dinámicamente (o se pre-compilan previamente a su distribución) de modo que el código que se consigue una ventaja en rendimiento substancial frente a muchos lenguajes interpretados.

La especificación de Servlets y JSPs define un API de programación y los requisitos para un contenedor (servidor de aplicaciones) dentro del cual se puedan desplegar estos componentes para formar aplicaciones web dinámicas completas. Hoy día existen multitud de contenedores (libres y comerciales) compatibles con estas especificaciones.

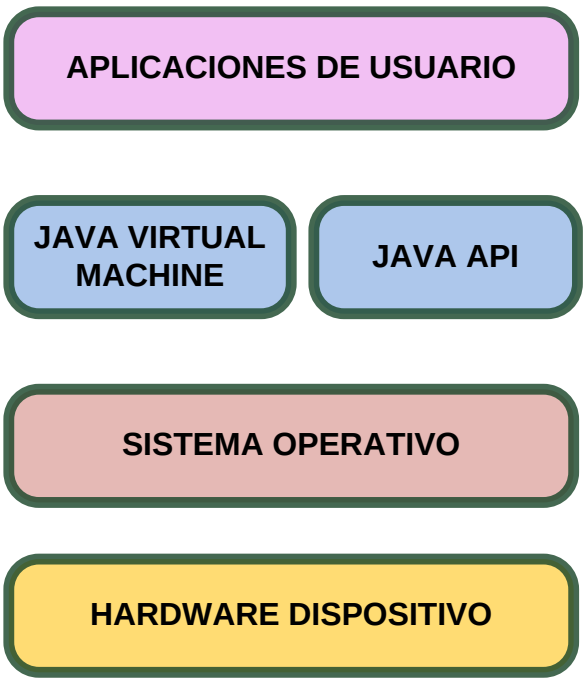
A partir de su expansión entre la comunidad de desarrolladores, estas tecnologías han dado paso a modelos de desarrollo mucho más elaborados con frameworks como Struts y Webworks que se sobreponen sobre los Servlets y las JSPs para conseguir un entorno de trabajo mucho más poderoso y segmentado en el que es posible la especialización de roles (desarrolladores, diseñadores gráficos, ...). A pesar de todo ello, las tecnologías que subyacen (Servlets y JSPs) son substancialmente las mismas.

Este modelo de trabajo se ha convertido en un estándar de-facto para el desarrollo de aplicaciones web dinámicas de servidor y otras tecnologías como ASP se han basado en él.

Actualmente, se ha unificado y estandarizado un framework para el desarrollo de aplicación con Java denominado Java Server Faces (JSF)

ARQUITECTURA JAVA

La arquitectura de Java puede ser vista como un modelo en capas como se muestra en la siguiente figura:



De abajo hacia arriba los componentes de esta arquitectura son los siguientes:

- **Hardware del Dispositivo:** Este componente de la arquitectura representa todos los elementos hardware del dispositivo en el cual se ejecuta la aplicación tales como RAM, ROM, Disco Duro, Dispositiovos E/S.

- **Sistema Operativo:** Este componente es el sistema operativo encargado de manejar el hardware subyacente. Dicho sistema operativo puede ser Windows, Linux, MAC u otro.

- **Java Virtual Machine:** Este componente de la arquitectura es el que garantiza la portabilidad de las aplicaciones Java. Dicho componente se encarga de interpretar los bytecodes de la aplicación y traducirlos a instrucciones entendibles por el sistema operativo y el hardware subyacente. Existe un tipo diferente de Java Virtual Machine para cada sistema operativo.

- **Java API:** Este componente de la arquitectura representa a todas las librerías de clases disponibles para la programación y ejecución de las aplicaciones de usuario. En estas librerías se encuentran todas las clases que hacen parte de cada una de las versiones de Java (JavaCard, JavaME, JavaSE o JavaEE)

- **Aplicaciones de Usuario:** Las aplicaciones de usuario son todas aquellas aplicaciones que se han construido y utilizando las herramientas y librerías de Java. Dichas aplicaciones pueden ser: aplicaciones de escritorio, aplicaciones con interfaz gráfica, aplicaciones de consola de comando, etc.