

## Notas técnicas de JAVA Nro. 5 – Tip en detalle

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

### Organización de memoria en JAVA Vs. Modelo Tradicional

<b>Tema:</b>	Java, JVM, objetos,
<b>Descripción:</b>	Explica la organización de memoria y asignación de objetos en la plataforma Java confrontándola con el esquema de administración de memoria de otros lenguajes más tradicionales como el C.
<b>Nivel:</b>	Avanzado
<b>Fecha pub:</b>	Diciembre 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](mailto: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
5. Organización de memoria en JAVA Vs. Modelo Tradicional

#### Próximos Tips:

Nivel Técnico avanzado

- Secretos del manejo de Fechas en Java
- 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

- I. Descripción del modelo tradicional de los programas en memoria
- II. Organización de la memoria en la Java Virtual Machine (JVM).
- III. Organización de la memoria con el objeto en ejecución.
- IV. Dónde obtener información adicional

---

### I. Introducción

El *objeto* es la unidad lógica de Java mediante la cual se representan conceptos puntuales en una aplicación. No obstante, físicamente un *objeto* particular en la memoria se constituye mediante distintas subdivisiones, acorde con el diseño de administración de memoria que hace la Java Virtual Machine.

Es interesante conocer las generalidades de la administración de memoria de una Java Virtual Machine, y cómo se reflejan en memoria las características de código JAVA. Esto permite también ganar comprensión sobre lo que implica el paradigma de objetos.

Para una mejor comprensión de la administración de la memoria que hace JAVA para la utilización de objetos, es interesante primero contrastarlo con el modelo utilizado por la mayoría de los lenguajes de programación convencionales (no orientados a objetos).

---

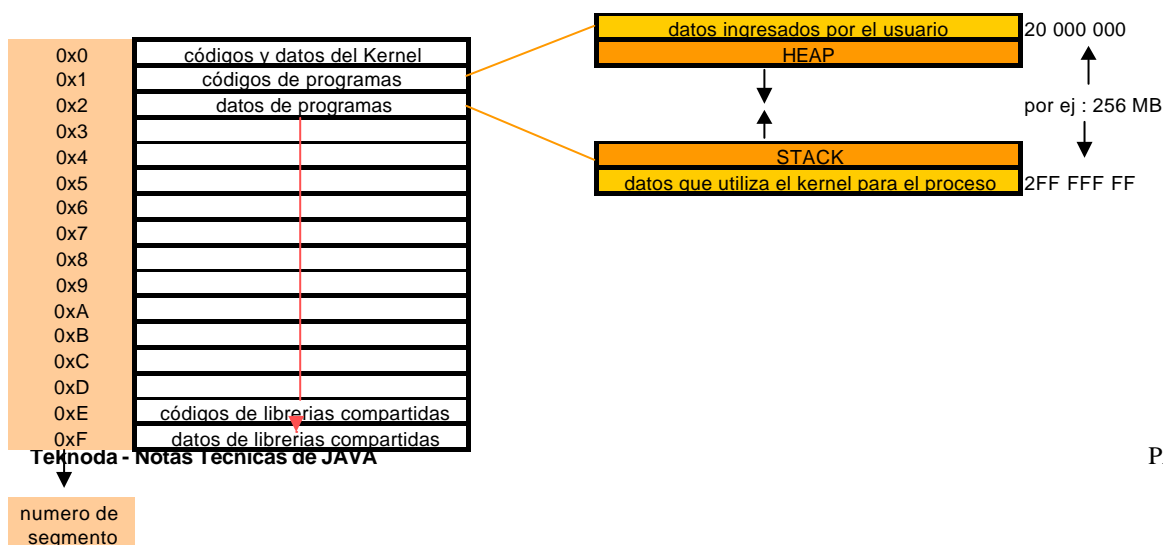
### II. Descripción del modelo tradicional de los programas en memoria

En los entornos tradicionales, durante la ejecución de una aplicación en memoria, ésta comporta una división en dos grandes partes:

- Una parte para *el código* propiamente dicho
- Una parte para *los datos* sobre los cuales ese código iba a operar.

De una forma simple, podemos decir que una parte del área de datos se alojaba en una memoria tipo “Stack” (pila estática) y otra en la memoria llamada “Heap” (pila dinámica). La memoria “Stack” se utilizaba para mantener todos aquellos datos para los cuales su estructura se mantendría inmutable durante toda la ejecución de la aplicación. Por ejemplo, una variable de tipo entera en C siempre sería de tipo entera, y un puntero o referencia, también sería un puntero o referencia para toda la ejecución de la aplicación.

Luego el “Heap” contenía todos aquellos datos cuya existencia dependiera del desarrollo de la ejecución de la aplicación. Esa condición hace que el enfoque de administración mediante Stack no sea práctica. El Heap es una región de memoria que permite un acceso aleatorio más transparente, mientras que una memoria de tipo Stack impone lógicamente un acceso ordenado.



---

### III. Organización de la Memoria en la Java Virtual Machine (JVM) .

En principio, Java tiene varias similitudes con el modelo tradicional de ejecución de programas. En la JVM existen principalmente dos tipos de variables:

- las primitivas (int, long, float, etc),
- las de referencia a objetos

Las primeras tienen una estructura fija y su contenido representa el dato en sí. Tanto la condición de estructuras de tamaño fijo como el de su tamaño relativamente pequeño hacen que sea práctica una administración de las mismas mediante el mismo mecanismo con memoria “Stack”.

También las variables de referencia tienen una estructura fija, pero su contenido no es un dato en sí, sino una dirección de memoria donde se encuentra la instancia del objeto que representa. Esto hace que sea posible también que las variables de referencia se administren mediante un “Stack”.

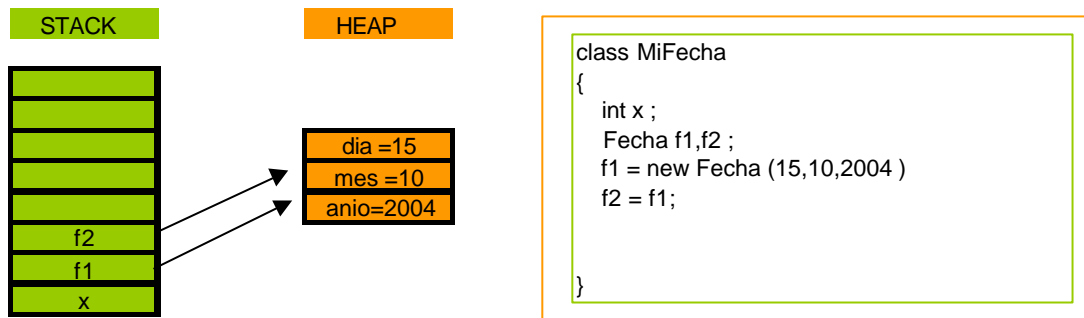
Luego, éstas contienen la dirección de memoria de una estructura mucho más compleja, una instancia de una Clase (a la cual llamaremos *objeto*). El objeto no solamente representa un dato, sino que representa un conjunto de datos asociados a un concepto que incluye operaciones capaces de actuar sobre estos mismos datos. Debido a la cantidad de información que mantienen y la complejidad que requiere su administración, los objetos se guardan en el “Heap”. Los objetos pueden contener múltiples valores, y la referencia al objeto debe mantenerse durante toda la vida de ese objeto.

Mientras exista una referencia a un objeto, ese objeto será considerado válido. La gran ventaja de Java respecto de la administración de memoria frente a muchos otros lenguajes es que en caso de no requerir más un objeto, al dejar de referenciarlo un mecanismo automático se encargara en un momento indeterminado de limpiar ese espacio de memoria y dejarlo libre para volver a ser utilizado, muy probablemente por otro objeto. Veamos un ejemplo práctico :

- El siguiente código se utiliza para la creación de la clase Fecha

```
class Fecha
{
    private int dia ;
    private int mes;
    private int anio;
}
```

- Luego creamos 2 objetos de la clase de la clase Fecha ( f1 y f2 ) y decidimos asignarles los mismos datos a f1 y f2 . Por lo tanto ,se realizan 2 referencias a la misma posición de memoria



El proceso de recuperar la memoria del “Heap” que ya no esta siendo utilizada por algún objeto referenciado desde las variables del “Stack” se denomina **Garbage Collector**. El mismo se refiere a un algoritmo que, aunque implementado distinto, cumple la función de establecer que objetos ya no son utilizables (han perdido toda referencia a ellos) y por lo tanto, puede reutilizarse la memoria que ocupan. Este algoritmo, no solamente tendrá algunas diferencias entre distintas JVMs, sino que es posible encontrar distintos algoritmos para distintas situaciones en una misma JVM.

En términos generales, el algoritmo podría hacer lo siguiente:

1. Recorrer todos los objetos existentes y asignarles una marca.
2. Recorrer todas las referencias a objetos y retirar la marca de los objetos referenciados.
3. Liberar el espacio para todos los objetos que aún mantienen la marca.

El “Garbage Collector” es el aspecto visible de la administración de memoria en Java

La localización de una clase y carga en memoria es el paso previo a la creación misma de una instancia particular. Este proceso se divide en:

- Carga
- Enlace
- Inicialización

La Carga (Loading) de una clase implica localizar su código binario, por ejemplo buscando en las distintas entradas en un File System según se determine mediante el CLASSPATH. Ese código binario, denominado bytecodes, se carga en la JVM para su posterior reorganización y utilización. Esa “reorganización” lleva el nombre de Enlace (Linking) y relaciona distintas entidades de código binario entre sí para permitir la ejecución de clases y objetos relacionados. El último paso es la Inicialización, que se lleva adelante ejecutando todos los inicializadores de las variables miembro estáticas.

En esta instancia del proceso, la clase cargada ocupa un lugar en la memoria y se encuentra potencialmente en condiciones de permitir la creación de instancias a partir de sí misma.

#### IV. Organización de la memoria con el objeto en ejecución.

Una opción de implementación particular de la administración de memoria para los objetos consiste en hacer que la referencia a un objeto tenga la dirección de una estructura con dos punteros que permiten llegar a dos

nuevas estructuras: primero, una tabla con información acerca de la clase particular a la cual pertenece el objeto, y luego, un puntero a la memoria asignada en el “Heap” para que el objeto pueda utilizar.

La primera estructura se conoce con el nombre de “Constant Pool”, y pertenece a la representación de la clase en memoria. Esta contiene todo aquello que sea compartido entre los distintos objetos del mismo tipo: constantes, referencias a métodos y variables, etc. Respecto de los lenguajes tradicionales se dice que esta estructura es comparable a la tabla de símbolos que se crea para cada aplicación”, siendo en este caso de mayor diversidad de información.

Cuando se inicia la ejecución de una aplicación Java, se utiliza al menos un “thread”. Ese “thread” es el flujo de ejecución de la aplicación y se crea utilizando la representación de la clase que dio origen al inicio de la aplicación. Ante la llamada a cada método del objeto “en ejecución” en el “thread” se asigna un espacio de memoria denominado “Frame”, que no es mas que una porción de memoria que implementa dentro de sí misma, la memoria tipo “Stack” que utilizan los objetos Java, siendo que en la práctica, cada método tiene un “Frame”, y por lo tanto no hay un único “Stack” por aplicación, sino que hay un “Frame” con un “Stack” por método en ejecución por cada thread en memoria.

En cada Frame existe un “Operand Stack” ó “Stack de Operandos” en el cual se guardan los resultados y los datos necesarios para llevar adelante las distintas operaciones cuando no son triviales, de la misma forma que se mantiene información sobre variables locales al método y referencias al Constant Pool.

Cuando un método termina, este área de memoria se libera (lo que significa que se liberan variables locales con referencias a *objetos* que existen en el Heap), dejando a estos a la espera del Garbage Collection si es que no hay referencias a esos mismos *objetos* en otros métodos.

Pero, en una situación en la cual “un método termina a otro”, se crea un nuevo “Frame” y se comienza a operar dentro de éste. Al momento de finalizar, ese “Frame” se destruye y se devuelve el control al “Frame del método llamador”.

---

## V. Dónde Obtener Información Adicional

<http://www.particle.kth.se/~lindsey/JavaCourse/Book/Supplements/Chapter01/JVM.html>

<http://www.particle.kth.se/~lindsey/JavaCourse/Book/Supplements/Chapter04/classLoaders.html>

<http://www.developer.com/java/other/article.php/2248831>

<http://java.sun.com/docs/hotspot/gc1.4.2/index.html>

<http://java.sun.com/docs/hotspot/gc1.4.2/faq.html>

<http://www.developer.com/java/other/article.php/2248831>

*Copyright Diciembre 2004 Teknoda S.A. JAVA es marca registrada de Sun. SAP, R/3 y ABAP son marcas registradas de SAP AG. AS/400 es marca registrada de IBM. Todas las marcas mencionadas son marcas registradas de las empresas proveedoras. La información contenida en este artículo ha sido recolectada en la tarea cotidiana por nuestros especialistas a partir de fuentes consideradas confiables. No obstante, por la posibilidad de error humano, mecánico, cambios de versión u otro, Teknoda no garantiza la exactitud o completud de la información aquí volcada.*

*Dudas o consultas [develop@teknoda.com](mailto:develop@teknoda.com)*