Áreas de Memoria

Los datos de los programas en Java se almacenan en uno de las tres áreas de memoria que el programador tiene a su disposición:

1. La zona de **memoria estática** es para datos que no cambian de tamaño, permite almacenar variables globales que son persistentes durante la ejecución de un programa.
2. El **heap** permite almacenar variables adquiridas dinámicamente (al crear objetos con la palabra **new)** durante la ejecución de un programa.
3. El **stack** permite almacenar argumentos y variables locales durante la ejecución de las funciones o métodos en las que están definidas.

El compilador asigna un espacio determinado para las variables y genera las referencias para acceder a las variables del stack y de la zona estática. El tamaño de las variables del stack y de la zona estática no puede cambiarse durante la ejecución del programa, es asignado en forma estática.

Cuando se carga un programa a ejecutar en memoria desde el disco duro solamente los datos de la zona estática son creados. El **heap y el stack son creados dinámicamente** durante la ejecución del programa.

Memoria estática

La zona estática de memoria permite almacenar variables globales y de tamaño estático. Si se encuentra una variable definida afuera de las funciones o métodos (incluyendo main) en un archivo fuente de código (de tipo .java), se la considera global a la clase, el compilador les asigna un espacio determinado y genera las referencias para accesarlas en la zona estática.

El tamaño de las variables no puede cambiarse durante la ejecución del programa, es asignado en forma estática.

El tiempo de vida de las variables de la zona estática es la duración del programa. Estas variables son visibles para todas las funciones o métodos que estén definidas después de ellas en el archivo en que se definan.

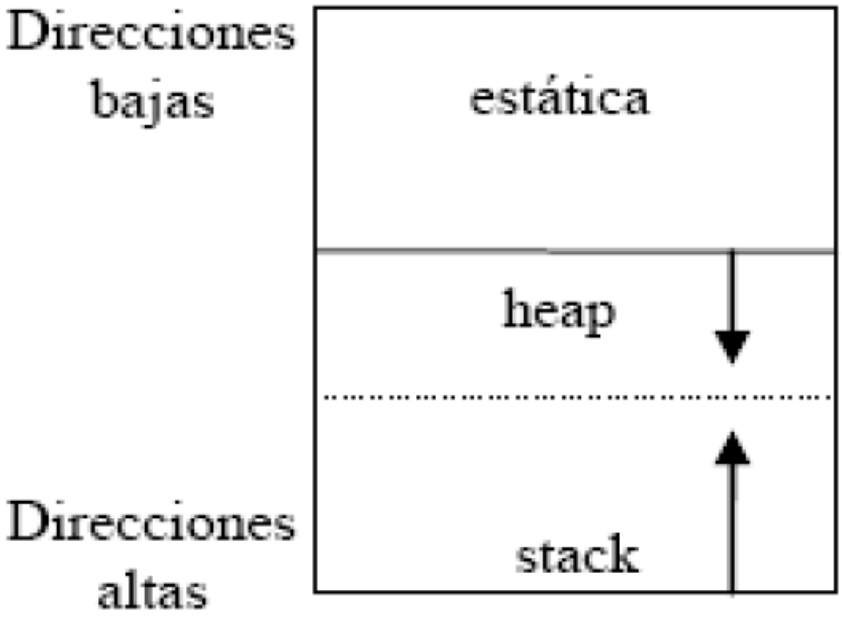
Heap o Monticulo

El heap es un area de memoria dinámica que el programa pide al sistema operativo utilizando la creación de objetos a través de la palabra **new**. Esta memoria se maneja vía punteros o referencias y es la responsabilidad del mismo proceso el liberar la memoria (recolector de basura) después de su uso. En java el recolector de basura (garbage collector) e s el encargado de liberar la memoria de los objetos que ya no están siendo utilizados.

Stack

El stack se utiliza para almacenar las variables denominadas automáticas, ellas existen durante la ejecución del método que las referencia. Los argumentos y variables locales, son asignados y desasignados en forma dinámica durante la ejecución de los método; pero en forma automática por código generado por el compilador, el programador no tiene responsabilidad en ese proceso. Esta organización permite direccionar eficientemente variables que serán usadas frecuentemente; a la vez posibilita ahorrar espacio de direccionamiento ya que se puede reutilizar el espacio de memoria

dedicado al método cuando éste termina; y también posibilita el diseño de métodos recursivos y reentrantes, asociando un espacio diferente para las variables por cada invocación del método. Es importante notar que varios métodos pueden emplear los mismos nombres para las variables locales y argumentos y esto no provoca confusión; existe independencia temporal de las variables de un método.



Si se emplea una global, con igual nombre que una local, dentro de un método se ve la local; y fuera existe la global. No pueden comunicarse los valores de variables locales de un método a otro. Obviamente como convención para tener mejor legibilidad es mejor no usar los mismos nombres a las variables globales y locales.

Cada función al ser invocada crea un frame en el stack o registro de activación, en el cual se almacenan los valores de los argumentos y de las variables locales.

Los valores de los argumentos son escritos en memoria, antes de dar inicio al código asociado al método. Es responsabilidad del método escribir valores iniciales a las variables locales, antes de que éstas sean utilizadas; es decir, que aparezcan en expresiones para lectura.

Diferencia entre Stack y Heap

Stack	Heap
Normalmente tiene un espacio limitado y fijo al iniciar la ejecución del programa (estática).	Puede crecer a medida que se requiere más espacio (dinámica).
La alocación se realiza más rápidamente.	Alocación más lenta que en la stack.
Los datos almacenados pueden usarse sin punteros o con punteros.	Los datos sólo se acceden mediante punteros.
Las variables se desalojan automáticamente.	Manejada manualmente por el programador (new / delete).
Almacenamiento contiguo	Almacenamiento no contiguo (puede producirse fragmentación)

Diferencias entre memoria estática y dinámica

Estática:

- Durante la ejecución del programa el tamaño de la estructura no cambia.
- Se define el tamaño en tiempo de compilación.

Dinámica

- Durante la ejecución del programa el tamaño de la estructura puede cambiar.
- Se define el tamaño en tiempo de ejecución.

Asignación de memoria estática

Consiste en el proceso de asignar memoria en tiempo de compilación antes de que el programa asociado sea ejecutado, a diferencia de la asignación dinámica o la automática donde la memoria se asigna a medida que se necesita en tiempo de ejecución.

Un módulo de programa (por ejemplo, función o método) declara datos estáticos de forma local, de forma que estos datos son inaccesibles desde otros módulos a menos que se les pasen referenciados como parámetros o que les sean devueltos por la función.

Se mantiene una copia simple de los datos estáticos, accesible a través de llamadas a la función en la cual han sido declarados.

El uso de variables estáticas dentro de una clase en la programación orientada a objetos permite que una copia individual de tales datos se comparta entre todos los objetos de esa clase.

Las constantes conocidas en tiempo de compilación, como literales de tipo cadena, se asignan normalmente de forma estática.

En programación orientada a objetos, el método usual para las tablas de clases también es la asignación estática de memoria.

Asignación de memoria dinámica

Es la asignación de almacenamiento de memoria para utilización por parte de un programa de computador durante el tiempo de ejecución de ese programa.

Es una manera de distribuir la propiedad de recursos de memoria limitada entre muchas piezas de código y datos.

Un objeto asignado dinámicamente permanece asignado hasta que es desasignado explícitamente, o por el programador o por un recolector de basura; esto es notablemente diferente de la asignación automática de memoria y de la asignación estática de memoria (la de las variables estáticas). Se dice que tal objeto tiene tiempo de vida dinámico.