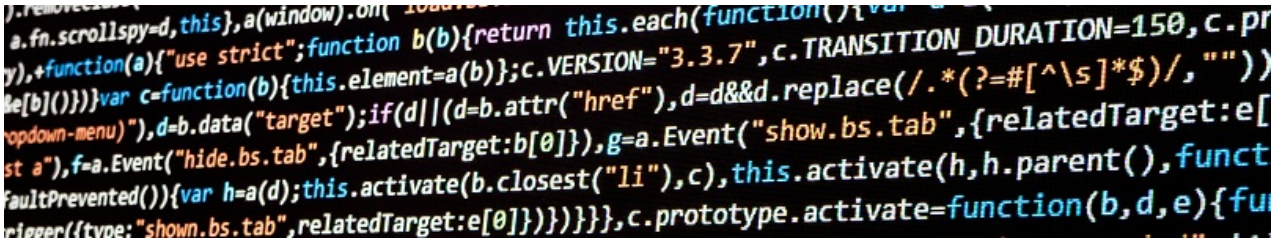
El blog informático de Edgar Jayo

Soy Edgar Jayo y quiero compartir conocimientos de Informática de diversos temas, con pequeños tips que en algún momento me sacaron de apuros para agilizar una tarea. También incluyo una sección Miscelánea donde publicaré temas variados relacionados con la informática.

PROGRAMACIÓN

Explicando la memoria Stack y Heap de Java (1/3)



Fecha: 30 octubre, 2020 Autor/a: ejayo 0 Comentarios
Comparto esta información que me parece importante, visto en un curso en la plataforma Udemy, al final dejo la fuente por si les interesa el curso, el cual me pareció muy completo.

Los programadores ya saben que un programa usa la memoria RAM para almacenar la información (del programa) mientras se está ejecutando. Al interior de Java existen dos clasificaciones para almacenar los valores del programa, estas son conocidas como **memoria Stack** y **memoria Heap**.

En este post (perdón por ser algo extenso) trataré de explicar de la mejor forma que son estas dos memorias y de paso hacer comparaciones con .Net.

¿Para qué se usan cada una?

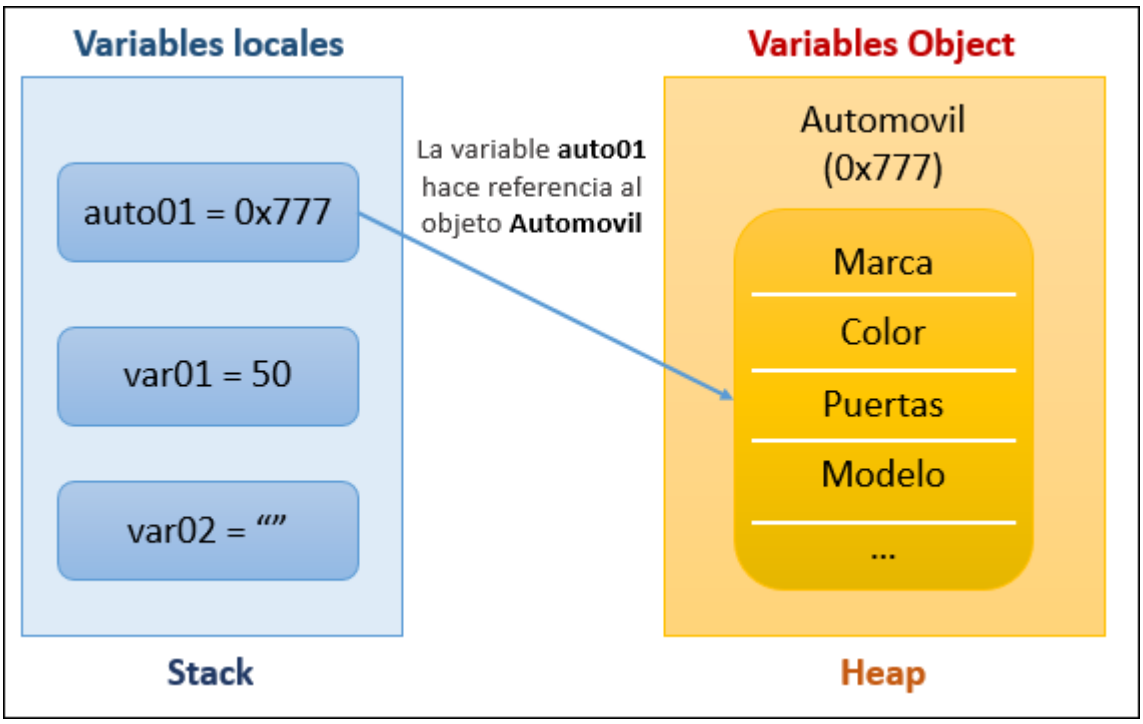
La **memoria Stack** se usa para almacenar las variables locales (cuyo ámbito de acción está limitada solo a la función donde se declaró) y también las llamadas de funciones en Java. Las variables almacenadas en esta memoria por lo general tienen un periodo de vida corto, viven hasta que terminen la función o método en el que se están ejecutando.

Por otro lado, la memoria **Heap** es utilizada para almacenar los objetos (incluyendo sus atributos), los objetos almacenados en este espacio de memoria normalmente tienen un tiempo de duración más prolongado que los almacenados en Stack.

Hasta este punto, ya debes haber recordado que las variables de un programa no almacenan el objeto, más bien guardan la referencia del objeto.

Veamos el siguiente gráfico:

```
Automovil auto01 = new Automovil();
int var01 = 50;
String var02 = "";
```



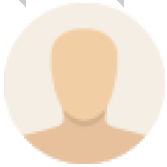
Podemos observar que la referencia del objeto que es representada por un hexadecimal por ejemplo el valor **‘0x777’**, esta contiene la referencia de memoria donde está almacenado este objeto **Automovil**, por lo tanto la variable local **auto01** almacena únicamente esta

referencia de memoria donde está almacenado, donde fue creado este objeto. Aprovechando este mismo gráfico, podemos ver que variables de tipo primitivo como **var01** y **var02**, almacenan directamente el valor donde está creada esta variable.

Si la variable local **auto01** no tuviera referencia a memoria (por ejemplo igual a Null) será eliminado por el Garbage Collector (recolector de basura), cuya función es eliminar los objetos que no estén siendo apuntados por ninguna variable, es decir que este objeto que se ha creado de tipo **Automovil** como está siendo apuntado o referenciado por la variable **auto01**, no será eliminado. Pero si la variable auto01 se libera o es igualado a Null (quitamos la referencia de memoria al objeto **Automovil**), nuestra variable será candidata a ser borrada por el **Garbage Collector**. En este caso la memoria **Heap** que es donde se almacenan los objetos en Java, de esta manera el **Garbage Collector** (recolector de basura) puede buscar aquellos objetos en la memoria Heap que ya no estén referenciados por ninguna otra variable y finalmente liberar el espacio en memoria que ocupaba dicho objeto.

Continuar parte 2 >

JAVA MEMORIA HEAP MEMORIA STACK



Publicado por ejayo

Ingeniero de Sistemas que actualmente me desempeño como líder de equipo, he participado en en roles que cubren el ciclo completo de un proyecto de TI: analista-programador, analista QA, analista funcional y líder de equipo. Ver todas las entradas de ejayo