###### Licenciatura en Ingeniería de Software

###### Taller de Programación Visual III

|  |  |
| --- | --- |
| UNIDAD 3 | Tarea # 8 |
| Nombre: Miguel Alejandro Guevara Mendoza | Fecha: 09/12/2018 |

**Investigación acerca de GC en java**

****

**Introducción**

Uno de los problemas más grandes en el desarrollo de un software es en el manejo de memoria, especialmente en el uso de threads (múltiples procesos), causando problemas de rendimiento a la hora de la ejecución del software y/o uso de demasiados recursos del dispositivo, esto genera que el dispositivo ande lento o pueda crashear. Una de las debilidades más notables de java es el uso de recursos, para nadie es un misterio de java consume muchos recursos pero eso no impide que tenga algunas herramientas que puedan ayudar al desarrollador.

Al igualmente que c#, java también posee su propio GC (Garbage collector) que se encarga de gestionar el uso de memoria, similar que el anteriormente mencionado c#, pero con algunas propiedades propias que lo diferencian, ya que no es infraestructura .NET, en este ensayo veremos las cualidades del garbage collector de java.

**Garbage collector**

Antes de continuar daré una breve explicación de lo que es el garbage collector, es una herramienta que se encarga de gestionar la memoria que usa el programa en ejecución, eliminando y reteniendo datos que no se estén usando todo de manera automática. Pero a diferencia de c# la memoria en java está estructurada de manera diferente, aquí la memoria se maneja por dos bloques principales los cuales son: young generation (joven o nueva generación) y old generation (vieja generación) ambos se encuentran en el llamado **heap**.

**Young generation**

En esta sección es donde se guardan los objetos que no llevan mucho tiempo de ser creados, a sí mismo esta sección se divide en 3 partes las cuales son las siguientes:

En **Eden Space** se encuentran los objetos que son recién creados, y cuando este deja de usarse el garbage de collector lo limpia. Y los objetos que aún siguen por decirlo de alguna manera vivos o activos pasan por otro filtro llamado **Survivor Space** (espacio de supervivencia) el cual está conformado por dos niveles. Cuando un objeto deja de ser usado por al poco tiempo en el survivor space, pasa al nivel uno del survivor space y después de un periodo de tiempo pasa a ser eliminado por el garbage collector. Si dicho objeto continúa usándose en el programa, pasa al nivel dos del survivor space en el cual pasa por los mismos filtros que pasó en los niveles anteriores en caso de ya no usarse los datos del objeto pasan a ser eliminados por el garbage collector.

**Old generation**

En esta sección está dividida por 2 niveles, usualmente por aquí se encuentran los objetos que tienen un ciclo de vida largo que pasan del young generation, e de aquí el nombre de old generation.

Uno de los niveles de esta sección es **Tenured Generation** el cual es el lugar donde se encuentran los objetos de vida larga y se almacenan. Por último la denominada **Permanent Generation** y es donde están cargadas las clases Java que la JVM necesita.

**Otros componentes**

Además de los componentes anteriormente mencionados, hay otros que conforman el garbage collector, los cuales son los siguientes:

**Mark**

Este sirve para crear un punto de referencia inicial a la hora de crear la memoria que se va a utilizar.

**Safepoint**

Un safepoint como su nombre lo indica es un punto o rango ubicado en la cuando se ejecuta un hilo para que así sea identificable, sirve como referencia para el mismo garbage collector y así saber cuál hilo manipular.

**Sweep**

Es el componente del garbage collector que escanea el **heap**, en caso de identificar los objetos como muertos (que quiere decir que ya no se utilizan), los retira de la memoria.

**Compact**

Este componente es desfragmentar los espacios en blanco que se quedan cuando se eliminan datos con el garbage collector de la memoria que se utiliza para el programa esto con el finalidad que a la hora de acceder los datos sea rápido, lo que hace el garbage collector es moviendo los bloques de memoria de manera ordenada, como lo hace la herramienta de Windows para el disco duro.

**Copy**

Este en vez de ser un componente es más bien un proceso que se encuentra en medio de compact el anterior mencionado, lo que se encarga es mover los datos de un punto a otro. Es por esta razón que se lleva de la mano con el componente compact.

**Conclusión**

Como pudimos ver en este ensayo, nos dimos cuenta de cómo funciona el garbage collector de java y al mismo tiempo como está conformado, si bien el garbage collector es demasiado útil para los desarrolladores en la optimización de memoria sin que nos demos cuenta de ello (por ejemplo yo) pero a pesar de que el compilador de java contamos con dicha función es bueno realizar correctas prácticas en el código ejemplo declarar las menor cantidad de variables posibles, ya que como lo dije en el principio del ensayo para nadie es un misterio que java consume demasiados recursos aunque si bien en una computadora de recursos decentes no es demasiado problema.

El problema surge a la hora de desarrollar en dispositivos con memoria limitada que funcionan bajo java, el ejemplo más significativo son los dispositivos con android, es recomendable hacer un debug del uso de memoria ya que por obvias razones estas dispositivos no cuentan con muchos recurso y si puedes modificar el garbage collector aunque esto último no es muy recomendable.

**Referencias**

youtube (7 abr. 2017) Understanding Java Garbage Collection. Obtenido de youtube:

<https://www.youtube.com/watch?v=Uj1_4shgXpk>