JProfiler

**CPU - Hot Spots**

El Hot Spots View muestra una lista de llamadas a cada método. En nuestro caso los métodos más llamados son setText (23%), que actualiza las estadísticas de cada Partido Político en un momento dado, y run en la clase EventDispatchThread (16%).

**CPU - Method Statistics**

La columna Total Time contiene el tiempo total ocurrido para cada método. En nuestro programa el método run en la clase Persona toma mucho más tiempo que los otros métodos.

Mejoramos mucho el performance entre las dos versiones del software, el tiempo total más alto bajó de 36 segundos a 1.868 segundos. Sobre todo, mejoramos mucho el método mover en la clase Persona: de un tiempo total de 2.384 segundos a 67.24 ms.

**Heap - Biggest Objects**

Desde el análisis de Biggest Objects se ve que el consumidor más grande de memoria es sun.awt.image.BufImgSurfaceData, consumiendo el 37% del total de la memoria asignada.

**Live Memory - All Objects**

Las barras horizontales muestran el número de instancias por cada clase de objetos. En nuestro caso el arreglo char[] ha sido el más usado, seguido por el objeto String.

**Live Memory - Class Tracker**

El Class Tracker view ofrece el gráfico del número de instancias en el tiempo por cada package. En la versión anterior el número de packages aumentaba con el tiempo, pero ahora el número se mantiene casi constante bajo 10.

**Telemetría**

La sección Telemetria ofrece muchos gráficos en función del tiempo:

• **GC Activity:** Muestra la actividad del “garbage collector”. En nuestro programa se queda siempre bajo el 2%. En la versión anterior este dato estaba entre 8% y 9% en promedio.

• **Classes:** Muestra el número total de clases cargadas por la JVM, dividido en clases de CPU-profiled y non-CPU-profiled. En la nueva versión del programa el número de clases no-CPU subió, de hecho añadimos nuevas funcionalidades y nuevas clases.

• **Threads:** Muestra el número total de threads vivos en la JVM, según los diferentes estados de los threads. En el análisis de la versión anterior del programa el número de threads bloqueados era mucho más alto, en la nueva versión casi no existen threads bloqueados.

• **CPU Load:** Muestra la carga de la CPU en el tiempo. Conseguimos disminuir mucho este número, de 30% a 2% en promedio entre las dos entregas. Esto se debe principalmente a que dejamos de trabajar las comparaciones con Strings (en vez comenzamos a utilizar enums), y el método para actualizar la interfaz solía llamarse de manera desmesurada, mientras que ahora lo llamamos de manera mucho más controlada y periódica.