

## Fugas de memoria

- Pérdidas de memoria causa común de los bloqueos en las apps
- Se produce fuga de memoria cuando se asigna memoria a un objeto pero nunca lo librea, el recolector de basura cree que todavía ese objeto se necesita porque otros objetos hacen referencia a él, estas referencias deben ser borradas

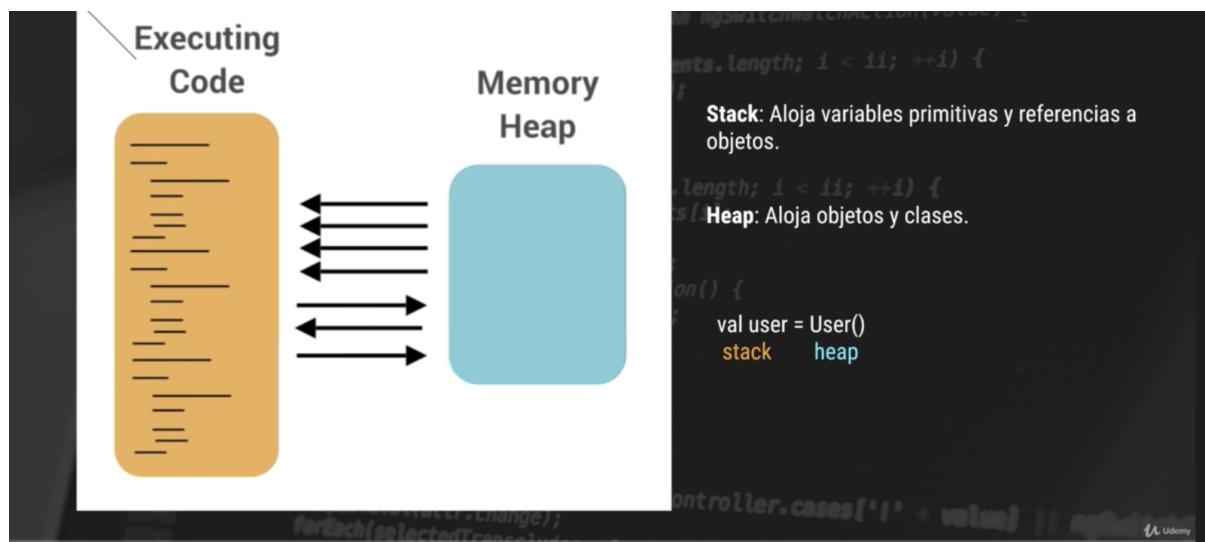
## Herramientas

- Android Profiler
- Leak canary

## Mejorando el rendimiento de una App Android

- Android administra la memoria por nosotros, si usamos la memoria de manera ineficiente, s.o no logra administrarla correctamente
- Limpiar los recursos de memoria lleva tiempo, lo que le quita tiempo a otras tareas
- Todos los procesos, servicios y aplicaciones requieren memoria para almacenar instrucciones y datos
- La App en ejecución asigna memoria para objetos y procesos en su HEAP de memoria asignada
- Asignación de memoria: es el proceso de reservar memoria para los objetos y procesos de nuestra app
- Main thread: hilo principal de la app, también llamado UI thread, si es bloqueado aparece un ANR: “Application not responding”
- Android runtime (ART) (Android 5.0 / API 21) y la máquina virtual Dalvik usan paginación y mapeo de memoria.
- La memoria que modifica una app permanece en la RAM.
- Recolección de basura: libera referencias de objetos para hacer que la memoria esté disponible.

## Manejo de memoria en Android

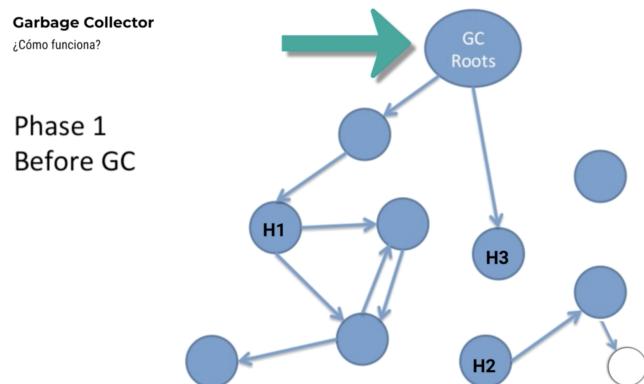


Podríamos decir que el Heap es una estructura dinámica de datos para almacenar los datos en ejecución, el heap permite reservar memoria dinámicamente lo cual da la ventaja la memoria dinámica, también se almacenan variables estáticas y globales. El stack almacena las variables declaradas en bloques anteriores.

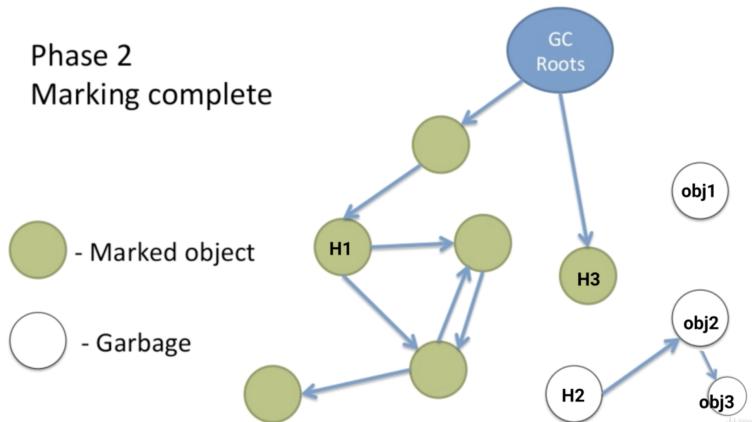
## Garbage Collector

- Encuentra recursos que no son necesarios
- Libera esos recursos del heap
- Es automático, no podemos ejecutarlo nosotros, el sistema operativo tiene un criterio para ejecutarlo
- Por cada iteración del garbage collector puede pasar que el usuario note un tartamudeo de la app, es decir una mala experiencia de usuario

## Funcionamiento Garbage Collector

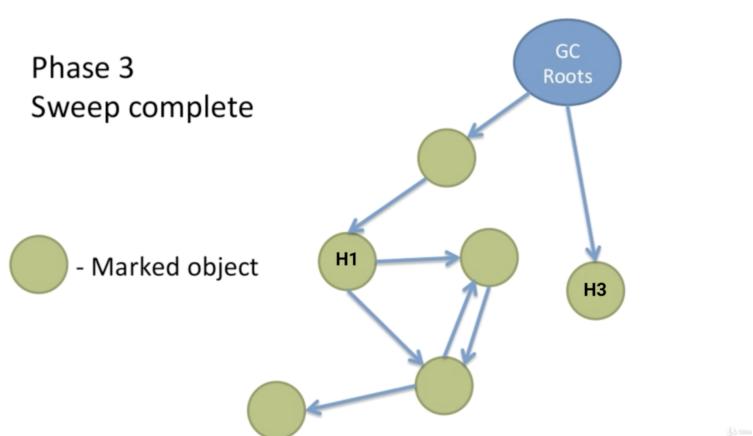


Phase 2  
Marking complete



Siendo H1, H2, H3 actividades que tienen objetos de referencia, si el usuario presiona el botón de atrás, se remueve la pantalla por lo que el garbage collector tiene que borrar las referencias entonces no van a ser más accesibles desde el heap.

Phase 3  
Sweep complete



### Perfilar la memoria de la app

- Nosotros no podemos controlar cuando se ejecuta el recolector de basura, y cuando se ejecuta se deja en pausa la ejecución del código de la app
- Generalmente, no se nota la ejecución del garbage collector, pero en otras ocasiones se nota que la app está lenta y se salta la ejecución de algunos frames. Sucede cuando el sistema asigna memoria más rápido a la app de lo que el sistema la puede recopilar, cuando pierde memoria no se puede recuperar el heap y se ralentiza el sistema
- El tamaño del heap es limitado y nos puede dar out of memory error, este depende de cuanta memoria ram tenga disponible el dispositivo

## Pérdidas de memoria o memory leaks

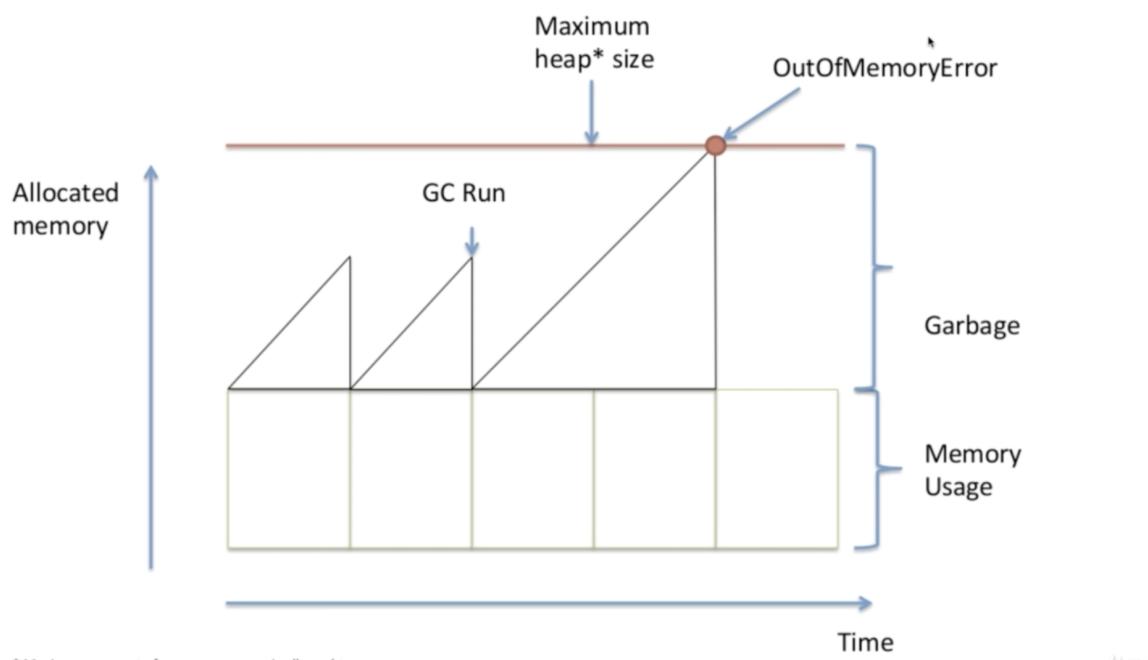
Se produce cuando se asigna memoria para objetos pero nunca se libera, la app opera en lo que queda del heap:

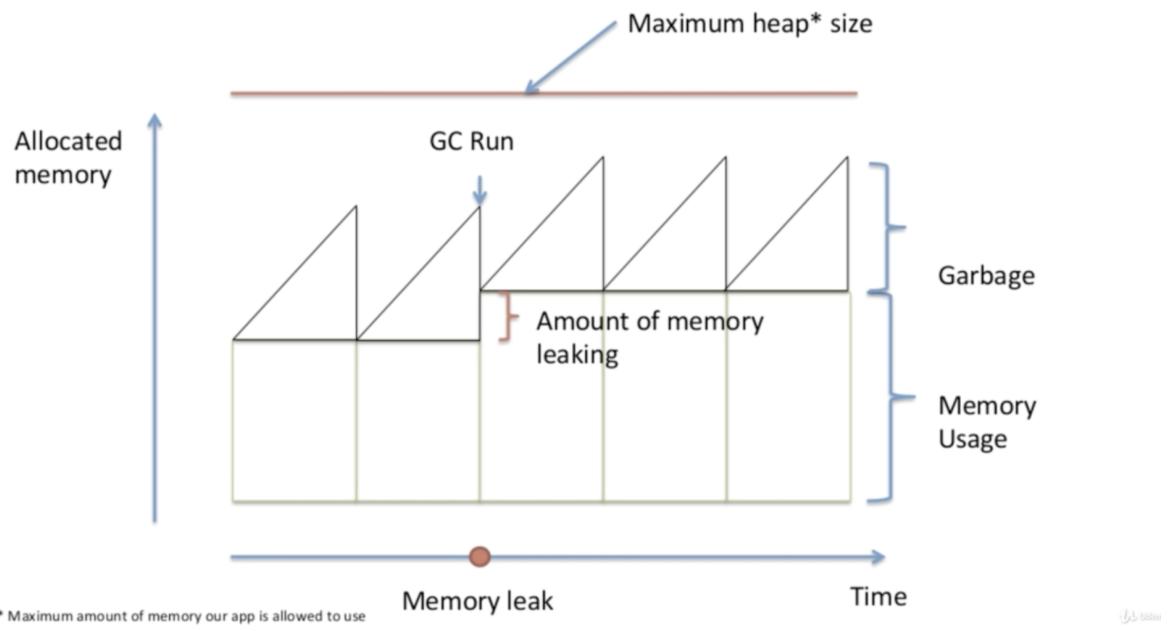
Las pérdidas de memoria pequeñas son las más difíciles de identificar, la app está cada vez más lenta. El resultado final es que se bloquea repentinamente

- El código asigna memoria para muchos objetos y nunca libera referencias, es decir nunca libera memoria
- Menos y menos memoria disponible, la app se ralentiza ya que aumentan los eventos de recolección de basura que lleva cada vez más tiempo
- No mas memoria, la app falla

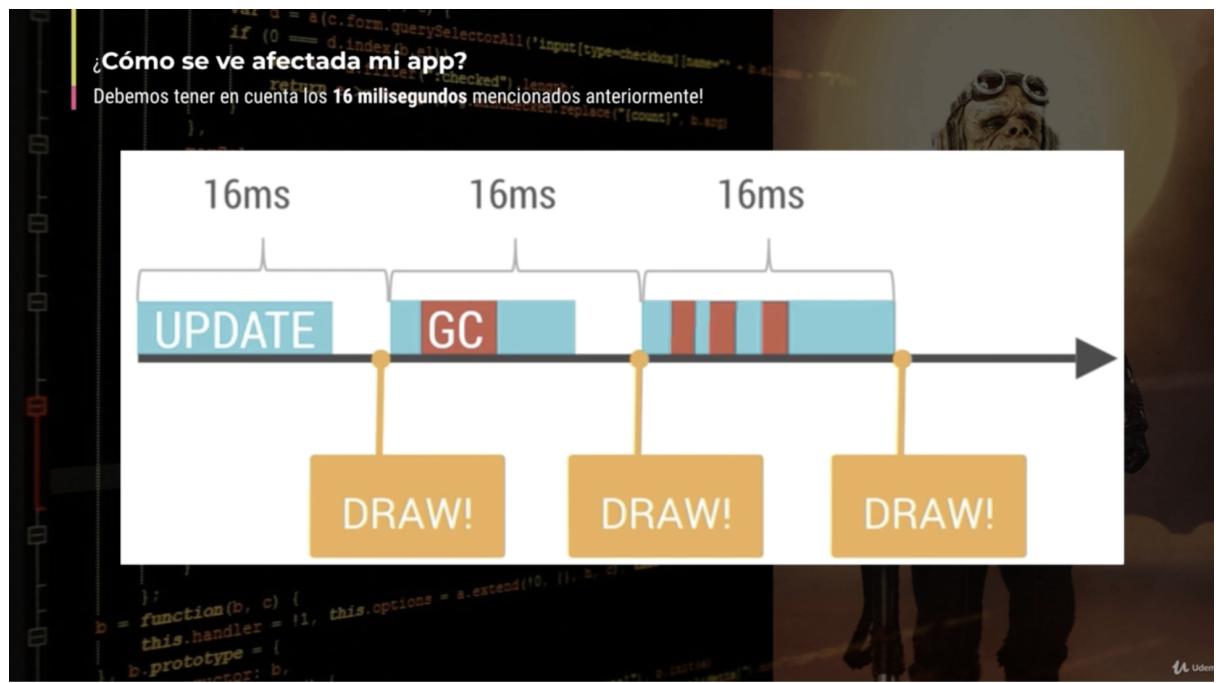
Memoria agotada: puede volverse escasa si se asigna y libera una gran cantidad de objetos en poco tiempo, se inician más recolecciones de basura

Gráfico





Dibujo de frames y ejecución de Garbage collector



### Memory leaks más comunes:

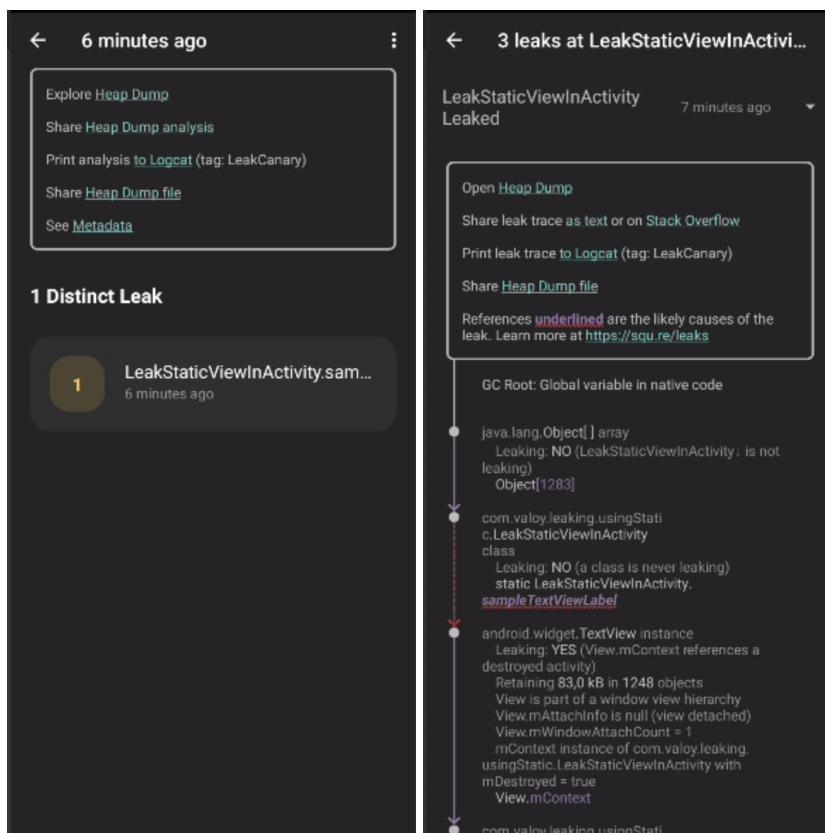
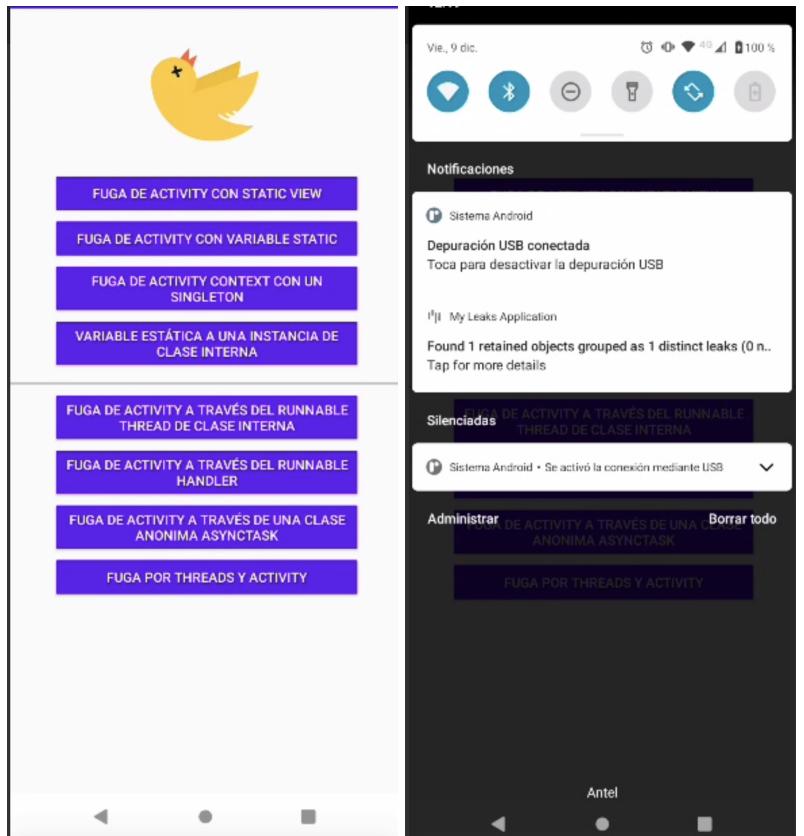
- Singletons referenciando componentes de Android: almacenar activity, dicha actividad permanecerá para siempre
- Suscribirse a listeners y no suscribirse: el más grave, puede ser que nuestra app finalice y la memoria no pueda ser liberada por el garbage collector
- Variables estáticas referenciando componentes de Android: variable estática apuntando a la clase context, el activity.context tiene muchas referencias
- Inner classes y clases anónimas: las clases internas tienen fuertes referencias a instancias de clases externas, al igual que con las clases anónimas

### Resolver algunos Memory Leaks relacionados a la Activity

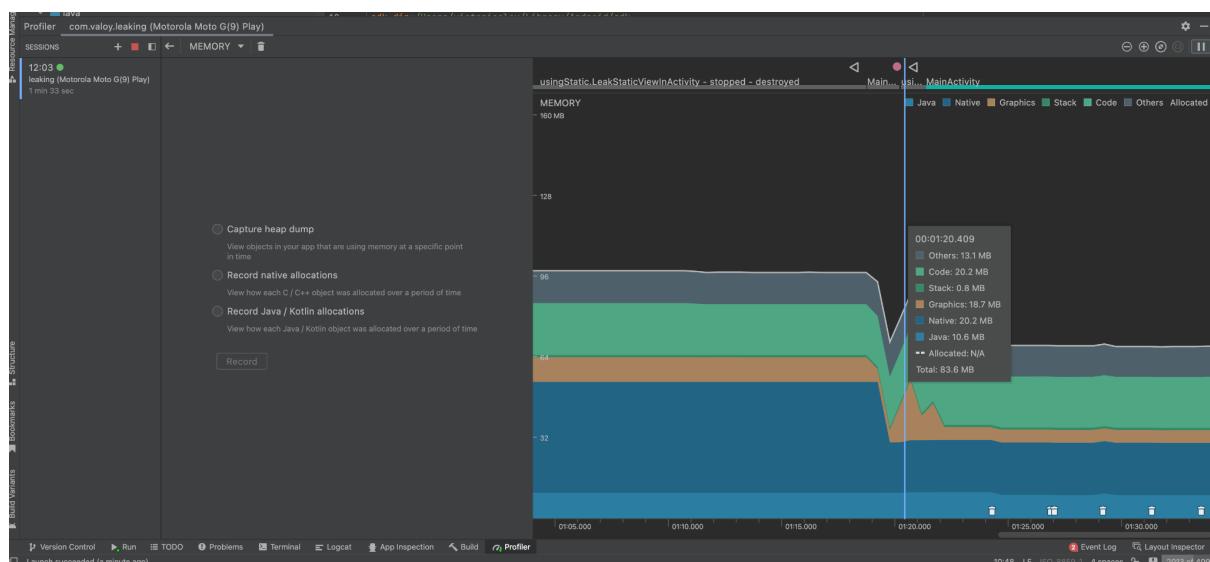
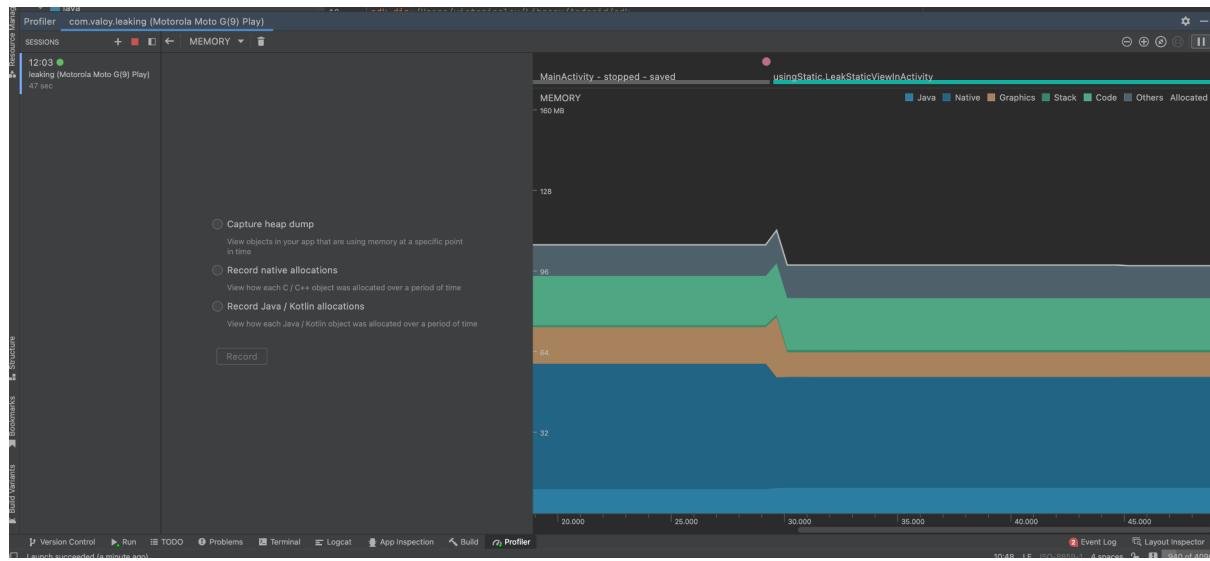
- Usar el contexto de la app, porque hace referencia al ciclo de vida de toda la app
- Se pasan contextos de actividad, y la actividad correspondiente al contexto se sale, la tarea ejecutada debe terminar y se liberan los recursos
- Cambie las clases internas de subprocessos a clases internas estáticas

## Ejemplo Fuga de activity con static view

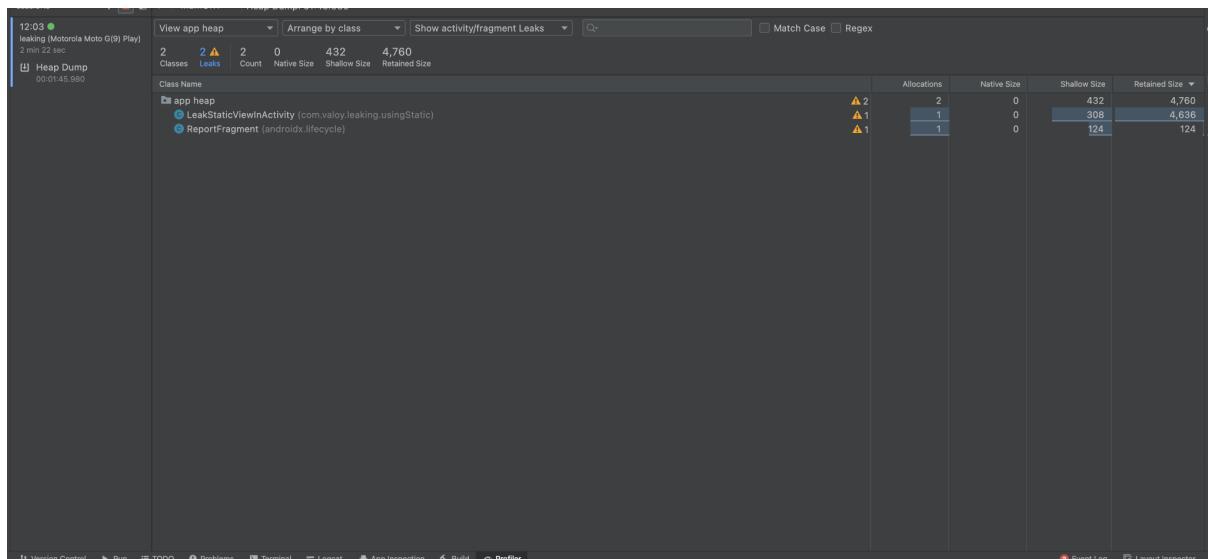
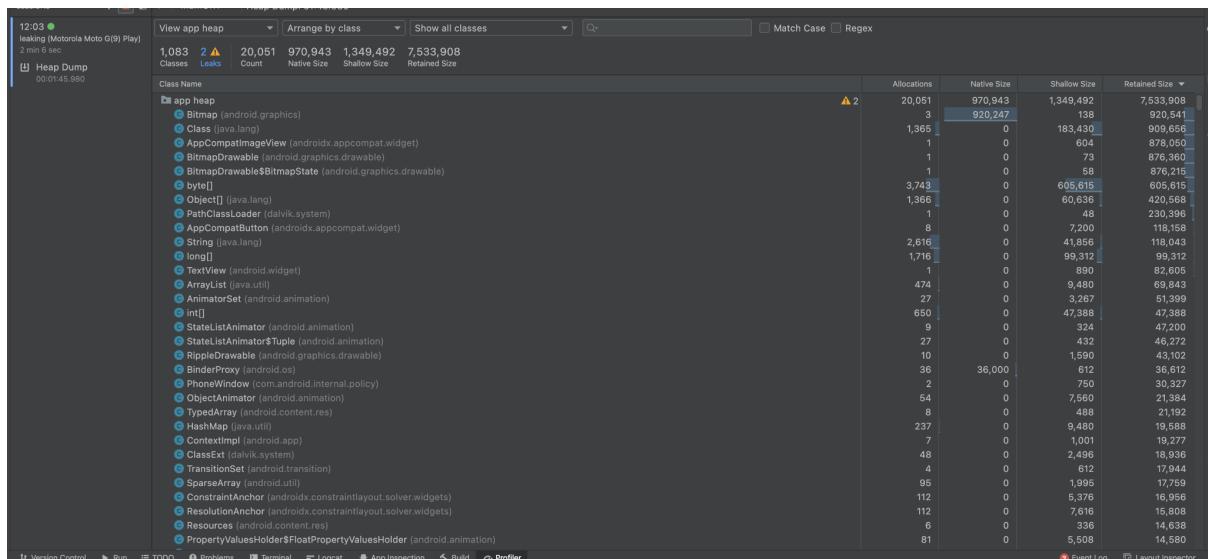
### Leak canary



## Memory profiler



## Capture head dump



## Record Java / Kotlin allocations

Se puede apreciar el call stack

