Breve guida su come scovare Memory Leaks usando Jprofiler

Introduzione

Data una applicazione java molto semplice, configurata per girare su Apache Tomcat, useremo JProfiler per scoprire come rivelare eventuali Memory Leak

Prerequisiti

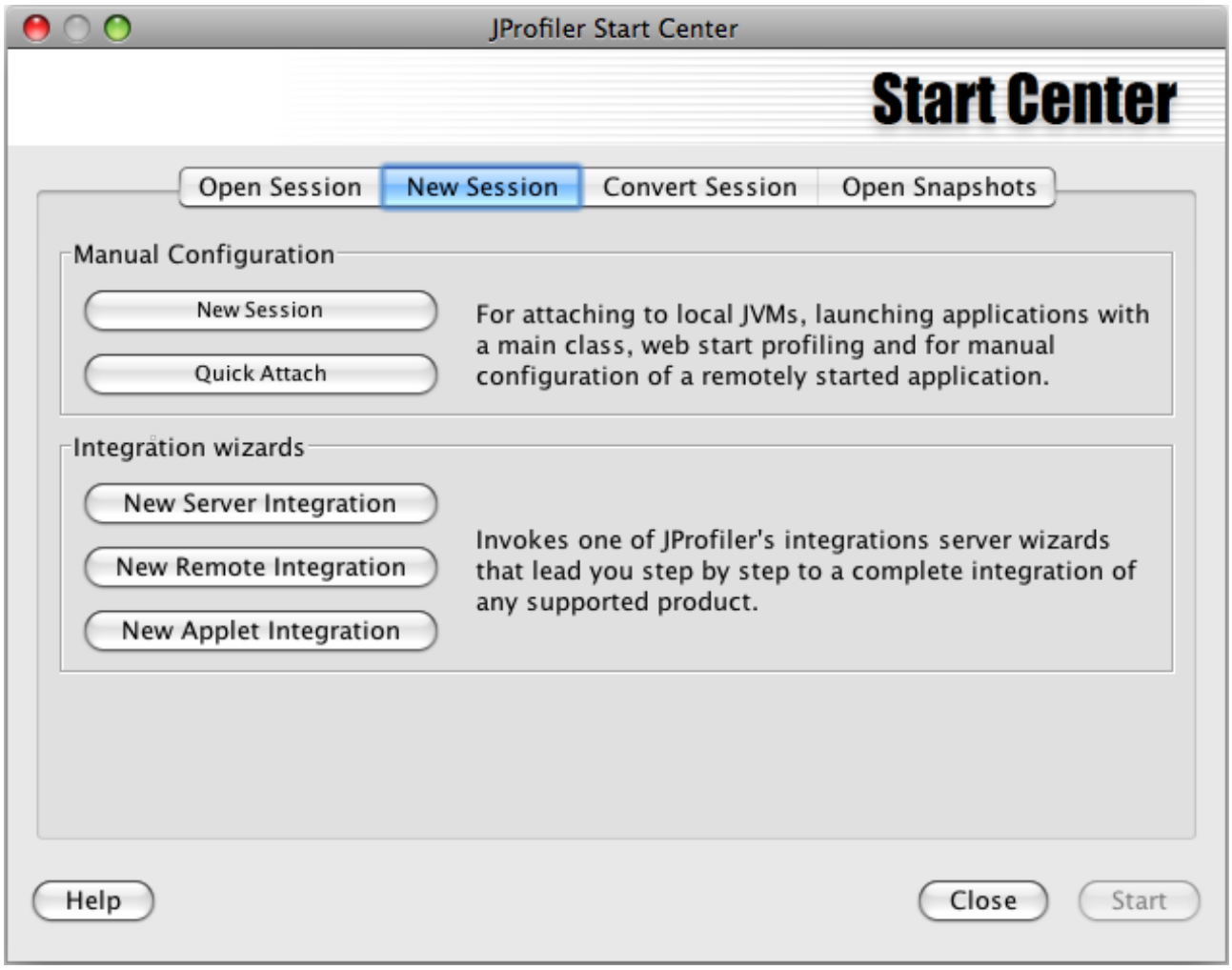
1) Scarica e installa jprofiler da
http://www.ej-technologies.com/products/jprofiler/overview.html

2) Clona con GIT il progetto da
https://github.com/tortitommaso/PerformanceAnalysis

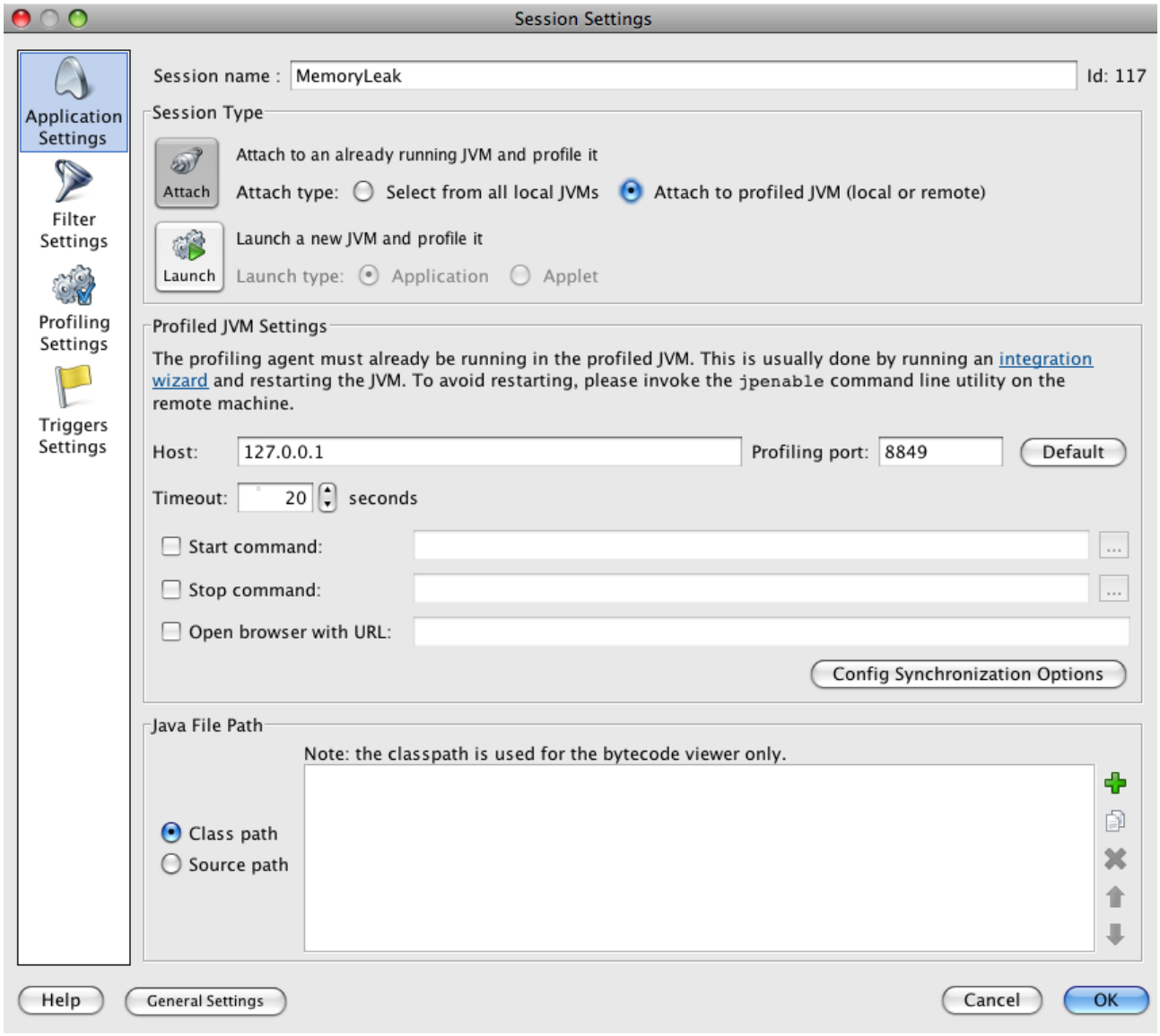
3) Apri il file script/server e personalizza le due variabili

JPROFILER_HOME
CATALINA_HOME

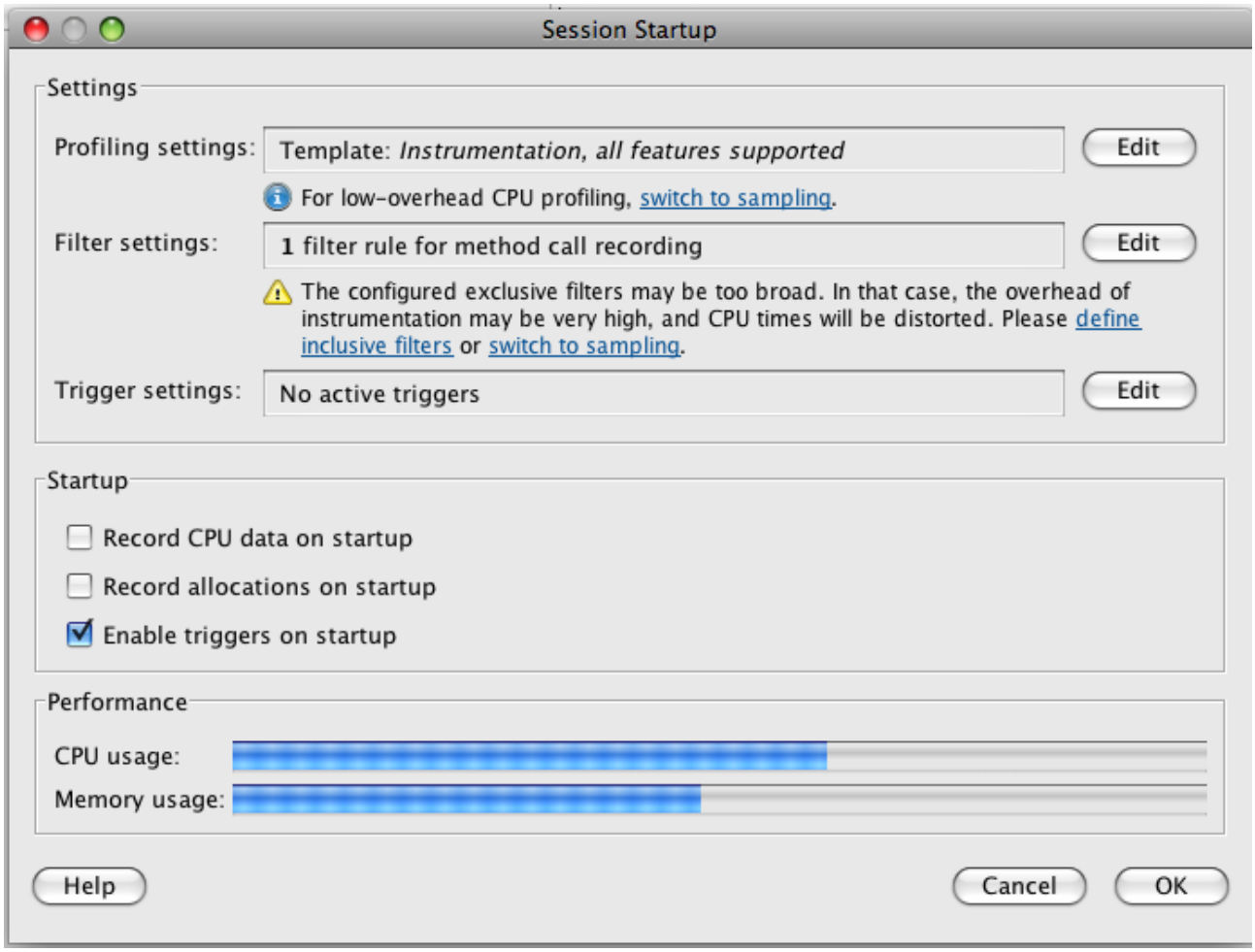
4) esegui script/server
5) Avvia Jprofiler e imposta una nuova connessione:



Seleziona New Session



Imposta come in immagine, scegli Instrumentation

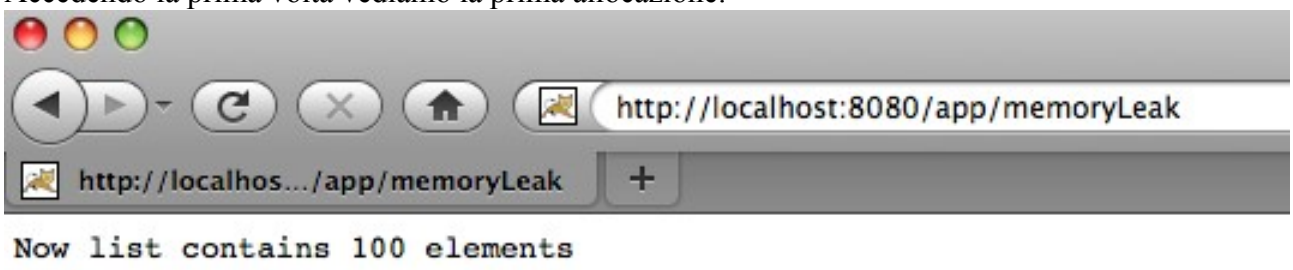


Accedendo all'applicazione all'url
http://localhost:8080/app/memoryLeak
viene invocata una servlet che alloca 100 nuovi interi in un array, senza mai deallocarli.

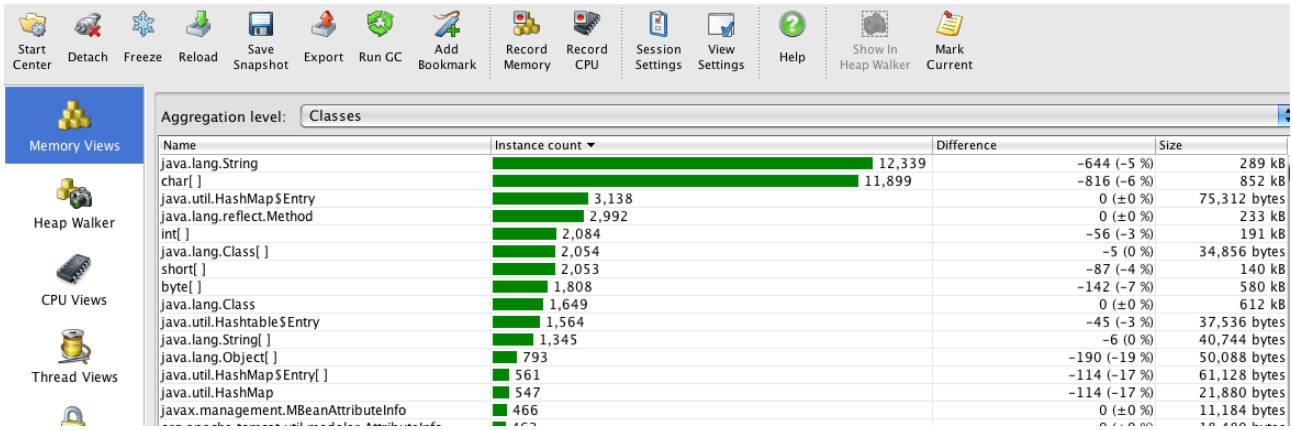
A fronte di un certo scenario utente, si ha un memory leak quando una certa quantità di memoria non viene mai deallocata. Occorre quindi stimolare il sistema con lo scenario d'esempio la prima volta, salvarsi lo stato attuale dell'allocazione della memoria, e ripetere lo scenario. A questo punto l'allocazione è cambiata. Invocando manualmente il Garbage Collector viene deallocata tutta la memoria possibile, quindi confrontando lo stato attuale con quello di partenza si possono studiare quelle parti del sistema 'sospette'.

Concretamente:

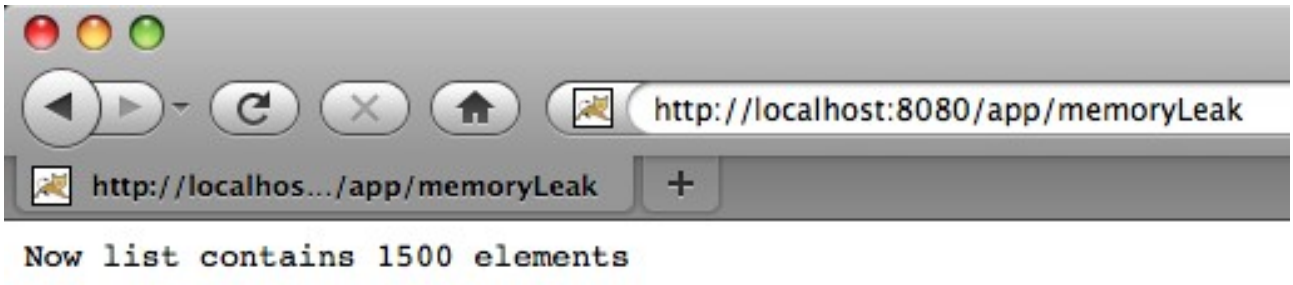
Accedendo la prima volta vediamo la prima allocazione:



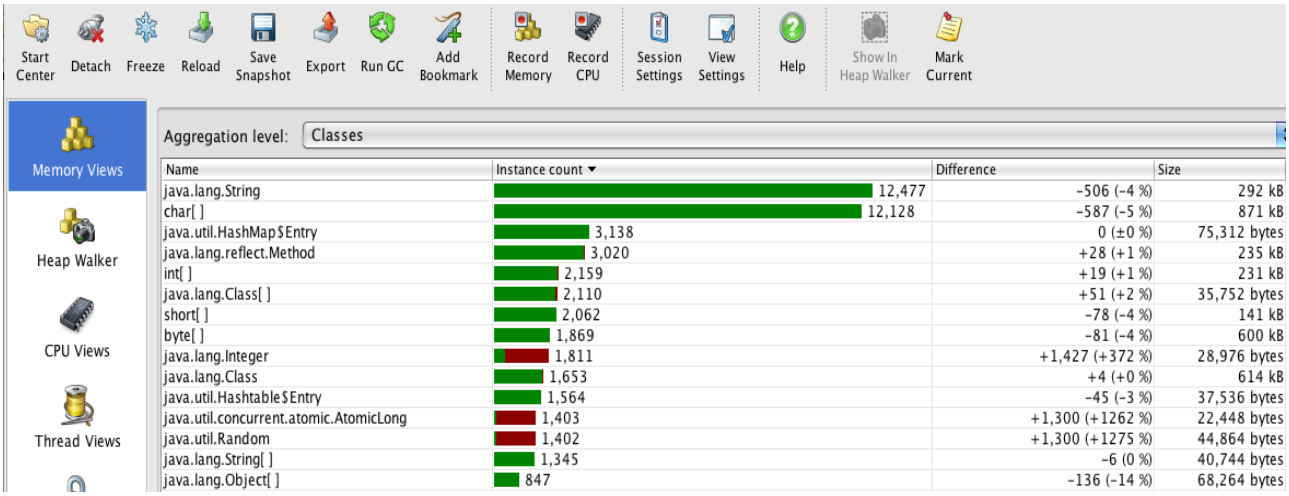
E in Jprofiler salviamo lo stato corrente cliccando su "Mark Current":



Facendo reload dell'url piu volte:

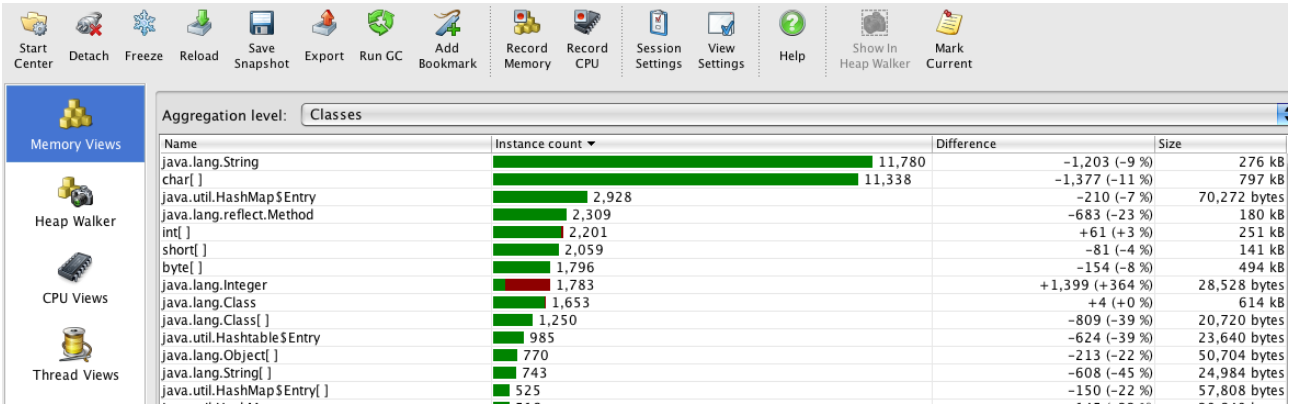


E jprofiler indica una diversa allocazione:



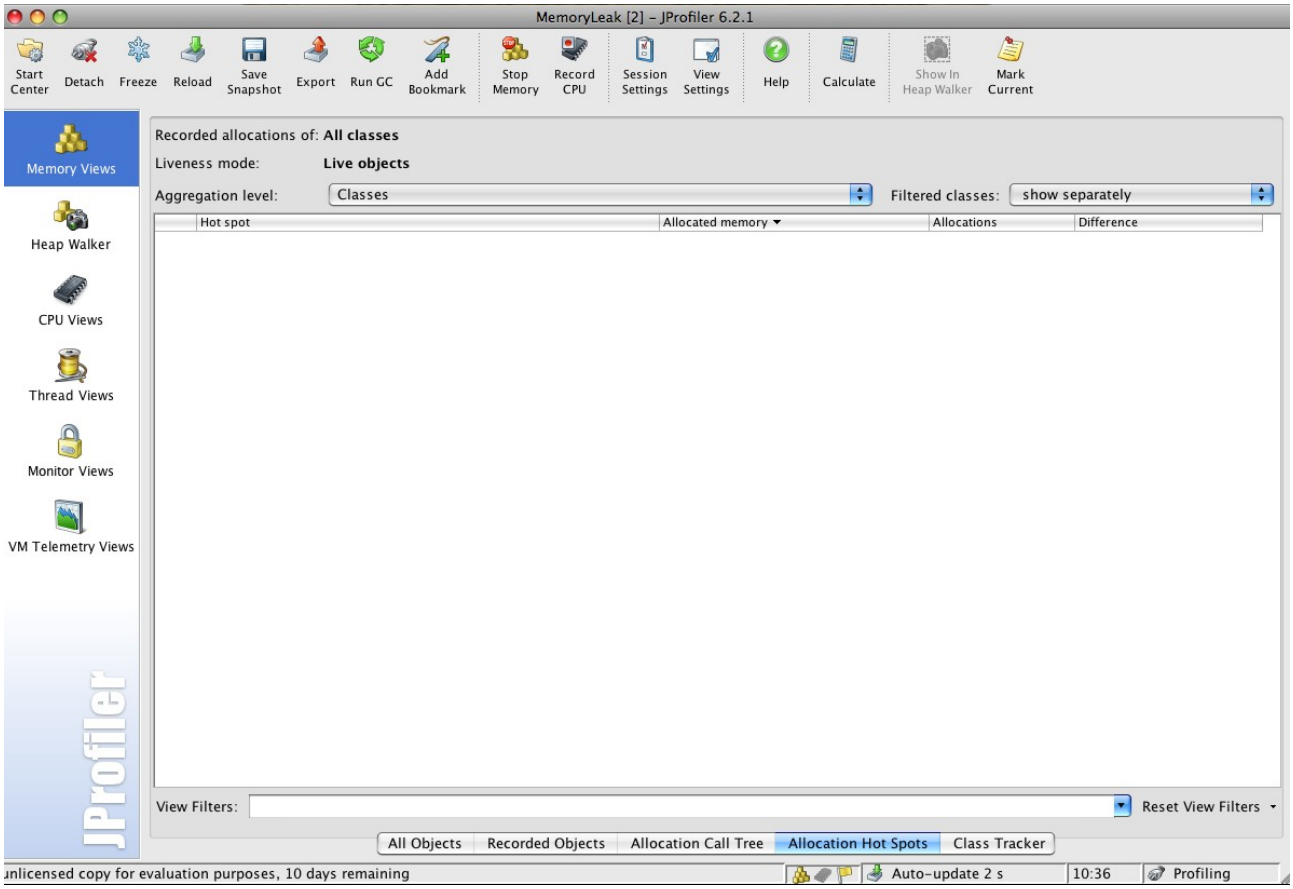
Possiamo vedere allocati piu java.lang.Integer dell'inizio e anche varie istanze di java.util.Random

Dopo aver lanciato "Run GC":

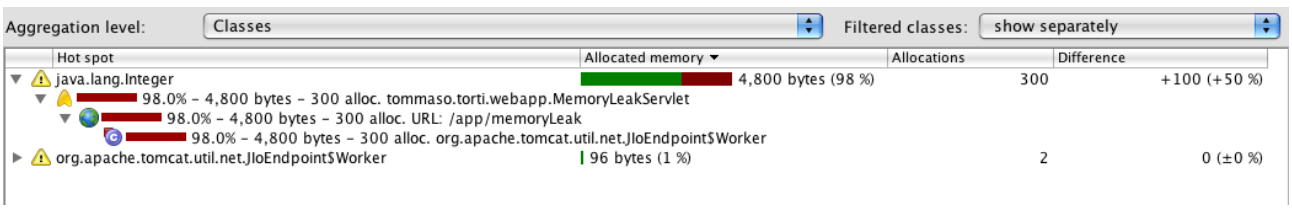


Notiamo quindi che c'è un gran numero di istanze di java.lang.Integer che non vengono deallocati.

Per scoprire qual è il punto dell'applicazione che genera istanze che non vengono deallocate si userà la vista Allocation Hot Spots, cliccando poi su Record Memory prima di rieseguire lo scenario:



Al termine dello scenario click su "Stop Memory" e "Run GC". Notiamo:



Sappiamo quindi che dovremo indagare la classe MemoryLeakServlet.

Il codice infatti dimostra il memory leak:

```
	private List list = new ArrayList();

	@Override
	protected void service(HttpServletRequest req, HttpServletResponse response)
throws ServletException, IOException {
		for (int i = 0; i < 100; i++) {
			Random random = new Random();
			list.add(random.nextInt());
		}
		response.getWriter().print("Now list contains " + list.size() + "
elements");
	}
```

Cambiando il codice in:

```
	@Override
	protected void service(HttpServletRequest req, HttpServletResponse response)
throws ServletException, IOException {
		List list = new ArrayList();
		for (int i = 0; i < 100; i++) {
			Random random = new Random();
			list.add(random.nextInt());
		}
		response.getWriter().print("Now list contains " + list.size() + "
elements");
	}
```

e ristartando il server e la procedura indicata notiamo che non c'è piu alcun memory leak.