

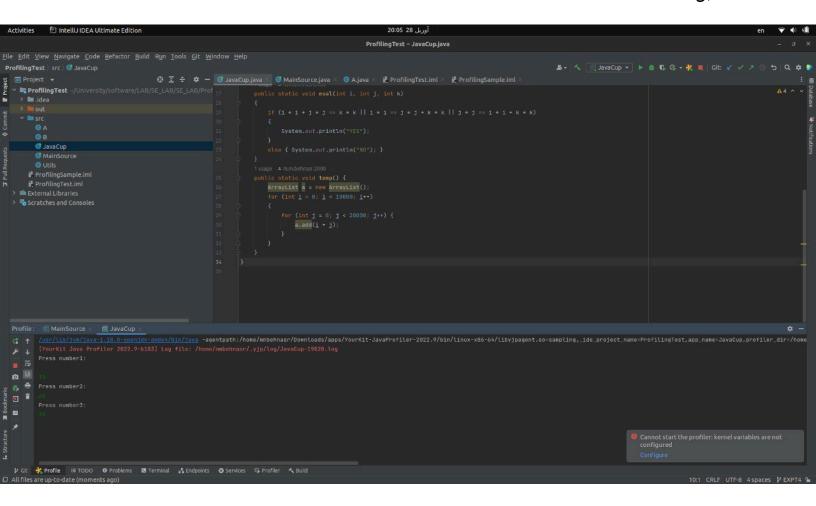
دانشگاه صنعتی شریف دانشکده مهندسی کامپیوتر

آزمایش چهارم مهندسی نرمافزار

محمدرضا دولتی ۹۷۱۱،۴۱۱

محمد مهدی بهنصر ۹۷۱۰۵۷۹۳ سوال ۱) برای این سوال در نرم افزار YourKit مشاهده می کنیم که بیشترین منبعی که مصرف می شود توسط تابع for میباشد که در یک for با 20000 تکرار درون یک for دیگر که 10000 بار تکرار دارد در حال اجراست و در هر تکرار یک مقدار را به Arraylist اضافه می کند. این موضوع سربار زیادی را برای CPU به وجود می آورد.

در تصویر زیر کد نوشته شده در JavaCup و تصاویر مربوط به اجرای برنامه در YourKit و InteliJ قابل مشاهده است.



آور لل 28:33 28:20

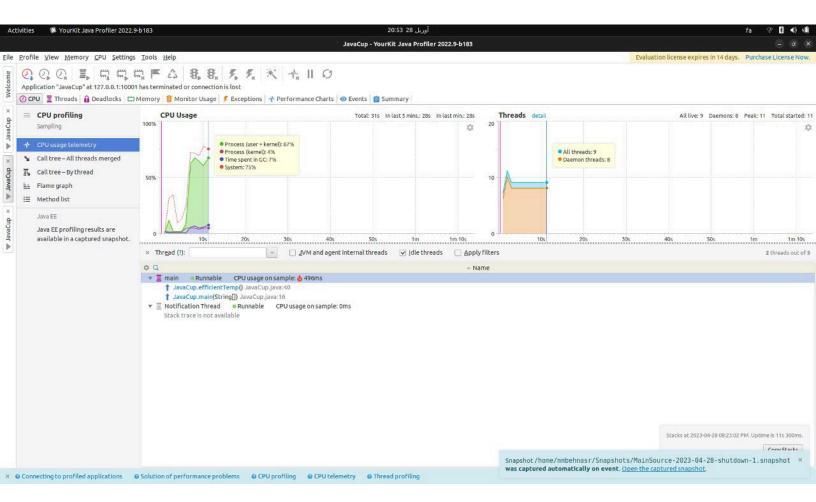
Activities

YourKit Java Profiler 2022.9-b183

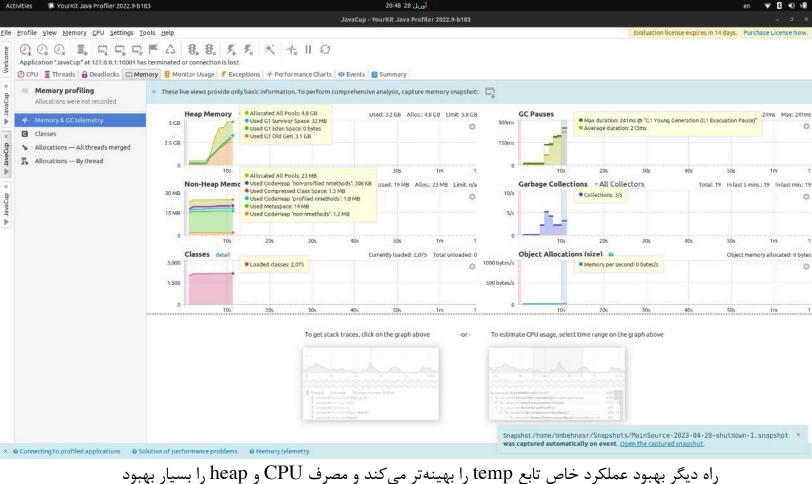
همانطور که از تصاویر صفحه ی قبل مشخص است، مصرف حافظه ی Heap در طول زمان افزایش می یابد (به علت بزرگی حجم آرایه نهایی نیز می باشد.) و در نهایت برنامه با پر شدن کامل این حافظه خاتمه پیدا می کند.

هم چنین با مشاهده ی نمودارهای دیگر می توان برداشت کرد که مصرف حافظه ی غیر Heap ای در طول زمان تغییری نخواهد کرد.

سوال ۲) (بهبود بخشیدن temp) برای راهحل اول بهبود این نمودارها و تابع temp از تابع efficientTemp استفاده می کنیم، این تابع در واقع همان وظایف تابع temp را انجام می دهد، با این تفاوت که باید سایز آرایه ی a را از همان ابتدا مشخص کنیم، در نتیجه از کپی کردنها بیهوده تفاوت که باید سایز آرایه جلوگیری می کند و همانطور که در تصاویری که در ادامه آمدهاند، قابل مشاهده است، میزان CPU Usage از 70% به 67% کاهش پیدا کرده است.

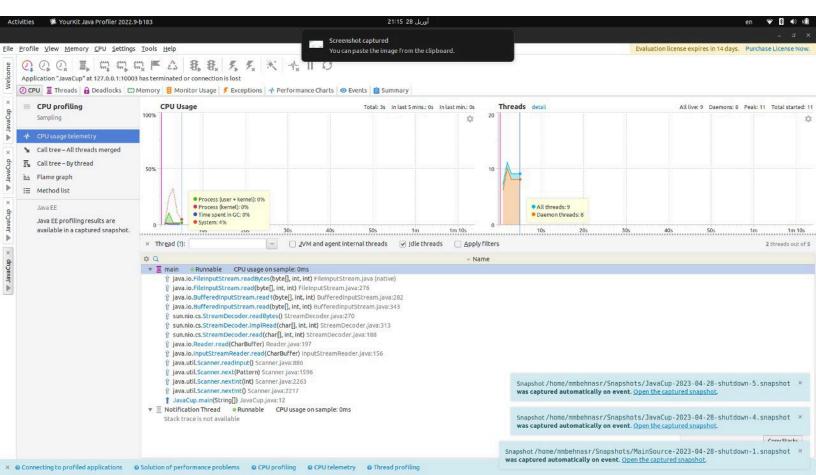


البته مصرف مموری همچنان مشابه قبل است، چون در نهایت هدف تابع پر کردن چنین لیست بزرگی است.



راه دیگر بهبود عملکرد خاص تابع temp را بهینهتر میکند و مصرف CPU و heap را بسیار بهبود میبخشد.

در ادامه از تابع efficientTempPlus استفاده می کنیم، این تابع مقادیر مورد نظر تابع temp را در arraylist a می ریزد. حال با اجرای برنامه با YourKit داریم:



To get stack traces, click on the graph above

Used: 18 MB Alloc: 23 MB Limit: n/a

آوربل 28 21:15

YourKit Java Profiler 2022.9-b183

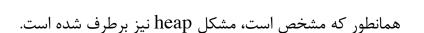
▼ JavaCup × ▼ JavaCup

Non-Hean Memory V All Pools

× @ Connecting to profiled applications @ Solution of performance problems @ Memory telemetry

Allocated All Pools: 23 MB

Allocated All Pools: 25 MB
Used Code-Heap 'non-profiled amethods': 252 KB
Used Compressed Class Space: 1.3 MB
Used Code-Heap 'profiled amethods': 1.4 MB
Used Code-Heap 'non-amethods': 1.2 MB
Used Mctaspace: 14 MB
Used Code-Heap 'non-amethods': 1.2 MB



Garbage Collections All Collectors

Object Allocations (size)

To estimate CPU usage, select time range on the graph above

was captured automatically on event. Open the ca

was captured automatically on event. Open the cap

Snapshot/home/mmbehnasr/Snapshots/JavaCup-2023-04-28-shutdown-5.snapshot

Snapshot/home/mmbehnasr/Snapshots/JavaCup-2023-04-28-shutdown-4.snapshot ×

Snapshot/home/mmbehnasr/Snapshots/MainSource-2023-04-28-shutdown-1.snapshot $\quad \times \mbox{ was captured automatically on event.} \begin{tabular}{ll} Open the captured snapshot. \begin{tabular}{ll} Open the captured snapshot.$

1000 bytes/s