

آزمایشگاه مهندسی نرم افزار

گزارش آزمایش پنجم

Profiling

دانشگاه صنعتی شریف

دانشکده مهندسی کامپیوتر

سینا الهی منش ۹۸۱۰۱۱۶۹

سارا زاهدی موحد ۹۸۱۷۰۸۴۹

تابستان ۱۴۰۲

آدرس مخزن گیت: <https://github.com/sarazm2000/Profiling>

نصب و راه اندازی نرم افزار YourKit

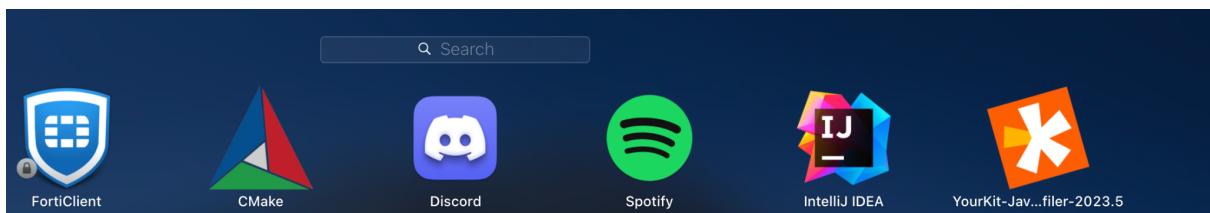
ابتدا ابزار مورد نیاز را از سایت معرفی شده دانلود کردیم.



macOS
arm64, x64

[Download disk image](#)

و سپس برنامه را نصب کردیم.



Get Evaluation License Key

Get a free 15 days evaluation license key for a fully functional version of the YourKit Java Profiler.

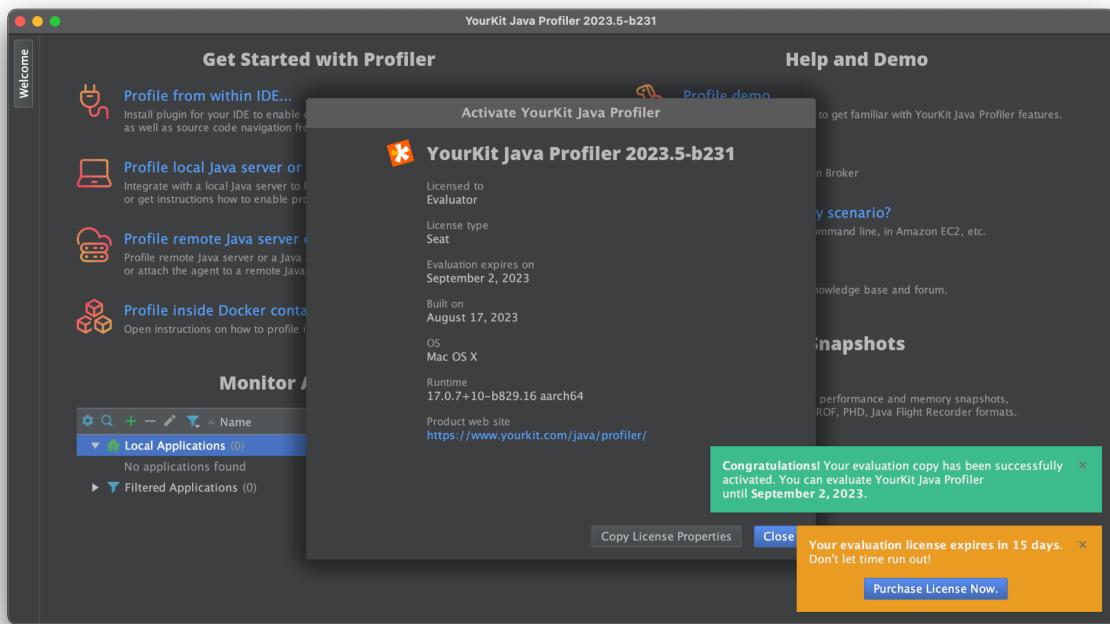
All the information we collect is for YourKit internal use only. We take your privacy seriously and will not disclose this information to any third parties.

sarazm.2000@gmail.com

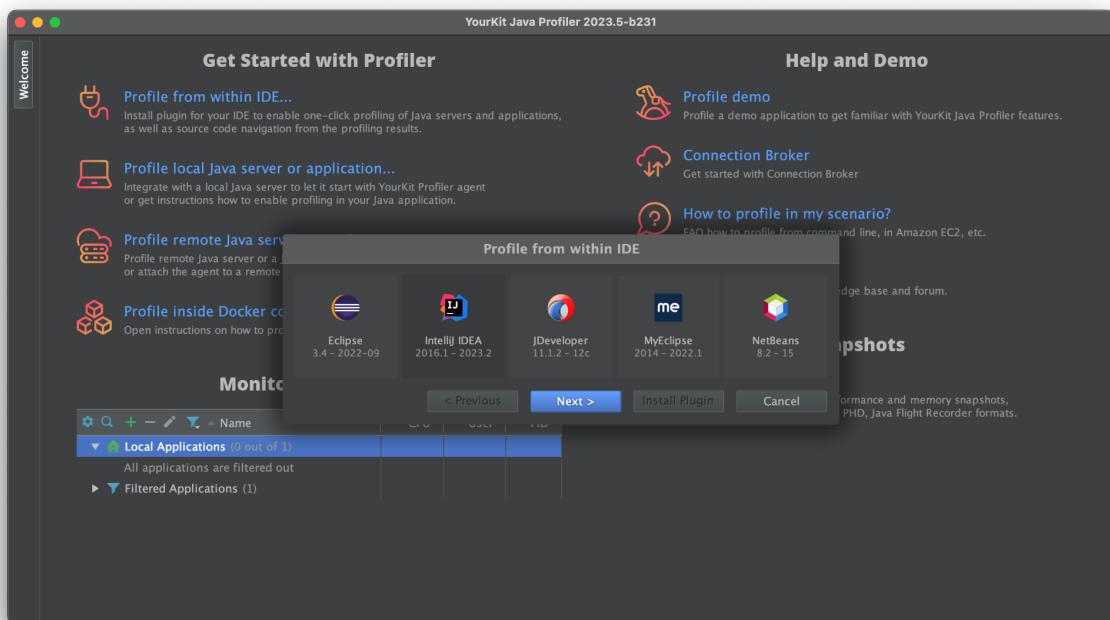
I have read and accept [Privacy Policy](#)

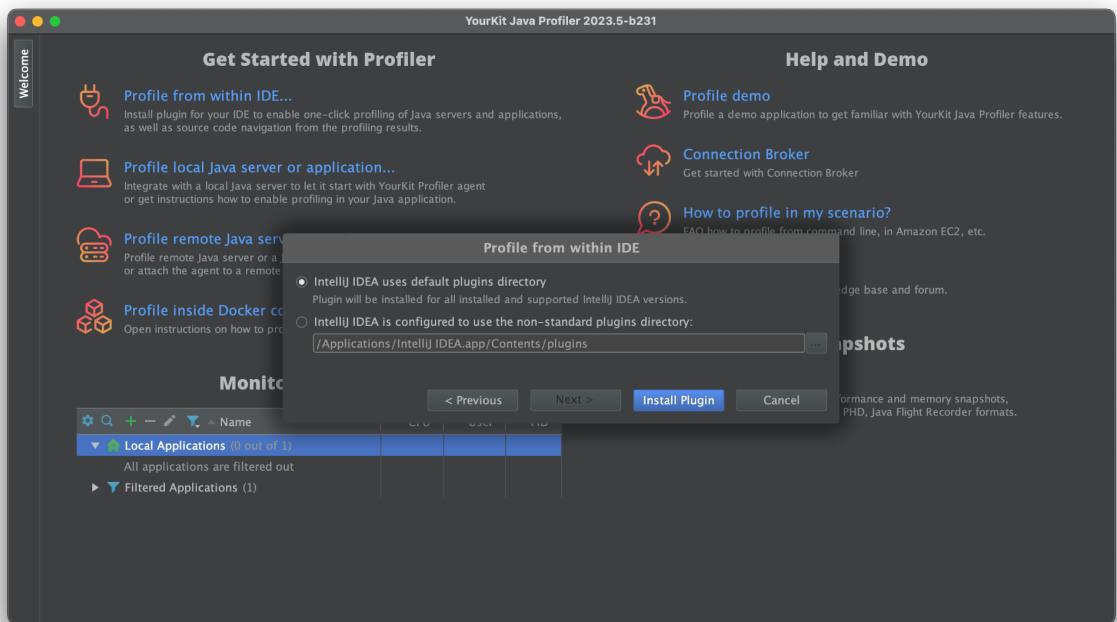
[Send Evaluation License](#)

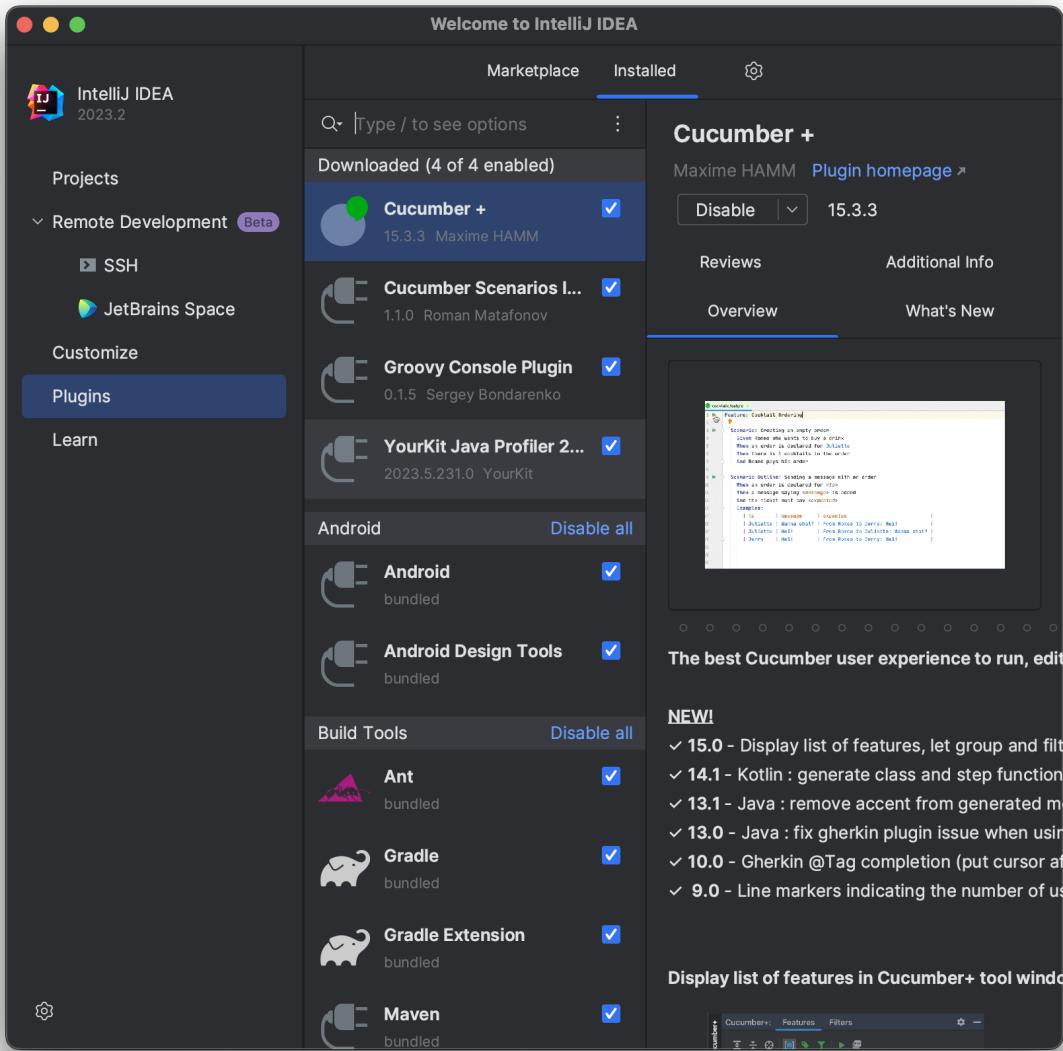
پس از آنکه لینک فعالسازی به آدرس ایمیل ارسال شد این فرایند با موفقیت انجام شد.



طبق مراحل گفته شده پیش رفتیم و پلاگین yourkit را به اینتلیجی افزودیم.



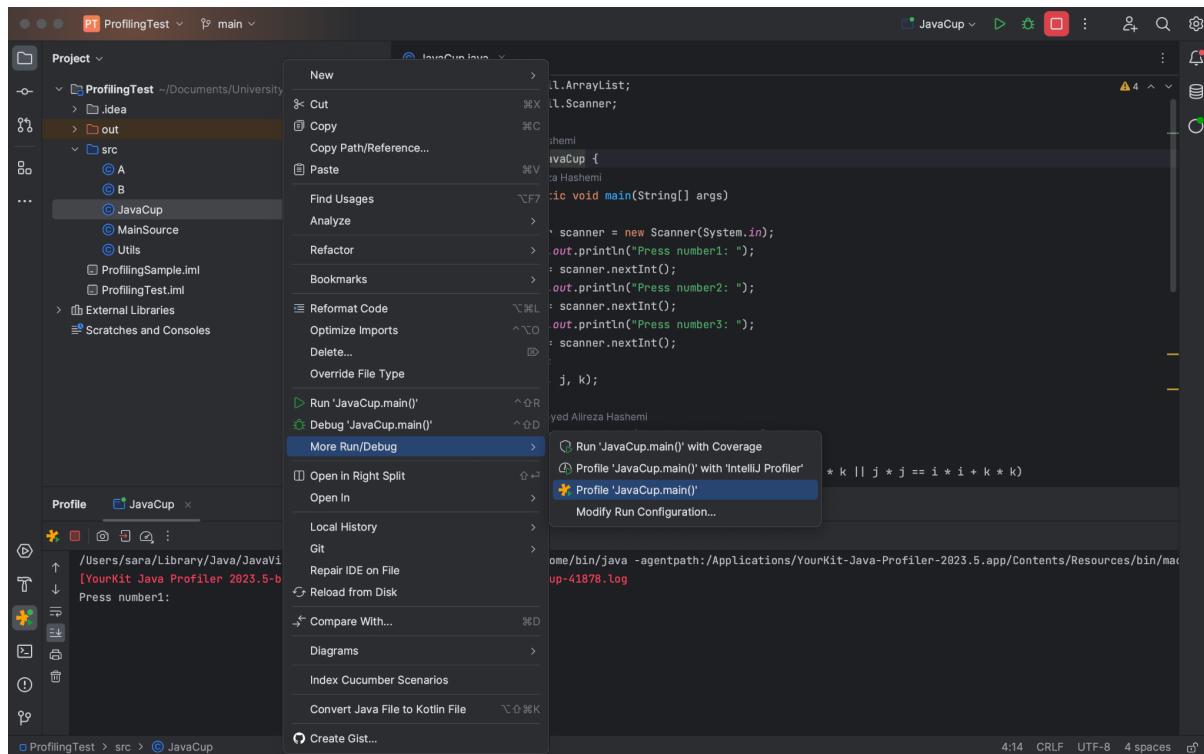




تمرین

۱- از پروژه ProfilingTest، عملیات Profiling را با استفاده از Yourkit بر روی کلاس JavaCup اجرا نموده و تابعی از پروژه که بیشترین مصرف منابع را دارد شناسایی کنید و آن را در گزارش خود در فایل README توضیح دهید. سپس نحوه پیاده‌سازی آن تابع را به گونه‌ای تغییر دهید که مصرف منابع نسبت به قبل بهتر شود.

مطابق خواسته تمرین عملیات پروفایل را روی کلاس گفته شده اجرا می‌کنیم



The screenshot shows an IDE interface with a project named "ProfilingTest". The "src" folder contains files "A.java", "B.java", "JavaCup.java", "MainSource.java", and "Utils.java". The "JavaCup.java" file is open, displaying the following code:

```

import java.util.ArrayList;
import java.util.Scanner;

public class JavaCup {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Press number1: ");
        int i = scanner.nextInt();
        System.out.println("Press number2: ");
        int j = scanner.nextInt();
        System.out.println("Press number3: ");
        int k = scanner.nextInt();
        temp();
        eval(i, j, k);
    }

    public static void eval(int i, int j, int k) {
        if (i * i + j * j == k * k || i * i == j * j + k * k || j * j == i * i + k * k)
    }
}

```

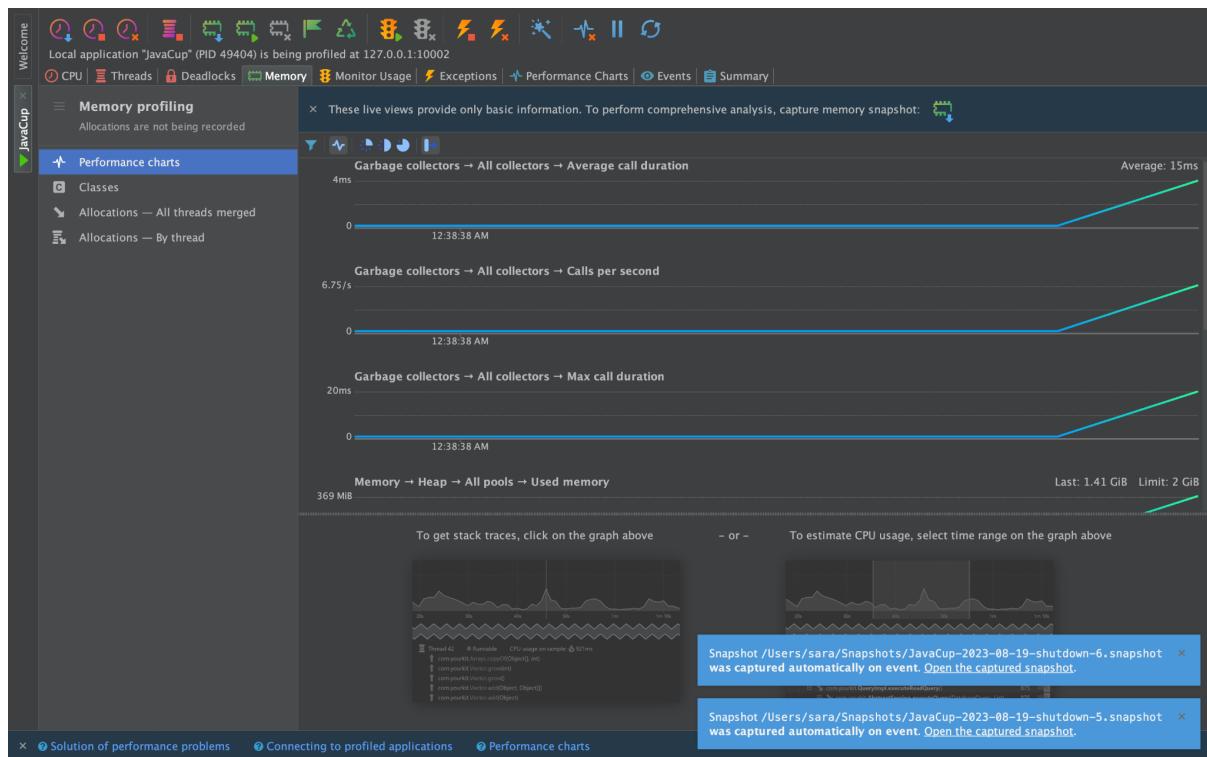
The terminal window below shows a Java profiler log:

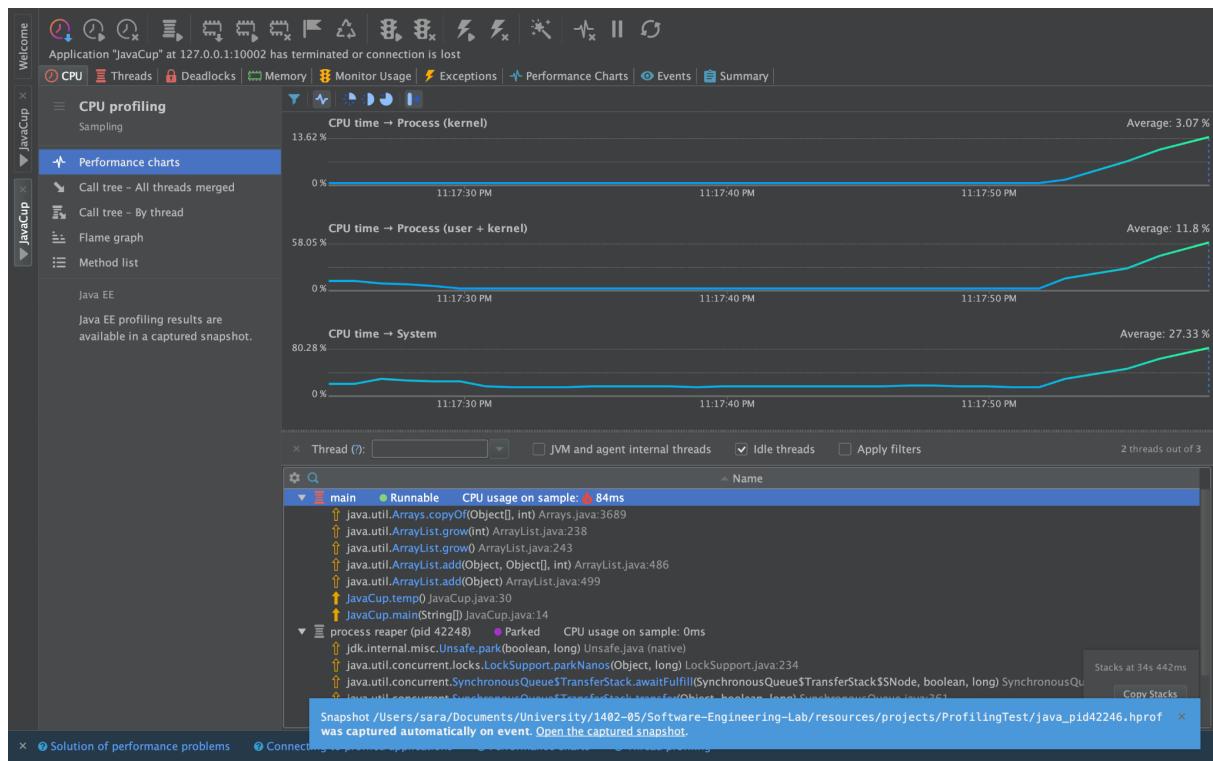
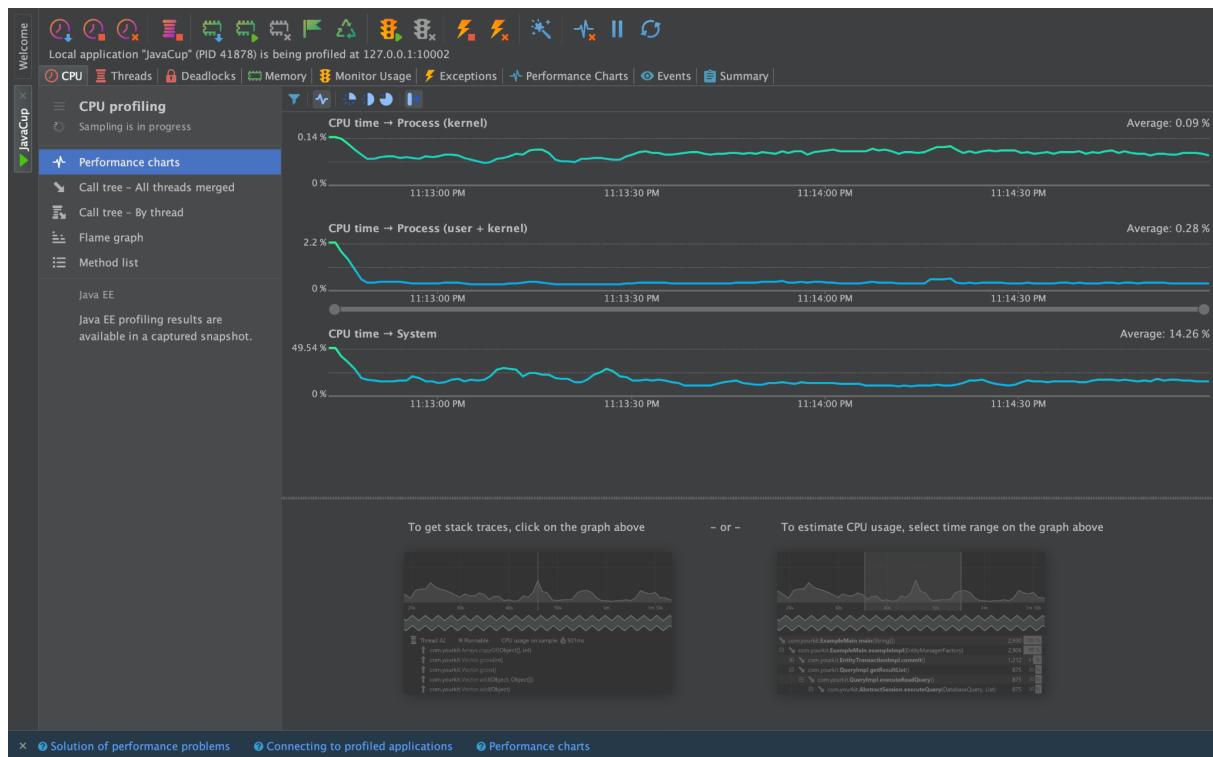
```

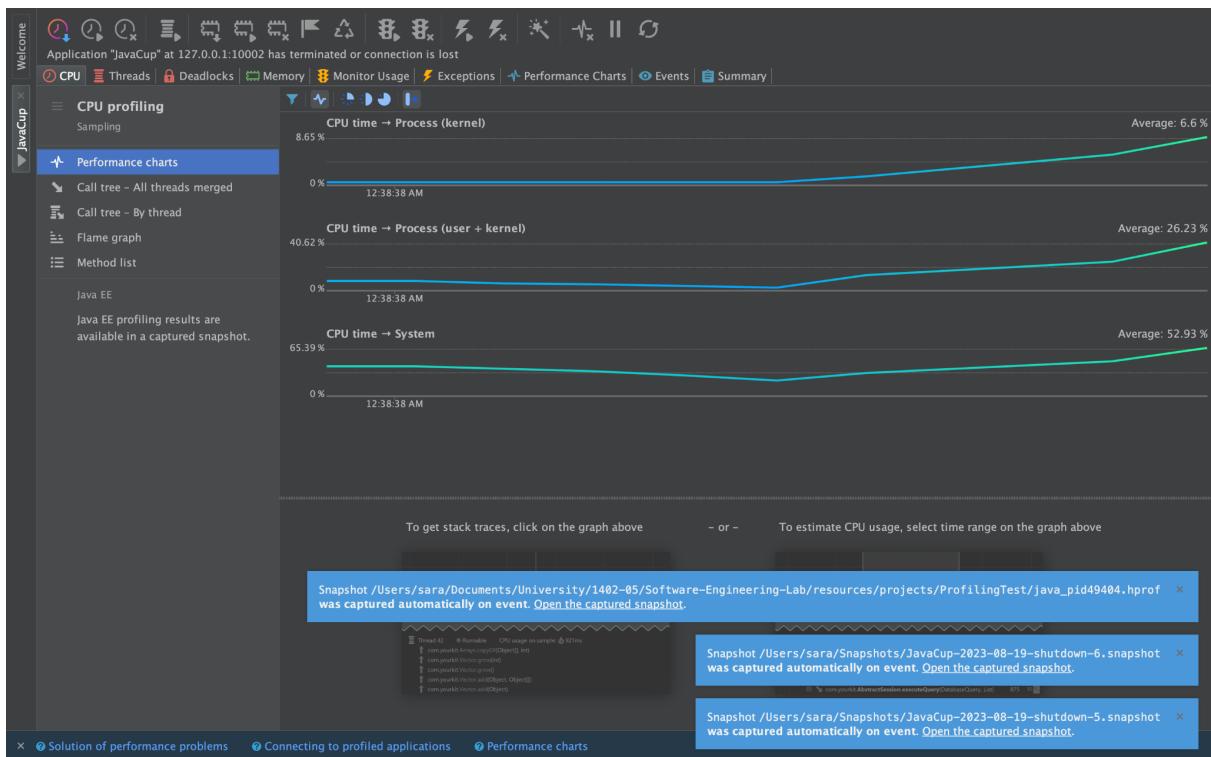
[YourKit Java Profiler 2023.5-b231] Log file: /Users/sara/.yjp/log/JavaCup-41878.log
Press number1:

```

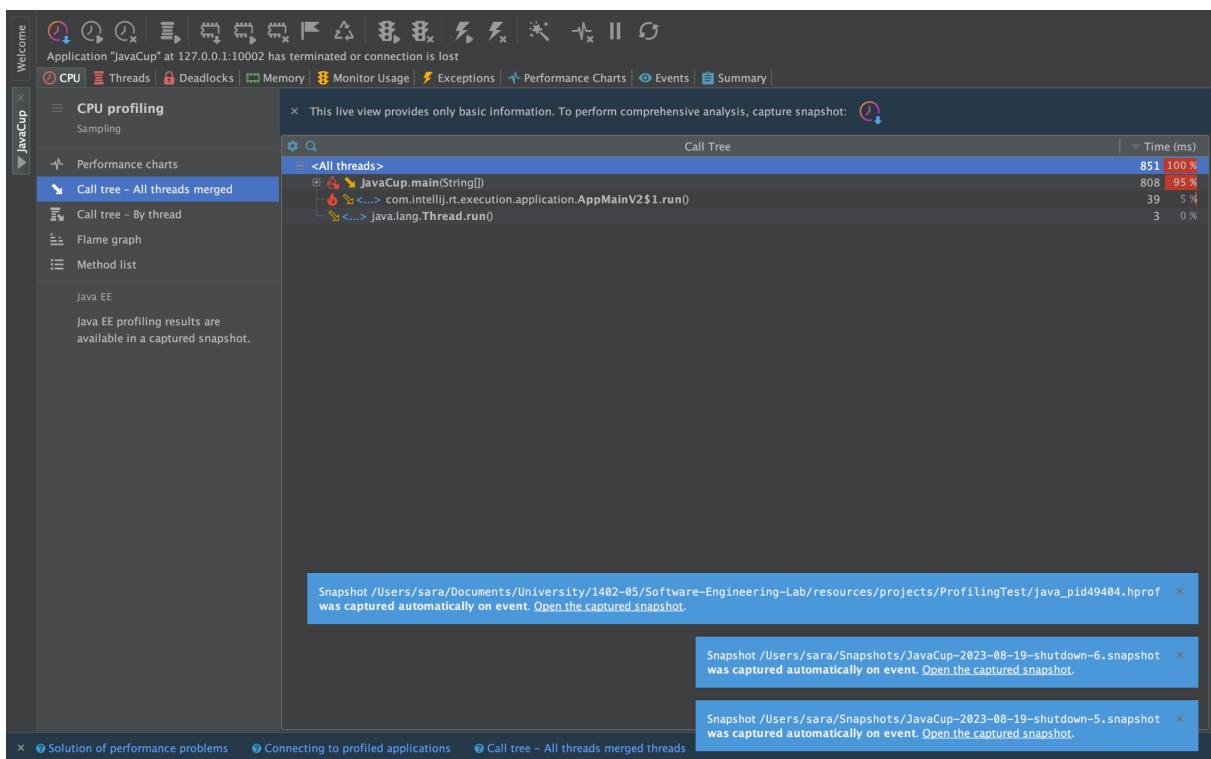
میتوانید صفحه YourKit را نیز بینید.

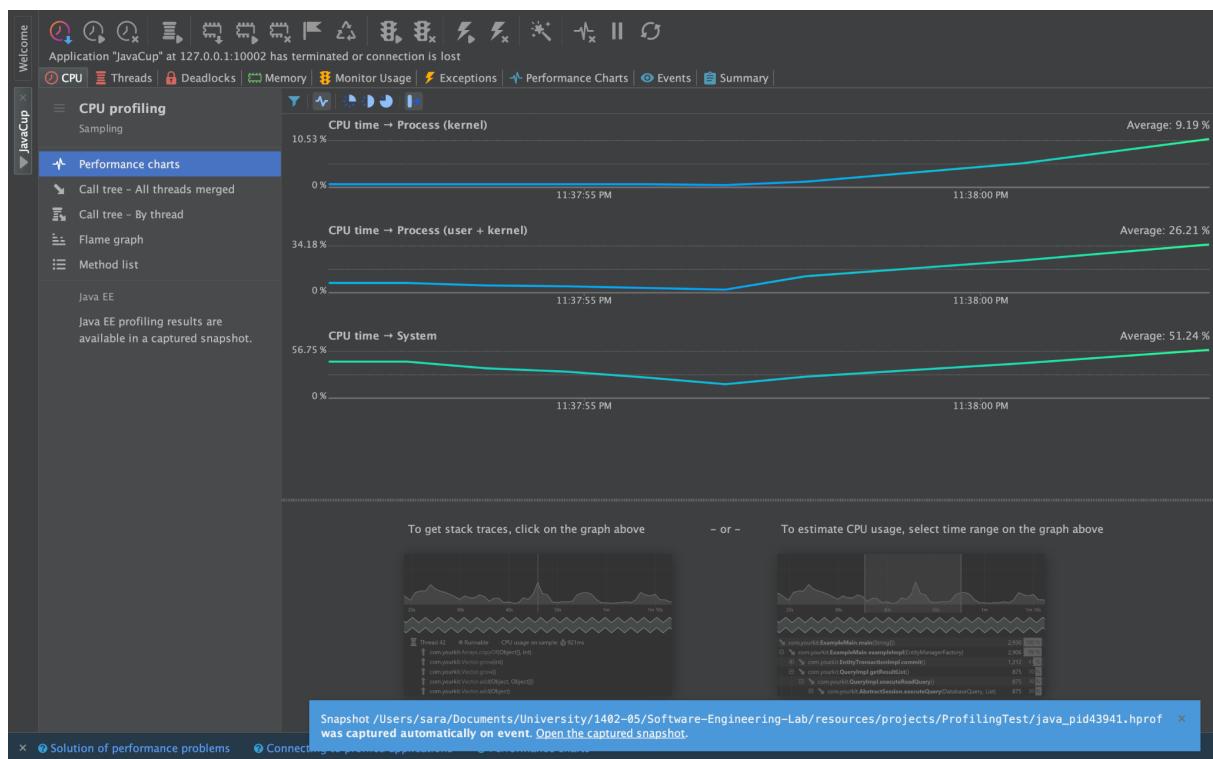


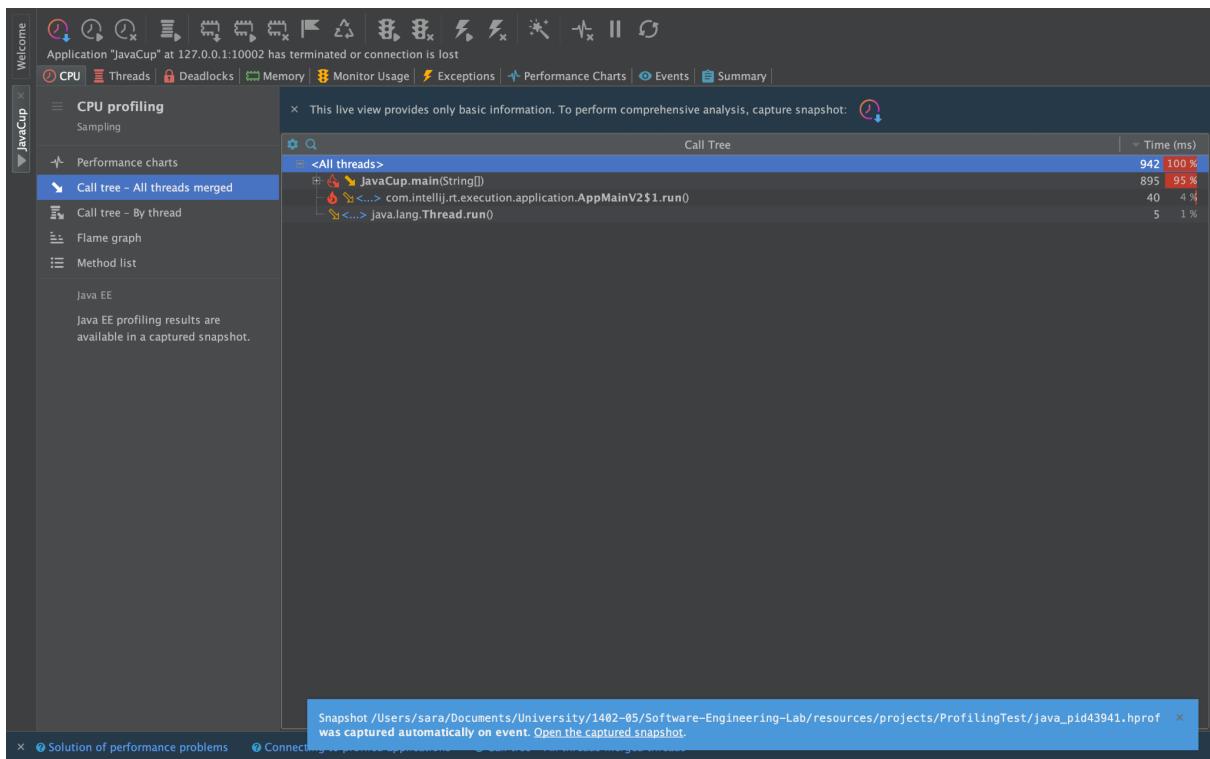
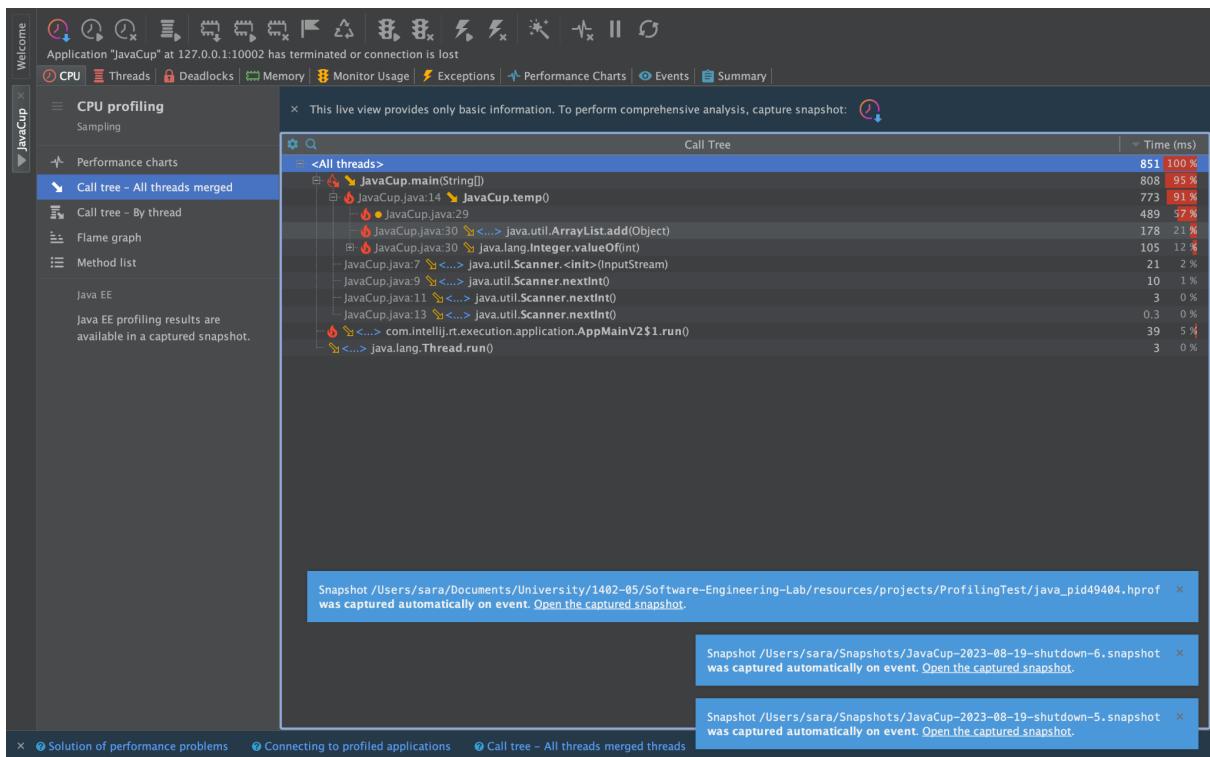


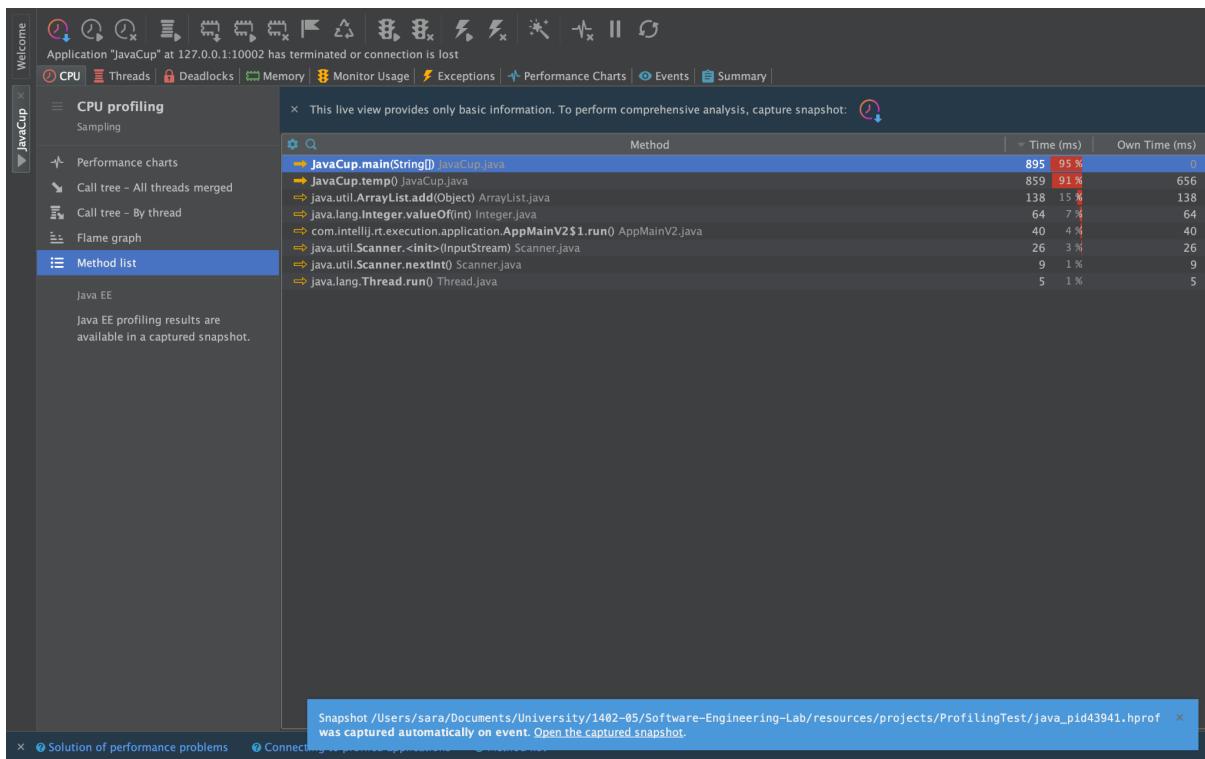
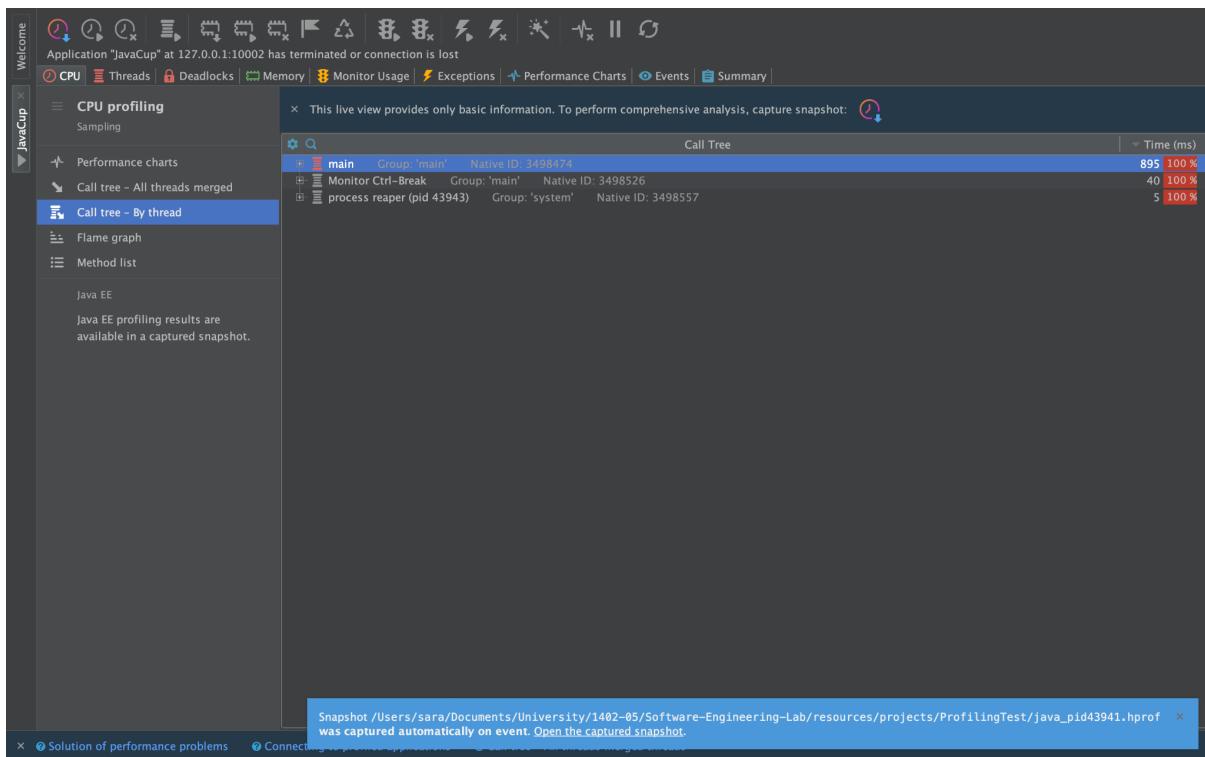


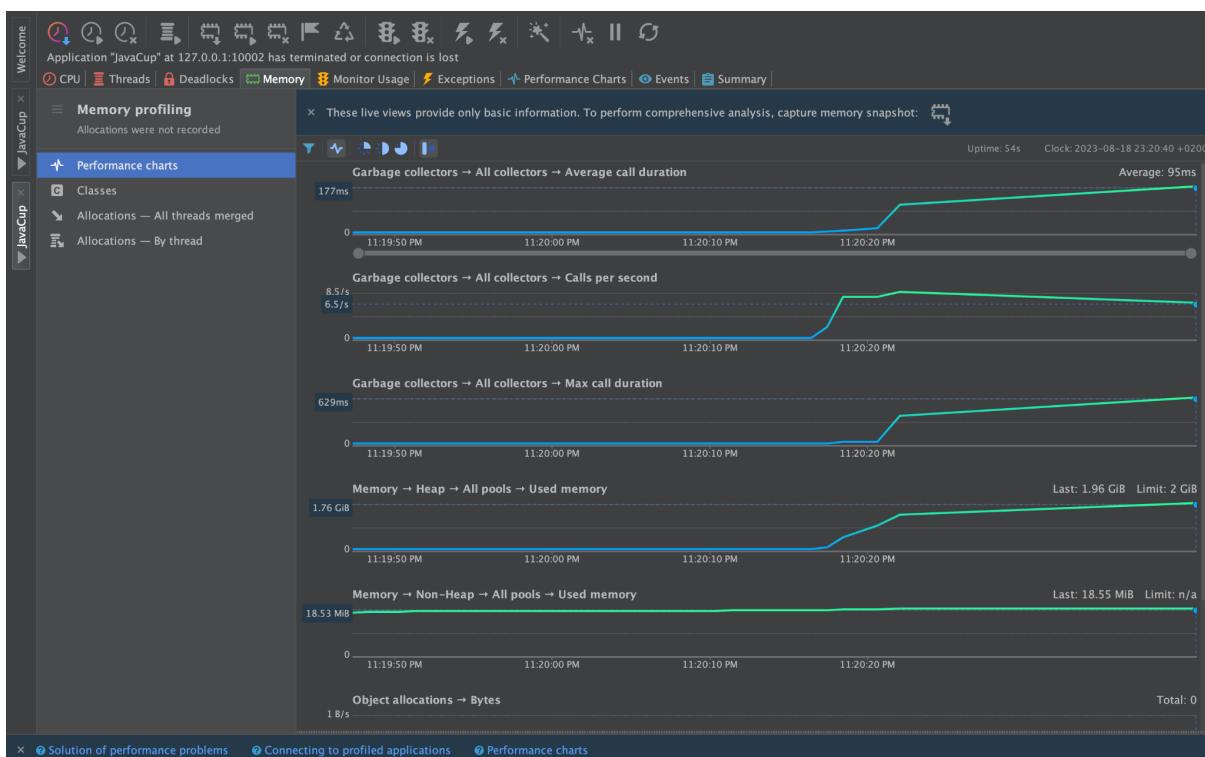
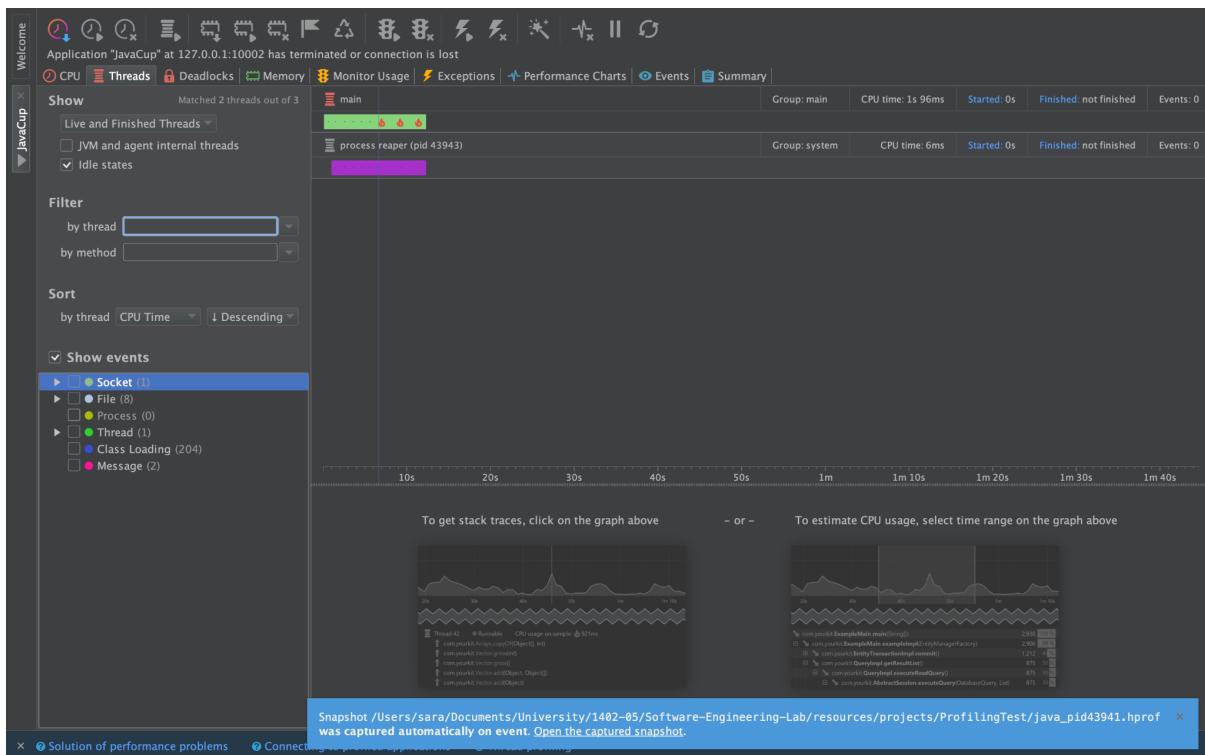
نمودار را از جهات مختلف بررسی میکنیم:

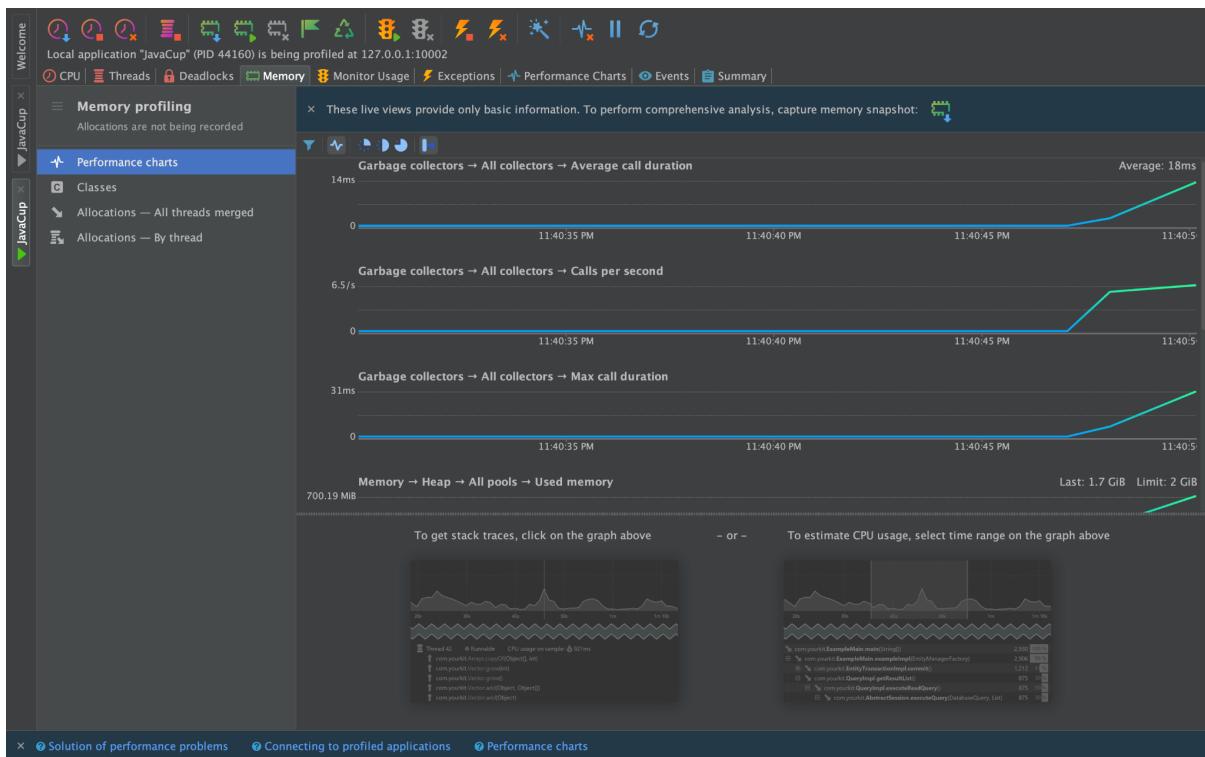




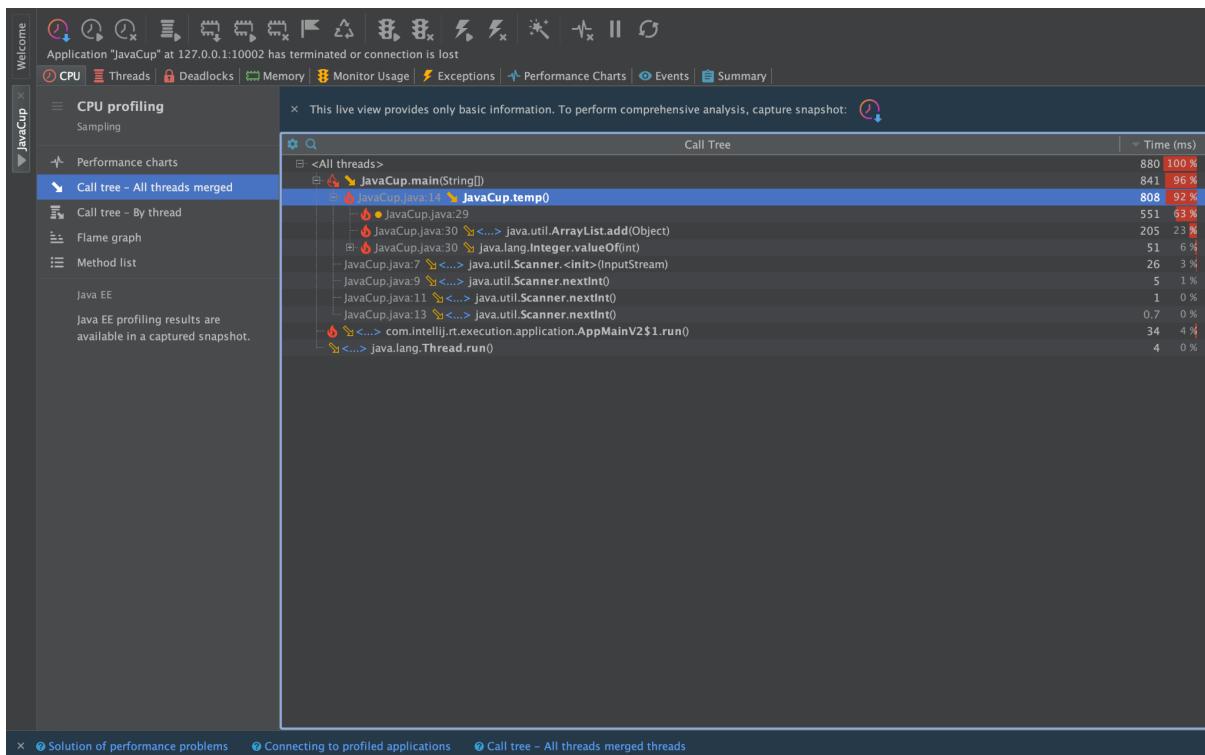








همانطور که در تصویر زیر مشخص است تابع `temp` باعث ایجاد این مشکل شده است که بخارتر اینکه زمان حلقه خیلی زیاد است رم پر میشود و برنامه با مشکل مواجه میشود.



تابع `temp` در ابتدا به شکل زیر است.

PT ProfilingTest ~/Documents/University/1402-05/Software-E...

Project

- ProfilingTest
- .idea
- out
- src
 - A
 - B
 - JavaCup
 - MainSource
 - Utils
 - java_pid41878.hprof
 - java_pid42246.hprof
 - java_pid42436.hprof
 - java_pid43649.hprof
 - java_pid43941.hprof
 - java_pid44160.hprof
 - ProfilingSample.iml
 - ProfilingTest.iml
- External Libraries
- Scratches and Consoles

Profile JavaCup

```

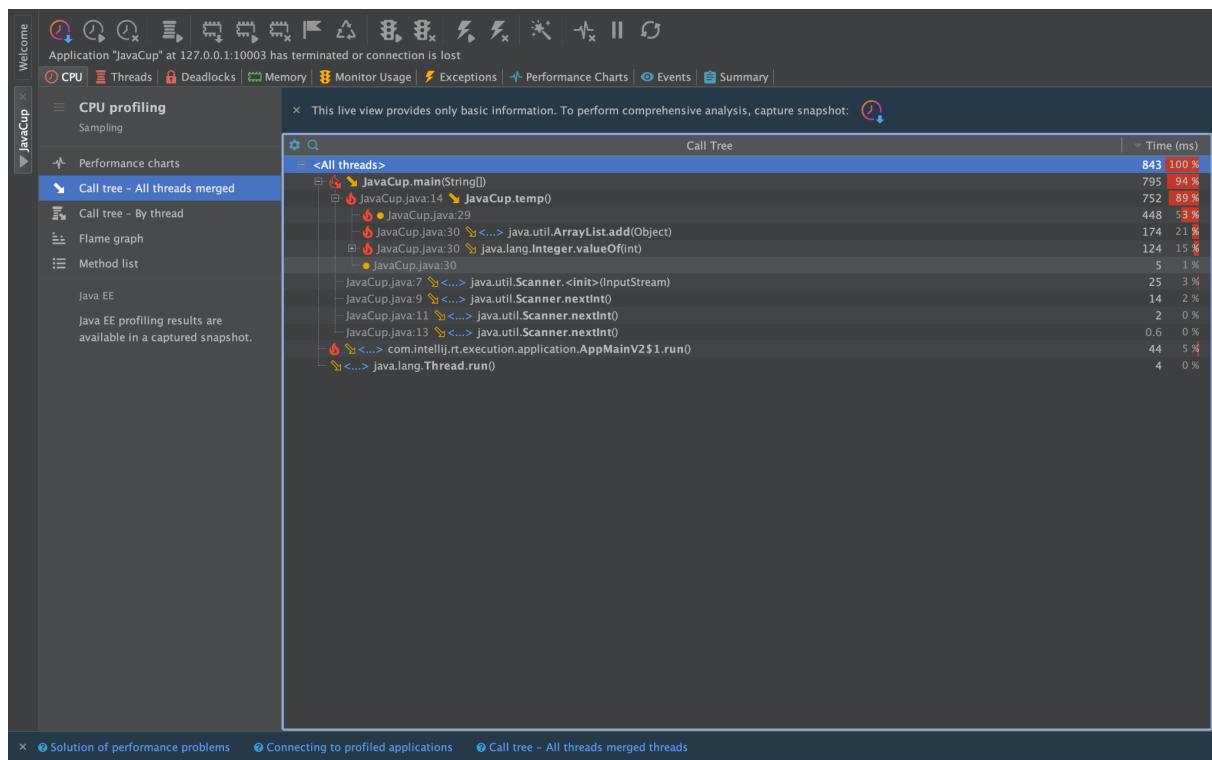
at java.base/java.util.Arrays.copyOf((Arrays.java:3689)
at java.base/java.util.ArrayList.grow(ArrayList.java:238)
at java.base/java.util.ArrayList.grow(ArrayList.java:243)
at java.base/java.util.ArrayList.add(ArrayList.java:484)
at java.base/java.util.ArrayList.add(ArrayList.java:499)
at JavaCup.temp(JavaCup.java:30)
at JavaCup.main(JavaCup.java:14)

```

Process finished with exit code 1

Externally added files can be added to Git
[View Files](#) [Always Add](#) [Don't Ask Again](#)

29:1 CRLF UTF-8 4 spaces



همانطور که در تصویر بالا میبینید بخش زیادی از منابع صرف اضافه کردن یک آبجکت به ArrayList چرا که این به صورت با اضافه کردن هر عنصر ارایه بزرگتر میشود و برای افزایش سایز آرایه مصرف میشود. به عنوان یک راه حل میتوانیم سایز ارایه را ثابت کنیم. در واقع ما داینامیک بودن آرایه را استاتیک کردیم و در میزان مصرف منابع را کاهش دادیم.

تابع temp را به شکل زیر در می اوریم.

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure with a file named "JavaCup.java" selected.
- Code Editor:** Displays the JavaCup.java code. The code defines a static void temp() method that prints "YES" if the size of array a is less than 20000, and "NO" otherwise. It also contains a nested loop that calculates a[i * 20000 + j] = i + j for i, j < 10000.
- Terminal:** Shows the output of running the code. It prompts for three numbers (number1, number2, number3) and then prints "NO".
- Status Bar:** Shows the time as 30:41, encoding as CRLF, character set as UTF-8, and code style as 4 spaces.

```
20     {
21         System.out.println("YES");
22     }
23     else { System.out.println("NO"); }
24 }
1 usage  * Seyed Alireza Hashemi *
public static void temp() {
    int[] a = new int[10000 * 20000];
    for (int i = 0; i < 10000; i++) {
        for (int j = 0; j < 20000; j++) {
            a[i * 20000 + j] = i + j;
        }
    }
}
Press number1:
1
Press number2:
1
Press number3:
1
NO
Process finished with exit code 0
```

Application "JavaCup" at 127.0.0.1:10002 has terminated or connection is lost

CPU | **Threads** | **Deadlocks** | **Memory** | **Monitor Usage** | **Exceptions** | **Performance Charts** | **Events** | **Summary**

CPU profiling

Sampling

Call tree - All threads merged

Call tree - By thread

Flame graph

Method list

Java EE

Java EE profiling results are available in a captured snapshot.

Call Tree

<All threads>

	Time (ms)	%
com.intellij.rt.execution.application.AppMainV2\$1.run0	38	100 %
JavaCup.main(String[])	20	5 %
JavaCup.java9 <...> java.util.Scanner.nextInt()	14	3.7 %
JavaCup.java9 <...> java.lang.Thread.run()	14	3.7 %
	3	8 %

Solution of performance problems | Connecting to profiled applications | Call tree - All threads merged threads

Snapshot /Users/sara/Snapshots/JavaCup-2023-08-19-shutdown-5.snapshot was captured automatically on event. Open the captured snapshot.

Application "JavaCup" at 127.0.0.1:10002 has terminated or connection is lost

CPU | **Threads** | **Deadlocks** | **Memory** | **Monitor Usage** | **Exceptions** | **Performance Charts** | **Events** | **Summary**

CPU profiling

Sampling

Performance charts

Call tree - All threads merged

Call tree - By thread

Flame graph

Method list

Java EE

Java EE profiling results are available in a captured snapshot.

CPU time → Process (kernel)

Average: 0.12 %

CPU time → Process (user + kernel)

Average: 2.85 %

CPU time → System

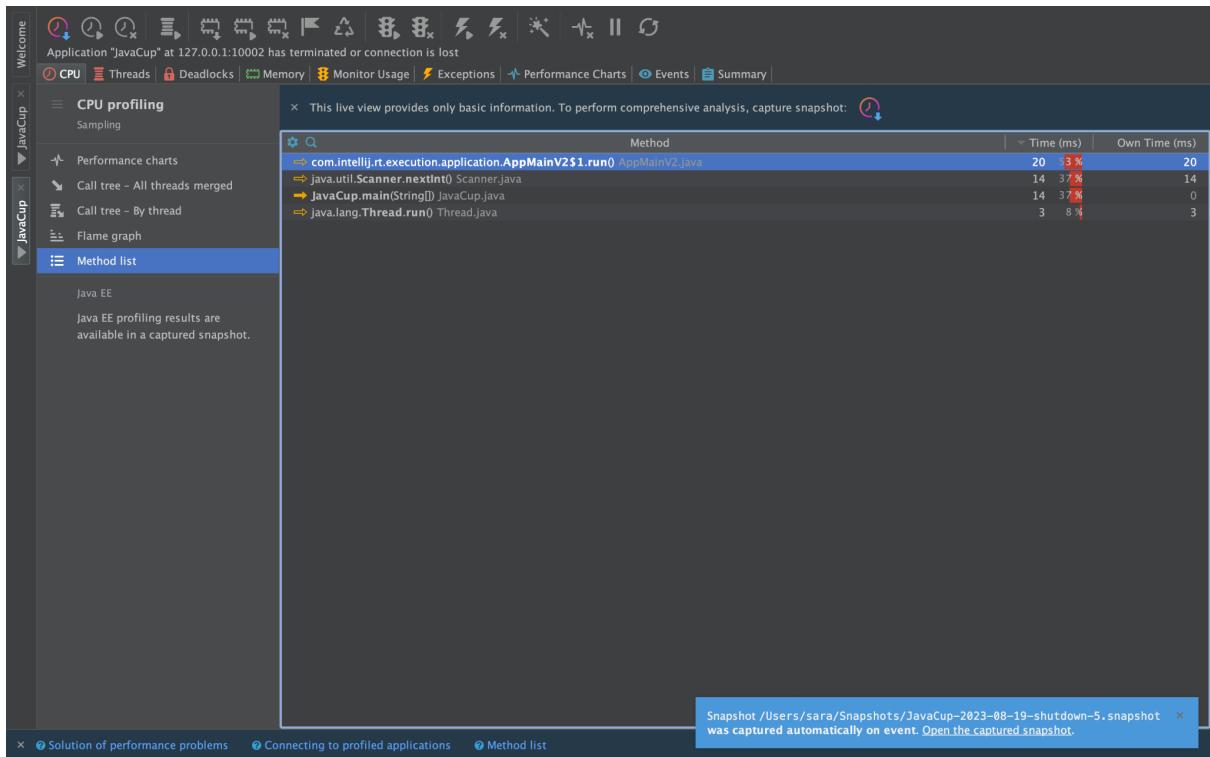
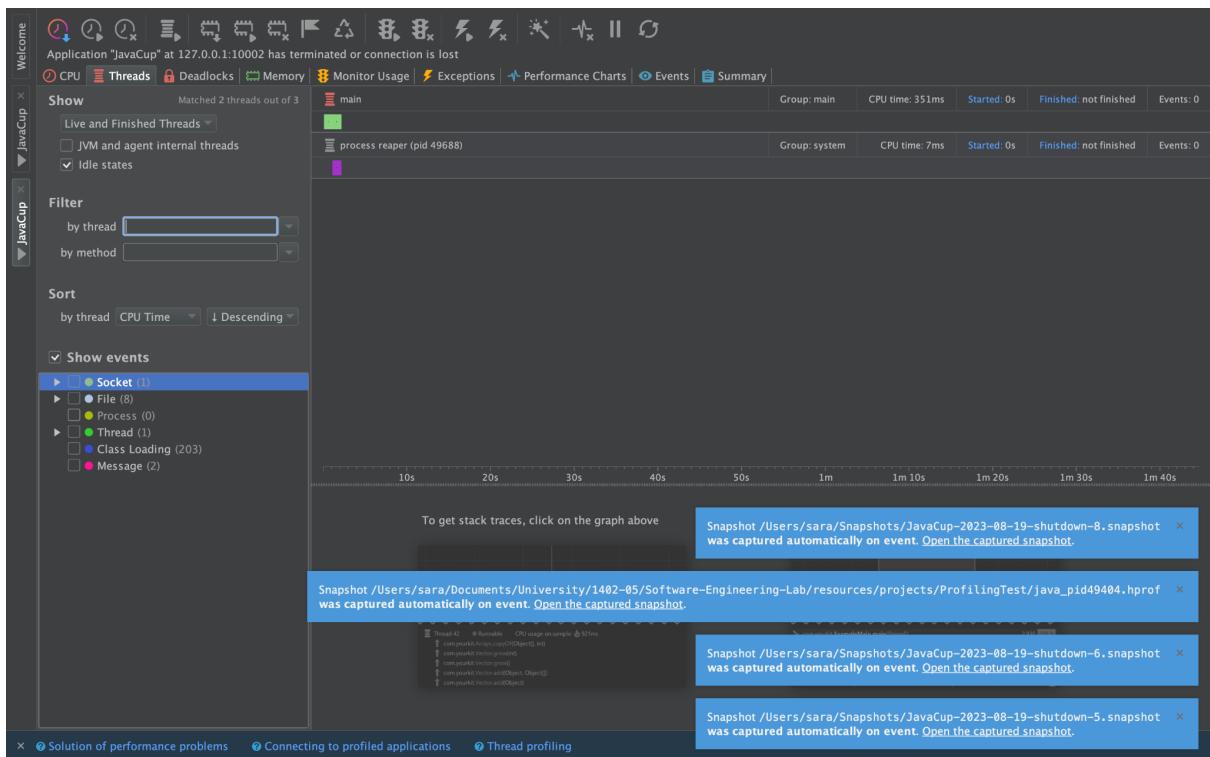
Average: 47.34 %

To get stack traces, click on the graph above

- or -

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

Snapshot /Users/sara/Snapshots/JavaCup-2023-08-19-shutdown-5.snapshot was captured automatically on event. Open the captured snapshot.



همانطور که میبینید مصرف منابع بسیار کاهش یافت و برنامه به اروری بر نمیخورد.

۲- قطعه کد دیگری (به جز الگوریتم‌های مرتب‌سازی) به زبان جاوا بنویسید و سپس عملیات Profiling را با استفاده از Yourkit بر روی آن اجرا و بخشی از کد که بیشترین مصرف منابع را دارد شناسایی کنید. الگوریتم پیاده‌سازی آن قسمت را به گونه‌ای تغییر دهید که این مشکل برطرف شده و مصرف منابع نسبت به حالت فعلی بهتر شود. توجه داشته باشید کامنت کردن کد و یا صرفنظر کردن از اجرای آن و موارد مشابه قابل قبول نیست. همراه با ارسال پروژه لازم است گزارشی در مورد عملکرد کد و نحوه برطرف شدن مشکل در پیاده‌سازی جدید را ارائه نمایید. گزارش باید حاوی تصاویری از وضعیت مصرف کلیه منابع در حالت قبل و بعد از پیاده‌سازی جدید باشد.

در این سوال ما یک کد ساده‌ای نوشته‌یم که یک عدد بزرگتر از ۱۰۰۰۰۰ را از ورودی بخواند و سپس یک عدد زیر ۲۱ را از ورودی بخواند. برای عدد اولی که از ورودی گرفتیم میخواهیم جمع تمام اعداد از ۱ تا آن عدد را حساب کنیم و برای عدد دوم میخواهیم فاکتوریل عدد را محاسبه کنیم. ما در ادامه در اجرای کد‌ها عدد اول را ۵۰۰۰۰۰ و عدد دوم را ۲۰ وارد میکنیم. در زیر پیاده‌سازی کد را مشاهده میکنید.

```
import java.util.Scanner;

public class SumAlgorithm {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter an integer number larger than
1000000:");
        long n = scanner.nextInt();

        System.out.println("Enter an integer number less than
21:");
        int m = scanner.nextInt();

        long result = sum(n);

        long factorial = calculateFactorial(m);

        System.out.println("Sum of numbers up to " + n + " is: " +
result);
        System.out.println("Factorial of " + m + " is equal to " +
factorial);
    }
}
```

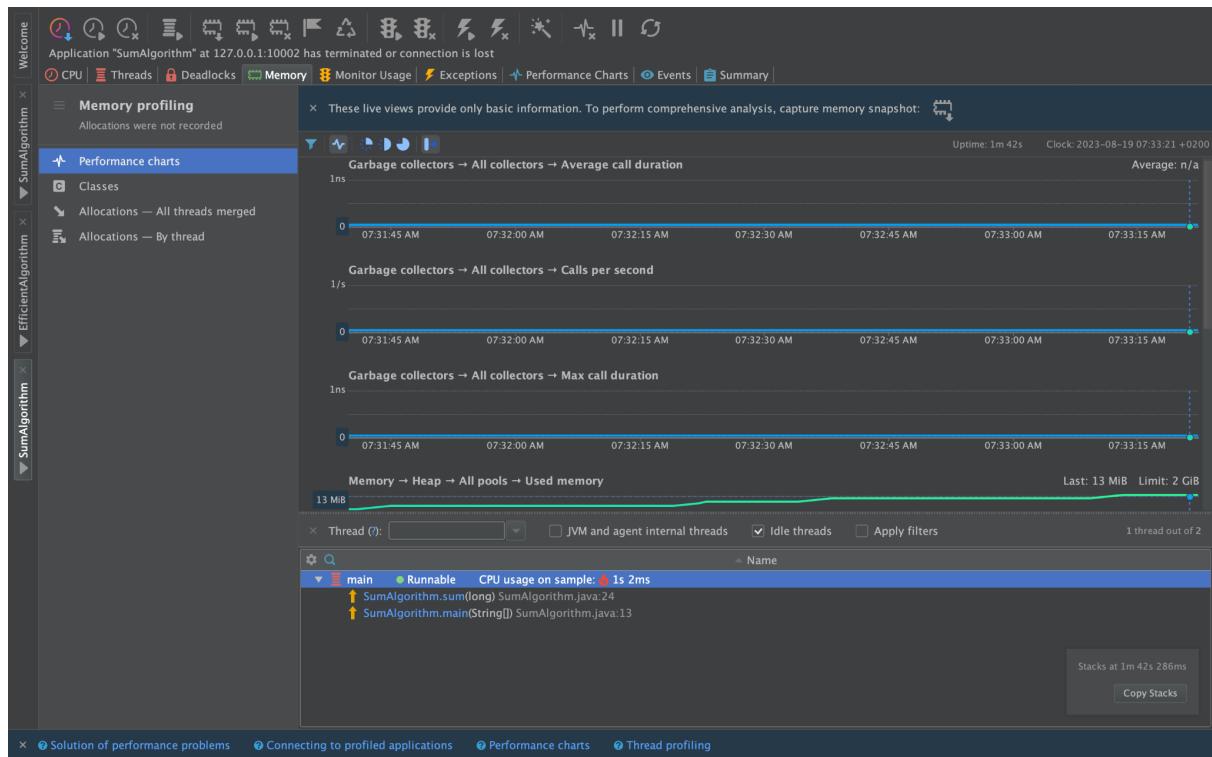
```

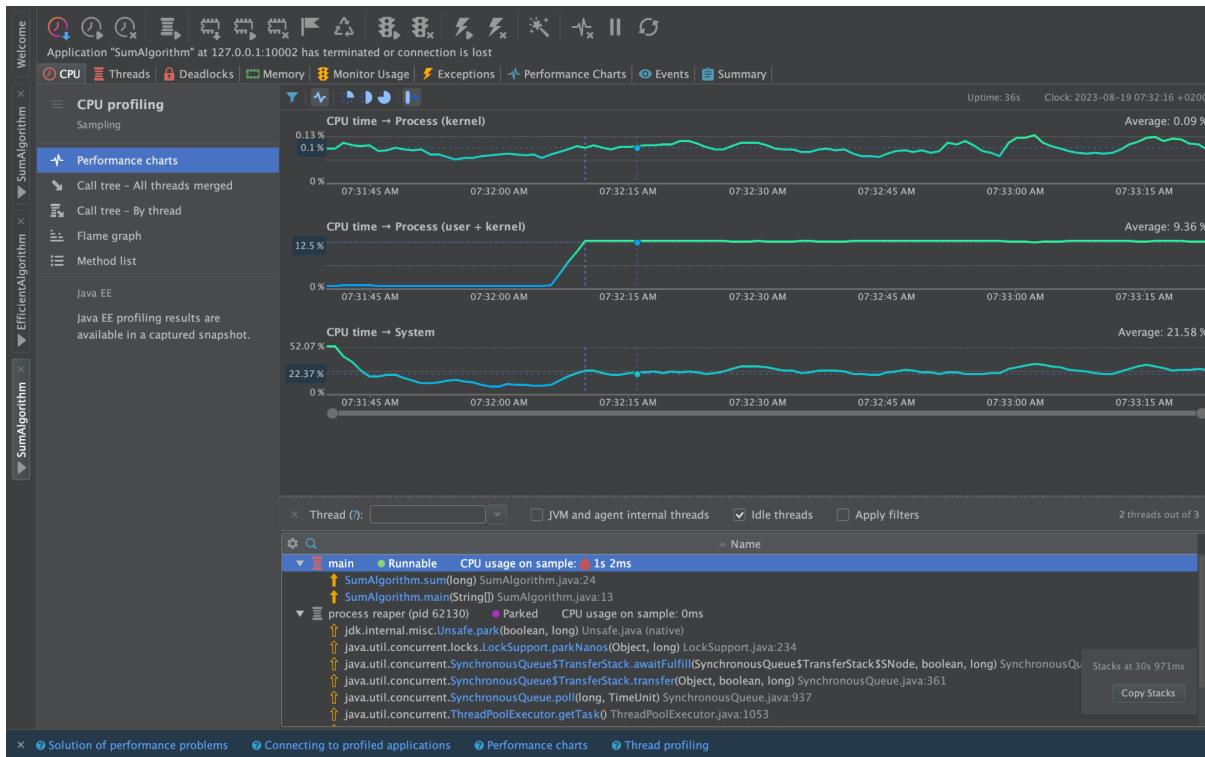
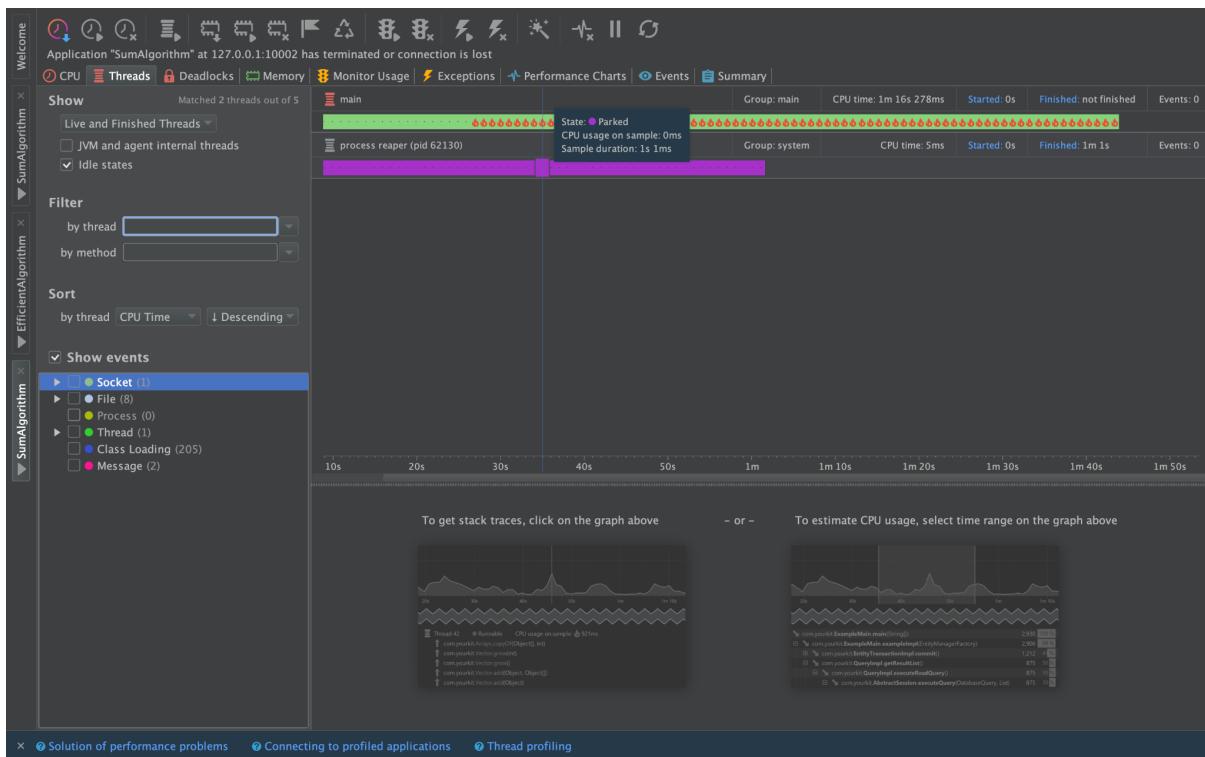
static long sum(long n) {
    long sum = 0;
    for (long i = 1; i <= n; i++) {
        for (long j = 1; j <= n; j++) {
            sum += i * j;
        }
    }
    return sum;
}

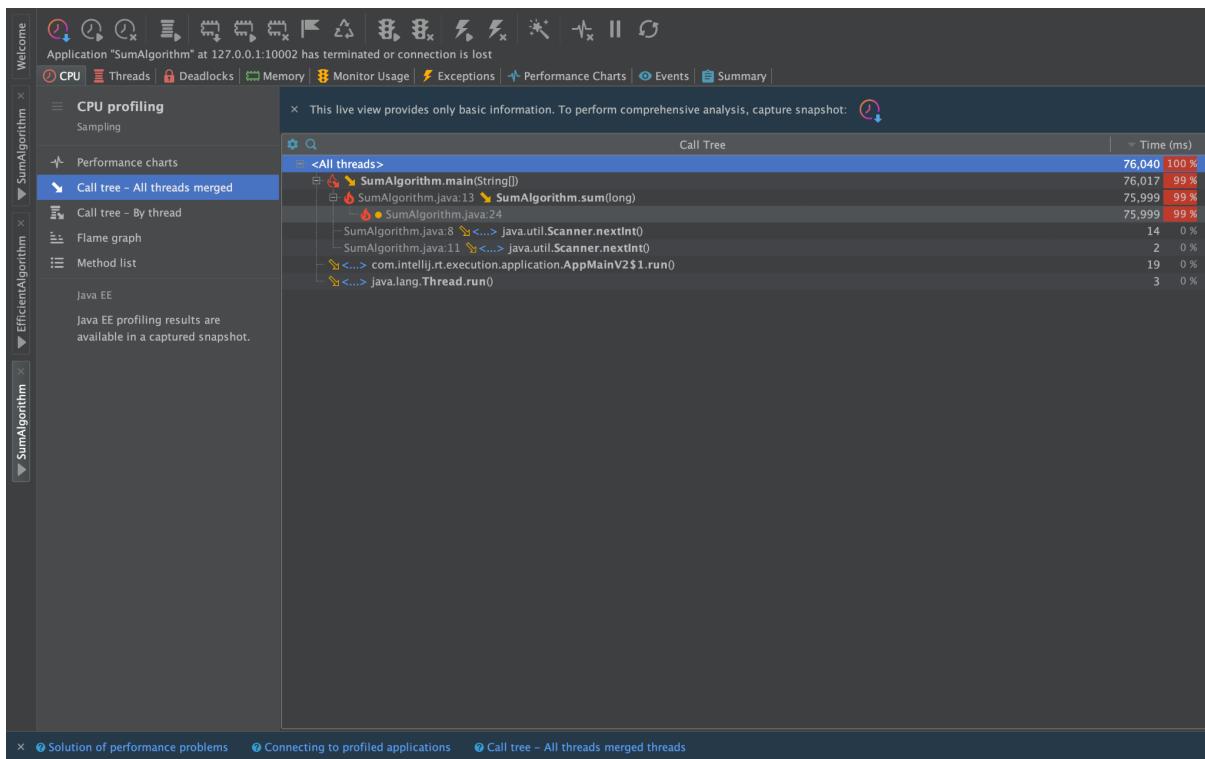
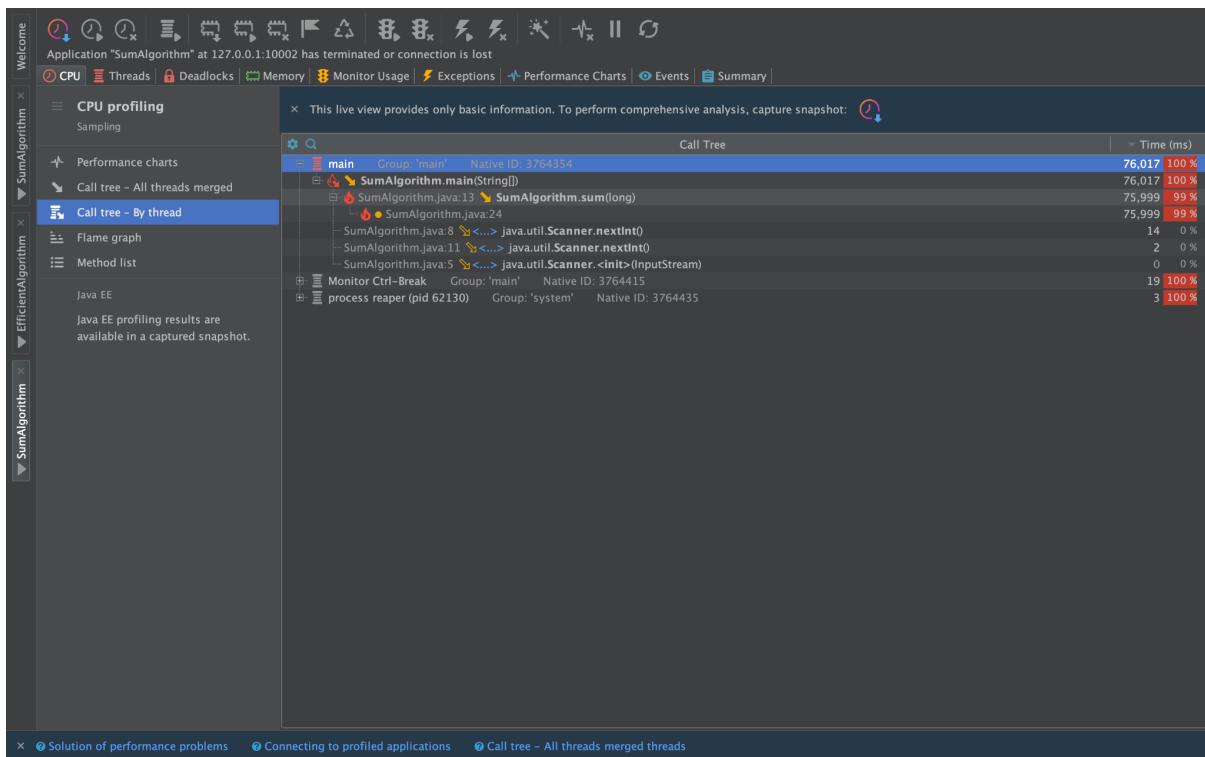
static long calculateFactorial(int n) {
    if (n == 0 || n == 1) {
        return 1;
    }
    return n * calculateFactorial(n - 1);
}

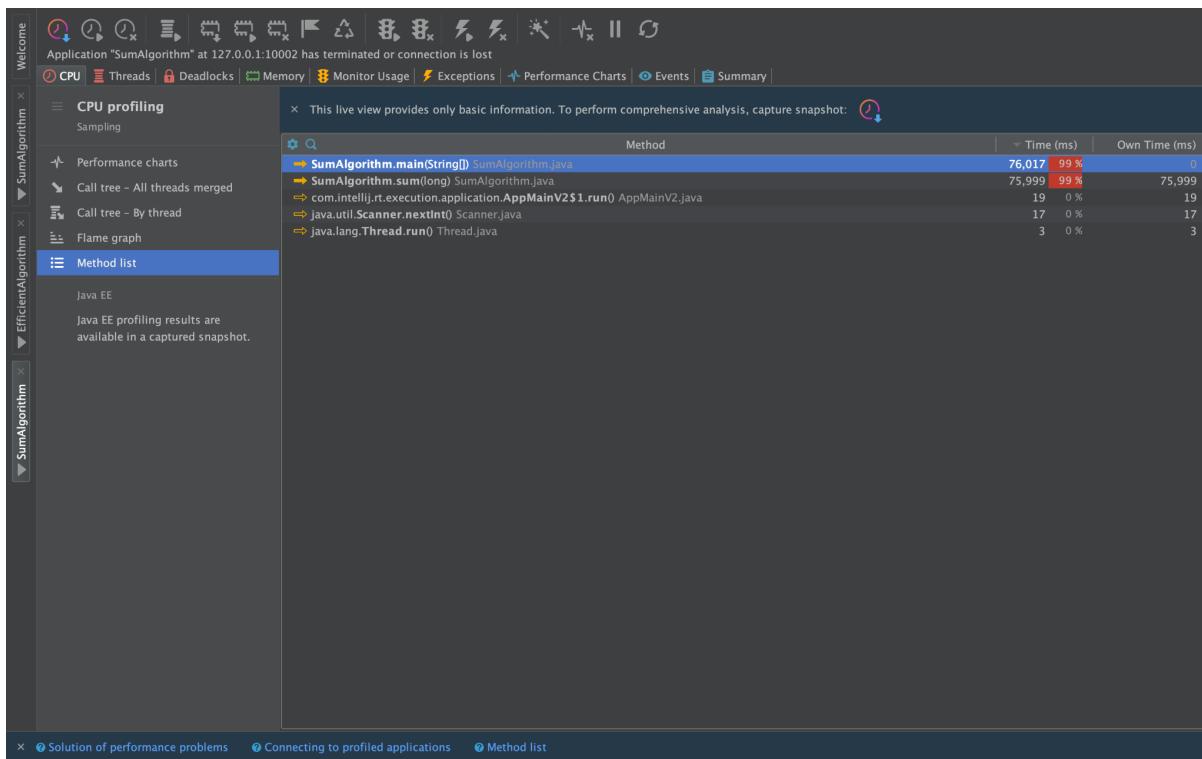
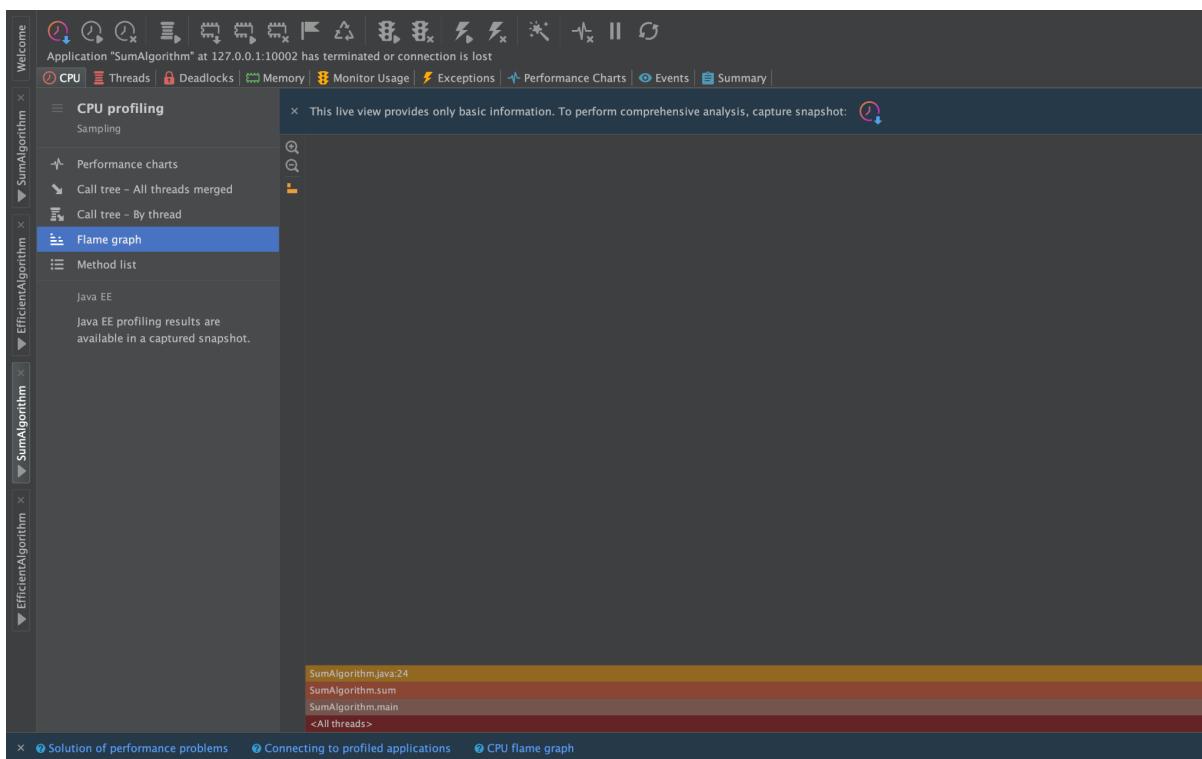
```

حال برای این برنامه profiling را اجرا میکنیم و آن را بررسی میکنیم.
وضعیت حافظه را در طول اجرا مشاهده میکنید.







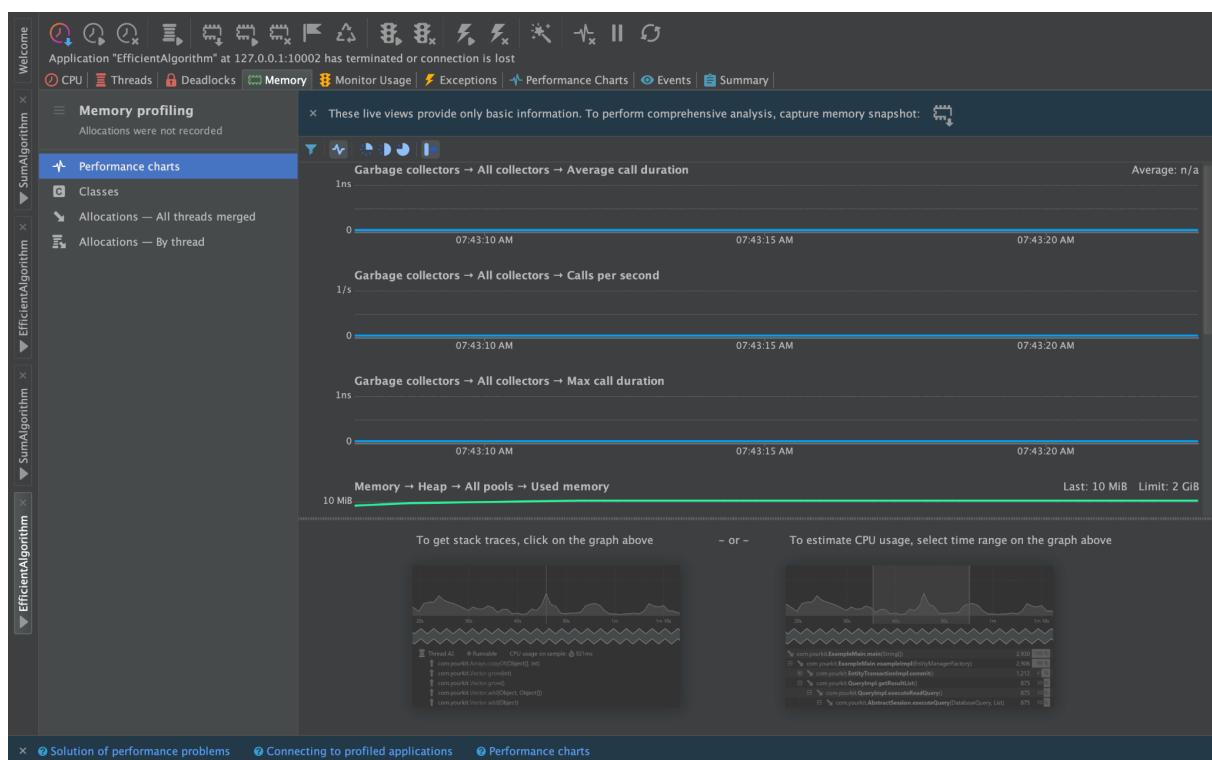


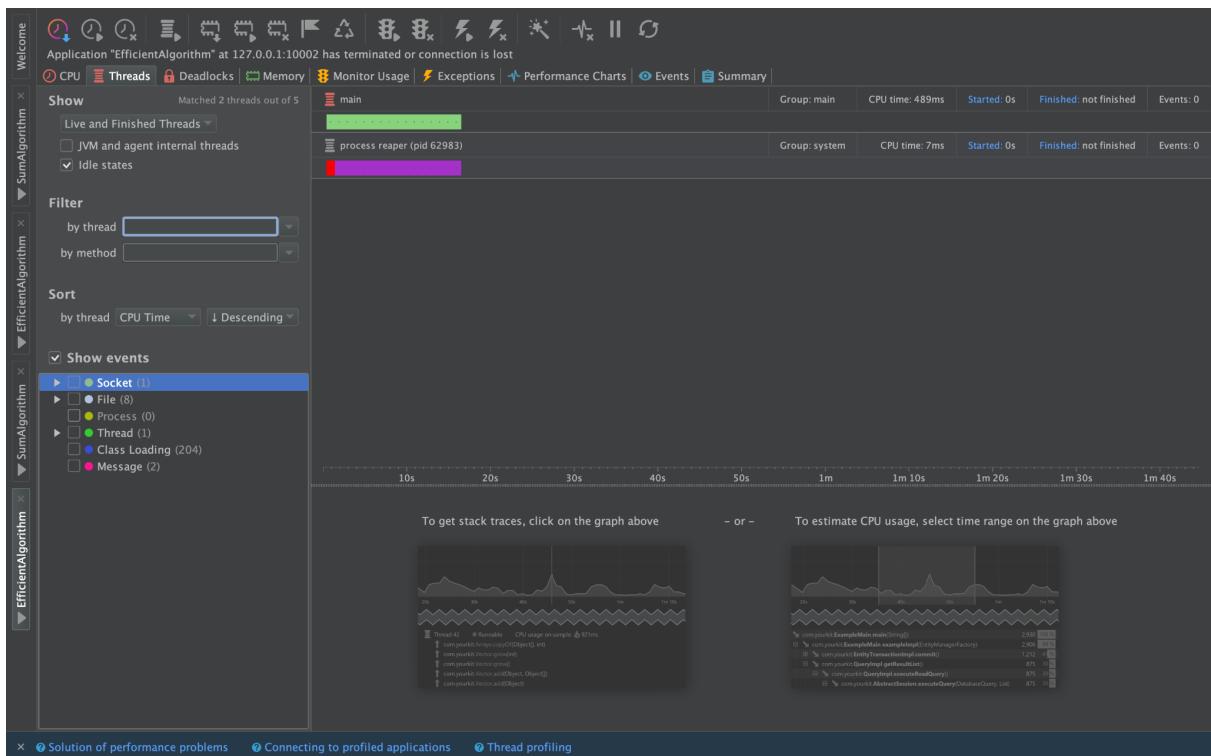
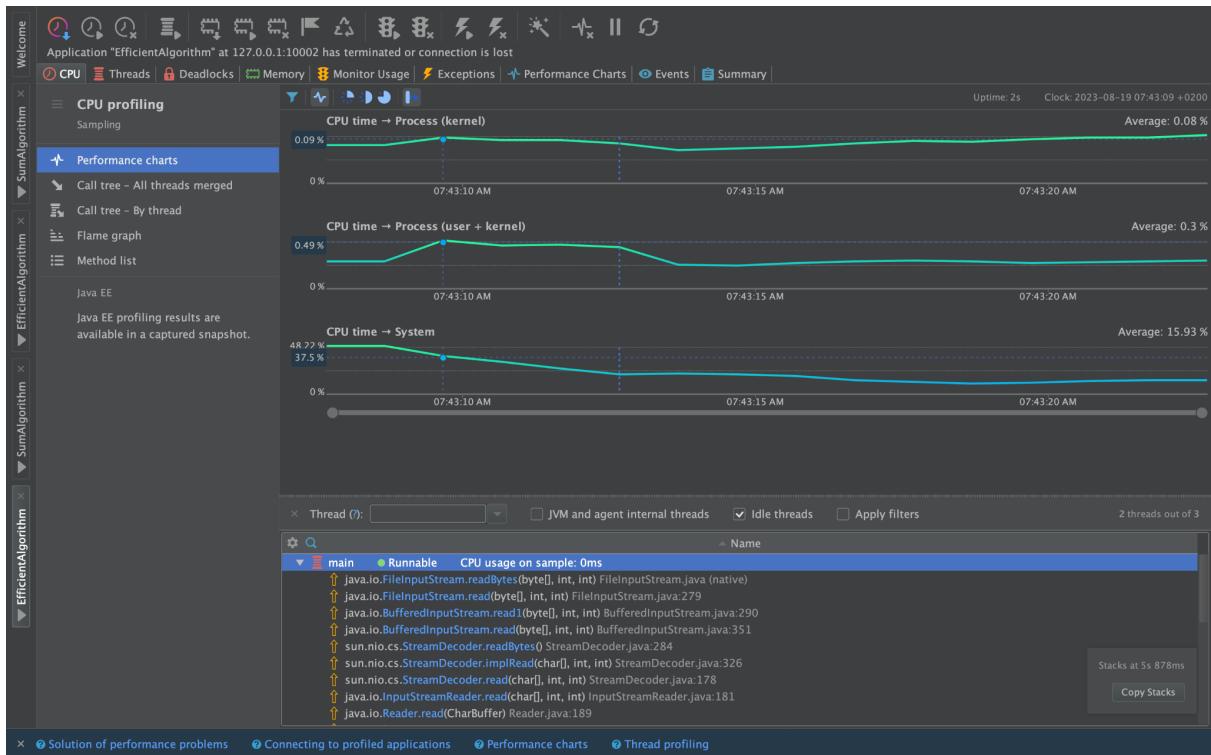
همانطور که در تصاویر مشاهده میکنید تابع `sum` منابع زیادی مصرف میکند. علت آن است که این تابع الگوریتمی دارد که اردر اجرای آن $O(n^2)$ است و این باعث میشود که مقدار زیادی CPU مصرف کند و زمان زیادی بگیرد. علت آن است که از ۲ حلقه‌ی تو در تو با اندازه بزرگ استفاده کردیم که باعث مصرف زیاد CPU شده است.

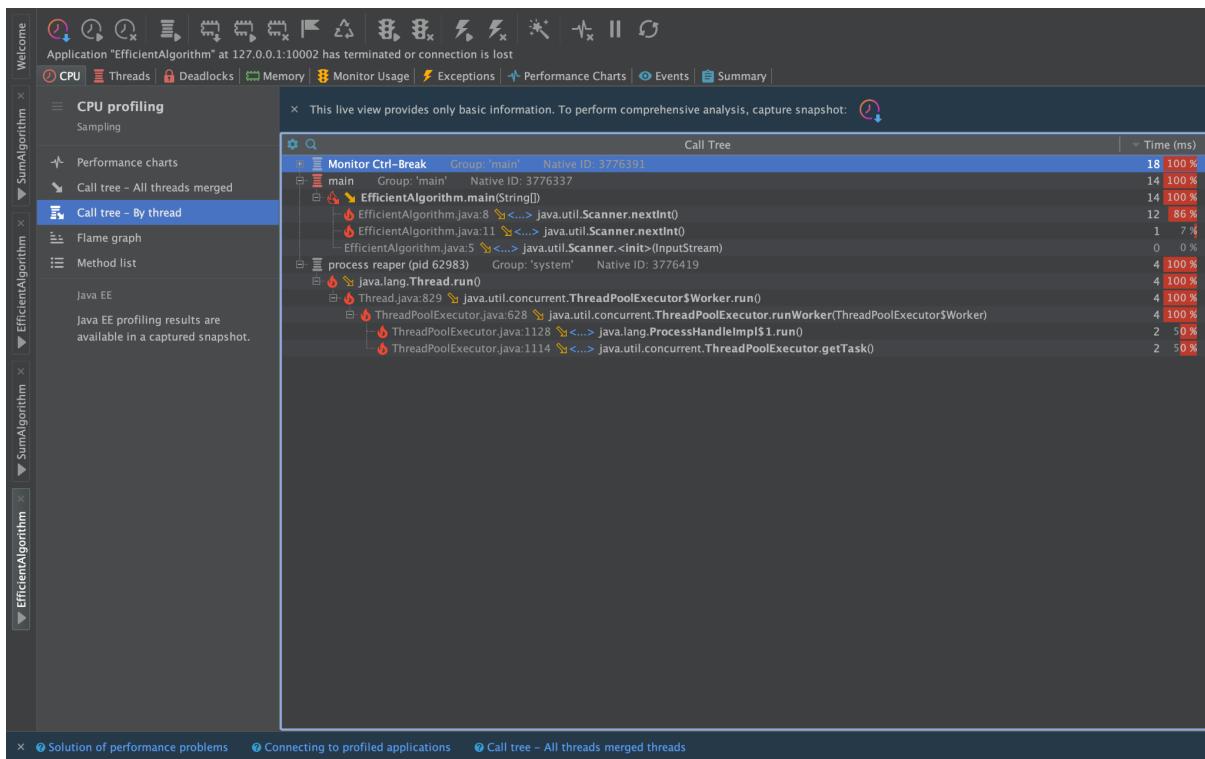
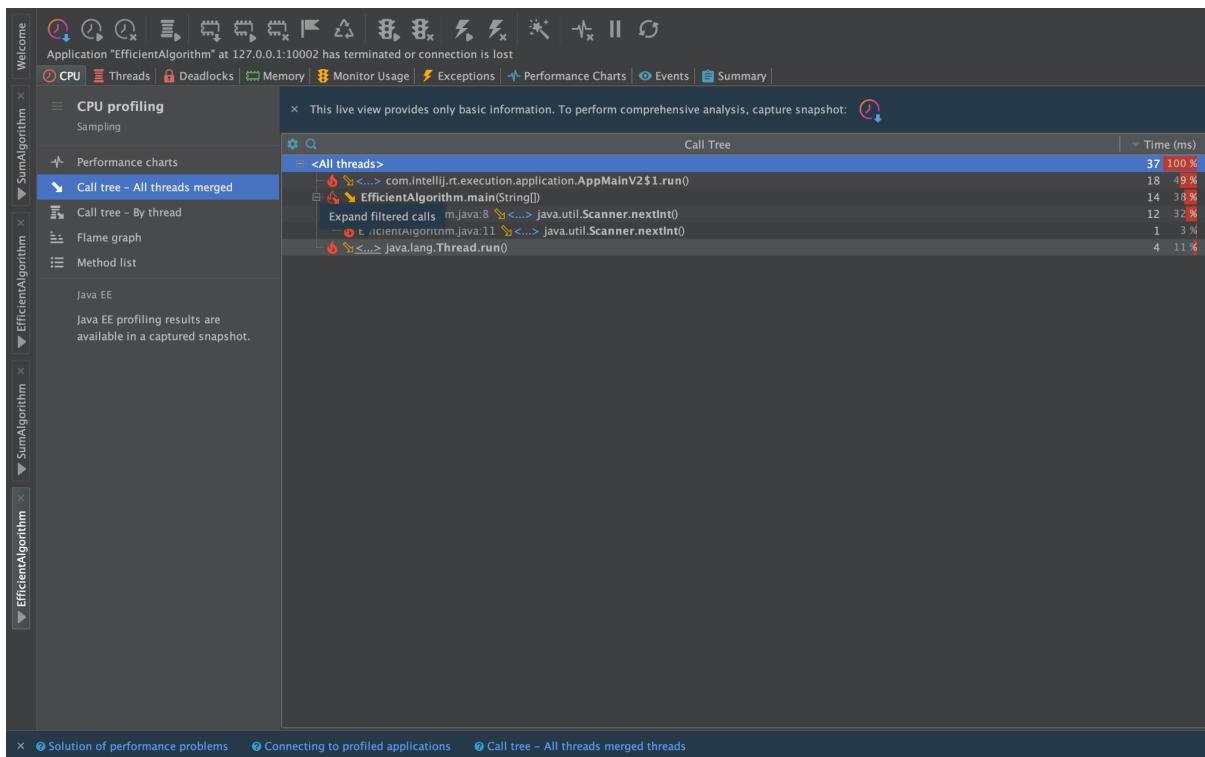
با تغییر تابع `sum` به شکل زیر از شرحله های بزرگ خلاص میشویم و مصرف منابع را کاهش میدهیم.

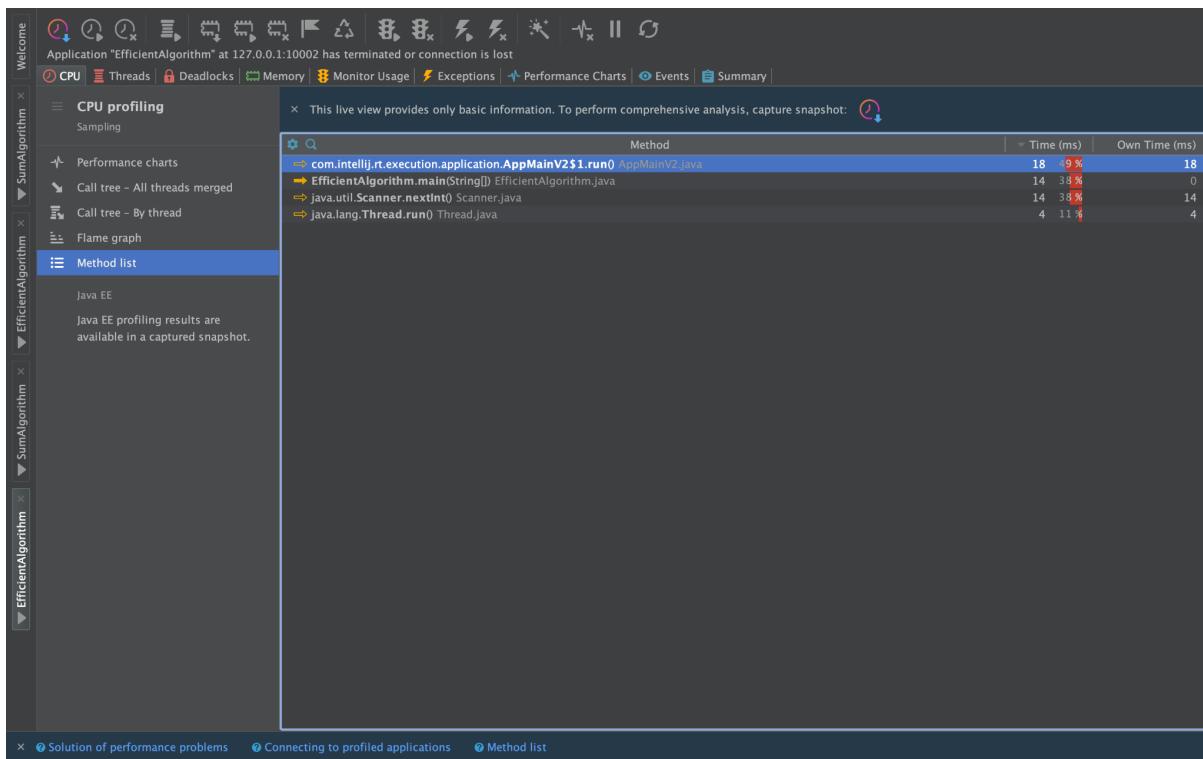
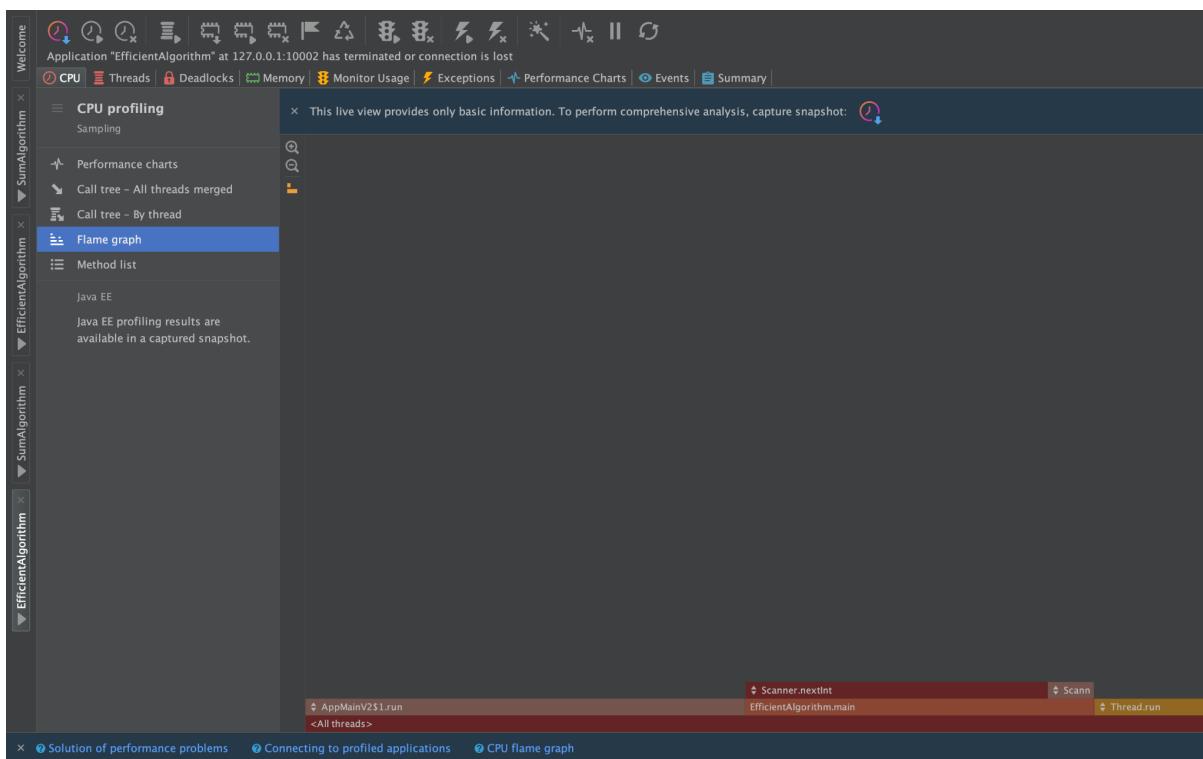
```
static long sum(long n) {
    return n * (n + 1) / 2;
}
```

حال پروفایلینگ را روی این برنامه اجرا میکنیم و نتیجه را در تصاویر زیر میبینیم.









همانطور که مشاهده میکنید توانستیم الگوریتم را طوری تغییر دهیم که مصرف منابع کاهش یابد.