 **گزارش کار آزمایش اول آزمایشگاه مهندسی نرم‌افزار**

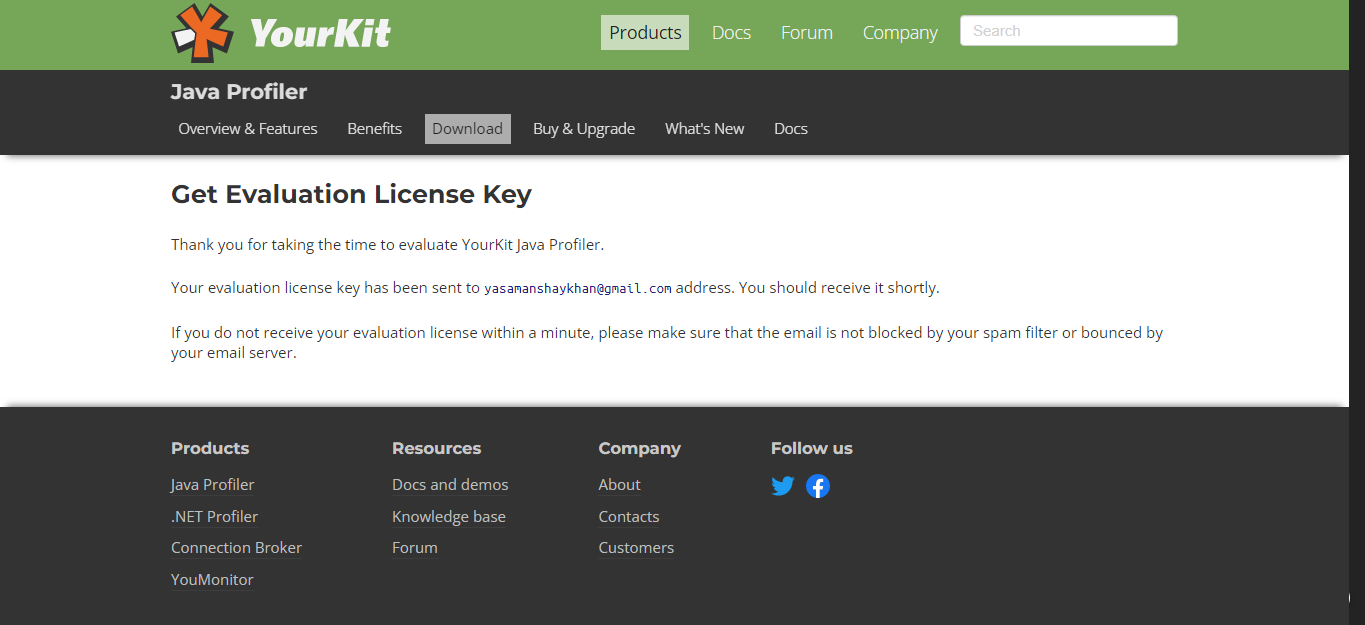
**دانشکده مهندسی کامپیوتر**

**لینک گیتهاب:** [**https://github.com/yasmansh/SE\_LAB**](https://github.com/yasmansh/SE_LAB) **و** اعضای گروه:

یاسمن شیخان - 97101915

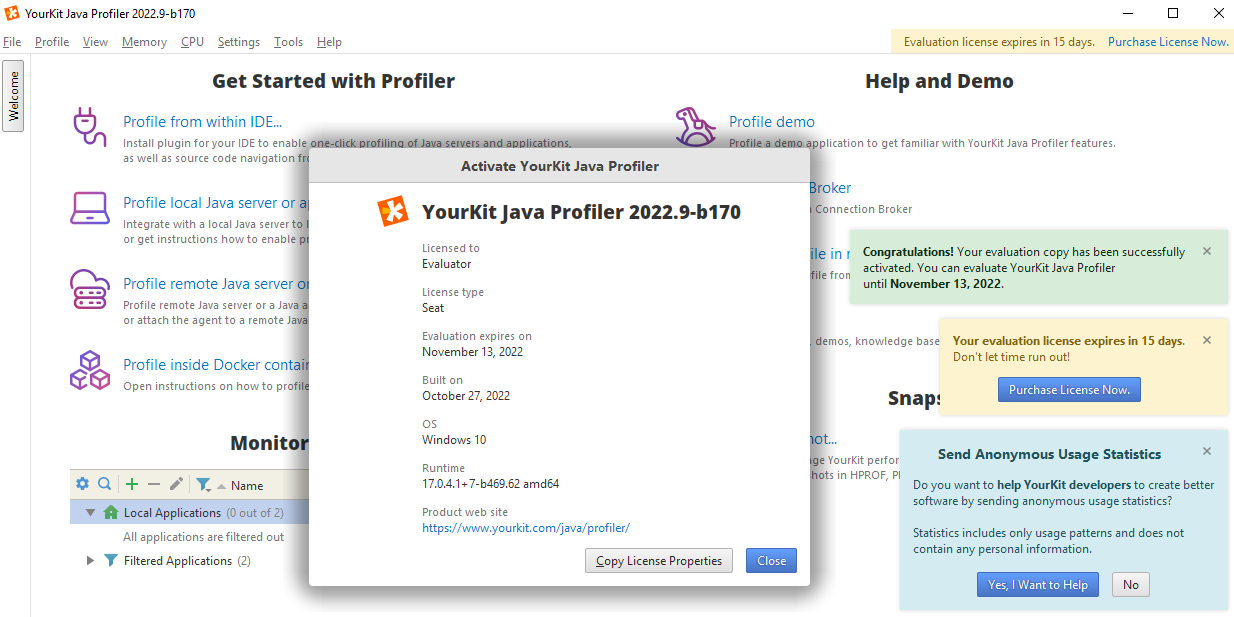
امیرحسین علی‌محمدی - 97110166

**ابتدا به بررسی تصاویری از مراحل نصب برنامه YourKit-JavaProfiler می‌پردازیم. با مراجعه به وبسایت** [**https://www.yourkit.com/download**](https://www.yourkit.com/download)**، آخرین نسخه مطابق با سیستم عامل(در اینجا ویندوز) را دانلود کرده و سپس آن را اجرا می‌کنیم. برای دریافت مجوز 15 روزه ایمیل خود را وارد و lisence key را دریافت می‌کنیم.**



شکل 1 – دریافت کلید مجوز 15روزه از طریق ایمیل

**سپس کلید ارسال شده را در برنامه وارد کرده و مجوز استفاده از آن فعال می‌شود.**

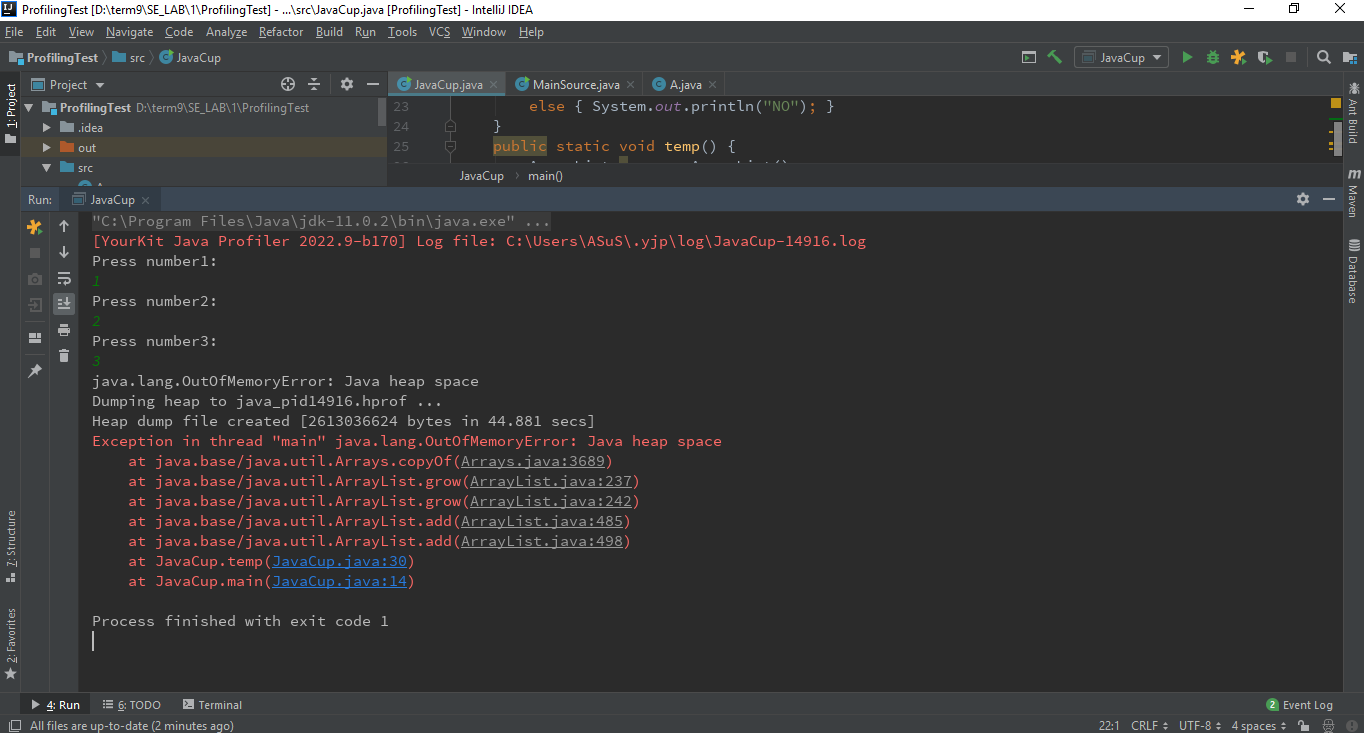


شکل 2 - تصویری از محیط برنامه پس از فعال‌سازی

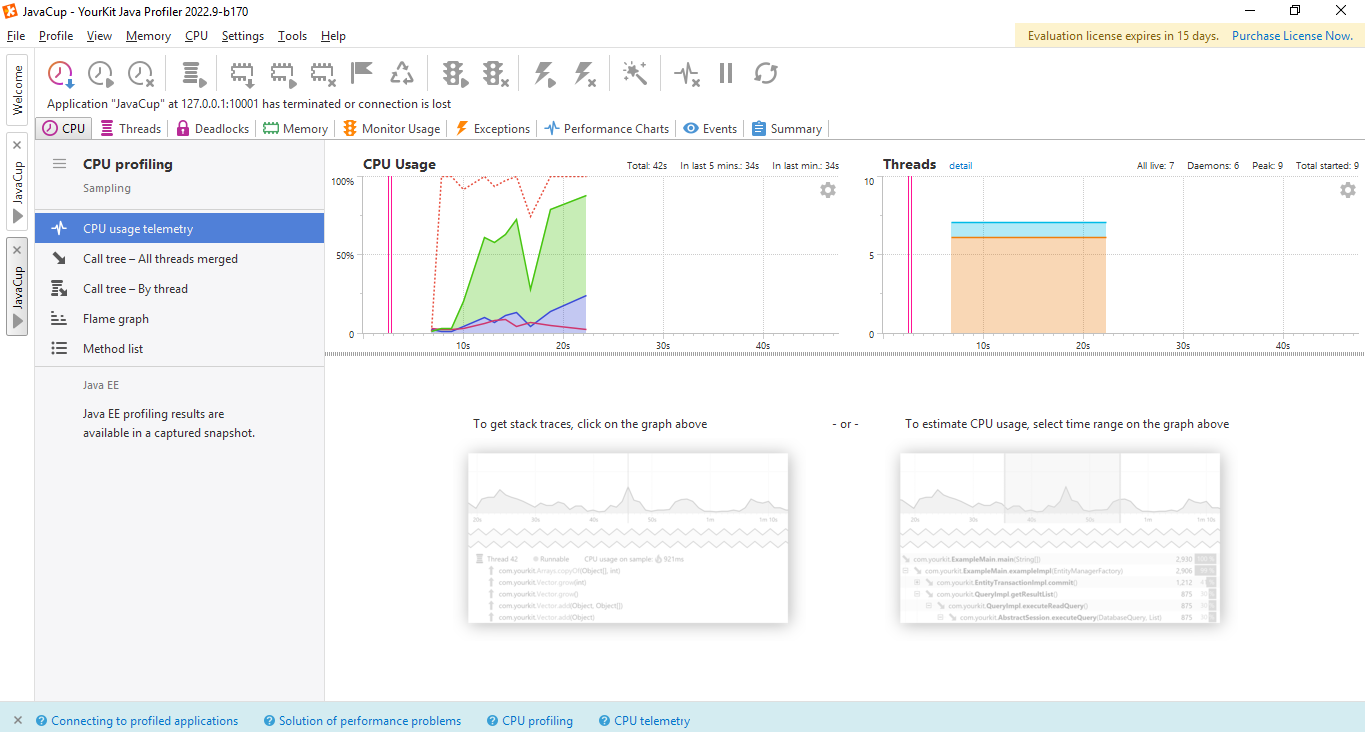
**در ادامه به بررسی هر دو بخش آزمایش می‌پردازیم.**

**بخش اول.**

**در پروژه ProfilingTest، عملیات Profiling را بر روی کلاس JavaCup اجرا کرده و با استفاده از YourKit به بررسی مصرف منابع، تحلیل حافظه و CPU و Threadها خواهیم پرداخت.**

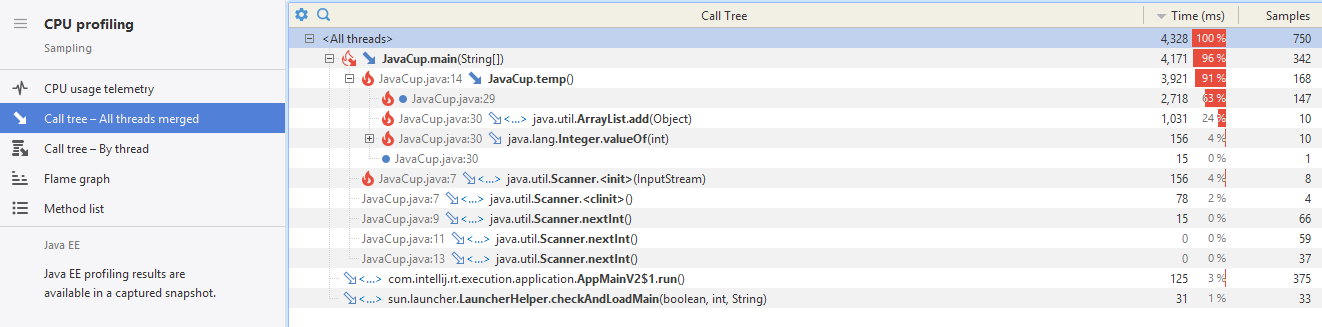


شکل 3 - خروجی برنامه بعد از اجرای عملیات profiling (قبل از بهبود)

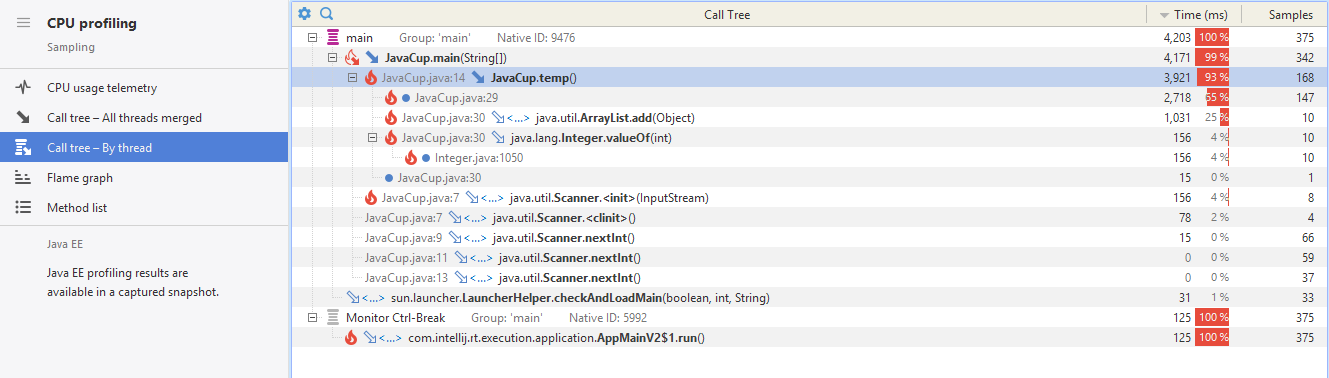


شکل 4 - نمودار استفاده از پردازنده (قبل از بهبود)

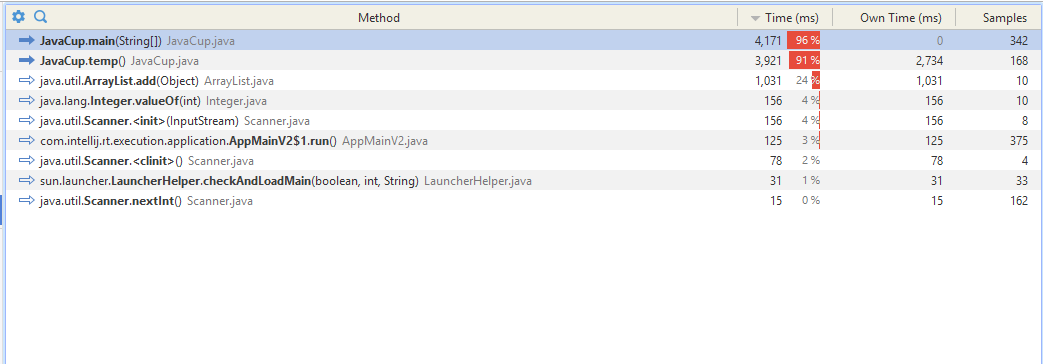
**طبق نمودار مربوط به threadها و زمان اجرای توابع مشاهده می‌شود 91درصد زمان اجرایی توسط تابع temp مصرف می‌شود.**



شکل 5- نمودار درختی مربوط به thread ها(قبل از بهبود)



شکل 6 - نمودار درختی thread ها و زمان اجرای توابع(قبل از بهبود)



شکل 7- نمودار زمان مصرفی و درصد زمانی مربوط به توابع(قبل از بهبود)

**همچنین طبق نمودارهای مربوط به حافظه مصرفی داریم:**

A picture containing graphical user interface

Description automatically generated

شکل 8- نمودار حافظه مصرفی (قبل از بهبود)



شکل 9- نمودار کارایی پردازنده و حافظه و threadها (قبل از بهبود)

**طبق بررسی نمودارهای بالا و حجم بالای مصرف منابع و زمان اجرایی توسط تابع temp، به بررسی دقیقتر آن میپردازیم.**

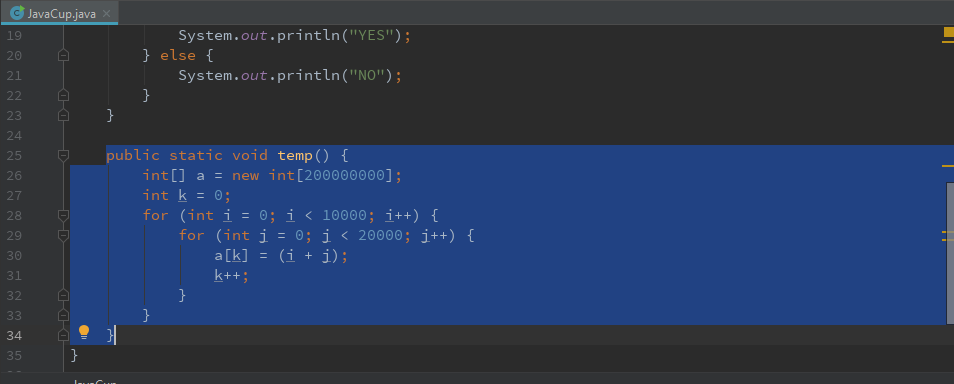
Text

Description automatically generated

شکل 10- تابع temp ( قبل از بهبود)

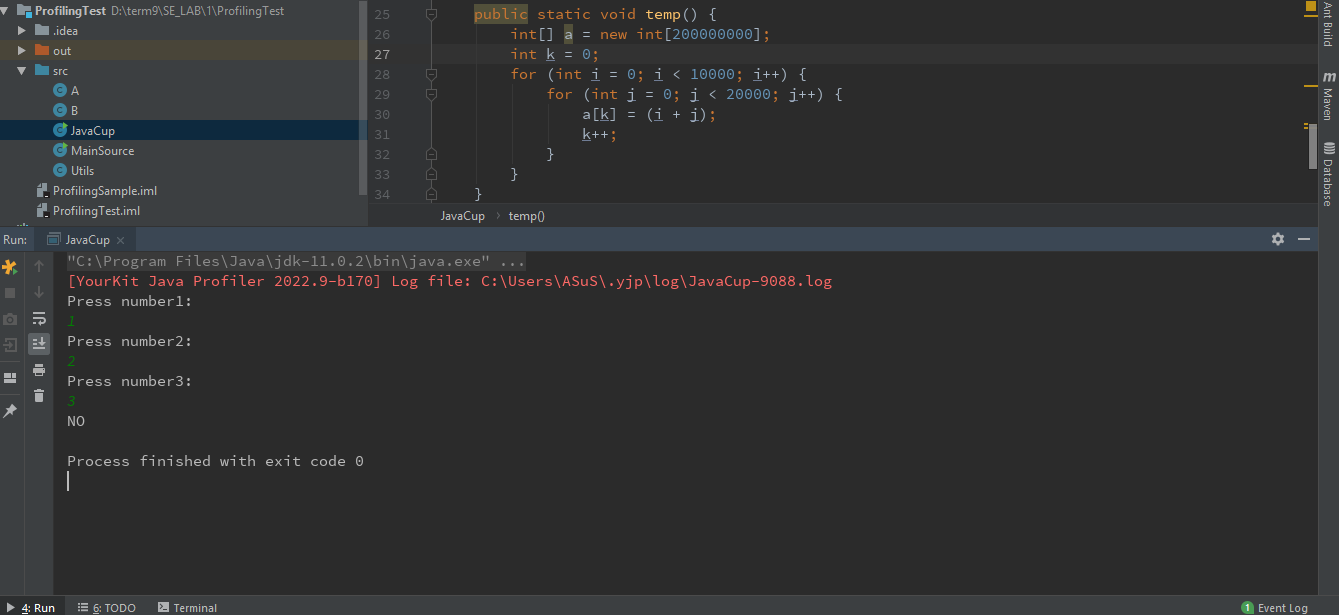
**در این تابع، از ArrayList جاوا استفاده شده و 10000 \* 20000 مرتبه به آن عضوی ر اضافه می‌کند. این داده ساختار به صورت فضای پویا پیاده‌سازی شده و هربار که فضای آن تکمیل شود، سایز خود را دوبرابر می‌کند. به همین علت منابع و زمان اجرایی بالایی را مصرف کرده و heap پر می‌شود(خطای پس از اجرای برنامه نیز به همین علت بود).**

**برای بهبود و جلوگیری از این مشکل، میتوان از داده ساختار لیست با طول مشخص استفاده کنیم و مجددا عملیات profiling را اجرا می‌کنیم.**

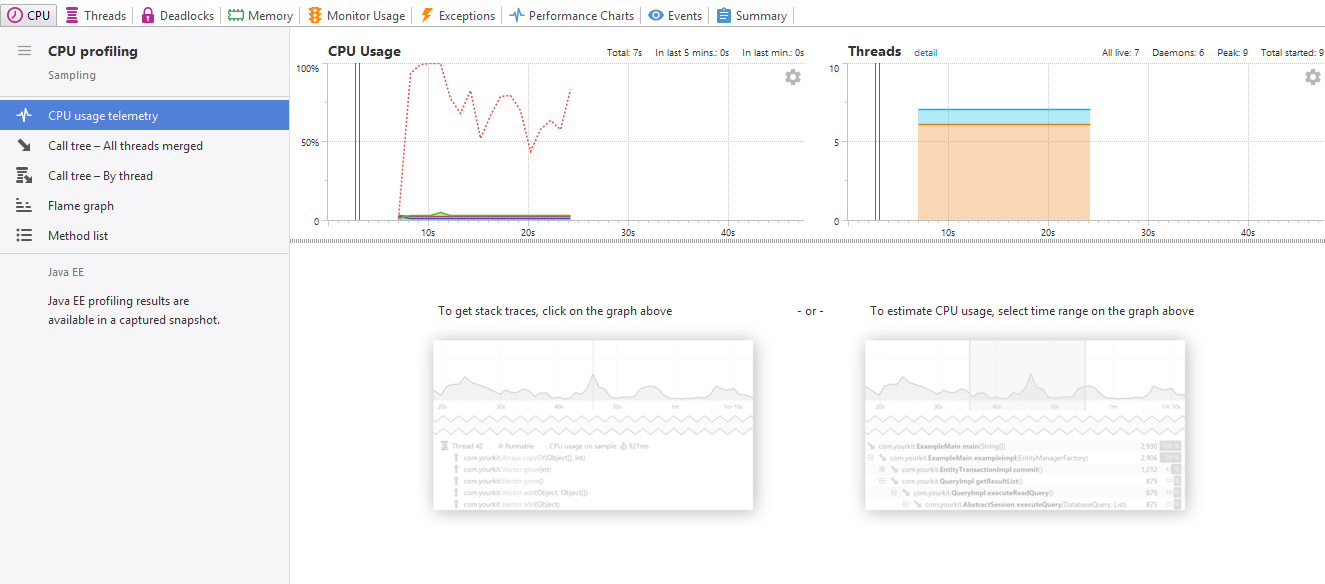


شکل 11- تابع temp( پس از بهبود)

**پس از اعمال این تغییر برنامه بدون خطا اجرا شد و داریم:**

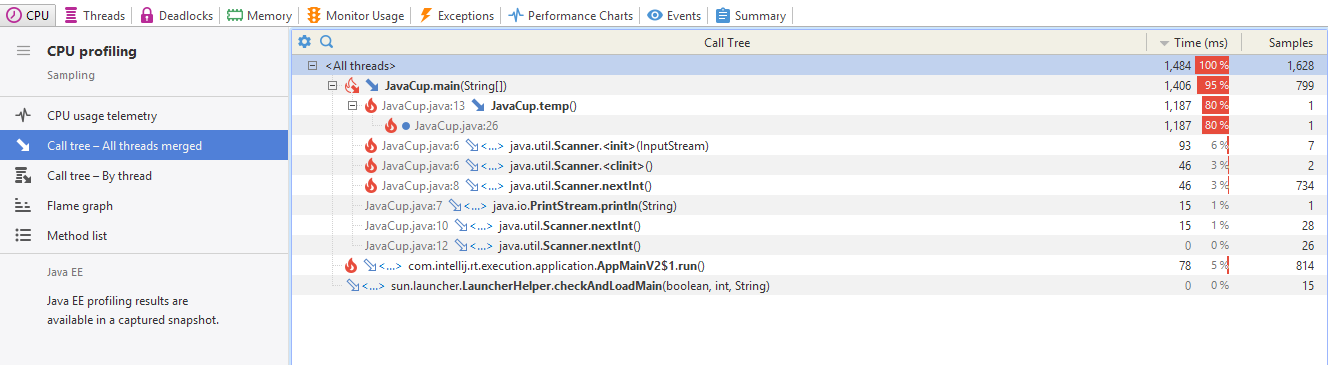


شکل 12- خروجی برنامه بعد از اجرای عملیات profiling (پس از بهبود)



شکل 13 - نمودار استفاده از پردازنده (پس از بهبود)

**مشاهده می‌شود زمان اجرایی مصرف شده توسط تابع temp به 80درصد کاهش یافت.**

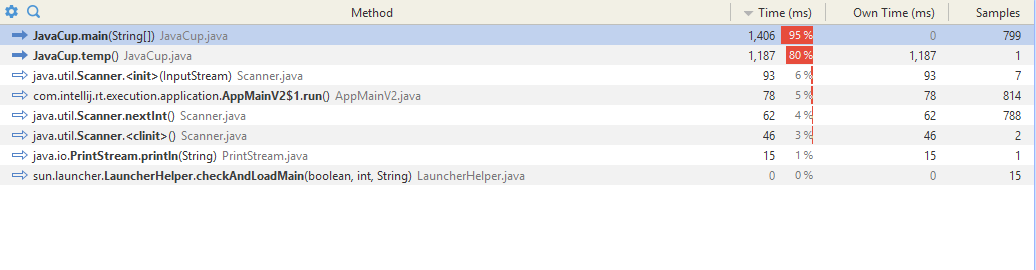


شکل 14- نمودار درختی مربوط به thread ها(پس از بهبود)

Graphical user interface, text, application

Description automatically generated

شکل 15 - نمودار درختی thread ها و زمان اجرای توابع(پس از بهبود)

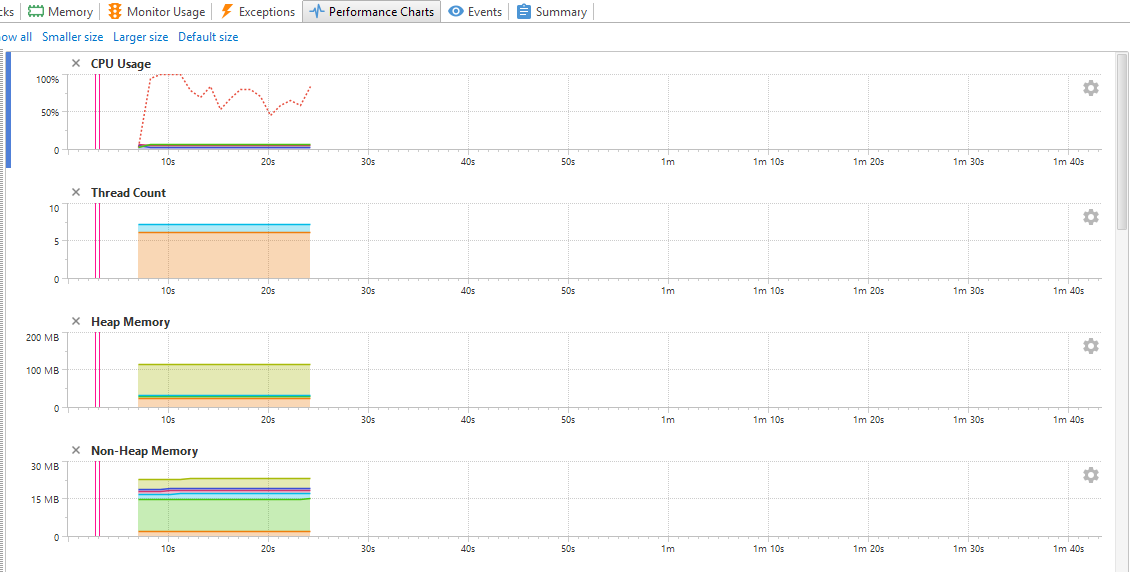


شکل 16- نمودار زمان مصرفی و درصد زمانی مربوط به توابع(پس از بهبود)

**همچنین طبق نمودارهای مربوط به حافظه مصرفی داریم:**



شکل 17- نمودار حافظه مصرفی (پس از بهبود)



شکل 18 - نمودار کارایی پردازنده و حافظه و threadها (پس از بهبود)

**طبق نمودارهای بالا، مشاهده می‌شود پس از تغییر تابع temp و استفاده از داده ساختار مناسب، منابع و زمان کمتری مصرف شد و میزان heap استفاده شده از 1.3GB به 20MB رسید که بهبود قابل توجه‌ای محسوب می‌شود. همچنین درصد زمان اجرایی تابع temp نیز کاهش یافت.**

**بخش دوم.**

**ب**