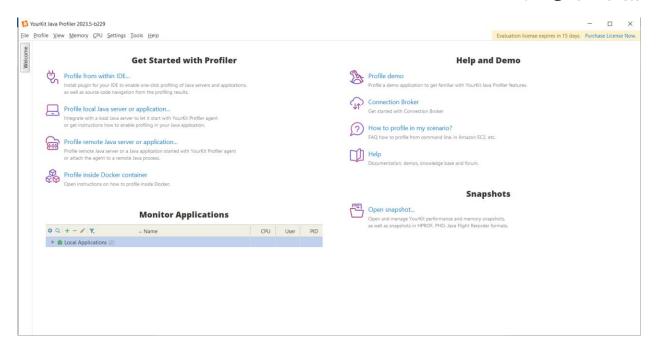
# شرح آزمایش

ابتدا نرم افزار Yourkit مخصوص زبان جاوا را دانلود و نصب می کنیم و طبق گفته کلاس، Yourkit ۱۵ Evaluation روزه را فعال می کنیم:



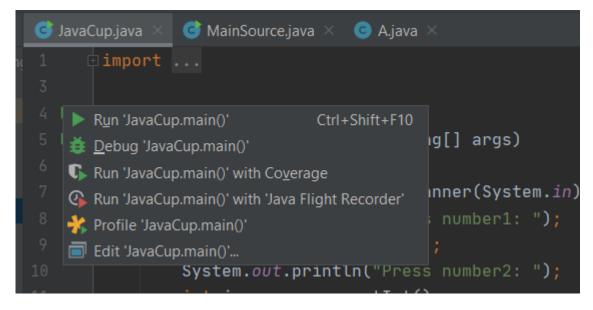
سپس روی گزینه Profile from within IDE کلیک می کنیم تا پلاگین Yourkit نصب شود.



	Profile from within IDE			
	IntelliJ IDEA uses default plugins directory			
l	Plugin will be installed for all installed and supported IntelliJ IDEA versions.			
l	O IntelliJ IDEA is configured to use the non-standard plugins directory:			
l				
	< <u>Previous</u> Ne <u>x</u> t > <u>Install Plugin</u> Cancel			

#### بخش اول

حال پروژه ذکر شده را در intellij باز می کنیم و عمل profiling را روی فایل JavaCup اجرا می کنیم:



ورودیهای مورد نیاز را وارد می کنیم و منتظر اتمام عملیات اجرا و profiling می شویم:

```
"C:\Program Files\Java\jdk1.8.0_271\bin\java.exe" ...
[YourKit Java Profiler 2023.5-b229] Log file: C:\Users\Asus\.yjp\log\JavaCup-22728.log
Press number1:
1
Press number2:
2
Press number3:
```

بعد از اتمام اجرا، تبهای مربوط به CPU و Method list و Method list را مشاهده می کنیم:



همانطور که مشخص است، در زمان اجرا، ۸۰ درصد پردازنده سیستم اشغال شده بود و Memory مربوط به Heap نیز به خاطر اینکه در حال ایجاد یک ArrayList بزرگ بودیم در حال پر شدن بود و تابعی که بیشترین مصرف را داشته، تابع temp بوده است.

```
public static void temp() {
    ArrayList a = new ArrayList();
    for (int i = 0; i < 10000; i++)
    {
        for (int j = 0; j < 20000; j++) {
            a.add(i + j);
        }
    }
}</pre>
```

یک راه برای کم کردن زمان اجرا و مصرف منابع این تابع این است که میتوانیم به جای ArrayList از Array از میناده کنیم؛ چون اندازه این آرایه ثابت است و قرار نیست تغییر کند. بعد از انجام این تغییر کد به شکل زیر در می آید:

```
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[20000*i + j] = i + j;
        }
    }
}</pre>
```

بعد از این تغییر، نتیجه profiling برنامه به این شکل خواهد بود:

	CPU time → Process (kernel)	
	CPU time → Process (user + kernel)	
	CPU time → System	
Garbage collectors → All collectors → Max call duration		
Memory → Heap → All pools → Used memory  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		Last: 11.54 MiB Limit: 1.75 GiB
Memory → Non-Heap → All pools → Used memory 2.67 MiB		Last: 12.67 MiB Limit: n/a
Ç Q → JavaCup.main(String[]) JavaCup.java ⇔ java.util.Scanner. <init>(InputStream) Scanner.java</init>	Method	▼ Time (ms)   Own Time (ms) 109
⇒ com.intellij.rt.execution.application.AppMainV2\$1.run() AppMainV2.ja ⇒ java.util.Scanner. <cli>clinit&gt;() Scanner.java ⇒ java.util.Scanner.nextInt() Scanner.java</cli>	va	46 29 46 31 20 5 31 15 10 9 15

که بهبود برنامه را نشان میدهد.

### بخش دوم

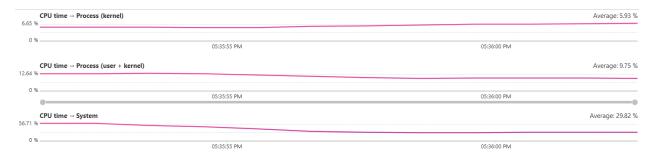
در این بخش یک قطعه کد در فایل hw5.java مینویسیم که وظیفه نوشتن به ترتیب اعداد ۱ تا ۱۰۰۰۰۰ را در فایلی به نام result.txt دارد. قطعه کد در ابتدا به این شکل است:

```
public class hw5 {
    public static void main(String[] args) {
        createFile();
        writeToFile();
    }

private static void writeToFile() {
    for (int i = 0; i < 1000000; i++) {
        try {
            FileWriter myWriter = new FileWriter(fileName: "result.txt", [appenditrue);
            Files.write(Paths.get( fint: "result.txt"), (i + "\n").getBytes(), StandardOpenOption.APPEND);
        myWriter.close();
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}

private static void createFile() {
    try {
        File myObj = new File( pathname: "result.txt");
        if (myObj.createNewFile()) {
            System.out.println("File created: " + myObj.getName());
        } else {
            System.out.println("File already exists.");
        }
    } catch (IOException e) {
        System.out.println("File already exists.");
        e.printStackTrace();
    }
}
</pre>
```

## حال عمل profiling را روی این کد انجام میدهیم:



Garbage collectors — All collectors — Max call duration			
0 05:35:55 PM	05:36:00 PM		
Memory — Heap → All pools → Used memory	Last: 8.62 MiB Limit: 1.75		
0	05:36:00 PM		
0			
Memory → Non-Heap → All pools → Used memory	Last: 14.4 MiB Limit:		
0	05:36:00 PM		
Object allocations — Bytes	Tota		
w5.main(String[]) hw5.java	11,578 97 %		
w5.writeToFile() hw5.java	11,546 97 % 3:		
ava.nio.file. <b>Files.write</b> (Path, byte[], OpenOption[]) Files.java	6,234 5 <mark>2 %</mark> 6,23		
ava.io. <b>FileWriter</b> .< <b>init</b> >(String, boolean) FileWriter.java	<b>2,765</b> 23 <mark>% 2,7</mark> 0		
ava.io. <b>OutputStreamWriter.close()</b> OutputStreamWriter.java	2,156 18 % 2,1		
ava.lang.ref. <b>Finalizer\$FinalizerThread.run(</b> ) Finalizer.java	203 2 % 21		
un.misc. <b>VMSupport.serializeAgentPropertiesToByteArray</b> () VMSupport.java	46 0 %		
om.intellij.rt.execution.application. <b>AppMainV2\$1.run</b> () AppMainV2.java	31 0 %		
w5.createFile() hw5.java	31 0 %		
ava.io. <b>PrintStream.println</b> (String) PrintStream.java	31 0%		
ava.lang.ref. <b>Reference\$ReferenceHandler.run(</b> ) Reference.java	31 0 %		
nya nia fila Batha gat (String String II) Batha inya	31 0%		

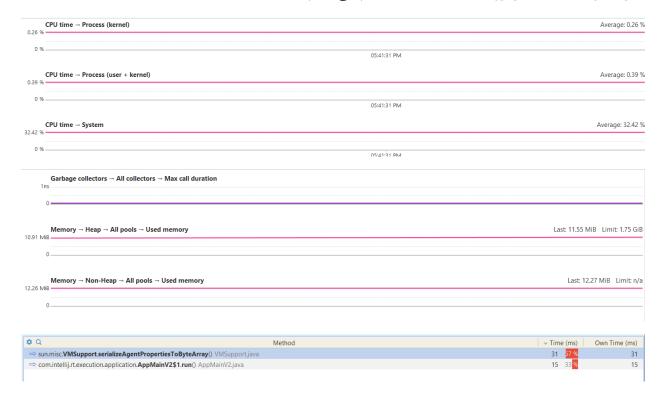
همانطور که مشاهده می شود، اجرای برنامه حدودا ۱۰ ثانیه طول کشیده است و حین اجرا، ۲۰ مگابایت حافظه و ۱۲ درصد پردازنده در گیر شده است. همچنین تابع writeToFile بیشترین مصرف منابع را به خود اختصاص داده است؛ بنابراین باید این تابع را بهبود دهیم. حال برای بهبود عملکرد برنامه، قطعه کد را به شکل زیر تغییر می دهیم:

```
public static void main(String[] args) {
    createFile();
    writeToFile2();
private static void writeToFile2() {
        StringBuilder s = new StringBuilder();
        for (int \underline{i} = 0; \underline{i} < 100000; \underline{i} + +) {
            s.append(i).append('\n');
        FileWriter myWriter = new FileWriter( fileName: "result.txt");
        myWriter.write(s.toString());
        myWriter.close();
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    System.out.println("Successfully wrote to the file.");
private static void createFile() {
        File myObj = new File( pathname: "result.txt");
        if (myObj.createNewFile()) {
            System.out.println("File created: " + myObj.getName());
        } else {
            System.out.println("File already exists.");
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
```

تغییرات اعمال شده به این شکل است: به جای اینکه هر بار در حلقه فایل را باز کنیم و عدد را بنویسیم و فایل را ببندیم، رشته مورد نظر را با یک StringBuilder میسازیم و در آخر فقط یک بار فایل را باز می کنیم و کل

رشته را در آن مینویسیم و فایل را میبندیم. چون عمل باز کردن فایل فرایند زمانبری است و تغییر یک StringBuilder با منابع و زمان بسیار کمتری انجام می گیرد.

#### حال عمل profiling را روى قطعه كد جديد انجام مي دهيم:



همانطور که در تصاویر مشخص است، اختلاف میزان مصرف دو برنامه خیلی زیاد است و برنامه بهبود زیادی پیدا کرده است و زمان اجرای آن از ۱۰ ثانیه به ۳۱ میلی ثانیه کاهش یافته است.