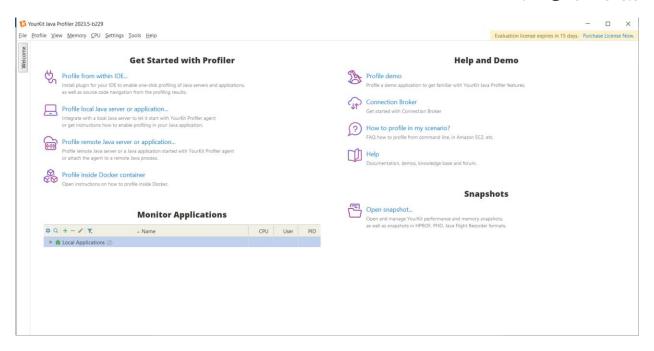
شرح آزمایش

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:



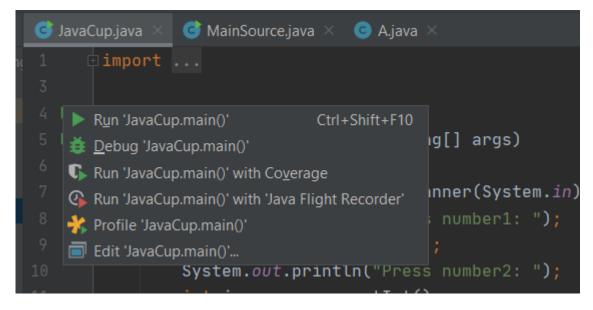
سپس روی گزینه Profile from within IDE کلیک می کنیم تا پلاگین Yourkit نصب شود.



	Profile	from within II	DE	
IntelliJ IDEA uses d Plugin will be installe	efault plugins directors of the state of the		DEA versions.	
O IntelliJ IDEA is conf	figured to use the no	n-standard plugi	ns directory:	
	< <u>P</u> revious	Ne <u>x</u> t >	<u>I</u> nstall Plugin	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) 1 % 0 %	
	CPU time → Process (user + kerne	
29	CPU time → System 65 %	
Garbage collectors — All collectors → Max call durat	on .	
0 Memory → Heap → All pools → Used memory 54 MiB		Last: 11.54 MiB Limit: 1.75 GiB
Memory → Non-Heap → All pools → Used memory		Last: 12.67 MiB Limit: n/a
Q JavaCup,main (String[]) JavaCup,java	Method	→ Time (ms) Own Time (ms) 109 70 56 0
⇒ java.util.Scanner. <init>(InputStream) Scanner.java ⇒ comintellij.rtexecution.application.AppMainV2\$1.run() App ⇒ java.util.Scanner.<clinit>0 Scanner.java ⇒ java.util.Scanner.nextInt() Scanner.java</clinit></init>	tainV2.java	62 44 8 62 46 29 46 31 20 8 31 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 (10Exception e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
   }
}

private static void createFile() {
   try {
        File myObj = new File( pathwame: "result.txt");
        if (myObj.createNewFile()) {
            System.out.println("File created: " + myObj.getName());
        } else {
            System.out.println("File already exists.");
        }
    } catch (10Exception e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}</pre>
```

حال عمل profiling را روی این کد انجام میدهیم:



0 05:35:55 PM	05:36:00 PM
Memory → Heap → All pools → Used memory	Last: 8.62 MiB Limit: 1.7
0 05:35:55 PM	05:36:00 PM
Memory → Non-Heap → All pools → Used memory	Last: 14.4 MiB Lim
0 05:35:55 PM	05:36:00 PM
Object allocations → Bytes	To
v5.main(String[]) hw5.java	11,578 97%
v5.main(String[]) hw5.java v5.writeToFile() hw5.java	
	11,578 97 %
r5.writeToFile() hw5.java	11,578 <mark>97 %</mark> 11,546 <mark>97</mark> %
/5.writeToFile () hw5.java ra.nio.file. Files.write (Path, byte[], OpenOption[]) Files.java	11,578 97 % 11,546 97 % 6,234 52% 6
/S.writeToFile() hw5.java ra.nio.file.Files.write(Path, byte[], OpenOption[]) Files.java ra.jo.FileWriter. <init>(String, boolean) FileWriter.java</init>	11,578 97 % 11,546 97 % 6,234 52 % 6,236 2,765 23 % 2
v5.writeToFile() hw5.java na.nio.file.Files.write(Path, byte(], OpenOption(]) Files.java na.io.FileWriter. <init>(String, boolean) FileWriter.java na.io.OutputStreamWriter.close() OutputStreamWriter.java na.lang.ref.Finalizer5Finalizer5Fread.run() Finalizerjava n.misc.VMSupport.serializeAgentPropertiesToByteArray() VMSupport.java</init>	11,578 97 % 11,546 97 % 6,234 52% 2,765 23 % 2 2,156 18 2 203 2 % 46 0 %
V.S.writeToFile() hw5.java va.nio.file.Files.write(Path, byte[], OpenOption[]) Files.java vaio.FilesWriter. <init*>(String, boolean) FileWriter.java vaio.OutputStreamWriter.close() OutputStreamWriter.java valang.ref.Finalizer\$FinalizerThread.run() Finalizer.java vailscyMSupport.serializeAgentPropertiesToByteArray() VMSupport.java m.intellij.rt.execution.application.AppMainV2\$1.run() AppMainV2.java</init*>	11,578 97 % 11,546 97 % 6,234 52% 6 2,765 23 6 2 2,156 18 6 2 203 2 %
V.S.writeToFile() hw5.java va.nio.file.Files.write(Path, byte[], OpenOption[]) Filesjava va.nio.file.Files.write(Path, byte[], OpenOption[]) Filesjava va.nio.File.Writer. <init*: (string,="" appmainv2.java="" boolean)="" filewriter.java="" finalizer.java="" hw5.java<="" m.intellij.rt.execution.application.appmainv2\$1.run()="" outputstreamwriter.java="" td="" v5.createfile()="" va.nio.finalizer\$finalizerthread.run()="" va.nio.outputstreamwriter.close()="" va.nisc.vmsupport.serializeagentpropertiestobytearray()="" vmsupport.java=""><td>11,578 97 % 11,546 97 % 6,234 5</td></init*:>	11,578 97 % 11,546 97 % 6,234 5
V.S.writeToFile() hw5.java va.nio.file.Files.write(Path, byte[], OpenOption[]) Files.java vaio.FilesWriter. <init*>(String, boolean) FileWriter.java vaio.OutputStreamWriter.close() OutputStreamWriter.java valang.ref.Finalizer\$FinalizerThread.run() Finalizer.java vailscyMSupport.serializeAgentPropertiesToByteArray() VMSupport.java m.intellij.rt.execution.application.AppMainV2\$1.run() AppMainV2.java</init*>	11,578 97 % 11,546 97 % 6,234 5 % 2,765 23 % 2 2,156 18 % 233 2 % 46 0 % 31 0 %

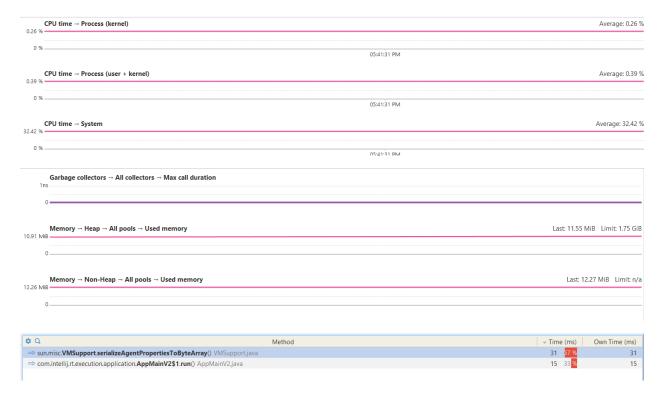
همانطور که مشاهده میشود، اجرای برنامه حدودا ۱۰ ثانیه طول کشیده است و حین اجرا، ۲۰ مگابایت حافظه و ۱۲ درصد پردازنده درگیر شده است. حال برای بهبود عملکرد برنامه، قطعه کد را به شکل زیر تغییر میدهیم:

```
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.