Design Patterns :-

1) Builder Design Pattern:

I am using lombok library to achieve builder pattern in my application.

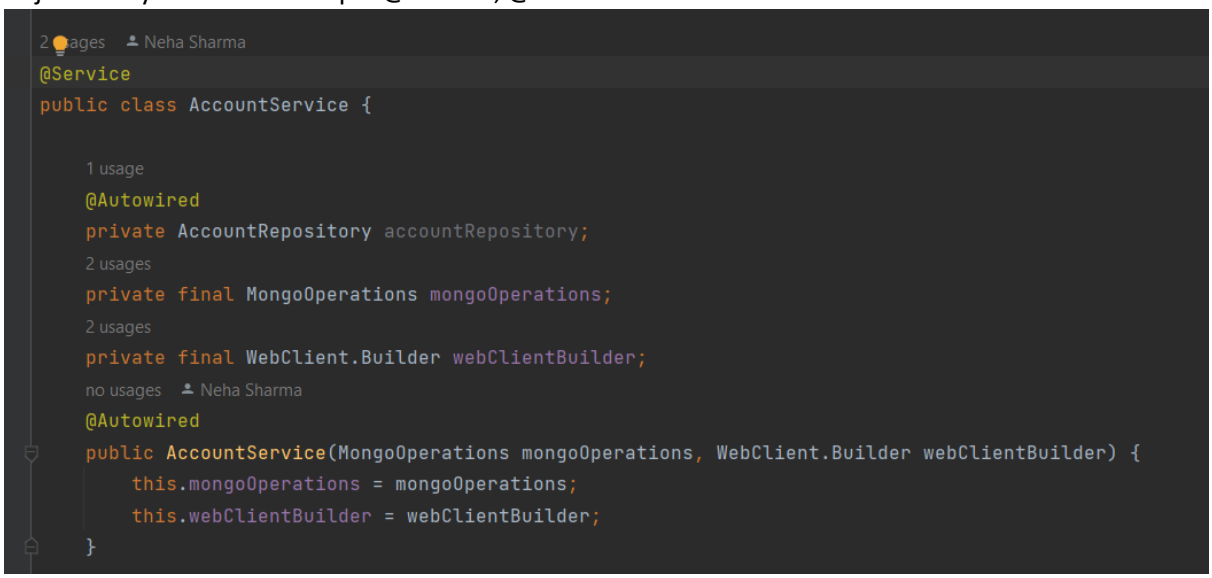
```
11 usages  Neha Sharma
@Getter
@Setter
@Document
public class Account {
    private String accountId;

    @Pattern(regexp = "^(facebook|twitter|instagram)$", message = "Account name must be either facebook, twitter, or instagram")
    private String accountName;
    private User user; // Reference to the User this account belongs to
}
```

I have used build method to build a complex object to implement builder design pattern:

```
//Builder Design Pattern
1 usage  Neha Sharma
private SocialMediaPost(PostBuilder builder) {
    this.id = builder.id;
    this.title = builder.title;
    this.content = builder.content;
    this.author = builder.author;
    this.tags = builder.tags;
    this.accountId = builder.accountId;
    this.authorId = builder.authorId;
}

4 usages  Neha Sharma
public static class PostBuilder {
    2 usages
    private ObjectId id;
    2 usages
    private String title;
    2 usages
```

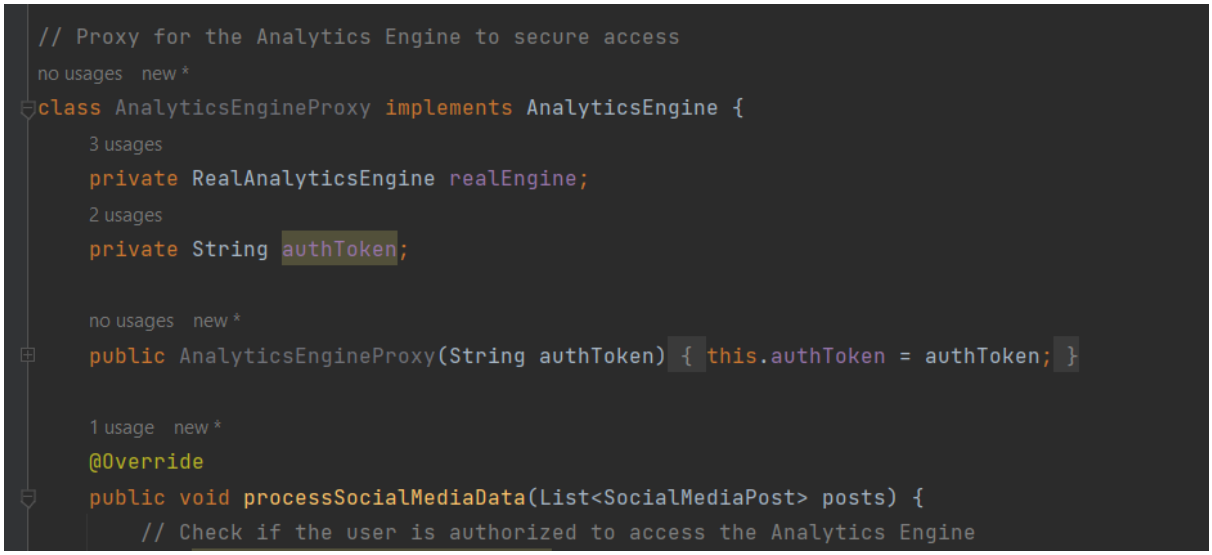
2) **Singleton Design pattern:**

I am using spring boot for my application, so I am using spring annotation to create Singleton object of my class. For example @Service, @RestController and so on.



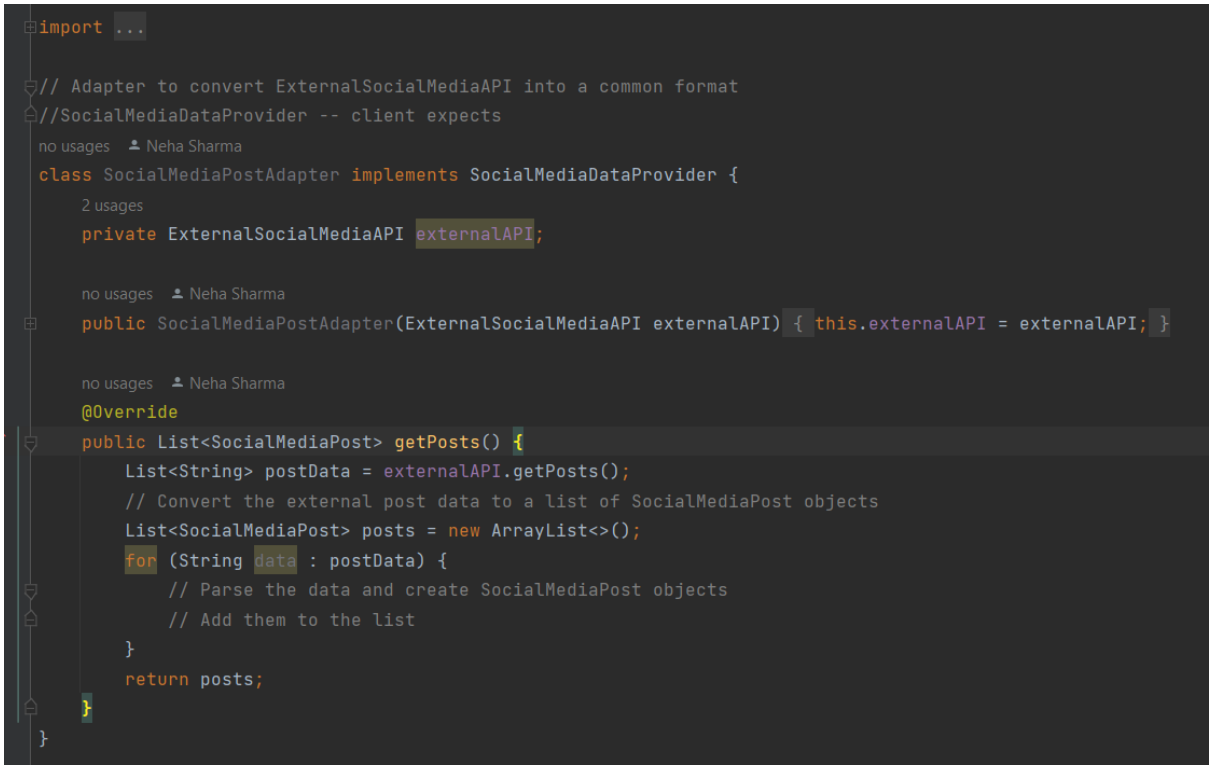
3) **Proxy Design Pattern:**

I am using Proxy Design Pattern for securing access to the Analytics Engine.



4) **Adapter Design Pattern:**

I am using Adapter Design Pattern for processing Social Media Posts from different sources.



```java
import ...

// Adapter to convert ExternalSocialMediaAPI into a common format
//SocialMediaDataProvider -- client expects
no usages  Neha Sharma
class SocialMediaPostAdapter implements SocialMediaDataProvider {
    2 usages
    private ExternalSocialMediaAPI externalAPI;

    no usages  Neha Sharma
    public SocialMediaPostAdapter(ExternalSocialMediaAPI externalAPI) { this.externalAPI = externalAPI; }

    no usages  Neha Sharma
    @Override
    public List<SocialMediaPost> getPosts() {
        List<String> postData = externalAPI.getPosts();
        // Convert the external post data to a list of SocialMediaPost objects
        List<SocialMediaPost> posts = new ArrayList<>();
        for (String data : postData) {
            // Parse the data and create SocialMediaPost objects
            // Add them to the list
        }
        return posts;
    }
}
```

SOLID Principles: -

Single Responsibility Principle:
I adopt microservice architecture for my application so that I can divide resposibilities between services. Also all my classes in my project are following Single responsibility principle.

Dependency Inversion Principle:- Spring Boot achieves the Dependency Inversion Principle (DIP) by supporting dependency injection through annotations like @Autowired, managing bean lifecycles with an Inversion of Control (IoC) container, enabling configuration of concrete implementations as beans, facilitating the use of profiles to switch between implementations, and encouraging constructor injection to enforce dependencies through abstractions, promoting modular and maintainable code adhering to DIP principles

Single Responsibility Principle:-
I am implementing single responsibility principle by not having more than one functionality specific code in a single class so that it can be modified or changed and would not effect any other classes.

Multithreading

1) Semaphore

I am using semaphore to implementing Rate limiting. For a specific API access, user should acquire control and then user can be able to access that API. No. of requests per minute can be configured as per requirement.

```java
import java.util.concurrent.Semaphore;

4 usages  Neha Sharma
public class RateLimiter {
    4 usages
    private Semaphore semaphore;

    no usages  Neha Sharma
    public RateLimiter(int maxRequestsPerSecond) { this.semaphore = new Semaphore(maxRequestsPerSecond); }

    // New constructor to set the number of permits
    1 usage  Neha Sharma
    public RateLimiter(int maxRequestsPerSecond, int permits) {
        this.semaphore = new Semaphore( permits: maxRequestsPerSecond * permits);
    }

    // Acquire multiple permits
    1 usage  Neha Sharma
    public void processRequest(int permits) {
        try {
            semaphore.acquire(permits);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

2) Concurrent Collections for Thread-Safe Data Storage:

3) Object Deep Locking:

I am using synchronized keyword to avoid race condition and maintain data consistency for multithreading environment:

```
    }
    1 usage   Neha Sharma *
    public synchronized Document toBson() {
        return new Document()
                .append("id", id)
                .append("author", author)
                .append("content", content)
                .append("tags", tags)
                .append("title", title)
                .append("accountId", accountId)
                .append("authorId", authorId);
    }
}
```

JVM Flags:

- **-Xss128k** –For setting up the stack size to 128kb.

-XX:+UseG1GC - To enable the G1 (Garbage-First) garbage collector in the JVM -

XX:MaxGCPauseMillis - Specifies the maximum pause time goal for the G1 garbage collector in milliseconds.

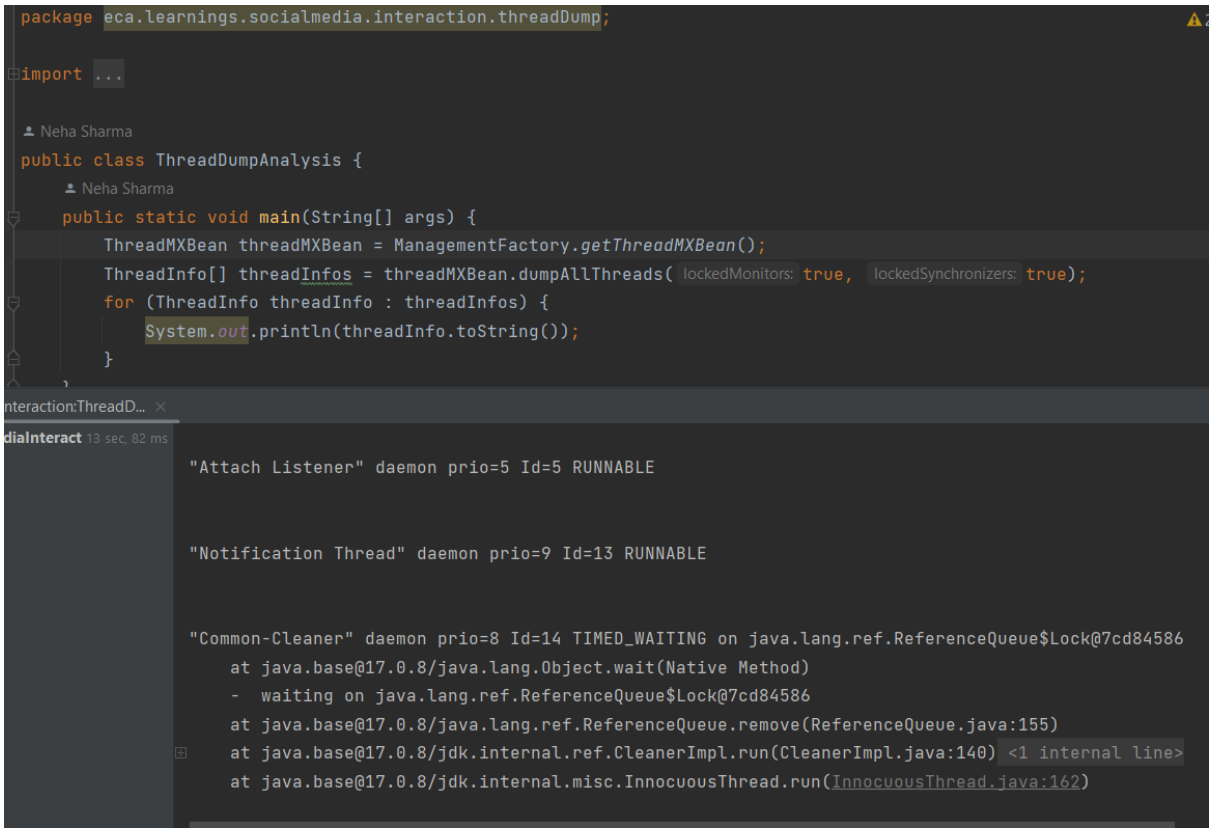
-XX:GCTimeRatio - Specifies the ratio of time spent in garbage collection versus application-level processing

```
FROM openjdk:17
VOLUME /tmp
EXPOSE 8080
ARG JAR_FILE=target/SocialMediaInteraction-1.0.0.jar
ADD ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
ENV JAVA_OPTS="-Xss512k -XX:+UseG1GC -XX:GCTimeRatio=4 -XX:MaxGCPauseMillis=200"

#Copy your shell script intop image
copy start.sh /start.sh

#Make the script excecutable
RUN chmod +x /start.sh

CMD ["/start.sh"
#!/bin/sh
exec java $JAVA_OPTS -jar /app.jar
```

Visual VM Monitoring

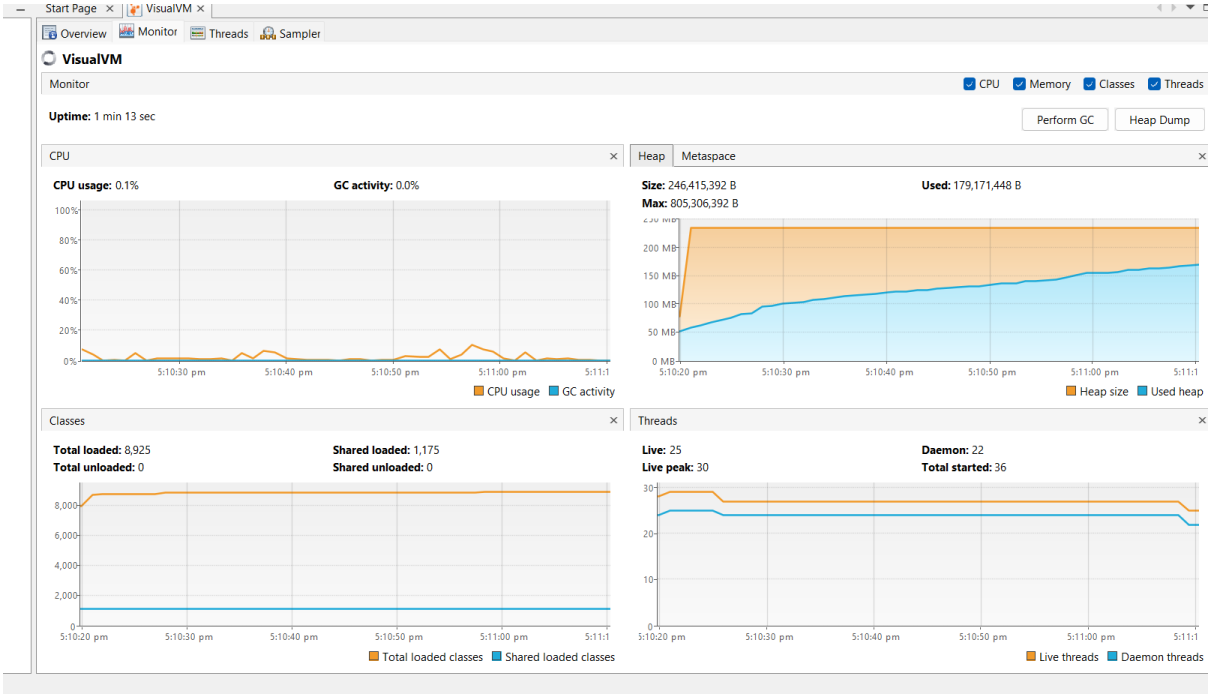
Threads :

Thread Dump:

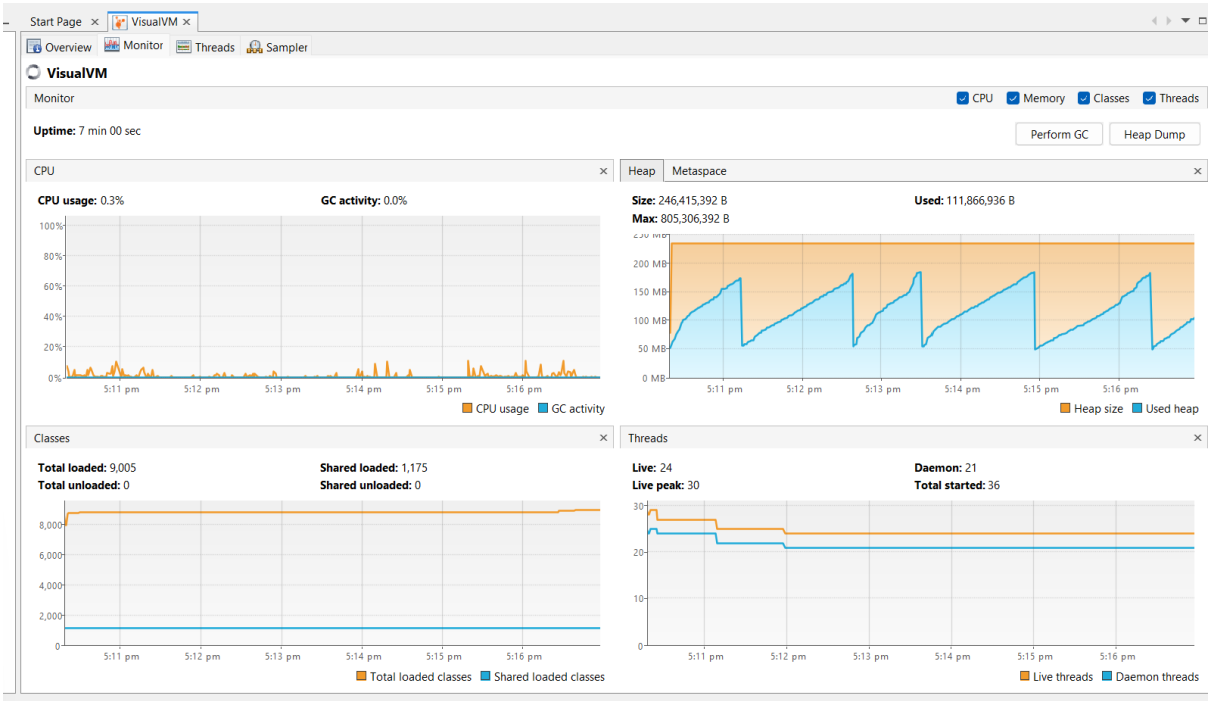


Monitoring by VisualVM:

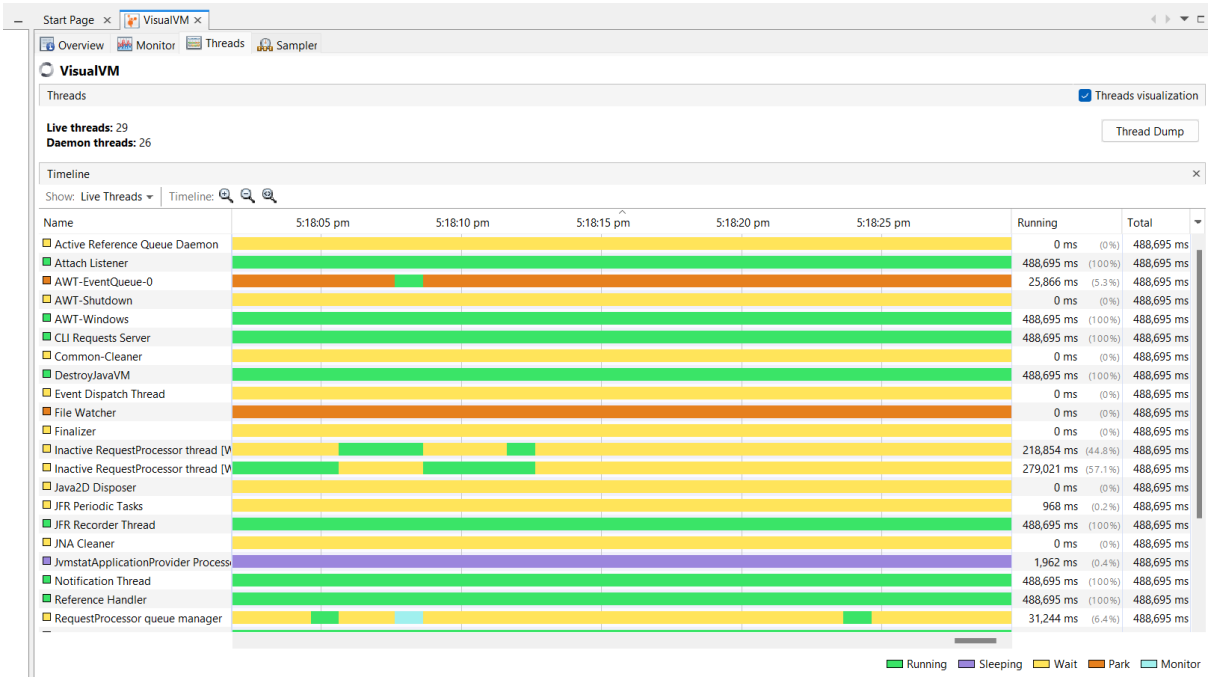
Before:



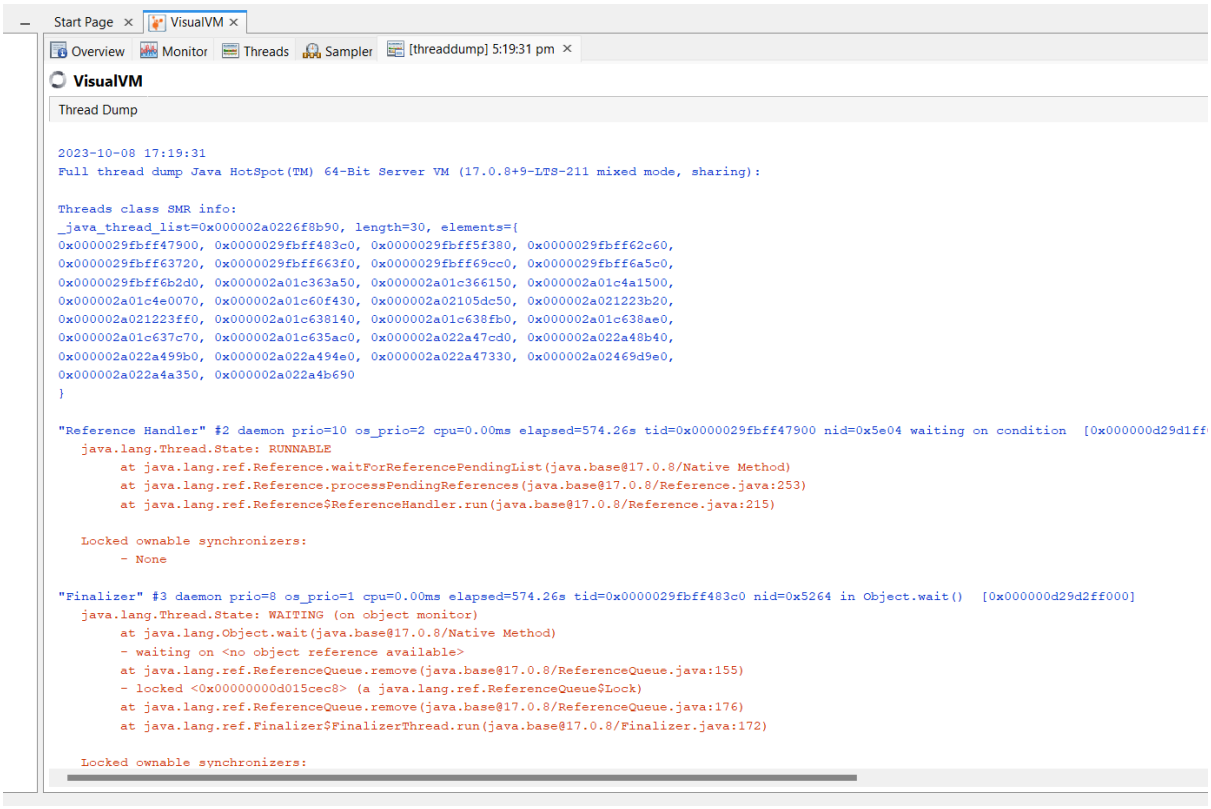
After:



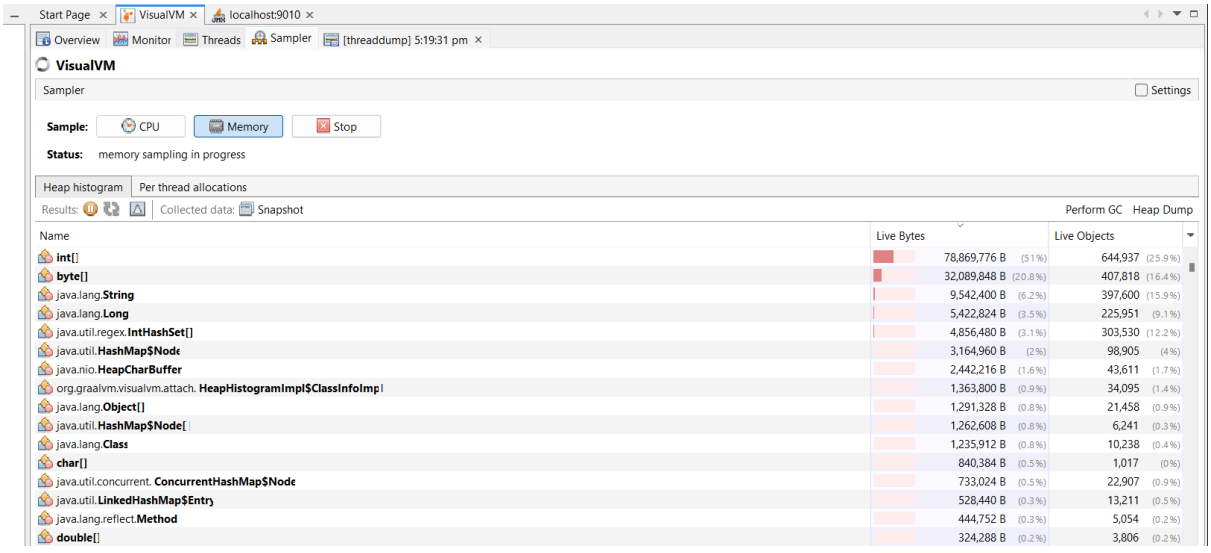
Threads:



ThreadDump:


Start Page
VisualVM
Overview
Monitor
Threads
Sampler
[threaddump] 5:19:31 pm
VisualVM
Thread Dump
2023-10-08 17:19:31
Full thread dump Java HotSpot(TM) 64-Bit Server VM (17.0.8+9-LTS-211 mixed mode, sharing):
Threads class SMR info:
_java_thread_list=0x000002a0226f8b90, length=30, elements={
0x0000029fbff47900, 0x0000029fbff483c0, 0x0000029fbff5f380, 0x0000029fbff62c60,
0x0000029fbff63720, 0x0000029fbff663f0, 0x0000029fbff69cc0, 0x0000029fbff6a5c0,
0x0000029fbff6b2d0, 0x000002a01c363a50, 0x000002a01c366150, 0x000002a01c4a1500,
0x000002a01c4e0070, 0x000002a01c60f430, 0x000002a02105dc50, 0x000002a021223b20,
0x000002a021223ff0, 0x000002a01c638140, 0x000002a01c638fb0, 0x000002a01c638ae0,
0x000002a01c637c70, 0x000002a01c635ac0, 0x000002a022a47cd0, 0x000002a022a48b40,
0x000002a022a499b0, 0x000002a022a494e0, 0x000002a022a47330, 0x000002a02469d9e0,
0x000002a022a4a350, 0x000002a022a4b690
}
"Reference Handler" #2 daemon prio=10 os_prio=2 cpu=0.00ms elapsed=574.26s tid=0x0000029fbff47900 nid=0x5e04 waiting on condition [0x000000d29d1ff
java.lang.Thread.State: RUNNABLE
at java.lang.ref.Reference.waitForReferencePendingList(java.base@17.0.8/Native Method)
at java.lang.ref.Reference.processPendingReferences(java.base@17.0.8/Reference.java:253)
at java.lang.ref.Reference$ReferenceHandler.run(java.base@17.0.8/Reference.java:215)
Locked ownable synchronizers:
- None
"Finalizer" #3 daemon prio=8 os_prio=1 cpu=0.00ms elapsed=574.26s tid=0x0000029fbff483c0 nid=0x5264 in Object.wait() [0x000000d29d2ff000]
java.lang.Thread.State: WAITING (on object monitor)
at java.lang.Object.wait(java.base@17.0.8/Native Method)
- waiting on <no object reference available>
at java.lang.ref.ReferenceQueue.remove(java.base@17.0.8/ReferenceQueue.java:155)
- locked <0x00000000d015cec8> (a java.lang.ref.ReferenceQueue$Lock)
at java.lang.ref.ReferenceQueue.remove(java.base@17.0.8/ReferenceQueue.java:176)
at java.lang.ref.Finalizer$FinalizerThread.run(java.base@17.0.8/Finalizer.java:172)
Locked ownable synchronizers:

Memory:


Start Page
VisualVM
localhost:9010
Overview
Monitor
Threads
Sampler
[threaddump] 5:19:31 pm
VisualVM
Sampler
Settings
Sample:
CPU
Memory
Stop
Status:
memory sampling in progress
Heap histogram
Per thread allocations
Results:
Collected data:
Snapshot
Perform GC
Heap Dump
Name
Live Bytes
Live Objects
int[] 78,869,776 B (51%) 644,937 (25.9%)
byte[] 32,089,848 B (20.8%) 407,818 (16.4%)
java.lang.String 9,542,400 B (6.2%) 397,600 (15.9%)
java.lang.Long 5,422,824 B (3.5%) 225,951 (9.1%)
java.util.regex.IntHashSet[] 4,856,480 B (3.1%) 303,530 (12.2%)
java.util.HashMap$Node 3,164,960 B (2%) 98,905 (4%)
java.nio.HeapCharBuffer 2,442,216 B (1.6%) 43,611 (1.7%)
org.graalvm.visualvm.attach.HeapHistogramImpl$ClassInfoImpl 1,363,800 B (0.9%) 34,095 (1.4%)
java.lang.Object[] 1,291,328 B (0.8%) 21,458 (0.9%)
java.util.HashMap$Node[] 1,262,608 B (0.8%) 6,241 (0.3%)
java.lang.Class 1,235,912 B (0.8%) 10,238 (0.4%)
char[] 840,384 B (0.5%) 1,017 (0%)
java.util.concurrent.ConcurrentHashMap$Node 733,024 B (0.5%) 22,907 (0.9%)
java.util.LinkedHashMap$Entry 528,440 B (0.3%) 13,211 (0.5%)
java.lang.reflect.Method 444,752 B (0.3%) 5,054 (0.2%)
double[] 324,288 B (0.2%) 3,806 (0.2%)

SonarLint

Used SonarLint to make code better with on-the-fly analysis and support for hundreds of deep static analysis rules to detect common mistakes, tricky bugs, and security issues.