In production, you see frequent OutOfMemoryError: Java Heap Space. What real-time tools and Java coding practices would you use to identify and fix memory leaks?

When faced with a `java.lang.OutOfMemoryError: Java heap space` in production, it means the JVM cannot allocate any more objects because the heap is full, and the Garbage Collector (GC) cannot free up enough space.

My approach combines **immediate stabilization** (restarting/scaling) with **deep-dive analysis** (heap dumps) to find the root cause, which is usually a **Memory Leak**.

1. Immediate Triage (Production)

- **Enable Heap Dumps:** Ensure the application is started with these JVM flags. This forces the JVM to save a snapshot of memory exactly when it crashes:[1] [2]

```
-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/var/logs/dumps/
```

- **Restart & Scale:** Restart the service to restore availability. If traffic is high, temporarily increase the number of instances (horizontal scaling) or increase the heap size (`-Xmx`).[3] [4]

2. Root Cause Analysis (The Tools)

Once I have the `.hprof` (heap dump) file, I analyze it offline using:

- **Eclipse Memory Analyzer (MAT):** The gold standard for this.
  - **Leak Suspects Report:** MAT automatically generates a pie chart showing which objects consume the most memory. It usually points to one specific class or collection holding 90% of the heap.[5] [6]
  - **Dominator Tree:** I look at this view to see the "Largest Objects" in memory.
- **VisualVM / JProfiler:** Useful for live monitoring if I can reproduce the issue in a staging environment.

3. Common Coding Mistakes (The Fixes)

Most memory leaks I've seen come from three specific patterns. Here is how I fix them:

A. The "Static Collection" Trap

- **The Bug:** Using a `static List` or `Map` as a local cache and never removing items. Since `static` fields live forever, the list grows until the server crashes.[7] [8]

```
// BAD: This grows forever
public static List<User> cache = new ArrayList<>();
```

- **The Fix:** Use a proper caching library like **Caffeine** or **Guava** that has automatic eviction (e.g., "expire after 10 minutes").

```
// GOOD: Auto-cleans itself
Cache<String, User> cache = Caffeine.newBuilder()
    .expireAfterWrite(10, TimeUnit.MINUTES)
    .maximumSize(1000)
    .build();
```

B. Unclosed Resources

- **The Bug:** Opening a Database Connection, File Stream, or Network Socket and forgetting to close it in a `finally` block.
- **The Fix:** Always use **Try-With-Resources**, which automatically closes everything.[7]

```
// GOOD
try (Connection conn = dataSource.getConnection()) {
    // use conn
} // auto-closed here
```

C. `equals()` and `hashCode()` Issues

- **The Bug:** Using a custom object as a Key in a `HashMap` without overriding `equals()` and `hashCode()`. The Map keeps adding duplicate keys because it thinks they are different objects, bloating memory.
- **The Fix:** Always generate these methods (use Lombok `@Data` or IDE generation) for any class used as a Map key.

⁂

1. https://blog.heaphero.io/java-outofmemoryerror-heap-space/
2. https://learn.microsoft.com/en-us/answers/questions/1518906/how-to-fix-terminating-due-to-java-lang-outofmemor
3. https://stackify.com/resolving-java-heap-space-outofmemoryerror/
4. https://sentry.io/answers/how-to-deal-with-java-lang-outofmemoryerror-java-heap-space-error/
5. http://www.hsufengko.com/notes/analyze-heap-dump-with-eclipse-memory-analyzer

6. https://www.youtube.com/watch?v=WNxexWyVWm0
7. https://last9.io/blog/how-to-spot-and-fix-memory-leaks-in-java/
8. https://www.nitorinfotech.com/blog/understanding-how-to-prevent-memory-leaks-in-java/
9. https://www.youtube.com/watch?v=L-_GpaWVgYs
10. https://www.reddit.com/r/javahelp/comments/1lys01b/outofmemoryerror_java_heap_space/
11. https://stackoverflow.com/questions/641462/can-using-too-many-static-variables-cause-a-memory-leak-in-java
12. https://blog.heaphero.io/analyzing-java-heap-dumps-for-memory-leak-detection/
13. https://last9.io/blog/java-lang-outofmemoryerror/
14. https://projects.eclipse.org/projects/tools.mat
15. https://stackoverflow.com/questions/37335/how-to-deal-with-java-lang-outofmemoryerror-java-heap-space-error