Oracle MOOC: JVM Troubleshooting

Lab 1: HotSpot JVM Memory Management

In this lab, you'll run a test program, MemoryUsage. You'll use various tools to monitor the effects of using different garbage collectors. Unless specified otherwise, all labs can be completed using either JDK 7, 8, 9, or later. Begin by unzipping `Lab1_Files.zip`.

Part 1: Monitor memory usage of a program using JConsole and Java Mission Control

1. Compile and run the Java program MemoryUsage
    a. Open a command prompt or terminal.
    b. To compile the program, type `javac MemoryUsage.java`
    c. To run the program, type `java -Xmx25m MemoryUsage`

2. Monitor memory allocation with JConsole.
    a. Launch the JConsole tool available in **<jdk>/bin** folder and connect to the running `MemoryUsage` process. This can be done by opening a command prompt or terminal and typing `JConsole`.
    b. Go to Memory tab in JConsole and monitor the growth of memory usage of all the memory pools.
    c. Clicking on each memory pool shown in the bottom-right corner of the view brings up the usage graph of that pool. Observe the memory usage of each of the memory pools.
    d. After a couple of minutes into the run, click on **Perform GC** button to invoke GC and collect the garbage.
    e. This would empty the Eden space and move the objects to either to Survivor or to old generation depending upon their age.
    f. Click on the Old Gen memory pool on the bottom-right corner. The Post-GC graph shows an increase in old generation occupancy.

3. Monitor with Java Mission Control
    a. Launch Java Mission Control. This can be done by opening a command prompt or terminal and typing `jmc`.
    b. The process `MemoryUsage` can be seen in the **JVM Browser** view. Click on the process and then double click on the **Mbean Server**. This brings up views displaying live monitored data from the process.
    c. Click on the **Overview** tab and monitor the memory usage. You can add more attributes to the Memory view that you would like to monitor by clicking on the **+**

appearing on the top-right of the Memory graph. For example we can add "old generation" or "eden" heap usage details to the graph.

Note: While adding, if you are using the search box to search for a specific memory pool, then you should use ***** before the name of the pool. For example, to search for 'eden' you should use the string '`*eden`'.

d. Go to **Memory** tab and observe the memory usage of all the pools listed there.

Part 2: Monitor memory usage and inspect if any of the memory spaces could potentially be leaking memory

1. Run the `MemoryUsage` program.
2. Start monitoring `MemoryUsage` process using **JConsole** or **JMC** as leaned in previous lab.
3. Let the process run for around 20 minutes.
4. Observe the memory usage of all the memory pools and inspect if any of the memory space is leaking memory.

Part 3: Monitor memory usage and inspect if any of the memory spaces could potentially be leaking memory

Note: With Java 8, Parallel Collector is the default collector. In this exercise we will run the given program with G1

1. Start the java process by typing:

```
java -Xmx25m -XX:+UseG1GC MemoryUsage
```

2. Start monitoring the process using **JConsole** or **JMC**.
3. Note the names of different memory spaces when the JVM is using G1 garbage collector and observe the difference in occupancy changes as compared to the Parallel collector.