



Linux DevOps Lab

JAVA PLATFORM

05_Extras. Spring Boot

Home task

Legal Notice: This document contains privileged and/or confidential information and may not be disclosed, distributed or reproduced without the prior written permission of EPAM®.

CONFIDENTIAL | Effective Date: 25-Jan-19

PREREQUISITES

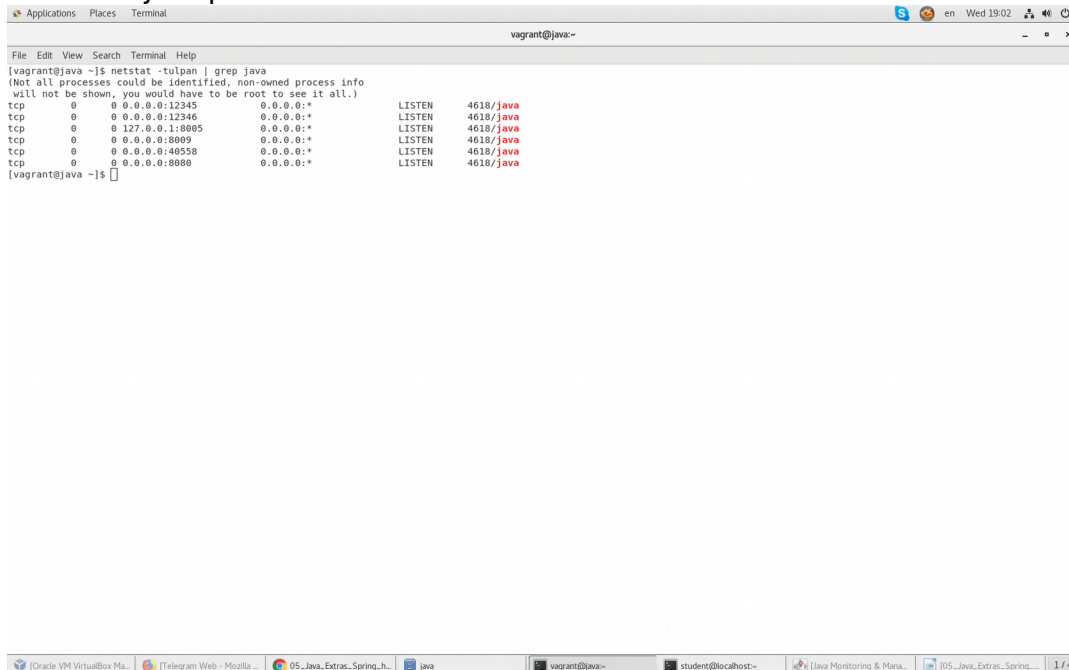
Oracle JDK 1.8 installed

GOAL

Understand principals of jmx and rmi

TASK

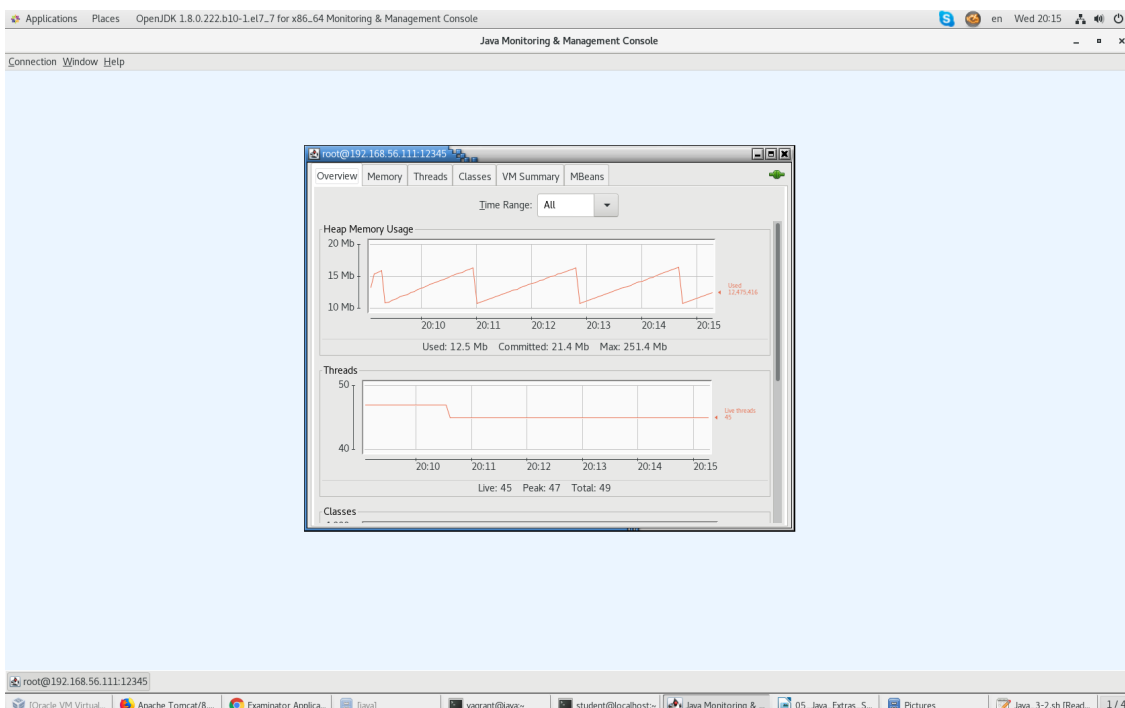
- 1) Deploy TestApp.war with next parameters:
 - o jmx ports 12345 12346



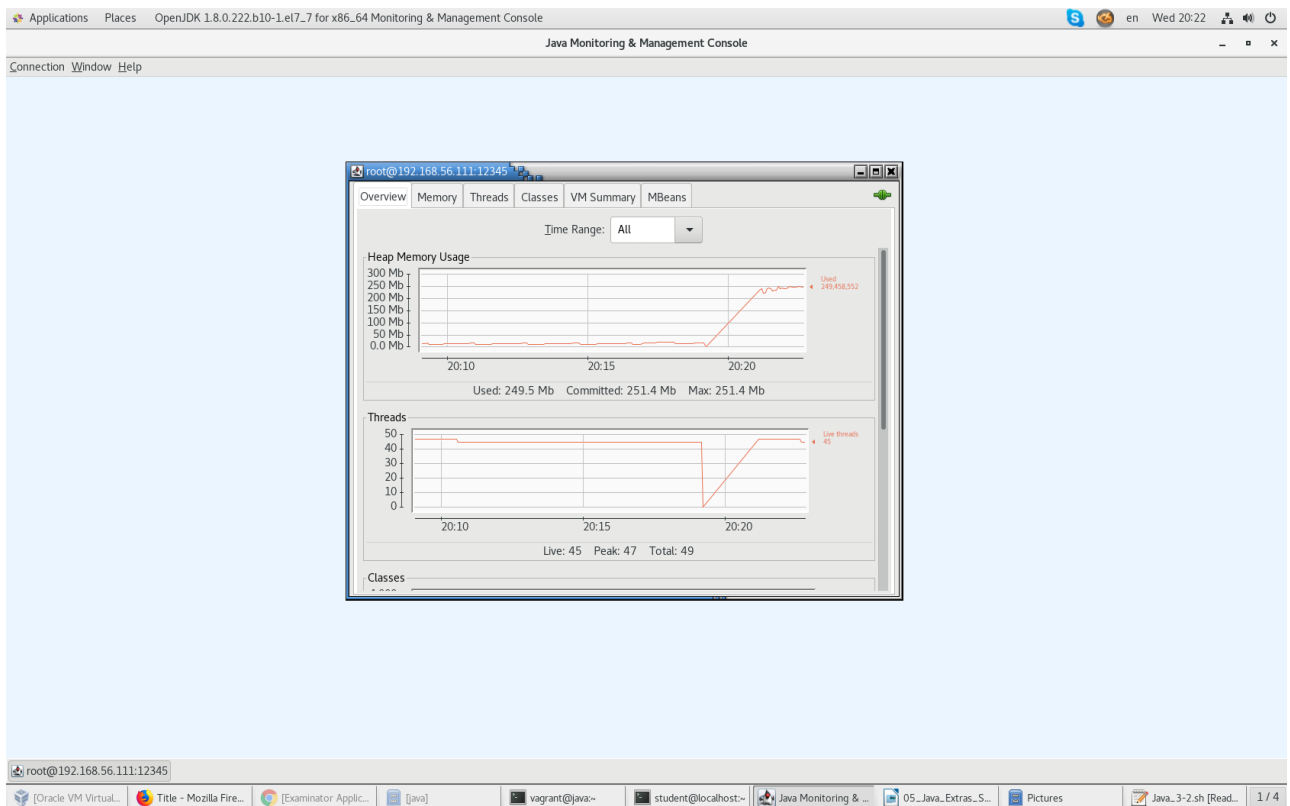
The screenshot shows a terminal window titled 'vagrant@java:~' with the following output from the command `netstat -tulpan | grep java`:

```
(vagrant@java ~)$ netstat -tulpan | grep java
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
tcp        0      0 0.0.0.0:12345        0.0.0.0:*           LISTEN     4618/java
tcp        0      0 0.0.0.0:12346        0.0.0.0:*           LISTEN     4618/java
tcp        0      0 127.0.0.1:8085        0.0.0.0:*           LISTEN     4618/java
tcp        0      0 0.0.0.0:8089        0.0.0.0:*           LISTEN     4618/java
tcp        0      0 0.0.0.0:40558        0.0.0.0:*           LISTEN     4618/java
tcp        0      0 0.0.0.0:8080        0.0.0.0:*           LISTEN     4618/java
```

- o connect via jconsole and retrieve heap and thread count information



- o write simple java application to gather Tomcat heap information and thread count
- 2) Press **Parse employees with memory leak** button and upload JSON-file with employees info

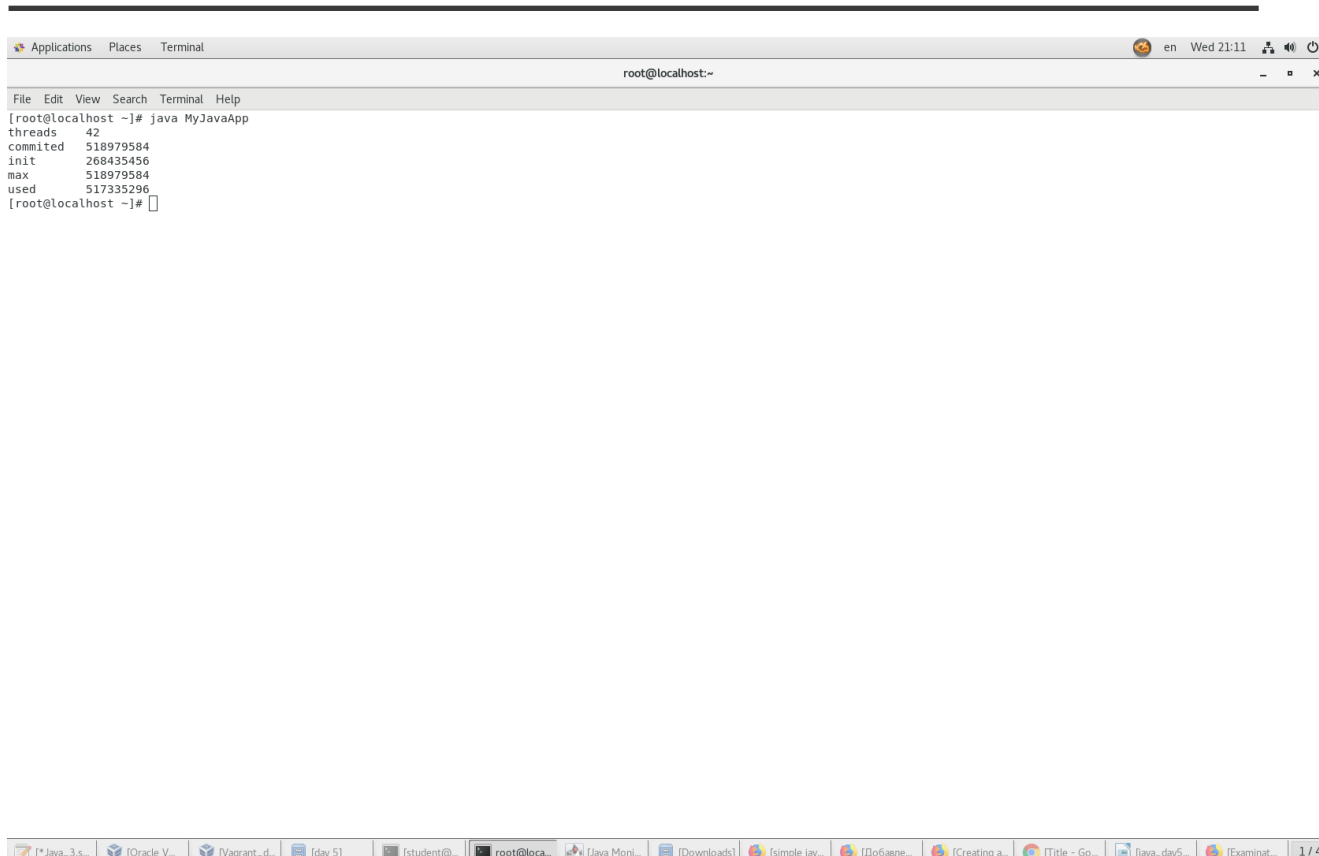


- 3) Run application before and after parsing, describe what you see.

```

File Edit View Search Terminal Help
[root@localhost ~]# java MyJavaApp
threads      42
committed   259522560
init         268435456
max          518979584
used         55903832
[root@localhost ~]#

```



```
File Edit View Search Terminal Help
[root@localhost ~]# java MyJavaApp
threads      42
committed    518979584
init          268435456
max           518979584
used          517335296
[root@localhost ~]#
```

Самым главным показателем здесь является количество используемой памяти. Мы можем видеть что heap заполнен в результате чего мы имеем Out of memory. GC ничего не вычищает. До распарсивания мы могли наблюдать работу GC т.к. размер используемой памяти рос и уменьшался. После распарсивания уменьшений размера хипа мы не наблюдаем. Дальнейшее выполнение программы не возможно.

Наша самописная программа дала нам такой же результат как и jconsole, только в случае с jconsole мы имеем числовые показатели и графики в динамике, самописная программа представила лишь числовые показатели в момент ее вызова.