Java --version

JVM = this is the execution engine =main component which executes the byte code

Allocates and deallocates memory for Java objects.

Garbage collection: Automatically reclaims unused memory

JRE= JVM + essential java libraries (implementation of java SE class library- a set of data structures, algorithms, api to access them)

JDK ((its full development kit )) = JRE + additional tools (like compiler javac) to facilitate performance monitoring , analysis and tuning ex:- debugger, profiler,

If you are changing from java8 –java 11 in eclipse you should change in 3 places

Build path

Java compiler

Project facets

|  |  |
| --- | --- |
| how to write class and var name  how to format the code & all info is here | <https://google.github.io/styleguide/javaguide.html> |

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | JDK | JRE | JVM |
| Purpose | Development of Java applications | Running Java applications | Execution of Java bytecode |
| Components | Compiler, JRE, development tools | JVM, core libraries | Bytecode interpreter, memory manager, etc. |
| Availability | Downloadable and installable | Included within JDK or separately downloadable | An integral part of JRE |
| Platform dependence | Platform-specific | Platform-specific | os specific jvms will be there  bec these should take byte code(which is plat independent) and it should convert that to plat dep code by interacting with o.s |
|  |  |  |  |

### Hotspot vm components

|  |  |
| --- | --- |
| Generally byte code is called intermediate code (platform independent code – means this is generated without interacting with underlying OS)  JIT compiler is the one which takes byte code and transforms it to native ( underlying os specific code )understandable code – bec os cant understand byte bec that byte code is generated without interacting with os  Many performance improvements are noticed from Jit compilation | VM runtime is the execution engine  Garbage collector = memory manager  VM runtime responsibilities    VM runtime will not create objects – creating objs Is the responsibility of garbage collector  **To see all jvm flags java -XX:+PrintFlagsFinal -version**  **java -XX:+PrintFlagsFinal -XX:+PrintGCDetails -version** |
| Garbage collector | 4 types of GC   1. 2 GC belongs to Stop the world GC = those 2 will pause the java app / stops all java threads and does the garbage coll process 2. Mostly concurrent GC 3. Generational garbage collector = this collector will divide the heap into 2 or more regions (like young generation & old generation)   newly created obj gets space in young generation, if they live longer those objects will be moved to old generation   1. Serial garbage collector –   Single threaded stop the world young generation collector  single threaded stop the world old gen collector   1. parallel Garbage collector - Multi threaded young and old generation collector 2. Concurrent garbage collector   ex:- G1 is an comb of mostly conc & stop the world old gen coll |

#### Heap layout

In java heap = RAM storage, all the objects in heap will be stored in RAM only

|  |  |
| --- | --- |
| 1st young/ Eden space will fill up if survived those will be pushed to s1, s1 survivour areas, if again survived after minor GC, then those will be moved to old generation area | VisualVM was previously distributed also in GraalVM 19~23.0 and Oracle JDK 6~8. But now we have to download the visual vm seperately  You can download the latest version of VisualVM from the official website:  <https://visualvm.github.io/?VisualVM_2.1.10>  <https://visualvm.github.io/download.html> |

#### Tools

|  |  |
| --- | --- |
| Many GC viewer tools are there here are the some  GC viewer tool  here it will show how much heap is filled  how many times gc triggered- still how much left | <https://www.tagtraum.com/gcviewer.html> |
| visual vm |  |

#### Analysing the performance

Easies way in prod is to Generate heap dump using java commands / from visual VM tool and import that dump file (hprof file) into eclipse it will tell us perfectly

1. We should know which Gc we are using – are we using any stop the world serial Garbage collector ? – if yes then switch to mostly concurrent GC
2. We should monitor target os system metrics – cpu usage, virtual mem usage, because if the target cpu(where our java apps are deployed) is having less space available

then our java apps will work slowly (you can check the “performance monitor” windows application to see current cpu usage in numbers)

1. Is application always waiting for IO operations – like analysing where it is taking time?- db hitting, or rest service invocation (Network I/O), file writing (DISK I/O)..

Ex:- on db invocation if we got lot of data our app cant store all billions of data in heap so alternatively we used sql cursors

1. Monitor the JVM metrics
2. We have to identify if there is any memory leak (using Visual GC or any tool) – if we notice heap is getting filled again and again **ever after GC trigger** then we should suspect something is wrong – we have to identify which class is keeping objects alive
3. Heavily Using static class members- since static unless the class is unloaded data always stays forever – so garbage collector cant delete those

If u are using those then its fine, else don’t un necessarily use them

1. Unclosed resources – like db connections

|  |  |
| --- | --- |
| import heap dump file into eclipse  eclipse will analyse and gives below report |  |
| eclipse will exactly tell us the class in which we have memory leaks  ex:- it shows as in Acache class we have a hashmap which holds lot of data |  |
| VISUAL vm also tells us the root class name where memory leak happened |  |

|  |  |
| --- | --- |
| all ways to handle leaks   * U can generate heap dumps using commands   java –XX:HeapDu  mpOnOutOfMemoryError |  |
|  | u can even start prod server happily by printing GC logs, it doesnot cause any over head |

Sample Risky code

Project examples

1. Implemented batch insertion in kafka

* Initially it was serial consumption – used to hit db for every record, later we started consuming in batch , and inserting all the records as a batch

1. Replaced with spring jdbc from hibernate

#### Monitoring JVM metrics

|  |  |
| --- | --- |
| Monitoring garbage collection – by seeing garbage collection logs  **java -XX:+PrintGCTimeStamps -jar myapp.jar**  this is like while running the app we are asking jvm to print the GC logs as well | **java -XX:+PrintFlagsFinal -XX:+PrintGCDetails -version**  **java -XX:+PrintGCDateStamps -version**  **java -XX:+PrintGCTimeStamps** |
|  |  |
|  |  |
|  |  |
|  |  |

Some other solutions

|  |  |
| --- | --- |
| 1st open the visual Vm tool and install plugin called “visual GC” and see params  and then identify – what is taking more time   1. Is heap getting filled frequently ? – then tune heap sizes 2. Is JIT compilation taking more time? 3. Either use visual vm or print the gc logs while running the app |  |
| Tuning the heap sizes- |  |
| is JIT compilation taking more time? |  |
|  |  |
|  |  |

Age of an object is nothing but no of times obje survived after garbage collection

#### **Jstatd with Visual Vm**

|  |  |
| --- | --- |
|  | we can connect remote machine to visual vm also  like if u want to see dev server metrics |

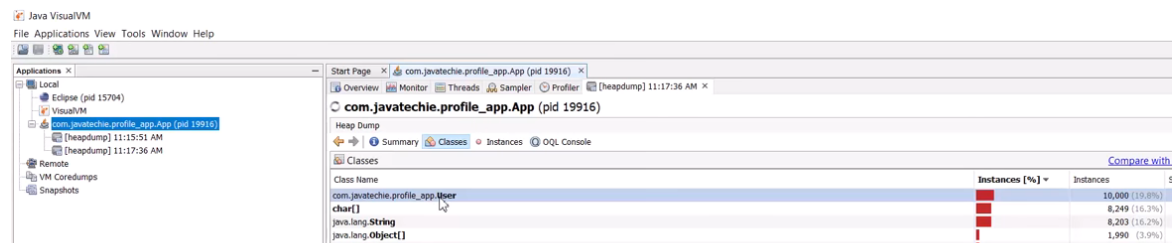
Generally with visual vm it will show only local java apps

But if u want to point the visual vm to remote prod running java apps, then we should use jstatd

#### Checking heap dump

In jvisual visual vm there is button called perform heap dump

Then u can open dump and check below



#### Sample Risky leaking code

|  |  |
| --- | --- |
|  |  |

### Solutions

|  |  |
| --- | --- |
|  |  |
|  |  |
| if db invocation is taking time |  |

#### SQL/ Db related improvements

|  |  |
| --- | --- |
| select only few columns – avoid \* | use spring JDBC instead of spring jpa |
| fetch only few rows at a time using cursors  and use stored procedures | if u are using mongodb database use sharding concepts |
| use joins instead of sub queries |  |

#### Spring related improvements

|  |  |
| --- | --- |
| if possible change ur architecture to async using kafka |  |
| batch insertion – we reduced the number of times to hit database – database load got reduced |  |
| too many REST calls = consuming more time   1. if u change arch- then use kafka else make async calls using @Async and completable future or 2. use any aynchronous api like Webclient to make REST calls 3. implement caching to cache the REST output |  |
|  |  |

#### Core java coding

|  |  |
| --- | --- |
| Never maintain un-used data in class static variables- bec those will not get unloaded untill u shutdown the app | use String builder instead of string- so that un- necessary object creation can be avoided – bec if those obj needs to be cleared again GC should run to identify unused and reclaim that memory |
| use async api like completableFuture | use parallel streams |
|  |  |

## Creating an immutable class

Immutable objects are those objects which can’t be changed after creation ex:- string is immutable

1. Don’t provide any setter methods (initialization must happen through constructor only), variables must be private and final so that with ref u should not change the value if public means anyone can access and modify the value
2. Prevent subclassing - make class as final so that no childs will be there , Make it as private constructor so that child class cons cant not call super class constructor , and provide a static factory method to get instance