In your local jconsole will be used
System.gc();
Runtime.getRuntime().gc()

Javac will convert the .java file to .class file with byte code . JVM inside has Just in Time compilation , The method most used in code will be compiled to native machine code . If application is running on MAC then JVM will compile most commonly used method to mac JVM code

JVM comes with 2 compilers called C1 and C2 , C1 is able to do Native Level 1 , 2 & 3 compilation
C2 is for Native compilation 4.

JVM will put the method compiled under C2 to Java code cache for optimal performance .
If you are using Java 7 and below then code cash size is 32 mega bye,if you using 32 bit JVM or 48 if you using 64 bit JVM megha bytes
If you are using Java 8 and above the code cash size is up to 240 megabytes .

C1 is client compiler
C2 is client and server compilers

If the objects are not available stack are eligible for GC
Any obejects on Heap which cannot be reached through the reference from the stack is " eligible for garbage collection"
Static objects are not eligible for GC

In java 8 during start of application heap memory occupied example initial there is a 20 mega bye then that memory will be constant through out the application. So that when System.GC() method is called memory cleanup will done but in Java 11 when application is started JVM will keep required memory but it will not use entire OS memory. So when transaction is increased it will have impact on performance because each time JVM request OS the memory . So when we running the application we need to run the application with VM argument -Xms300m (m for megha bytes)

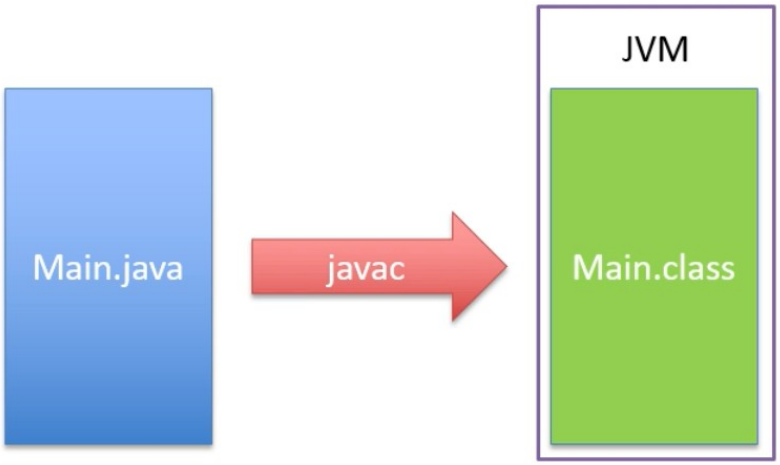
Finalyze method is deprecated in Java 9 and above and it is not good practise to use
Softleaks : When ab object remain reference when no longer needed

If the Java Vusal VM where we see the graph of memory used once the used memory line is constant then VM is actually looking for memory but not available .

Command to Generate the Heap Dump is
-XX: +HeapDumpOnOutOfMemoryError
-XX:HeapDumpPath=someFilepath

Tool to analyze the Heap Dump: Eclispse Memory Analyzer & it can be downloaded from
https://www.eclipse.org/mat

What happens inside the JVM?



Thursday, January 11, 2024 4:08 PM

Array and ArrayList

Array	ArrayList
Array objects are of fixed length.	ArrayList objects are of variable length.
Array does not support generics.	ArrayList supports generics.
Array can store both primitive types as well as reference types.	ArrayList can store only reference types.
Length of the Array is provided by the length variable.	Length of the ArrayList is provided by the size() method

Difference between linkedList and ArrayList

ArrayList	LinkedList
1) ArrayList internally uses a **dynamic array** to store the elements.	LinkedList internally uses a **doubly linked list** to store the elements.
2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory.	Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
3) An ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.
4) ArrayList is **better for storing and accessing**	LinkedList is **better for manipulating**

5) The memory location for the elements of an ArrayList is contiguous.	The location for the elements of a linked list is not contagious.
6) Generally, when an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList.	There is no case of default capacity in a LinkedList. In LinkedList, an empty list is created when a LinkedList is initialized.
7) To be precise, an ArrayList is a resizable array.	LinkedList implements the doubly linked list of the list interface.

A doubly linked list has nodes that keep information about its previous as well as next nodes using the previous and next pointers respectively. Also, the previous pointer of the first node and the next pointer of the last node is set to null in the doubly linked list

Difference between Interface and Abstract class

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.

7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

Interface is a not class and by default all methods inside the interface are public and abstract.
All the variable defined inside the interface is static and final

When we implement the multithreading by extending the Thread class then we can initiate our own class
But when we implement the Runnable interface we need to give implementation to Run method and create the instance of Thread class

Multiple Interface implementation is possible in java but we can extend only one class

Synchronized block of code is written with in the method to make sure that particular block of code needs to be thread safe . If whole method needs to be synchronized then you can define key word as Synchronized

If we define multiple method inside class as Synchronized then one method will be executed at a time since by default lock is applied for method.
If static method and non static synchronized method can be executed parallel by 2 different threads

Racecondition : It occurs when 2 or more threads update the same value , As consequence , leave the value in a undefined or inconsistent state.
Volatile key words basically will be useful in singleton , In this case memory will be updated directly in RAM instead of updating in Cache
In the below code remove method q.wait() is called to make sure that if q does not have any item to remove it will wait . In add method there is method called notifyAll() will notify all waiting threads

```java
public BlockingQueue(int cap)
{
    q = new LinkedList<>();
    capacity = cap;
}

public boolean add(int item)
{
    synchronized(q) {
        if(q.size() == capacity)
            // do something
        q.add(item);
        q.notifyAll();
        return true;
    }
}

public int remove()
{
    synchronized(q) {
        if(q.size() == 0)
        {
            try {
                q.wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        int element = q.poll();
        return element;
    }
}
```

Writable

Stack Vs Heap

Picture in picture

edure

Stack	Heap
• Stack is used for static memory allocation	• Heap is used for dynamic memory allocation
• Variables that are allocated on the stack are accessible directly from memory, thus these can run very fast	• Accessing objects on the heap takes more time
• Memory allocation happens when the program is compiled	• Memory allocation begins at runtime

Stack and Heap are ways in which Java

SUBSCRIBE