Java Memory Management

Difficulty Level : Medium Last Updated : 14 Dec, 2018

This article will focus on Java memory management, how the heap works, reference types, garbage collection, and also related concepts.

Why Learn Java Memory Management?
We all know that Java itself manages the memory and needs no explicit intervention of the programmer. Garbage collector itself ensures that the unused space gets cleaned and memory can be freed when not needed. So what's the role of programmer and why a programmer needs to learn about the Java Memory Management ? Being a programmer, you don't need to bother with problems like destroying objects, all credits to the garbage collector. However the automatic garbage collection doesn't guarantee everything. If we don't know how the memory management works, often we will end up amidst things that are not managed by JVM (Java Virtual Machine). There are some objects that aren't eligible for the automatic garbage collection.

Hence knowing the memory management is essential as it will benefit the programmer to write high performance based programs that will not crash, or if does so, the programmer will know how to debug or overcome the crashes.

Introduction:

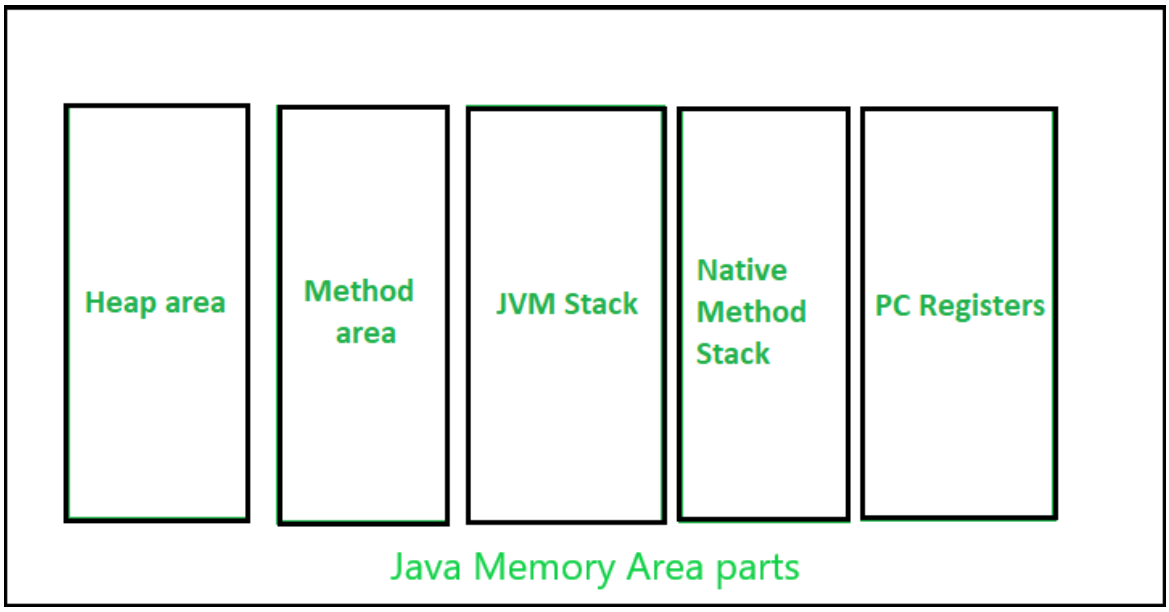
In every programming language, the memory is a vital resource and is also scarce in nature. Hence it's essential that the memory is managed thoroughly without any leaks. Allocation and deallocation of memory is a critical task and requires a lot of care and consideration. However in Java, unlike other programming language, the JVM and to be specific Garbage Collector has the role of managing memory allocation so that the programmer needs not to. Whereas in other programming languages such as C the programmer has direct access to the memory who allocates memory in his code, thereby creating a lot of scope for leaks.

The major concepts in Java Memory Management :

- JVM Memory Structure
- Working of Garbage Collector

Java Memory Structure:

JVM defines various run time data area which are used during execution of a program. Some of the areas are created by the JVM whereas some are created by the threads that are used in a program. However, the memory area created by JVM is destroyed only when the JVM exits. The data areas of thread are created during instantiation and destroyed when the thread exits.



JVM Memory area parts

Let's study these parts of memory area in detail:

Heap :

- It is a shared runtime data area and stores the actual object in a memory. It is instantiated during the virtual machine startup.
- This memory is allocated for all class instances and array. Heap can be of fixed or dynamic size depending upon the system's configuration.
- JVM provides the user control to initialize or vary the size of heap as per the requirement. When a new keyword is used, object is assigned a space in heap, but the reference of the same exists onto the stack.
- There exists one and only one heap for a running JVM process.

Scanner sc = new Scanner(System.in);

The above statement creates the object of Scanner class which gets allocated to heap whereas the reference 'sc' gets pushed to the stack.

Note: *Garbage collection in heap area is mandatory.*

Method Area:

- It is a logical part of the heap area and is created on virtual machine startup.
- This memory is allocated for class structures, method data and constructor field data, and also for interfaces or special method used in class. Heap can be of fixed or dynamic size depending upon the system's configuration.
- Can be of a fixed size or expanded as required by the computation. Needs not to be contiguous.

Note: *Though method area is logically a part of heap, it may or may not be garbage collected even if garbage collection is compulsory in heap area.*

JVM Stacks:

- A stack is created at the same time when a thread is created and is used to store data and partial results which will be needed while returning value for method and performing dynamic linking.
- Stacks can either be of fixed or dynamic size. The size of a stack can be chosen independently when it is created.
- The memory for stack needs not to be contiguous.

Native method Stacks:

Also called as C stacks, native method stacks are not written in Java language. This memory is allocated for each thread when its created. And it can be of fixed or dynamic nature.

Program counter (PC) registers:

Each JVM thread which carries out the task of a specific method has a program counter register associated with it. The non native method has a PC which stores the address of the available JVM instruction whereas in a native method, the value of program counter is

undefined. PC register is capable of storing the return address or a native pointer on some specific platform.

Working of a Garbage Collector:

- JVM triggers this process and as per the JVM garbage collection process is done or else withheld. It reduces the burden of programmer by automatically performing the allocation or deallocation of memory.
- Garbage collection process causes the rest of the processes or threads to be paused and thus is costly in nature. This problem is unacceptable for the client but can be eliminated by applying several garbage collector based algorithms. This process of applying algorithm is often termed as **Garbage Collector tuning** and is important for improving the performance of a program.
- Another solution is the generational garbage collectors that adds an age field to the objects that are assigned a memory. As more and more objects are created, the list of garbage grows thereby increasing the garbage collection time. On the basis of how many clock cycles the objects have survived, objects are grouped and are allocated an 'age' accordingly. This way the garbage collection work gets distributed.
- In the current scenario, all garbage collectors are generational, and hence, optimal.

> ***Note: System.gc()*** *and* ***Runtime.gc()*** *are the methods which requests for Garbage collection to JVM explicitly but it doesn't ensures garbage collection as the final deci sion of garbage collection is of JVM only.*

Knowing how the program and it's data is stored or organized is essential as it helps when the programmer intends to write an optimized code in terms of resources and it's consumption. Also it helps in finding the memory leaks or inconsistency, and helps in debugging memory related errors. However, the memory management concept is extremely vast and therefore one must put his best to study it as much as possible to improve the knowledge of the same.