Does assigning NULL really help Garbage Collection?

Date: 2020-09-15 00:02:03

Assign NULL to an object.

One of the reasons of my friend hating java was “Java variable needs to be assigned NULL to help GC.”. And actually, I remember someone told me that we could call the System.gc() for manually triggering the Garbage Collection.

What is Garbage Collection (GC)?

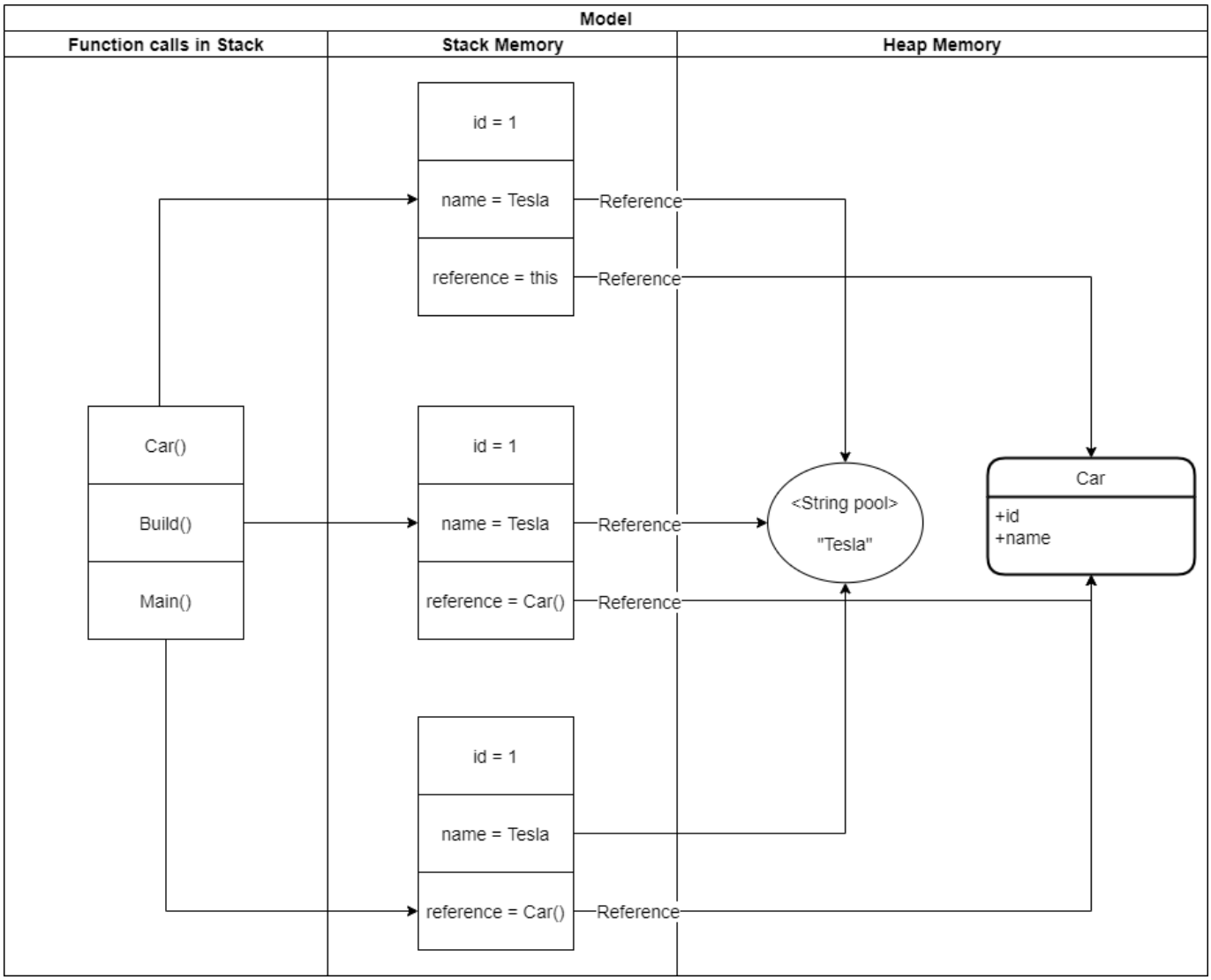
GC is an automatical storage management system. In the GC perspective, It automatically collects all USELESS resources (Such as an object which is not being referenced). The GC thread monitors the Heap resource within the application lifecycle.

Stack and Heap

First of all, I have to explain what are the Stack and Heap memories and how GC works on Java. There are lines of code right here for the example.

```
class Car {
    int id;
    String name;
    public Car(int id, String name) {
        this.id = id;
        this.name = name;
    }
}
public class Main {
    private static Car build(int id, String name) {
        return new Car(id, name);
    }
    public static void main(String[] args) {
        int id = 1;
        String name = "Tesla";
        Car car = null;
        car = build(id, name);
    }
}
```

What happens in the code?



The variable of the Car object in Stack which references to a Heap memory resource in each function calls. Assigning NULL to Car object only redirects the reference of Car to a NULL reference. The occupied resource still in Heap and probably not going to trigger the GC operation whether the Young Generation (Survivor Space 1 & 2) not being full.
Meanwhile, the string "Tesla" probably will not immediately be collected if you assign NULL to the name of the Car object as well.

How about explicitly calls System.gc() method?

I haven't encountered that many good reasons to invoke System.gc() from the application. Here is the code inside System.gc():

```
public static void gc() {
    Runtime.getRuntime().gc();
}
```

As java has another thread for the GC usage while the resources in Heap being insufficient. Triggering the System.gc() is like sending a signal to the JVM for preparing the GC operation. And the GC operation is unpredictable and delay, that means you could not know when the GC started. GC can also reject your request if all the resources are still being referenced (Memory Leak?) and it will throw OutOfMemory Exception when the Heap memory is fully occupied.

How to optimize the GC operation?

According to the diagram above, when an object created, the functions in the Stack will keep the reference only. At the same time, Heap will assign a block of resources and keep the object. The lifecycle of stack memory is generally shorter then Heap, the stack memory collection is always faster than Heap and invisible to other threads. I encourage to frequently use the variables in function scope instead of global. StackOverFlow Exception will be thrown if the stack memory is fully occupied.