♦ Q37-Q42: Top 50+ Core on Java Garbage Collection Interview Questions & Answers

Posted on February 13, 2015 by Arulkumaran Kumaraswamipillai — No Comments ↓

0 Like Share 0 G+1

As a Java developer, you may not need to know how the JVM works, but the most important topic that you must know is Java Garbage Collection. How the **G**arbage **C**ollection and object referencing work in Java.

Top 50+ Core Java Interview Questions Links:

Q01-Q10 | Q11-Q23 on OOP | Q24-Q36 on classes, interfaces and generics | Q43-Q54 on Objects

Q37. What do you know about the Java garbage collector? When does the garbage collection occur?
A37. Each time an object is created in Java, it goes into the area of memory known as heap. The Java heap is called the garbage collectable heap. The garbage collection cannot be forced. The garbage collector runs in low memory situations. When it runs, it releases the memory allocated by an unreachable object. The garbage collector runs on a low priority daemon (i.e. background) thread. You can nicely ask the garbage collector to collect garbage by calling *System.gc( )* but you can't force it.

Q38. What is an unreachable object?
A38. An object's life has no meaning unless something has reference to it. If you can't reach it then you can't ask it to do anything. Then the object becomes unreachable and the garbage collector will figure it out. Java automatically collects all the unreachable objects periodically and releases the memory consumed by those unreachable objects to be used by the future reachable objects.

You can use the following options with the Java command to enable tracing for garbage collection events.

```
java -verbose:gc

```

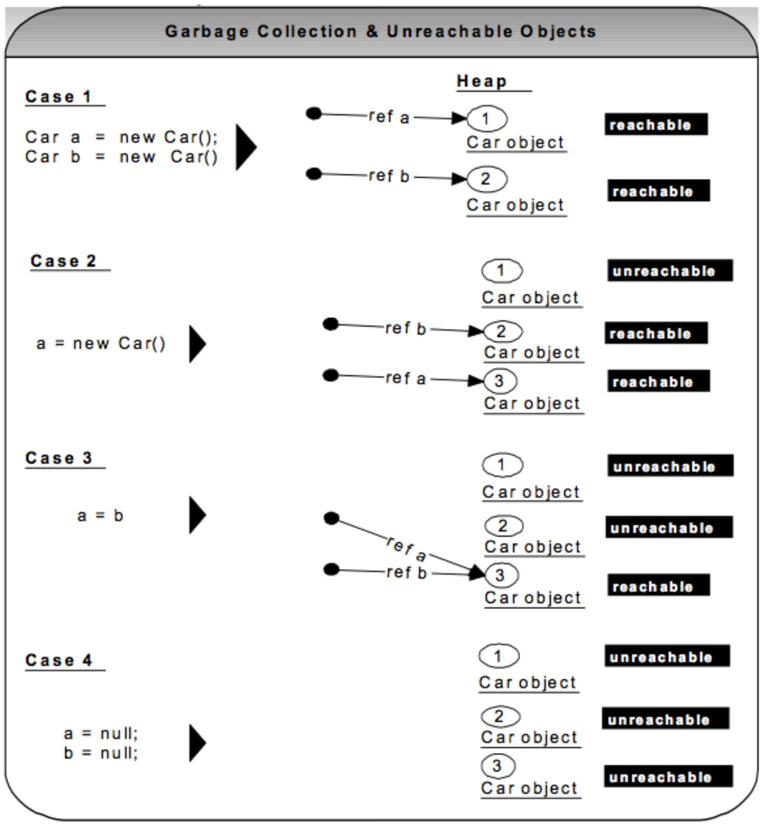
SQL, XML, UML, JSC
Hadoop & BigData Int
Java Architecture Inte
Scala Interview Q&As
Spring, Hibernate, & I
Testing & Profiling/Sa
Other Interview Q&A 1
▶Free Java Intervie

16 Technical Key Areas

open all | close all
Best Practice (6)
Coding (26)
Concurrency (6)
Design Concepts (7)
Design Patterns (11)
Exception Handling (3
Java Debugging (21)
Judging Experience I
Low Latency (7)
Memory Managemen
Performance (13)
QoS (8)
Scalability (4)
SDLC (6)
Security (13)
Transaction Managen

80+ step by step Java Tutorials

open all | close all
Setting up Tutorial (6)
Tutorial - Diagnosis (2
Akka Tutorial (9)
Core Java Tutorials (2



Java Garbage Collection reachable vs unreachable objects

Q39. What is the difference between a weak reference and a soft reference? Which one would you use for caching?
A39. **Weak reference**: A weak reference, simply put, is a reference that isn't strong enough to force an object to remain in memory. Weak references allow you to leverage the garbage collector's ability to determine reachability for you, so you don't have to do it yourself. You create a weak reference like this:

```
Car c1 = new Car( );          //referent is c1 is
WeakReference<Car> wr = new WeakReference<Car>(c1
</code>
```

A weak reference is a holder for a reference to an object, called the referent. Weak references and weak collections are powerful tools for heap management, allowing the application to use a more sophisticated notion of reachability, rather than the "all or nothing" reachability offered by ordinary (i.e. strong) references.

Hadoop & Spark Tuto
JEE Tutorials (19)
Scala Tutorials (1)
Spring & HIbernate Tu
Tools Tutorials (19)
Other Tutorials (45)

100+ Java pre-interview coding tests

open all | close all
Can you write code?
♦ Complete the given
Converting from A to I
Designing your classe
Java Data Structures
Passing the unit tests
What is wrong with th
Writing Code Home A
Written Test Core Jav
Written Test JEE (1)

How good are your .....?

open all | close all
Career Making Know-
Job Hunting & Resum

A *WeakHashMap* stores the keys using *WeakReference* objects, which means that as soon as the key is not referenced from somewhere else in your program, the entry may be removed and is available for garbage collection. One common use of *WeakReferences* and *WeakHashMaps* in particular is for adding properties to objects. If the objects you are adding properties to tend to get destroyed and created a lot, you can end up with a lot of old objects in your map taking up a lot of memory. If you use a *WeakHashMap* instead the objects will leave your map as soon as they are no longer used by the rest of your program, which is the desired behavior.

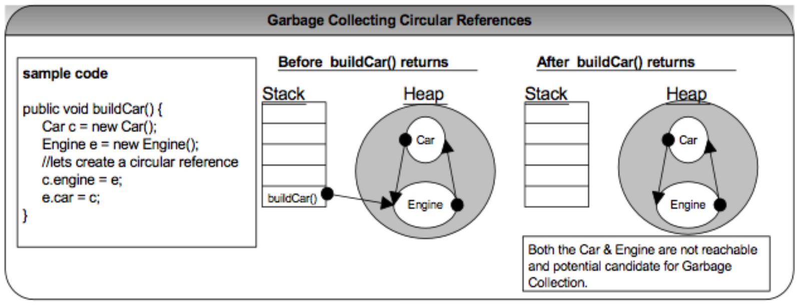
Soft reference is similar to a weak reference, except that it is less eager to throw away the object to which it refers. An object which is only weakly reachable will be discarded at the next garbage collection cycle, but an object which is softly reachable will generally stick around for a while as long as there is enough memory. Hence the soft references are good candidates for a cache.

```
byte[ ] cache = new byte[1024];
//... populate the cache. The referent is  cache
SoftReference<byte> sr = new    SoftReference<byt

```

The garbage collector may or may not reclaim a softly reachable object depending on how recently the object was created or accessed, but is required to clear all soft references before throwing an ***OutOfMemoryError***.

Note: The weak references are **eagerly** garbage collected, and the soft references are **lazily** garbage collected under low memory situations.

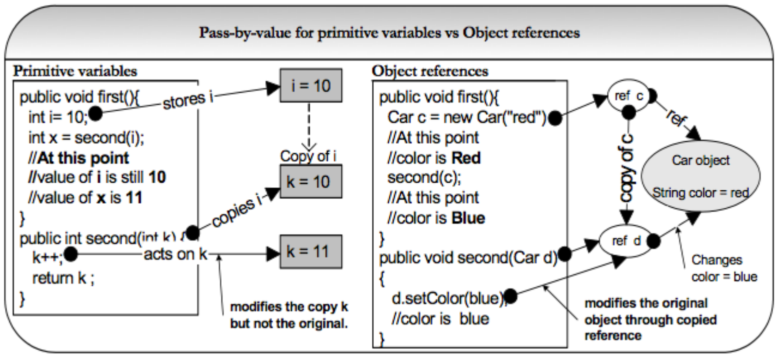
Q40. If you have a circular reference of objects, but you no longer reference it from an execution thread, will this object be a potential candidate for garbage collection?
A40. Yes. Refer diagram below.


Garbage Collecting Circular References
sample code
Before buildCar() returns
After buildCar() returns
Stack
Heap
buildCar()
Car
Engine
Both the Car & Engine are not reachable and potential candidate for Garbage Collection.

```
public void buildCar() {
    Car c = new Car();
    Engine e = new Engine();
    //lets create a circular reference
    c.engine = e;
    e.car = c;
}
```

Java GC cyclic reference

Q41. What is the main difference between pass-by-reference and pass-by-value? Which one does Java use?
A41. Other languages use pass-by-reference or pass-by-pointer. But in Java no matter what type of argument you pass the corresponding parameter (primitive variable or object reference) will get a copy of that data, which is exactly how **pass-by-value** (i.e. copy-by-value) works.

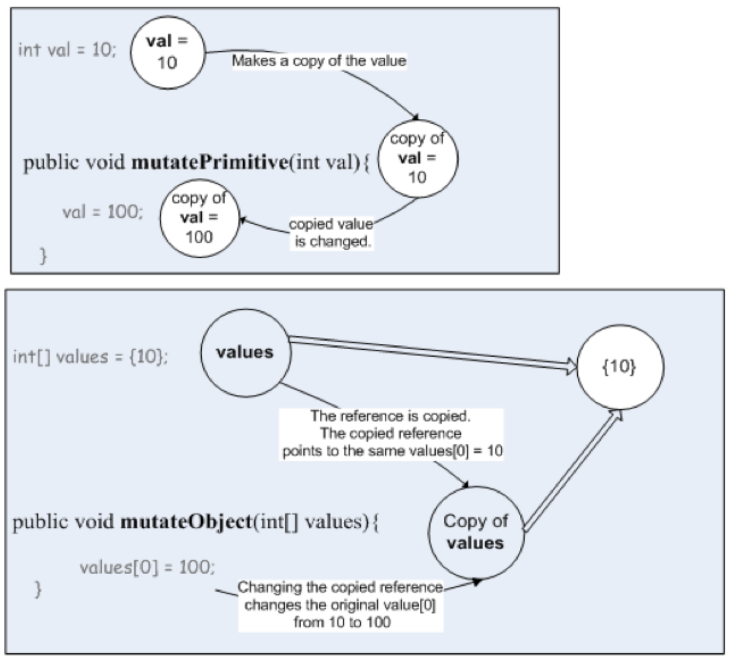


```
public void first(){
  int i= 10;
  int x = second(i);
  //At this point
  //value of i is still 10
  //value of x is 11
}
public int second(int k) {
   k++;
   return k ;
}
```

```
public void first(){
  Car c = new Car("red")
  //At this point
  //color is Red
  second(c);
  //At this point
  //color is Blue
}
public void second(Car d)
{
   d.setColor(blue);
   //color is  blue
}
```

pass-by-ref vs pass-by-value

In Java, if a calling method passes a reference of an object as an argument to the called method, then the passed-in reference gets copied first and then passed to the called method. Both the original reference that was passed-in and the copied reference will be pointing to the same object. So no matter which reference you use, you will be always modifying the same original object, which is how the **pass-by-reference** works as well.


int val = 10;
val = 10
Makes a copy of the value
copy of val = 10
public void mutatePrimitive(int val){
val = 100;
copy of val = 100
copied value is changed.
}
int[] values = {10};
values
{10}
The reference is copied. The copied reference points to the same values[0] = 10
public void mutateObject(int[] values){
Copy of values
values[0] = 100;
}
Changing the copied reference changes the original value[0] from 10 to 100

pass-by-ref vs pass-by-value

If your method call involves inter-process (e.g. between two JVMs) communication, then the reference of the calling method has a different address space to the called method sitting in a separate process (i.e. separate JVM). Hence inter-process communication involves calling method passing objects as arguments to called method **by-value** in a serialized form.

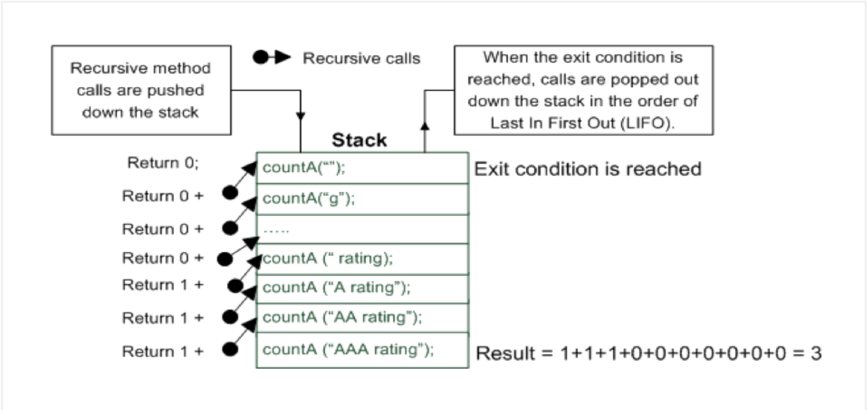
Q42. How would you take advantage of Java being a stack based language? What is a reentrant method?
A42. Recursive method calls are possible with stack based languages and re-entrant methods.

A **re-entrant** method would be one that can safely be entered, even when the same method is being executed, further down the call stack of the same thread. A non-re-entrant method would not be safe to use in that way. For example, writing or logging to a file can potentially corrupt that file, if that method were to be re-entrant.

A function is **recursive** if it calls itself. Given enough stack space, recursive method calls are perfectly valid in Java though it is tough to debug. Recursive functions are useful in removing iterations from many sorts of algorithms. **All recursive functions are re-entrant**, but not all re-entrant functions are recursive.

Stack uses LIFO (Last In First Out), so it remembers its 'caller' and knows whom to return when the function has to return. Recursion makes use of system stack for storing the return addresses of the function calls.

```java
public class RecursiveCall {

    public int countA(String input) {

        // exit condition – recursive calls must
        //otherwise you will get stack overflow
        if (input == null || input.length( ) ==
            return 0;
        }

        int count = 0;

        //check first character of the input
        if (input.substring(0, 1).equals("A")) {
            count = 1;
        }

        //recursive call to evaluate rest of the
        //(i.e.  2nd character onwards)
        return count + countA(input.substring(1)
    }

    public static void main(String[ ] args) {
         System.out.println(new RecursiveCall( )
    }
}

```


Recursive method calls are pushed down the stack
Recursive calls
When the exit condition is reached, calls are popped out down the stack in the order of Last In First Out (LIFO).
Stack
Return 0;
Return 0 +
Return 0 +
Return 0 +
Return 1 +
Return 1 +
Return 1 +
countA("");
countA("g");
.....
countA (" rating);
countA ("A rating");
countA ("AA rating");
countA ("AAA rating");
Exit condition is reached
Result = 1+1+1+0+0+0+0+0+0+0 = 3

Java is a stack based language

Recursion might not be the efficient way to code, but recursive functions are shorter, simpler, and easier to read and understand. Recursive functions are very handy in working with tree structures and avoiding unsightly nested for loops. If a particular recursive function is identified to be a real performance bottleneck as it is invoked very frequently or it is easy enough to implement using iteration like the sample code below, then favor iteration over recursion.

Top 50+ Core Java Interview Questions Links:

Q01-Q10 | Q11-Q23 on OOP | Q24-Q36 on classes, interfaces and generics | Q43-Q54 on Objects

Popular Posts

- ♦ 11 Spring boot interview questions & answers

  825 views
- ♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

  767 views
- 18 Java scenarios based interview Questions and Answers

  400 views
- 001A: ♦ 7+ Java integration styles & patterns interview questions & answers

  388 views
- 01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

  295 views
- ♦ 7 Java debugging interview questions & answers

  293 views
- 01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

  285 views
- ♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

  279 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

240 views

001B: ♦ Java architecture & design concepts interview questions & answers

201 views

Bio | **Latest Posts**

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ ♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

♦ Q43-Q54: Top 50+ Core on Java Objects Interview Questions & Answers ›

Posted in FAQ Core Java Job Interview Q&A Essentials, member-paid, Top 50+ FAQ Core Java Interview Q&A

Tags: Core Java FAQs, Java/JEE FAQs, Novice FAQs, TopX

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from "Blah blah" to "Wow"? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category ▼

© Disclaimer