Garbage Collection Interview Questions in Java

**Garbage Collection** is one of the most important topics of Java. Garbage collection allows programs to perform automatic memory management. **Garbage Collection interview questions** are usually asked by interviewers.

1) Why Garbage Collection is necessary in Java?

In programming languages such as C and C++, the developer programmatically reclaims the space that is allocated to an object in the memory. In Java programming, the user is not responsible for managing the space memory used by the objects. In JVM, a garbage collection routine is added as its part, which is responsible for identifying and deleting objects that are no longer in use in memory.

2) What is the drawback to Garbage Collection?

The main drawback of Garbage Collection is that it freezes all those threads which are currently active at the time of occurring memory recovery phase. The garbage collection algorithms take time in seconds or minutes to run, and due to this, garbage collection routines can't be scheduled.

3) Explain the structure of Java Heap.

A chunk of memory shared among all threads is referred to as Java Heap. In Java, Heap is a combination of three generations such as new generation, old generation and PermGen space. To learn more about Heap structure, go through the following link:

69.4M

1.3K

Exception Handling in Java - Javatpoint

<https://www.javatpoint.com/java-heap>

4) Explain PermGen space in Java.

The internal representation of the Java classes held by JVM is included in the PermGen space. PermGen space contains garbage data collected in the same way as the heap's other parts (Young generation and Old generation) collected. To learn more about Heap structure, go through the following link:

<https://www.javatpoint.com/permgen-space-java>

5) Explain the difference between a minor, major and full garbage collection.

There is no official documentation or specification which can define the difference between major, minor and full garbage collection. We can define each one of them in the following way:

1. Full garbage collection works on the tenured space.
2. Major garbage collection works on the survivor space.
3. Minor garbage collection works on the Eden space to perform a mark-and-sweep routine.

6) How will you identify major and minor garbage collections in Java?

We can identify the major and minor garbage collections based on the output. If the garbage collection logging is enabled by using -**XX:PrintGCDetails** or **verbose:gc**, the minor collection prints **"GC"**, whereas the major collection prints **"Full GC"**.

7) Differentiate ParNew and DefNew Young Generation garbage collectors.

**ParNew** and **DefNew** both are types of Young Generation garbage collectors. A multi-threaded garbage collector that is used with the concurrent Mark Sweep is referred to as **a ParNew** young generation garbage collector, whereas a single-threaded garbage collector that is used with the serial garbage collector is referred to as a **DefNew** young generation garbage collector.

8) Explain the use of the Finalize() method of the garbage collector.

The **Finalize()** method is called by the garbage collector before collecting any object that is eligible for the garbage collector. The **Finalize()** method is used to get the last chance to object to cleaning and free remaining resource.

9) Can we force the Garbage collector to run at any time?

No, we cannot force Garbage collection in Java. Although, we can request it by calling **system.gc()** or its cousin **Runtime.getRunitime().get()**. It's not guaranteed that GC will run immediately as a result of calling these methods.

10) Does Garbage collection occur in permanent generation space in JVM?

Yes, Garbage collection can occur in PermGen space. If PermGen cross a threshold or it is full, it can trigger Full Garbage Collector.

11) When does a Java object become available for garbage collection?

An object becomes available for garbage collection when:

1. It is marked as null.
2. It goes out of scope.
3. Within an application, if it is no longer referenced by any non-null objects.

12) What do you mean by mark-and-sweep?

**Mark** and **Sweep** are the two states of garbage collection. In the **Mark** stage, JVM identifies whether an object is still needed or not. The object is marked for garbage collection when the object is not needed.

In the **Sweep** stage, JVM performs memory reclamation and garbage collection algorithms.

13) Explain the difference between Minor, Major, and Full garbage collection.

There is no official documentation that can define the difference between **Minor, Major,** and **Full** garbage collection. However, it is commonly understood that:

[Learn more](https://aax-fe-sin.amazon-adsystem.com/x/c/RK0YubMRxFWbAmY8cpbSJfkAAAGF_rVVTAUAAAalBAAzcHhfdHhuX2JpZDEgICBOL0EgICAgICAgICAgICByFwjR/https:/www.jockey.in/athleisure/sweatshirt-men?utm_source=AmazonDSPSS&utm_medium=Video&utm_campaign=IA_Jockey_AmazonDSPSS_ElevatedEssentials2_Video_Views_Male_IM11_Jan_2023" \t "_blank)

1. The **Mark** and **Sweep** stages of garbage collection are performed on the **Eden space** in **Minor** garbage collection.
2. The **Major** garbage collection works in the **survivor space**.
3. The **Full** garbage collection works on the **tenured space**.

14) What is a memory leak, and how does it affect garbage collection?

A situation where a garbage collector fails to identify and remove an unused object from memory is referred to as a memory leak.

The memory consumption is increased by the memory leak. After increasing the memory consumption, JVM is forced to clear more space for new objects. **Mark** and **Sweep** stages of garbage collection run more frequently. It free up less memory each time these stages run until there is no heap space left.

15) Can we trigger garbage collection from the code?

Yes, we can trigger the garbage collection by issuing a request to the JVM. We use the System.gc() command for issuing the request. However, there is no guarantee on when the JVM will respond to such a request.

16) Which part of the memory from Stack or Heap is involved in GC?

A heap part of the memory is involved in GC.

17) Explain the responsibilities of GC.

The main responsibility of garbage collection is to free up unused memory. GC ensures that the available memory will be used efficiently but does not guarantee that there will be enough memory to run the program.

18) What is a daemon thread? Is GC a daemon thread?

A thread which runs behind the application for performing background operations is referred to as a **daemon** thread.

Yes, GC is a daemon thread which starts by JVM.

19) Explain the ways for making an object eligible for GC when it is no longer needed?

There are the following three ways to make an object eligible for GC:

**By setting all available object references to null**

When the purpose of creating an object is achieved, we can set the reference of this object to null to make it available for GC.

1. **public** **class** GCExample1 {
2. // main() method start
3. **public** **static** **void** main (String [] args){
4. String strData = "Garbage Collector Example by javaTpoint.";
5. // strData is not eligible for garbage collector yet.
6. strData = **null**;          // set reference of String object to null
7. // strData is now eligible for garbage collector
8. }
9. }

**By making reference variable for referring to another object**

Decouple the reference variable from the object and set it refer to another object, so the object which it was referring to before reassigning is eligible for Garbage Collection.

1. publc **class** GCExample2 {
2. // main() method start
3. **public** **static** **void** main(String [] args){
4. String strData1 = "Garbage Collector Example String1 by javaTpoint.";
5. String strData2 = "Garbage Collector Example String2 by javaTpoint.";
6. // print strData1
7. System.out.println(strData1);
8. //strData1 is not eligible for Garbage Collector yet
9. strData1 = strData2;
10. /\* Now the strData1 variable that refers to the String object "Garbage Collector Example2 String by javaTpoint." and the object "Garbage Collector Example String1 by javaTpoint." is not referred by any variable and hence is eligible for Garbage Collector\*/
11. }
12. }

**By creating Islands of Isolation**

When we have two instance variables referring to the instance of the same class, and these two variables also refer to each other and the object referred by these variables don't have any other valid reference, these two objects are eligible for GC.

1. **public** **class** GCExample3 {
2. GCExample3 obj1;
3. // main method start
4. **public** **static** **void** main(String [] str){
5. // instance reference variable
6. GCExample3 obj2 = **new** GCExample3();
7. GCExample3 obj3 = **new** GCExample3();
8. obj2.obj1 = obj3; //gc1 refers to gc2
9. obj3.obj1 = obj2; //gc2 refers to gc1
10. // set null to obj2 and obj3
11. obj2 = **null**;
12. obj3 = **null**;
13. //obj2 and obj3 refer to each other and have no other valid references,
14. //so, obj2 and obj3 are eligible for Garbage collection here
15. }
16. }

20) What is the advantage of the Garbage collection?

The main advantage of the automatic garbage collection in Java is that it removes the burden of manual memory allocation and deallocation from us so that we can focus on problem solving.

21) What is the purpose of overriding finalize() method?

The finalize() method in Java also called finalizer is a method defined in java.lang.Object. It is called by Garbage collector just before collecting any object which is eligible for GC. Thus finalize() method provides last chance to object to do cleanup and free any remaining resource.

22) What are the different ways to make an object eligible for GC when it is no longer needed?

1. Set all available object references to null
2. Make the reference variable to refer to another object
3. Creating Islands of Isolation

23) How many times does the garbage collector calls the finalize() method for an object?

Only once.

24) is garbage collector a foreground or background thread?

Garbage collector is a background daemon thread. A daemon thread runs behind the application that is started by the JVM. The thread stops when all non-daemon foreground thread stops.

25) What is garbage collection time?

The more live objects are found, the longer the suspension, which has a direct impact on response time and throughput. This fundamental tenet of garbage collection and the resulting effect on application execution is called the garbage-collection pause or GC pause time.

https://www.theserverside.com/feature/Java-garbage-collection-interview-questions-and-answers

**1. Why is garbage collection necessary in Java?**

In many programming languages, such as [C and C++](https://www.theserverside.com/opinion/Is-Java-slow-Compared-to-C-its-faster-than-you-think), when an object is no longer needed by a program, the developer must take programmatic steps to reclaim any space the object was allocated in memory.

This approach can be incredibly efficient when implemented properly. However, history has shown that when this process is done poorly, [memory leaks](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Fix-Java-memory-leaks-JVM-heap-dumps-Recorder-Mission-Control) can occur and [crash an application](https://www.theserverside.com/post/The-benefits-and-drawbacks-of-Javas-fail-safe-iterators).

When the Java language was created, Sun engineers decided that developers should not be responsible for managing the memory used by the objects they create. Instead, a garbage collection routine would be part of the JVM; this routine identifies objects that are no longer in use and deletes them from memory.

**2. When does a Java object become available for garbage collection?**

An object becomes available for garbage collection when it is marked as null, goes out of scope or is no longer referenced by any non-null objects within an application. In simple terms, a Java object becomes available for garbage collection when it is no longer in use by the application.

**3. What does mark-and-sweep mean?**

You can break garbage collection in Java into two major stages. The first is the mark stage, where the JVM looks at every object in memory and identifies whether it is still needed or not. If the object is not needed, it is marked for garbage collection.

The sweep is the second stage, where the JVM performs the garbage collection and memory reclamation.

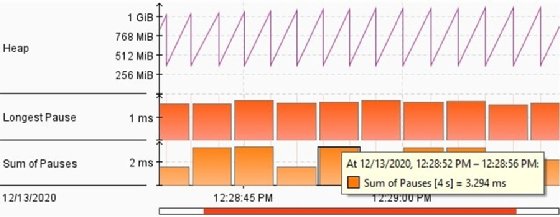
Garbage collection algorithms that employ this sequence of events are known as mark-and-sweep garbage collectors.

**4. What is the drawback to garbage collection?**

The primary drawback to garbage collection is that it freezes all active threads when the memory reclamation phase takes place.

A full garbage collection cycle will run for several seconds -- or potentially even several minutes. Furthermore, garbage collection routines can't be scheduled. Imagine you're running a high-volume trading program. Now imagine a garbage collection routine happening two minutes before the stock market closes. A stop-the-world event on the JVM at that moment in time would lead to a large number of unhappy application users.

Poorly timed garbage collection can make an enterprise system look unpredictable and unreliable. Understandably, a great deal of work has been done in the Java garbage collection arena in order to minimize the impact a Java garbage collection cycle has on active systems.



TECHTARGET

Most Java garbage collection interview questions include the topic of JVM pause times and the significance of stop-the-world events.

**5. What is generational garbage collection?**

The JVM splits allocated memory into four separate spaces:

* eden
* survivor
* tenured
* metaspace

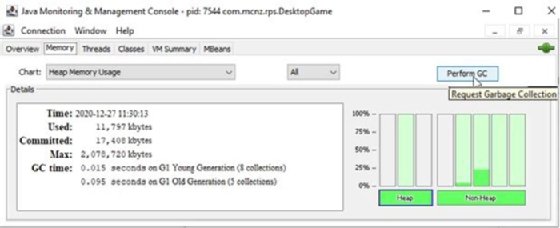
Low-level JVM components, such as the string buffer and compiled classes, are allocated memory in the metaspace. This space goes relatively unchanged over time. When people talk about garbage collection, the focus is typically on the eden, survivor and tenured spaces.

When an object is first created, it is placed in the eden space. If garbage collection occurs and the object is still referenced, it gets moved to the survivor space. If enough garbage collections happen and an object in the survivor space never gets collected, it is then moved to the tenured space.

Eden, survivor and the tenured space are all garbage collected separately, with eden collected the most often and the tenured space collected the least. This helps to improve performance, as the weak generational hypothesis tells us that long-lived objects are likely to remain active, making an inspection of their garbage collection eligibility a waste of time.

Furthermore, objects in the eden space are more likely to be short-lived and eligible for removal, so a scan of the eden space is more likely to free up a large block of memory.

Division of the garbage collector into eden, survivor, tenured and the metaspace areas greatly improves JVM performance.

JConsole provides insights on the time required to garbage collect young (eden + survivor) and old (tenured) sections of the Java heap.

**6. What's the difference between a minor, major and full garbage collection?**

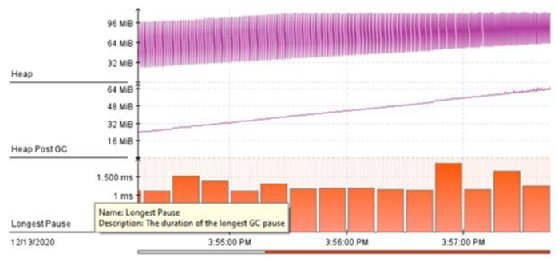
There is no official specification that defines how a major, minor and full garbage collection cycle differ. However, it is commonly understood that:

* A minor garbage collection does a mark-and-sweep routine on the eden space.
* A major garbage collection works on the survivor space.
* A full garbage collection works on the tenured space.

Since an event that triggers a full garbage collection will normally trigger a sweep of the eden, survivor and metaspace, a full garbage collection cycle is often said to include these areas of the Java heap as well.

**7. How does a Java memory leak affect garbage collection?**

A memory leak increases memory consumption, and the JVM is forced to run more often to clear space for new objects. Garbage collection routines will run more frequently, and free up less memory each time they run, until eventually there is no heap space left.

Java garbage collection interview questions often require the interviewee to answer questions about memory leaks, as seen profiled here in Java Mission Control.

**8. When would you choose the parallel garbage collector (GC) over Concurrent Mark Sweep (CMS) or the G1 garbage collector?**

The G1 garbage collector works best when a system can dedicate a large amount of memory to the heap.

CMS uses extra threads and processing power to perform garbage collection routines without any perceived impact on application performance. It also works best with heaps smaller than 32 GB in size.

If a system does not have an extensive amount of memory dedicated to the heap or surplus processing power to allocate to CMS, a simple parallel GC is the correct choice.

Also, a parallel GC will often collect more garbage over a given period when compared to other algorithms. However, the tradeoff is longer stop-the-world pauses. If pause times are not a concern, parallel garbage collection can be the best choice.

**9. Can you trigger garbage collection from code?**

The System.gc() command can issue a request to the JVM to prioritize garbage collection, but the non-deterministic nature of the garbage collection algorithms means there is no guarantee on when the JVM will respond to such a request.

Common wisdom is to avoid the System.gc() command in code and find other ways to configure the JVM's Java garbage collection algorithms to achieve optimal memory management performance.

However, it's worth mentioning that while the specification says the JVM may ignore a call to System.gc(), the reality is that no current implementation does unless specifically configured to do so. "And even then, there is a tooling path that ignores that config," says Java performance expert [Kirk Pepperdine](http://www.kodewerk.com/about.html).

**10. What strategy can you use to minimize the impact of stop-the-world garbage collection routines in enterprise systems?**

One strategy is to cluster your server and assign more than enough memory to the Java heap than any individual cluster member would ever consume in a day.

Then, during non-peak hours, take one cluster member offline at a time, allowing the other members to handle the workload. At that point, either restart the JVM or force a Java garbage collection with the Java Diagnostic Command (JCMD):

C:>jdk11\bin\jcmd GC.run

When a tool like [Java Mission Control](https://www.theserverside.com/definition/Java-Mission-Control) or JConsole verifies that garbage collection successfully occurred, have the server rejoin the cluster and then perform the same steps on other cluster members.

Memory management is an important part of managing the runtime of Java-based applications and microservices. Those interested in landing a job that involves [Java performance tuning](https://www.theserverside.com/feature/Use-this-Java-performance-tuning-guide-to-optimize-your-JVM) need to be able to answer these Java garbage collection interview questions.

https://javarevisited.blogspot.com/2012/10/10-garbage-collection-interview-question-answer.html#axzz7ro8HGtlp

**Question 1 - What is the structure of Java Heap? What is Perm Gen space in Heap?**

Answer: In order to better perform in Garbage collection questions in any Java interview, It’s important to have a basic understanding of  Java Heap space. To learn more about the heap, see my post [10 points on Java heap space](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html).   
  
By the way, Heap is divided into different generations e.g. new generation, old generation, and PermGen space.PermGen space is used to store class’s metadata and filling of PermGen space can cause [java.lang.OutOfMemory: PermGen space](http://javarevisited.blogspot.sg/2012/01/tomcat-javalangoutofmemoryerror-permgen.html). It's also worth noting to remember the [JVM option to configure PermGen](http://javarevisited.blogspot.sg/2011/09/javalangoutofmemoryerror-permgen-space.html) space in Java.

**Question 2 - How do you identify minor and major garbage collections in Java?**

Answer: Minor collection prints “GC” if garbage collection [logging](http://javarevisited.blogspot.sg/2011/05/top-10-tips-on-logging-in-java.html) is enabled using –verbose:gc or -XX:PrintGCDetails, while Major collection prints “Full GC”. This Garbage collection

 interview question is based on an understanding of the Garbage collection output. As more and more Interviewer is asking a question to check the candidate’s ability to understand GC output, this topic becomes even more important.

**Question 3 - What is the difference between ParNew and DefNew Young Generation Garbage collector?**

Answer: This *Garbage Collection interview question* is recently asked by one of my friends. It requires more than average knowledge on GC to answer this question.   
  
By the way, ParNew and DefNew are two young generation garbage collector. ParNew is a multi-threaded GC used along with concurrent Mark Sweep while DefNew is a single-threaded GC used along with Serial Garbage Collector.

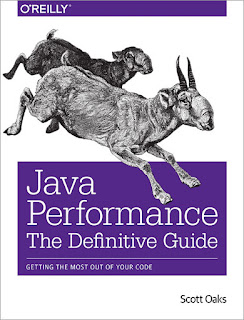
And, if you are serious about improving your advanced JVM skill and learn things like taking and analyzing heap dumps then highly recommend you to join these [Java performance courses](https://javarevisited.blogspot.com/2019/04/top-5-courses-to-learn-jvm-internals.html) on Udemy.

**Question 4 - How do you find GC resulted due to calling System.gc()?**

Answer: Another GC interview question is based on GC output. Similar to the major and minor collection, there will be the word “System” included in the Garbage collection output.

**Question 5 - What is the difference between Serial and Throughput Garbage collectors?**

Answer: Serial Garbage collector is a stop the world GC which stops application thread from running during both [minor and major collection](http://javarevisited.blogspot.sg/2011/04/garbage-collection-in-java.html). Serial Garbage collector can be enabled using JVM option -XX:UseSerialGC and it's designed for Java application which doesn't have pause time requirement and has client configuration.   
 **The Serial Garbage collector** also defaulted GC in JDK 1.4 before ergonomics was introduced in JDK 1.5. Serial GC is most suited for small applications with fewer [threads](http://javarevisited.blogspot.sg/2011/02/how-to-implement-thread-in-java.html) while throughput GG is more suited for large applications. On the other hand Throughput garbage collector is a parallel collector where minor and major collection happens in parallel taking full advantage of all the system resources available like multiple processors.   
  
Though both major and minor collection runs on stop-the-world fashion and introduced pause in the application. Throughput Garbage collector can be enable using -XX:UseParallelGC or -XX:UseOldParallelGC.   
  
It increases the overall throughput of application my minimizing time spent in Garbage collection but still has long pauses during full GC.   
  
This is a kind of *Garbage collection interview question* that gives you an opportunity to show your knowledge in detail while answering. I always suggest answering these kinds of questions in detail.  See [Java Performance The Definitive Guide](https://javarevisited.blogspot.com/2018/07/top-5-java-performance-tuning-books-for.html) for more details on this topic.

[](https://www.java67.com/2019/08/best-books-to-learn-java-virtual-machine-in-depth.html)

**Question 6 – When does an Object become eligible for Garbage collection in Java?**

Answer: An object becomes [eligible for garbage collection](http://javarevisited.blogspot.sg/2011/04/garbage-collection-in-java.html) when there is no live reference for that object or it can not be reached by any live thread. The cyclic reference doesn’t count as a live reference and if two objects are pointing to each other and there is no live reference for any of them, then both are eligible for GC. Also Garbage collection thread is a [daemon thread](http://javarevisited.blogspot.sg/2012/03/what-is-daemon-thread-in-java-and.html) that will run by JVM based upon the GC algorithm and when runs it collects all objects which are eligible for GC.

**Question 7 - What is finalize method in Java? When does Garbage collector calls finalize method in Java?**

Answer: Finalize method in Java also called finalizer is a method defined in java.lang.Object and called by Garbage collector before collecting any object which is eligible for GC. Finalize() method provides the last chance to object to do cleanup and free any remaining resource, to learn more about finalizers, read [What is finalize method in Java](http://javarevisited.blogspot.sg/2012/03/finalize-method-in-java-tutorial.html).

**Question 8 - If Object A has reference to Object B and Object B refer to Object A, apart from that there is no live reference to either object A or B, Does they are eligible for Garbage collection?**

This Garbage collection interview questions is related question 5 “When object become eligible for Garbage collection”. An object becomes eligible for Garbage collection if there is no live reference for it. It can not be accessible from any Thread and cyclic dependency doesn’t prevent Object from being Garbage collected. This means in this case both Object A and Object B are eligible of Garbage collection. See [How Garbage collection works in Java](http://javarevisited.blogspot.com/2011/04/garbage-collection-in-java.html) for more details.

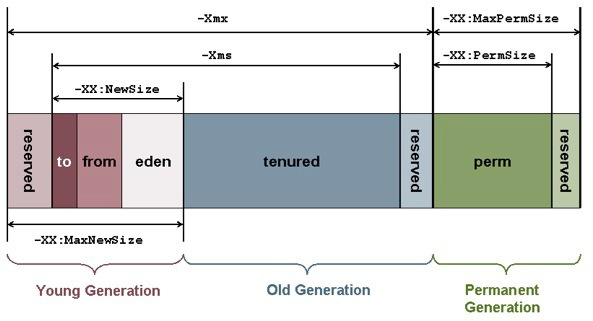
**Question 9 -Can we force the Garbage collector to run at any time?**

Answer: No, you can not force Garbage collection in Java. Though you can request it by calling Sytem.gc() or its cousin Runtime.getRuntime().gc(). It’s not guaranteed that GC will run immediately as a result of calling these methods.

**Question 10 - Does Garbage collection occur in permanent generation space in JVM?**

Answer : This  is a tricky Garbage collection interview question as many programmers are not sure whether PermGen space is part of [Java heap space](http://javarevisited.blogspot.sg/2011/08/increase-heap-size-maven-ant.html) or not and since it maintains class Meta data and String pool, whether its eligible for garbage collection or not.

By the way Garbage Collection does occur in PermGen space and if PermGen space is full or cross a threshold, it can trigger Full GC. If you look at the output of GC you will find that PermGen space is also garbage collected. This is why correct sizing of PermGen space is important to avoid frequent full GC. You can control size of PermGen space by [JVM options](http://javarevisited.blogspot.sg/2011/11/hotspot-jvm-options-java-examples.html) -XX:PermGenSize and -XX:MaxPermGenSize.

[](https://medium.com/javarevisited/7-best-courses-to-learn-jvm-garbage-collection-and-performance-tuning-for-experienced-java-331705180686)

**Question 11 : How to you monitor garbage collection activities?**

Answer: One of my favorite interview questions on [Garbage collection](https://www.blogger.com/javarevisited.blogspot.in/2011/04/garbage-collection-in-java.html), just to check whether the candidate has ever monitored GC activities or not. You can monitor garbage collection activities either offline or real-time. You can use tools like **JConsole** and **VisualVM** VM with its Visual GC plug-in to monitor real time garbage collection activities and memory status of JVM or you can redirect Garbage collection output to a log file for offline analysis by using -XlogGC=&lt;PATH&gt; JVM parameter. Anyway you should always enable GC options like -XX:PrintGCDetails -X:verboseGC and -XX:PrintGCTimeStamps as it doesn't impact [application performance](http://javarevisited.blogspot.sg/2012/01/improve-performance-java-database.html) much but provide useful states for performance monitoring.

**Question 12: Look at below Garbage collection output and answer following question :**

[GC

       [ParNew: 1512K->64K(1512K), 0.0635032 secs]

       15604K->13569K(600345K), 0.0636056 secs]

       [Times: user=0.03 sys=0.00, real=0.06 secs]

 1. Is this output of Major Collection or Minor Collection ?

 2. Which young Generation Garbage collector is used ?

 3. What is size of Young Generation, Old Generation and total Heap Size?

 4. How much memory is freed from Garbage collection ?

 5. How much time is taken for Garbage collection ?

 6. What is current Occupancy of Young Generation ?

This Garbage collection Interview questions is completely based on GC output. Following are answers of above GC questions which will not only help you to answer these question but also help you to understand and interpret GC output.

**Answer 1**:  It's Minor collection because of "GC" word, In case of Major collection, you would see "Full GC".

**Answer 2**: This output is of multi-threaded Young Generation Garbage collector "ParNew", which is used along with CMS concurrent Garbage collector.

**Answer 3**: [1512K] which is written in bracket is total size of Young Generation, which include Eden and two survivor space. 1512K on left of arrow is occupancy of Yong Generation before GC and 64K is occupancy after GC. On the next line value if bracket is total heap size which is (600345K). If we subtract size of young generation to total heap size we can calculate size of Old Generation. This line also shows occupancy of heap before and after Garbage collection.

**Answer 4**: As answered in previous garbage collection interview question, second line shows heap occupancy before and after Garbage collection. If we subtract value of right side 13569K, to value on left side 15604K, we can get total memory freed by GC.

**Answer 5**: 0.0636056 secs on second line denotes total time it took to collect dead objects during Garbage collection. It also include time taken to GC young generation which is shown in first line (0635032 secs).

**Answer 6**: 64K

Here is few more interesting *Garbage collection Interview question* for your practice, I haven’t provided answers of all garbage collection interview questions. If you know the answer than you can post via comments.

**Question -  What is the difference between -XX:ParallelGC and -XX:ParallelOldGC?**

Question - When do you ConcurrentMarkSweep Garbage collector and Throughput GC?

Question -  What is difference between ConcurrentMarkSweep and G1 garbage collector?

Question -  Have you done any garbage collection tuning? What was your approach**?**

Read more: <https://javarevisited.blogspot.com/2012/10/10-garbage-collection-interview-question-answer.html#ixzz7ro8cWqUh>

https://java-questions.com/garbagecollection-interview-questions.html

**Q1) Which part of the memory is involved in Garbage Collection? Stack or Heap?**

**Ans)** Heap

**Q2)What is responsiblity of Garbage Collector?**

**Ans)** Garbage collector frees the memory occupied by the unreachable objects during the java program by deleting these unreachable objects.  
It ensures that the available memory will be used efficiently, but does not guarantee that there will be sufficient memory for the program to run.

**Q3) Is garbage collector a daemon thread?**

**Ans)** Yes GC is a daemon thread. A daemon thread runs behind the application. It is started by JVM. The thread stops when all non-daemon threads stop.

**Q4)How is Garbage Collection managed?**

**Ans)**The JVM controls the Garbage Collector; it decides when to run the Garbage Collector. JVM runs the Garbage Collector when it realizes that the memory is running low. The behavior of GC can be tuned by passing parameters to JVM. One can request the Garbage Collection to happen from within the java program but there is no guarantee that this request will be taken care of by jvm.

**Q5) When does an object become eligible for garbage collection?**

**Ans)** An object becomes eligible for Garbage Collection when no live thread can access it.

**Q6) What are the different ways to make an object eligible for Garbage Collection when it is no longer needed?**

Ans)

* **Set all available object references to null** once the purpose of creating the object is served :
* public class GarbageCollnTest1 {
* public static void main (String [] args){
* String str = "Set the object ref to null";
* //String object referenced by variable str is not eligible for GC yet
* str = null;
* /\*String object referenced by variable str becomes eligible for GC \*/
* }

}

* **Make the reference variable to refer to another object** : Decouple the reference variable from the object and set it refer to another object, so the object which it was referring to before reassigning is eligible for Garbage Collection.
* publc class GarbageCollnTest2 {
* public static void main(String [] args){
* String str1 = "Garbage collected after use";
* String str2 = "Another String";
* System.out.println(str1);
* //String object referred by str1 is not eligible for GC yet
* str1 = str2;
* /\* Now the str1 variable referes to the String object "Another String" and the object "Garbage collected after use" is not referred by any variable and hence is eligible for GC \*/
* }

}

* **Creating Islands of Isolation**: If you have two instance reference variables which are referring to the instances of the same class, and these two reference variables refer to each other and the objects referred by these reference variables do not have any other valid reference then these two objects are said to form an Island of Isolation and are eligible for Garbage Collection.
* public class GCTest3 {
* GCTest3 g;
* public static void main(String [] str){
* GCTest3 gc1 = new GCTest3();
* GCTest3 gc2 = new GCTest3();
* gc1.g = gc2; //gc1 refers to gc2
* gc2.g = gc1; //gc2 refers to gc1
* gc1 = null;
* gc2 = null;
* //gc1 and gc2 refer to each other and have no other valid //references
* //gc1 and gc2 form Island of Isolation
* //gc1 and gc2 are eligible for Garbage collection here
* }

}

**Q7) Can the Garbage Collection be forced by any means?**

Ans)No. The Garbage Collection can not be forced, though there are few ways by which it can be requested there is no guarantee that these requests will be taken care of by JVM.

**Q8) How can the Garbage Collection be requested?**

Ans) There are two ways in which we can request the jvm to execute the Garbage Collection.

* The methods to perform the garbage collections are present in the Runtime class provided by java. The Runtime class is a Singleton for each java main program. The method getRuntime() returns a singleton instance of the Runtime class. The method gc() can be invoked using this instance of Runtime to request the garbage collection.
* Call the System class System.gc() method which will request the jvm to perform GC.

**Q9) What is the purpose of overriding finalize() method?**

Ans) The finalize() method should be overridden for an object to include the clean up code or to dispose of the system resources that should to be done before the object is garbage collected.

**Q11) How many times does the garbage collector calls the finalize() method for an object?**

Ans) Only once.

**Q12) What happens if an uncaught exception is thrown from during the execution of the finalize() method of an object?**

Ans) The exception will be ignored and the garbage collection (finalization) of that object terminates.

**Q13) What are different ways to call garbage collector?**

Ans) Garbage collection can be invoked using **System.gc() or Runtime.getRuntime().gc()**.

**Q14) How to enable/disable call of finalize() method of exit of the application**

Ans) **Runtime.getRuntime().runFinalizersOnExit(boolean value)** . Passing the boolean value will either disable or enable the finalize() call.