Top 50 Garbage collection interview Questions and answers for experienced and freshers

Garbage collection is very important topic in java interview.
Interviewers always have great focus on **garbage collection**. So, I have tried to covered this in 50 questions which could be framed from garbage collection. To answers these questions in comprehensive manner I have given programs and detailed explanation for for each and every question.
These questions will be very handy for **fresh learners**, **experienced and architect level java developers**. Also read about **JVM (java virtual machine)** and Top 30 JVM(Java Virtual Machine) interview Questions and answers

Question 1. Where are objects created in memory? On stack or heap?

Answer. Let's start garbage collection interview question with very basic question. All Java objects are always created on **heap** in java.

Question 2. What is **Garbage Collection** process in java?

Answer. Basic garbage collection interview question. Definitely all developers must know it :)
GC (Garbage collection) is the process by which JVM cleans objects (unused objects) from heap to reclaim heap space in java.

Question 3. What is **Automatic Garbage Collection** in JVM heap memory in java?

Answer. Very important thing you must know in garbage collection interview.
Automatic garbage collection is the process of

- **Identifying objects which are in use** in java heap memory and
- **Which objects are not in use** in java heap memory and

- **deleting the unused objects** in java heap memory.

Question 4. How to **Identify objects which are in use** in JVM heap memory in java?

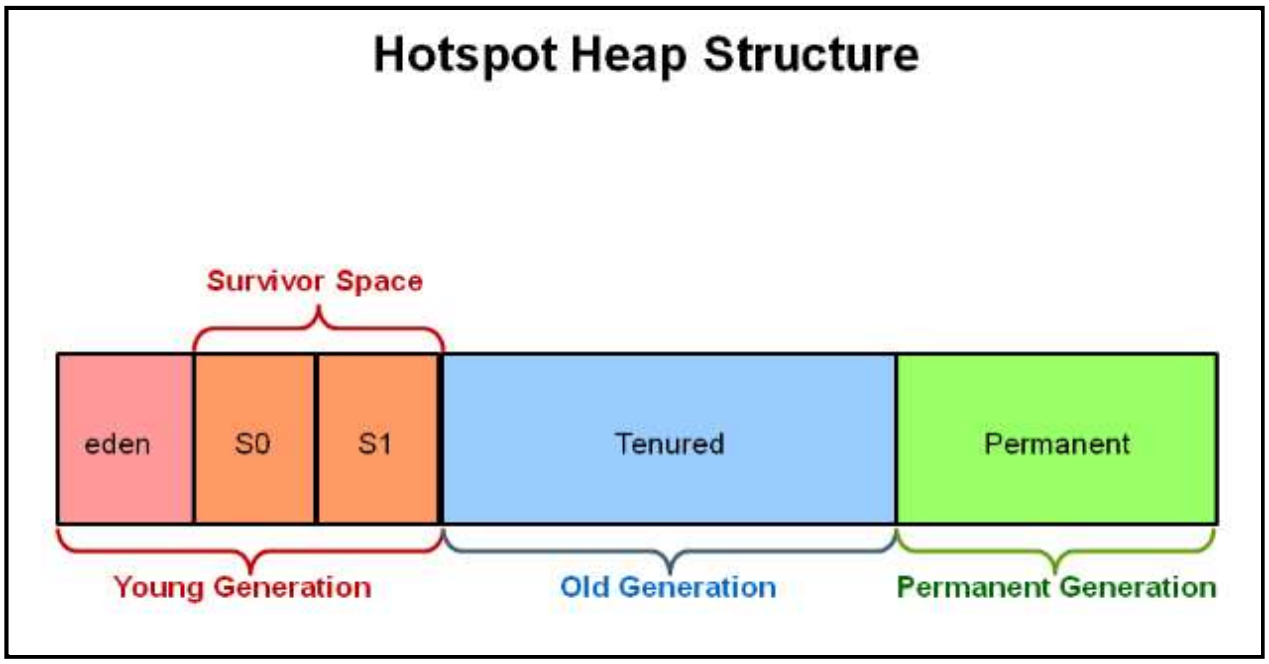
Answer. It is very **basic** garbage collection interview question.
Objects in use (or **referenced objects**) are those objects which are still needed by java program, some part of java program is still pointing to that object.

Question 5. Which **objects are not in use** in JVM heap memory in java?

Answer. Another very **basic** garbage collection interview question.
Objects not in use (or **unreferenced objects**) are those objects which are not needed by java program, no part of java program is pointing to that object.
So, these unused objects can be cleaned in GC (garbage collection) process and memory used by an unreferenced object can be reclaimed.

Question 6. Explain JVM Heap memory (Hotspot heap structure) with diagram in java?

Answer. A **very very important** garbage collection interview question for **freshers**, **experienced** software professionals. Architects must know it on finger tips.



JVM Heap memory (Hotspot heap structure) in java consists of following elements>

1. **Young** Generation
    - 1a) **Eden**,
    - 1b) **S0 (Survivor** space 0**)**
    - 1c) **S1 (Survivor** space 1**)**
2. **Old** Generation (Tenured)
3. **Permanent** Generation.

So, JVM Heap memory (Hotspot heap structure) is divided into three parts **Young** Generation, **Old** Generation (tenured) and **Permanent** Generation.

Young Generation is further divided into **Eden**, **S0 (Survivor** space 0**)** and **S1 (Survivor** space 1**).**

Read in detail about JVM Heap memory (**Hotspot heap structure**) with **diagram** in java

Question 7. What is **Throughput** in gc(garbage collection) in java ?

Answer. **Basic terms** you should know about garbage collection interview.
In short, Throughput is the **time not spent** in garbage collection (GC) (in percent).

Throughput focuses on maximizing the amount of work by an application in a specific period of time. Examples of how throughput might be measured include >

- The number of transactions completed in a given time.
- The number of jobs that a batch program can complete in an hour.
- The number of database queries that can be completed in an hour.

Question 8. What are **pauses** in gc(garbage collection) in java ?

Answer. Another basic term which is used very often in garbage collection. Pauses is applications pauses i.e. when **application doesn't gives any response** because of garbage collection (GC).

Question 9. What is **Young Generation** in JVM Heap memory in java?

Answer. This is important garbage collection interview question.

New objects are allocated in Young generation.

Young Generation consists of >

- 1a) **Eden**,
- 1b) **S0 (Survivor** space 0**)**
- 1c) **S1 (Survivor** space 1**)**

Young Generation is further divided into three parts **Eden**, **S0 (Survivor** space 0**)** and **S1 (Survivor** space 1**).**

Read in more detail about > What is **Young Generation** in JVM in java

Question 10. What is **Old Generation** in JVM Heap memory in java?

Answer. This is another important garbage collection interview question.

The **Old Generation**(tenured generation) is used for storing the long surviving aged objects (Some of the objects which aren't cleaned up **survive in young generation and gets aged**. Eventually such objects are **moved from young to old generation**).

Major garbage collection *occurs in* **Old Generation.**

At what time (or what age) objects are **moved from young to old generation** in JVM *heap**?***
There is some threshold set for young generation object and when that age is met, the object gets moved to the old generation.

What is **major garbage collection** in java? (We will read about it in detail in upcoming interview questions)
When the old generation fills up, this causes a ***major garbage collection***. Objects are cleaned up from old generation.

Read : **JVM (java virtual machine)** in detail in java and How **Garbage Collection (GC) works internally**

Question 11. What is **Permanent Generation** in JVM Heap memory in java?

Answer. This is third generation in java heap and hence is very important garbage collection interview question. It is complex question and hence experienced software professionals must know it.

Permanent generation (Permgen**) Space contains metadata required by JVM to describe the classes and methods used in the application.**

The permanent generation is included in a **full garbage collection** in java.

The permanent generation space is populated at runtime by JVM based on classes in use in the application.

The permanent generation space also contains **Java SE library classes and methods** in java.

JVM garbage collects those classes when classes are no longer required and space may be needed for other classes in java.
Read in more detail about > What is **Permanent Generation** in JVM in java

Question 12. What is **Minor garbage collection** in JVM Heap memory in java?

Answer. It's important to know about minor garbage collection before interview.

Minor garbage collection *occurs in* **Young Generation**

New objects are allocated in Young generation.

When the young generation fills up, this causes a ***minor garbage collection***.
All the unreferenced (dead) objects are cleaned up from young generation.

When objects are **moved from young to old generation** in JVM *heap**?***
Some of the objects which aren't cleaned up **survive in young generation and gets aged**.
Eventually such objects are **moved from young to old generation**.

What is **Stop the World Event?**
Minor garbage collections are called **Stop the World** events.

All the non-daemon threads running in application are stopped during minor garbage collections (i.e. the application stops for while)**.**
Daemon threads performs minor garbage collection. (Daemon threads are low priority threads which runs intermittently in background for doing garbage collection)**.**

Read in more detail about >What is **Minor garbage collection** in JVM

Question 13. What is **Major garbage collection** in JVM Heap memory in java?

Answer. It's easy garbage collection interview question. Prepare it well.

***Major garbage collection occurs in* Old Generation**

The **Old Generation** is used for storing the long surviving aged objects.

At what time (or what age) objects are **moved from young to old generation** in JVM *heap**?***
There is some threshold set for young generation object and when that age is met, the object gets moved to the old generation during garbage collection in java.

When the old generation fills up, this causes a ***major garbage collection***. Objects are cleaned up from old generation.
Major collection is much slower than minor garbage collection in jvm heap **because it involves all live objects**.

Major garbage collection are **Stop the World Event** in java**?**
Major garbage collections are also called Stop the World events.
All the non-daemon threads running in application are stopped during major garbage collections.
Daemon threads performs major garbage collection.

Major garbage collections **in responsive applications** in java?
Major garbage collections should be minimized for responsive applications because applications must not be stopped for long.

Optimizing Major garbage collections in responsive applications in java?
Selection of appropriate garbage collector for the old generation space affects the length of the "Stop the World" event for a major garbage collection.

Read in more detail about > What is **Major garbage collection** in JVM

Question 14. What is **Full garbage collection** in JVM Heap memory in java?

Answer. You must be able to differentiate between **garbage collection** and **full garbage collection**. Don't mess up between two terms.

Full garbage collection occurs in permanent generation in java

Permanent generation Space contains metadata required by JVM to describe the classes and methods used in the application.

The permanent generation space is populated at runtime by JVM based on classes in use in the application.

JVM garbage collects those classes when classes are no longer required and space may be needed for other classes in java.

Read in more detail about > What is **Full garbage collection** in JVM

Question 15. Mention some of the most important VM (JVM) PARAMETERS **you have used for Young** Generation in JVM Heap memory?

Answer. It is important and challenging garbage collection interview question for experienced developers to specifically identify and answer jvm parameters for young generation.

-Xmn : -Xmn sets the size of young generation.
Example of using **-Xmn** VM (JVM) option in java >
java -Xmn512m MyJavaProgram
For more explanation and example - Read : **-Xmn** JVM parameters

-XX:NewRatio : NewRatio controls the size of young generation.
Example of using -XX:NewRatio VM option in java >
-XX:NewRatio=3 means that the ratio between the young and old/tenured generation is 1:3.

-XX:NewSize - NewSize is **minimum size of young generation** which is allocated at initialization of JVM.

-XX:MaxNewSize - MaxNewSize is the **maximum size of young generation** that JVM can use.

-XX:SurvivorRatio : (for survivor space)
SurvivorRatio can be used to **tune the size of the survivor spaces**, but this is often not as important for performance.
Example of using -XX:SurvivorRatio > **-XX:SurvivorRatio=6 sets the ratio between each survivor space and eden to be 1:6**.

Question 16. Mention some of the most important VM (JVM) PARAMETERS **you have used for Old** Generation (tenured) in JVM Heap memory?

Answer. Another is difficult and challenging garbage collection interview question for experienced developers to specifically identify and answer jvm parameters for old generation.

-XX:NewRatio : NewRatio controls the size of young and old generation.

Example of using -XX:NewRatio, **-XX:NewRatio=3 means that the ratio between the young and old/tenured generation is 1:3.**
For more explanation and example **-XX:NewRatio** JVM parameters

Question 17. Mention some of the most important VM (JVM) PARAMETERS **you have used for Permanent** Generation?

Answer. Another difficult garbage collection interview question for experienced developers to specifically identify and answer jvm parameters for permanent generation.

-XX:PermSize: It's is initial value of Permanent Space which is allocated at startup of JVM.

Example of using -XX:PermSize VM (JVM) option in java >
java -XX:PermSize=512m MyJavaProgram
It will set initial value of Permanent Space as 512 megabytes to JVM

-XX:MaxPermSize: It's maximum value of Permanent Space that JVM can allot up to.

Example of using **-XX:MaxPermSize** VM (JVM) option in java >
java -XX:MaxPermSize=512m MyJavaProgram
It will set maximum value of Permanent Space as 512 megabytes to JVM

For more details - What are **-XX:PermSize** and **-XX:MaxPermSize** with Differences

Question 18. What are different **Garbage collectors available in JVM**?

Answer. Very very important garbage collection interview question. Experienced developers must know about all of them.

Different type of garbage collectors in java>

Serial collector / Serial GC (Garbage collector) in java

Throughput GC (Garbage collector) or **Parallel collector** in java

Concurrent Mark Sweep (CMS) collector / **concurrent low pause garbage collector**

G1 garbage collector / Garbage first collector

PS Scavenge

PS MarkSweep

ParNew collector

Question 19 . What is **Serial collector / Serial GC (Garbage collector) in java**?

Answer. Freshers must know about this garbage collection interview question.

Features of Serial GC (Garbage collector) in java >

- Serial collector is also called **Serial GC** (Garbage collector) in java.
- Serial collector is simply also called **Serial collector** in java.
- Serial GC (Garbage collector) is **rarely used** in java.
- Serial GC is designed **for** the **single threaded environments** in java.
- In Serial GC (Garbage collector) , both **minor and major garbage collections are done serially** by **one thread** in java.
- Serial GC uses a **mark-compact collection method**. The serial garbage collector is the **default** for client style machines in **Java SE 5 and 6**.

When to Use the Serial GC (garbage Collector) in java >

- The Serial GC is the garbage collector of choice for most **applications that do not have low pause time requirements and run on client-style machines**.
- Serial garbage collector is also popular in **environments where a high number of JVMs are run on the same machine**.

Vm (JVM) option for enabling **serial GC (garbage Collector)** in java >

-XX:+UseSerialGC

Example of Passing Serial GC in Command Line for starting jar>

java -Xms256m -Xms512m **-XX:+UseSerialGC** -jar d:\MyJar.jar

Read in lots of detail about **Serial collector / Serial GC (Garbage collector)**

Question 20. What is **Throughput GC (Garbage collector) or Parallel collector in java**?

Answer. Experienced developers must know about this garbage collection interview question.

1. Features of Throughput GC (Garbage collector) in java >

- Throughput collector is also called
  - Throughput GC (garbage collector)
  - ParallelGC (garbage collector)
  - Throughput collector
  - ParallelGC collector
- Throughput garbage collector is the **default garbage collector for JVM** in java**.**
- Throughput garbage collector **uses multiple threads to execute a minor collection** and so reduces the serial execution time of the application in java.

2. When to Use the Throughput GC (Garbage collector) in java >

The Throughput garbage collector should be used when application can **afford low pauses** in java.
And application is running on host with multiple CPU's in java.

3. **Vm** (JVM) option for enabling throughput GC (Garbage collector) in java >

-XX:+UseParallelGC

Example of using **throughput** collector in Command Line for starting jar>
java -Xms256m -Xms512m **-XX:+UseParallelGC** -jar d:\MyJar.jar

With this **Vm** (JVM) option you get a

- **Multi-threaded young** generation garbage collector in java,
- **single-threaded old** generation garbage collector in java and

- **single-threaded compaction** of **old** generation in java.

Vm (JVM) option for enabling **throughput collector** with **n number of threads** in java >

-XX:ParallelGCThreads=<numberOfThreads>

Another Vm (JVM) option for enabling **throughput collector** in java >
-XX:+UseParallelOldGC

4. Goals for Throughput GC (Garbage collector) in java >

- Maximum pause time goal (Highest priority)
- Throughput goal
- Minimum footprint goal (Lowest priority)

5.Read in more **detail** about following features of Throughput GC (Garbage collector) in java

Read : **JVM (java virtual machine)** in detail in java and How **Garbage Collection (GC) works internally**
Top 30 JVM(Java Virtual Machine) interview Questions and answers

Question 21. What is **Concurrent Mark Sweep (CMS) Collector / concurrent low pause collector in java**?

Answer. Very important garbage collection interview question for experienced developers and software architects.

1. Features of Concurrent Mark Sweep (CMS) Collector / concurrent low pause collector in java >

- Concurrent Mark Sweep Collector is also called
  - concurrent low pause collector
  - concurrent low pause GC (garbage collector)

  - CMS GC (garbage Collector)
  - CMS Collector

  - concurrent low pause collector
  - concurrent low pause GC (garbage collector)

- Concurrent Mark Sweep (CMS) collector **collects the old/tenured generation** in java**.**
- Concurrent Mark Sweep (CMS) Collector **minimize the pauses** by doing most of the **garbage collection work concurrently with the application threads** in java**.**
- Concurrent Mark Sweep (CMS) Collector **on live objects >**

  Concurrent Mark Sweep (CMS) Collector **does not copy or compact the live objects**. A garbage collection is done **without moving the live objects**. If fragmentation becomes a problem, allocate a larger heap in java**.**

2. When to Use the Concurrent Low Pause Collector in java

- Concurrent Low Pause Collector should be used if your **applications that require low garbage collection pause times** in java**.**
- Concurrent Low Pause Collector should be used when your **application can** afford to **share processor resources with the garbage collector while** the **application is running** in java**.**
- Concurrent Low Pause Collector is beneficial to applications which have a relatively **large set of long-lived data** (a large tenured generation) and run on machines with **two or more processors** in java**.**

Examples when to use Concurrent Mark Sweep (CMS) collector / concurrent low pause collector should be used for >

Example 1 - Desktop UI application that **respond to events**,
Example 2 - Web server responding to a request and
Example 3 - Database responding to queries.

3. Vm (JVM) option for enabling Concurrent Mark Sweep (CMS) Collector in java >

-XX:+UseConcMarkSweepGC

Example of using Concurrent Mark Sweep (CMS) collector **/ concurrent low pause collector** in Command Line for starting jar>
java -Xms256m -Xms512m **-XX:+UseConcMarkSweepGC** -jar d:\MyJar.jar

4. Concurrent Mark Sweep (CMS) Collector / concurrent low pause collector working in detail in java >

As mentioned above Concurrent Mark Sweep (CMS) collector **collects the old/tenured generation (i.e.** performs ***Major garbage collection*** process**).**

5. Heap Structure for CMS garbage Collector

CMS garbage collectors didies heap into three sections: **young** generation, **old** generation, and **permanent** generation of a fixed memory size.

Young Generation is further divided into **Eden**, **S0 (Survivor** space 0**)** and **S1 (Survivor** space 1**).**

5.4.6. **Detailed Steps** in GC (garbage collection) cycle in Concurrent Mark Sweep (CMS) Collector / concurrent low pause garbage collector in java >

Young Generation GC (garbage Collection) in java

- **Live objects** are **copied** from the **Eden space and survivor space** to the **other survivor space**.
- Any **older objects** that have reached their aging threshold are **promoted to old generation**.

After Young generation GC (garbage Collection) in java

- After a young GC, the **Eden space and one of the survivor spaces is cleared**.
- promoted objects (**older objects** that have reached their aging threshold in young GC) are are **available in old generation**.

Old Generation GC (garbage Collection) **with CMS** in java

1. **Initial mark** phase **-** (**First pause** happens/ stop the world event) **- mark live/reachable objects** (Example **-** objects on thread stack, static objects etc.) and elsewhere in the **heap** (Example **-** the young generation).

2. **Concurrent marking** phase **- (**No pause phase) **-** finds **live objects** while the application continues to execute.

3. **Remark -** (**Second pause** happens/ stop the world events) - It **finds objects** that were **missed during the concurrent marking phase due to the concurrent execution of the application threads**.

   Old Generation GC (garbage Collection) **- Sweep phase** (Concurrent Sweep phase) in java

4. **Sweep** phase **-** do the concurrent **sweep**, memory is freed up.
   - Objects that were not marked in the previous phase are deallocated in place.
   - There is no compaction

   Unmarked objects are equal to **Dead Objects.**

   Old Generation GC (garbage Collection) **- After Sweeping**

5. **Reset** phase **-** do the concurrent **reset**.

Read in more **detail** about following features of Concurrent Mark Sweep (CMS) collector / concurrent low pause garbage collector in java

Question 22. What is **G1 Garbage Collector (or Garbage First) in java**?

Answer. Very important garbage collection interview question for senior software developers. But I must say freshers must have at least some knowledge of this garbage collector.

1. The G1 garbage collector **features -**

- G1 garbage collector is also called
  - G1 garbage collector
  - G1 collector
  - G1 GC (garbage collector)
  - Garbage first collector

- G1 garbage collector was **introduced in Java 7**
- G1 garbage collector - default garbage collector in **Java 9**
- G1 garbage collector was designed to replace CMS collector(Concurrent Mark-Sweep garbage Collector).
- G1 garbage collector is **parallel**,
  - G1 garbage collector is **concurrent**, and
  - G1 garbage collector is **incrementally compacting low-pause** garbage collector in java.
- G1 garbage collector has much better layout from the other garbage collectors like serial, throughput and CMS garbage collectors in java.
- G1(Garbage First) collector **compacts sufficiently to completely avoid the use of fine-grained free lists for allocation**, and instead relies on regions.
- G1(Garbage First) collector **allows customizations** by allowing users to specify pause times.
- G1 Garbage Collector (or Garbage First) limits GC **pause times and maximizes throughput.**

2. **Vm** (JVM) option for enabling G1 Garbage Collector (or Garbage First) in java >

-XX:+UseG1GC

Example of using G1 Garbage Collector in Command Line for starting jar>
java -Xms256m -Xms512m **-XX:+UseG1GC** -jar d:\MyJar.jar

3. G1(Garbage First) collector functioning >

CMS garbage collectors divides heap into three sections: young generation, old generation, and permanent generation of a fixed memory size.

All memory objects end up in one of these three sections.

The G1 collector takes a different approach than CMS garbage collector in partitioning java heap memory.

The heap is split/**partitioned** into **many fixed sized regions** (eden, survivor, old generation regions), **but** there is not a **fixed size** for them. This provides **greater flexibility in memory usage.**

4. When to use G1 garbage collector >

G1 must be used when applications that require **large heaps** with limited GC latency.
Example - Application that require

- **heaps around 5-6GB or larger** and
- **pause time** required **below 0.5 seconds**

5. When to switch from CMS (or old garbage collectors) to G1 garbage collector >

Applications using CMS garbage collector may switch to G1 when >

- **Full GC** durations are too **long** or too **frequent**.
- The **rate** of **object allocation** or **promotion varies** significantly.
- **Long garbage collection** (longer than 0.5 to 1 second)

6. The G1(Garbage First) collector working Step by Step >

The G1 collector takes a different approach than CMS garbage collector in partitioning java heap memory.

6.1. G1(Garbage First) garbage collector Heap Structure >

The heap is split/**partitioned** into **many fixed sized regions** (eden, survivor, old generation regions), but there is not a fixed size for them. This provides greater flexibility in memory usage**.**

Each **region's size** is chosen **by JVM at startup**.
Generally heap is divided into **2000 regions** by JVM varying in size from **1 to 32Mb**.

6.2. G1(Garbage First) garbage collector Heap Allocation >

As mentioned above there are following region in heap >
Eden, **survivor** and **old** generation region. Also,
Humongous and unused regions are there in heap.

6.3. Young Generation in G1 garbage collector

Generally heap is divided into **2000 regions** by JVM.

Minimum size of region can be **1Mb** and
Maximum size of region can be **32Mb**.

Regions are not required to be contiguous like CMS garbage collector.

Young GC in G1 garbage collector

- **Live objects** are copied or moved **to survivor regions.**
- If objects aging threshold is met it get promoted to **old** generation regions.
- It is **STW** (stop the world) event. Eden size and survivor size is calculated for the next young GC.
- The young GC is done parallely using multiple threads.

End of a Young GC with G1 garbage collector

At this stage **Live objects have been evacuated (copied or moved) to** >

- **survivor** regions or
- **old** generation regions.

6.4. Old Generation Collection with G1 garbage collector

G1 collector is low pause collector for old generation objects.

Initial Mark -

- It is **STW** (stop the world) event.
- With G1, it is **piggybacked on a normal young GC**. Mark survivor regions (root regions) which may have references to objects in old generation.

Root Region Scanning -

- **Scan survivor regions for references into the old generation**.
- This happens while the **application continues to run**. The phase must be **completed before** a young GC can occur.

Concurrent Marking -

- **Find live objects over the entire heap.**
- This happens while the **application is running**.
- This phase can be interrupted by young generation garbage collections.

Remark (Stop the World Event) -

- **Completes the marking of live object in the heap.**
- Uses an algorithm called **snapshot-at-the-beginning** (SATB) which is much **faster** than algorithm used in the **CMS** collector.

Cleanup (Stop the World Event and Concurrent) -

- **Performs accounting on live objects and completely free regions**. (Stop the world)

- Young generation and old generation are reclaimed at the same time
- Old generation regions are selected based on their liveness.

- **Scrubs** the Remembered Sets. (Stop the world)
- **Reset** the **empty regions** and return them to the free list. (Concurrent)

7. Read in more **detail** about following features of G1 garbage collector / Garbage first collector in java.

Question 23. What are Difference Serial and Throughput gc (garbage Collector) ?

Answer. It is important garbage collection interview question.
Serial collector **uses one thread to execute garbage collection.**
Throughput collector **uses multiple threads to execute garbage collection.**

Serial GC is the garbage collector of choice for applications that do **not** have **low pause time requirements** and run on client-style machines.
Throughput GC is the garbage collector of choice for applications that have **low pause time requirements**.

Question 24. Which methods is called for garbage collection **in** java?

Answer. It is very important garbage collection interview question.
Gc (garbage collector) **calls finalize** method **for garbage collection**.
finalize **method is called only once by garbage collector for an object in java.**
finalize **method is** in java.lang.Object class.

Program to call System.gc() method in Java >

```
public class RunGarbageCollectorExample {
   public static void main(String[] args) {
          System.out.println("About to call garbage collection - using System.gc() method");
          System.gc();
          System.out.println("garbage collection - using System.gc() method called");
   }
}
/*OUTPUT
About to call garbage collection - using System.gc() method
garbage collection - using System.gc() method called
*/
```

Question 25. Can we force early garbage collection **in** java?

Answer. Another garbage collection interview question for experienced developers
Yes it is possible to force early garbage collection in java. But how?

We can ***force early gc*** *(garbage collection) in java* by using following methods >

```
System.gc();
Runtime.getRuntime().gc();

System.runFinalization();
Runtime.getRuntime().runFinalization();
```

Question 26. Is it good practice to call System.gc() in Java?

Answer. *Answer in short is* ***No****, but let's learn why we must not call System.gc() in java >*

- *First of all calling System.gc() does not guarantee that it will immediately start performing garbage collection.*
- *Even calling System.gc() may not do anything. Call to perform garbage collection may be ignored completely.*
- *JVM is different for different platforms because java is platform independent language, so you never know about which garbage collector your jvm will run i.e. what algorithm does it follow to perform garbage collection.*

Read in detail : Is it good practice to call System.gc() in Java?

Question 27. Is garbage collection guaranteed when we call finalize() methods?

Answer. *Answer in short is* ***No.***

By calling finalize method ***JVM runs the finalize() methods of any objects pending finalization*** *i.e. objects which have been discarded but there finalize method is yet to be run. After finalize method is executed JVM reclaims space from all the discarded objects in java.*

Calling finalize methods does ***not guarantee*** *that it will* ***immediately start performing garbage collection****.*

Finalize method execution is not assured - We must not override finalize method to write some critical code of application because methods execution is not assured. Writing some critical code in finalize method and relying on it may make application to go horribly wrong in java.

Question 28. Which thread performs garbage collection **in** java?

Answer. It is simple garbage collection interview question.
Daemon threads are low priority threads which **runs intermittently in background** for doing **garbage collection (gc) in java**.

Question 29. Tell us something about **ParNew collector in** java?

Answer. This is garbage collection interview question for 5 years+ experienced developers only. **ParNew collector** is the young generation collector. It is the parallel copy collector, it uses multiple threads in parallel. Vm parameter for enabling ParNew collector is -XX:+UseParNewGC.

Question 30. Do you know about PS Scavenge and PS MarkSweep in java?

Answer. This is garbage collection interview question for 3 to 5 years+ experienced developers only.

PS Scavenge >

- **PS Scavenge is the Young generation collectors**
- **It is the parallel scavenge collector.**
- **PS Scavenge uses multiple threads in parallel for garbage collection.**
- **Vm parameter for enabling PS Scavenge >**

  -XX:+UseParallelGC

PS MarkSweep >

- **PS MarkSweep is the old generation collector.**
- **PS MarkSweep is the parallel scavenge mark sweep collector.**
- **It uses the multiple threads in parallel for garbage collection.**

- **Vm parameter for enabling PS MarkSweep >**

  -XX:+UseParallelOldGC

Read : PS Scavenge and PS MarkSweep

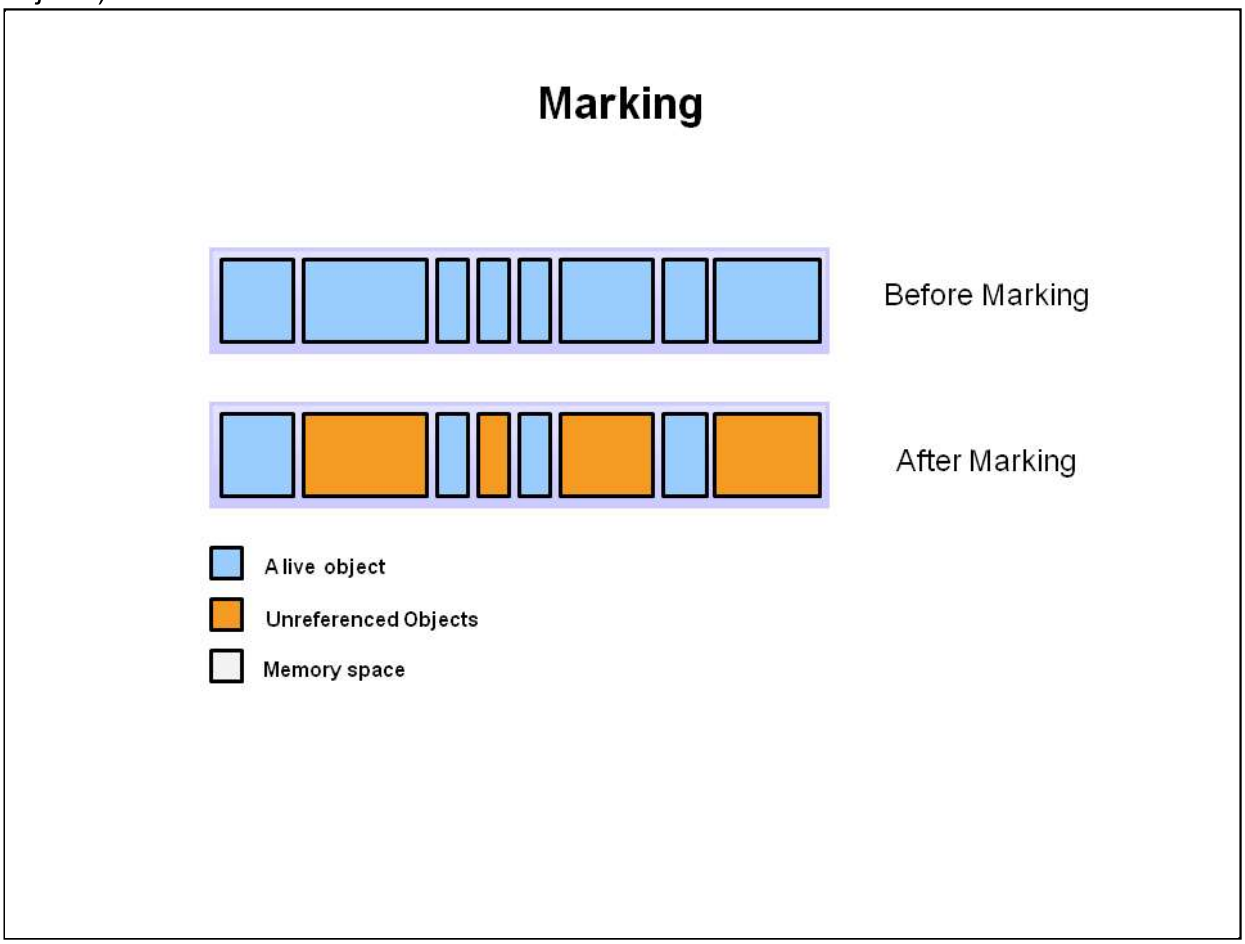
Read : **JVM (java virtual machine)** in detail in java and How **Garbage Collection (GC) works internally**
Top 30 JVM(Java Virtual Machine) interview Questions and answers

Question 31. How garbage collection is done using **Marking and deletion** in java?

Answer. It is very very **important** garbage collection interview question for all java developers.

1) Marking

Marking is a process in which gc (garbage collector) identifies which parts of memory (occupied by objects) are in use and which are not.



Before Marking >

All the objects are shown in **blue,** at this stage

- some of objects might be **in use** (referenced objects) and
- some of objects might **not be in use** (unreferenced objects) **.**

After Marking >

Objects in use (or **referenced objects** or **Alive objects**) are shown in **blue**.

Objects not in use (or **unreferenced objects**) objects are shown in **Orange**.

2) Deletion

Step 2a : Normal Deletion

- **Normal deletion removes all the unreferenced objects** and
- **leaves referenced objects and pointers to free space**.

Step 2b : Deletion with Compacting

Deletion with Compacting is done to improve the performance than normal deleting**.**

- Deletion with Compacting **removes all the unreferenced objects** and
- **compacts the remaining referenced objects** by **moving all the referenced objects together.**

Read in more detail with diagrams about **- Marking and deleting objects** for garbage collection in java **- Mark and sweep algorithm**

Question 32. Mention how to use different garbage collectors by passing vm parameters in java?

Answer. It is very important to know how to use different garbage collectors by passing vm parameters.

1) Vm (JVM) option for enabling **serial GC (garbage Collector)** in java > **-XX:+UseSerialGC**

Example of Passing Serial GC in Command Line for starting jar>

java -Xms256m -Xms512m **-XX:+UseSerialGC** -jar d:\MyJar.jar

2) Vm (JVM) option for enabling **throughput GC (Garbage collector)** in java > **-XX:+UseParallelGC**

Or (throughput collector with n number of threads)
-XX:ParallelGCThreads=<numberOfThreads>
Or
-XX:+UseParallelOldGC

3) Vm (JVM) option for enabling **Concurrent Mark Sweep** (CMS) Collector in java >
-XX:+UseConcMarkSweepGC

Or (CMS garbage collector with n number of threads)
-XX:ParallelCMSThreads=<n>

4) Vm (JVM) option for enabling G1 Garbage Collector
-XX:+UseG1GC

Question 33. Can you make objects eligible for garbage collection in java?

Answer. Another tricky and challenging garbage collection interview question for experienced developers in java.
Yes, we can make object explicitly eligible for garbage collection.
Object which is set explicitly set to **null** becomes **eligible for gc** (garbage collection) in java .
Example 1 >

```
String s="abc"; //s is currently not eligible for gc (garbage collection) in java.
s = null;   //Now, s is currently  eligible for gc (garbage collection) in java.
```

Example 2 >

```
List list =new ArrayList(); //list is currently not eligible for gc (garbage collection).
list = null;   //Now, list is currently  eligible for gc (garbage collection).
```

Question 34. What is **Difference in garbage collection in C/C++ and Java (Hint : In terms of memory allocation and deallocation of objects)?**

Answer. It will test of your awareness and skills to compare java with other languages.
In java garbage collection (memory allocation and deallocation of objects) is an **automatic** process.
But, In C and C++ memory allocation and deallocation of objects) is a **manual** process.

Question 35. Does **variables declared inside block** becomes eligible for gc (garbage collection) **when we exit that block** in

java?

Answer. Yes, All the **variables** declared **inside block** becomes **eligible for gc** (garbage collection) **when** we **exit** that **block** (As scope of those variable is only that block) in java.

Example of garbage collection while using block in java -

```
class MyClass {
    public static void main(String[] args) {
            boolean var = false;
            if (var) { // begin block 1
                    int x = 1; // x is declared inside block
                    //.........
                    //code inside block...
                    //.........
            } // end block 1 //And now x is eligible for gc (garbage collection)
            else { // begin block 2
                    int y = 1;
                    //.........
                    //code inside block...
                    //.........
            } // end block 2 //And now y is eligible for gc (garbage collection)
    }
}
```

Question 36. Have you used **verbose** for understanding garbage collection?

Answer. It is garbage collection interview question for experienced developers and software architects.

- **[GC 325407K->83000K(776768K), 0.2300771 secs]**
  - **GC** - GC indicates **minor Garbage Collection** (i.e. in young generation).
  - 325407K - The combined size of **live objects before** gc(garbage collection).
  - 83000K - The combined size of **live objects after** gc(garbage collection).
  - 0.2300771 secs - **time** it took for gc(garbage collection) to occur.

- .[Full GC 325407K->83000K(776768K), 0.2300771 secs]
  - **Full GC - Full GC Indicates major garbage collection** (i.e. in tenured generation).

Read more about **-verbose:gc**

Question 37. **What** is **monitoring and analyzing** the garbage collection in java?

Answer. It is very important garbage collection interview question for specially 3 years+ experienced developers.

Monitoring and analyzing garbage collection in java can be used to get following information >

1. Understanding the garbage collection process.
2. Understanding how JVM is currently working.
3. What type of garbage collection algorithms are used
4. Improving garbage collection performance,
5. Analyzing Java heap dumps,
6. Monitoring live Java applications,
7. Analyze profiling data,
8. Detecting Memory leak for classes and arrays,
9. Detecting Thread deadlock,
10. Detecting abnormal thread termination,
11. Detecting **outOfMemoryError** problems,
12. Finding System and process CPU utilization thresholds,
13. Find Heap usage thresholds.
14. Find time taken in Garbage collection and Finalizer queue length.

Question 38. **How** to monitor and analyze the garbage collection in java ?

Answer. This is in continuation to above garbage collection interview question. We can use different tools to generate the garbage collection information and then analyze it.

1. **VisualVM** -

   VisualVM is most popular way to generate Thread Dump and is most widely used by developers. It helps us in analyzing threads performance, thread states, CPU consumed by threads etc.
   For more details please read :

2. **JSTACK** - Java stack traces

jstack is very easy way to generate Thread dump and is widely used by developers.

3. **-verbose:gc** VM argument
4. **Jstat** - Java Virtual Machine Statistics Monitoring Tool

   "The **jstat** command displays performance statistics for an instrumented Java HotSpot VM. The target JVM is identified by its virtual machine identifier, or vmid option."

5. **JHAT - Java Heap Analysis Tool.**
6. jconsole - jconsole option can be used to obtain a heap dump.
7. hprof
8. eclipse plugin
9. **HPjmeter**
10. JFR (Java Flight Recorder) can be used for detecting memory leak.

Read in detail : How to monitor and analyze the garbage collection in 10 ways in java

Question 39. What is **memory leak** in java? **Consequences** of memory leak?

Answer. It is must know question especially for **experienced** developers, **freshers** must at least have little information on it.

Memory leak happens when **number of objects(these objects are not needed) created becomes large and time spent in garbage collection increases.**

Ultimately application becomes very slow, non responsive and ends up throwing OutOfmemoryError.

"Memory leaks ends up throwing OutOfmemoryError but OutOfmemoryError doesn't means memory leak in java".

Consequences of memory leak >

- Application becomes very slow.
- Time spent in garbage collection increases.

Question 40. Can you please explain some scenarios where you have faced memory leak, OR scenarios where memory leak could happen in java?

Answer. It is must know question for **experienced** developers, 3 to 5 years developers must know at least three scenarios and 5 years+ developes must know at least 5 scenarios. And for software architects this count must be eight.
Here I have mentioned summary of memory leak points, for detail on each of these points, please refer Detecting and fixing memory leak in java

Scenarios where memory leak can happen in java >

1) Static variables/ fields are not garbage collected and can cause memory leak in java >

Static variables are only garbage collected when the class loader which has loaded the class in which static field is there is garbage collected.

So, be cautious as these static variables can create a memory leak in java.

For more details click here - Static variables are not garbage collected?

2) Thread Local Variables can cause memory leak in java >

A thread local variable is member field in the Thread class.
Such thread local variable can be used to hold the thread state.
But, thread local variable aren't garbage collector till thread is alive.

3) Memory leak while using Autoboxing and unboxing in java >

For addition of numbers we must prefer to use primitive data type, not the Object wrapper class.
Addition of numbers using Integers turns out into very **costly** operation in terms of **performance, boxing/unboxing** and **unnecessary object formations.**
Just imagine a situation where 1000000's... of number are being added using Integer, it will literally explode our heap memory and boxing/unboxing operations will have adverse effect on performance.

Read : Autoboxing and unboxing in java - How it works internally in detail with 10 examples- Widening, AutoBoxing and Var-args

4) Avoid memory leak using WeakHashMap in java

An entry in a WeakHashMap will be automatically removed by garbage collector when its key is no longer in ordinary use. So, using WeakHashMap in place of HashMap can help us in avoiding memory leaks.

5) Using custom key in map without Overriding equals() and hashCode() method can cause memory leak >

If custom key is used and equals() and hashCode() method are not overridden then, key will not be retrieved by using get() method.
Because get() method internally calls equals() and hashCode() method for retrieving keys. So, these keys will neither be used nor be garbage collected, so it's a clear case of memory leak.

So, to avoid memory leak while using custom key you must always

- Override equals() and hashCode() method

Learn how you can use custom key (Employee object) in custom HashMap Custom implementation - put, get, remove

So, to avoid memory leak you must try use String, Integer, Long, Double, Float, Short and any other wrapper class as key in HashMap.

For more details read : How Integer is used as key in Hashmap

3.6) Close JDBC Statement, PreparedStatement, CallableStatement , ResultSet and Connections in java to avoid memory leaks >

You must ensure that you close all the JDBC ***Statement***, ***PreparedStatement***, ***CallableStatement*** , ResultSet and Connections in java to **avoid memory leaks.** You must always close all the above mentioned objects in finally block in java because finally block is **always executed** irrespective of exception is thrown or not by java code.

Also read : **Java JDBC best practices tutorial**

7) Memory leaks can also be caused by native methods in java > Memory

allocated through native methods can cause some serious memory leak.

8) Memory leak in web applications in java >

Unused Objects stored in application scope are memory leak because they are not collected until web application is stopped.

Read : **JVM (java virtual machine)** in detail in java and How **Garbage Collection (GC) works internally**

Question 41. Comment on relation between OutOfMemoryError and garbage collection in java?

Answer. There is huge **relationship** between OutOfMemoryError and garbage collection in java.

OutOfMemoryError may be thrown when an **excessive amount of time** is being by jvm in performing **garbage collection** and very little memory is being freed.

A long lived application might be unintentionally **holding references to objects** and this **prevents the objects from being garbage collected**. Holding of objects for a long time is also a **kind of memory leak** in java.

Question 42. Discuss **OutOfMemoryError: GC Overhead limit exceeded** in java?

Answer. OutOfMemoryError: GC Overhead limit exceeded **- indicates that** the **garbage collector is running all the time** and **Java program is making** very **slow progress**.

After a GC (**garbage collection)**, if the **garbage collector** is spending more than **98% of its time in doing garbage collection** and if **less than 2% of the java heap memory space is reclaimed,** then **OutOfMemoryError -** GC Overhead limit exceeded **-** is thrown in java.

This OutOfMemoryError is generally thrown because all the **live objects are not getting garbage collected properly and java heap space is not available for new objects.**

Question 43. What you know about **OutOfMemoryError: permgen** in java?

Answer. Another tricky and important garbage collection interview question which will check your in depth knowledge about jvm and garbage collection. Generally when we are facing java.lang.**OutOfMemoryError - Java permgen space**, then we need to change permgen size of tomcat or eclipse or JVM wherever you are facing this error.

Read In detail > **OutOfMemoryError: Permgen space - How to set or change permgen size in tomcat server, eclipse?**

Question 44. How to **Solve** OutOfMemoryError : **unable** to **create new native Thread** by **passing appropriate jvm parameter**?

Answer. A

You can **resolve** "java.lang.OutOfMemoryError : unable to create new native Thread" by setting the **appropriate size using -Xss** vm option.

Solution 1 to "java.lang.OutOfMemoryError : unable to create new native Thread" >
Try to increase the the -Xss value so that new threads gets enough stack space.

Solution 2 to "java.lang.OutOfMemoryError : unable to create new native Thread" >
Alternatively you could also increase the heap size available using **-**Xms and -Xmx options and then try to **increase and set appropriate -Xss value**.

Example of using -Xss

Pass memory value you want to allocate to thread stack with -Xss.
java -Xss512m MyJavaProgram
It will set the default stack size of JVM to 512 megabytes.

Question 45. How to **Solve** OutOfMemoryError : **Java heap space** by **passing appropriate jvm parameter**?

Answer. A

OutOfMemoryError : Java heap space **-** is thrown whenever there is **insufficient** space to **allocate an object** in the **Java heap.**

Does **Exception in thread threadName -** java.lang.**OutOfMemoryError - Java heap space** indicates memory leak?
No, this **OutOfMemoryError** does not necessarily means that it is memory leak.

Increase the heap size using -Xms and -Xmx jvm parameters as a solution to this issue.

Must read: How to set, change, increase or decrease heap size in tomcat server and eclipse to avoid OutOfMemoryError ?

>How to set or change permgen size in tomcat server, eclipse?

Question 46. How to set appropriate heap size in eclipse in java?

Answer. We can make changes in **eclipse.ini** file.

Where we can configure

- -Xms (minimum heap size which is allocated at initialization of JVM),
- -Xmx (maximum heap size that JVM can use.)
- -XX:MaxPermSize: It's maximum value of **Permanent Space** that JVM can allot up to.

Question 47. What is **default garbage collectors** for Java 7?

Answer. Another very important garbage collection interview question for all developers.

default garbage collector for Java 7 >

For **server class machine - parallel collector.**

For **client class machine -** serial collector.

Question 48. What is **default garbage collectors** for Java 8 ?

Answer.

What is default garbage collector for Java 8 >

For **server class machine - parallel collector.**

For **client class machine -** serial collector.

Question 49. What is **default garbage collectors** for Java 9?

Answer. Another very important garbage collection interview question for experienced developers.

default garbage collector for Java 9 >

G1 garbage collector

Read in detail : What are default garbage collectors for Java 7, 8 and 9

interview Question 50. Throw some light on garbage collection and WeakHashMap in java?

Answer. Another very tricky garbage collection interview question for experienced developers in java.

java.util.WeakHashMap is hash table based implementation of the Map interface, with *weak keys*. An entry in a WeakHashMap will be automatically removed by garbage collector when its key is no longer in ordinary use. Mapping for a given key will not prevent the key from being discarded by the garbage collector, (i.e. made finalizable, finalized, and then reclaimed). When a key has been discarded its entry is removed from the map in java.

java.util.**WeakHashMap** is implementation of the java.util.Map interface in java.

The behavior of the java.util.WeakHashMap class depends upon garbage collector
The behavior of the WeakHashMap class depends upon garbage collector in java. Because the garbage collector may discard keys at any time, in WeakHashMap it may look like some unknown thread is silently removing entries.
Each key object in a WeakHashMap is stored indirectly as the referent of a weak reference. Therefore a key will be removed automatically only after the weak references to it, both inside and outside of the map, have been cleared by the garbage collector.

Read more in **Collection** interview questions