**Thread**

**Difference between Thread and Runnable interface in Java**

1) Java doesn't support multiple inheritance, which means you can only extend one class in Java so once you extended Thread class you lost your chance and can not extend or inherit another class in Java.

2) In Object oriented programming extending a class generally means adding new functionality, modifying or improving behaviors. If we are not making any modification on Thread than use Runnable interface instead.

3) Runnable interface represent a Task which can be executed by either plain Thread or Executors or any other means. so logical separation of Task as Runnable than Thread is good design decision.

4) Separating task as Runnable means we can reuse the task and also has liberty to execute it from different means. since you can not restart a Thread once it completes. again Runnable vs Thread for task, Runnable is winner.

5) Java designer recognizes this and that's why Executors accept Runnable as Task and they have worker thread which executes those task.

6) Inheriting all Thread methods are additional overhead just for representing a Task which can can be done easily with Runnable.

**Callable vs Runnable interface**

Difference between Runnable and Callable in JavaAs I explained major differences between Callable and Runnable interface in last section. Sometime this question is also asked as difference between call() and run() method in Java. All the points discussed here is equally related to that question as well. Let's see them in point format for better understanding :

1) Runnable interface is older than Callable, there from JDK 1.0, while Callable is added on Java 5.0.

2) Runnable interface has run() method to define task while Callable interface uses call() method for task definition.

3) run() method does not return any value, it's return type is void while call method returns value. Callable interface is a generic parameterized interface and Type of value is provided, when instance of Callable implementation is created.

4) Another difference on run and call method is that run method can not throw checked exception, while call method can throw checked exception in Java.

That's all on Difference between Callable and Runnable interface in Java or difference between call() and run() method. Both are very useful interface from core Java and good understanding of where to use Runnable and Callable is must for any good Java developer. In next article we will see example of Callable interface along with FutureTask to learn How to use Callable interface in Java.

Important point related to volatile keyword in Java

Since volatile keyword is used to make any variable volatile in Java environment, its good to know more about What is volatile keyword, what is its limitation and How to use volatile keyword in Java.

**What is volatile variable in Java and when to use it**

1) Volatile keyword can only be applied to variable, it can not be applied to class or method. using volatile keyword along with class and method is compiler error.

2) volatile is also refereed as modifier in Java.

**When to use Volatile variable in Java**

This is the most important thing to learn while learning about volatile variable in Java. When to use volatile variable in Java is also a famous multi-threading interview question in Java. here are some of the scenario where you can use volatile variable in Java :

1) Any variable which is shared between multiple threads should be made variable, in order to ensure that all thread must see latest value of volatile variable.

2) A signal to compiler and JIT to ensure that compiler does not change ordering or volatile variable and moves them out of synchronized context.

3) You want to save cost of synchronization as volatile variables are less expensive than synchronization.

That's all on What is volatile variable in Java, When to use volatile variable in Java. volatile variable is an important concept to understand and use. Its also very important in terms of Java interview point of view.

**Why wait (), notify () and notifyAll () must be called from synchronized block or method in Java**

Why wait(), notify() and notifyAll() method must be called from synchronized block or method in JavaMost of Java developer knows that wait() ,notify() and notifyAll() method of object class must have to be called inside synchronized method or synchronized block in Java but how many times we thought why ? Recently this questions was asked to in Java interview to one of my friend, he pondered for a moment and replied that if we don't call wait () or notify () method from synchronized context we will receive IllegalMonitorStateException in java. He was right in terms of behavior of language but as per him interviewer was not completely satisfied with the answer and wanted to explain more about it. After the interview he discussed the same questions with me and I thought he might have told about race condition between wait () and notify () in Java that could exists if we don't call them inside synchronized method or block. Let’s see how it could happen:

We use wait () and notify () or notifyAll () method mostly for inter-thread communication. One thread is waiting after checking a condition e.g. In Producer Consumer example Producer Thread is waiting if buffer is full and Consumer thread notify Producer thread after he creates a space in buffer by consuming an element. calling notify() or notifyAll() issues a notification to a single or multiple thread that a condition has changed and once notification thread leaves synchronized block , all the threads which are waiting fight for object lock on which they are waiting and lucky thread returns from wait() method after reacquiring the lock and proceed further. Let’s divide this whole operation in steps to see a possibility of race condition between wait () and notify () method in Java, we will use Produce Consumer thread example to understand the scenario better:

1. The Producer thread tests the condition (buffer is full or not) and confirms that it must wait (after finding buffer is full).

2. The Consumer thread sets the condition after consuming an element from buffer.

3. The Consumer thread calls the notify () method; this goes unheard since the Producer thread is not yet waiting.

4. The Producer thread calls the wait () method and goes into waiting state.

So due to race condition here we potential lost a notification and if we use buffer or just one element Produce thread will be waiting forever and your program will hang.

Now let's think how does this potential race condition get resolved? This race condition is resolved by using synchronized keyword and locking provided by java. In order to call the wait (), notify () or notifyAll () methods in Java, we must have obtained the lock for the object on which we're calling the method. Since the wait () method in Java also releases the lock prior to waiting and reacquires the lock prior to returning from the wait () method, we must use this lock to ensure that checking the condition (buffer is full or not) and setting the condition (taking element from buffer) is atomic which can be achieved by using synchronized method or block in Java.

**What is ClassLoader in Java?**

Answer : This was one of advanced question few years ago, but in span of two to three years, this has become very common. When a Java program is converted into .class file by Java compiler which is collection of byte code class loader is responsible to load that class file from file system,network or any other location. This class loader is nothing but also a class from which location they are loading the class according to that class loaders are three types :

1.Bootstrap

2.Extension

3.System class loader .

Java class loaders are used to load classes at runtime. ClassLoader in Java works on three principle: delegation, visibility and uniqueness. Delegation principle forward request of class loading to parent class loader and only loads the class, if parent is not able to find or load class. Visibility principle allows child class loader to see all the classes loaded by parent ClassLoader, but parent class loader can not see classes loaded by child. Uniqueness principle allows to load a class exactly once, which is basically achieved by delegation and ensures that child ClassLoader doesn't reload the class already loaded by parent. Correct understanding of class loader is must to resolve issues like NoClassDefFoundError in Java and java.lang.ClassNotFoundException, which are related to class loading. ClassLoader is also an important topic in advanced Java Interviews, where good knowledge of working of Java ClassLoader and How classpath works in Java is expected from Java programmer. I have always seen questions like, Can one class be loaded by two different ClassLoader in Java on various Java Interviews. In this Java programming tutorial, we will learn what is ClassLoader in Java, How ClassLoader works in Java and some specifics about Java ClassLoader.

What is ClassLoader in Java

ClassLoader in Java is a class which is used to load class files in Java. Java code is compiled into class file by javac compiler and JVM executes Java program, by executing byte codes written in class file. ClassLoader is responsible for loading class files from file system, network or any other source. There are three default class loader used in Java, Bootstrap , Extension and System or Application class loader.

Every class loader has a predefined location, from where they loads class files. Bootstrap ClassLoader is responsible for loading standard JDK class files from rt.jar and it is parent of all class loaders in Java. Bootstrap class loader don't have any parents, if you call String.class.getClassLoader() it will return null and any code based on that may throw NullPointerException in Java. Bootstrap class loader is also known as Primordial ClassLoader in Java.

Extension ClassLoader delegates class loading request to its parent, Bootstrap and if unsuccessful, loads class form jre/lib/ext directory or any other directory pointed by java.ext.dirs system property. Extension ClassLoader in JVM is implemented by sun.misc.Launcher$ExtClassLoader.

Third default class loader used by JVM to load Java classes is called System or Application class loader and it is responsible for loading application specific classes from CLASSPATH environment variable, -classpath or -cp command line option, Class-Path attribute of Manifest file inside JAR. Application class loader is a child of Extension ClassLoader and its implemented by sun.misc.Launcher$AppClassLoader class. Also, except Bootstrap class loader, which is implemented in native language mostly in C, all Java class loaders are implemented using java.lang.ClassLoader.

In short here is the location from which Bootstrap, Extension and Application ClassLoader load Class files.

1) Bootstrap ClassLoader - JRE/lib/rt.jar

2) Extension ClassLoader - JRE/lib/ext or any directory denoted by java.ext.dirs

3) Application ClassLoader - CLASSPATH environment variable, -classpath or -cp option, Class-Path attribute of Manifest inside JAR file.

Class Loader in Java BootStrap Extension and Application

How ClassLoader works in Java

What is ClassLoader in Java, How classloader works in JavaAs I explained earlier Java ClassLoader works in three principles : delegation, visibility and uniqueness. In this section we will see those rules in detail and understand working of Java ClassLoader with example. By the way here is a diagram which explains How ClassLoader load class in Java using delegation.

How class loader works in Java - class loading

Delegation principles

As discussed on when a class is loaded and initialized in Java, a class is loaded in Java, when its needed. Suppose you have an application specific class called Abc.class, first request of loading this class will come to Application ClassLoader which will delegate to its parent Extension ClassLoader which further delegates to Primordial or Bootstrap class loader. Primordial will look for that class in rt.jar and since that class is not there, request comes to Extension class loader which looks on jre/lib/ext directory and tries to locate this class there, if class is found there than Extension class loader will load that class and Application class loader will never load that class but if its not loaded by extension class-loader than Application class loader loads it from Classpath in Java. Remember Classpath is used to load class files while PATH is used to locate executable like javac or java command.

Visibility Principle

According to visibility principle, Child ClassLoader can see class loaded by Parent ClassLoader but vice-versa is not true. Which mean if class Abc is loaded by Application class loader than trying to load class ABC explicitly using extension ClassLoader will throw either java.lang.ClassNotFoundException. as shown in below Example

Uniqueness Principle

According to this principle a class loaded by Parent should not be loaded by Child ClassLoader again. Though its completely possible to write class loader which violates Delegation and Uniqueness principles and loads class by itself, its not something which is beneficial. You should follow all class loader principle while writing your own ClassLoader.

How to load class explicitly in Java

Java provides API to explicitly load a class by Class.forName(classname) and Class.forName(classname, initialized, classloader), remember JDBC code which is used to load JDBC drives we have seen in Java program to Connect Oracle database. As shown in above example you can pass name of ClassLoader which should be used to load that particular class along with binary name of class. Class is loaded by calling loadClass() method of java.lang.ClassLoader class which calls findClass() method to locate bytecodes for corresponding class. In this example Extension ClassLoader uses java.net.URLClassLoader which search for class files and resources in JAR and directories. any search path which is ended using "/" is considered directory. If findClass() does not found the class than it throws java.lang.ClassNotFoundException and if it finds it calls defineClass() to convert bytecodes into a .class instance which is returned to the caller.

Where to use ClassLoader in Java

ClassLoader in Java is a powerful concept and used at many places. One of the popular example of ClassLoader is AppletClassLoader which is used to load class by Applet, since Applets are mostly loaded from internet rather than local file system, By using separate ClassLoader you can also loads same class from multiple sources and they will be treated as different class in JVM. J2EE uses multiple class loaders to load class from different location like classes from WAR file will be loaded by Web-app ClassLoader while classes bundled in EJB-JAR is loaded by another class loader. Some web server also supports hot deploy functionality which is implemented using ClassLoader. You can also use ClassLoader to load classes from database or any other persistent store.

That's all about What is ClassLoader in Java and How ClassLoader works in Java. We have seen delegation, visibility and uniqueness principles which is quite important to debug or troubleshoot any ClassLoader related issues in Java. In summary knowledge of How ClassLoader works in Java is must for any Java developer or architect to design Java application and packaging.

Top 50 Java Thread Interview Questions Answers for Experienced

You go to any Java interview, senior or junior, experience or freshers, you are bound to see a couple of questions from the thread, concurrency, and multi-threading. In fact, this built-in concurrency support is one of the strongest points of Java programming language and helped it to gain popularity among enterprise world and programmers equally. Most of lucrative Java developer position demands excellent core Java multi-threading skills and experience in developing, debugging and tuning high-performance low latency concurrent Java applications. This is the reason, it is one of the most sought after skill on Java interviews. The multithreading and concurrency are also hard to master concept and only good developers with solid experience can effectively deal with concurrency issues.

In a typical Java interview, Interviewer slowly starts from basic concepts of Thread by asking questions like, why you need threads, how to create threads, which one is better way to create threads e.g. by extending thread class or implementing Runnable and then slowly goes into Concurrency issues, challenges faced during development of concurrent Java applications, Java memory model, higher-order concurrency utilities introduced in JDK 1.5, principles and design patterns of concurrent Java applications, classical multi-threading problems e.g. producer-consumer, dining philosopher, reader-writer or simply bounded buffer problems.

Though you need good knowledge and solid experience to do well on Java interviews focused on advanced multithreading and concurrency skill, I strongly recommend Java programmers to read Effective Java and Java Concurrency in Practice twice before going to interview. They do not only help you to answer questions better but also help you to present your idea clearly.

Java multithreading interview questions answers

Since its also not enough just to know basics of threading, you must know how to deal with concurrency problems e.g. deadlock, race conditions, memory inconsistency and various thread safety related issues. These skills are thoroughly get tested by presenting various multi-threading and concurrency problems.

Many Java developers are used to only look and read interview questions before going for the interview, which is not bad but you should not be too far away. Also collecting questions and going through the same exercise is too much time consuming, that's why I have created this list of top 50 Java multi-threading and concurrency related questions, collected from various interviews. I am only going to add new and recent interview questions as and when I am going to discover them.

By the way, I have not provided answers to some questions here, Why? because I expect most of Java developer to know the answers to this question and if not, also answers are widely available by using Google. If you don't find the answer to any particular question, you can always ask me in the comments section. You can even find answers to few questions on the link provided or my earlier post Top 12 Java Thread Questions with Answers.

50 Interview questions from Java Multi-threading and Concurrency

Here is our list of top questions from Java thread, concurrency, and multi-threading. You can use this list to prepare well for your Java interview.

1) What is Thread in Java? (answer)

The thread is an independent path of execution. It's way to take advantage of multiple CPU available in a machine. By employing multiple threads you can speed up CPU bound task. For example, if one thread takes 100 milliseconds to do a job, you can use 10 thread to reduce that task into 10 milliseconds. Java provides excellent support for multithreading at the language level, and it's also one of the strong selling points.

2) What is the difference between Thread and Process in Java? (answer)

The thread is a subset of Process, in other words, one process can contain multiple threads. Two process runs on different memory space, but all threads share same memory space. Don't confuse this with stack memory, which is different for the different thread and used to store local data to that thread. For more detail see the answer.

3) How do you implement Thread in Java? (answer)

At the language level, there are two ways to implement Thread in Java. An instance of java.lang.Thread represent a thread but it needs a task to execute, which is an instance of interface java.lang.Runnable. Since Thread class itself implement Runnable, you can override run() method either by extending Thread class or just implementing Runnable interface. For detailed answer and discussion see this article.

4) When to use Runnable vs Thread in Java? (answer)

This is a follow-up of previous multi-threading interview question. As we know we can implement thread either by extending Thread class or implementing Runnable interface, the question arise, which one is better and when to use one? This question will be easy to answer if you know that Java programming language doesn't support multiple inheritances of class, but it allows you to implement multiple interfaces. Which means, it's better to implement Runnable then extends Thread if you also want to extend another class e.g. Canvas or CommandListener. For more points and discussion you can also refer this post.

6) What is the difference between start() and run() method of Thread class? (answer)

One of trick Java question from early days, but still good enough to differentiate between shallow understanding of Java threading model start() method is used to start newly created thread, while start() internally calls run() method, there is difference calling run() method directly. When you invoke run() as normal method, its called in the same thread, no new thread is started, which is the case when you call start() method. Read this answer for much more detailed discussion.

7) What is the difference between Runnable and Callable in Java? (answer)

Both Runnable and Callable represent task which is intended to be executed in a separate thread. Runnable is there from JDK 1.0 while Callable was added on JDK 1.5. Main difference between these two is that Callable's call() method can return value and throw Exception, which was not possible with Runnable's run() method. Callable return Future object, which can hold the result of computation. See my blog post on the same topic for a more in-depth answer to this question.

8) What is the difference between CyclicBarrier and CountDownLatch in Java? (answer)

Though both CyclicBarrier and CountDownLatch wait for number of threads on one or more events, the main difference between them is that you can not re-use CountDownLatch once count reaches to zero, but you can reuse same CyclicBarrier even after barrier is broken. See this answer for few more points and sample code example.

9) What is Java Memory model? (answer)

Java Memory model is set of rules and guidelines which allows Java programs to behave deterministically across multiple memory architecture, CPU, and operating system. It's particularly important in case of multi-threading. Java Memory Model provides some guarantee on which changes made by one thread should be visible to others, one of them is happens-before relationship. This relationship defines several rules which allows programmers to anticipate and reason behaviour of concurrent Java programs. For example, happens-before relationship guarantees :

Each action in a thread happens-before every action in that thread that comes later in the program order, this is known as program order rule.

An unlock on a monitor lock happens-before every subsequent lock on that same monitor lock, also known as Monitor lock rule.

A write to a volatile field happens-before every subsequent read of that same field, known as Volatile variable rule.

A call to Thread.start on a thread happens-before any other thread detects that thread has terminated, either by successfully return from Thread.join() or by Thread.isAlive() returning false, also known as Thread start rule.

A thread calling interrupt on another thread happens-before the interrupted thread detects the interrupt( either by having InterruptedException thrown, or invoking isInterrupted or interrupted), popularly known as Thread Interruption rule.

The end of a constructor for an object happens-before the start of the finalizer for that object, known as Finalizer rule.

If A happens-before B, and B happens-before C, then A happens-before C, which means happens-before guarantees Transitivity.

I strongly suggest reading Chapter 16 of Java Concurrency in Practice to understand Java Memory model in more detail.

10) What is volatile variable in Java? (answer)

volatile is a special modifier, which can only be used with instance variables. In concurrent Java programs, changes made by multiple threads on instance variables is not visible to other in absence of any synchronizers e.g. synchronized keyword or locks. Volatile variable guarantees that a write will happen before any subsequent read: as stated: "volatile variable rule" in previous question. Read this answer to learn more about volatile variable and when to use them.

11) What is thread-safety? is Vector a thread-safe class? (Yes, see details)

Thread-safety is a property of an object or code which guarantees that if executed or used by multiple threads in any manner e.g. read vs write it will behave as expected. For example, a thread-safe counter object will not miss any count if same instance of that counter is shared among multiple threads. Apparently, you can also divide collection classes in two category, thread-safe and non-thread-safe. Vector is indeed a thread-safe class and it achieves thread-safety by synchronizing methods which modify state of Vector, on the other hand, its counterpart ArrayList is not thread-safe.

12) What is race condition in Java? Given one example? (answer)

Race condition are cause of some subtle programming bugs when Java programs are exposed to concurrent execution environment. As the name suggests, a race condition occurs due to race between multiple threads, if a thread which is supposed to execute first lost the race and executed second, behaviour of code changes, which surface as non-deterministic bugs. This is one of the hardest bugs to find and re-produce because of random nature of racing between threads. One example of race condition is out-of-order processing, see this answer for some more example of race conditions in Java programs.

13) How to stop a thread in Java? (answer(answer)

I always said that Java provides rich APIs for everything but ironically Java doesn't provide a sure shot way of stopping thread. There was some control methods in JDK 1.0 e.g. stop(), suspend() and resume() which was deprecated in later releases due to potential deadlock threats, from then Java API designers has not made any effort to provide a consistent, thread-safe and elegant way to stop threads. Programmers mainly rely on the fact that thread stops automatically as soon as they finish execution of run() or call() method. To manually stop, programmers either take advantage of volatile boolean variable and check in every iteration if run method has loops or interrupt threads to abruptly cancel tasks. See this tutorial for sample code of stopping thread in Java.

14) What happens when an Exception occurs in a thread? (answer)

This is one of the good tricky Java question I have seen in interviews. In simple words, If not caught thread will die, if an uncaught exception handler is registered then it will get a call back. Thread.UncaughtExceptionHandler is an interface, defined as nested interface for handlers invoked when a Thread abruptly terminates due to an uncaught exception. When a thread is about to terminate due to an uncaught exception the Java Virtual Machine will query the thread for its UncaughtExceptionHandler using Thread.getUncaughtExceptionHandler() and will invoke the handler's uncaughtException() method, passing the thread and the exception as arguments.

15) How do you share data between two thread in Java? (answer)

You can share data between threads by using shared object, or concurrent data structure like BlockingQueue. See this tutorial to learn inter-thread communication in Java. It implements Producer consumer pattern using wait and notify methods, which involves sharing objects between two threads.

Java concurrency questions for experienced programmers

16) What is the difference between notify and notifyAll in Java? (answer)

This is another tricky questions from core Java interviews, since multiple threads can wait on single monitor lock, Java API designer provides method to inform only one of them or all of them, once waiting condition changes, but they provide half implementation. There notify() method doesn't provide any way to choose a particular thread, that's why its only useful when you know that there is only one thread is waiting. On the other hand, notifyAll() sends notification to all threads and allows them to compete for locks, which ensures that at-least one thread will proceed further. See my blog post on similar topic for a more detailed answer and code example.

17) Why wait, notify and notifyAll are not inside thread class? (answer)

This is a design related question, which checks what candidate thinks about existing system or does he ever thought of something which is so common but looks in-appropriate at first. In order to answer this question, you have to give some reasons why it make sense for these three method to be in Object class, and why not on Thread class. One reason which is obvious is that Java provides lock at object level not at thread level. Every object has lock, which is acquired by thread. Now if thread needs to wait for certain lock it make sense to call wait() on that object rather than on that thread. Had wait() method declared on Thread class, it was not clear that for which lock thread was waiting. In short, since wait, notify and notifyAll operate at lock level, it make sense to defined it on object class because lock belongs to object. You can also see this article for more elaborate answer of this question.

18) What is ThreadLocal variable in Java? (answer)

ThreadLocal variables are special kind of variable available to Java programmer. Just like instance variable is per instance, ThreadLocal variable is per thread. It's a nice way to achieve thread-safety of expensive-to-create objects, for example you can make SimpleDateFormat thread-safe using ThreadLocal. Since that class is expensive, its not good to use it in local scope, which requires separate instance on each invocation. By providing each thread their own copy, you shoot two birds with one arrow. First, you reduce number of instance of expensive object by reusing fixed number of instances, and Second, you achieve thread-safety without paying cost of synchronization or immutability. Another good example of thread local variable is ThreadLocalRandom class, which reduces number of instances of expensive-to-create Random object in multi-threading environment. See this answer to learn more about thread local variables in Java.

19) What is FutureTask in Java? (answer)

FutureTask represents a cancellable asynchronous computation in concurrent Java application. This class provides a base implementation of Future, with methods to start and cancel a computation, query to see if the computation is complete, and retrieve the result of the computation. The result can only be retrieved when the computation has completed; the get methods will block if the computation has not yet completed. A FutureTask object can be used to wrap a Callable or Runnable object. Since FutureTask also implements Runnable, it can be submitted to an Executor for execution.

20) What is the difference between the interrupted() and isInterrupted() method in Java? (answer)

Main difference between interrupted() and isInterrupted() is that former clears the interrupt status while later does not. The interrupt mechanism in Java multi-threading is implemented using an internal flag known as the interrupt status. Interrupting a thread by calling Thread.interrupt() sets this flag. When interrupted thread checks for an interrupt by invoking the static method Thread.interrupted(), interrupt status is cleared. The non-static isInterrupted() method, which is used by one thread to query the interrupt status of another, does not change the interrupt status flag. By convention, any method that exits by throwing an InterruptedException clears interrupt status when it does so. However, it's always possible that interrupt status will immediately be set again, by another thread invoking interrupt

21) Why wait and notify method are called from synchronized block? (answer)

Main reason for calling wait and notify method from either synchronized block or method is that it made mandatory by Java API. If you don't call them from synchronized context, your code will throw IllegalMonitorStateException. A more subtle reason is to avoid the race condition between wait and notify calls. To learn more about this, check my similarly titled post here.

22) Why should you check condition for waiting in a loop? (answer)

Its possible for a waiting thread to receive false alerts and spurious wake up calls, if it doesn't check the waiting condition in loop, it will simply exit even if condition is not met. As such, when a waiting thread wakes up, it cannot assume that the state it was waiting for is still valid. It may have been valid in the past, but the state may have been changed after the notify() method was called and before the waiting thread woke up. That's why it always better to call wait() method from loop, you can even create template for calling wait and notify in Eclipse. To learn more about this question, I would recommend you to read Effective Java items on thread and synchronization.

23) What is the difference between synchronized and concurrent collection in Java? (answer)

Though both synchronized and concurrent collection provides thread-safe collection suitable for multi-threaded and concurrent access, later is more scalable than former. Before Java 1.5, Java programmers only had synchronized collection which becomes source of contention if multiple thread access them concurrently, which hampers scalability of system. Java 5 introduced concurrent collections like ConcurrentHashMap, which not only provides thread-safety but also improves scalability by using modern techniques like lock stripping and partitioning internal table. See this answer for more differences between synchronized and concurrent collection in Java.

24) What is the difference between Stack and Heap in Java? (answer)

Why does someone this question as part of multi-threading and concurrency? because Stack is a memory area which is closely associated with threads. To answer this question, both stack and heap are specific memories in Java application. Each thread has their own stack, which is used to store local variables, method parameters and call stack. Variable stored in one Thread's stack is not visible to other. On another hand, the heap is a common memory area which is shared by all threads. Objects whether local or at any level is created inside heap. To improve performance thread tends to cache values from heap into their stack, which can create problems if that variable is modified by more than one thread, this is where volatile variables come into the picture. volatile suggest threads read the value of variable always from main memory. See this article for learning more about stack and heap in Java to answer this question in greater detail.

Java thread interview questions with answers difficult ones

25) What is thread pool? Why should you thread pool in Java? (answer)

Creating thread is expensive in terms of time and resource. If you create thread at time of request processing it will slow down your response time, also there is only a limited number of threads a process can create. To avoid both of these issues, a pool of thread is created when application starts-up and threads are reused for request processing. This pool of thread is known as "thread pool" and threads are known as worker thread. From JDK 1.5 release, Java API provides Executor framework, which allows you to create different types of thread pools e.g. single thread pool, which process one task at a time, fixed thread pool (a pool of fixed number of threads) or cached thread pool (an expandable thread pool suitable for applications with many short lived tasks). See this article to learn more about thread pools in Java to prepare detailed answer of this question.

26) Write code to solve Producer Consumer problem in Java? (answer)

Most of the threading problem you solved in the real world are of the category of Producer consumer pattern, where one thread is producing task and another thread is consuming that. You must know how to do inter thread communication to solve this problem. At the lowest level, you can use wait and notify to solve this problem, and at a high level, you can leverage Semaphore or BlockingQueue to implement Producer consumer pattern, as shown in this tutorial.

27) How do you avoid deadlock in Java? Write Code?

deadlock in multithreading Java

Deadlock is a condition in which two threads wait for each other to take action which allows them to move further. It's a serious issue because when it happen your program hangs and doesn't do the task it is intended for. In order for deadlock to happen, following four conditions must be true:

Mutual Exclusion : At least one resource must be held in a non-shareable mode. Only one process can use the resource at any given instant of time.

Hold and Wait: A process is currently holding, at least, one resource and requesting additional resources which are being held by other processes.

No Pre-emption: The operating system must not de-allocate resources once they have been allocated; they must be released by the holding process voluntarily.

Circular Wait: A process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource.

The easiest way to avoid deadlock is to prevent Circular wait, and this can be done by acquiring locks in a particular order and releasing them in reverse order so that a thread can only proceed to acquire a lock if it held the other one. Check this tutorial for the actual code example and detailed discussion on techniques for avoiding deadlock in Java.

28) What is the difference between livelock and deadlock in Java? (answer)

This question is extension of previous interview question. A livelock is similar to a deadlock, except that the states of the threads or processes involved in the livelock constantly change with regard to one another, without any one progressing further. Livelock is a special case of resource starvation. A real-world example of livelock occurs when two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass, but they end up swaying from side to side without making any progress because they both repeatedly move the same way at the same time. In short, the main difference between livelock and deadlock is that in former state of process change but no progress is made.

29) How do you check if a Thread holds a lock or not? (answer)

I didn't even know that you can check if a Thread already holds lock before this question hits me in a telephonic round of Java interview. There is a method called holdsLock() on java.lang.Thread, it returns true if and only if the current thread holds the monitor lock on the specified object. You can also check this article for the more detailed answer.

30) How do you take thread dump in Java? (answer)

There are multiple ways to take thread dump of Java process depending upon operating system. When you take thread dump, JVM dumps state of all threads in log files or standard error console. In windows you can use Ctrl + Break key combination to take thread dump, on Linux you can use kill -3 command for same. You can also use a tool called jstack for taking thread dump, it operate on process id, which can be found using another tool called jps.

31) Which JVM parameter is used to control stack size of a thread? (answer)

This is the simple one, -Xss parameter is used to control stack size of Thread in Java. You can see this list of JVM options to learn more about this parameter.

32) What is the difference between synchronized and ReentrantLock in Java? (answer)

There were days when the only way to provide mutual exclusion in Java was via synchronized keyword, but it has several shortcomings e.g. you can not extend lock beyond a method or block boundary, you can not give up trying for a lock etc. Java 5 solves this problem by providing more sophisticated control via Lock interface. ReentrantLock is a common implementation of Lock interface and provides re-entrant mutual exclusion Lock with the same basic behavior and semantics as the implicit monitor lock accessed using synchronized methods and statements, but with extended capabilities. See this article learn about those capabilities and some more differences between synchronized vs ReentrantLock in Java.

33) There are three threads T1, T2, and T3? How do you ensure sequence T1, T2, T3 in Java? (answer)

Sequencing in multi-threading can be achieved by different means but you can simply use the join() method of thread class to start a thread when another one has finished its execution. To ensure three threads execute you need to start the last one first e.g. T3 and then call join methods in reverse order e.g. T3 calls T2. join and T2 calls T1.join, these ways T1 will finish first and T3 will finish last. To learn more about join method, see this tutorial.

34) What does yield method of Thread class do? (answer)

Yield method is one way to request current thread to relinquish CPU so that other thread can get a chance to execute. Yield is a static method and only guarantees that current thread will relinquish the CPU but doesn't say anything about which other thread will get CPU. Its possible for the same thread to get CPU back and start its execution again. See this article to learn more about yield method and to answer this question better.

35) What is the concurrency level of ConcurrentHashMap in Java? (answer)

ConcurrentHashMap achieves it's scalability and thread-safety by partitioning actual map into a number of sections. This partitioning is achieved using concurrency level. Its optional parameter of ConcurrentHashMap constructor and it's default value is 16. The table is internally partitioned to try to permit the indicated number of concurrent updates without contention. To learn more about concurrency level and internal resizing, see my post How ConcurrentHashMap works in Java.

36) What is Semaphore in Java? (answer)

Semaphore in Java is a new kind of synchronizer. It's a counting semaphore. Conceptually, a semaphore maintains a set of permits. Each acquire() blocks if necessary until a permit is available, and then takes it. Each release() adds a permit, potentially releasing a blocking acquirer. However, no actual permit objects are used; the Semaphore just keeps a count of the number available and acts accordingly. Semaphore is used to protect an expensive resource which is available in fixed number e.g. database connection in the pool. See this article to learn more about counting Semaphore in Java.

37) What happens if you submit a task when the queue of the thread pool is already filled? (answer)

This is another tricky question on my list. Many programmers will think that it will block until a task is cleared but its true. ThreadPoolExecutor's submit() method throws RejectedExecutionException if the task cannot be scheduled for execution.

38) What is the difference between the submit() and execute() method thread pool in Java? (answer)

Both methods are ways to submit a task to thread pools but there is a slight difference between them. execute(Runnable command) is defined in Executor interface and executes given task in future, but more importantly, it does not return anything. Its return type is void. On other hand submit() is an overloaded method, it can take either Runnable or Callable task and can return Future object which can hold the pending result of computation. This method is defined on ExecutorService interface, which extends Executor interface, and every other thread pool class e.g. ThreadPoolExecutor or ScheduledThreadPoolExecutor gets these methods. To learn more about thread pools you can check this article.

39) What is blocking method in Java? (answer)

A blocking method is a method which blocks until the task is done, for example, accept() method of ServerSocket blocks until a client is connected. here blocking means control will not return to the caller until the task is finished. On the other hand, there is an asynchronous or non-blocking method which returns even before the task is finished. To learn more about blocking method see this answer.

40) Is Swing thread-safe? What do you mean by Swing thread-safe? (answer)

You can simply this question as No, Swing is not thread-safe, but you have to explain what you mean by that even if the interviewer doesn't ask about it. When we say swing is not thread-safe we usually refer its component, which can not be modified in multiple threads. All update to GUI components has to be done on AWT thread, and Swing provides synchronous and asynchronous callback methods to schedule such updates. You can also read my article to learn more about swing and thread-safety to better answer this question. Even next two questions are also related to this concept.

41) What is the difference between invokeAndWait and invokeLater in Java? (answer)

These are two methods Swing API provides Java developers for updating GUI components from threads other than Event dispatcher thread. InvokeAndWait() synchronously update GUI component, for example, a progress bar, once progress is made, the bar should also be updated to reflect that change. If progress is tracked in a different thread, it has to call invokeAndWait() to schedule an update of that component by Event dispatcher thread. On another hand, invokeLater() is an asynchronous call to update components. You can also refer this answer for more points.

42) Which method of Swing API are thread-safe in Java? (answer)

This question is again related to swing and thread-safety though components are not thread-safe there is a certain method which can be safely called from multiple threads. I know about repaint(), and revalidate() being thread-safe but there are other methods on different swing components e.g. setText() method of JTextComponent, insert() and append() method of JTextArea class.

43) How to create an Immutable object in Java? (answer)

This question might not look related to multi-threading and concurrency, but it is. Immutability helps to simplify already complex concurrent code in Java. Since immutable object can be shared without any synchronization its very dear to Java developers. Core value object, which is meant to be shared among thread should be immutable for performance and simplicity. Unfortunately there is no @Immutable annotation in Java, which can make your object immutable, hard work must be done by Java developers. You need to keep basics like initializing state in constructor, no setter methods, no leaking of reference, keeping separate copy of mutable object to create Immutable object. For step by step guide see my post, how to make an object Immutable in Java. This will give you enough material to answer this question with confidence.

44) What is ReadWriteLock in Java? (answer)

In general, read write lock is the result of lock stripping technique to improve the performance of concurrent applications. In Java, ReadWriteLock is an interface which was added in Java 5 release. A ReadWriteLock maintains a pair of associated locks, one for read-only operations and one for writing. The read lock may be held simultaneously by multiple reader threads, so long as there are no writers. The write lock is exclusive. If you want you can implement this interface with your own set of rules, otherwise you can use ReentrantReadWriteLock, which comes along with JDK and supports a maximum of 65535 recursive write locks and 65535 read locks.

45) What is busy spin in multi-threading? (answer)

Busy spin is a technique which concurrent programmers employ to make a thread wait on certain condition. Unlike traditional methods e.g. wait(), sleep() or yield() which all involves relinquishing CPU control, this method does not relinquish CPU, instead it the just runs empty loop. Why would someone do that? to preserve CPU caches. In a multi-core system, it's possible for a paused thread to resume on a different core, which means rebuilding cache again. To avoid cost of rebuilding cache, programmer prefer to wait for much smaller time doing busy spin. You can also see this answer to learn more about this question.

46) What is the difference between the volatile and atomic variable in Java? (answer)

This is an interesting question for Java programmer, at first, volatile and atomic variable look very similar, but they are different. Volatile variable provides you happens-before guarantee that a write will happen before any subsequent write, it doesn't guarantee atomicity. For example count++ operation will not become atomic just by declaring count variable as volatile. On the other hand AtomicInteger class provides atomic method to perform such compound operation atomically e.g. getAndIncrement() is atomic replacement of increment operator. It can be used to atomically increment current value by one. Similarly you have atomic version for other data type and reference variable as well.

47) What happens if a thread throws an Exception inside synchronized block? (answer)

This is one more tricky question for average Java programmer, if he can bring the fact about whether lock is released or not is a key indicator of his understanding. To answer this question, no matter how you exist synchronized block, either normally by finishing execution or abruptly by throwing exception, thread releases the lock it acquired while entering that synchronized block. This is actually one of the reasons I like synchronized block over lock interface, which requires explicit attention to release lock, generally this is achieved by releasing the lock in a finally block.

48) What is double checked locking of Singleton? (answer)

This is one of the very popular question on Java interviews, and despite its popularity, chances of candidate answering this question satisfactory is only 50%. Half of the time, they failed to write code for double checked locking and half of the time they failed how it was broken and fixed on Java 1.5. This is actually an old way of creating thread-safe singleton, which tries to optimize performance by only locking when Singleton instance is created first time, but because of complexity and the fact it was broken for JDK 1.4, I personally don't like it. Anyway, even if you not prefer this approach its good to know from interview point of view. Since this question deserve a detailed answer, I have answered in a separate post, you can read my post how double checked locking on Singleton works to learn more about it.

49) How to create thread-safe Singleton in Java? (answer)

This question is actually follow-up of the previous question. If you say you don't like double checked locking then Interviewer is bound to ask about alternative ways of creating thread-safe Singleton class. There are actually man, you can take advantage of class loading and static variable initialization feature of JVM to create instance of Singleton, or you can leverage powerful enumeration type in Java to create Singleton. I actually preferred that way, you can also read this article to learn more about it and see some sample code.

50) List down 3 multi-threading best practice you follow? (answer)

This is my favorite question because I believe that you must follow certain best practices while writing concurrent code which helps in performance, debugging and maintenance. Following are three best practices, I think an average Java programmer should follow:

Always give meaningful name to your threadThis goes a long way to find a bug or trace an execution in concurrent code. OrderProcessor, QuoteProcessor or TradeProcessor is much better than Thread-1. Thread-2 and Thread-3. The name should say about task done by that thread. All major framework and even JDK follow this best practice.

Avoid locking or Reduce scope of Synchronization

Locking is costly and context switching is even costlier. Try to avoid synchronization and locking as much as possible and at a bare minimum, you should reduce critical section. That's why I prefer synchronized block over synchronized method because it gives you absolute control on the scope of locking.

Prefer Synchronizers over wait and notify

Synchronizers like CountDownLatch, Semaphore, CyclicBarrier or Exchanger simplifies coding. It's very difficult to implement complex control flow right using wait and notify. Secondly, these classes are written and maintained by best in business and there is good chance that they are optimized or replaced by better performance code in subsequent JDK releases. By using higher level synchronization utilities, you automatically get all these benefits.

Prefer Concurrent Collection over Synchronized Collection

This is another simple best practice which is easy to follow but reap good benefits. Concurrent collection are more scalable than their synchronized counterpart, that's why its better to use them while writing concurrent code. So next time if you need map, think about ConcurrentHashMap before thinking Hashtable. See my article Concurrent Collections in Java, to learn more about modern collection classes and how to make best use of them.

51) How do you force to start a Thread in Java? (answer)

This question is like how do you force garbage collection in Java, there is no way though you can make a request using System.gc() but it's not guaranteed. On Java multi-threading there is absolute no way to force start a thread, this is controlled by thread scheduler and Java exposes no API to control thread schedule. This is still a random bit in Java.

52) What is the fork-join framework in Java? (answer)

The fork join framework, introduced in JDK 7 is a powerful tool available to Java developer to take advantage of multiple processors of modern day servers. It is designed for work that can be broken into smaller pieces recursively. The goal is to use all the available processing power to enhance the performance of your application. One significant advantage of The fork/join framework is that it uses a work-stealing algorithm. Worker threads that run out of things to do can steal tasks from other threads that are still busy. See this article for the much more detailed answer to this question.

53) What is the difference between calling wait() and sleep() method in Java multi-threading? (answer)

Though both wait and sleep introduce some form of pause in Java application, they are the tool for different needs. Wait method is used for inter thread communication, it relinquishes lock if waiting for a condition is true and wait for notification when due to an action of another thread waiting condition becomes false. On the other hand sleep() method is just to relinquish CPU or stop execution of current thread for specified time duration. Calling sleep method doesn't release the lock held by current thread. You can also take look at this article to answer this question with more details.

That's all on this list of top 50 Java multi-threading and concurrency interview questions. I have not shared answers of all the questions but provided enough hints and links to explore further and find answers by yourselves. As I said, let me know if you don't find answer of any particular question and I will add answer here.

You can use this list to not only to prepare for your core Java and programming interviews but also to check your knowledge about basics of threads, multi-threading, concurrency, design patterns and threading issues like race conditions, deadlock and thread safety problems.

My intention is to make this list of question as mother of all list of Java Multi-threading questions, but this can not be done without your help. You can also share any question with us, which has been asked to you or any question for which you yet to find an answer.

This master list is equally useful to Java developers of all levels of experience. You can read through this list even if you have 2 to 3 years of working experience as a junior developer or 5 to 6 years as senior developer. It's even useful for freshers and beginners to expand their knowledge. I will add new and latest multi-threading question as and when I come across, and I request you all to ask, share and answer questions via comments to keep this list relevant to all Java programmers.

**Collection**

**Q1 What is Collection ? What is a Collections Framework ? What are the benefits of Java Collections Framework ?**

**Collection : A collection (also called as container) is an object that groups multiple elements into a single unit.**

**Collections Framework : Collections framework provides unified architecture for manipulating and representing collections.**

**Benefits of Collections Framework :**

**1. Improves program quality and speed**

**2. Increases the chances of reusability of software**

**3. Decreases programming effort.**

**Q2 What is the root interface in collection hierarchy ?**

Root interface in collection hierarchy is Collection interface . Few interviewer may argue that

Collection interface extends Iterable interface. So iterable should be the root interface. But you should reply iterable interface present in java.lang package not in java.util package .It is clearly mentioned in Oracle Collection docs , that Collection interface is a member of the Java Collections framework. For Iterable interface Oracle doc , iterable interface is not mentioned as a part of the Java Collections framework .So if the question includes collection hierarchy , then you should answer the question as Collection interface (which is found in java.util package).

**Q3 What is the difference between Collection and Collections ?**

Collection is an interface while Collections is a java class , both are present in java.util package and part of java collections framework.

**Q4 Which collection classes are synchronized or thread-safe ?**

Stack, Properties , Vector and Hashtable can be used in multi threaded environment because they are synchronized classes (or thread-safe).

**Q5 Name the core Collection interfaces ?**

Java Collection Interfaces or Hierarchy

The list of core collection interfaces are : just mention the important ones

Important : Collection , Set , Queue , List , Map

Other interface also in the list : SortedSet, SortedMap , Deque, ListIterator etc.

**Q6 What is the difference between List and Set ?**

Set contain only unique elements while List can contain duplicate elements.Set is unordered while List is ordered . List maintains the order in which the objects are added .

**Q7 What is the difference between Map and Set ?**

Map object has unique keys each containing some value, while Set contain only unique values.

**Q8 What are the classes implementing List and Set interface ?**

Class implementing List interface : ArrayList , Vector , LinkedList ,Class implementing Set interface : HashSet , TreeSet

**Q9 What is an iterator ?**

Iterator is an interface . It is found in java.util package. It provides methods to iterate over any Collection**.**

**Q10 What is the difference between Iterator and Enumeration ?**

The main difference between Iterator and Enumeration is that Iterator has remove() method while Enumeration doesn't.

Hence , using Iterator we can manipulate objects by adding and removing the objects from the collections. Enumeration behaves like a read only interface as it can only traverse the objects and fetch it **.**

**Q11 Which design pattern followed by Iterator ?**

It follows iterator design pattern. Iterator design pattern provides us to navigate through the collection of objects by using a common interface without letting us know about the underlying implementation**.**

Enumeration is an example of Iterator design pattern.

**Q12 Which methods you need to override to use any object as key in HashMap ?**

To use any object as key in HashMap , it needs to implement equals() and hashCode() method .

**Q13 What is the difference between Queue and Stack ?**

Queue is a data structure which is based on FIFO ( first in first out ) property . An example of Queue in real world is buying movie tickets in the multiplex or cinema theaters**.**Stack is a data structure which is based on LIFO (last in first out) property . An example of Stack in real world is insertion or removal of CD from the CD case.

**Q14 How to reverse the List in Collections ?**

There is a built in reverse method in Collections class . reverse(List list) accepts list as parameter.

Collections.reverse(listobject);

**Q15 How to convert the array of strings into the list ?**

Arrays class of java.util package contains the method asList() which accepts the array as parameter**.**

So,String[] wordArray = {"Love Yourself" , "Alive is Awesome" , "Be in present"};

List wordList = Arrays.asList(wordArray);

**Q16 What is the difference between ArrayList and Vector**

1. Synchronization and Thread-Safe

Vector is synchronized while ArrayList is not synchronized . Synchronization and thread safe means at a time only one thread can access the code .In Vector class all the methods are synchronized .Thats why the Vector object is already synchronized when it is created .

2. Performance

Vector is slow as it is thread safe . In comparison ArrayList is fast as it is non synchronized . Thus in ArrayList two or more threads can access the code at the same time , while Vector is limited to one thread at a time.

3. Automatic Increase in Capacity

A Vector defaults to doubling size of its array . While when you insert an element into the ArrayList , it increases its Array size by 50% By default ArrayList size is 10 . It checks whether it reaches the last element then it will create the new array ,copy the new data of last array to new array ,then old array is garbage collected by the Java Virtual Machine (JVM) .

4. Set Increment Size

ArrayList does not define the increment size . Vector defines the increment size .You can find the following method in Vector Class public synchronized void setSize(int i) { //some code }There is no setSize() method or any other method in ArrayList which can manually set the increment size.

5. Enumerator

Other than Hashtable ,Vector is the only other class which uses both Enumeration and Iterator .While ArrayList can only use Iterator for traversing an ArrayList .

6. Introduction in Java

java.util.Vector class was there in java since the very first version of the java development kit (jdk).java.util.ArrayList was introduced in java version 1.2 , as part of Java Collections framework . In java version 1.2 , Vector class has been refactored to implement the List Inteface .

**Q17 What is the difference between HashMap and Hashtable ?**

**Difference between HashMap and HashTable / HashMap vs HashTable**

**1. Synchronization or Thread Safe :**

This is the most important difference between two . HashMap is non synchronized and not thread safe.On the other hand, HashTable is thread safe and synchronized**.**

When to use HashMap ? answer is if your application do not require any multi-threading task, in other words hashmap is better for non-threading applications. HashTable should be used in multithreading applications.

**2. Null keys and null values :**

Hashmap allows one null key and any number of null values, while Hashtable do not allow null keys and null values in the HashTable object.

**3. Iterating the values:**

Hashmap object values are iterated by using iterator .HashTable is the only class other than vector which uses enumerator to iterate the values of HashTable object**.**

**difference between hashmap and hashtable**

**4. Fail-fast iterator :**

The iterator in Hashmap is fail-fast iterator while the enumerator for Hashtable is not.According to Oracle Docs, if the Hashtable is structurally modified at any time after the iterator is created in any way except the iterator's own remove method , then the iterator will throw ConcurrentModification Exception.

Structural modification means adding or removing elements from the Collection object (here hashmap or hashtable) . Thus the enumerations returned by the Hashtable keys and elements methods are not fail fast.We have already explained the difference between iterator and enumeration.

**5. Performance :**

Hashmap is much faster and uses less memory than Hashtable as former is unsynchronized . Unsynchronized objects are often much better in performance in compare to synchronized object like Hashtable in single threaded environment.

**6. Superclass and Legacy :**

Hashtable is a subclass of Dictionary class which is now obsolete in Jdk 1.7 ,so ,it is not used anymore. It is better off externally synchronizing a HashMap or using a ConcurrentMap implementation (e.g ConcurrentHashMap).HashMap is the subclass of the AbstractMap class. Although Hashtable and HashMap has different superclasses but they both are implementations **1. Insertion Order :**

Both HashMap and Hashtable does not guarantee that the order of the map will remain constant over time. Instead use LinkedHashMap, as the order remains constant over time.

**2. Map interface :**

Both HashMap and Hashtable implements Map interface **.**

**3. Put and get method :**

Both HashMap and Hashtable provides constant time performance for put and get methods assuming that the objects are distributed uniformly across the bucket.

**4. Internal working :**

Both HashMap and Hashtable works on the Principle of Hashing . We have already discussed how hashmap works in java .

**When to use HashMap and Hashtable?**

**1. Single Threaded Application**

HashMap should be preferred over Hashtable for the non-threaded applications. In simple words , use HashMap in unsynchronized or single threaded applications .

**2. Multi Threaded Application**

We should avoid using Hashtable, as the class is now obsolete in latest Jdk 1.8 . Oracle has provided a better replacement of Hashtable named ConcurrentHashMap. For multithreaded application prefer ConcurrentHashMap instead of Hashtable**.**

**HashMap Hashtable**

Synchronized No Yes

Thread-Safe No Yes

Null Keys and Null values One null key ,Any null values Not permit null keys and values

Iterator type Fail fast iterator Fail safe iterator

Performance Fast Slow in comparison

Superclass and Legacy AbstractMap , No Dictionary , Yes

**Q18 What is the difference between peek(),poll() and remove() method of the Queue interface ?**

Both poll() and remove() method is used to remove head object of the Queue. The main difference lies when the Queue is empty().

If Queue is empty then poll() method will return null . While in similar case , remove() method will throw NoSuchElementException .

peek() method retrieves but does not remove the head of the Queue. If queue is empty then peek() method also returns null.

**Q19 What is the difference between Iterator and ListIterator**.

Using Iterator we can traverse the list of objects in forward direction . But ListIterator can traverse the collection in both directions that is forward as well as backward.

**Q20 What is the difference between Array and ArrayList in Java ?**

1. Resizable : Array is static in size that is fixed length data structure, One can not change the length after creating the Array object.

ArrayList is dynamic in size . Each ArrayList object has instance variable capacity which indicates the size of the ArrayList. As elements are added to an ArrayList its capacity grows automatically.

2. Performance : Performance of Array and ArrayList depends on the operation you are performing :

resize() opertation : Automatic resize of ArrayList will slow down the performance as it will use temporary array to copy elements from the old array to new array.

ArrayList is internally backed by Array during resizing as it calls the native implemented method System.arrayCopy(src,srcPos,dest,destPos,length) .

add() or get() operation : adding an element or retrieving an element from the array or arraylist object has almost same performance , as for ArrayList object these operations run in constant time.

3. Primitives : ArrayList can not contains primitive data types (like int , float , double) it can only contains Object while Array can contain both primitive data types as well as objects.

One get a misconception that we can store primitives(int,float,double) in ArrayList , but it is not true

Suppose we have ArrayList object ,

ArrayList arraylistobject = new ArrayList();

arraylistobject.add(23); // try to add 23 (primitive)

JVM through Autoboxing(converting primitives to equivalent objects internally) ensures that only objects are added to the arraylist object.

thus , above step internally works like this :

arraylistobject.add( new Integer(23));

// Converted int primitive to Integer object and added to arraylistobject

Difference between Array and Arraylist in Java with Example

4. Iterating the values : We can use iterator to iterate through ArrayList . The iterators returned by the ArrayList class's iterator and listiterator method are fail-fast. We can use for loop or for each loop to iterate through array .

5. Type-Safety : In Java , one can ensure Type Safety through Generics. while Array is a homogeneous data structure , thus it will contain objects of specific class or primitives of specific data type. In array if one try to store the different data type other than the specified while creating the array object , ArrayStoreException is thrown.

for example

String temp[] = new String[2]; // creates a string array of size 2

temp[0] = new Integer(12); // throws ArrayStoreException, trying to add Integer object in String[]

6. Length : Length of the ArrayList is provided by the size() method while Each array object has the length variable which returns the length of the array

for example :

Integer arrayobject[] = new Integer[3];

arraylength= arrayobject.length ; //uses arrayobject length variable

ArrayList arraylistobject = new ArrayList();

arraylistobject.add(12);

arraylistobject.size(); //uses arraylistobject size method

7. Adding elements : We can insert elements into the arraylist object using the add() method while in array we insert elements using the assignment operator.

for example :

Integer addarrayobject[] = new Integer[3];

addarrayobject[0]= new Integer(8) ; //new object is added to the array object

8. Multi-dimensional : Array can be multi dimensional , while ArrayList is always single dimensional.

example of multidimensional array:

Integer addarrayobject[][] = new Integer[3][2];

addarrayobject[0][0]= new Integer(8)

Similarities Between Array and ArrayList

1. add and get method : Performance of Array and ArrayList are similar for the add and get operations .Both operations runs in constant time.

2. Duplicate elements : Both array and arraylist can contain duplicate elements.

3. Null Values : Both can store null values and uses index to refer to their elements.

4. Unordered : Both does not guarantee ordered elements.

Array ArrayList

Resizable No Yes

Primitives Yes No

Iterating values for, for each Iterator , for each

Length length variable size method

Performance Fast Slow in comparision

Multidimensional Yes No

Add Elements Assignment operator add method

**Q21 What is the difference between HashSet and TreeSet ?**

Difference between HashSet and TreeSet

1. Ordering : HashSet stores the object in random order . There is no guarantee that the element we inserted first in the HashSet will be printed first in the output . For example

import java.util.HashSet;

public class HashSetExample {

public static void main(String[] args) {

HashSet<String> obj1= new HashSet<String>();

obj1.add("Alive");

obj1.add("is");

obj1.add("Awesome");

System.out.println(obj1);

}

}

OUTPUT : [is, Awesome, Alive]

Elements are sorted according to the natural ordering of its elements in TreeSet. If the objects can not

be sorted in natural order than use compareTo() method to sort the elements of TreeSet object .

import java.util.TreeSet;

public class TreeSetExample {

public static void main(String[] args) {

TreeSet<String> obj1= new TreeSet<String>();

obj1.add("Alive");

obj1.add("is");

obj1.add("Awesome");

System.out.println(obj1);

}

}

OUTPUT : [Alive, Awesome, is]

2. Null value : HashSet can store null object while TreeSet does not allow null object. If one try to store null object in TreeSet object , it will throw Null Pointer Exception.

3. Performance : HashSet take constant time performance for the basic operations like add, remove contains and size.While TreeSet guarantees log(n) time cost for the basic operations (add,remove,contains).

4. Speed : HashSet is much faster than TreeSet,as performance time of HashSet is constant against the log time of TreeSet for most operations (add,remove ,contains and size) . Iteration performance of HashSet mainly depends on the load factor and initial capacity parameters.

5. Internal implementation : As we have already discussed How hashset internally works in java thus, in one line HashSet are internally backed by hashmap. While TreeSet is backed by a Navigable TreeMap.

Difference between HashSet and TreeSet in Java with Example

6. Functionality : TreeSet is rich in functionality as compare to HashSet. Functions like pollFirst(),pollLast(),first(),last(),ceiling(),lower() etc. makes TreeSet easier to use than HashSet.

7. Comparision : HashSet uses equals() method for comparison in java while TreeSet uses compareTo() method for maintaining ordering

To whom priority is given TreeSet comparator or Comparable.compareTo()

Suppose there are elements in TreeSet which can be naturally sorted by the TreeSet , but we also added our own sorting method by implementing Comparable interface compareTo() method .

Then to whom priority is given

Answer to the above question is that the Comparator passed into the TreeSet constructor has been given priority.

According to Oracle Java docs

public TreeSet(Comparator comparator)

Constructs a new, empty tree set, sorted according to the specified comparator

Parameters:

comparator - the comparator that will be used to order this set. If null, the natural ordering of the elements will be used

**Similarities Between HashSet and TreeSet**

1. Unique Elements : Since HashSet and TreeSet both implements Set interface . Both are allowed to store only unique elements in their objects. Thus there can never be any duplicate elements inside the HashSet and TreeSet objects.

2. Not Thread Safe : HashSet and TreeSet both are not synchronized or not thread safe.HashSet and TreeSet, both implementations are not synchronized. If multiple threads access a hash set/ tree set concurrently, and at least one of the threads modifies the set, it must be synchronized externally.

3. Clone() method copy technique: Both HashSet and TreeSet uses shallow copy technique to create a clone of their objects .

4. Fail-fast Iterators : The iterators returned by this class's method are fail-fast: if the set is modified at any time after the iterator is created, in any way except through the iterator's own remove method, the iterator will throw a ConcurrentModificationException. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

**When to prefer TreeSet over HashSet**

1. Sorted unique elements are required instead of unique elements.The sorted list given by TreeSet is always in ascending order.

2. TreeSet has greater locality than HashSet.

If two entries are near by in the order , then TreeSet places them near each other in data structure and hence in memory, while HashSet spreads the entries all over memory regardless of the keys they are associated to.

As we know Data reads from the hard drive takes much more latency time than data read from the cache or memory. In case data needs to be read from hard drive than prefer TreeSet as it has greater locality than HashSet.

3. TreeSet uses Red- Black tree algorithm underneath to sort out the elements. When one need to perform read/write operations frequently , then TreeSet is a good choice.

**Q23 What is the difference between HashMap and ConcurrentHashMap ?**

**1. Thread -Safe :**

ConcurrentHashMap is thread-safe that is the code can be accessed by single thread at a time . while HashMap is not thread-safe .difference between hashmap and concurrenthashmap in java

**2. Synchronization Method :**

HashMap can be synchronized by using synchronizedMap(HashMap) method . By using this method we get a HashMap object which is equivalent to the HashTable object . So every modification is performed on Map is locked on Map object

**import java.util.\*;**

**public class HashMapSynchronization {**

**public static void main(String[] args) {**

**// create map**

**Map<String,String> map = new HashMap<String,String>();**

**// populate the map**

**map.put("1","ALIVE ");**

**map.put("2","IS");**

**map.put("3","AWESOME");**

**// create a synchronized map**

**Map<String,String> syncMap = Collections.synchronizedMap(map);**

**System.out.println("Synchronized map :"+syncMap);**

**}**

**}**

ConcurrentHashMap synchronizes or locks on the certain portion of the Map . To optimize the performance of ConcurrentHashMap , Map is divided into different partitions depending upon the Concurrency level . So that we do not need to synchronize the whole Map Object

**3. Null Key**

ConcurrentHashMap does not allow NULL values . So the key can not be null in ConcurrentHashMap .While In HashMap there can only be one null key

**4. Performance**

In multiple threaded environment HashMap is usually faster than ConcurrentHashMap . As only single thread can access the certain portion of the Map and thus reducing the performance . While in HashMap any number of threads can access the code at the same time **.**

**Q24 Arrange the following in the ascending order (performance):**

HashMap , Hashtable , ConcurrentHashMap and Collections.SynchronizedMap Hashtable < Collections.SynchronizedMap < ConcurrentHashMap < HashMap

**Q25 How HashMap works in Java ?**

**How Hashmap works in Java**

HashMap works on the principle of Hashing . To understand Hashing , we should understand the three terms first i.e Hash Function , Hash Value and Bucket .

**What is Hash Function , Hash Value and Bucket ?**

hashCode() function which returns an integer value is the Hash function. The important point to note that , this method is present in Object class ( Mother of all class ) .

**This is the code for the hash function(also known as hashCode method) in Object Class :**

**public native int hashCode();**

The most important point to note from the above line : hashCode method return int value .So the Hash value is the int value returned by the hash function .

**What is bucket ?**

A bucket is used to store key value pairs . A bucket can have multiple key-value pairs . In hash map, bucket used simple linked list to store objects .

After understanding the terms we are ready to move next step , How hash map works in java or How get() works internally in java .

**Code inside Java Api (HashMap class internal implementation) for HashMap get(Obejct key) method**

**1. Public V get(Object key)**

**{**

**2. if (key ==null)**

**3. //Some code**

**4. int hash = hash(key.hashCode());**

**5. // if key found in hash table then return value**

**6. // else return null**

**}**

**Hash map works on the principle of hashing**

HashMap get(Key k) method calls hashCode method on the key object and applies returned hashValue to its own static hash function to find a bucket location(backing array) where keys and values are stored in form of a nested class called Entry (Map.Entry) . So you have concluded that from the previous line that Both key and value is stored in the bucket as a form of Entry object . So thinking that Only value is stored in the bucket is not correct and will not give a good impression on the interviewer **.**

\* Whenever we call get( Key k ) method on the HashMap object . First it checks that whether key is null or not . Note that there can only be one null key in HashMap .

If key is null , then Null keys always map to hash 0, thus index 0

If key is not null then , it will call hashfunction on the key object , see line 4 in above method i.e. key.hashCode() ,so after key.hashCode() returns hashValue , line 4 looks like

**4. int hash = hash(hashValue)**

, and now ,it applies returned hashValue into its own hashing function .We might wonder why we are calculating the hashvalue again using hash(hashValue). Answer is ,It defends against poor quality hash functions.

Now step 4 final hashvalue is used to find the bucket location at which the Entry object is stored . Entry object stores in the bucket like this (hash,key,value,bucketindex) .

**Interviewer: What if when two different keys have the same hashcode ?**

Solution, equals() method comes to rescue.Here candidate gets puzzled. Since bucket is one and we have two objects with the same hashcode .Candidate usually forgets that bucket is a simple linked list.

The bucket is the linked list effectively . Its not a LinkedList as in a java.util.LinkedList - It's a separate (simpler) implementation just for the map .

So we traverse through linked list , comparing keys in each entries using keys.equals() until it return true. Then the corresponding entry object Value is returned

**how hashmap works internally in java**

When the functions 'equals' traverses through the linked list does it traverses from start to end one by one...in other words brute method. Or the linked list is sorted based on key and then it traverses?

**Answer is when an element is added/retrieved, same procedure follows:**

a. Using key.hashCode() [ see above step 4],determine initial hashvalue for the key

b. Pass intial hashvalue as hashValue in hash(hashValue) function, to calculate the final hashvalue.

c. Final hash value is then passed as a first parameter in the indexFor(int ,int )method .

The second parameter is length which is a constant in HashMap Java Api , represented by DEFAULT\_INITIAL\_CAPACITY

**The default value of DEFAULT\_INITIAL\_CAPACITY is 16 in HashMap Java Api .**

indexFor(int,int) method returns the first entry in the appropriate bucket. The linked list in the bucket is then iterated over - (the end is found and the element is added or the key is matched and the value is returned )

**Explanation about indexFor(int,int) is below :**

**/\*\***

**\* Returns index for hash code h.**

**\*/**

**static int indexFor(int h, int length) {**

**return h & (length-1);**

**}**

The above function indexFor() works because Java HashMaps always have a capacity, i.e. number of buckets, as a power of 2.

Let's work with a capacity of 256,which is 0x100, but it could work with any power of 2. Subtracting 1

from a power of 2 yields the exact bit mask needed to bitwise-and with the hash to get the proper bucket index, of range 0 to length - 1.

256 - 1 = 255

0x100 - 0x1 = 0xFF

E.g. a hash of 257 (0x101) gets bitwise-anded with 0xFF to yield a bucket number of 1.

**Interviewer: What if when two keys are same and have the same hashcode ?**

If key needs to be inserted and already inserted hashkey's hashcodes are same, and keys are also same(via reference or using equals() method) then override the previous key value pair with the current key value pair.

The other important point to note is that in Map ,Any class(String etc.) can serve as a key if and only if it overrides the equals() and hashCode() method

**Interviewer: How will you measure the performance of HashMap?**

An instance of HashMap has two parameters that affect its performance: initial capacity and load factor.

The capacity is the number of buckets in the hash table( HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls.), and the initial capacity is simply the capacity at the time the hash table is created.

The load factor is a measure of how full the hash table is allowed to get before its capacity is automatically increased. When the number of entries in the hash table exceeds the product of the load factor and the current capacity, the hash table is rehashed (that is, internal data structures are rebuilt) so that the hash table has approximately twice the number of buckets.

In HashMap class, the default value of load factor is (.75) **.**

**Interviewer : What is the time complexity of Hashmap get() and put() method ?**

The hashmap implementation provides constant time performance for (get and put) basic operationsi.e the complexity of get() and put() **is O(1) , assuming the hash function disperses the elements properly among the buckets.**

**Q26 What is the difference between LinkedList and ArrayList in Java ?**

**1. Implementation :**

ArrayList is the resizable array implementation of list interface , while LinkedList is the Doubly-linked list implementation of the list interface**.**

**2. Performance :**

Performance of ArrayList and LinkedList depends on the type of operation

a. get(int index) or search operation : ArrayList get(int index) operation runs in constant time i.e O(1) while LinkedList get(int index) operation run time is O(n) .

The reason behind ArrayList being faster than LinkedList is that ArrayList uses index based system for its elements as it internally uses array data structure , on the other hand ,

LinkedList does not provide index based access for its elements as it iterates either from the beginning or end (whichever is closer) to retrieve the node at the specified element index.

b. insert() or add(Object) operation : Insertions in LinkedList are generally fast as compare to ArrayList.

In LinkedList adding or insertion is O(1) operation . While in ArrayList, if array is full i.e worst case, there is extra cost of resizing array and copying elements to the new array , which makes runtime of add operation in ArrayList O(n) , otherwise it is O(1) .

c. remove(int) operation : Remove operation in LinkedList is generally same as ArrayList i.e. O(n).

In LinkedList , there are two overloaded remove methods. one is remove() without any parameter which removes the head of the list and runs in constant time O(1) .

The other overloaded remove method in LinkedList is remove(int) or remove(Object) which removes the Object or int passed as parameter . This method traverses the LinkedList until it found the Object and unlink it from the original list . Hence this method run time is O(n).

While in ArrayList remove(int) method involves copying elements from old array to new updated array , hence its run time is O(n).

**3. Reverse Iterator :**

LinkedList can be iterated in reverse direction using descendingIterator() while there is no descendingIterator() in ArrayList , so we need to write our own code to iterate over the ArrayList in reverse direction.

**4. Initial Capacity :**

If the constructor is not overloaded , then ArrayList creates an empty list of initial capacity 10 , while LinkedList only constructs the empty list without any initial capacity.

**5. Memory Overhead :**

Memory overhead in LinkedList is more as compared to ArrayList as node in LinkedList needs to maintain the addresses of next and previous node. While in ArrayList each index only holds the actual object(data).

**Example of ArrayList and LinkedList :**

**import java.util.ArrayList;**

**import java.util.LinkedList;**

**public class ArrayListLinkedListExample {**

**public static void main(String[] args) {**

**ArrayList<String> arrlistobj = new ArrayList<String>();**

**arrlistobj.add("1. Alive is awesome");**

**arrlistobj.add("2. Love yourself");**

**System.out.println("ArrayList object output :"+ arrlistobj);**

**LinkedList llobj = new LinkedList();**

**llobj.add("1. Alive is awesome");**

**llobj.add("2. Love yourself");**

**System.out.println("LinkedList object output :"+llobj);**

**}**

**}**

**Output:**

ArrayList object output :[1. Alive is awesome, 2. Love yourself]

LinkedList object output :[1. Alive is awesome, 2. Love yourself]

**Similarities between ArrayList and LinkedList :**

**1. Not synchronized :**

Both ArrayList and LinkedList are not synchronized , and can be made synchronized explicitly using Collections.synchronizedList() method.

**2. clone() operation :**

Both ArrayList and LinkedList returns a shallow copy of the original object ,i.e. the elements themselves are not cloned.

**3. Iterators :**

The iterators returned by ArrayList and LinkedList class's iterator and listIterator methods are fail-fast. Fail fast iterators throw ConcurrentModificationException . We have already discussed the difference between fail-fast and fail-safe iterators.

**difference between arraylist and linkedlist in java**

**4. Insertion Order :**

As ArrayList and LinkedList are the implementation of List interface,so, they both inherit properties of List . They both preserves the order of the elements in the way they are added to the ArrayList or LinkedList object.

**When to Use ArrayList and LinkedList :**

In real world applications , you will more frequently use ArrayList than LinkedList. But in a very specific situations LinkedList can be preferred**.**

1. ArrayList is preferred when there are more get(int) or search operations need to be performed as every search operation runtime is O(1).

2. If application requires more insert(int) , delete(int) operations then the get(int) operations then LinkedList is preferred as they do not need to maintain back and forth like arraylist to preserve continues indices.

**ArrayList LinkedList**

**Implementation Resizable Array Douby-LinkedList**

ReverseIterator No Yes , descendingIterator()

Initial Capacity 10 Constructs empty list

get(int) operation Fast Slow in comparision

add(int) operation Slow in comparision Fast

Memory Overhead No Yes

**Q27 What are Comparable and Comparator interfaces ? List the difference between them ?**

**1. Sort sequence :**

In comparable ,Only one sort sequence can be created while in comparator many sort sequences can be created **.**

**2. Methods Used :**

Comparator interface in Java has method public int compare (Object o1, Object o2) which returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second. While Comparable interface has method public int compareTo(Object o) which returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

**3. Objects needed for Comparision :**

If you see then logical difference between these two is Comparator in Java compare two objects provided to it , while Comparable interface compares "this" reference with the object specified. So only one object is provided which is then compared to "this" reference.

**4 Modify Classes :**

One has to modify the class whose instances you want to sort while in comparator one build a class separate from the class whose instances one want to sort .

**5. Package :**

Comparator in Java is defined in java.util package while Comparable interface in Java is defined in java.lang package, which very much says that Comparator should be used as an utility to sort objects which Comparable should be provided by default

**Points to Remember regarding Comparable and Comparator :**

1. Comparable in Java is used to implement natural ordering of object. In Java API String, Date and wrapper classes implements Comparable interface.Its always good practice to override compareTo() for value objects**.**

2. If any class implement Comparable interface in Java then collection of that object either list or Array can be sorted automatically by using Collections.sort() or Arrays.sort() method and object will be sorted based on there natural order defined by CompareTo method.

3. Objects which implement Comparable in Java can be used as keys in a SortedMap like treemap or elements in a SortedSet for example TreeSet, without specifying any Comparator.

**Before Moving onto the examples of Comparable and Comparator we need to learn more about the few important functions**

**Functions with the Comparable Interface :**

Often, it is not enough to simply know whether two strings are identical. For sorting applications, you need to know which is less than, equal to, or greater than the next. A string is less than another if it comes before the other in dictionary order. A string is greater than another if it comes after the other in dictionary order. The String method compareTo( ) serves this purpose. It has this general form**:**

**1) public int compareTo(Object o) :**

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object **.**

Here , str is the String being compared with the invoking String. The result of the comparison is returned and is interpreted as shown here:

**Value Meaning**

Less than zero The invoking string is less .

Greater than zero The invoking string is greater .

Zero The two strings are equal.

**Functions with the Comparator Interface :**

**1) public int compare (Object o1,Object o2)**

Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.

**2) public boolean equals(Object obj)**

Indicates whether some other object is "equal to" this Comparator. This method must obey the general contract of Object.equals(Object). Additionally, this method can return true only if the specified Object is also a comparator and it imposes the same ordering as this comparator. Thus, comp1.equals(comp2) implies that sgn(comp1.compare(o1, o2))==sgn(comp2.compare(o1, o2)) for every object reference o1 and o2 .

**Q28 Why Map interface does not extend the Collection interface in Java Collections Framework ?**

One liner answer : Map interface is not compatible with the Collection interface.

Explanation : Since Map requires key as well as value , for example , if we want to add key-value pair then we will use put(Object key , Object value) . So there are two parameters required to add element to the HashMap object . In Collection interface add(Object o) has only one parameter.

The other reasons are Map supports valueSet , keySet as well as other appropriate methods which have just different views from the Collection interface.

**Q29 When to use ArrayList and when to use LinkedList in application?**

ArrayList has constant time search operation O(1) .Hence, ArrayList is preferred when there are more get() or search operation .

Insertion , Deletion operations take constant time O(1) for LinkedList. Hence, LinkedList is preferred when there are more insertions or deletions involved in the application.

**Q31 How HashSet works internally in java ?**

Set Implementation Internally in Java .Each and every element in the set is unique . So that there is no duplicate element in set .So in java if we want to add elements in the set then we write code like this

public class JavaHungry {

public static void main(String[] args)

{

// TODO Auto-generated method stub

HashSet<Object> hashset = new HashSet<Object>();

hashset.add(3);

hashset.add("Java Hungry");

hashset.add("Blogspot");

System.out.println("Set is "+hashset);

}

}

It will print the result : Set is [3, Java Hungry, Blogspot]

Now let add duplicate element in the above code

public class JavaHungry {

public static void main(String[] args)

{

HashSet<Object> hashset = new HashSet<Object>();

hashset.add(3);

hashset.add("Java Hungry");

hashset.add("Blogspot");

hashset.add(3); // duplicate elements

hashset.add("Java Hungry"); // duplicate elements

System.out.println("Set is "+hashset);

}

}

It will print the result : Set is [3, Java Hungry, Blogspot]

Now , what happens internally when you pass duplicate elements in the add() method of the Set object , It will return false and do not add to the HashSet , as the element is already present .So far so good .

But the main problem arises that how it returns false . So here is the answer

When you open the HashSet implementation of the add() method in Java Apis that is rt.jar , you will find the following code in it

public class HashSet<E>

extends AbstractSet<E>

implements Set<E>, Cloneable, java.io.Serializable

{

private transient HashMap<E,Object> map;

// Dummy value to associate with an Object in the backing Map

private static final Object PRESENT = new Object();

public HashSet() {

map = new HashMap<>();

}

// SOME CODE ,i.e Other methods in Hash Set

public boolean add(E e) {

return map.put(e, PRESENT)==null;

}

// SOME CODE ,i.e Other methods in Hash Set

}

So , we are achieving uniqueness in Set,internally in java through HashMap . Whenever you create an object of HashSet it will create an object of HashMap as you can see in the italic lines in the above code .

We already discussed How HashMap works internally in java

As we know in HashMap each key is unique . So what we do in the set is that we pass the argument in the add(Elemene E) that is E as a key in the HashMap . Now we need to associate some value to the key , so what Java apis developer did is to pass the Dummy value that is ( new Object () ) which is referred by Object reference PRESENT .

So , actually when you are adding a line in HashSet like hashset.add(3) what java does internally is that it will put that element E here 3 as a key in the HashMap(created during HashSet object creation) and some dummy value that is Object's object is passed as a value to the key .

Now if you see the code of the HashMap put(Key k,Value V) method , you will find something like this

public V put(K key, V value) {

//Some code

}

The main point to notice in above code is that put (key,value) will return

1. null , if key is unique and added to the map

2. Old Value of the key , if key is duplicate

So , in HashSet add() method , we check the return value of map.put(key,value) method with null value

i.e.

public boolean add(E e) {

return map.put(e, PRESENT)==null;

}

So , if map.put(key,value) returns null ,then

map.put(e, PRESENT)==null will return true and element is added to the HashSet.

So , if map.put(key,value) returns old value of the key ,then

map.put(e, PRESENT)==null will return false and element is not added to the HashSet .

**Q33 How HashMap works in Java ?**

How Hashmap works in Java

HashMap works on the principle of Hashing . To understand Hashing , we should understand the three terms first i.e Hash Function , Hash Value and Bucket .

**What is Hash Function , Hash Value and Bucket ?**

hashCode() function which returns an integer value is the Hash function. The important point to note that , this method is present in Object class ( Mother of all class ) .

This is the code for the hash function(also known as hashCode method) in Object Class :

public native int hashCode();

The most important point to note from the above line : hashCode method return int value .

So the Hash value is the int value returned by the hash function .

So summarize the terms in the diagram below

how hash map works in java

**What is bucket ?**

A bucket is used to store key value pairs . A bucket can have multiple key-value pairs . In hash map, bucket used simple linked list to store objects .

After understanding the terms we are ready to move next step , How hash map works in java or How get() works internally in java .

Code inside Java Api (HashMap class internal implementation) for HashMap get(Obejct key) method

1. Public V get(Object key)

{

2. if (key ==null)

3. //Some code

4. int hash = hash(key.hashCode());

5. // if key found in hash table then return value

6. // else return null

}

Hash map works on the principle of hashing

HashMap get(Key k) method calls hashCode method on the key object and applies returned hashValue to its own static hash function to find a bucket location(backing array) where keys and values are stored in form of a nested class called Entry (Map.Entry) . So you have concluded that from the previous line that Both key and value is stored in the bucket as a form of Entry object . So thinking that Only value is stored in the bucket is not correct and will not give a good impression on the interviewer .

\* Whenever we call get( Key k ) method on the HashMap object . First it checks that whether key is null or not . Note that there can only be one null key in HashMap .

If key is null , then Null keys always map to hash 0, thus index 0.

If key is not null then , it will call hashfunction on the key object , see line 4 in above method i.e. key.hashCode() ,so after key.hashCode() returns hashValue , line 4 looks like

4. int hash = hash(hashValue) , and now ,it applies returned hashValue into its own hashing function .

We might wonder why we are calculating the hashvalue again using hash(hashValue). Answer is ,It defends against poor quality hash functions.

Now step 4 final hashvalue is used to find the bucket location at which the Entry object is stored . Entry object stores in the bucket like this (hash,key,value,bucketindex) .

Interviewer: What if when two different keys have the same hashcode ?

Solution, equals() method comes to rescue.Here candidate gets puzzled. Since bucket is one and we have two objects with the same hashcode .Candidate usually forgets that bucket is a simple linked list.

The bucket is the linked list effectively . Its not a LinkedList as in a java.util.LinkedList - It's a separate (simpler) implementation just for the map .

So we traverse through linked list , comparing keys in each entries using keys.equals() until it return true. Then the corresponding entry object Value is returned .

how hashmap works internally in java

One of our readers Jammy asked a very good question

When the functions 'equals' traverses through the linked list does it traverses from start to end one by one...in other words brute method. Or the linked list is sorted based on key and then it traverses?

Answer is when an element is added/retrieved, same procedure follows:

a. Using key.hashCode() [ see above step 4],determine initial hashvalue for the key

b. Pass intial hashvalue as hashValue in hash(hashValue) function, to calculate the final hashvalue.

c. Final hash value is then passed as a first parameter in the indexFor(int ,int )method .

The second parameter is length which is a constant in HashMap Java Api , represented by DEFAULT\_INITIAL\_CAPACITY

The default value of DEFAULT\_INITIAL\_CAPACITY is 16 in HashMap Java Api .

indexFor(int,int) method returns the first entry in the appropriate bucket. The linked list in the bucket is then iterated over - (the end is found and the element is added or the key is matched and the value is returned )

Explanation about indexFor(int,int) is below :

/\*\*

\* Returns index for hash code h.

\*/

static int indexFor(int h, int length) {

return h & (length-1);

}

The above function indexFor() works because Java HashMaps always have a capacity, i.e. number of buckets, as a power of 2.

Let's work with a capacity of 256,which is 0x100, but it could work with any power of 2. Subtracting 1

from a power of 2 yields the exact bit mask needed to bitwise-and with the hash to get the proper bucket index, of range 0 to length - 1.

256 - 1 = 255

0x100 - 0x1 = 0xFF

E.g. a hash of 257 (0x101) gets bitwise-anded with 0xFF to yield a bucket number of 1.

Interviewer: What if when two keys are same and have the same hashcode ?

If key needs to be inserted and already inserted hashkey's hashcodes are same, and keys are also same(via reference or using equals() method) then override the previous key value pair with the current key value pair.

The other important point to note is that in Map ,Any class(String etc.) can serve as a key if and only if it overrides the equals() and hashCode() method .

Interviewer: How will you measure the performance of HashMap?

According to Oracle Java docs,

An instance of HashMap has two parameters that affect its performance: initial capacity and load factor.

The capacity is the number of buckets in the hash table( HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls.), and the initial capacity is simply the capacity at the time the hash table is created.

The load factor is a measure of how full the hash table is allowed to get before its capacity is automatically increased. When the number of entries in the hash table exceeds the product of the load factor and the current capacity, the hash table is rehashed (that is, internal data structures are rebuilt) so that the hash table has approximately twice the number of buckets.

In HashMap class, the default value of load factor is (.75) .

Interviewer : What is the time complexity of Hashmap get() and put() method ?

The hashmap implementation provides constant time performance for (get and put) basic operations

i.e the complexity of get() and put() is O(1) , assuming the hash function disperses the elements properly among the buckets.

**Q34 How remove(key) method works in HashMap ?**

How Remove method works internally in Java

In HashMap we need key and value to add element to the HashMap object.

So if we add element to the HashMap object then the code will be like this :

public class JavaHungry {

public static void main(String[] args)

{

HashMap hashmap = new HashMap();

hashmap.put("Java", 1);

hashmap.put("Hungry" , 2);

hashmap.put("Blogspot" , 3);

Iterator iteratorobject = hashmap.keySet().iterator();

while(iteratorobject.hasNext()){

String hashmapkey = iteratorobject.next();

System.out.print(hashmap.get(hashmapkey));

}

}

}

It will print the result : [321]

Remember the result can be any order , so if you run the above code , then output could also be

[123] [231] [132] [213] [312]

as the HashMap is unordered.

Till now , we have added three key-value pairs to the hashmap object , now let us remove one key-value pair from the hashmap object**.**

**public class JavaHungry {**

**public static void main(String[] args)**

**{**

**HashMap hashmap = new HashMap();**

**hashmap.put("Java", 1);**

**hashmap.put("Hungry" , 2);**

**hashmap.put("Blogspot" , 3);**

**hashmap.remove("Java");**

**Iterator iteratorobject = hashmap.keySet().iterator();**

**while(iteratorobject.hasNext()){**

**String hashmapkey = iteratorobject.next();**

**System.out.print(hashmap.get(hashmapkey));**

**}**

**}**

**}**

It will print the result : [32] As HashMap is unordered , the result could also be [23] but the main point to note that hashmap.remove("Java") would remove the "Java" key and value associated with the key that is 1 .

So far so good , But the main question is how remove method removes key-value pair in the hashmap object .

Before moving onto the internal implementation of remove method of HashMap we need to understand the Entry object.

**What is Entry Object**

Map.Entry is the static nested class that stores the key/value pair that forms one element of HashMap.

Entry object stores in the bucket in the following way (hash,key,value,bucketindex)

The main point to note from the above line is that we need hashvalue and bucketindex besides key to get access to the desired Entry object in HashMap.

When you open the HashMap implementation of the remove(key) method in Java Apis that is rt.jar , you will find the following code in it :

**public class HashMap<K,V>**

**extends AbstractMap<K,V>**

**implements Map<K,V>, Cloneable, java.io.Serializable**

**{**

**// SOME CODE ,i.e Other methods in HashMap**

**1. public V remove (Object key){**

**2. Entry<K,V> e = removeEntryForKey(key);**

**3. return (e==null ? null : e.value);**

**}**

**// SOME CODE ,i.e Other methods in HashMap**

**}**

In the line 2 of remove(key) we are calling method removeEntryForKey(key). The main purpose of removeEntryForKey(key) method is it removes and returns the entry associated with the specified key in the HashMap. So let us understand how removeEntryForKey(key) removes the entry object .

**How remove method works in HashMap with Example**

**There are two possible scenarios for the key object ,**

**1. If key is not null**

**2. If key is null**

**Interviewer : How remove(key) method works internally in Java ?**

As we know to find the desired Entry object which is to be removed in the HashMap we need hashValue , key and bucketindex . So remove(key) method calls removeEntryForKey(key) method internally , which calculate the final hashValue of the key object , and then use that hashValue in the indexFor(int,int) method to find the first entry object in the appropriate bucket.

Since bucket(table) is a LinkedList effectively , we start traversing from the first entry object which we got by using indexFor(int,int) method in the bucket. For each entry object in the bucket we compare whether hashValue and the key is equal to the calculated hashValue in the first step and the key passed as a parameter in the remove(key) method.

If desired Entry object is found , then we removed that single entry object from the LinkedList.

Removing a single Entry object from the LinkedList is implemented just like removing a single object from the LinkedList.

**Entry object returned by the removeEntryForKey(key) method is then stored in the local variable e of type Entry in the remove(key) method.**

**If(e==null)**

**return null**

**else**

**return value of removed Entry object.**

**public class HashMap<K,V>**

**extends AbstractMap<K,V>**

**implements Map<K,V>, Cloneable, java.io.Serializable**

**{**

**// SOME CODE ,i.e Other methods in HashMap**

**final Entry<K,V> removeEntryForKey(Object key) {**

**// Calculate hash value of the key passed as a parameter in remove method**

**1. int hash = (key == null) ? 0 : hash(key. hashCode());**

**// index for returns the first Entry in the appropriate bucket**

**// Here , table is an array of Entry objects ,i.e. Entry[] table**

**2. int i = indexFor(hash, table.length);**

**// Below Code to remove a single object from the simple LinkedList that is**

**// removing the desired Entry object from the LinkedList**

**3. Entry<K,V> prev = table[i];**

**4. Entry<K,V> e = prev;**

**5. while (e != null) {**

**6. Entry<K,V> next = e.next;**

**Object k;**

**// If Entry object's key and hash value equal to the above hashvalue and key**

**7. if (e.hash == hash &&**

**((k = e.key) == key || (key != null && key.equals(k)))) {**

**8. modCount++;**

**// Reduce size of Entry[] table by 1**

**9. size--;**

**// Remove the Entry object : Two case to remove**

**// Case 1 : only single element in the bucket , then prev==e**

**10. if (prev == e)**

**11. table[i] = next;**

**// Case 2 : if more than one element present in the bucket ,**

**12. else**

**13. prev.next = next;**

**// recordRemoval() method is invoked whenever entry is removed from the table**

**14. e.recordRemoval(this);**

**15. return e;**

**}**

**16. prev = e;**

**17. e = next;**

**}**

**18. return e;**

**}**

**}**

**// SOME CODE ,i.e Other methods in HashMap**

**}**

**Interviewer :**

What is the purpose of calling recordRemoval() method in the removeEntryForKey(key) since it is the concrete method without any body.

recordRemoval() method is a concrete method without any body. It is invoked whenever the Entry is removed from the table . Since LinkedHashMap extends HashMap , thus this method is overridden in the LinkedHashMap's Entry in order to maintain its linked list of entries.

Interviewer : What is the time complexity of performing remove operation in HashMap using remove(key)

Best Case time complexity of remove(key) : O(1)

Worst Case time complexity of remove(key) : O(n)

**Interviewer:**

What happens if the null key is passed in the remove(key) method ?

The following lines will be executed in the removeEntryForKey(key) method

In line 1 , the value of hash will be 0 .

In line 2 , the value of indexFor will return 0 thus i=0.

In line 3 , Entry prev = table[0]

Entry prev = null

In line 4, Entry e = null

In line 5 , while loop condition returns false

In line 18 , return e (which is null)

**thus null is returned to the remove(key) method , which will in turn return null.**

**Interviewer :**

Explain the removeEntryForKey(key) method in HashMap in detail , considering key is not null?

In the line 1 of removeEntryForKey(key), if the key passed as a parameter is not null then , it will call hashfunction on the key object , so after key.hashCode() returns hashValue , so line 1 will look like

**int hash = hash (hashValue)**

we are now applying returned hashvalue into its own hashfunction. To defend against poor quality hash functions , we are calculating the hashValue again using hash(hashValue) in the above line.

We have hashValue and key , now we need to find the bucketindex of the desired Entry object

In line 2 , indexFor(int,int) , for the given hashValue, returns the first entry in the appropriate bucket.

So we get the first entry of the desired bucket.

In line 3 , we start traversing from the first entry in the bucket ,till we get the desired Entry object, prev is used to store the first entry in the bucket . Here table is an array of Entry objects i.e Entry[] table .

In line 4, We created an Entry instance variable e which holds the prev value ,i.e, the first entry in the appropriate bucket.

Below Explanation is about removing a single object from the simple LinkedList that is removing the desired Entry object from the LinkedList

In line 5 , We iterate thorugh the Entry[] starting from the e ,till we get the desired Entry object

To check if we get the desired Entry object ,we need hashValue , key and bucketindex. We need to iterate through the bucket and its index one by one and comparing the hashValue and key of each Entry object.

In line 7 , If condition is true then we get the desired Entry object which we need to remove from the hashmap object .

Then two cases arises , whether bucket has single Entry object or it has more than one Entry object ,

If bucket has single Entry object , then,

In line 10 , (prev==e) condition will be true

else bucket has more than one Entry object

In line 12 else condition will be run

In line 14 , recordRemoval() method is called on the desired Entry object whenever the Entry is removed from the table.

In line 15 , the removed Entry object is returned to the remove(key) method.

**Q36 How TreeMap works in Java ?**

**What is a Tree Map ?**

Treemap class is like HashMap which stores key- value pairs . The major difference is that Treemap sorts

**the key in ascending order.**

**According to Java doc :**

Treemap is sorted according to the natural ordering of its keys, or by a Comparator provided at map creation time, depending on which constructor is used.

This implementation provides guaranteed log(n) time cost for the containsKey, get, put and remove operations. Algorithms are adaptations of those in Cormen, Leiserson, and Rivest's Introduction to Algorithms.

**How TreeMap works in java ?**

TreeMap is a Red-Black tree based NavigableMap implementation.In other words , it sorts the TreeMap object keys using Red-Black tree algorithm.

So we learned that TreeMap uses Red Black tree algorithm internally to sort the elements.

Red Black algorithm is a complex algorithm . We should read the pseudo code of Red Black algorithm in order to understand the internal implementation .

**Red Black tree has the following properties :**

1. As the name of the algorithm suggests ,color of every node in the tree is either red or black.

2. Root node must be Black in color.

3. Red node can not have a red color neighbor node.

4. All paths from root node to the null should consist the same number of black nodes .

**Rotation in Red Black Tree :**

how treemap works in java

Rotations maintains the inorder ordering of the keys(x,y,z).

A rotation can be maintained in O(1) time.

You can find more about the red black tree algorithm here

Interviewer : Why and when we use TreeMap ?

We need TreeMap to get the sorted list of keys in ascending order.

Interviewer : What is the runtime performance of the get() method in TreeMap and HashMap ,where n represents the number of elements ?

According to TreeMap Java doc,

TreeMap implementation provides guaranteed log(n) time cost for the containsKey,get,put and remove operations.

According to HashMap Java doc :

HashMap implementation provides constant-time performance for the basic operations (get and put), assuming the hash function disperses the elements properly among the buckets.

One liner : TreeMap : log(n) HashMap : Constant time performance assuming elements disperses properly

**Interviewer : What is "natural ordering" in TreeMap ?**

"Natural" ordering is the ordering implied by the implementation of the Comparable interface by the objects used as keys in the TreeMap. Essentially, RBTree must be able to tell which key is smaller than the other key, and there are two ways to supply that logic to the RBTree implementation:

1.Implement Comparable interface in the class(es) used as keys to TreeMap, or

2.Supply an implementation of the Comparator that would do comparing outside the key class itself.

Natural ordering is the order provided by the Comparable interface .If somebody puts the key that do not implement natural order then it will throw ClassCastException.

**Interviewer : Why do we need TreeMap when we have sortedMap ?**

sortedMap is a interface and TreeMap is the class implementing it .As we know one can not create objects of the interface . Interface tells us which methods a sortedMap implementation should provide .TreeMap is such an implementation.

**Interviewer : Which data structure you will prefer in your code : HashMap or TreeMap ?**

HashMap is faster while TreeMap is sorted .Thus we choose them according to their advantage.

If you do not want to sort the elements but just to insert and retrieve the elements then use HashMap **.**

But if you want to maintain the order of the elements then TreeMap should be preferred because the result is alphabetically sorted .While iterating HashMap there is no ordering of the elements ,on the other hand , TreeMap iterates in the natural key order.

**Interviewer : What happens if the TreeMap is concurrently modified while iterating the elements ?**

The iterator fails fast and quickly if structurally modified at any time after the iterator is created (in any way except through the iterator's own remove method ). We already discussed the difference between Fail-fast and Fail safe iterators .

**Interviewer : Which copy technique (deep or shallow ) is used by the TreeMap clone() method ?**

According to docjar , clone() method returns the shallow copy of the TreeMap instance . In shallow copy object B points to object A location in memory . In other words , both object A and B are sharing the same elements .The keys and values themselves are not cloned .

**Interviewer : Why java's treemap does not allow an initial size ?**

HashMap reallocates its internals as the new one gets inserted while TreeMap does not reallocate nodes on adding new ones. Thus , the size of the TreeMap dynamically increases if needed , without shuffling the internals. So it is meaningless to set the initial size of the TreeMap .

**Q38 What is the difference between Fail- fast iterator and Fail-safe iterator ?**

**What is Concurrent Modification ?**

When one or more thread is iterating over the collection, in between, one thread changes the structure of the collection (either adding the element to the collection or by deleting the element in the collection or by updating the value at particular position in the collection) is known as Concurrent Modification

**Difference between Fail Fast iterator and Fail Safe iterator**

**Fail fast Iterator**

Fail fast iterator while iterating through the collection , instantly throws Concurrent Modification Exception if there is structural modification of the collection . Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

**Fail-fast iterator can throw ConcurrentModificationException in two scenarios :**

**difference between fail fast iterator and fail safe iterator**

**Single Threaded Environment**

After the creation of the iterator , structure is modified at any time by any method other than iterator's own remove method.

**Multiple Threaded Environment**

If one thread is modifying the structure of the collection while other thread is iterating over it .According to Oracle docs , the fail-fast behavior of an iterator cannot be guaranteed as it is, generally speaking, impossible to make any hard guarantees in the presence of unsynchronized concurrent modification. Fail-fast iterators throw ConcurrentModificationException on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness: the fail-fast behavior of iterators should be used only to detect bugs.

**Interviewer : How Fail Fast Iterator come to know that the internal structure is modified ?**

Iterator read internal data structure (object array) directly . The internal data structure(i.e object array) should not be modified while iterating through the collection. To ensure this it maintains an internal flag "mods" .Iterator checks the "mods" flag whenever it gets the next value (using hasNext() method and next() method). Value of mods flag changes whenever there is an structural modification. Thus indicating iterator to throw ConcurrentModificationException

**Fail Safe Iterator :**

Fail Safe Iterator makes copy of the internal data structure (object array) and iterates over the copied data structure.Any structural modification done to the iterator affects the copied data structure. So , original data structure remains structurally unchanged .Hence , no ConcurrentModificationException throws by the fail safe iterator.

**Two issues associated with Fail Safe Iterator are :**

**1. Overhead of maintaining the copied data structure i.e memory.**

**2. Fail safe iterator does not guarantee that the data being read is the data currently in the original data structure.**

According to Oracle docs , fail safe iterator is ordinarily too costly, but may be more efficient than alternatives when traversal operations vastly outnumber mutations, and is useful when you cannot or don’t want to synchronize traversals, yet need to preclude interference among concurrent threads. The "snapshot" style iterator method uses a reference to the state of the array at the point that the iterator was created. This array never changes during the lifetime of the iterator, so interference is impossible and the iterator is guaranteed not to throw ConcurrentModificationException.The iterator will not reflect additions, removals, or changes to the list since the iterator was created. Element-changing operations on iterators themselves (remove(), set(), and add()) are not supported. These methods throw UnsupportedOperationException.

**Example of Fail Fast Iterator and Fail Safe Iterator**

**import java.util.HashMap;**

**import java.util.Iterator;**

**import java.util.Map;**

**public class FailFastExample**

**{**

**public static void main(String[] args)**

**{**

**Map<String,String> premiumPhone = new HashMap<String,String>();**

**premiumPhone.put("Apple", "iPhone");**

**premiumPhone.put("HTC", "HTC one");**

**premiumPhone.put("Samsung","S5");**

**Iterator iterator = premiumPhone.keySet().iterator();**

**while (iterator.hasNext())**

**{**

**System.out.println(premiumPhone.get(iterator.next()));**

**premiumPhone.put("Sony", "Xperia Z");**

**}**

**}**

**}**

**Output :**

**iPhone**

Exception in thread "main" java.util.ConcurrentModificationException

at java.util.HashMap$HashIterator.nextEntry(Unknown Source)

at java.util.HashMap$KeyIterator.next(Unknown Source)

at FailFastExample.main(FailFastExample.java:20)

**Fail Safe Iterator Example :**

**import java.util.concurrent.ConcurrentHashMap;**

**import java.util.Iterator;**

**public class FailSafeExample**

**{**

**public static void main(String[] args)**

**{**

**ConcurrentHashMap<String,String> premiumPhone =**

**new ConcurrentHashMap<String,String>();**

**premiumPhone.put("Apple", "iPhone");**

**premiumPhone.put("HTC", "HTC one");**

**premiumPhone.put("Samsung","S5");**

**Iterator iterator = premiumPhone.keySet().iterator();**

**while (iterator.hasNext())**

**{**

**System.out.println(premiumPhone.get(iterator.next()));**

**premiumPhone.put("Sony", "Xperia Z");**

**}**

**}**

**}**

**Output**

**S5**

**HTC one**

**iPhone**

**Recap : Difference between Fail Fast Iterator and Fail Safe Iterator**

**Fail Fast Iterator Fail Safe Iterator**

Throw ConcurrentModification Exception Yes No

Clone object No Yes

Memory Overhead No Yes

Examples HashMap,Vector,ArrayList,HashSet

CopyOnWriteArrayList, ConcurrentHashMap

**Q40 How do you use a custom object as key in Collection classes like HashMap ?**

If one is using the custom object as key then one needs to override equals() and hashCode() method

and one also need to fulfill the contract.

If you want to store the custom object in the SortedCollections like SortedMap then one needs to make sure that equals() method is consistent to the compareTo() method. If inconsistent , then collection will not follow their contracts ,that is , Sets may allow duplicate elements.

**Q41 What is hash-collision in Hashtable ? How it was handled in Java?**

In Hashtable , if two different keys have the same hash value then it lead to hash -collision. A bucket of type linkedlist used to hold the different keys of same hash value.

**Q42 Explain the importance of hashCode() and equals() method ? Explain the contract also ?**

HashMap object uses Key object hashCode() method and equals() method to find out the index to put the key-value pair. If we want to get value from the HashMap same both methods are used . Somehow, if both methods are not implemented correctly , it will result in two keys producing the same hashCode() and equals() output. The problem will arise that HashMap will treat both output same instead of different and overwrite the most recent key-value pair with the previous key-value pair.

Similarly all the collection classes that does not allow the duplicate values use hashCode() and equals() method to find the duplicate elements.So it is very important to implement them correctly.

Contract of hashCode() and equals() method

a. If object1.equals(object2) , then object1.hashCode() == object2.hashCode() should always be true.

b. If object1.hashCode() == object2.hashCode() is true does not guarantee object1.equals(object2)

**Q43 What is EnumSet in Java ?**

EnumSet is a specialized Set implementation for use with enum types. All of the elements in an enum set must come from a single enum type that is specified explicitly or implicitly , when the set is created.

The iterator never throws ConcurrentModificationException and is weakly consistent.

Advantage over HashSet:

All basic operations of EnumSet execute in constant time . It is most likely to be much faster than HashSet counterparts.

It is a part of Java Collections Framework since jdk 1.5.

**Q44 What are concurrentCollectionClasses?**

In jdk1.5 , Java Api developers had introduced new package called java.util.concurrent that have thread-safe collection classes as they allow collections to be modified while iterating . The iterator is fail-fast that is it will throw ConcurrentModificationException.

Some examples of concurrentCollectionClasses are :

a. CopyOnWriteArrayList

b. ConcurrentHashMap

**Q45 How do you convert a given Collection to SynchronizedCollection ?**

One line code : Collections.synchronizedCollection(Collection collectionObj) will convert a given collection to synchronized collection.

**Q46 What is IdentityHashMap ?**

IdentityHashMap

IdentityHashMap is a class present in java.util package. It implements the Map interface with a hash table , using reference equality instead of object equality when comparing keys and values.In other words , in IdentityHashMap two keys k1 and k2 are considered equal if only if (k1==k2).IdentityHashMap is not synchronized. Iterators returned by the iterator() method are fail-fast , hence , will throw ConcurrentModificationException.

**Q47 What is WeakHashMap ?**

WeakHashMap :

WeakHashMap is a class present in java.util package similar to IdentityHashMap. It is a Hashtable based implementation of Map interface with weak keys. An entry in WeakHashMap will automatically be removed when its key is no longer in ordinary use. More precisely the presence of a mapping for a given key will not prevent the key from being discarded by the garbage collector.

It permits null keys and null values.

Like most collection classes this class is not synchronized.A synchronized WeakHashMap may be constructed using the Collections.synchronizedMap() method.

Iterators returned by the iterator() method are fail-fast , hence , will throw ConcurrentModificationException.

**Q48 How will you make Collections readOnly ?**

We can make the Collection readOnly by using the following lines code:

General : Collections.unmodifiableCollection(Collection c)

Collections.unmodifiableMap(Map m)

Collections.unmodifiableList(List l)

Collections.unmodifiableSet(Set s)

**Q49 What is UnsupportedOperationException?**

This exception is thrown to indicate that the requested operation is not supported.

Example of UnsupportedOperationException:

In other words, if you call add() or remove() method on the readOnly collection . We know readOnly collection can not be modified . Hence , UnsupportedOperationException will be thrown.

**Q50 Suppose there is an Employee class. We add Employee class objects to the ArrayList.** Mention the steps need to be taken , if I want to sort the objects in ArrayList using the employeeId attribute present in Employee class.

a. Implement the Comparable interface for the Employee class and now to compare the objects by employeeId we will override the emp1.compareTo(emp2)

b. We will now call Collections class sort method and pass the list as argument , that is ,

Collections.sort(empList)

If you want to add more java collections interview questions and answers or in case you have any doubts related to the Java Collections framework , then please mention in the comments.

**STRING**

**1) Is String a keyword in java?**

No. String is not a keyword in java. String is a final class in java.lang package which is used to represent the set of characters in java**.**

**2) Is String a primitive type or derived type?**

String is a derived type.

**3) In how many ways you can create string objects in java?**

There are two ways to create string objects in java. One is using *new* operator and another one is using string *literals*. The objects created using new operator are stored in the heap memory and objects created using string literals are stored in string constant pool.[?](http://javaconceptoftheday.com/java-string-interview-questions-and-answers/)

|  |  |
| --- | --- |
| 1  2  3 | String s1 = new String("abc");          //Creating string object using new operator    String s2 = "abc";        //Creating string object using string literal |

**4) What is string constant pool?**

String objects are most used data objects in Java. Hence, java has a special arrangement to store the string objects. String Constant Pool is one such arrangement. String Constant Pool is the memory space in heap memory specially allocated to store the string objects created using string literals. In String Constant Pool, there will be no two string objects having the same content.

Whenever you create a string object using string literal, JVM first checks the content of the object to be created. If there exist an object in the string constant pool with the same content, then it returns the reference of that object. It doesn’t create a new object. If the content is different from the existing objects then only it creates new object.

**5) What is special about string objects as compared to objects of other derived types?**

One special thing about string objects is that you can create string objects without using new operator i.e using string literals. This is not possible with other derived types (except wrapper classes). One more special thing about strings is that you can concatenate two string objects using ‘+’. This is the relaxation java gives to string objects as they will be used most of the time while coding. And also java provides string constant pool to store the string objects.

**6) What do you mean by mutable and immutable objects?**

Immutable objects are like constants. You can’t modify them once they are created. They are final in nature. Whereas mutable objects are concerned, you can perform modifications to them.

**7) Which is the final class in these three classes – String, StringBuffer and StringBuilder?**

All three are final. (Interviewer will ask this type of questions to confuse you)

**8) What is the difference between String, StringBuffer and StringBuilder?**

String objects created using **java.lang.String** class are immutable. Once they are created, they can not be modified. If you try to modify them, a new string object will be created with modified content. This property of String class may cause some memory issues for applications which need frequent modification of string objects. To overcome this behavior of String class, two more classes are introduced in Java to represent the strings. They are **StringBuffer** and **StringBuilder**. Both these classes are also members of  **java.lang**package same as String class.

Immutability :

This is main reason why StringBuffer and StringBuilder are introduced. As objects of String class are immutable, objects of StringBuffer and StringBuilder class are **mutable**. You can change the contents of StringBuffer and StringBuider objects at any time of execution. When you change the content, new objects are not created. Instead of that the changes are applied to existing object. Thus solving memory issues may caused by String class.

Object Creation :

You have to use ‘**new**‘ operator to create objects to StringBuffer and StringBuilder classes. You can’t use string literals to create objects to these classes. For example, you can’t write **StringBuffer sb = “JAVA”** or**StringBuilder sb = “JAVA”**. It gives compile time error. But, you can use both string literals and new operator to create objects to String class.

Storage Area :

As objects of StringBuffer and StringBuilder are created using only new operator, they are stored in **heap memory**. Where as objects of String class are created using both string literals and new operator, they are stored in string constant pool as well as heap memory.

Thread Safety :

Any immutable object in java is thread safety. Because they are unchangeable once they are created. Any type of thread can’t change the content of immutable object. This applies to objects of String class also. Of the StringBuffer and StringBuilder objects, only StringBuffer objects are thread safety. All necessary methods in StringBuffer class are synchronized so that only one thread can enter into it’s object at any point of time. Where as StringBuilder objects are not thread safety.

Performance :

Because of thread safety property of String and StringBuffer classes, they reduces the performance of multithreaded applications. Because, multiple threads can’t enter into objects of these classes simultaneously. One thread has to wait until another thread is finished with them. But, you will not find performance problems if you use StringBuilder class. Becuase, multiple threads can enter into objects of this class. But, be aware that StringBuilder is not thread safety.

## equals() and hashCode() Methods :

In StringBuffer and StringBuilder classes, equals() and hashCode methods are not overrided. Where as in String class they are overrided.

## toString() Method :

toString() method is overrided in all three classes. You can also convert StringBuffer and StringBuilder objects to String type by calling toString() method on them.

**9) Why StringBuffer and StringBuilder classes are introduced in java when there already exist String class to represent the set of characters?**

The objects of String class are immutable in nature. i.e you can’t modify them once they are created. If you try to modify them, a new object will be created with modified content. This may cause memory and performance issues if you are performing lots of string modifications in your code. To overcome these issues, StingBuffer and StringBuilder classes are introduced in java.

**12) Which one will you prefer among “==” and equals() method to compare two string objects?**

I prefer equals() method because it compares two string objects based on their content. That provides more logical comparison of two string objects. If you use “==” operator, it checks only references of two objects are equal or not. It may not be suitable in all situations. So, rather stick to equals() method to compare two string objects

**“==” operator**, **equals() method** and **hashcode() method**s are used to check the equality of any type of objects in Java. In this article, we will discuss which is the better way to check the equality of two string objects.

**“==” operator** compares the two objects on their physical address. That means if two references are pointing to same object in the memory, then comparing those two references using “==” operator will return true. For example, if s1 and s2 are two references pointing to same object in the memory, then invoking **s1 == s2** will return true. This type of comparison is called **“Shallow Comparison”**.

**equals() method**, if not overrided, will perform same comparison as “==” operator does i.e comparing the objects on their physical address. So, it is always recommended that you should override equals() method in your class so that it provides field by field comparison of two objects. This type of comparison is called **“Deep Comparison”**.

In java.lang.String class, equals() method is overrided to provide the comparison of two string objects based on their contents. That means, any two string objects having same content will be equal according to equals() method. For example, if s1 and s2 are two string objects having the same content, then invoking **s1.equals(s2)**will return true.

**hashCode() method** returns hash code value of an object in the Integer form. It is recommended that whenever you override equals() method, you should also override hashCode() method so that two equal objects according to equals() method must return same hash code values. This is the general contract between equals() and hashCode() methods that must be maintained all the time.

In java.lang.String class, hashCode() method is also overrided so that two equal string objects according to equals() method will return same hash code values. That means, if s1 and s2 are two equal string objects according to equals() method, then invoking **s1.hashCode() == s2.hashCode()** will return true.

Let’s apply these three methods on string objects and try to analyse their output.

Define two string objects like below,

[?](http://javaconceptoftheday.com/when-to-use-equals-hashcode-on-strings/)

|  |  |
| --- | --- |
| 1  2  3 | String s1 = "JAVA";    String s2 = "JAVA"; |

Now apply above methods on these two objects.

**s1 == s2** —> will return true as both are pointing to same object in the constant pool.  
**s1.equals(s2)** —> will also return true as both are referring to same object.  
**s1.hashCode() == s2.hashCode()** —> It also returns true.

This type of comparison is straight forward. There is no speculation about this comparison. Let’s define the string objects like below,

[?](http://javaconceptoftheday.com/when-to-use-equals-hashcode-on-strings/)

|  |  |
| --- | --- |
| 1  2  3 | String s1 = new String("JAVA");    String s2 = new String("JAVA"); |

**s1 == s2** —> will return false because s1 and s2 are referring to two different objects in the memory.  
**s1.equals(s2)** —> will return true as both the objects have same content.  
**s1.hashCode() == s2.hashCode()** —> It will also return true because two equals string objects according to equals() method will have same hash code values.

Comparing the string objects defined like below will also give same result as the above.

[?](http://javaconceptoftheday.com/when-to-use-equals-hashcode-on-strings/)

|  |  |
| --- | --- |
| 1  2  3 | String s1 = "JAVA";    String s2 = new String("JAVA"); |

**s1 == s2** —> will return false because s1 and s2 are referring to two different objects in the memory.  
**s1.equals(s2)** —> will return true as both the objects have same content.  
**s1.hashCode() == s2.hashCode()** —> It will also return true.

Now, you may conclude that If there is a requirement of comparing two string objects on their physical address, then use “==” operator and if there is a requirement of comparing two string objects on their contents, then use equals() method or hashCode() method.

Hold on…. Before jumping onto conclusion, compare these two string objects.

[?](http://javaconceptoftheday.com/when-to-use-equals-hashcode-on-strings/)

|  |  |
| --- | --- |
| 1  2  3 | String s1 = "0-42L";    String s2 = "0-43-"; |

**s1 == s2** —> will return false as s1 and s2 are referring to two different objects in the memory. **(Expected…)**  
**s1.equals(s2)** —> It will also return false as both the objects have different content. **(Expected…)**  
**s1.hashCode() == s2.hashCode()** —> It will return true. **(???….)**

This is because, **two unequal string objects according to equals() method may have same hash code values**. Therefore, it is recommended not to use hashCode() method to compare two string objects. You may not get expected result.

## Conclusion :

When you want to check the equality of two string objects on their physical existence in the memory, then use “==” operator. If you want to check the equality of two string objects depending upon their contents, then use equals() method. It is recommended not to use hashCode() method to check the equality of two string objects. You may get unexpected result.

**16) Where exactly string constant pool is located in the memory?**

Inside the heap memory. JVM reserves some part of the heap memory to store string objects created using string literals

In Java, strings are special. Java gives some special attention to string types that no other types enjoy such attention. For example, to create the string objects you need not to use ‘**new**‘ keyword. Where as to create other type of objects you have to use ‘new’ keyword. Like this, strings enjoy some special attention by the java. This attention is worth the while, because the strings are used almost everywhere while developing any kind of applications.

While storing the string objects in the memory also, they are specially treated by the Java. After reading this article, you will come to know how they are specially treated in the memory.

We all know that JVM divides the allocated memory to a Java program into two parts. one is **Stack** and another one is **heap**. Stack is used for execution purpose and heap is used for storage purpose. In that heap memory, JVM allocates some memory specially meant for string literals. This part of the heap memory is called **String Constant Pool**.

Whenever you create a string object using string literal, that object is stored in the **string constant pool** and whenever you create a string object using new keyword, such object is stored in the heap memory.

For example, when you create string objects like below, they will be stored in the String Constant Pool.

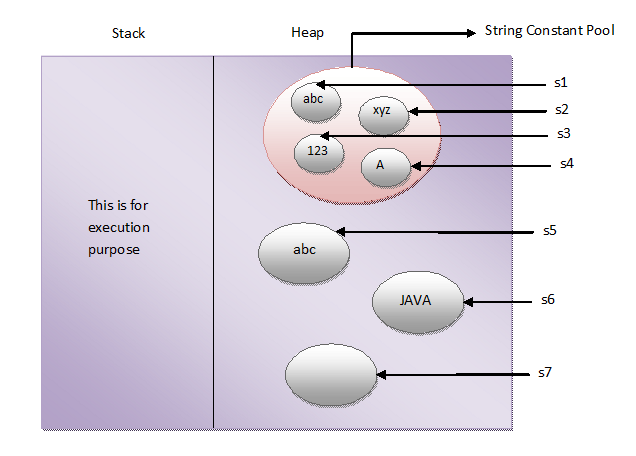
[?](http://javaconceptoftheday.com/how-the-strings-are-stored-in-the-memory/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | String s1 = "abc";    String s2 = "xyz";    String s3 = "123";    String s4 = "A"; |

And when you create string objects using new keyword like below, they will be stored in the heap memory.

[?](http://javaconceptoftheday.com/how-the-strings-are-stored-in-the-memory/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | String s5 = new String("abc");    char[] c = {'J', 'A', 'V', 'A'};    String s6 = new String(c);    String s7 = new String(new StringBuffer()); |

This is how String Constant Pool looks like in the memory.  


One more interesting thing about String Constant Pool is that, **pool space is allocated to an object depending upon it’s content**. There will be no two objects in the pool having the same content.

This is what happens when you create string objects using string literal,

**“When you create a string object using string literal, JVM first checks the content of to be created object. If there exist an object in the pool with the same content, then it returns the reference of that object. It doesn’t create new object. If the content is different from the existing objects then only it creates new object.”**

But, when you create string objects using new keyword, a new object is created whether the content is same or not.

This can be proved by using “==” operator. As “==” operator returns true if two objects have same physical address in the memory otherwise it will return false. In the below example, s1 and s2 are created using string literal “abc”. So, s1 == s2 returns true. Where as s3 and s4 are created using new operator having the same content. But, s3 == s4 returns false.

[?](http://javaconceptoftheday.com/how-the-strings-are-stored-in-the-memory/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | public class StringExamples  {      public static void main(String[] args)      {          //Creating string objects using literals            String s1 = "abc";            String s2 = "abc";            System.out.println(s1 == s2);        //Output : true            //Creating string objects using new operator            String s3 = new String("abc");            String s4 = new String("abc");            System.out.println(s3 == s4);        //Output : false      }  } |

**In simple words, there can not be two string objects with same content in the string constant pool. But, there can be two string objects with the same content in the heap memory.**

**18) What is string intern?**

String object in the string constant pool is called as String Intern. You can create an exact copy of heap memory string object in string constant pool. This process of creating an exact copy of heap memory string object in the string constant pool is called interning. intern() method is used for interning.

String objects in java are stored in two places in memory. One is **String Constant Pool** and another one is**Heap Memory**. String objects created using string literals are stored in String Constant Pool where as string objects created using new operator are stored in heap memory.

## Why You Need String Constant Pool? :

String objects are most used objects in the development of any kind of applications. Therefore, there has to be a special arrangement to store these objects. String Constant Pool is one such special arrangement. In string constant pool, there will be no two objects with the same content. Heap memory can have any number of objects with same content.

Just imagine creating 1000 string objects with same content in heap memory and one string object with that content in String Constant Pool. Which one saves the memory?. which one will save the time?. Which one will be accessed faster?. It is, of course, String Constant Pool. That’s why you need String Constant Pool.

## What Is String Intern? :

**String intern** or simply **intern** refers to string object in the String Constant Pool. **Interning** is the process of creating a string object in String Constant Pool which will be exact copy of string object in heap memory.

## intern() Method :

**intern()**methodof java.lang.String class is used to perform interning i.e creating an exact copy of heap string object in string constant pool. When you call this method on a string object, first it checks whether there exist an object with the same content in the String Constant Pool. If object does not exist in the pool, it will create an object with the same content in the string constant pool and returns the reference of that object. If object exist in the pool than it returns reference of that object without creating a new object.

Look at the below example. Object ‘s1’ will be created in heap memory as we are using new operator to create it. When we call intern() method on s1, it creates a new string object in the string constant pool with “JAVA” as it’s content and assigns it’s reference to s2. So, **s1 == s2** will return false because they are two different objects in the memory and s1.equals(s2) will return true because they have same content.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | public class StringExamples  {      public static void main(String[] args)      {          String s1 = new String("JAVA");            String s2 = s1.intern();       //Creating String Intern            System.out.println(s1 == s2);       //Output : false            System.out.println(s1.equals(s2));    //Output : true      }  } |

Look at this example. Object s1 will be created in string constant pool as we are using string literal to create it and object s2 will be created in heap memory as we are using new operator to create it. When you call intern() method on s2, it returns reference of object to which s1 is pointing as it’s content is same as s2. It does not create a new object in the pool. So, **S1 == s3** will return true as both are pointing to same object in the pool.

[?](http://javaconceptoftheday.com/string-intern-java/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | public class StringExamples  {      public static void main(String[] args)      {          String s1 = "JAVA";            String s2 = new String("JAVA");            String s3 = s2.intern();       //Creating String Intern            System.out.println(s1 == s3);       //Output : true      }  } |

## String Literals Are Automatically Interned :

When you call intern() on the string object created using string literals it returns reference of itself. Because, you can’t have two string objects in the pool with same content. That means string literals are automatically interned in java.

[?](http://javaconceptoftheday.com/string-intern-java/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public class StringExamples  {      public static void main(String[] args)      {          String s1 = "JAVA";            String s2 = s1.intern();       //Creating String Intern            System.out.println(s1 == s2);       //Output : true      }  } |

## What is the use of interning the string?

**To Save The memory Space :**

Using interned string, you can save the memory space. If you are using lots of string objects with same content in your code, than it is better to create an intern of that string in the pool. Use that intern string whenever you need it instead of creating a new object in the heap. It saves the memory space.

**For Faster Comparison :**

Assume that there are two string objects s1 and s2 in heap memory and you need to perform comparison of these two objects more often in your code. Then using s1.intern() == s2.intern() will be more fast then s1.equals(s2). Because, equals() method performs character by character comparison where as “==” operator just compares references of objects.

**EXCEPTION**

### What is Exception in Java?

Exception is an error event that can happen during the execution of a program and disrupts it’s normal flow. Exception can arise from different kind of situations such as wrong data entered by user, hardware failure, network connection failure etc.

Whenever any error occurs while executing a java statement, an exception object is created and then[**JRE**](http://www.journaldev.com/546/difference-between-jdk-jre-and-jvm-in-java) tries to find exception handler to handle the exception. If suitable exception handler is found then the exception object is passed to the handler code to process the exception, known as **catching the exception**. If no handler is found then application throws the exception to runtime environment and JRE terminates the program.

**Java Exception handling** framework is used to handle runtime errors only, compile time errors are not handled by exception handling framework.

### What are the Exception Handling Keywords in Java?

There are four keywords used in java exception handling.

1. **throw**: Sometimes we explicitly want to create exception object and then throw it to halt the normal processing of the program. **throw** keyword is used to throw exception to the runtime to handle it.
2. **throws**: When we are throwing any checked exception in a method and not handling it, then we need to use throws keyword in method signature to let caller program know the exceptions that might be thrown by the method. The caller method might handle these exceptions or propagate it to it’s caller method using throws keyword. We can provide multiple exceptions in the throws clause and it can be used with **main()** method also.
3. **try-catch**: We use try-catch block for exception handling in our code. try is the start of the block and catch is at the end of try block to handle the exceptions. We can have multiple catch blocks with a try and try-catch block can be nested also. catch block requires a parameter that should be of type Exception.
4. **finally**: finally block is optional and can be used only with try-catch block. Since exception halts the process of execution, we might have some resources open that will not get closed, so we can use finally block. finally block gets executed always, whether exception occurrs or not.