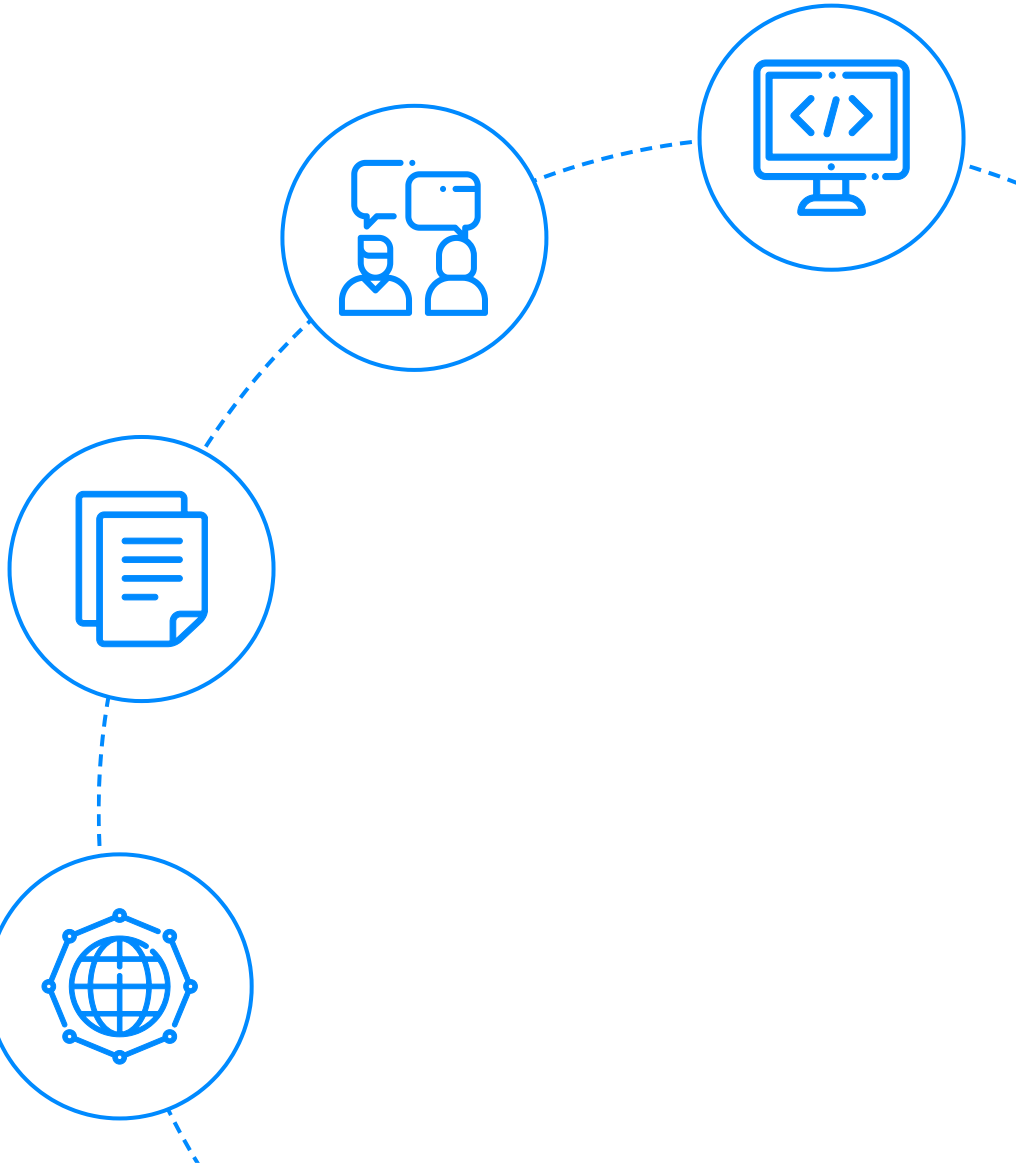
InterviewBit

Multithreading Interview Questions



To view the live version of the page, click here.

Contents

Multithreading Interview Questions in Java for Freshers

Multithreading Interview Questions in Java for Experienced

21. What is the synchronization process? Why use it?
22. What is synchronized method and synchronized block? Which one should be preferred?
23. What is thread starvation?
24. What is Livelock? What happens when it occurs?
25. What is BlockingQueue?
26. Can you start a thread twice?
27. Explain context switching.
28. What is CyclicBarrier and CountDownLatch?
29. What do you mean by inter-thread communication?
30. What is Thread Scheduler and Time Slicing?
31. What is a shutdown hook?
32. What is busy spinning?
33. What is ConcurrentHashMap and Hashtable? In java, why is ConcurrentHashMap considered faster than Hashtable?
34. Explain thread priority.
35. What do you mean by the ThreadLocal variable in Java?
36. What is semaphore?
37. Explain Thread Group. Why should we not use it?
38. What is the ExecutorService interface?
39. What will happen if we don't override the thread class run() method?
40. What is the lock interface? Why is it better to use a lock interface rather than a synchronized block.?

Multithreading Interview Questions in Java for Experienced

41. Is it possible to call the run() method directly to start a new thread?
42. Is it possible that each thread can have its stack in multithreaded programming?

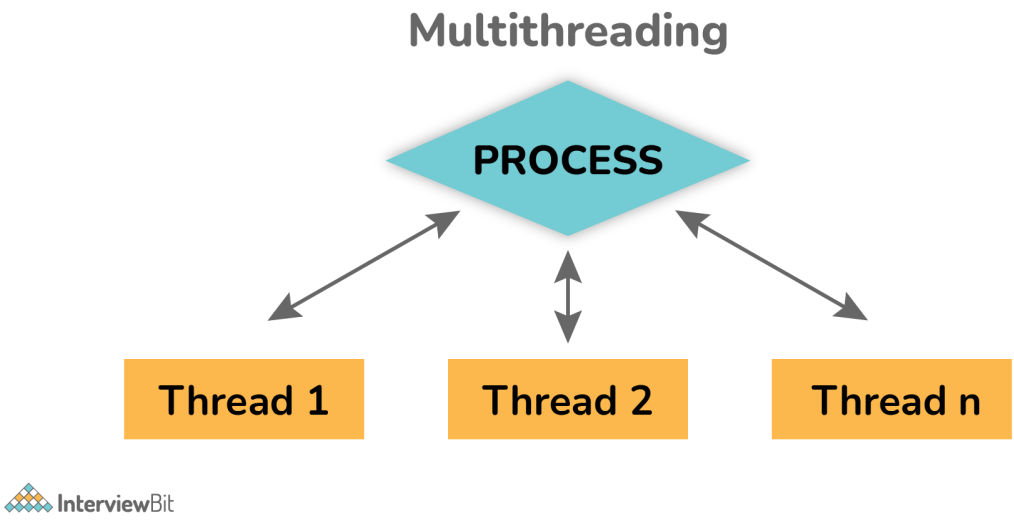
Conclusion

43. Conclusion

Let's get Started

What do you mean by Multithreading? Why is it important?

Multithreading means multiple threads and is considered one of the most important features of Java. As the name suggests, it is the ability of a CPU to execute multiple threads independently at the same time but sharing the process resources simultaneously. Its main purpose is to provide simultaneous execution of multiple threads to utilize the CPU time as much as possible. It is a Java feature where one can subdivide the specific program into two or more threads to make the execution of the program fast and easy.



Multithreading Interview Questions in Java for Freshers

1. What are the benefits of using Multithreading?

There are various benefits of multithreading as given below:

- Allow the program to run continuously even if a part of it is blocked.
- Improve performance as compared to traditional parallel programs that use multiple processes.
- Allows to write effective programs that utilize maximum CPU time
- Improves the responsiveness of complex applications or programs.
- Increase use of CPU resources and reduce costs of maintenance.
- Saves time and parallelism tasks.
- If an exception occurs in a single thread, it will not affect other threads as threads are independent.
- Less resource-intensive than executing multiple processes at the same time.

2. What is Thread in Java?

Threads are basically the lightweight and smallest unit of processing that can be managed independently by a scheduler. Threads are referred to as parts of a process that simply let a program execute efficiently with other parts or threads of the process at the same time. Using threads, one can perform complicated tasks in the easiest way. It is considered the simplest way to take advantage of multiple CPUs available in a machine. They share the common address space and are independent of each other.

3. What are the two ways of implementing thread in Java?

There are basically two ways of implementing thread in java as given below:

- Implementing **Runnable** interface in Java

Example:

```
class MultithreadingDemo extends Thread
{
  public void run()
 {
     System.out.println("My thread is in running state.");
 }
  public static void main(String args[])
 {
    MultithreadingDemoobj=new MultithreadingDemo();
        obj.start();
  }
}
```

Output:

```
My thread is in running state.
```

- Extending the **Thread** class.

Example:

```
class MultithreadingDemo implements Runnable
{
   public void run()
 {
      System.out.println("My thread is in running state.");
  }
    public static void main(String args[])
 {
      MultithreadingDemo obj=new MultithreadingDemo();
      Threadtobj =new Thread(obj);       tobj.start();
 }
}
```

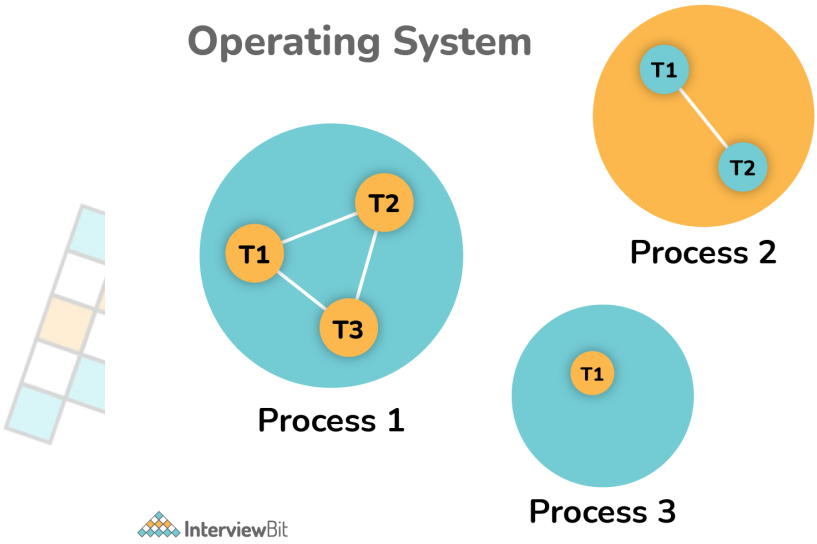
Output:

```
My thread is in running state.
```

4. What's the difference between thread and process?

Thread: It simply refers to the smallest units of the particular process. It has the ability to execute different parts (referred to as thread) of the program at the same time.

Process: It simply refers to a program that is in execution i.e., an active program. A process can be handled using PCB (Process Control Block).



Thread	Process
It is a subset of a subunit of a process.	It is a program in execution containing multiple threads.
In this, inter-thread communication is faster, less expensive, easy and efficient because threads share the same memory address of the process they belong to.	In this, inter-process communication is slower, expensive, and complex because each process has different memory space or address.,
These are easier to create, lightweight, and have less overhead.	These are difficult to create, heavyweight, and have more overhead.
It requires less time for creation, termination, and context switching.	It requires more time for creation, termination, and context switching.
Processes with multiple threads use fewer resources.	Processes without threads use more resources.
Threads are parts of a process, so they are dependent on each other but each thread executes independently.	Processes are independent of each other.
There is a need for synchronization in threads to avoid unexpected scenarios or problems.	There is no need for synchronization in each process.
They share data and information with each other.	They do not share data with each other.

5. What's the difference between class lock and object lock?

Class Lock: In java, each and every class has a unique lock usually referred to as a class level lock. These locks are achieved using the keyword 'static synchronized' and can be used to make static data thread-safe. It is generally used when one wants to prevent multiple threads from entering a synchronized block.

Example:

```
public class ClassLevelLockExample
{
  public void classLevelLockMethod()
 {
      synchronized (ClassLevelLockExample.class)
        {
             //DO your stuff here
        }
 }
}
```

Object Lock: In java, each and every object has a unique lock usually referred to as an object-level lock. These locks are achieved using the keyword 'synchronized' and can be used to protect non-static data. It is generally used when one wants to synchronize a non-static method or block so that only the thread will be able to execute the code block on a given instance of the class.

Example:

```
public class ObjectLevelLockExample
{
  public void objectLevelLockMethod()
 {
      synchronized (this)
        {
             //DO your stuff here
        }
 }
}
```

6. What's the difference between User thread and Daemon thread?

User and Daemon are basically two types of thread used in Java by using a 'Thread Class'.

User Thread (Non-Daemon Thread): In Java, user threads have a specific life cycle and its life is independent of any other thread. JVM (Java Virtual Machine) waits for any of the user threads to complete its tasks before terminating it. When user threads are finished, JVM terminates the whole program along with associated daemon threads.

Daemon Thread: In Java, daemon threads are basically referred to as a service provider that provides services and support to user threads. There are basically two methods available in thread class for daemon thread: setDaemon() and isDaemon().

User Thread vs Daemon Thread

User Thread	Daemon Thread
JVM waits for user threads to finish their tasks before termination.	JVM does not wait for daemon threads to finish their tasks before termination.
These threads are normally created by the user for executing tasks concurrently.	These threads are normally created by JVM.
They are used for critical tasks or core work of an application.	They are not used for any critical tasks but to do some supporting tasks.
These threads are referred to as high-priority tasks, therefore are required for running in the foreground.	These threads are referred to as low priority threads, therefore are especially required for supporting background tasks like garbage collection, releasing memory of unused objects, etc.

7. How can we create daemon threads?

We can create daemon threads in java using the thread class **setDaemon(true)**. It is used to mark the current thread as daemon thread or user thread. **isDaemon()** method is generally used to check whether the current thread is daemon or not. If the thread is a daemon, it will return true otherwise it returns false.

Example:

Program to illustrate the use of setDaemon() and isDaemon() method.

```
public class DaemonThread extends Thread
{
   public DaemonThread(String name){
       super(name);
   }
   public void run()
   {
       // Checking whether the thread is Daemon or not
       if(Thread.currentThread().isDaemon())
       {
           System.out.println(getName() + " is Daemon thread");
       }
       else
       {
           System.out.println(getName() + " is User thread");
       }
   }
   public static void main(String[] args)
   {
       DaemonThread t1 = new DaemonThread("t1");
       DaemonThread t2 = new DaemonThread("t2");
       DaemonThread t3 = new DaemonThread("t3");
       // Setting user thread t1 to Daemon
       t1.setDaemon(true);
       // starting first 2 threads
       t1.start();
       t2.start();
       // Setting user thread t3 to Daemon
       t3.setDaemon(true);
       t3.start();
   }
}
```

Output:

```
t1 is Daemon thread
t3 is Daemon thread
t2 is User thread
```

But one can only call the **setDaemon()** method before start() method otherwise it will definitely throw IllegalThreadStateException as shown below:

```
public class DaemonThread extends Thread
{
   public void run()
   {
       System.out.println("Thread name: " + Thread.currentThread().getName());
       System.out.println("Check if its DaemonThread: "
                       + Thread.currentThread().isDaemon());
   }
   public static void main(String[] args)
   {
       DaemonThread t1 = new DaemonThread();
       DaemonThread t2 = new DaemonThread();
       t1.start();
       // Exception as the thread is already started
       t1.setDaemon(true);
       t2.start();
   }
}
```

Output:

```
Thread name: Thread-0
Check if its DaemonThread: false
```

8. What are the wait() and sleep() methods?

wait(): As the name suggests, it is a non-static method that causes the current thread to wait and go to sleep until some other threads call the notify () or notifyAll() method for the object's monitor (lock). It simply releases the lock and is mostly used for inter-thread communication. It is defined in the object class, and should only be called from a synchronized context.

Example:

```
synchronized(monitor)
{
monitor.wait();       Here Lock Is Released by Current Thread
}
```

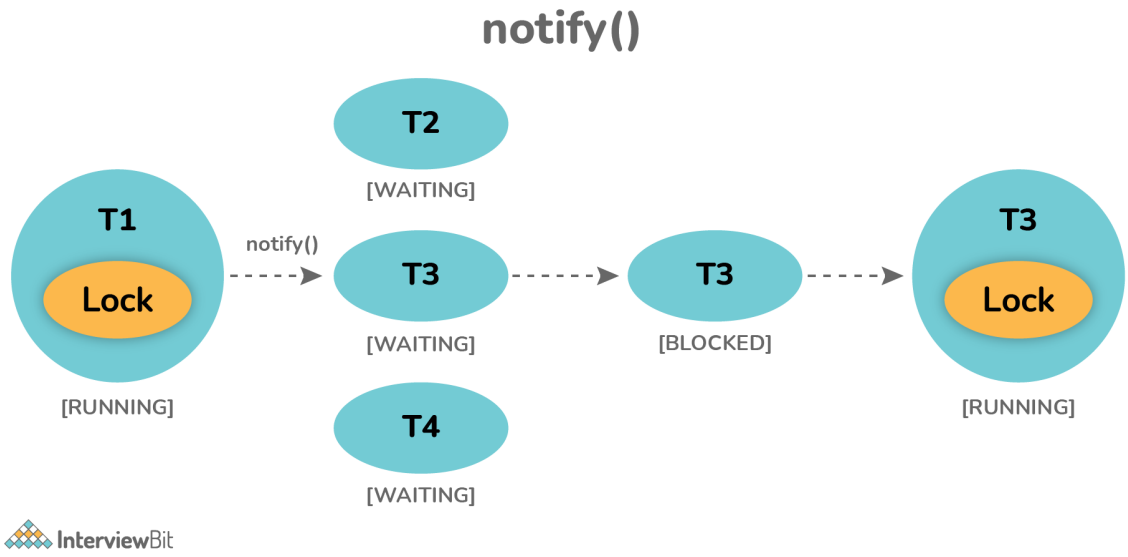
sleep(): As the name suggests, it is a static method that pauses or stops the execution of the current thread for some specified period. It doesn't release the lock while waiting and is mostly used to introduce pause on execution. It is defined in thread class, and no need to call from a synchronized context.

Example:

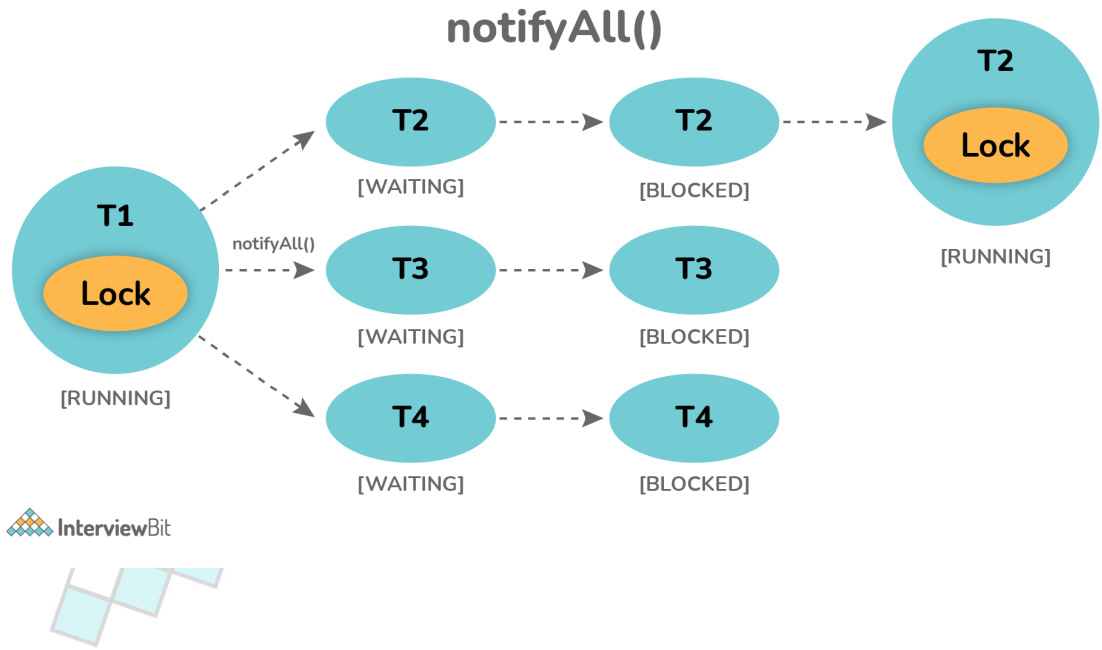
```
synchronized(monitor)
{
Thread.sleep(1000);     Here Lock Is Held by The Current Thread
//after 1000 milliseconds, the current thread will wake up, or after we call that is ir
}
```

9. What's the difference between notify() and notifyAll()?

notify(): It sends a notification and wakes up only a single thread instead of multiple threads that are waiting on the object's monitor.



notifyAll(): It sends notifications and wakes up all threads and allows them to compete for the object's monitor instead of a single thread.



10. Why wait(), notify(), and notifyAll() methods are present in Object class?

We know that every object has a monitor that allows the thread to hold a lock on the object. But the thread class doesn't contain any monitors. Thread usually waits for the object's monitor (lock) by calling the wait() method on an object, and notify other threads that are waiting for the same lock using notify() or notifyAll() method. Therefore, these three methods are called on objects only and allow all threads to communicate with each that are created on that object.

11. What is Runnable and Callable Interface? Write the difference between them.

Both the interfaces are generally used to encapsulate tasks that are needed to be executed by another thread. But there are some differences between them as given below:

Running Interface: This interface is basically available in Java right from the beginning. It is simply used to execute code on a concurrent thread.
Callable Interface: This interface is basically a new one that was introduced as a part of the concurrency package. It addresses the limitation of runnable interfaces along with some major changes like generics, enum, static imports, variable argument method, etc. It uses generics to define the return type of object.

```
public interface Runnable
{
  public abstract void run();
}
public interface Callable<V>
{
V call() throws Exception;
}
```

Runnable Interface vs Callable Interface

Runnable Interface	Callable Interface
It does not return any result and therefore, cannot throw a checked exception.	It returns a result and therefore, can throw an exception.
It cannot be passed to invokeAll method.	It can be passed to invokeAll method.
It was introduced in JDK 1.0.	It was introduced in JDK 5.0, so one cannot use it before Java 5.
It simply belongs to Java.lang.	It simply belongs to java.util.concurrent.
It uses the run() method to define a task.	It uses the call() method to define a task.
To use this interface, one needs to override the run() method.	To use this interface, one needs to override the call() method.

12. What is the start() and run() method of Thread class?

start(): In simple words, the start() method is used to start or begin the execution of a newly created thread. When the start() method is called, a new thread is created and this newly created thread executes the task that is kept in the run() method. One can call the start() method only once.

run(): In simple words, the run() method is used to start or begin the execution of the same thread. When the run() method is called, no new thread is created as in the case of the start() method. This method is executed by the current thread. One can call the run() method multiple times.

13. Explain thread pool?

A Thread pool is simply a collection of pre-initialized or worker threads at the start-up that can be used to execute tasks and put back in the pool when completed. It is referred to as pool threads in which a group of fixed-size threads is created. By reducing the number of application threads and managing their lifecycle, one can mitigate the issue of performance using a thread pool. Using threads, performance can be enhanced and better system stability can occur. To create the thread pools, java.util.concurrent.Executors class usually provides factory methods.

14. What's the purpose of the join() method?

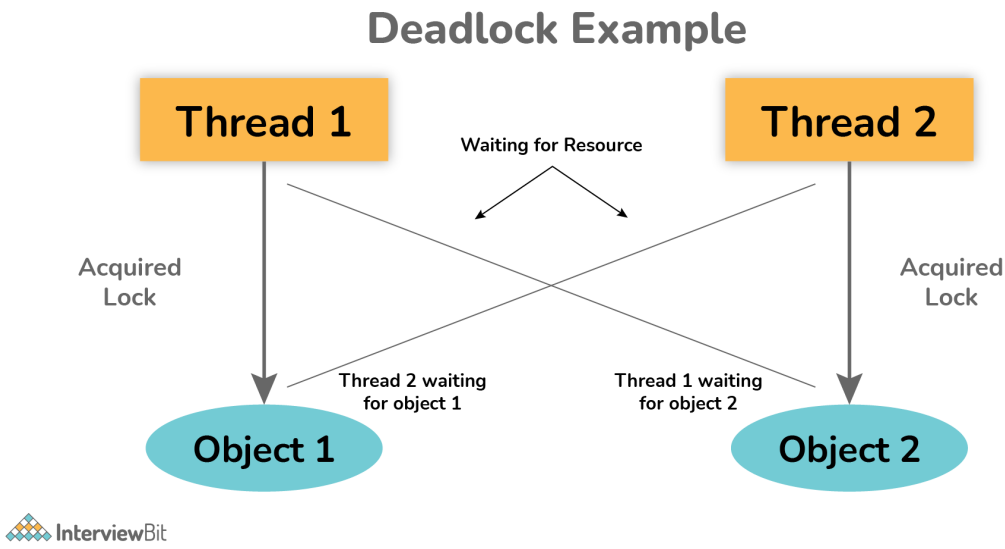
join() method is generally used to pause the execution of a current thread unless and until the specified thread on which join is called is dead or completed. To stop a thread from running until another thread gets ended, this method can be used. It joins the start of a thread execution to the end of another thread's execution. It is considered the final method of a thread class.

15. What do you mean by garbage collection?

Garbage collection is basically a process of managing memory automatically. It uses several GC algorithms among which the popular one includes Mark and Sweep. The process includes three phases i.e., marking, deletion, and compaction/copying. In simple words, a garbage collector finds objects that are no longer required by the program and then delete or remove these unused objects to free up the memory space.

16. Explain the meaning of the deadlock and when it can occur?

Deadlock, as the name suggests, is a situation where multiple threads are blocked forever. It generally occurs when multiple threads hold locks on different resources and are waiting for other resources to complete their task.



The above diagram shows a deadlock situation where two threads are blocked forever. Thread 1 is holding Object 1 but needs object 2 to complete processing whereas Thread 2 is holding Object 2 but needs object 1 first. In such conditions, both of them will hold lock forever and will never complete tasks.

17. Explain volatile variables in Java?

A volatile variable is basically a keyword that is used to ensure and address the visibility of changes to variables in multithreaded programming. This keyword cannot be used with classes and methods, instead can be used with variables. It is simply used to achieve thread-safety. If you mark any variable as volatile, then all the threads can read its value directly from the main memory rather than CPU cache, so that each thread can get an updated value of the variable.

18. How do threads communicate with each other?

Threads can communicate using three methods i.e., wait(), notify(), and notifyAll().

19. Can two threads execute two methods (static and non-static concurrently)?

Yes, it is possible. If both the threads acquire locks on different objects, then they can execute concurrently without any problem.

20. What is the purpose of the finalize() method?

Finalize() method is basically a method of Object class specially used to perform cleanup operations on unmanaged resources just before garbage collection. It is not at all intended to be called a normal method. After the complete execution of finalize() method, the object gets destroyed automatically.

Multithreading Interview Questions in Java for Experienced

21. What is the synchronization process? Why use it?

Synchronization is basically a process in java that enables a simple strategy for avoiding thread interference and memory consistency errors. This process makes sure that resource will be only used one thread at a time when one thread tries to access a shared resource. It can be achieved in three different ways as given below:

- By the synchronized method
- By synchronized block
- By static synchronization

Syntax:

```
synchronized (object)
{
   //statement to be synchronized
}
```

22. What is synchronized method and synchronized block? Which one should be preferred?

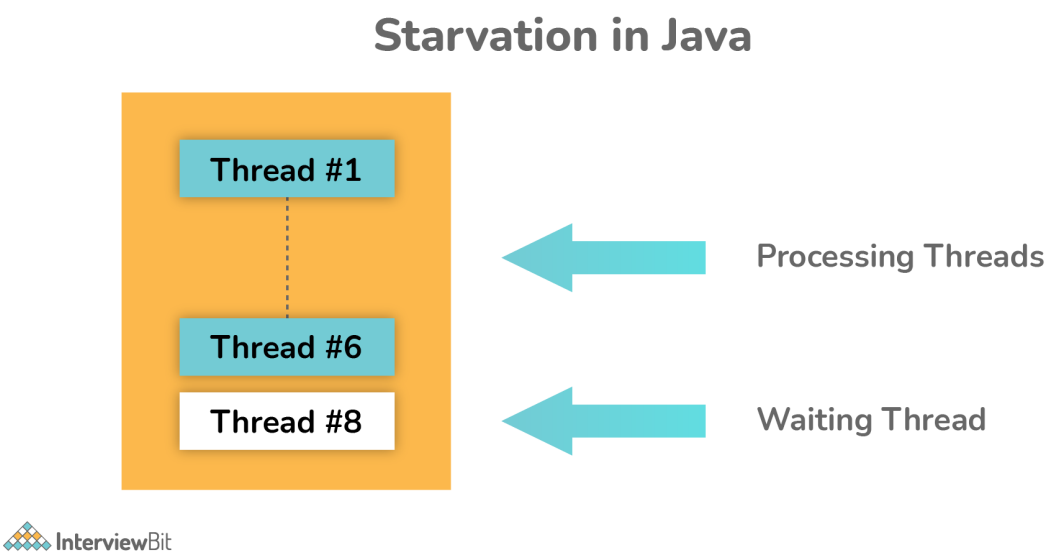
Synchronized Method: In this method, the thread acquires a lock on the object when they enter the synchronized method and releases the lock either normally or by throwing an exception when they leave the method. No other thread can use the whole method unless and until the current thread finishes its execution and release the lock. It can be used when one wants to lock on the entire functionality of a particular method.

Synchronized Block: In this method, the thread acquires a lock on the object between parentheses after the synchronized keyword, and releases the lock when they leave the block. No other thread can acquire a lock on the locked object unless and until the synchronized block exists. It can be used when one wants to keep other parts of the programs accessible to other threads.

Synchronized blocks should be preferred more as it boosts the performance of a particular program. It only locks a certain part of the program (critical section) rather than the entire method and therefore leads to less contention.

23. What is thread starvation?

Thread starvation is basically a situation or condition where a thread won't be able to have regular access to shared resources and therefore is unable to proceed or make progress. This is because other threads have high priority and occupy the resources for too long. This usually happens with low-priority threads that do not get CPU for its execution to carry on.

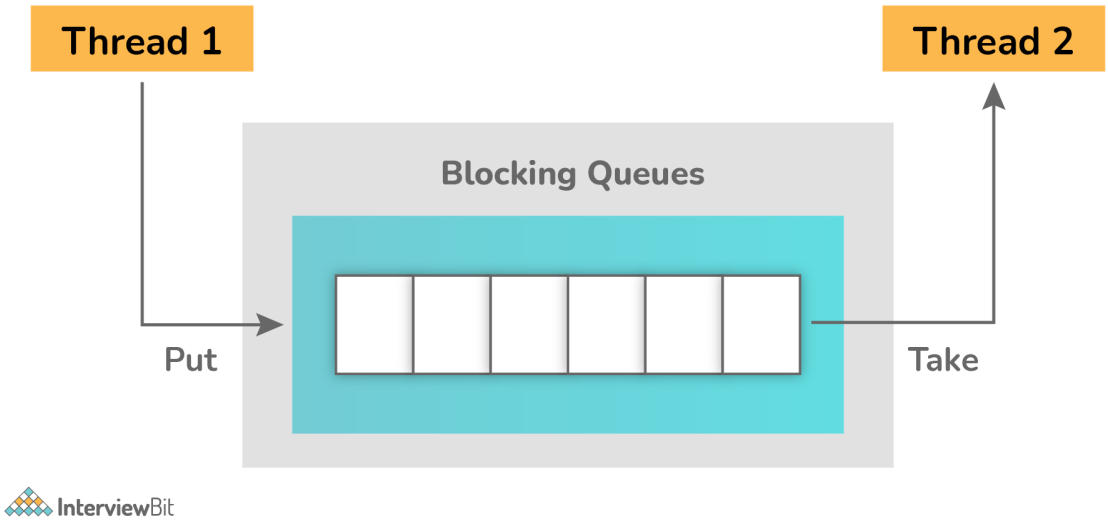


24. What is Livelock? What happens when it occurs?

Similar to deadlock, livelock is also another concurrency problem. In this case, the state of threads changes between one another without making any progress. Threads are not blocked but their execution is stopped due to the unavailability of resources.

25. What is BlockingQueue?

BlockingQueue basically represents a queue that is thread-safe. Producer thread inserts resource/element into the queue using put() method unless it gets full and consumer thread takes resources from the queue using take() method until it gets empty. But if a thread tries to dequeue from an empty queue, then a particular thread will be blocked until some other thread inserts an item into the queue, or if a thread tries to insert an item into a queue that is already full, then a particular thread will be blocked until some threads take away an item from the queue.



Example:

```
package org.arpit.java2blog;

import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;

public class BlockingQueuePCExample {

   public static void main(String[] args) {

       BlockingQueue<String> queue=new ArrayBlockingQueue<>(5);
       Producer producer=new Producer(queue);
       Consumer consumer=new Consumer(queue);
       Thread producerThread = new Thread(producer);
       Thread consumerThread = new Thread(consumer);

       producerThread.start();
       consumerThread.start();

   }

   static class Producer implements Runnable {

       BlockingQueue<String> queue=null;

       public Producer(BlockingQueue queue) {
           super();
           this.queue = queue;
       }

       @Override
       public void run() {

               try {
                   System.out.println("Producing element 1");
                   queue.put("Element 1");
                   Thread.sleep(1000);
                   System.out.println("Producing element 2");
                   queue.put("Element 2");
                   Thread.sleep(1000);
                   System.out.println("Producing element 3");
                   queue.put("Element 3");
               } catch (InterruptedException e) {

                   e.printStackTrace();
               }
       }
   }

   static class Consumer implements Runnable {

       BlockingQueue<String> queue=null;

       public Consumer(BlockingQueue queue) {
           super();
           this.queue = queue;
       }
```

Output:

```
Producing element 1
Consumed Element 1
Producing element 2
Consumed Element 2
Producing element 3
Consumed Element 3
```

26. Can you start a thread twice?

No, it's not at all possible to restart a thread once a thread gets started and completes its execution. Thread only runs once and if you try to run it for a second time, then it will throw a runtime exception i.e., java.lang.IllegalThreadStateException.

Example:

```
public class TestThreadTwice1 extends Thread{
public void run(){
System.out.println(" thread is executing now.......");
}
public static void main(String args[]){
TestThreadTwice1 t1=new TestThreadTwice1();
t1.start();
t1.start();
}
}
```

Output:

```
thread is executing now.......
Exception in thread "main" java.lang.IllegalThreadStateException
```

27. Explain context switching.

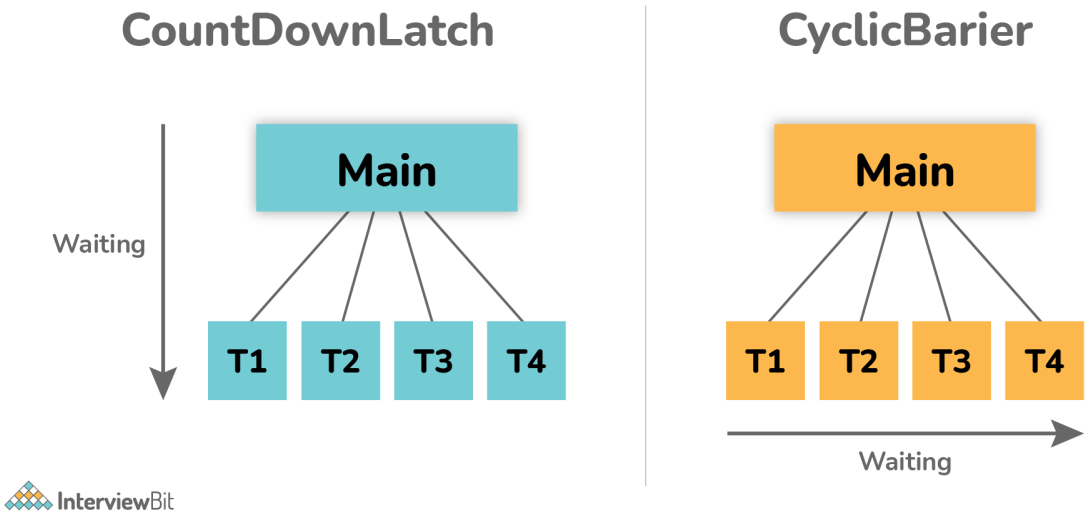
Context switching is basically an important feature of multithreading. It is referred to as switching of CPU from one thread or process to another one. It allows multiple processes to share the same CPU. In context switching, the state of thread or process is stored so that the execution of the thread can be resumed later if required.

28. What is CyclicBarrier and CountDownLatch?

CyclicBarrier and CountDownLatch, both are required for managing multithreaded programming. But there is some difference between them as given below:

CyclicBarrier: It is a tool to synchronize threads processing using some algorithm. It enables a set of threads to wait for each other till they reach a common execution point or common barrier points, and then let them further continue execution. One can reuse the same CyclicBarrier even if the barrier is broken by setting it.

CountDownLatch: It is a tool that enables main threads to wait until mandatory operations are performed and completed by other threads. In simple words, it makes sure that a thread waits until the execution in another thread completes before it starts its execution. One cannot reuse the same CountDownLatch once the count reaches 0.



29. What do you mean by inter-thread communication?

Inter-thread communication, as the name suggests, is a process or mechanism using which multiple threads can communicate with each other. It is especially used to avoid thread polling in java and can be obtained using wait(), notify(), and notifyAll() methods.

30. What is Thread Scheduler and Time Slicing?

Thread Scheduler: It is a component of JVM that is used to decide which thread will execute next if multiple threads are waiting to get the chance of execution. By looking at the priority assigned to each thread that is READY, the thread scheduler selects the next run to execute. To schedule the threads, it mainly uses two mechanisms: Preemptive Scheduling and Time slicing scheduling.

Time Slicing: It is especially used to divide CPU time and allocate them to active threads. In this, each thread will get a predefined slice of time to execute. When the time expires, a particular thread has to wait till other threads get their chances to use their time in a round-robin fashion. Every running thread will get executed for a fixed time period.

31. What is a shutdown hook?

A shutdown hook is simply a thread that is invoked implicitly before JVM shuts down. It is one of the most important features of JVM because it provides the capacity to do resource cleanup or save application state JVM shuts down. By calling the halt(int) method of the Runtime class, the shutdown hook can be stopped. Using the following method, one can add a shutdown hook.

```
public void addShutdownHook(Thread hook){}
Runtime r=Runtime.getRuntime();
r.addShutdownHook(new MyThread());
```

32. What is busy spinning?

Busy Spinning, also known as Busy-waiting, is a technique in which one thread waits for some condition to happen, without calling wait or sleep methods and releasing the CPU. In this condition, one can pause a thread by making it run an empty loop for a certain time period, and it does not even give CPY control. Therefore, it is used to preserve CPU caches and avoid the cost of rebuilding cache.

33. What is ConcurrentHashMap and Hashtable? In java, why is ConcurrentHashMap considered faster than Hashtable?

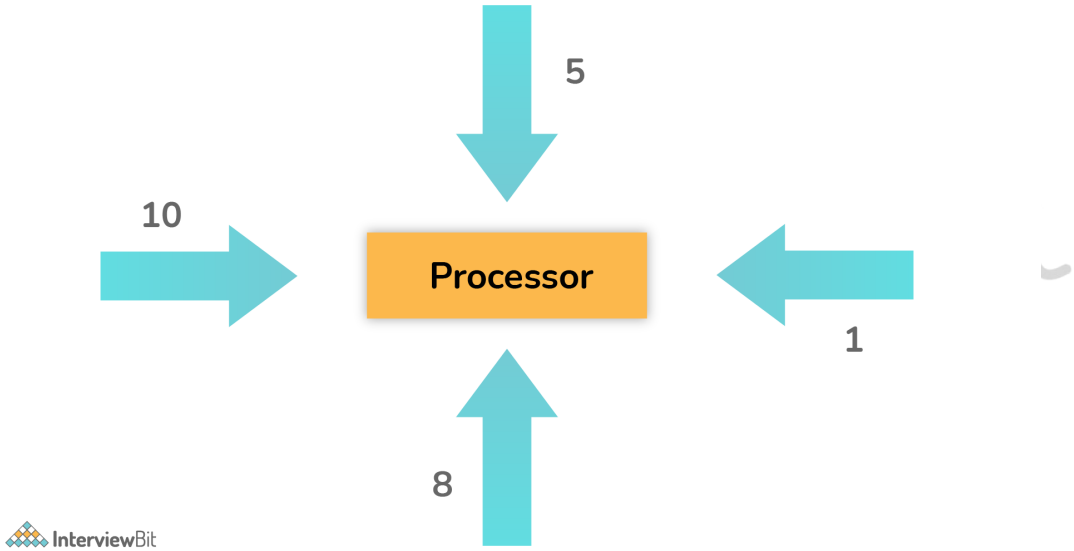
ConcurrentHashMap: It was introduced in Java 1.5 to store data using multiple buckets. As the name suggests, it allows concurrent read and writes operations to the map. It only locks a certain portion of the map while doing iteration to provide thread safety so that other readers can still have access to the map without waiting for iteration to complete.

Hashtable: It is a thread-safe legacy class that was introduced in old versions of java to store key or value pairs using a hash table. It does not provide any lock-free read, unlike ConcurrentHashMap. It just locks the entire map while doing iteration.

ConcurrentHashMap and Hashtable, both are thread-safe but ConcurrentHashMap generally avoids read locks and improves performance, unlike Hashtable. ConcurrentHashMap also provides lock-free reads, unlike Hashtable. Therefore, ConcurrentHashMap is considered faster than Hashtable especially when the number of readers is more as compared to the number of writers.

34. Explain thread priority.

Thread priority simply means that threads with the highest priority will get a chance for execution prior to low-priority threads. One can specify the priority but it's not necessary that the highest priority thread will get executed before the lower-priority thread. Thread scheduler assigns processor to thread on the basis of thread priority. The range of priority changes between 1-10 from lowest priority to highest priority.



35. What do you mean by the ThreadLocal variable in Java?

ThreadLocal variables are special kinds of variables created and provided by the Java ThreadLocal class. These variables are only allowed to be read and written by the same thread. Two threads cannot be able to see each other's ThreadLocal variable, so even if they will execute the same code, then there won't be any race condition and the code will be thread-safe.

Example:

```
public class ThreadLocalExp
{
     public static class MyRunnable implements Runnable
   {
       private ThreadLocal<Integer> threadLocal =
              new ThreadLocal<Integer>();
      @Override
       public void run() {
           threadLocal.set( (int) (Math.random() * 50D) );
           try
           {
               Thread.sleep(1000);
           } catch (InterruptedException e) {
           }
           System.out.println(threadLocal.get());
       }
   }
   public static void main(String[] args)
   {
       MyRunnable runnableInstance = new MyRunnable();
       Thread t1 = new Thread(runnableInstance);
       Thread t2 = new Thread(runnableInstance);
      // this will call run() method
       t1.start();
       t2.start();
   }
}
```

Output:

```
10
33
10 33
```

36. What is semaphore?

Semaphore is regarded as a thread synchronization construct that is usually required to control and manage the access to the shared resource using counters. It simply sets the limit of the thread. The semaphore class is defined within the package java.util.concurrent and can be used to send signals between threads to avoid missed signals or to guard critical sections. It can also be used to implement resource pools or bounded collection.

37. Explain Thread Group. Why should we not use it?

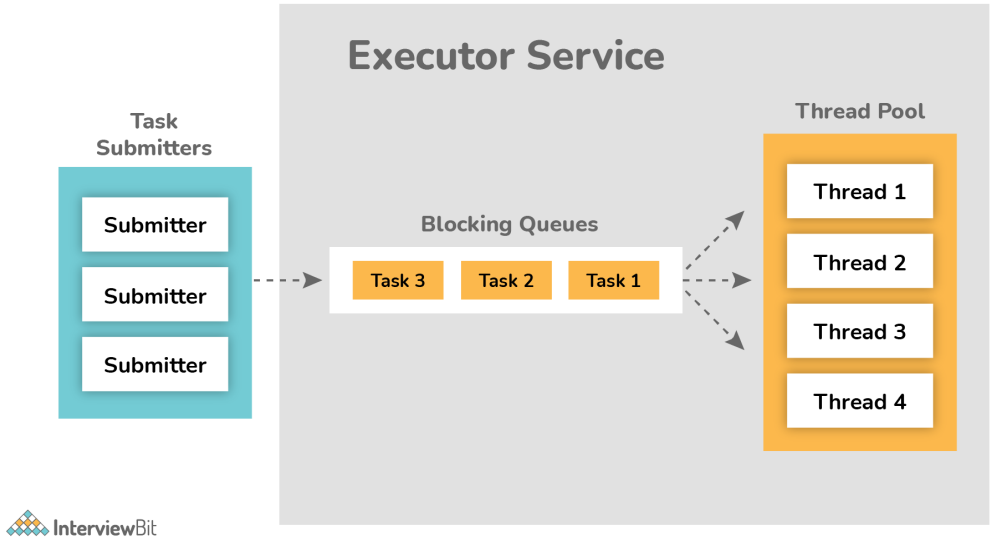
ThreadGroup is a class that is used to create multiple groups of threads in a single object. This group of threads is present in the form of three structures in which every thread group has a parent except the initial thread. Thread groups can contain other thread groups also. A thread is only allowed to have access to information about its own thread group, not other thread groups.

Previously in the old version of Java, the only functionality that did not work without a thread group was uncaughtException(Thread t, Throwable e). But now in Java 5 versions, there is Thread.setUncaughtExceptionHandler(UncaughtExceptionHandler). So now even that works without thread groups and therefore, there is no need to use thread groups.

```
t1.setUncaughtExceptionHandler(new UncaughtExceptionHandler()
{
@Override
public void uncaughtException(Thread t, Throwable e)
{
System.out.println("exception occured:"+e.getMessage());
}
};
```

38. What is the ExecutorService interface?

ExecutorService interface is basically a sub-interface of Executor interface with some additional methods or features that help in managing and controlling the execution of threads. It enables us to execute tasks asynchronously on threads.



Example:

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class TestThread {
                                    public static void main(final String[] arguments) th
ExecutorService e = Executors.newSingleThreadExecutor();

     try {
       e.submit(new Thread());
        System.out.println("Shutdown executor");
        e.shutdown();
        e.awaitTermination(5, TimeUnit.SECONDS);
  } catch (InterruptedException ex) {
       System.err.println("tasks interrupted");
  } finally {

        if (!e.isTerminated()) {
           System.err.println("cancel non-finished tasks");
     }
        e.shutdownNow();
        System.out.println("shutdown finished");
  }
  }

  static class Task implements Runnable {

     public void run() {

        try {
        Long duration = (long) (Math.random() * 20);
           System.out.println("Running Task!");
           TimeUnit.SECONDS.sleep(duration);
     } catch (InterruptedException ex) {
           ex.printStackTrace();
     }
  }
 }
}
```

Output:

```
Shutdown executor
shutdown finished
```

39. What will happen if we don't override the thread class run() method?

Nothing will happen as such if we don't override the run() method. The compiler will not show any error. It will execute the run() method of thread class and we will just don't get any output because the run() method is with an empty implementation.

Example:

```
class MyThread extends Thread {
  //don't override run() method
}
public class DontOverrideRun {
  public static void main(String[] args) {
         System.out.println("Started Main.");
         MyThread thread1=new MyThread();
      thread1.start();
         System.out.println("Ended Main.");
  }
}
```

Output:

```
Started Main.
Ended Main.
```

40. What is the lock interface? Why is it better to use a lock interface rather than a synchronized block.?

Lock interface was introduced in Java 1.5 and is generally used as a synchronization mechanism to provide important operations for blocking.

Advantages of using Lock interface over Synchronization block:

- Methods of Lock interface i.e., Lock() and Unlock() can be called in different methods. It is the main advantage of a lock interface over a synchronized block because the synchronized block is fully contained in a single method.
- Lock interface is more flexible and makes sure that the longest waiting thread gets a fair chance for execution, unlike the synchronization block.

41. Is it possible to call the run() method directly to start a new thread?

No, it's not possible at all. You need to call the start method to create a new thread otherwise run method won't create a new thread. Instead, it will execute in the current thread.

42. Is it possible that each thread can have its stack in multithreaded programming?

Of course, it is possible. In multithreaded programming, each thread maintains its own separate stack area in memory because of which every thread is independent of each other rather than dependent.

Conclusion

43. Conclusion

Overall, multithreading is a very essential part of Java and modern software development. It is very helpful in making the program more efficient and also reduces the usage of storage resources. In this article, we have discussed important interview questions related to multithreading along with answers that were asked mostly in the Interviews and will help you to crack your interviews.

Recommended Tutorials:
Practice
Java Developer Skills

Links to More Interview Questions

C Interview Questions	Php Interview Questions	C Sharp Interview Questions
Web Api Interview Questions	Hibernate Interview Questions	Node Js Interview Questions
Cpp Interview Questions	Oops Interview Questions	Devops Interview Questions
Machine Learning Interview Questions	Docker Interview Questions	Mysql Interview Questions
Css Interview Questions	Laravel Interview Questions	Asp Net Interview Questions
Django Interview Questions	Dot Net Interview Questions	Kubernetes Interview Questions
Operating System Interview Questions	React Native Interview Questions	Aws Interview Questions
Git Interview Questions	Java 8 Interview Questions	Mongodb Interview Questions
Dbms Interview Questions	Spring Boot Interview Questions	Power Bi Interview Questions
Pl Sql Interview Questions	Tableau Interview Questions	Linux Interview Questions
Ansible Interview Questions	Java Interview Questions	Jenkins Interview Questions