Latest Java Multi threading Interview Questions 2023

Description

This is going to be a comprehensive post on multi threading. In this post you will see all the questions from the very basic to the most advanced ones. You go over the question sequentially so that you get the basic idea of multi threading, even if you are an absolute beginner. If you are already aware of multithreading then you can go by any sequence.

What is a Thread in Java

Thread is the smallest unit of execution in a process. Process consists of multiple threads running as part of it. A process has a state associated with it and the state is shared with all the threads. This is also called as a global shared state which is shared among all the threads. In addition to this, the threads have their own private state. Threads of a process can share the resources allocated to that particular process, including memory address space.

What are the benefits of Threads

- Usage of Threads yield higher throughput (amount of work done per unit of time).
- Allows to write effective programs that utilize maximum CPU time.
- Increase use of CPU resources and reduce costs of maintenance.
- Efficient Utilization of resources.
- Thread creation is considered light weight as compared to spawning a new process.
- Less resource-intensive than executing multiple processes at the same time.

What are the problems or issues of using Threads

- **Higher cost of code maintenance,** a new programmer or and existing programmer will find it difficult to go over the code flow.
- **Increased system resource utilization**, Each new thread creation consumes additional memory, CPU cycles for book-keeping and waste of time in context switches.
- **Very difficult to fix bugs,** sometimes it is very difficult to fix bugs as the reason may be due to multiple thread usage.
- **Application or program slowness,** Acquiring and releasing locks may increase the execution time. There should be proper coordination amongst threads.

What are the different types of threads

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

What is the difference between the User thread and the Daemon thread

User Thread	Daemon Thread
JVM waits for user threads to finish their tasks before termination.Â	JVM does not wait for daemon threads to finish their tasks before termination.
User threads are created by the users for executing the task concurrently.	Daemon threads are created by the JVM.
User threads are referred to as high priority threads	Daemon threads are referred to as low priority threads
User threads are normally application threads which run in the foreground.	Daemon threads are normally perform background tasks like garbage collection etc.

Can we create daemon threads

We can create daemon threads by calling **t1.setDaemon(true)** , where t1 is a user thread. **isDaemon()** Â method is used to check whether the current thread is daemon thread or not. If the thread is a daemon, it will return true otherwise it returns false. Â

Difference between Program vs Process vs Thread

Program – A program is a set of instructions and associated data that resides on the disk and is loaded by the operating system to perform some task. The operating system's kernel is first asked to create a new process, which is an environment in which a program executes.
Example – Microsoft word is a program that contains a set of instructions written to execute the program to save or read the contents present on disk

Process – A Process is defined as a program in execution. In other words, It is an execution instance of the program. Processes don't share any resources amongst themselves
Example – When you double click on the Microsoft word, the process is started. If you open multiple word files then you can consider it as multiple process.

Thread– Thread is the smallest unit of execution in a process. Process consists of multiple threads

running as part of it. A process has a state associated with it and the state is shared with all the threads. This is also called as a global shared state which is shared among all the threads. In addition to this, the threads have their own private state. Threads of a process can share the resources allocated to that particular process, including memory address space.

Difference between Process and Thread

Thread	Process
Thread is the smallest executable unit of a process	Process is an executable instance of a program.
Thread is easier to create, lightweight, and have less overhead.	Processes are difficult to create, heavyweight, and have more overhead.
Multiple threads utilize the same memory space of the process which it belongs to.	Process has its own memory space.
Processes with multiple threads use fewer resources.	Processes without threads use more resources.
Inter thread communication is fast as they share memory or resources.	Inter process communication is slow as they have separate memory or resources.
Context switching between threads is inexpensive.	Context switching between process is expensive.
There is a need for synchronization in threads to avoid unexpected scenarios or problems.	There is no need for synchronization in each process.

What are the two ways of implementing thread in Java

There are basically two ways of implementing threads in Java.

- By Extending a Thread class

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

Output :- Hello My thread is in running state.

- By Implementing Runnable interface.

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

Output :- Hello My thread is in running state.

Is it possible to start a thread twice

No, It is not possible to start a thread more than once. If we try to start the thread more than once, it throws IllegalThreadStateException runtime exception.

Can we call the run() method instead of start()

Yes it is possible to call run() method, but there is not be any context-switching between the threads. When we call the start() method, it internally calls the run() method, which creates a new stack for a thread while directly calling the run() will not create a new stack.

What is the difference between Thread class and Runnable interface for creating a Thread

- Once you extend the Thread class, you cannot extend any other class because Java does not support multiple inheritance.
- While implementing the Runnable interface, we can extend other class if required.
- When you extend the thread class, each thread will create a unique object and associates with it.
- While implementing the Runnable interface, multiple threads will share the same object.
- Thread class provides multiple methods like isAlive, getPriority etc, but Runnable interface provides only run() method.

Difference between Synchronous and Asynchronous

Synchronous	Asynchronous
In Synchronous execution, when caller invokes a method, unless the control returns back from the method, next line of code is not executed.	In Asynchronous execution, when a caller invokes a method, the caller will not wait for the method response. It will execute the rest of the program.
Synchronous execution blocks at each method call before proceeding to the next line of code.	Asynchronous execution refers to execution that does not block when invoking methods.
Already supported in Java.	Java has become more robust starting with Java 8 when it comes tot Asynchronous programming.

Whatâ€™s the difference between class lock and object lock

Class level lock – Class level lock is used when we want to prevent multiple threads to enter into the synchronized block in any of all available runtime instances. Class level lock is used to make static data thread safe. There is always one class level lock even though multiple objects of the class exist.

```
public class ClassLevelLock
{
  public void classLevelLockMethod()
 {
     synchronized (ClassLevelLock.class)
       {
            //code here
       }
 }
}
```

Object level lock – Object level lock is used when we want only one thread to access the non-static method or non-static block. Every object of the class can have their own lock.

```
public class ObjectLevelLock
{
  public void objectLevelLockMethod()
```

```
 {
     synchronized (this)
       {
            //code here
       }
 }
}
```

What is Thread safety in multithreading

If multiple threads consume the code without causing race condition or state corruption.

What is the use of Thread sleep() method in Java

sleep() method is used to block a thread for a defined time i.e. It pauses the execution of the thread for a specific time.

Difference between wait() and sleep() methods in Java

wait()

- wait() method is defined in Object class.
- wait() is a non static method which causes the current thread to wait and sleep until another threads call the notify() or notifyAll() method for the object's monitor (lock).
- It releases the monitor for the object and it is placed in the wait queue.
- wait() method is used for inter-thread communication.
- wait() method should always be called from the synchronized method or block.
- wait() method releases the lock.

sleep()

- sleep() method is defined in Thread class.
- sleep() is a static method which stops the execution of the current thread for specific time duration and gives priority to another thread if available.
- This method does not release the lock while waiting.
- sleep() method should always be called from the synchronized block or method,
- When the waiting time completed then again previous thread changes its state from waiting to runnable and comes in running state, and the whole process works so on till the execution doesn't complete.
- sleep() method doesn't release the lock.

Difference between wait() and notify() in Java

wait() – When thread executes the wait() method, it releases the monitor for the object and it is placed in the wait queue. The thread must hold the monitor of the object on which it will call wait. Otherwise an IllegalMonitor exception is raised.

notify() – notify() can only be called by the thread which owns the monitor for the object on which notify() is being called else an illegal monitor exception is thrown. The notify method, will awaken one of the threads in the associated wait queue, i.e., waiting on the thread's monitor. However, this thread will not be scheduled for execution immediately and will compete with other active threads that are trying to synchronize on the same object. The thread which executed notify will also need to give up the object's monitor, before any one of the competing threads can acquire the monitor and proceed forward.

notifyAll() – This method is the same as the notify() one except that it wakes up all the threads that are waiting on the object's monitor.

Difference between notify() and notifyAll()

notify() method is used to wake up only a single thread instead of multiple threads which are waiting on the objects monitor.

notifyAll() method wakes up all the threads and allows them to compete for the objects monitor instead of a single thread.

What is Thread pool

A thread pool is a collection of threads which are waiting for the task to be allocated. When a job is assigned to a thread in the thread pool, the moment it completes the execution, the threads are put back into the thread pool. The thread pool consists of fixed number of threads. java.util.concurrent.Executors class is used to create thread pools.

What are the different types of Thread Pools

There are 4 different types of Thread pools

- **newFixedThreadPool** – newFixedThreadPool has a fixed number of threads. Once a thread completes its task, it can be reused to perform another task from the queue.
- **newSingleThreadExecutor** – newSingleThreadExecutor uses a single worker thread to take tasks off the queue and execute them. If the threads dies for any reason, the executor will replace with the new one.
- **newCachedThreadPool** – newCachedThreadPool will create new threads whenever required

and uses older ones when they become available. If the thread is idle for a configurable amount of time then it will get terminated. The newCachedThreadPool is a good choice for short lived asynchronous tasks.

- **newScheduledThreadPool** – newScheduledThreadPool is used to execute task periodically or after a delayed time.

What is Thread.join() in Java multi threading

join() method is used to achieve inter-thread communication. join() method is used to pause the execution of current thread for the some other threads to finish their task. Lets say there are two threads t1 and t2, you can make t1 to hold the execution for some time so that t2 can finish its task. Once t2 finishes its task, t1 resumes its execution. In order for this to happen, you should call join() method on t2 within t1. There are 3 forms of join() method available in Thread class.

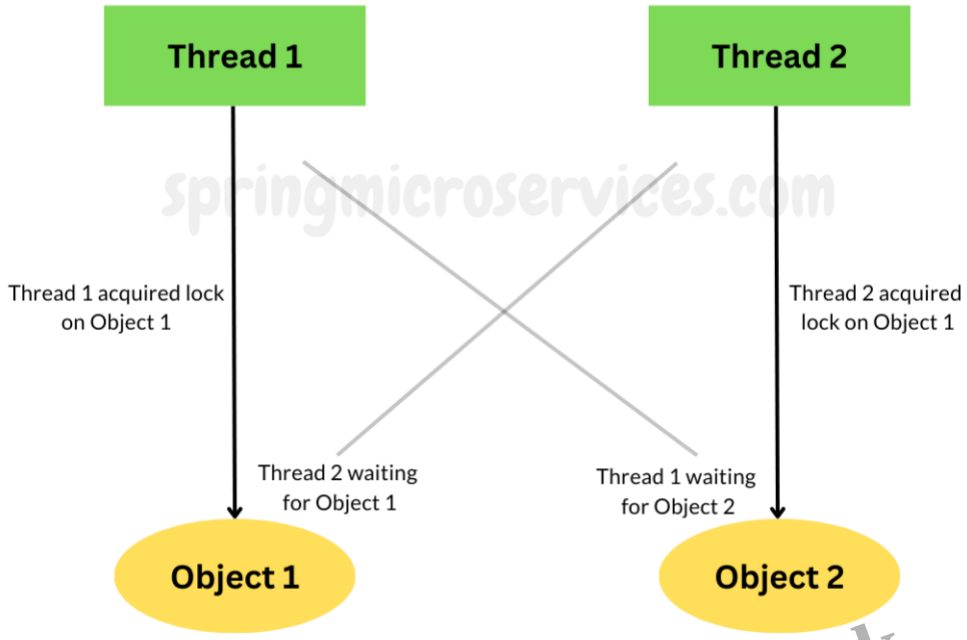
Different join method	Description
public final void join() throws InterruptedException	Currently executing thread waits for a thread to finish itâ€™s task on which it is called
public final void join(long millis) throws InterruptedException	Currently executing thread milliseconds for a thread to finish itâ€™s task on which it is called.
public final void join(long millis, int nanos) throws InterruptedException	Currently executing thread waits milliseconds plus nanoseconds for a thread to finish itâ€™s task on which it is called.

What is use of java.util.concurrent package

java.util.concurrent package provides several classes that can be used for solving concurrency problems. It provides thread safe data structures such asÂ ConcurrentHashMap.

What is a Deadlock in multithreading

Deadlock is a condition when two or more threads fail to move forward with their task because, the resource required by the first thread is held by the second, and the resource required by the second thread is held by the first.



What is Volatile keyword in Java

A volatile keyword is used along with a variable so that the changes made to the value are visible to all the threads as they read the value from the main memory rather than CPU cache. In short volatile keyword is used to maintain thread safety.

Whenever a thread works with a counter variable, it may keep a copy of the counter variable in the CPU cache and manipulate it rather than writing to the main memory. If a variables is marked as volatile, all the threads reads the value directly from main memory rather than CPU cache, so that every thread gets the updated value of the variable instead of stale data from CPU cache.

If a variable is declared volatile then whenever a thread writes or reads to the volatile variable, the read and write always happen in the main memory. All the variables that are visible to the writing thread also gets written-out to the main memory alongside the volatile variable. Similarly, all the variables visible to the reading thread alongside the volatile variable will have the latest values visible to the reading thread.

How to threads communicate with each other or How is inter thread communication achieved.

The process of communication between synchronized threads is called as inter thread communication. **wait()**, **notify()**, and **notifyAll()** methods are used for inter thread communication.

What is Synchronized keyword in Java multithreading.

Synchronized keyword is used to achieve thread synchronization in Java. Synchronized keyword is used to restrict access to critical sections one thread at a time. Synchronized keyword can be used along with method names or it can itself be a construct of its own. Synchronization avoids thread interference and memory consistency errors. Synchronization makes sure that only one thread tries to access a shared resource.

```
public synchronized void methodName() {
    //code here
}
```

```
synchronized (flag) {
     //you code here
}
```


Every object in Java has an entity associated called a “monitor lock”. Once the thread gets access to the monitor of the object, it can invoke the methods marked as synchronized. During the locked period, no other thread will be allowed to invoke methods on the object marked as synchronized. When the first thread release the monitor or exists the synchronized method then only other threads will be allowed to access it. For static methods, the monitor will be the class object, which is distinct from the monitor of each instance of the same class. One thing to note is monitor is released when any uncaught exception occurs in synchronized method.

Important thing to note is overuse of synchronized keyword might reduce the throughput.

Difference between Synchronized method and synchronized block? which one to use when?

Synchronized method

```
public synchronized void methodName() {
    //code here
}
```

Synchronized methods are methods where the thread acquires the lock on the object before entering into it, and exists the method when method execution is complete. The lock can get released when any exception occurs. No other thread can enter into the method, unless the current thread finish the

execution or release the lock.

Synchronized block

```
synchronized (flag) {
     //you code here
}
```

Synchronized block is the section of code which are enclosed with parenthesis along with synchronized keyword as seen in the above example. Similar to synchronized methods, the thread acquires the lock on the object between parenthesis after the synchronized keyword, and releases the lock once they leave.

Synchronized block can be used over Synchronized methods, if you want other parts of the program accessible to other threads. Synchronized blocks boost performance because it locks only a certain portion of the method rather than entire method,

What is Concurrency

A system which executes several programs in the overlapping time intervals is called as concurrent system or concurrency. In concurrency, the execution of two or more programs does not happen simultaneously. The programs do make progress in their execution but at a given point in time a programs suspends and other carries on with the execution. The main goal is to minimize latency and maximize throughput.

Operating system running on a single core machine is concurrent in nature. It processes one task at a given point of time. But as you do multiple things. But all the tasks managed by the operating system appear to make progress because of Operating systems concurrent nature. Each task gets a slice of CPU time to execute and move ahead.

Example – If you can think of a juggler from circus, he would handle several balls at the same time. But at a specific time he can have only one ball in his hand. Each ball gets a time slice.

Difference between Concurrency and Parallelism

Concurrency	Parallelism
Concurrency is when several programs run in the overlapping time intervals. Operating system running on a single core machine is concurrent in nature. It processes one task at a given point of time.	Parallelism is running multiple programs at the same time. Operating system running on multicore processors can execute multiple programs independently and at the same time.
Concurrent system need not be parallel.	Parallel is in fact concurrent.

Concurrency	Parallelism
Concurrency is ***dealing*** with lot of tasks at once.	Parallelism is ***doing*** lot of tasks at once.
Example – Customers waiting in two queues in front of a single coffee machine, single customer being addressed alternatively from both queues one by one.	**Example** – Serving each customer queue with a dedicated coffee machine.

System can be both concurrent and parallel. E.g. multitasking operating system running on multicore machine.

Difference between I/O bound and CPU Bound

Programs require one of the below four resources in order to execute. Depending upon which resource is required heavily they are categorized into I/O bound vs CPU bound.

- Networking Resources
- Memory
- CPU Time
- Disk Storage

I/O Bound –

CPU bound – Programs that are CPU intensive and require high CPU utilization are called CPU bound programs. Tasks such as image processing, data manipulation, matrix multiplication are CPU bound. More CPU mean more efficient and faster execution.

I/O bound – I/O bound programs spend most of their time reading or writing to main memory or network interfaces. I/O bound programs have low CPU utilization as compared to CPU bound programs.

What is Critical Section

When more than one thread of the application is trying to execute same piece of code concurrently is called as Critical section. It has a risk of exposing any shared data or resources used by the application.

What is Race Condition

Race condition occurs when threads try to access program variables or shared resources that might be accessed or evaluated by the other threads at the same time which causes application data to be inconsistent. Race condition takes place when threads run through the critical section without proper thread synchronization.

What is Liveness in multithreading

The ability of an application or a program to execute the task in timely manner is called liveness. This is exactly opposite of deadlock.

What is Live-Lock in multithreading

Live-Lock occurs when two threads continuously react in response to the action by other thread action, but those two threads are still not making progress in their task. The best example is to think of two person John and Doe trying to cross each other in a hallway. John moves to the left to let Doe pass, and and Doe moves to the right to let John pass. Both are blocking each other. John see he’s blocking Doe again and moves to his right and Doe moves to his left seeing he’s blocking John. They never cross each other hence blocking each other.

What is Starvation in multithreading

Starvation in multithreading is when an application thread does not get CPU time or access to shared resource. Other threads (greedy ðŸ˜) continuously access shared resource not letting the starved thread make any kind of progress.

What is Reentrant Lock

Reentrant lock is a class which implements the Lock interface. When any object of the class is locked twice in succession, then it would result in deadlock. The same thread gets blocked on itself. Re-entrant locks allow for re-locking or re-entering of a synchronization lock.

What is Mutex

Mutex also known as mutual exclusion is used to protect shared data by making sure that only one thread accesses it. Shared data can either be an array, a primitive type or linked list. A mutex allows only a single thread to access a resource or critical section.

When a thread acquires a mutex, all other threads trying to acquire the same mutex are blocked until the first thread releases the mutex. Once the first thread releases the mutex, then any one of the waiting threads is allowed to acquire the mutex and make progress.

What is Semaphore

Semaphore is used to limit access to a collection of resources. To understand Semaphores properly

we can think of database connections analogy. Consider there are only 20 connections available and 100 threads are requesting for the connection. At a given time only 20 threads are allowed to connect to the database. The new thread will only be allowed when any of the existing connected threads finishes its execution. Semaphores can also be used for signaling among threads.

Difference between Mutex and Semaphore

Mutex	Semaphore
Mutex implies mutual exclusion which is enables only single thread to access shared resource or critical section at a given time.	Semaphore is used for signaling among threads. Semaphores can also be used as mutex.
Mutex is owned by thread.	There is no concept of ownership in Semaphore.
If a particular thread has locked a mutex then same thread has to unlock it.	A semaphore can be acted upon by different threads.
Example – Think of Mutex as a single runway in an airport, where only a single jet can land or take off at a given point in time.	**Example** – Think of Semaphores as bike rental service in Goa. A bike rental firm can only rent out N number of available bikes. If all the available bikes are rented out, then new customers need to wait till any bikes is available.

What is context switching

Context switching is basically switching of CPU from one thread or process to another. It allows multiple processes to share the same CPU. The state of process or thread is stored so that the execution can be resumed later if required.

What is Thread starvation

Thread starvation is a situation which happens when a thread does not get access to shared resources and due of this it is not able to proceed further. Thread starvation mostly happens with low priority threads as they don’t get CPU for its execution because most of the high priority threads occupy the resources for a long time.

What is Reentrant Lock

Reentrant Lock is similar to the implicit monitor lock accessed when using synchronized blocks or methods. With the reentrant lock, you are free to lock and unlock it in different methods but not with different threads. If you attempt to unlock a reentrant lock object by a thread which didn’t lock it initially, you will get an *IllegalMonitorStateException*. This behavior is similar to when a thread attempts to

unlock a thread mutex.

What is Cyclic barrier

Cyclic barrier is a synchronization mechanism present in **java.util.concurrent** package, which allows multiple threads to wait for each other at a common point known as barrier before continuing execution. The threads wait for each other by calling **await()** method on the **CyclicBarrier**.

CyclicBarrier has an integer (which is initialized) that denotes the number of threads that need to call **await()** method on the barrier. The second argument in CyclicBarrier's constructor is a **Runnable** instance that includes the action to be executed once the last thread arrives.

What is CountDownLatch

CountDownLatch which is part of **java.util.concurrent** package is used to block a single or multiple threads while other threads complete their operation. A **CountdownLatch** is initialized with the number of threads that are required to wait until other threads complete their operation. Every time a thread finishes its task, the thread invokes **countDown()** which decrements the counter by 1. When the count reaches zero, the threads which were waiting on the **await()** method are notified and it resumes the execution.

What is Thread Scheduler

Thread scheduler is a component of JVM that decides which thread to execute if multiple threads are waiting to get the chance of execution. Based on the thread priority, thread scheduler selects the next READY thread to execute. Preemptive Scheduling or Time slicing scheduling is used to schedule threads.

What is Time Slicing

Time Slicing is used to divide the CPU time and allocate them to active threads. Each thread will get a slice of time to execute. Every thread will get a chance to execute in a round robin manner. Each thread gets executed in for a fixed time period.

What is Preemptive multitasking

Preemptive multitasking is when the operating systems preempts (prevent) a program to allow another waiting task to run on the CPU. The operating system scheduler decides which program or thread are allowed to use the CPU and for how much time. Preemptive multitasking is a core feature of Unix based systems.

What is Cooperative multitasking

As the name suggests, cooperative multitasking involves programs to voluntarily give up the control to the scheduler so that other program can run on the CPU. The program or thread may give up the control if it becomes idle or logically blocked. Control can also be given up if the allocated time period has expired.

Difference between preemptive scheduling and time slicing

In preemptive scheduling, the high priority task executes until it enters into dead or waiting state. Whereas in time slicing, a task executes for a predefined time slice and the reenters the pool of ready tasks. Later scheduler decides which task should execute next based on priority.

What is a shutdown hook

Shutdown hook is thread which is used to perform the resource cleanup before the JVM shutdown. Shutdown hook is a thread which is invoked implicitly. Below code will add introduce a shutdown hook.

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

What is Thread priority

Thread priority means that threads with the highest priority will get more preference over threads with low priority. You can change the priority from low to high, but there is no guarantee that it will get more preference over low priority threads. The priority of threads ranges from 1 to 10, where 1 being the lowest priority and 10 being the highest.

What is Executor framework

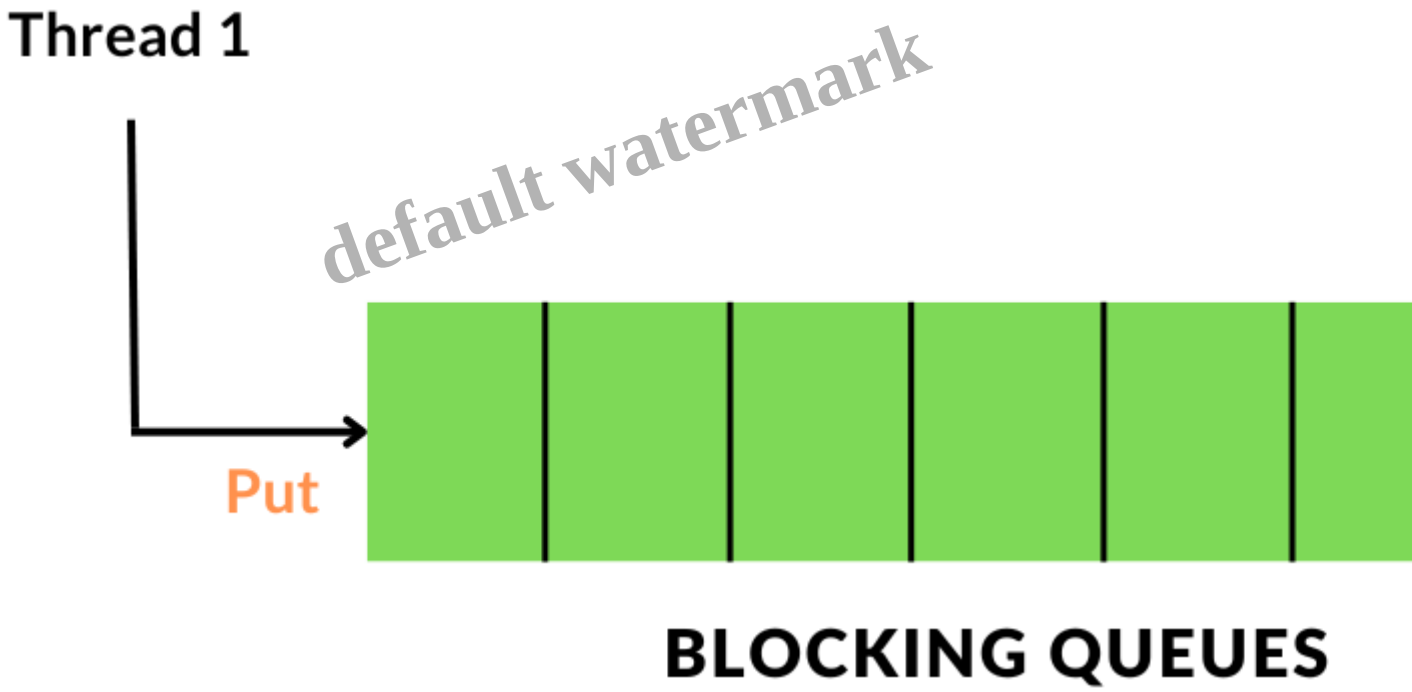
Executor framework is like a thread management system in your application which helps the developers with regards to thread house-keeping. The classes in Executor framework are categorized into two i.e. Task Submission and Task Execution.

Does each thread have its own stack?

Yes, each thread has its own separate stack area in memory. Every thread is independent of each other rather than dependent.

What is a BlockingQueue

A blocking queue is a queue which has limited capacity to hold the elements. Producer thread holds the job of inserting elements into the queue using put() method and Consumer thread holds the job of consuming elements using take() method. The Producer thread is blockedÂ if there's no more capacity to add the new items. Similarly, the queue blocks the consumer if there are no items in the queue. Also, the queue notifies a blocked enqueuing thread or producer when space becomes available and a blocked dequeuing thread or consumer when an item becomes available in the queue.



What do you mean by the ThreadLocal variable in Java

ThreadLocal variable is a type of variable which is read and writen by the same thread. Two threads cannot see each others ThreadLocal variable. ThreadLocal variable can be declared in the below way.

```
ThreadLocal<Integer> counter = ThreadLocal.withInitial(() -> 0);
```

Category

1. Interview

Date Created
May 29, 2023
Author
kk-ravi144gmail-com