Java Multithreading and Concurrency Interview Questions

Multithreading and Synchronization are considered as the typical chapter in java programming. In game development companies, multithreading related interview questions are asked mostly. A list of frequently asked java multithreading and concurrency interview questions is given below.

Multithreading Interview Questions

1) What is multithreading?

Multithreading is a process of executing multiple threads simultaneously. Multithreading is used to obtain the multitasking. It consumes less memory and gives the fast and efficient performance. Its main advantages are:

- Threads share the same address space.
- The thread is lightweight.
- The cost of communication between the processes is low.

More details.

2) What is the thread?

A thread is a lightweight subprocess. It is a separate path of execution because each thread runs in a different stack frame. A process may contain multiple threads. Threads share the process resources, but still, they execute independently.

More details.

3) Differentiate between process and thread?

There are the following differences between the process and thread.

- A Program in the execution is called the process whereas; A thread is a subset of the process
- Processes are independent whereas threads are the subset of process.
- Process have different address space in memory, while threads contain a shared address space.
- Context switching is faster between the threads as compared to processes.
- Inter-process communication is slower and expensive than inter-thread communication.

- Any change in Parent process doesn't affect the child process whereas changes in parent thread can affect the child thread.

Java Multithreading
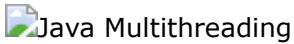
4) What do you understand by inter-thread communication?

- The process of communication between synchronized threads is termed as inter-thread communication.

- Inter-thread communication is used to avoid thread polling in Java.

- The thread is paused running in its critical section, and another thread is allowed to enter (or lock) in the same critical section to be executed.

- It can be obtained by wait(), notify(), and notifyAll() methods.

5) What is the purpose of wait() method in Java?

The wait() method is provided by the Object class in Java. This method is used for inter-thread communication in Java. The java.lang.Object.wait() is used to pause the current thread, and wait until another thread does not call the notify() or notifyAll() method. Its syntax is given below.

public final void wait()

6) Why must wait() method be called from the synchronized block?

We must call the wait method otherwise it will throw **java.lang.IllegalMonitorStateException** exception. Moreover, we need wait() method for inter-thread communication with notify() and notifyAll(). Therefore It must be present in the synchronized block for the proper and correct communication.

7) What are the advantages of multithreading?

Multithreading programming has the following advantages:

- Multithreading allows an application/program to be always reactive for input, even already running with some background tasks

- Multithreading allows the faster execution of tasks, as threads execute independently.

- Multithreading provides better utilization of cache memory as threads share the common memory resources.

- Multithreading reduces the number of the required server as one server can execute multiple threads at a time.

8) What are the states in the lifecycle of a Thread?

A thread can have one of the following states during its lifetime:

1. **New:** In this state, a Thread class object is created using a new operator, but the thread is not alive. Thread doesn't start until we call the start() method.
2. **Runnable:** In this state, the thread is ready to run after calling the start() method. However, the thread is not yet selected by the thread scheduler.
3. **Running:** In this state, the thread scheduler picks the thread from the ready state, and the thread is running.
4. **Waiting/Blocked:** In this state, a thread is not running but still alive, or it is waiting for the other thread to finish.
5. **Dead/Terminated:** A thread is in terminated or dead state when the run() method exits.
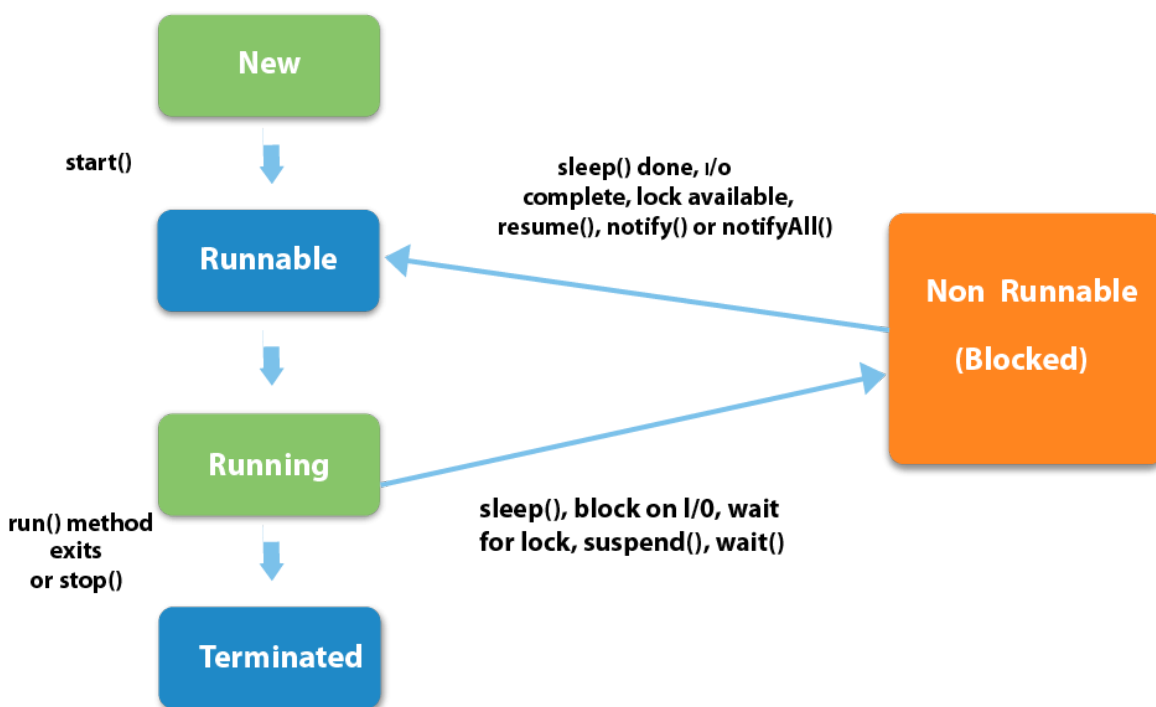
9) What is the difference between preemptive scheduling and time slicing?

Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence. Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

10) What is context switching?

In Context switching the state of the process (or thread) is stored so that it can be restored and execution can be resumed from the same point later. Context switching enables the multiple processes to share the same CPU.

11) Differentiate between the Thread class and Runnable interface for creating a Thread?

The Thread can be created by using two ways.

- By extending the Thread class
- By implementing the Thread class

However, the primary differences between both the ways are given below:

- By extending the Thread class, we cannot extend any other class, as Java does not allow multiple inheritances while implementing the Runnable interface; we can also extend other base class(if required).
- By extending the Thread class, each of thread creates the unique object and associates with it while implementing the Runnable interface; multiple threads share the same object
- Thread class provides various inbuilt methods such as getPriority(), isAlive and many more while the Runnable interface provides a single method, i.e., run().

12) What does join() method?

The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task. Join method is overloaded in Thread class in the following ways.

- public void join()throws InterruptedException
- public void join(long milliseconds)throws InterruptedException

More details.

13) Describe the purpose and working of sleep() method.

The sleep() method in java is used to block a thread for a particular time, which means it pause the execution of a thread for a specific time. There are two methods of doing so.

Syntax:

- public static void sleep(long milliseconds)throws InterruptedException
- public static void sleep(long milliseconds, int nanos)throws InterruptedException

Working of sleep() method

When we call the sleep() method, it pauses the execution of the current thread for the given time and gives priority to another thread(if available). Moreover, 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.

14) What is the difference between wait() and sleep() method?

wait()	sleep()
1) The wait() method is defined in Object class.	The sleep() method is defined in Thread class.
2) The wait() method releases the lock.	The sleep() method doesn't release the lock.

15) Is it possible to start a thread twice?

No, we cannot restart the thread, as once a thread started and executed, it goes to the Dead state. Therefore, if we try to start a thread twice, it will give a runtimeException "java.lang.IllegalThreadStateException". Consider the following example.

```java
public class Multithread1 extends Thread
{
  public void run()
   {
    try {
       System.out.println("thread is executing now........");
    } catch(Exception e) {
    }
   }
  public static void main (String[] args) {
     Multithread1 m1= new Multithread1();
     m1.start();
     m1.start();
  }
}
```

Output

```
thread is executing now........
Exception in thread "main" java.lang.IllegalThreadStateException
        at java.lang.Thread.start(Thread.java:708)
        at Multithread1.main(Multithread1.java:13)
```

More details.

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

Yes, calling run() method directly is valid, but it will not work as a thread instead it will work as a normal object. There will not be 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.

More details.

17) What about the daemon threads?

The daemon threads are the low priority threads that provide the background support and services to the user threads. Daemon thread gets automatically terminated by the JVM if the program remains with the daemon thread only, and all other user threads are ended/died. There are two methods for daemon thread available in the Thread class:

- **public void setDaemon(boolean status):** It used to mark the thread daemon thread or a user thread.

- **public boolean isDaemon():** It checks the thread is daemon or not.

More details.

18)Can we make the user thread as daemon thread if the thread is started?

No, if you do so, it will throw IllegalThreadStateException. Therefore, we can only create a daemon thread before starting the thread.

```
class Testdaemon1 extends Thread{
public void run(){
        System.out.println("Running thread is daemon...");
}
public static void main (String[] args) {
    Testdaemon1 td= new Testdaemon1();
    td.start();
    setDaemon(true);// It will throw the exception: td.
  }
```

```
}
```

Output

```
Running thread is daemon...
Exception in thread "main" java.lang.IllegalThreadStateException
at java.lang.Thread.setDaemon(Thread.java:1359)
at Testdaemon1.main(Testdaemon1.java:8)
```

More details.

19)What is shutdown hook?

The shutdown hook is a thread that is invoked implicitly before JVM shuts down. So we can use it to perform clean up the resource or save the state when JVM shuts down normally or abruptly. We can add shutdown hook by using the following method:

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

Some important points about shutdown hooks are :

- Shutdown hooks initialized but can only be started when JVM shutdown occurred.

- Shutdown hooks are more reliable than the finalizer() because there are very fewer chances that shutdown hooks not run.

- The shutdown hook can be stopped by calling the halt(int) method of Runtime class.

More details.

20)When should we interrupt a thread?

We should interrupt a thread when we want to break out the sleep or wait state of a thread. We can interrupt a thread by calling the interrupt() throwing the InterruptedException.

More details.

21) What is the synchronization?

Synchronization is the capability to control the access of multiple threads to any shared resource. It is used:

1. To prevent thread interference.
2. To prevent consistency problem.

When the multiple threads try to do the same task, there is a possibility of an erroneous result, hence to remove this issue, Java uses the process of synchronization which allows only one thread to be executed at a time. Synchronization can be achieved in three ways:

- by the synchronized method
- by synchronized block
- by static synchronization

Syntax for synchronized block

```
synchronized(object reference expression)
  {
    //code block
  }
```

More details.

22) What is the purpose of the Synchronized block?

The Synchronized block can be used to perform synchronization on any specific resource of the method. Only one thread at a time can execute on a particular resource, and all other threads which attempt to enter the synchronized block are blocked.

- Synchronized block is used to lock an object for any shared resource.
- The scope of the synchronized block is limited to the block on which, it is applied. Its scope is smaller than a method.

More details.

23)Can Java object be locked down for exclusive use by a given thread?

Yes. You can lock an object by putting it in a "synchronized" block. The locked object is inaccessible to any thread other than the one that explicitly claimed it.

24) What is static synchronization?

If you make any static method as synchronized, the lock will be on the class not on the object. If we use the synchronized keyword before a method so it will lock the object (one thread can access an object at a time) but if we use static synchronized so it will lock a class (one thread can access a class at a time). More details.

25)What is the difference between notify() and notifyAll()?

The notify() is used to unblock one waiting thread whereas notifyAll() method is used to unblock all the threads in waiting state.

26)What is the deadlock?

Deadlock is a situation in which every thread is waiting for a resource which is held by some other waiting thread. In this situation, Neither of the thread executes nor it gets the chance to be executed. Instead, there exists a universal waiting state among all the threads. Deadlock is a very complicated situation which can break our code at runtime.

More details.

27) How to detect a deadlock condition? How can it be avoided?

We can detect the deadlock condition by running the code on cmd and collecting the Thread Dump, and if any deadlock is present in the code, then a message will appear on cmd.

Ways to avoid the deadlock condition in Java:

- **Avoid Nested lock:** Nested lock is the common reason for deadlock as deadlock occurs when we provide locks to various threads so we should give one lock to only one thread at some particular time.

- **Avoid unnecessary locks:** we must avoid the locks which are not required.

- **Using thread join:** Thread join helps to wait for a thread until another thread doesn't finish its execution so we can avoid deadlock by maximum use of join method.

28) What is Thread Scheduler in java?

In Java, when we create the threads, they are supervised with the help of a Thread Scheduler, which is the part of JVM. Thread scheduler is only responsible for deciding which thread should be executed. Thread scheduler uses two mechanisms for scheduling the threads: Preemptive and Time Slicing.

Java thread scheduler also works for deciding the following for a thread:

- It selects the priority of the thread.

- It determines the waiting time for a thread

- It checks the Nature of thread

29) Does each thread have its stack in multithreaded programming?

Yes, in multithreaded programming every thread maintains its own or separate stack area in memory due to which every thread is independent of each other.

30) How is the safety of a thread achieved?

If a method or class object can be used by multiple threads at a time without any race condition, then the class is thread-safe. Thread safety is used to make a program safe to use in multithreaded programming. It can be achieved by the following ways:

- Synchronization
- Using Volatile keyword
- Using a lock based mechanism
- Use of atomic wrapper classes

31) What is race-condition?

A Race condition is a problem which occurs in the multithreaded programming when various threads execute simultaneously accessing a shared resource at the same time. The proper use of synchronization can avoid the Race condition.

32) What is the volatile keyword in java?

Volatile keyword is used in multithreaded programming to achieve the thread safety, as a change in one volatile variable is visible to all other threads so one variable can be used by one thread at a time.

33) What do you understand by thread pool?

- Java Thread pool represents a group of worker threads, which are waiting for the task to be allocated.
- Threads in the thread pool are supervised by the service provider which pulls one thread from the pool and assign a job to it.
- After completion of the given task, thread again came to the thread pool.
- The size of the thread pool depends on the total number of threads kept at reserve for execution.

The advantages of the thread pool are :

- Using a thread pool, performance can be enhanced.

- Using a thread pool, better system stability can occur.

Concurrency Interview Questions

34) What are the main components of concurrency API?

Concurrency API can be developed using the class and interfaces of java.util.Concurrent package. There are the following classes and interfaces in java.util.Concurrent package.

- Executor
- FarkJoinPool
- ExecutorService
- ScheduledExecutorService
- Future
- TimeUnit(Enum)
- CountDownLatch
- CyclicBarrier
- Semaphore
- ThreadFactory
- BlockingQueue
- DelayQueue
- Locks
- Phaser

35) What is the Executor interface in Concurrency API in Java?

The Executor Interface provided by the package java.util.concurrent is the simple interface used to execute the new task. The execute() method of Executor interface is used to execute some given command. The syntax of the execute() method is given below.

void execute(Runnable command)

Consider the following example:

```
import java.util.concurrent.Executor;
import java.util.concurrent.Executors;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

public class TestThread {
```

```
    public static void main(final String[] arguments) throws InterruptedException {
        Executor e = Executors.newCachedThreadPool();
        e.execute(new Thread());
        ThreadPoolExecutor pool = (ThreadPoolExecutor)e;
        pool.shutdown();
    }

    static class Thread implements Runnable {
        public void run() {
            try {
                Long duration = (long) (Math.random() * 5);
                System.out.println("Running Thread!");
                TimeUnit.SECONDS.sleep(duration);
                System.out.println("Thread Completed");
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

Output

```
Running Thread!
Thread Completed
```

36) What is BlockingQueue?

The java.util.concurrent.BlockingQueue is the subinterface of Queue that supports the operations such as waiting for the space availability before inserting a new value or waiting for the queue to become non-empty before retrieving an element from it. Consider the following example.

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

public class TestThread {

    public static void main(final String[] arguments) throws InterruptedException {
        BlockingQueue<Integer> queue = new ArrayBlockingQueue<Integer>(10);
```

```java
    Insert i = new Insert(queue);
    Retrieve r = new Retrieve(queue);

    new Thread(i).start();
    new Thread(r).start();

    Thread.sleep(2000);
  }


  static class Insert implements Runnable {
    private BlockingQueue<Integer> queue;

    public Insert(BlockingQueue queue) {
      this.queue = queue;
    }

    @Override
    public void run() {
      Random random = new Random();

      try {
        int result = random.nextInt(200);
        Thread.sleep(1000);
        queue.put(result);
        System.out.println("Added: " + result);

        result = random.nextInt(10);
        Thread.sleep(1000);
        queue.put(result);
        System.out.println("Added: " + result);

        result = random.nextInt(50);
        Thread.sleep(1000);
        queue.put(result);
        System.out.println("Added: " + result);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
  }
```

```java
static class Retrieve implements Runnable {
  private BlockingQueue<Integer> queue;

  public Retrieve(BlockingQueue queue) {
    this.queue = queue;
  }

  @Override
  public void run() {

    try {
      System.out.println("Removed: " + queue.take());
      System.out.println("Removed: " + queue.take());
      System.out.println("Removed: " + queue.take());
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
}
```

Output

```
Added: 96
Removed: 96
Added: 8
Removed: 8
Added: 5
Removed: 5
```

37) How to implement producer-consumer problem by using BlockingQueue?

The producer-consumer problem can be solved by using BlockingQueue in the following way.

```java
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.logging.Level;
import java.util.logging.Logger;
```

```java
public class ProducerConsumerProblem {
    public static void main(String args[]){
        //Creating shared object
        BlockingQueue sharedQueue = new LinkedBlockingQueue();

        //Creating Producer and Consumer Thread
        Thread prod = new Thread(new Producer(sharedQueue));
        Thread cons = new Thread(new Consumer(sharedQueue));

        //Starting producer and Consumer thread
        prod.start();
        cons.start();
    }

}

//Producer Class in java
class Producer implements Runnable {

    private final BlockingQueue sharedQueue;

    public Producer(BlockingQueue sharedQueue) {
        this.sharedQueue = sharedQueue;
    }

    @Override
    public void run() {
        for(int i=0; i<10; i++){
            try {
                System.out.println("Produced: " + i);
                sharedQueue.put(i);
            } catch (InterruptedException ex) {
                Logger.getLogger(Producer.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }

}

//Consumer Class in Java
class Consumer implements Runnable{
```

```
    private final BlockingQueue sharedQueue;


    public Consumer (BlockingQueue sharedQueue) {
        this.sharedQueue = sharedQueue;
    }


    @Override
    public void run() {
        while(true){
            try {
                System.out.println("Consumed: "+ sharedQueue.take());
            } catch (InterruptedException ex) {
                Logger.getLogger(Consumer.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
}
```

Output

```
Produced: 0
Produced: 1
Produced: 2
Produced: 3
Produced: 4
Produced: 5
Produced: 6
Produced: 7
Produced: 8
Produced: 9
Consumed: 0
Consumed: 1
Consumed: 2
Consumed: 3
Consumed: 4
Consumed: 5
Consumed: 6
Consumed: 7
Consumed: 8
Consumed: 9
```

38) What is the difference between Java Callable interface and Runnable interface?

The Callable interface and Runnable interface both are used by the classes which wanted to execute with multiple threads. However, there are two main differences between the both :

- A Callable <V> interface can return a result, whereas the Runnable interface cannot return any result.

- A Callable <V> interface can throw a checked exception, whereas the Runnable interface cannot throw checked exception.

- A Callable <V> interface cannot be used before the Java 5 whereas the Runnable interface can be used.

39) What is the Atomic action in Concurrency in Java?

- The Atomic action is the operation which can be performed in a single unit of a task without any interference of the other operations.

- The Atomic action cannot be stopped in between the task. Once started it fill stop after the completion of the task only.

- An increment operation such as a++ does not allow an atomic action.

- All reads and writes operation for the primitive variable (except long and double) are the atomic operation.

- All reads and writes operation for the volatile variable (including long and double) are the atomic operation.

- The Atomic methods are available in java.util.Concurrent package.

40) What is lock interface in Concurrency API in Java?

The java.util.concurrent.locks.Lock interface is used as the synchronization mechanism. It works similar to the synchronized block. There are a few differences between the lock and synchronized block that are given below.

- Lock interface provides the guarantee of sequence in which the waiting thread will be given the access, whereas the synchronized block doesn't guarantee it.

- Lock interface provides the option of timeout if the lock is not granted whereas the synchronized block doesn't provide that.

- The methods of Lock interface, i.e., Lock() and Unlock() can be called in different methods whereas single synchronized block must be fully contained in a single method.

41) Explain the ExecutorService Interface.

The ExecutorService Interface is the subinterface of Executor interface and adds the features to manage the lifecycle. Consider the following example.

```java
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) throws InterruptedException {
      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
```

42) What is the difference between Synchronous programming and Asynchronous programming regarding a thread?

Synchronous programming: In Synchronous programming model, a thread is assigned to complete a task and hence thread started working on it, and it is only available for other tasks once it will end the assigned task.

Asynchronous Programming: In Asynchronous programming, one job can be completed by multiple threads and hence it provides maximum usability of the various threads.

43) What do you understand by Callable and Future in Java?

Java Callable interface: In Java5 callable interface was provided by the package java.util.concurrent. It is similar to the Runnable interface but it can return a result, and it can throw an Exception. It also provides a run() method for execution of a thread. Java Callable can return any object as it uses Generic.

Syntax:

public interface Callable<V>

Java Future interface: Java Future interface gives the result of a concurrent process. The Callable interface returns the object of java.util.concurrent.Future.

Java Future provides following methods for implementation.

- **cancel(boolean mayInterruptIfRunning):** It is used to cancel the execution of the assigned task.

- **get():** It waits for the time if execution not completed and then retrieved the result.

- **isCancelled():** It returns the Boolean value as it returns true if the task was canceled before the completion.

- **isDone():** It returns true if the job is completed successfully else returns false.

44. What is the difference between ScheduledExecutorService and ExecutorService interface?

ExecutorServcie and ScheduledExecutorService both are the interfaces of java.util.Concurrent package but scheduledExecutorService provides some additional methods to execute the Runnable and Callable tasks with the delay or every fixed time period.

45) Define FutureTask class in Java?

Java FutureTask class provides a base implementation of the Future interface. The result can only be obtained if the execution of one task is completed, and if the computation is not achieved then get method will be blocked. If the execution is completed, then it cannot be re-started and can't be canceled.

Syntax

public class FutureTask<V> extends Object implements RunnableFuture<V>

← prev   next →

1	2	3	4	5	6	7	8

Java Basics Interview Questions	**Java OOPs Interview Questions**
Java Multithreading Interview Questions	**Java String & Exception Interview Questions**
Java Collection Interview Questions	**JDBC Interview Questions**
Servlet Interview Questions	**JSP Interview Questions**
Spring Interview Questions	**Hibernate Interview Questions**
PL/SQL Interview Questions	**SQL Interview Questions**
Oracle Interview Questions	**Android Interview Questions**
SQL Server Interview Questions	**MySQL Interview Questions**