**Multithreading:**

**It is program concept where program is divided into two or more sub-program called Thread which can run** **simultaneously.**

**Thread**:-

1. Thread is used for **dividing** the **task** of an application into separate sub-processes which can run **simultaneously.**

2. Program can be sub divided into the number of small sub processes. **Each** **small** sub process is called as **thread**.

3. Each and every java program have at **least one thread**

4. First thread is created when you invoke the **static main method** of your Java class.

Example:

MS Power Point then –

1. OS will start the process which would run MS Power Point.
2. We can consider the MS Power Point as main process, in order to make interactive execution, MS Power Point process can create multiple sub processes i.e thread.
3. **When we type anything in the slide then *spell checker* thread can be automatically created**. We can also consider the ***auto-correction*** as thread while typing.

Example:1

class X implements Runnable

{

public static void main(String args[])

{

/\* Missing code? \*/

}

public void run() {}

}

To start Thread:

X run = new X();

Thread t = new Thread(run);

t.start();

Example:2

public class MyRunnable implements Runnable

{

public void run()

{

// some code here

}

}

create and start this thread

new Thread(new MyRunnable()).start();

wait(): causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.

notify() - wakes up a **single thread** that is waiting on this object's monitor.

notifyAll() - wakes up **all threads** that are waiting on this object's monitor.

run() **method to a thread is like the main() method to an application.** Starting the thread causes the object's run method to be called in that separately executing thread.run() contain the body of the thread.

start()method registers a thread in a **thread scheduler** and causes this thread to begin execution; the Java Virtual Machine calls the run method of this thread.

Stop() method kill the thread execution.

The Runnable interface only contains 1 method, the void run() method therefore it must be implemented.

**wait()** – until call notify(), notifyAll() from object

**sleep()** – until at least time expire or call interrupt().

**Thread will leave the running state after calling:**

**1.wait()** always causes the current thread to go into the object's wait pool.

**2.sleep()** will always pause the currently running thread for at least the duration specified in the sleep argument (unless an interrupted exception is thrown).

**3.join** ()assuming that the thread you're calling join() on is alive, the thread calling join() will immediately block until the thread you're calling join() on is no longer alive.

Difference Between : sleep(1000) and wait(1000)

The sleep method is used when the thread has to be put aside for a fixed amount of time.

Ex: sleep(1000), puts the thread aside for exactly one second.

The wait method is used to put the thread aside for up to the specified time. It could wait for a much lesser time if it receives a notify() or notifyAll() call.

1. **public** **void** addShutdownHook(Thread hook){}
2. The shutdown hook can be used to perform cleanup resource or **save the state when JVM shuts down normally or abruptly.** Performing clean resource means closing log file, sending some alerts or something else. So **if you want to execute some code before JVM shuts down, use shutdown hook.**

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated

How to create thread

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

### **Commonly used Constructors of Thread class:**

|  |
| --- |
| * Thread() * Thread(String name) * Thread(Runnable r) * Thread(Runnable r,String name) * ThreadGroup(String name) * ThreadGroup(ThreadGroup parent, String name) |
|  |
|  |

### **Starting a thread:**

|  |
| --- |
| **start() method** of Thread class is used to start a newly created thread. It performs following tasks:   * A new thread starts(with new callstack). * The thread moves from New state to the Runnable state. * When the thread gets a chance to execute, its target run() method will run. |

### **1) Java Thread Example by extending Thread class**

**class** Multi **extends** Thread{

**public** **void** run(){

System.out.println("thread is running...");

}

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

Multi t1=**new** Multi();

t1.start();

 }

}

### **Java Thread Example by implementing Runnable interface**

**class** Multi3 **implements** Runnable{

**public** **void** run(){

System.out.println("thread is running...");

}

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

Multi3 m1=**new** Multi3();

Thread t1 =**new** Thread(m1);

t1.start();

 }

}

# Thread Scheduler in Java

**Thread scheduler** in java is the part of the JVM that decides which thread should run.

There is no guarantee that which runnable thread will be chosen to run by the thread scheduler.

Only one thread at a time can run in a single process.

# Can we start a thread twice

No. After starting a thread, it can never be started again. If you does so, an IllegalThreadStateException

Let's understand it by the example given below:

1. **public** **class** TestThreadTwice1 **extends** Thread{
2. **public** **void** run(){
3. System.out.println("running...");
4. }
5. **public** **static** **void** main(String args[]){
6. TestThreadTwice1 t1=**new** TestThreadTwice1();
7. t1.start();
8. t1.start();
9. }
10. }

Output:

running

Exception in thread "main" java.lang.IllegalThreadStateException

# What if we call run() method directly instead start() method?

1. **class** TestCallRun2 **extends** Thread{
2. **public** **void** run(){
3. **for**(**int** i=1;i<5;i++){
4. **try**{Thread.sleep(500);}**catch**(InterruptedException e){System.out.println(e);}
5. System.out.println(i);
6. }
7. }
8. **public** **static** **void** main(String args[]){
9. TestCallRun2 t1=**new** TestCallRun2();
10. TestCallRun2 t2=**new** TestCallRun2();
12. t1.run();
13. t2.run();
14. }
15. }

Output:1

2

3

4

5

1

2

3

4

5

As you can see in the above program that there is no context-switching because here t1 and t2 will be treated as normal object not thread object.

# The join() method

It causes the currently running threads to stop executing until the thread it joins with completes its task.

# Priority of a Thread (Thread Priority):

|  |
| --- |
| Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses. |

## **3 constants defined in Thread class:**

|  |
| --- |
| 1. public static int MIN\_PRIORITY 2. public static int NORM\_PRIORITY 3. public static int MAX\_PRIORITY |

|  |
| --- |
| **Default priority of a thread is 5 (NORM\_PRIORITY). The value of MIN\_PRIORITY is 1 and the value of MAX\_PRIORITY is 10.** |

**Thread Class Constructors:**

**1. Thread() :**

Default constructor – To create thread with default name and priority.

**2. Thread(Runnable target)**

This constructor will create a thread from the runnable object.

**3. Thread(Runnable target, String name)**

This constructor will create thread from runnable object with name as passed in the second argument

**4. Thread(String name)**

This constructor will create a thread with the name as per argument passed.

#### What is need of Synchronization?

Let’s say you want to count number of request you got for a particular URL. If you get two requests at the same time, then count may be inconsistent.

**Without Synchronization:**

public class RequestCounter {

private int count;

public int incrementCount()

{

count++;

return count;

}

}

[**Arpit Mandliya**](https://java2blog.com/author/arpitmandliya/)

Last updated : June 7th, 2019

[**NO COMMENTS**](https://java2blog.com/java-multithreading-interview-questions-and-answers/#respond)

# Java Multithreading interview questions and answers

[**Previous**](https://java2blog.com/java-string-interview-questions-and-answers/)

[**Next**](https://java2blog.com/java-exception-handling-interview-questions-and-answers/)

In this tutorial, we are going to see Multitheading interview questions with answers.  
Here is list of Multithreading interview questions.

#### 1. What is thread in java?

**Answer:**  
Thread can be called as light weight process. It can be referred as smallest part of process which can be executed concurrently with other parts(threads) of process.

#### 2. What is Multithreading?

**Answer:**  
Multithreading is execution of multiple threads concurrently. Java supports multithreading , so it allows your application to perform two or more task concurrently. Multithreading can be of advantage specially when now a days, machine has multiple CPUs, so multiple tasks can be executed concurrently.

#### 3. What are ways to create a thread in java?

**Answer:**  
There are two ways to create a thread in java

* By extending thread class
* By implementing Runnable interface.



You can read about more at [**Java thread example**](https://www.java2blog.com/?p=346).

#### 4. Thread vs Runnable which is better approach to create a thread?

**Answer:**  
Implementing Runnable interface is considered to be better approach than Extending Thread due to following reasons.

* Java does not support multiple inheritance so if you extend Thread class and you can not extend any other class which is needed in most of the cases.
* Runnable interface represents a task and this can be executed with help of Thread class or Executors.
* When you use inheritance, it is because you want to extend some properties of parent, modify or improve class behavior. But if you are extending thread class just to create thread, so it may not be recommended behavior for Object Oriented Programming.

#### 5. What are differences between thread and process?

**Answer:**  
You can go through [**difference between process and thread**](https://www.java2blog.com/?p=12) to see the differences.

#### 6. What are differences between Sleep and wait in java?

|  |  |  |
| --- | --- | --- |
| **Parameter** | **wait** | **sleep** |
| Synchonized | wait should be called from synchronized context i.e. from block or method, If you do not call it using synchronized context, it will throw IllegalMonitorStateException | It need not be called from synchronized block or methods |
| Calls on | wait method operates on Object and defined in Object class | Sleep method operates on current thread and is in java.lang.Thread |
| Release of lock | wait release lock of object on which it is called and also other locks if it holds any | Sleep method does not release lock at all |
| Wake up condition | until call notify() or notifyAll() from Object class | Until time expires or calls interrupt() |
| static | wait is non static method | sleep is static method |

You can refer [**difference between sleep and wait in java**](https://www.java2blog.com/?p=360) for more details.

#### 7. Define states of thread in java?

**Answer:**  
There are 5 states of thread in java

**New**: When you create a thread object and it is not alive yet.

**Runnable:**When you call start method of thread, it goes into Runnable state. Whether it will execute immediately or execute after some times , depends on thread scheduler.

**Running :**When thread is being executed, it goes to running state.

**Blocked :**When thread waits for some resources or some other thread to complete (due to thread’s join), it goes to blocked state.

**Dead:**When thread’s run method returns, thread goes to dead state.

#### 8. Can we call run method directly to start a thread?

**Answer:**  
No, you can not directly call run method to start a thread. You need to call start method to create a new thread. If you call run method directly , it won’t create a new thread and it will be in same stack as main.  
You can refer [**can we call run method directly to start a thread**](https://www.java2blog.com/?p=206) for more details

#### 9. Can we start a thread twice in java?

**Answer:**  
No, Once you have started a thread, it can not be started again. If you try to start thread again , it will throw IllegalThreadStateException.  
You can refer [**can we start thread twice**](https://www.java2blog.com/?p=207) for more details

#### 10. How to make a main thread wait until all other threads finished execution?

**Answer:**  
You can make use of join method to achieve above scenario.  
You can read more about [**join method**](https://www.java2blog.com/?p=344).

#### 11. What are daemon threads?

**Answer:**

Daemon threads are low-priority background threads which provide services to user threads. Its life depends on user threads. If no user thread is running then JVM can exit even if daemon threads are running. JVM does not wait for daemon threads to finish.

#### 12. How can you change user thread to daemon thread?

**Answer:**setDaemon method can be used to mark thread as user thread. If you put setDaemon(true), it makes thread as daemon.

#### 13. What is Synchronization?

**Answer:**

Synchronization is ability to restrict access to shared resource to only one thread. When two or more threads need access to shared resource, there has to be some mechanism such that shared resource will be used by only one thread. The process by which we can achieve it is called Synchronization.

#### 14. What is need of Synchronization?

Let’s understand this with the help of an example.

Let’s say you want to count number of request you got for a particular URL. If you get two requests at the same time, then count may be inconsistent.

**Without Synchronization:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | package org.arpit.java2blog;    public class RequestCounter {    private int count;    public int incrementCount()  {  count++;  return count;  }  } |

**For example:**  
Thread T1 sees count as 20 and increment it to 21. At the same time, thread t2 also sees count as 20 and increment it to 21. This shows that count became inconsistent.

**With Synchronization:**

You can achieve Synchronization using two ways.

* synchronized method
* synchronized block

**You can not use synchronized with  instance or class variables.**

* **Synchronized method**

public class RequestCounter {

private int count;

public synchronized int incrementCount()

{

  count++;

  return count;

}

}

#### **synchronized block**

public class RequestCounter {

private int count;

public int incrementCount() {

  synchronized (this) {

   count++;

   return count;

  }

}

}

# Daemon thread in Java

Daemon thread is a **low priority** thread that runs in background to perform tasks such as garbage collection.

public void setDaemon(boolean ytru/false)

public boolean isDaemon()

**If you call the setDaemon() method after starting the thread, it would throw IllegalThreadStateException.**

DaemonThread t2 = new DaemonThread();

        t1.start();

        // Exception as the thread is already started

        t1.setDaemon(true);