Wait method :

Forces the current thread to wait until some other thread invokes notify() or notifyAll() on the same object.

Wait(long time out ) - we can specify the timeout after which the thread will be woken up automatically.

It’s a native method



Notify :

Choice of thread to wake up is non-deterministic and depends on Implmenetation.

Diagram

Description automatically generated

Diagram

Description automatically generated

New state :

* It remains in the state till the start() methos of the thread class is invoked.
* Calling method other than start() or stop will cause exception to be throwed.

Runnable state :

* New state to runnable state but not started running yet.
* JVM thread scheduler is responsible for decising which thread should start out of thread pool.

Running State

* When scheduler selects a thread and starts running it , then the thread state is changed from runnable to running.

Non Runnable State :

* Thread becomes to non-runnable state in below mentioned reasons
  + When sleep() method is invoked and thread sleeps for specific period of time.
  + When suspend() method is invoked.
  + When wait() method is invoked , thread waits for notification of free resource , for completion of another thread or waits to acquire a lock on object
  + When thread is blocking on I/O and is waiting for completion of it.

It cannot continue it operation until moved to runnable state. It do4es not consume any CPU cycles.

To make non-runnable state to runnable state

* + If thread was on sleep, then once specific time period expires or when the interrupt() method is called.
  + If thread was suspended , then when the resume () method is called.
  + If thread was waiting for free resource or completion of another thread, then when the other thread signals the waiting thread wusing notify() or notifyall() methods.
  + If thread was blocked on I/O , then once the I/O completes.

Terminated State

* + Thread transitions to this state once the run method has finished its execution, when the stop() method is invoked or when a failure like an unhandled exception has occurred.

**Thread Priority**

Every thread in Java has a priority that helps the JVM thread scheduler to determine the order of execution. Java priority ranges from MIN\_PRIORITY(minimum) of 1 to MAX\_PRIORITY(maximum) of 10. The threads that have higher priority are given precedence over the lower priority threads by the thread scheduler but it does not guarantee that it will execute threads based on the priority because the scheduling mechanisms differ from JVM to JVM.

**Does Threads Have A Default Priority?**

The default priority of the main thread in an application is set to 5 by the JVM. All other threads that get created inherits the priority value of its parent thread. For example, after the priority of the main thread is explicitly changed to 8, and if a new thread is created in the main thread, the newly created thread will have a priority of 8.

**How To Create Threads?**

In Java, threads can be created by using two mechanisms;

* Extending the Thread Class
* Implementing the Runnable Interface

**Creating Threads By Extending The Thread Class**

Creation of a thread by extending the Thread class can be done in three steps. Initially create a class that extends the *java.lang.Thread* class. As the second step, override the *run()* method of the Thread class and write the code/logic that needs to be executed inside the *run()* method. As the final step, create an object of the new class and call the *start()* method using that object to start the thread execution. The *start()* method will invoke the *run()* method on the thread object.

**Creating Threads By Implementing The Runnable Interface**

Creation of a thread by implementing the runnable interface is also somewhat similar to the thread creation process using the Thread class. In this scenario, initially create a new class which implements the *java.lang.Runnable* interface. Once done, override the *run()* method and write the code/logic inside it. As the next step, create an object of the newly created class and use that object as a constructor argument to create an object from the Thread class. Finally, call the *start()* method using the created Thread class object.

**Difference Between Thread Class And Runnable Interface Implementations?**

* Java does not support multiple inheritance, therefore cannot extend the Thread class in classes that already inherit another class. But if the runnable interface is implemented, then the class is still capable of extending another class.
* Creating threads by extending the Thread class gives the advantage to use its inbuilt methods like *yield()*, *sleep()* etc. which are not available if the runnable interface is implemented.

**Thread Class Methods**

This section discusses about some of the frequently used static and non-static methods available in the thread class and how those methods work.

**Set/Get Priority Methods**

The *getPriority()* and *setPriority()* methods of the thread class are used to check and update the priorities of the threads which range from a minimum of one to a maximum of ten.

**Join Method**

When a thread invokes this method on a second thread, it notifies that the invoking thread will be waiting for a specified time(time as a method parameter is optional) till the execution of the second thread completes for reasons like getting the output etc. When this method is invoked, the invoking thread will be transitioned to the waiting state. Once transitioned, it cannot directly go back to the running state but rather to the runnable state under one of the following scenarios;

* If second thread finishes before given time.
* If second thread does not finish on given time.
* If invoking thread gets interrupted.

**Yield Method**

Once the *yield()* method is invoked by a thread, it hints the thread scheduler that it’s willing to give a chance to other threads. It will also cause the thread to be transitioned from running state to runnable state thus relinquishing the CPU. The thread scheduler checks for threads in runnable state that have the same priority as the thread that got transitioned. If there are no any waiting threads at all or there are no any threads with the same priority then the *yield()* method will be ignored and the earlier thread will continue executing and if there are threads that match the priority, then the thread scheduler will execute one out of them.

**Sleep Method**

The sleep method causes the currently running thread to pause its execution for a given time period. The thread will be transitioned back to runnable state if the specified duration of sleep is over or when the thread is interrupted. The sleep method has two constructors; first one to specify the sleep time in milliseconds and the second one to specify the sleep time in milliseconds and nanoseconds.

**Interrupt Method**

This method is used to interrupt the thread executions of threads that are in the non runnable state(sleep, wait ,etc.) by throwing *InterruptedException*. If the *interrupt()* method is called on threads that are not in non-runnable state, those threads will not get interrupted but the interrupt flag will be set to true where if those threads go to non-runnable state at a later point, then those will be interrupted.

**Conclusion**

As discussed within the article, threading is a basic but one of the most important features in the Java programming language. This article was able to provide an informative overview about threads. You can use the knowledge gained from this to dive in and explore more advanced concepts and topics about threading.