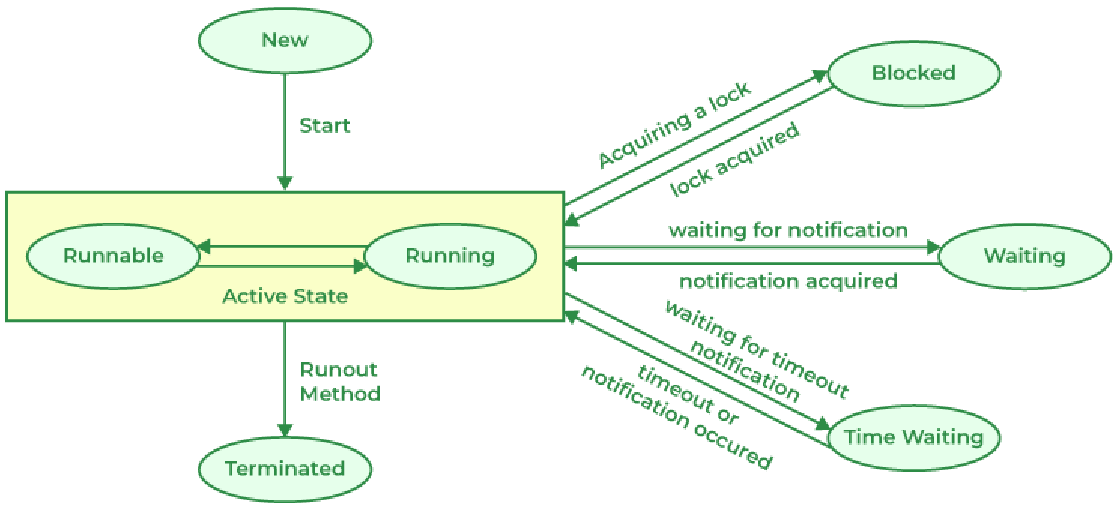
Lifecycle and States of a Thread in Java

Last Updated : 14 Jan, 2025

A thread in Java can exist in any one of the following states at any given time. A thread lies only in one of the shown states at any instant:

1. New State
2. Runnable State
3. Blocked State
4. Waiting State
5. Timed Waiting State
6. Terminated State

The diagram below represents various states of a thread at any instant.



Life Cycle of a Thread

There are multiple states of the thread in a lifecycle as mentioned below:

1. **New Thread:** When a new thread is created, it is in the new state. The thread has not yet started to run when the thread is in this state. When a thread lies in the new state, its code is yet to be run and hasn’t started to execute.
2. **Runnable State:** A thread that is ready to run is moved to a runnable state. In this state, a thread might actually be running or it might be ready to run at any instant of time. It is the responsibility of the thread scheduler to give the thread, time to run. A multi-threaded program allocates a fixed amount of time to each individual thread. Each and every thread get a small amount of time to run. After running for a while, a thread pauses and gives up the CPU so that other threads can run.
3. **Blocked:** The thread will be in blocked state when it is trying to acquire a lock but currently the lock is acquired by the other thread. The thread will move from the blocked state to runnable state when it acquires the lock.
4. **Waiting state:** The thread will be in waiting state when it calls wait() method or join() method. It will move to the runnable state when other thread will notify or that thread will be terminated.
5. **Timed Waiting:** A thread lies in a timed waiting state when it calls a method with a time-out parameter. A thread lies in this state until the timeout is completed or until a notification is received. For example, when a thread calls sleep or a conditional wait, it is moved to a timed waiting state.
6. **Terminated State:** A thread terminates because of either of the following reasons:
   - Because it exits normally. This happens when the code of the thread has been entirely executed by the program.
   - Because there occurred some unusual erroneous event, like a segmentation fault or an unhandled exception.

Implementing the Thread States in Java

In Java, to get the current state of the thread, use **Thread.getState()** method to get the current state of the thread. Java provides

java.lang.Thread.State class that defines the ENUM constants for the state of a thread, as a summary of which is given below:

1. New

Thread state for a thread that has not yet started.

```
public static final Thread.State NEW
```

2. Runnable

Thread state for a runnable thread. A thread in the runnable state is executing in the Java virtual machine but it may be waiting for other resources from the operating system such as a processor.

```
public static final Thread.State RUNNABLE
```

3. Blocked

Thread state for a thread blocked waiting for a monitor lock. A thread in the blocked state is waiting for a monitor lock to enter a synchronized block/method or reenter a synchronized block/method after calling Object.wait().

```
public static final Thread.State BLOCKED
```

4. Waiting

Thread state for a waiting thread. A thread is in the waiting state due to calling one of the following methods:

- Object.wait with no timeout
- Thread.join with no timeout
- LockSupport.park

```
public static final Thread.State WAITING
```

5. Timed Waiting

Thread state for a waiting thread with a specified waiting time. A thread is in the timed waiting state due to calling one of the following methods with a specified positive waiting time:

- Thread.sleep
- Object.wait with timeout
- Thread.join with timeout
- LockSupport.parkNanos
- LockSupport.parkUntil

```
public static final Thread.State TIMED_WAITING
```

6. Terminated

Thread state for a terminated thread. The thread has completed execution.

Declaration:

```
public static final Thread.State TERMINATED
```

Example

Below is the implementation of the thread states mentioned above:

```java
// Java program to demonstrate thread states
class thread implements Runnable
{
      // Overriding the run method
      @Override
    public void run()
    {
        // Moving thread2 to timed waiting state
        try {
            Thread.sleep(1500);
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("State of thread1 while it called"
                           + " join() method on thread2 -"
                           + Test.thread1.getState());
```

```
          try {
            Thread.sleep(200);
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public class Test implements Runnable {
    public static Thread thread1;
    public static Test obj;

      // Overriding the run method
      @Override
      public void run()
    {
        thread myThread = new thread();
        Thread thread2 = new Thread(myThread);

        // thread2 created and is currently in the NEW
        // state.
        System.out.println("State of thread2 after creating it - "
                          + thread2.getState());

        thread2.start();

        // thread2 moved to Runnable state
        System.out.println("State of thread2 after calling .start()"
                           + " method on it - " + thread2.getState());

        // moving thread2 to timed waiting state
        try {
            // moving thread2 to timed waiting state
            Thread.sleep(200);
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("State of thread2 after calling .sleep()"
                           + " method on it - " + thread2.getState());

        try {
            // waiting for thread2 to die
            thread2.join();
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("State of thread2 when it has finished "
                           + "it's execution - " + thread2.getState());
    }

    public static void main(String[] args)
    {
        obj = new Test();
        thread1 = new Thread(obj);
        // thread1 created and is currently in the NEW
        // state.
        System.out.println("State of thread1 after creating it - "
                           + thread1.getState());

        thread1.start();

        // thread1 moved to Runnable state
        System.out.println("State of thread1 after calling .start()"
                            + " method on it - " + thread1.getState());
    }
}
```

Output

```
State of thread1 after creating it - NEW
State of thread1 after calling .start() method on it - RUNNABLE
State of thread2 after creating it - NEW
State of thread2 after calling .start() method on it - RUNNABLE
State of thread2 after calling .sleep() method on it -
TIMED_WAITING
State of thread1 while it called join() method on thread2 -
WAITING
State of thread2 when it has finished it's execution - TERMINATED
```

Explanation of the above Program:

- When a new thread is created, the thread is in the NEW state. When the start() method is called on a thread, the thread scheduler moves it to the Runnable state.
- Whenever the join() method is called on a thread instance, the current thread executing that statement will wait for this thread to move to the Terminated state.
- So, before the final statement is printed on the console, the program calls join() on thread2 making the thread1 wait while thread2 completes its execution and is moved to the Terminated state.
- The thread1 goes to the Waiting state because it is waiting for thread2 to complete its execution as it has called join on thread2.

Kickstart your Java journey with our online course on Java Programming, covering everything from basics to advanced concepts. Complete real-world coding challenges and **gain hands-on**