Multithreading Theory

Josh Wilcox (jw14g24)

March 17, 2025

Table of Contents

States of a Java thread

- `New`
  - Thread that has not yet started
- `Runnable`
  - Thread executing in the JVM
- `Blocked`
  - The thread is blocked wiaiting for a monitor lock
- `Waiting`
  - Waiting indefinitely for another thread to perform a particular action
- `Timed_Waiting`
  - Thread is waiting for another thread to perform a particular action for up to a specified time
- `Terminated`
  - The thread has terminated

Managing Threads

Set and Get names of Threads

- Use constructions to define names of our threads
  - `public Thread(String name)`
  - `public Thread(Runnable target, String name)`
- Normally, we set threads' names before they execute, but it is allowed to change aferwards:
  - `setName(String name)`
- `getName()` returns the name of a thread as a string

Checking whether threads have started

- We can use `isAlive()` to check whether a thread has been started

```java
public class ThreadExample extends Thread {
    public void run() {
        System.out.println("Thread is running...");
    }

    public static void main(String[] args) {
        ThreadExample t1 = new ThreadExample();
        System.out.println("Thread started: " + t1.isAlive());
        t1.start();
        System.out.println("Thread started: " + t1.isAlive());
    }
}
```

Wait for threads to finish

- We can use `join()` to wait for a certain thread to finish before continuing

```java
public class ThreadExample extends Thread {
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println(i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }

    public static void main(String[] args) {
        ThreadExample t1 = new ThreadExample();
        t1.start();
        try {
            t1.join();
        } catch (InterruptedException e) {
            System.out.println(e);
        }
        System.out.println("Thread has finished executing.");
    }
}
```

Daemon Thread and User Thread

- Java offers two types of threads: user threads and daemon threads
- User threads are **high-priority**
  - JVM will wait for any user thread to complete its task before terminating it
- Daemon threads are
  - **Low Priority**
  - Only role is to **provide services to user threads**
  - Will not prevent the JVM from exiting (JVM quits only when all User threads have finished)
  - New threads created inside a Daemon thread are also daemon threads
- We use the `setDaemon(boolean on)` to set a thread to a Daemon thread

```
1  public class DaemonThreadExample implements Runnable {
2      public void run() {
3          if (Thread.currentThread().isDaemon()) {
4              System.out.println("Daemon thread is running...");
5          } else {
6              System.out.println("User thread is running...");
7          }
8      }
9
10     public static void main(String[] args) {
11         DaemonThreadExample runnable = new DaemonThreadExample();
12
13         Thread userThread = new Thread(runnable);
14         Thread daemonThread = new Thread(runnable);
15         daemonThread.setDaemon(true);
16
17         userThread.start();
18         daemonThread.start();
19     }
20 }
```

Lifecycle of a thread

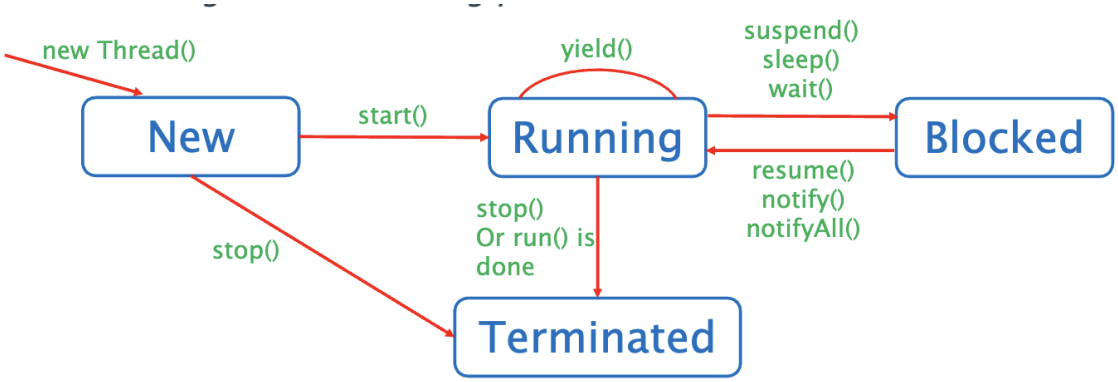

Figure: Life Cycle of a Thread

- `New`
  - Thread that has not yet started
  - Created using `new Thread()`
- `Runnable`
  - Thread executing in the JVM
  - Transitioned using `start()`
- `Blocked`
  - The thread is blocked waiting for a monitor lock
  - Can occur due to `suspend()`, `sleep()`, or `wait()`
- `Waiting`
  - Waiting indefinitely for another thread to perform a particular action
  - Can be resumed using `resume()`, `notify()`, or `notifyAll()`
- `Timed_Waiting`
  - Thread is waiting for another thread to perform a particular action for up to a specified time
  - Can be resumed using `yield()`
- `Terminated`
  - The thread has terminated
  - Can occur due to `stop()` or when `run()` is done

Stopping A Thread

- Using `stop()`, `suspend()` and `stop()` is deprecated
- Instead, use a "should run" boolean and a while loop in each thread like below:

```
1  public class StopThread implements Runnable {
2  private boolean flag = true;
3
4  public void run() {
5      int i = 0;
6      while (this.flag) {
7          System.out.println(Thread.currentThread().getName() + " is executing: i = " + (i++));
8      }
9  }
10
11 public void stop() {
12     this.flag = false;
13 }
14
15 public static void main(String[] args) {
16     StopThread mt = new StopThread();
17     Thread t = new Thread(mt);
18     t.start();
19
20     try {
21         Thread.sleep(500);
22     } catch (InterruptedException e) {
23         e.printStackTrace();
24     }
25
26     mt.stop();
27     System.out.println("Thread has been stopped.");
28 }
29 }
```