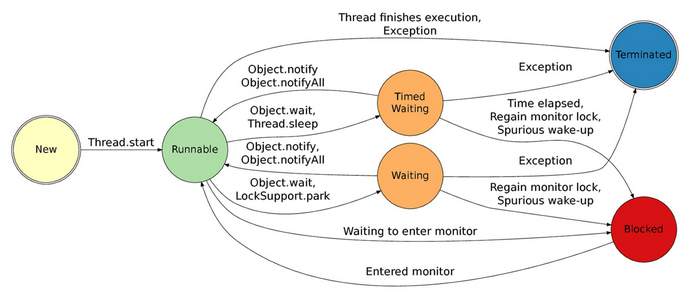
**Linux**

cat CPNFx.dat | awk -F \| '{if($3~/^200/) print $0}'

**States of Thread**

NEW/RUNNABLE/BLOCKED/WAITING/TIMED\_WAITING/TERMINATED



**How to create a thread**

1. **Implements Runnable**

**static** **class** CreationImplementsRunnable **implements** Runnable {

**public** **void** run() {

System.***out***.println(Thread.*currentThread*().getName());

}

}

Thread creationImplementRunnable = **new** Thread(**new** CreationImplementsRunnable(), "Thread-creationImplementRunnable");

creationImplementRunnable.start();

1. **Extends Thread**

**static** **class** CreationExtendsThread **extends** Thread {

**public** CreationExtendsThread(String name) {

**super**(name);

}

@Override

**public** **void** run() {

System.***out***.println(Thread.*currentThread*().getName());

}

}

Thread creationExtendsThread = **new** CreationExtendsThread("Thread-creationExtendsThread");

creationExtendsThread.start();

1. **FutureTask is the implemention of Runnable, so, below is somewhat like #1**

**static** **class** CreationImplementsCallable **implements** Callable<String> {

**public** String call() **throws** Exception {

System.***out***.println(Thread.*currentThread*().getName());

**return** "return from CreationImplementsCallable";

}

}

Thread creationImplementsCallable1 = **new** Thread(**new** FutureTask<String>(**new** CreationImplementsCallable()));

creationImplementsCallable1.start();

1. **ExecutorService.submit(Callable), can’t get the return value**

**static** **class** CreationImplementsCallable **implements** Callable<String> {

**public** String call() **throws** Exception {

System.***out***.println(Thread.*currentThread*().getName());

**return** "return from CreationImplementsCallable";

}

}

ExecutorService executorService1 = Executors.*newFixedThreadPool*(1);

executorService1.submit(**new** CreationImplementsCallable());

executorService1.shutdown();

1. **ExecutorService.submit(FutureTask), can get the return value**

**static** **class** CreationImplementsCallable **implements** Callable<String> {

**public** String call() **throws** Exception {

System.***out***.println(Thread.*currentThread*().getName());

**return** "return from CreationImplementsCallable";

}

}

ExecutorService executorService2 = Executors.*newFixedThreadPool*(1);

FutureTask<String> task = **new** FutureTask<String>(**new** CreationImplementsCallable());

executorService2.submit(task);

executorService2.shutdown();

**try** {

System.***out***.println(task.get());

} **catch** (InterruptedException e) {

e.printStackTrace();

} **catch** (ExecutionException e) {

e.printStackTrace();

}

Notify

although notify() wakes up one of the waiting threads, the first thing that that thread needs to do is re-acquire the lock that our thread is currently holding. So **after calling notify(), we should exit the synchronized block as quickly as possible**. If we do something like this:

public void returnConnection(Connection conn) {

synchronized (connections) {

connections.add(conn);

connections.notify();

// bad: woken thread can't start until we

// come out of synchronized block!

**updateStatistics(conn);**

}

}

then the woken thread won't be able to proceed until our call to updateStatistics() returns.

wait

Calling wait() automatically releases the **lock of the object you are waiting on**, but does **not release locks on other objects**! So in the following case, the caller will still hold the lock to object1 during the wait:

synchronized (object1) {

synchronized (object2) {

object2.wait();

}

}

Calling notify() does not "transfer control" to notified threads immediately: it merely marks them as "runnable". A notified thread will not be able to run until the following things happen: (1) the notifying thread releases its lock on the object being notified; (2) the thread scheduler next shcedules the notified thread. (On some systems such as Windows, this will happen fairly quickly, because threads are given a temporary priority boost when woken from a wait state, but it generally won't happen until the next interrupt.)