Threads

SIMANTA MITRA

Learning Objectives

At the end of this lesson, students should

- know what's a thread
- know why threads are needed
- know how to write simple Java programs using threads
- learn a few basic things about threads

MOTIVATION FOR THREADS

MOTIVATION FOR THREADS

- One of the major concepts we learned in programming is that program execution proceeds step by step. Until a step is completed, the next step cannot be started.

- Consider the following situations

situation 1: Printing a large file

- You are editing a large file. You ask the document editor to print it for you. While it is being printed, the editor does not allow you to continue editing your file (as the printing needs to be completed first).

- Nowadays, document editors do print jobs in the background - allowing you to continue editing.

situation 2: loading a web page

- You just navigated to a web page and there are many pictures and videos on the web page.

- You have a slow internet connection and have to wait a long while before you can start reading the information on the web page.

- Most good web pages will load text first and allow to start reading while it continues to upload the pictures in the background.

Threads

- In both the previous situations, we want more than one set of tasks to be going on at the same time (i.e. concurrently).
- Typically, there are many programs on a computer and these can be executed by the computer at the same time. Executing programs are called PROCESSES.

- However, it is also possible to have separate sets of tasks being concurrently executed in the SAME PROGRAM. These are called threads.

JAVA THREADS

Threads in Java

- Java provides TWO ways of creating threads.

  - First way: extend Thread class and implement the run method

  - Second way: implement Runnable class, create Thread object, and implement the run method

```
public class SimpleThread1 extends Thread {

        String name;

        public SimpleThread1(String s) { name = s; }

        public void run() {
          for (int i = 1; i < 1000; i++) {
                System.out.println("This is thread " + name);
          }
        }

        public static void main(String[] args) {

                Thread firstThread = new SimpleThread1("T1");
                Thread secondThread = new SimpleThread1("T2");

                firstThread.start();
                secondThread.start();
        }
}
```

```
public class SimpleThread2 implements Runnable {

        String name;

        public SimpleThread2(String s) { name = s; }

        public void run() {
         for (int i = 1; i < 1000; i++) {
                    System.out.println("This is thread " + name);
          }
        }

        public static void main(String[] args) {

                Thread firstThread = new Thread( new SimpleThread2("T1"));
                Thread secondThread = new Thread( new SimpleThread2("T2"));

                firstThread.start();
                secondThread.start();
        }

}
```

Notes

- In the previous examples, there were actually THREE user threads: the main thread (which stops when the main method is done), the firstThread, and the secondThread.

- When you run the example, you will see that the results from the threads are interleaved. Each thread prints when it gets the timeslice - but stops when it is put back in the ready queue by the scheduler.

WHAT TO KNOW ABOUT THREADS

What to know about threads

- Thread lifecycle, Thread states
- Thread methods
- Issues in data sharing (synchronized statement)
- Synchronization between threads (wait/ notify)
- Problems in using threading
- We will briefly go over thread states and methods but will skip the other parts.

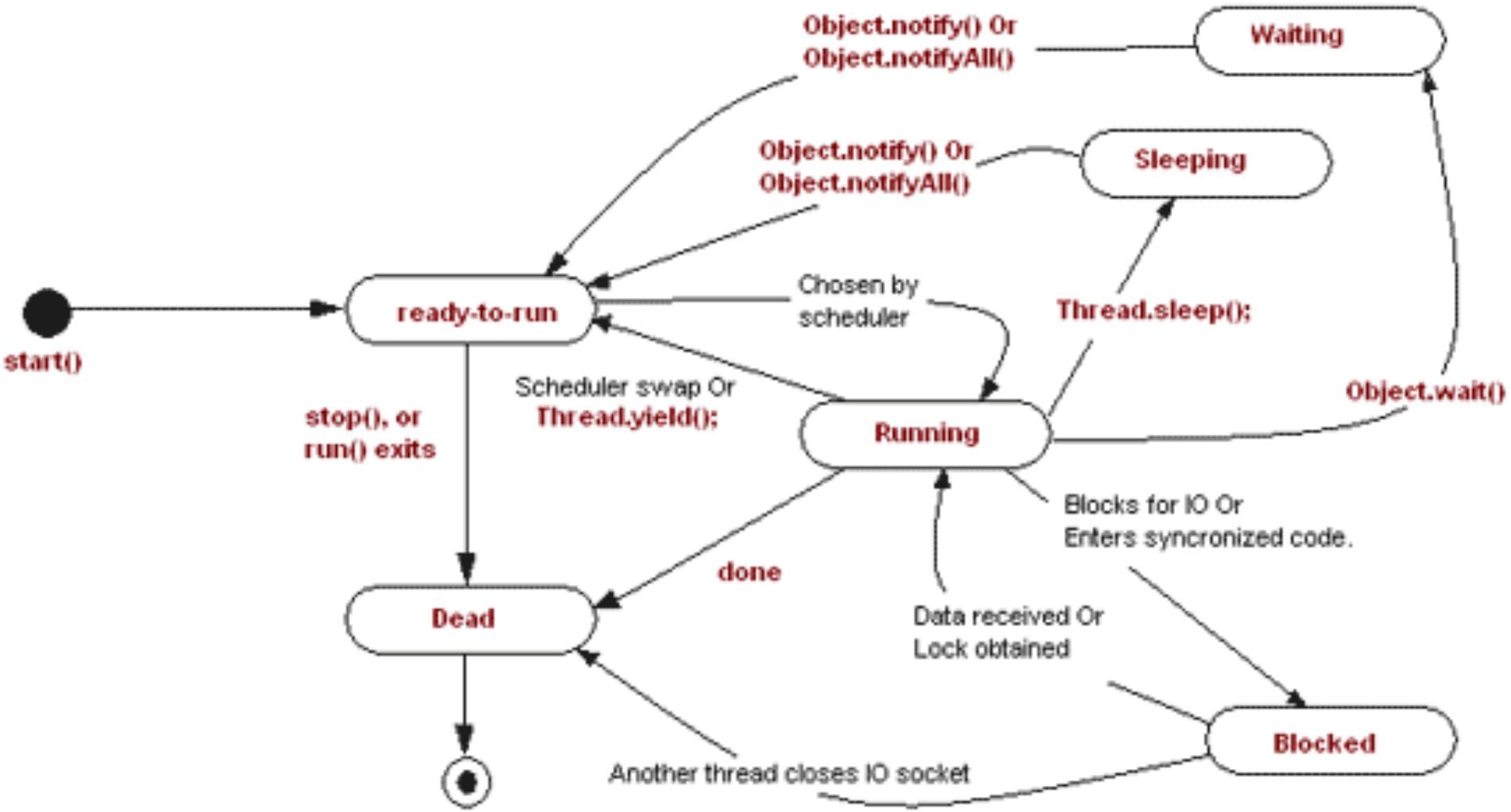
Thread States

- created: When we do a new Thread(), the thread has been created.
- ready-to-run: When we invoke the start method on the thread object, the scheduler puts the thread in a "ready" Queue. The scheduler lets the first thread on this queue to run for some time, and then the next one on the queue and so on.
- running: The run method of the thread object is started. When the thread's timeslice is over, the thread is put back in the ready queue. It can continue with the run method, when it is allowed to execute again by the scheduler.
- dead: When the run method is completed, the thread is considered to be done executing and in the dead state. Some threads never die (as their run methods are never-ending loops).

Thread states

- For now, we will ignore the sleep, waiting, and blocked states.

Thread Lifecycle/methods



Thread methods

- Some useful thread methods for now are:
  - start(): this puts the thread in the ready queue.
  - yield(): this allows the thread to give up its time slice and get put back at the end of the ready queue.
  - sleep(x): allows the thread to sleep for x milliseconds before being woken up to run again.

Advanced Threads

- There is a lot to know about threads. See:

Good resource: http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html

COMS430 --- good class to take (about concurrency)

Summary

To summarize

Good resource: http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html

You can either extend Thread or implement Runnable to create Java Threads.

Threads can be started by "start" method.

Then, their run method starts executing. Threads get switched out by the scheduler.

Multiple threads can be running at the same time.