Threads

CS151

Chris Pollett

Nov. 28, 2005.

Outline

- Thread Basics
- Thread Safety

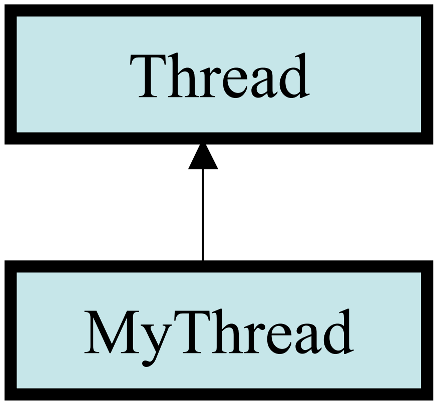
Thread Basics I

- A *thread* is a single sequential flow of control within a program.
- Java supports *multithreaded* (aka *concurrent*) programming.
- That is, the run-time on top of the OS can make it appear as if several threads are running at the time using various kind of time-sharing.
- Threads are useful for:
  - reactive systems -- systems which continuously monitor arrays of sensors and react to control systems according to sensor readings
  - GUIs -- that would like to performs tasks but still be able to react to user input.
  - Servers -- which might need to handle several clients simultaneously
  - Computer with multiple processors.
- Threads are different from processes. Processes are managed by the OS. They take longer to create and have more limited interprocess communication.

Thread Basics II

- In Java, a thread is an instance of the class java.lang.Thread.
- In UML to denote an active object like a thread one makes the class box bold face like below.
- The run() method of Thread is a hook into which you place your concurrent content:

```
public class MyThread extends Thread
{
     public void run()
     {
        //your code
     }
}
```



- To start an instance of your class running one does: new MyThread.start();

Example Threaded Program

```
public class Counter1 extends Thread
{
	protected int count, inc, delay;
	public Counter1(int init, int inc, int delay)
	{
		this.count = init; this.inc = inc; this.delay = delay;
	}
	public void run()
	{
		while(true){
		try
		{
		   System.out.print(count + “ “);  count += inc; sleep(delay);
		}
		catch(InterruptedException e){ e.printStackTrace();}}
	}
	public static main (String[] args) {new Counter1(0,1,33).start(); new Counter1(0,-1,100).start();}
} //increasing counter should display about 3 times for every one time decreasing counter goes
```

Runnable Interface

- Sometimes one wants to use Threads but one wants to be able to extend some other class other than Thread.
- To do this one can implement the Runnable interface. To implement this interface one has to implement a public void run() method:

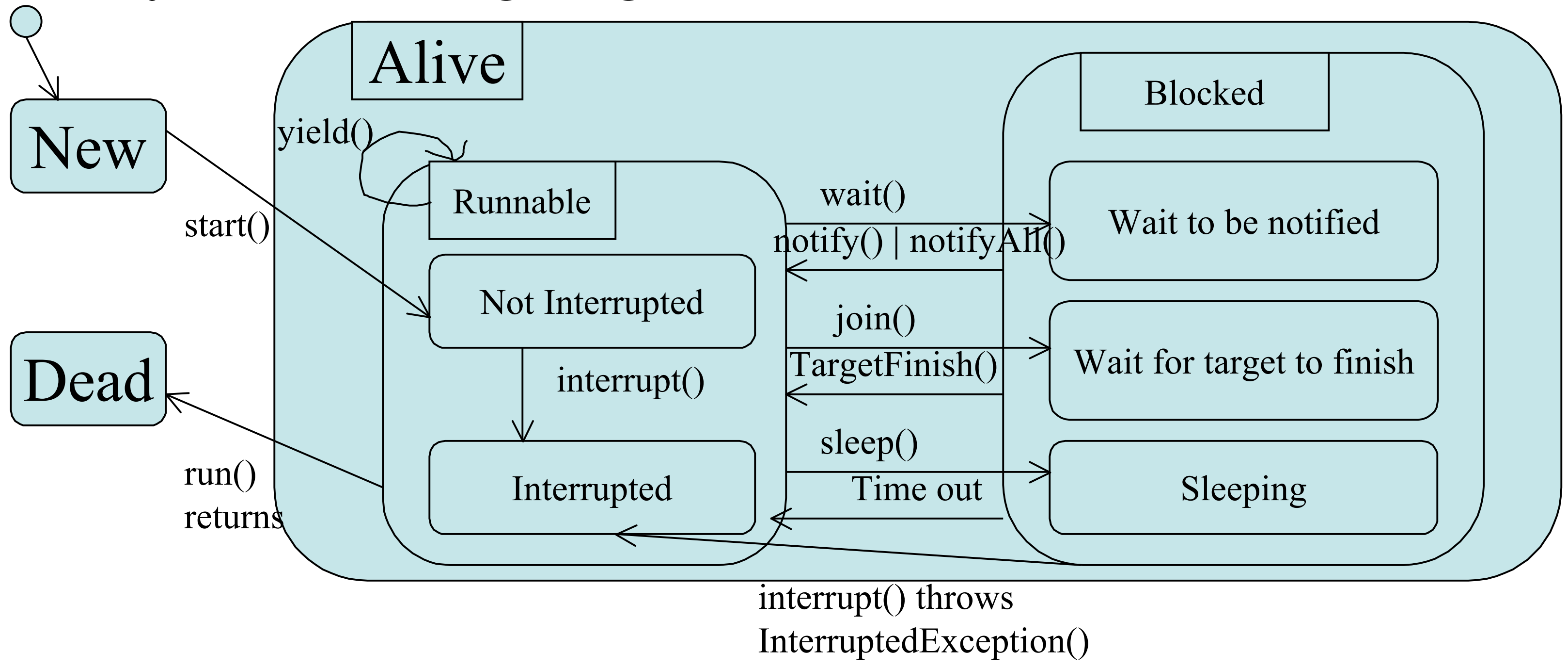
```
public class MyRunnable extends Something implements Runnable
{
    public void run()
    { // do stuff
    }
}
```

- To get a thread which uses this run method one can do:

```
new Thread(new MyRunnable()).start();
```

Controlling Threads

- The following chart illustrates the states that a thread can be in as well as the transitions between them is illustrated by the following diagram:



Methods of java.lang.Thread

- start() -- makes the thread transition from the New state (after creation) to the alive state
- sleep(t) -- from the Runnable state makes a thread enter the blocked to for t milliseconds. After the time out period is over it re-enters the Runnable state
- join() -- from the Runnable state if join() of another Thread is called, it causes the thread which made the call to go into the Blocked state and wait for other thread to finish, at which point it re-enter the Runnable state.
- yield() -- from the Runnable state leaves the thread in the Runnable state but lets another thread in the Runnable state have a chance to run
- interrupt() -- if the thread is in the Runnable state, the interupted flag is set. If the thread is in the Blocked state, it is awakened and enters the Runnable state and an interuppted exception is thrown.
- isAlive() -- returns true if the thread is in the Alive state
- isInterrupted() - returns true if the interrupted flag is set.

Thread Priority and Scheduling

- The JVM implements a rather simple scheduling strategy to determine which of the Runnable threads should be running.
- Each thread has an integer priority attribute which is assigned when the thread is created. By default, this priority is the priority of the thread that created the new thread.
- The JVM has a queue from each priority and selects a Runnable thread from the queue with the highest priority (how the thread is selected from the queue is arbitrary)
- Threads of higher priority when the become Runnable will preempt a thread of lower priority. That is, if a thread of higher priority becomes Runnable, the current thread will stop running so it can run. the current thread though stays Runnable.

Changing whose Running

- The thread that is currently running relinquishes control when one of the following occurs:
  - Yielding -- its yield() method is invoked
  - Blocking -- sleep(), join(), or wait()
  - Preempting -- when a thread with higher priority becomes available
  - Switching -- when its time slice expires (some OSs don’t support time-slicing).

Thread Safety

- *Safety conditions* are conditions that should hold throughout the lifetime of the program and ensure that nothing bad should happen.
- An important safety condition is the consistency of object states.
- For example, if you have two ATM transactions going on at the same time on the same account you shouldn’t be able to use this to make money that didn’t exist.
- This might be possible if they both check the current balance before either of them withdraws money. This is an example of a *race condition*.
- A thread is said to be *thread-safe* if its ensures the consistency of the states of the objects and the results of method invocations.
- To maintain the consistency of states one can try to *synchronize* who gets access to certain critical region of a code. Critical regions are regions of code only one thread should be allowed to execute at a time.

Synchronization

- In Java synchronization can be applied to a block of code or to a method:

```
class MyClass
{
        synchronized void aMethod() {/* do something */}
        /*
                This is equivalent to
                aMethod(){ synchronized(this){//do something
                }}
        */
}

or for a block

synchronized(ref)
{
  // do something
}
```

- Basically, each object has a lock and to run a synchronized method the thread must first get the lock. Locks are released when the thread lives the critical region. The lock can also be released using wait().