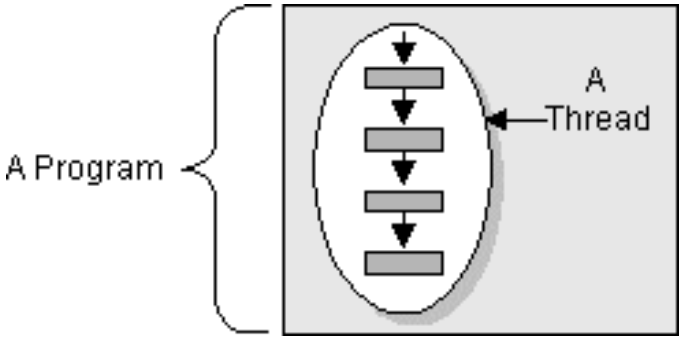
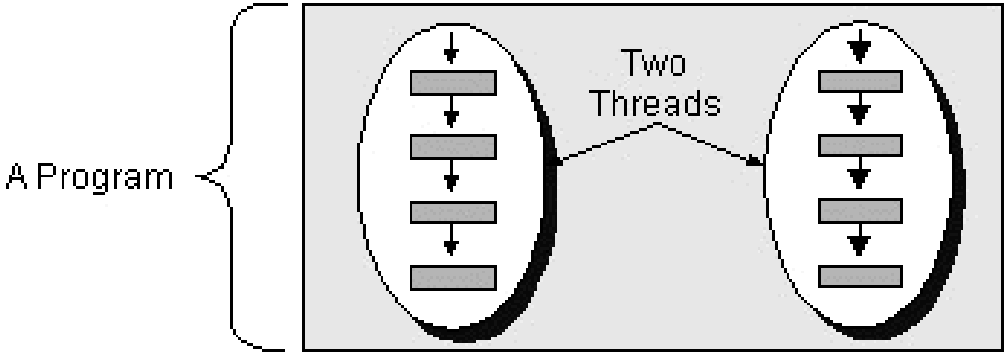
Thread : a single sequential flow of control within a program





http://java.sun.com/docs/books/tutorial/essential/threads/index.html

Thread Class

Threads in Java are represented as objects

- Commonly used methods:
  - **run**() - the code that the thread executes
  - **start**() - the method called to make a thread "eligible" for running.
  - **isAlive**() - determines if thread is "alive" (more later)
  - **join**() - waits for the completion of a thread
  - **getName**() / **setName**(str)

- Static methods:
  - **sleep**(milliseconds) - causes the current thread to sleep or delay for the given time period.
  - **currentThread**() - returns the Thread object of the currently active thread.

Thread Class Constructors

Thread()
Allocates a new Thread object.

Thread(String name)
Allocates a new Thread object, with a name.

Thread(Runnable target)
Allocates a new Thread object for the given target object.

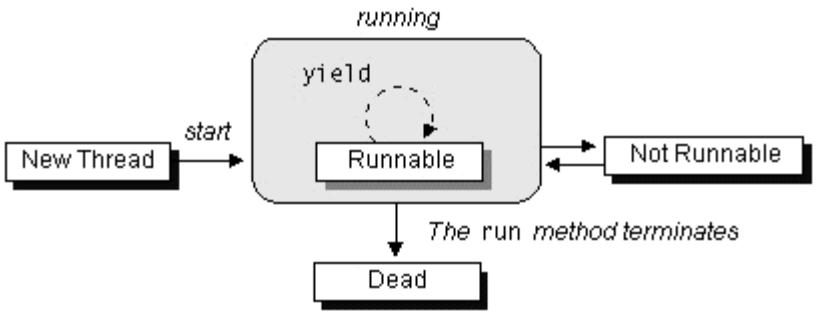
Thread(Runnable target, String name)
Allocates a new Thread object for the given target object, with a name

Runnable Interface

public void run() - method invoked by a Thread start()

Thread States

- **Ready**
- **Running**
- **Not Runnable - Waiting, Sleeping, Suspended, Blocked**
- **Dead**



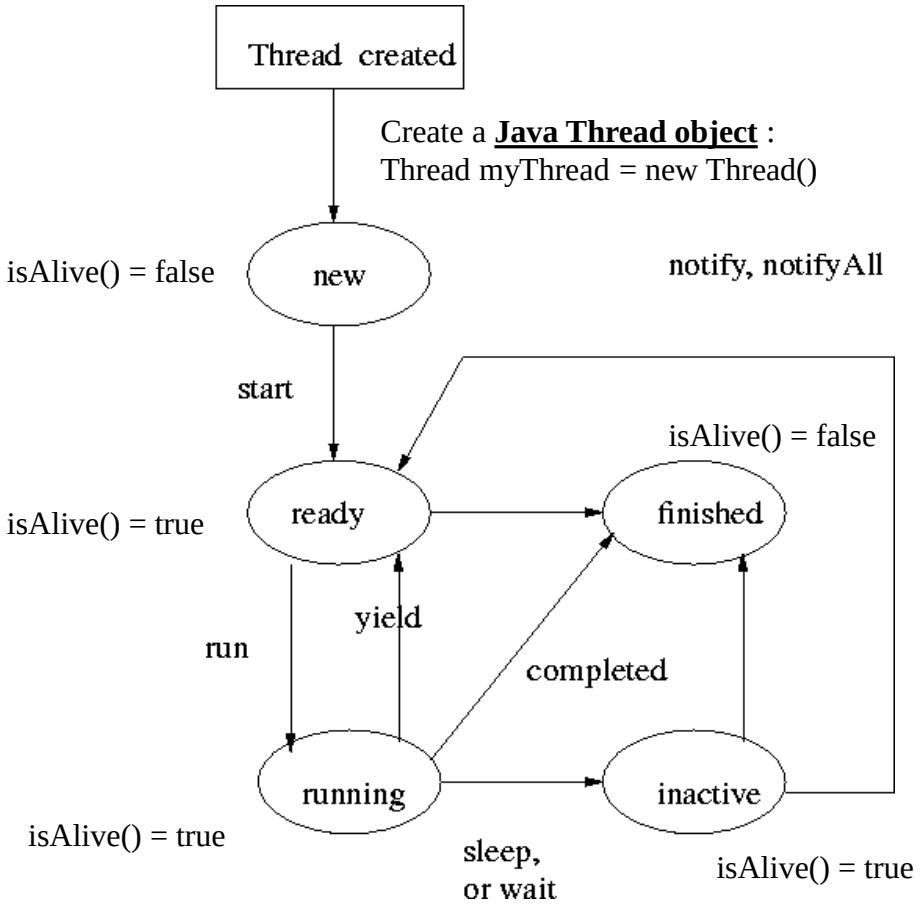
When you invoke start(), a new thread is marked ***ready*** and is placed in the Thread queue.

A thread is placed in the ***waiting*** state, or becomes Not Runnable, when one of these events occurs:

- Its sleep method is invoked.
- The thread calls the wait method to wait for a specific condition to be satisifed.
- The thread is blocking on I/O.

When the run() method terminates, the Thread **dies**. A dead Thread cannot be restarted.

State-Transition Diagram



(Calling a method which violates a state transition results in an IllegalThreadStateException)

Thread state transition criteria

A thread becomes Runnable when one of these events occurs:

- After the initial call to the Thread's start() method.
- If a thread had been put to sleep, then the specified number of milliseconds have elapsed.
- If a thread is waiting for a condition, then another object has notified the waiting thread of a change in condition by calling notify or notifyAll.
- If a thread was blocked on I/O, then the I/O has completed.

A thread becomes Not Runnable when one of these events occurs:

- Its sleep method is invoked.
- The thread calls the wait method to wait for a specific condition to be satisifed.
- The thread is blocking on I/O.

A thread dies when:

- Its run method completes.
- Threads typically arrange for their own death by executing the run() method with some loop condition.
- A dead thread cannot be restarted.

Thread Scheduling & Priority

•Most computers have only one CPU, so threads must share the CPU with other threads. The execution of multiple threads on a single CPU, in some order, is called ***scheduling***. The Java runtime supports a very simple, deterministic scheduling algorithm known as fixed priority scheduling.

•Each Java thread is given a numeric ***priority*** between MIN_PRIORITY and MAX_PRIORITY (constants defined in the Thread class). At any given time, when multiple threads are ready to be executed, the thread with the highest priority is chosen for execution. Only when that thread stops, or is suspended for some reason, will a lower priority thread start executing.

•Scheduling of the CPU is fully ***preemptive***. If a thread with a higher priority than the currently executing thread needs to execute, the higher priority thread is immediately scheduled.

•The Java runtime will not preempt the currently running thread for another thread of the same priority. In other words, the Java runtime does not ***time-slice***. However, the system implementation of threads underlying the Java Thread class may support time-slicing. Do not write code that relies on time-slicing.

•In addition, a given thread may, at any time, give up its right to execute by calling the yield method. Threads can only yield the CPU to other threads of the same priority--attempts to yield to a lower priority thread are ignored.

•When all the runnable threads in the system have the same priority, the scheduler chooses the next thread to run in a simple, non-preemptive, round-robin scheduling order.

http://java.sun.com/docs/books/tutorial/essential/threads/priority.html

“Selfish” run-method -

```
public int tick = 1;

public void run() {
    while (tick < 10000000)
        tick++;
}
```


CPU greedy!

“Polite” run-method -

```
public int tick = 1;

public void run() {
    while (tick < 10000000)
        tick++;
        if( (tick % 5000) == 0 ) {
           yield();
        }
}
```


Allows waiting threads to run

Synchronizing Threads

race condition the condition that arises from multiple, asynchronously executing threads trying to access a single object at the same time and getting the wrong result.

critical sections code segments within a program that access the same object from separate, concurrent threads

`synchronized` Java keyword used to protect a critical section which can be a block or a method. The Java VM then associates a ***lock*** with every access to the object which has synchronized code.

Synchronized method

```
public synchronized void setFlag() {
   myFlag = true;
return;
}
```

This locks all synchronized access to the entire object of which setFlag() is a method of.

Synchronized block

```
public void setElement(Object[] anArray,
                       int index ){
    synchronized( anArray ){
          anArray[index] = true;
    } // end block
}
```

This locks all synchronized access to the anArray object only.

wait, notify, notifyAll

wait() - waits for a condition to occur. This is a method of the Object class and must be called from within a synchronized method or block.

When wait is called:

- the current thread is suspended or placed in the wait queue (non-runnable state)
- the synchronization lock for the target object is released, but all other locks held by the thread are retained.

notify() - notifies a thread waiting for a condition that the condition has occurred. This is a method of the Object class and must be called from within a synchronized method or block.

When notify() is called:

- an arbitrary thread waiting for the condition attempts to regain the synchronization lock it relinquished as a result of its wait() call.
- After obtaining the lock it resumes execution at the point of its wait()

notifyAll() - works the same as notify except that the steps occur for all threads waiting in the wait queue for the traget object.

(Concurrent Programming in Java - Doug Lea)

Other Thread Topics

Thread Groups:

- all threads belong to a thread group by default (system)
- new groups created using ThreadGroup class, allows setting priority, collection of threads

Thread methods:

- isAlive() –boolean method indicates if thread is alive or dead
- join() – typically used by a parent thread to wait until its child thread has completed.

Synchronizing class methods :

- lock for synchronized class methods, independent of object locks for instances of that class.

```
synchronized static void someClassMethod() { … }
```

Daemon Threads:

- provides a general service in the background, but is not part of the program.
- the Java VM terminates when all non-daemon threads have terminated.
- there are isDaemon() and setDaemon() methods in Thread.

Synchronized Collections:

- remember that all Collection framework implementations are not synchronized
- use the Collections class methods to synchronize the implementation of your choice:

```
List list = Collections.synchronizedList( new ArrayList() );
```