JAVA

Agenda

- Thread
- Collection

Thread

Thread

- Program in execution is called process

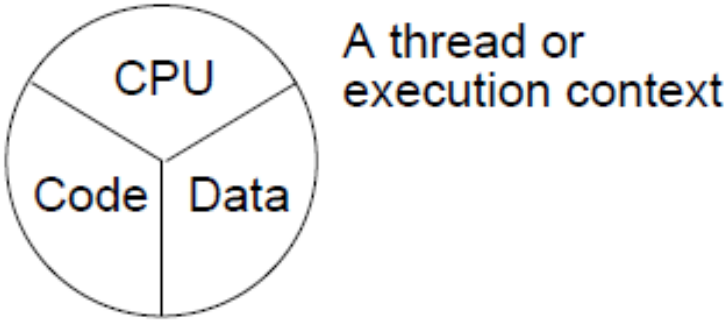
- Thread is :
  - a light weight sub-process.
  - a single sequential flow of control within a program
  - is the path followed when executing a program

What are threads?

Threads are a virtual CPU.

The three parts of a thread are:

- CPU
- Code
- Data



Selecting a Way to Create Threads

1. Implement Runnable
2. Extend Thread

Creating the Thread

- The Runnable interface defines a single method, run

- The Runnable object is passed to the Thread constructor, as in the HelloRunnable example:

```
public class HelloRunnable implements Runnable
{
      public void run()
     {
     System.out.println("Hello from a thread!");
     }
      public static void main(String args[])
      {
      (new Thread(new HelloRunnable())).start();
     }
}
```

Subclass Thread

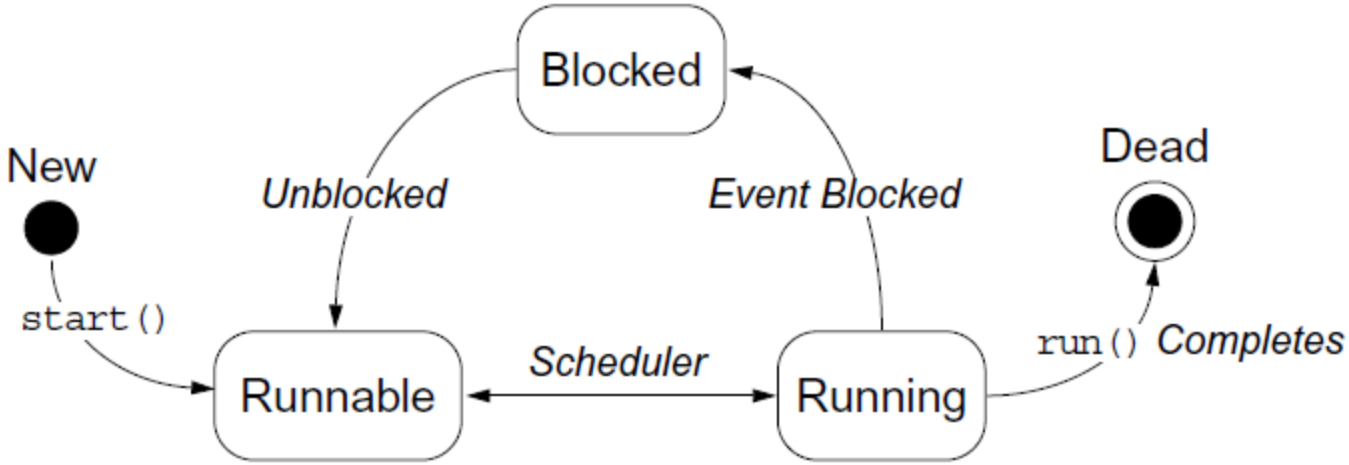
- An application can subclass Thread, providing its own implementation of run, as in the HelloThread example:

```
public class HelloThread extends Thread
  {
   public void run()
      { System.out.println("Hello from a thread!"); }
   public static void main(String args[])
     { (new HelloThread()).start(); }
  }
```

Stages - thread

- **New:** A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.
- **Runnable:** After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting:** Sometimes a thread transitions to the waiting state while the thread waits for another thread to perform a task.A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed waiting:** A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated:** A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Thread Scheduling



SN	Methods with Description
1	**public void start()** Starts the thread in a separate path of execution, then invokes the run() method on this Thread object.
2	**public void run()** If this Thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object.
3	**public final void setName(String name)** Changes the name of the Thread object. There is also a getName() method for retrieving the name.
4	**public final void setPriority(int priority)** Sets the priority of this Thread object. The possible values are between 1 and 10.
5	**public final void setDaemon(boolean on)** A parameter of true denotes this Thread as a daemon thread.
6	**public final void join(long millisec)** The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.
7	**public void interrupt()** Interrupts this thread, causing it to continue execution if it was blocked for any reason.
8	**public final boolean isAlive()** Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.

Basic Control of Threads

- Test threads:
  - isAlive()
- Access thread priority:
  - getPriority()
  - setPriority()
- Put threads on hold:
  - Thread.sleep() // static method
  - join()
  - Thread.yield() // static method

Pausing Execution with Sleep

```
public class Runner implements Runnable {
 public void run() {
 while (true) {
 // do lots of interesting stuff
 // ...
 // Give other threads a chance
 try {
 Thread.sleep(10);
 } catch (InterruptedException e) {
 // This thread's sleep was interrupted
 // by another thread
 }
 }
 }
 }
```

Multi Threading

Multithreaded programming has these characteristics:

- Multiple threads are from one Runnable instance.
- Threads share the same data and code.

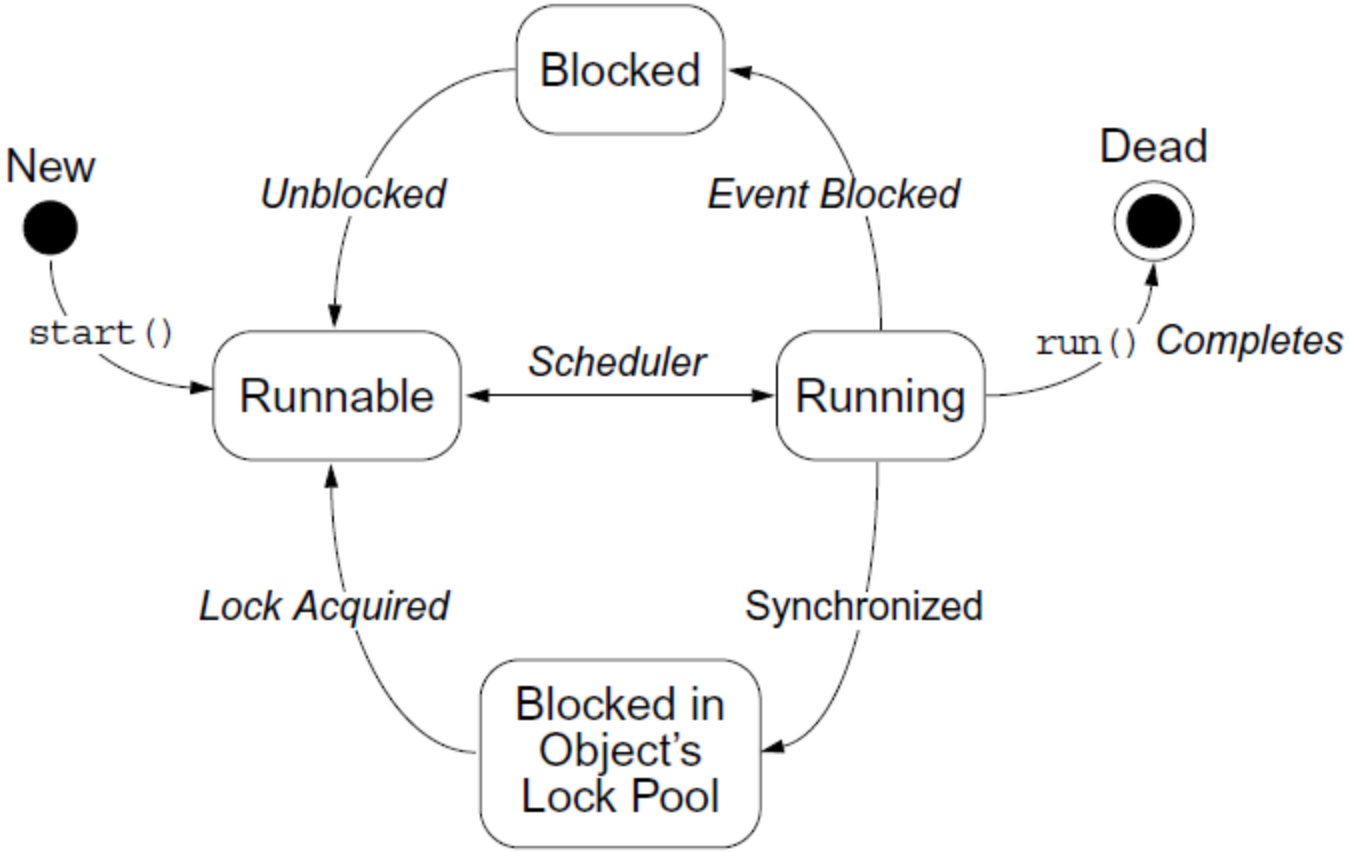
- For example:

```
Thread t1 = new Thread(r);
Thread t2 = new Thread(r);
```

Using synchronized

```
public void push(char c)
{
synchronized(this)
 {
// The push method code
}
}
```

```
public synchronized void push(char c)
{
// The push method code
}
```


Blocked
New
Unblocked
Event Blocked
Dead
start()
Runnable
Scheduler
Running
run() Completes
Lock Acquired
Synchronized
Blocked in Object's Lock Pool

Thread Interaction – wait and notify

Scenario:

Consider yourself and a cab driver as two threads.

• The problem:

How do you determine when you are at your destination?

• The solution:

• You notify the cab driver of your destination and relax.

• The driver drives and notifies you upon arrival at your destination.

State Diagram – wait and notify