Threads in Java

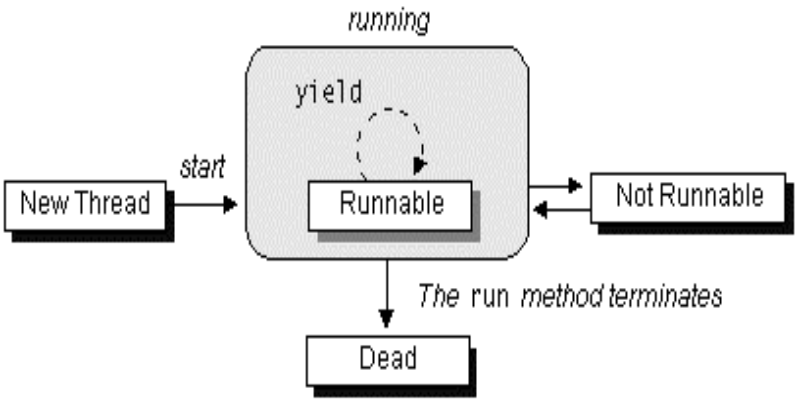
Basic Mechanisms

CS-680 Michael Weiss

Basic Thread Mechanisms

- Thread creation
- Synchronized statements
- Coordination: wait, notify, notifyAll
- Coordination: join
- Interrupts
- Miscellaneous:
  - sleep
  - isAlive
  - currentThread
  - yield
  - setPriority
  - volatile

Lifecyle of a Thread

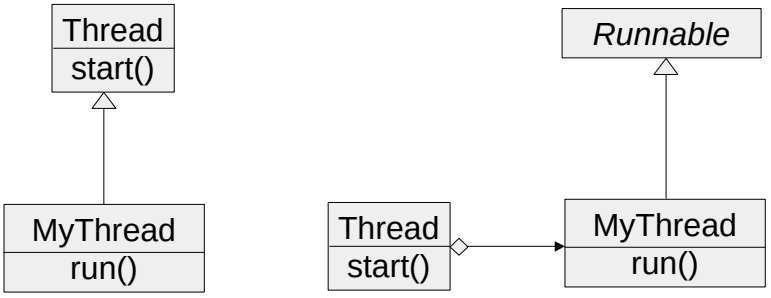


java.util.concurrent

- Based on Doug Lea's code, developed over many years.
- Part of standard Java starting with Java 1.5.
- In "real life": ***look at java.util.concurrent first***, before rolling your own.
  - Would you implement your own hashtable, or use java.util.HashMap?

Creating Threads

Two mechanisms



Creating Threads

```
class MyThread extends Thread {
    public void run() {
        //do my thing
    }
}

class Driver {
    public void static main() {
        MyThread myThread =
            new MyThread();
        myThread.start();
    }
}
```

```
class MyThread implements
    Runnable {
        public void run() {
            //do my thing
        }
}
class Driver {
    public void static main() {
        MyThread myThread =
            new MyThread();
        Thread thread =
            new Thread(myThread);
        thread.start();
    }
}
```

Synchronized Methods

```
class X {
   int i=0;
   synchronized void inc() {
      i++;
      System.out.println(i);
   }
   synchronized void dec() {
      i--;
      System.out.println(i);
   }
}
```

Thread acquires lock on instance of class X before it can enter inc()

Thread blocks until it can acquire lock

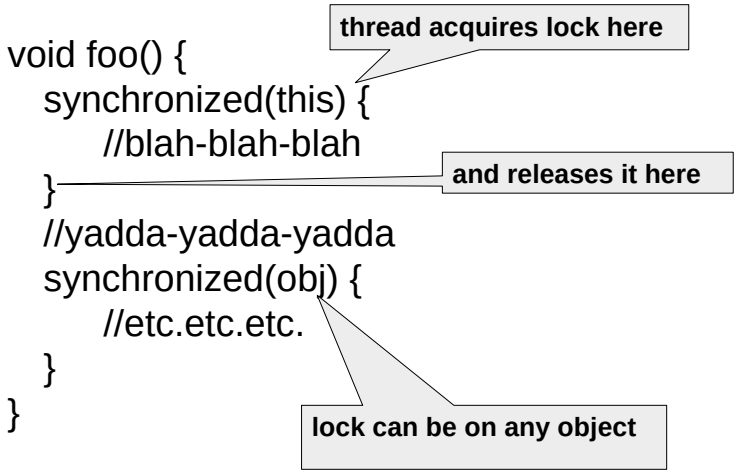
Thread releases lock on exit

Without locks, could have:

Locks

- **Object** owns lock; thread acquires lock temporarily
- One lock per object
- Locks are re-entrant: nested synchronized calls no problem
- Lock holds set of blocked threads
- When lock is released, some blocked thread is awakened
- Static synchronized methods: lock is held by Class object (e.g., Foo.class)

Synchronized Statements



Wait, Notify, NotifyAll

- obj.wait():
  1. thread must hold lock of obj, otherwise IllegalMonitorStateException
  2. thread releases lock and is added to the ***wait set*** of obj; thread becomes blocked
  3. thread keeps all other locks
- obj.notify():
  1. thread must hold lock of obj, like wait()
  2. some other thread in wait set of obj is awakened; other thread now blocks on lock of obj
  3. when the notifying thread releases lock, other thread has chance to proceed
- obj.notifyAll():
  - Just like notify, except all threads in wait set are awakened

Wait, Notify standard usage pattern

```
synchronized(this) {
  while (!condition) {
    try {
        wait();
    } catch (InterruptedException e) {
    }
    // do stuff
  }
}
```

```
synchronized(this) {
  try {
    // do stuff
    condition = true;
    notify();
  } catch (InterruptedException e) {
  }
}
```


join

otherThread.join():

1. current thread is suspended
2. when otherThread dies, the thread that did the join is resumed

Interrupts

thread.interrupt()

- if thread is blocked, thread is wakened with an InterruptedException and isInterrupted()==false
- if thread is runnable, interrupted status is set to true: isInterrupted()==true

Miscellaneous

- thread.sleep(int milliseconds)
  - causes thread to sleep for specified time
- thread.isAlive()
  - true if thread is runnable or blocked
  - false if thread is new or dead
- Thread.currentThread()
  - static method returning the current thread
- yield; setPriority:
  - hints to JVM runtime
  - no guarantees
- volatile keyword
  - tells compiler other threads may be changing a field

Concepts and Buzzwords

- Safety properties, thread-safety
  - Race conditions, critical regions
  - Mutual Exclusion (mutex)
- Liveness properties and problems
  - Starvation, indefinite postponement
  - Dormancy
  - Deadlock
- Thread coordination
  - Producer-consumer
  - Barrier synchronization
  - Futures