Plan for Today

wait and notify
Concurrency on Mars!

Project Time

wait and notify

Wait, Wait Don’t Notify Me!

synchronized(obj) { code }

Provides mutual exclusion: code inside synchronized can only run when lock of obj is held

obj.wait()

Gives up lock on obj; puts current thread in waiting set for obj

obj.notify(), obj.notifyAll()

Don’t give up lock; selects one (notify) or all (notifyAll) threads in waiting set for obj and wakes them up (to be scheduled)

Methods inherited from class java.lang.Object

public final void wait() throws InterruptedException

Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object. In other words, this method behaves exactly as if it simply performs the call wait(0).

public final void wait(long timeout) throws InterruptedException

Causes the current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

public final void notify()

Wakes up a single thread that is waiting on this object’s monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. A thread waits on an object’s monitor by calling one of the wait methods.

The awakened thread will not be able to proceed until the current thread relinquishes the lock on this object

Synchronizing

Thread A

synchronized (o)

Thread B

synchronized (o) {

wait()

// end synchronized

司 each thread A, B, C

Thread A

synchronized (o)

Thread B

synchronized (o) {

wait()

Thread C

synchronized (o) {

wait()

Thread D

synchronized (o) {

wait()
class IncThread extends Thread {
    private Counter c;
    public IncThread (Counter p_c) { c = p_c; }
    public void run () {
        while (true) {
            synchronized (c) {
                c.increment ();
                System.err.println ("Running inc thread: "+ currentThread () + ...);
                c.notify ();
            }
        }
    }
}

class DecThread extends Thread {
    ...
    public void run () {
        while (true) {
            synchronized (c) {
                while (c.getValue () <= 0) {
                    try { c.wait (); } catch (InterruptedException e) { ; }
                }
                c.decrement ();
                System.err.println ("Running dec thread: "+ ...);
            }
        }
    }
}

Counter c = new Counter ();
IncThread ithread = new IncThread (c);
DecThread dthread = new DecThread (c);
ithread.setPriority (Thread.NORM_PRIORITY);
ithread.start ();
dthread.setPriority (Thread.MAX_PRIORITY);
dthread.start ();

Running inc thread: Thread[Thread-0,5,main] / Value: 1
Running dec thread: Thread[Thread-1,10,main] / Value: 0
Running inc thread: Thread[Thread-0,5,main] / Value: 1
Running dec thread: Thread[Thread-1,10,main] / Value: 0
Running inc thread: Thread[Thread-0,5,main] / Value: 1
Running dec thread: Thread[Thread-1,10,main] / Value: 0

Priorities
• In general, threads with higher priorities will be scheduled preferentially.
• There are no guarantees: up to Java scheduler
  Thread class:
      void setPriority (int newPriority)
          // MODIFIES: this
          // EFFECTS: Changes the priority of this
          //     thread to newPriority.

Priorities, Priorities
ithread.setPriority (Thread.NORM_PRIORITY);
ithread.start ();
dthread.setPriority (Thread.MIN_PRIORITY);
dthread.start ();

Why deprecate stop?
• What should happen to all the locks a thread owns when it is stopped?
• What if an invariant is temporarily broken in a method?
Suspending Threads

```java
public final void suspend()
```

Suspends this thread. If the thread is alive, it is suspended and makes no further progress unless and until it is resumed.

**Deprecated.** This method has been deprecated, as it is inherently deadlock-prone. If the target thread holds a lock on the monitor protecting a critical system resource when it is suspended, no thread can access this resource until the target thread is resumed. If the thread that would resume the target thread attempts to lock this monitor prior to calling resume, deadlock results. Such deadlocks typically manifest themselves as “frozen” processes.

Can’t stop, can’t suspend, what can you do?

```java
public void interrupt()
```

Interrupts this thread.

If this thread is blocked in an invocation of the `wait()`, `wait(long)`, or `wait(long, int)` methods of the `Object` class, or of the `join()`, `join(long)`, `join(long, int)`, `sleep(long)`, or `sleep(long, int)` methods of this class, then its interrupt status will be cleared and it will receive an `InterruptedException`.

... If none of the previous conditions hold then this thread’s interrupt status will be set.

Being Interrupted

```java
public boolean isInterrupted()
```

**MODIFIES:** nothing  
**EFFECTS:** Returns true iff this thread has been interrupted.

Counter c = new Counter();
IncThread ithread = new IncThread(c);
DecThread dthread = new DecThread(c);
ithread.setPriority(Thread.NORM_PRIORITY);
ithread.start();
dthread.setPriority(Thread.MAX_PRIORITY);
dthread.start();
dthread.interrupt();

Interrupts are just “polite” requests! The thread can ignore it and keep going...

Mars Pathfinder

Landed on Mars
July 4, 1997


Mary Beth Murrill, a spokeswoman for NASA's Jet Propulsion Laboratory, said transmission of the panoramic shot took “a lot of processing power.” She likened the data overload to what happens with a personal computer “when we ask it to do too many things at once.”

The project manager, Brian Muirhead, said that to prevent a recurrence, controllers would schedule activities one after another, instead of at the same time. It was the second time the Pathfinder’s computer had reset itself while trying to carry out several activities at once.

In response, controllers reprogrammed the computer over the weekend to slow down the rate of activities and avoid another reset. But today, about an hour into a two-hour transmission session, it happened again.
Priority-Based Scheduling

Scheduler ensures that the highest priority task that can run is always running.

Lower priority tasks run only when no higher priority task can run.

Standard JavaVM scheduler does not do this, but many operating systems for embedded systems do including the vxWorks used on the Pathfinder.

What could go wrong with priority-based scheduling?

Priority Inversion

Priority Inversion on Mars

Meterological data task (low priority) → Bus Management Task (high priority) → Data collection task (medium priority)

For details, see Glenn Reeves account:
http://research.microsoft.com/~mbj/Mars_Pathfinder/Authoritative_Account.html

Solutions?

Priority Inheritance

If a low priority task holds a resource needed by a high priority task, the low priority task temporarily inherits the high task’s priority.

Priority Ceilings

Associate minimum priorities with resources: only a high priority task can acquire the lock on an important resource.

Charge

- Computers are single-threaded (or 2/4/8+-threaded) machines that provide their owner the illusion of infinite threads.
- Brains are massively multi-threaded machines that provide their owner with the illusion of a single thread.

```
Thread work = new Thread (project);
work.setPriority (Thread.MAX_PRIORITY);
work.start ();
```