Not Hogging the CPU

You can call sleep(int msec).
  Sleep approximately that long.

You can call yield().
  Gives up the CPU.
Daemon Threads

- Daemon threads run in the background of Java.
- Use them for general services.
  - e.g. the garbage collector.
- Daemon threads are automatically terminated when the program exits.

Methods
- isDaemon()
- setDaemon(boolean s)

- Threads created by daemons are automatically daemons.
Shared Resources and Synchronization Problems

- Any object, variable, method, or other resource accessed by more than one thread is a potential problem.
  - Extra problems with multiple resources.
- By default, any thread can be suspended and resumed at any time.
  - Between any two source code instructions.
  - Even within one source code instruction!
- Need to make things “thread-safe”.
  - Many standard Java methods are.
Some things that can go wrong:

- A thread sees a value, then does something based on that value. Meanwhile, another thread changes the value.
- A thread reads a value, modifies it, and stores it back. In the meantime, another thread has already modified the variable. One of the updates is lost.
- A thread modifies a series of variables as a group. Another thread reads them while the update is partially done, seeing inconsistent data.
Synchronization Techniques

- Semaphores
  - Invented by Edsgar Dijkstra.
  - Locks which allow/bar execution at points in the program.

- Monitors
  - Invented by Per Brinch-Hansen, implemented in Mesa at Xerox PARC.
  - Functions (methods) where only one can be invoked at a time (other threads will block).

- Both are ways to enforce “critical sections”.

Critical Sections In General

- Only one thread / task can be inside the critical section at the same time.
  - Other threads / tasks must block.
- Some policy controls who gets awakened.
  - FIFO
  - Random
  - Priority.
Critical Sections in Java

- Synchronized methods
- Synchronized blocks
Synchronized Methods

- Java implements monitors as “synchronized methods”.
- Within each object, if two or more methods are marked synchronized, only one of them can be invoked (active) at a time.
  - Any other thread trying to invoke it will block.
  - One lock per object.
  - One extra lock per class for the static (class) fields.
Example

class Foo {
    int i, j; // Must be incremented and decremented together.

    void synchronized incrementCounters() {
        i++;  j++;
    }

    void synchronized decrementCounters() {
        i--;  j--;
    }
}
Synchronized Methods II

- Choose your synchronized methods well.
- They must do enough to be effective.
- They must be small enough not to block other threads which can safely run.
Synchronized Blocks

- Synchronized blocks are inside a method.
- A thread must acquire the lock on the specified object before entering the synchronized block.
- The thread automatically releases the lock on exiting the block.
Synchronized Block Example

```java
public void run() {
    while (true) {
        synchronized(this) {
            t1.setText(toString(i++));
            t2.setText(toString(j++));
        }
        try {
            sleep (500);
        } catch (InterruptedException e) {}
    }
}
```
Synchronized Blocks II

z What object should you use for the block?

z Whichever makes the most sense, e.g.

  y The object this method is bound to.
  y The object being updated.
  y One of the objects being updated.

  x Always use the same object!
  x For multiple items, a synchronized method is usually cleaner and safer.
Thread States

- New: Thread object has been created, but not yet started.
- Runnable: Can be run.
- Dead: Has returned from its run() method.
- Blocked: Something is stopping it from running.
Becoming Blocked

- Thread has invoked sleep().
- Some method has invoked suspend() on the thread. [Deprecated in Java 1.2 because it holds locks]
  - Will only become runnable when the thread object gets the resume() method.
Becoming Blocked II

z Thread has invoked wait().

  y Will become runnable when the thread gets the notify() or notifyAll() message.

z Waiting for I/O

z Waiting for a lock.
Wait and Notify

- Wait **does** release locks.
- If a thread invokes `wait()`, it will resume with `notify()`.
- If a thread invokes `wait(msec)`, it will resume after timeout or `notify()`.
notify() and notifyAll()

- Invoking notify() on an object awakens one thread waiting on this object’s lock.
  - Run-time system picks the thread.

- Invoking notifyAll() on an object awakens all threads waiting on this object’s lock.
  - They may then have to compete for locks.
Priority

- You can invoke getPriority(n) and setPriority(n) on a thread.
- Larger number means higher priority.
- Priority only controls how much time the thread gets, not whether it gets any time at all.
Next Classes

z Thursday:
  y Java I/O
  y Java Networking

z Friday:
  y Future of Java

z Monday:
  y Review for Exam 2 (Java)

z Tuesday:
  y Exam 2 (Java)