Thread Pools

- Use a collection of worker threads, not just one
- Can limit maximum number and priorities of threads
- Dynamic worker thread management
  - Sophisticated policy controls
- Often faster than thread-per-message for I/O bound actions

Web Server Using Thread Pool

```java
import EDU.oswego.cs.dl.util.concurrent.PooledExecutor;

private PooledExecutor pool; // WebServer1
public synchronized void startServer() throws ...
    {
        // ... as before
        pool = new PooledExecutor();
        serverThread = new Thread(new ConnectionHandler());
        serverThread.start();
    }

private class ConnectionHandler implements Runnable {
    public void run() {
        RequestHandler r = null;
        try {
            while (!Thread.interrupted()) {
                r = new RequestHandler(server.accept());
                pool.execute(r);
            }
            // ... as before
        }
    }

    public void shutdownServer() throws ...
    {
        // ... as before
        serverThread.interrupt();
        serverThread.join();
        pool.interruptAll();
        server.close();
    }
```

© 2003 David Holmes and Doug Lea
Policies and Parameters for Thread Pools

- The kind of channel used as task queue
  - Unbounded queue, bounded queue, synchronous hand-off, priority queue, ordering by task dependencies, stream, socket

- Bounding resources
  - Maximum number of threads
  - Minimum number of threads
  - “Warm” versus on-demand threads
  - Keepalive interval until idle threads die
    - Later replaced by new threads if necessary

- Saturation policy
  - Block, drop, producer-runs, etc

- These policies and parameters can interact in subtle ways!

Pools in Connection-Based Designs

- For systems with many open connections (sockets), but relatively few active at any given time
- Multiplex the delegations to worker threads via polling
  - Requires underlying support for select/poll and nonblocking I/O
  - Supported in JDK1.4 java.nio