

CMSC 433 – Programming Language  
Technologies and Paradigms  
Spring 2003

Threads and Synchronization  
April 8, 2003

### Locks on Static Fields

- static synchronized -- lock class object
- (non-static) synchronized -- lock this
  - Not the same object!
  - If need to lock class object, use
    - `synchronized (Foo.class) { ... }`
  - Still avoid acquiring two locks at once
    - (e.g., don't synchronize on this if you do that)

80

### Tools Available for Project 4

- Static bug-finding tool
  - Looks for suspicious patterns in source code
  - Finds some common wait/notify problems
  - Not guaranteed to be correct
- Dynamic lock checking tool
  - Every shared object must be guarded by a lock
  - Again, not guaranteed to be correct

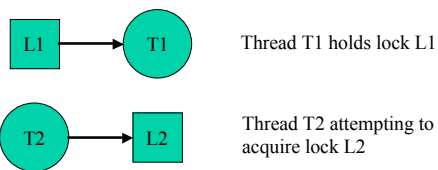
81

### Deadlock

- Quite possible to create code that deadlocks
  - Thread 1 holds lock on **A**
  - Thread 2 holds lock on **B**
  - Thread 1 is trying to acquire a lock on **B**
  - Thread 2 is trying to acquire a lock on **A**
  - Deadlock!
- Not easy to detect when deadlock has occurred
  - other than by the fact that nothing is happening

82

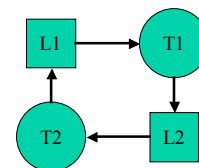
### Deadlock: Wait graphs



Deadlock occurs when there is a cycle in the graph

83

### Wait graph example



T1 holds lock on **L1**  
T2 holds lock on **L2**  
T1 is trying to acquire a lock on **L2**  
T2 is trying to acquire a lock on **L1**

84

## Livelock

- Deadlock arises when *blocked* threads cannot execute
- *Livelock* occurs when threads actually are executing, but no work gets done.
  - Use `notify()` rather than `notifyAll()` and the wrong thread keeps waking up with its condition not met

85

## Field Visibility

- Threads might cache values
- Obtaining a lock forces the thread to get fresh values
- Releasing a lock forces the thread to flush out all pending writes
- **volatile** variables are never cached
- **sleep(...)** doesn't force fresh values
- Many compilers don't perform these optimizations
  - but some do (Hotspot server does)
- Problem might also occur with multiple CPUs

86

## Guidelines to simple/safe multi-threaded programming

- Synchronize access to shared data
- Don't hold a lock on more than one object at a time
  - could cause deadlock
- Hold a lock for as little time as possible
  - reduces blocking waiting for locks
- While holding a lock, don't call a method you don't understand
  - e.g., a method provided by someone else, especially if you can't be sure what it locks

87

## Guidelines (cont.)

- Have to go beyond these guidelines for more complex situations
  - but need to understand threading and synchronization well
- Next: More discussion of threads
  - Pugh, Lea, Holmes, slides from Java One
  - Lea's `util.concurrent` package

88