Thread Synchronization: Built-in Monitor Locks

28 March 2008
CMPT166
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Thread synchronization

- Threads are run by the Executor
- If two threads wish to modify a shared object, we need synchronization
  - Mutual exclusion (mutex): only one thread accesses shared object at a time
  - Locks: a way to implement mutex
    - Thread asks for lock before modifying object
    - If it gets the lock, it can modify
    - If not, wait (block) until the lock is freed
    - Free the lock when done modifying

Lock interface

- Any object can be a lock if it implements Lock
  - In package java.util.concurrent.locks
  - Two methods: .lock() and .unlock()
    - .lock() will wait until the lock is freed
    - If many threads are waiting, which one gets it first?
- ReentrantLock: can set fairness policy
  - Longest-waiting thread gets the lock first
- Deadlock happens when each thread is waiting on a lock held by another thread

synchronized block

- Every Java object has a built-in monitor lock
- Code that needs exclusive access to an object can use a synchronized block:

```
synchronized ( studentDB ) {
    studentDB.addStudent();
}
```

- Waits for and acquires the monitor lock on the object
- Releases the lock when done
- Don't need to implement the Lock interface
  - See AddApples.java