CMSC 330: Organization of Programming Languages

Threads

Synchronization

- Refers to mechanisms allowing a programmer to control the execution order of some operations across different threads in a concurrent program.
- Different languages have adopted different mechanisms to allow the programmer to synchronize threads.
- Java has several mechanisms; we'll look at locks first.

Locks (Java 1.5)

```
interface Lock {
  void lock();
  void unlock();
  ... /* Some more stuff, also */
}
class ReentrantLock implements Lock { ... }
```

- Only one thread can hold a lock at once
  - Other threads that try to acquire it *block* (or become suspended) until the lock becomes available
- *Reentrant lock* can be reacquired by same thread
  - As many times as desired
  - No other thread may acquire a lock until has been released same number of times it has been acquired

Avoiding Interference: Synchronization

```
public class Example extends Thread {
  private static int cnt = 0;
  static Lock lock = new ReentrantLock();
  public void run() {
    lock.lock();
    int y = cnt;
    cnt = y + 1;
    lock.unlock();
    }
  }
  ...
}
```

***Lock**, for protecting the shared state*

***Acquires** the lock; Only succeeds if not held by another thread*

***Releases** the lock*

Applying Synchronization

```
int cnt = 0;
t1.run() {
     lock.lock();
     int y = cnt;
     cnt = y + 1;
     lock.unlock();
}
t2.run() {
     lock.lock();
     int y = cnt;
     cnt = y + 1;
     lock.unlock();
}
```

Shared state cnt = 0

T1 acquires the lock

Applying Synchronization

```
int cnt = 0;
t1.run() {
     lock.lock();
     int y = cnt;
     cnt = y + 1;
     lock.unlock();
}
t2.run() {
     lock.lock();
     int y = cnt;
     cnt = y + 1;
     lock.unlock();
}
```

Shared state cnt = 0

y = 0

T1 reads cnt into y