Concurrent Programming:
The LAB
More Resources

- Sun’s Java documentation
  - [http://java.sun.com/javase/6/docs/api/](http://java.sun.com/javase/6/docs/api/)


- Concurrent Programming in Java: Design Principles and Patterns by Doug Lea (ISBN-10: 0201310090)
Rules for Monitors

- Always hold lock when operating on a condition var
- Grab lock at beginning of function, release at end
- Always test predicated state in while loop
  - while(condition == false) { cv.await(); }
  - NOT if(condition == false) {cv.await();}
  - while makes signal a hint
- (Almost) never use sleep
The absence of block-structured locking removes the automatic release of locks that occurs with synchronized methods and statements. In most cases, the following idiom should be used:

```java
Lock l = ...;
l.lock();
try {
    // access the resource protected by this lock
} finally {
    l.unlock();
}
```
Using Locks Correctly

- Java provides convenient mechanism.

```java
import java.util.concurrent.locks.ReentrantLock;

aLock.lock();
try {
    ...
} finally {
    aLock.unlock();
}
return 0;
```
Using Locks Correctly

- **super calls the super-class version of the method**

```java
public class RogueCoarse extends GalleryRogue{
    RogueCoarse(String title,
                 int _pauseInterval,
                 int _maxInterval,
                 GalleryPanel _panel,
                 int _color) {
        super(title, _pauseInterval, _maxInterval, _panel, _color);
        lanes = _panel.getLaneLocks();
    }
```
Using Locks Correctly

- static fields belong to class, not object

```java
public class RoguePurple extends GalleryRogue{
    private static ReentrantLock rogueLock
        = new ReentrantLock(true);
    private static Condition blueHere
        = rogueLock.newCondition();
    private static Condition purpleDone
        = rogueLock.newCondition();
    private static int red_lane;
}
```