Lock

```
public interface Lock {
	public void lock();
	public void unlock();
	public Condition newCondition();
}
```

Condition

```
public interface Condition {
	public void await() throws InterruptedException;
	public void signal();
	public void signalAll();
}
```

ReentrantLock

```
public class ReentrantLock implements Lock {
	Thread owner = null;
	Object mutex = new Object();
	int counter;

	@Override
	public void lock() {
		Thread currentThread = Thread.currentThread();
		synchronized (mutex) {
			if(counter < 0)
				throw new IllegalMonitorStateException("counter < 0");
			while(owner != null && owner != currentThread) {
				try {
					mutex.wait();
				} catch(InterruptedException e) {
					e.printStackTrace();
				}
			}
		}

		if(owner == null) {
			owner = currentThread;
		}
		System.out.println(owner.getName() + " ha acquisito il lock");
		++counter;
	}
```

```
	@Override
	public void unlock() {
		Thread currentThread = Thread.currentThread();
		synchronized (mutex) {
			if(owner != currentThread) {
				throw new IllegalMonitorStateException();
			}
			if(counter <= 0)
				throw new IllegalMonitorStateException("counter <= 0");
			--counter;
			if(counter == 0) {
				owner = null;
				mutex.notify();
			}
		}
	}

	@Override
	public Condition newCondition() {
		return new InnerCondition();
	}
	class InnerCondition implements Condition {
		Object conditionMutex = new Object();
		@Override
		public void await() throws InterruptedException {
			unlock();
			synchronized (conditionMutex) {
				conditionMutex.wait();
			}
			lock();
		}
		@Override
		public void signal() {
			synchronized (mutex) {
				if(owner != Thread.currentThread())
					throw new IllegalMonitorStateException();
			}
			synchronized (conditionMutex) {
				conditionMutex.notify();
			}
		}
		@Override
		public void signalAll() {
			synchronized (mutex) {
				if(owner != Thread.currentThread())
					throw new IllegalMonitorStateException();
			}
			synchronized (conditionMutex) {
				conditionMutex.notifyAll();
			}
		}
	}
}
```