Интерфейс Lock

Для управления доступом к общему ресурсу в качестве альтернативы оператору synchronized мы можем использовать блокировки. Функциональность блокировок заключена в пакете **java.util.concurrent.locks**.

Вначале поток пытается получить доступ к общему ресурсу. Если он свободен, то на него накладывает блокировку. После завершения работы блокировка с общего ресурса снимается. Если же ресурс не свободен и на него уже наложена блокировка, то поток ожидает, пока эта блокировка не будет снята.

Классы блокировок реализуют интерфейс **Lock**, который определяет следующие методы:

- **void lock()**: ожидает, пока не будет получена блокировка
- **void lockInterruptibly() throws InterruptedException**: ожидает, пока не будет получена блокировка, если поток не прерван
- **boolean tryLock()**: пытается получить блокировку, если блокировка получена, то возвращает true. Если блокировка не получена, то возвращает false. В отличие от метода lock() не ожидает получения блокировки, если она недоступна
- **void unlock()**: снимает блокировку
- **Condition newCondition()**: возвращает объект Condition, который связан с текущей блокировкой

Организация блокировки в общем случае довольно проста: для получения блокировки вызывается метод lock(), а после окончания работы с общими ресурсами вызывается метод unlock(), который снимает блокировку.

Объект Condition позволяет управлять блокировкой.

Как правило, для работы с блокировками используется класс ReentrantLock из пакета java.util.concurrent.locks. Данный класс реализует интерфейс Lock.

Интерфейс Condition

Применение условий в блокировках позволяет добиться контроля над управлением доступом к потокам. Условие блокировки представлет собой объект интерфейса **Condition** из пакета *java.util.concurrent.locks*.

Применение объектов Condition во многом аналогично использованию методов wait/notify/notifyAll класса Object. В частности, мы можем использовать следующие методы интерфейса Condition:

- **await**: поток ожидает, пока не будет выполнено некоторое условие и пока другой поток не вызовет методы signal/signalAll. Во многом аналогичен методу wait класса Object
- **signal**: сигнализирует, что поток, у которого ранее был вызван метод await(), может продолжить работу. Применение аналогично использованию методу notify класса Object
- **signalAll**: сигнализирует всем потокам, у которых ранее был вызван метод await(), что они могут продолжить работу. Аналогичен методу notifyAll() класса Object

Эти методы вызываются из блока кода, который попадает под действие блокировки ReentrantLock. Сначала, используя эту блокировку, нам надо получить объект Condition:

```
ReentrantLock locker = new ReentrantLock();
Condition condition = locker.newCondition();
```

Как правило, сначала проверяется условие доступа. Если соблюдается условие, то поток ожидает, пока условие не изменится:

```
while (условие)
    condition.await();
```

После выполнения всех действий другим потокам подается сигнал об изменении условия:

```
condition.signalAll();
```

Важно в конце вызвать метод signal/signalAll, чтобы избежать возможности взаимоблокировки потоков.

У Condition все-таки есть определенные возможности, которых нет у wait/notify. А именно то, что у одного Lock может быть несколько разных Condition, и разные потоки могут ожидать выполнения разных условий (Condition) на одном и том же локе.