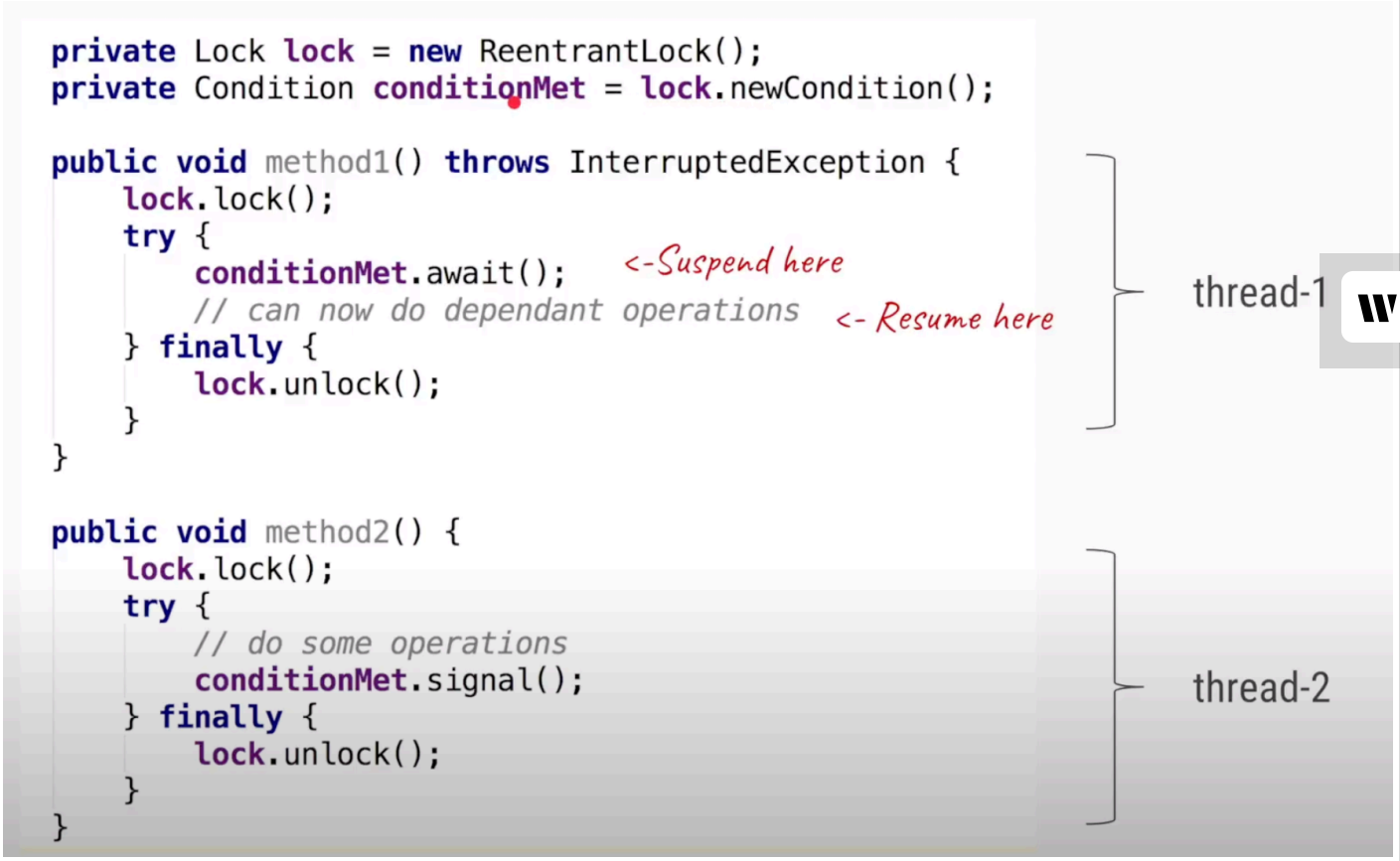
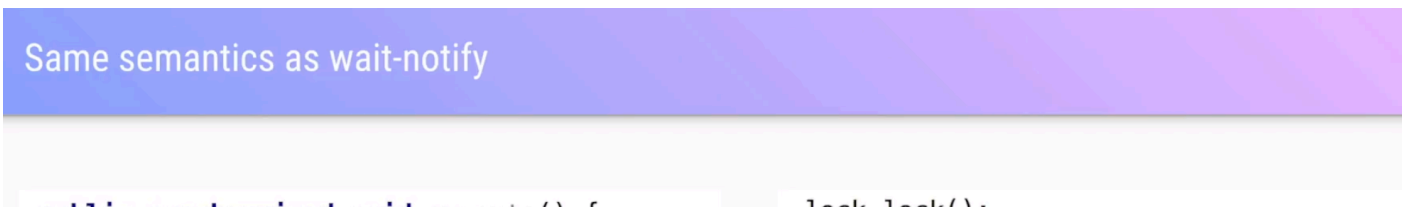
Multithreading: Locks: Condition class

Condition Class:

Condition.await will **suspend** thread and when any other thread will give **signal** then longest time duration awaiting thread will start executing again and in case of signalAll all awaiting thread will start executing.



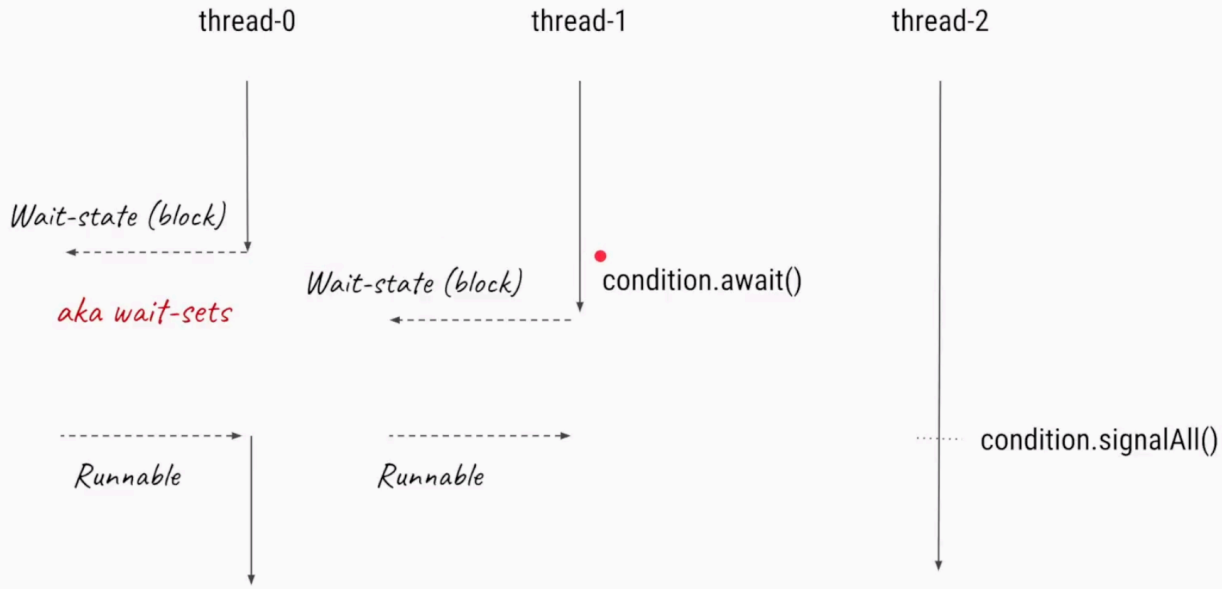
Similarity with wait-notify



```java
public synchronized void execute() {

    // wait for monitor notify
    try {
        monitor.wait();
    } catch (InterruptedException e) {
        System.err.println("Interrupted");
    }

    // notify thread waiting on the monitor
    monitor.notify();

    // notify all threads
    monitor.notifyAll();
}
```

```java
lock.lock();

// wait for monitor notify
try {
    condition.wait();
} catch (InterruptedException e) {
    System.err.println("Interrupted");
}

// notify thread waiting on the monitor
condition.signal();

// notify all threads
condition.signalAll();

lock.unlock();
```

signalAll / wait-sets / fairness



Here with **condition.signal** thread 0 will start operating as this is waiting since a long before.

Note : Sometimes thread wake up without any signal , to stop it we could add this condition inside a loop

Perform await in loop always

```
public String consume() throws InterruptedException {
    lock.lock();
    try {

        while (count == 0)
            added.await();

        return getData();

    } finally {
        lock.unlock();
    }
}
```

Spurious wake ups

Producer consumer problem using these concepts:

```
private Lock lock = new ReentrantLock();
private Condition added = lock.newCondition();
private Condition removed = lock.newCondition();
```

```
public void produce() throws Interrup
    lock.lock();
    try {
        while (count == MAX_COUNT)
            removed.await();

        addData();
        added.signal();

    } finally {
        lock.unlock();
    }
}
```

```
public String consume() throws Inte
    lock.lock();
    try {
        while (count == 0)
            added.await();

        String data = getData();
        removed.signal();

        return data;
    } finally {
        lock.unlock();
    }
}
```

Producer Consumer