Java ConditionObject:
Structure & Functionality

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Understand what condition variables are
- Note a human known use of condition variables
- Know what pattern they implement
- Recognize common use cases where condition variables are applied
- Recognize the structure & functionality of Java ConditionObject
Overview of Java

ConditionObject
Overview of Java `ConditionObject`

- `ConditionObject` provides the condition variable abstraction

```java
public class ConditionObject
    implements Condition, java.io.Serializable {
    ...
```

Class `AbstractQueuedSynchronizer.ConditionObject`

```java
java.lang.Object
    java.util.concurrent.locks.AbstractQueuedSynchronizer.ConditionObject

All implemented interfaces:
    Serializable, Condition

Enclosing class:
    AbstractQueuedSynchronizer
```

```java
public class AbstractQueuedSynchronizer.ConditionObject
    extends Object
    implements Condition, Serializable

Condition implementation for a AbstractQueuedSynchronizer serving as the basis of a Lock implementation.

Method documentation for this class describes mechanics, not behavioral specifications from the point of view of Lock and Condition users. Exported versions of this class will in general need to be accompanied by documentation describing condition semantics that rely on those of the associated AbstractQueuedSynchronizer.
```

See `docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/AbstractQueuedSynchronizer.ConditionObject.html`
Overview of Java ConditionObject

• ConditionObject provides the condition variable abstraction
• Implements Condition interface

public class ConditionObject implements Condition, java.io.Serializable {

Interface Condition

All Known Implementing Classes:
AbstractQueuedLongSynchronizer.ConditionObject, AbstractQueuedSynchronizer.ConditionObject

public interface Condition

Condition factors out the Object monitor methods (wait, notify and notifyAll) into distinct objects to give the effect of having multiple wait-sets per object, by combining them with the use of arbitrary Lock implementations. Where a Lock replaces the use of synchronized methods and statements, a Condition replaces the use of the Object monitor methods.

Conditions (also known as condition queues or condition variables) provide a means for one thread to suspend execution (to "wait") until notified by another thread that some state condition may now be true. Because access to this shared state information occurs in different threads, it must be protected, so a lock of some form is associated with the condition. The key property that waiting for a condition provides is that it atomically releases the associated lock and suspends the current thread, just like Object.wait.

A Condition instance is intrinsically bound to a lock. To obtain a Condition instance for a particular Lock instance use its newCondition() method.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/Condition.html
Overview of Java ConditionObject

- ConditionObject is nested within the AbstractQueuedSynchronizer class
- This framework is used by Java synchronizers that rely on FIFO wait queues

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/AbstractQueuedSynchronizer.html
Overview of Java ConditionObject

- A ConditionObject provides a “wait queue” of nodes

See gee.cs.oswego.edu/dl/papers/aqs.pdf
Overview of Java ConditionObject

- A ConditionObject provides a “wait queue” of nodes
- Enables a set of threads (i.e., the “wait set”) to coordinate their interactions
A ConditionObject provides a "wait queue" of nodes

- Enables a set of threads (i.e., the "wait set") to coordinate their interactions
- e.g., by selecting the order & conditions under which they run
Overview of Java ConditionObject

- A ConditionObject is *always* used with a lock

```
Consumer

ArrayBlockingQueue
  take()
  put()
  take()

ConditionObject
  await()
  signal()
  signalAll()

Producer

ReentrantLock
  lock()
  unlock()
  newCondition()

<<uses>>

See earlier part on "Java ReentrantLock"
```
A ConditionObject is *always* used with a lock
- This lock protects shared state in a condition expression from concurrent manipulation

Overview of Java ConditionObject

- **Consumer**
  - `take()`

- **ArrayBlockingQueue**
  - `put()`
  - `take()`

- **ConditionObject**
  - `await()`
  - `signal()`
  - `signalAll()`

- **Producer**
  - `put()`

- **ReentrantLock**
  - `lock()`
  - `unlock()`
  - `newCondition()`

```
<<uses>>
```

```
<<uses>>
```
A ConditionObject is *always* used with a lock

- This lock protects shared state in a condition expression from concurrent manipulation

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/ReentrantLock.html#newCondition](docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/ReentrantLock.html#newCondition)

newCondition() is a factory method that returns a ConditionObject that can be used with this lock
Overview of Java ConditionObject

- Both ReentrantLock & ConditionObject have internal queues

```
Consumer

ArrayBlocking Queue
  put()
  take()

Producer
  put()

ConditionObject
  await()
  signal()
  signalAll()

Reentrant Lock
  lock()
  unlock()
  newCondition()
```

<<uses>>
Overview of Java ConditionObject

- Both ReentrantLock & ConditionObject have internal queues

Queues up threads that are waiting to acquire the lock
Overview of Java ConditionObject

- Both ReentrantLock & ConditionObject have internal queues

Queues up threads waiting for some condition(s) to become true

Diagram:

- Consumer
- Producer
- ArrayBlockingQueue
  - put()
  - take()
- ConditionObject
  - await()
  - signal()
  - signalAll()
- ReentrantLock
  - lock()
  - unlock()
  - newCondition()
Overview of Java ConditionObject

- User-defined Java objects can have multiple ConditionObjects (COs)

```
Consumer

ArrayBlockingQueue
  - put()
  - take()
  - put()
  - take()

Producer

ConditionObject
  - await()
  - signal()
  - signalAll()

ReentrantLock
  - lock()
  - unlock()
  - newCondition()
```

*Two COs: notEmpty & notFull*
Overview of Java ConditionObject

- User-defined Java objects can have multiple ConditionObjects (COs)
  - Multiple COs enable more sophisticated & efficient ways to coordinate multiple threads.

![Diagram showing the relationships between Consumer, Producer, ArrayBlockingQueue, ConditionObject, and ReentrantLock.](Diagram.png)

- Consumer
- Producer
- ArrayBlockingQueue
  - put()
  - take()
- ConditionObject
  - await()
  - signal()
  - signalAll()
- ReentrantLock
  - lock()
  - unlock()
  - newCondition()
Overview of Java ConditionObject

- User-defined Java objects can have multiple ConditionObjects (COs)
- Multiple COs enable more sophisticated & efficient ways to coordinate multiple threads
- e.g., multiple wait-sets per user object that share a lock & are notified on different conditions

Overview of Java ConditionObject

- In contrast, Java’s built-in monitor objects only support *one* monitor condition

See [github.com/douglascraigschmidt/LiveLessons/tree/master/SimpleBlockingQueue](https://github.com/douglascraigschmidt/LiveLessons/tree/master/SimpleBlockingQueue)
Overview of Java ConditionObject

- In contrast, Java’s built-in monitor objects only support one monitor condition

```
Simple BlockingQueue
  put()
  take()

Producer
  put()

Consumer
  take()

Wait Queue
  wait()
  notify()
  notifyAll()

Entrance Queue
```

i.e., there’s just a single “wait queue”

See upcoming lesson on “Java Built-in Monitor Objects”
Overview of Java ConditionObject

- In contrast, Java’s built-in monitor objects only support *one* monitor condition.
- Yields inefficient programs that require excessive notifications & use of notifyAll().

See [www.dre.vanderbilt.edu/~schmidt/C++2Java.html#concurrency](http://www.dre.vanderbilt.edu/~schmidt/C++2Java.html#concurrency)
Overview of Java ConditionObject

- In contrast, Java’s built-in monitor objects only support one monitor condition.
- Yields inefficient programs that require excessive notifications & use of notifyAll().
- e.g., producers & consumers must both wake up on every change to the queue, even if a given thread can’t proceed.

```java
synchronized(this) {
    while (mList.isEmpty())
        wait();
    notifyAll();
    return mList.poll();
}
```

End of Java ConditionObject:
Structure & Functionality