Java ConditionObject (Part 1)

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

```
Lock l = new Lock()
Condition cond = l.newCondition()
...

l.lock()
while (conditionNotSatisfied())
    cond.await()
    doOperationProcessing()
```
Learning Objectives in this Part of the Lesson

- Understand what condition variables are
- Know what pattern they implement

```java
Lock l = new Lock();
Condition cond = l.newCondition();
...
l.lock();
while (conditionNotSatisfied())
    cond.await();
doOperationProcessing();
```
Learning Objectives in this Part of the Lesson

- Understand what condition variables are
- Know what pattern they implement

Condition variables can be tricky, so I recommend you rewatch this lesson & read the links carefully

```java
Lock l = new Lock()
Condition cond = l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
    cond.await()
doOperationProcessing()
```
Overview of Condition Variables
Overview of Condition Variables

• A CV is a synchronizer that allows a thread to (repeatedly) suspend its execution until a condition is satisfied

See blog.dcoles.net/2012/02/understanding-how-to-use-condition.html

Wheel of Pain – Conan the Barbarian
Overview of Condition Variables

• A CV is a synchronizer that allows a thread to (repeatedly) suspend its execution until a condition is satisfied
• A thread whose execution is suspended on a CV is said to be “blocked” on the CV

Tree of Woe – Conan the Barbarian
Overview of Condition Variables

• A CV is implemented as a queue of threads that wait for certain condition(s) to be satisfied

```
Lock l = new Lock()
Condition cond = l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
    cond.await()
doOperationProcessing()
```

See [en.wikipedia.org/wiki/Monitor_(synchronization)#Condition_variables](en.wikipedia.org/wiki/Monitor_(synchronization)#Condition_variables)
Overview of Condition Variables

- A CV is implemented as a queue of threads that wait for certain condition(s) to be satisfied
- This queue of threads is known as the “wait set”

```
Lock l = new Lock()
Condition cond = l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
    cond.await()
doOperationProcessing()
```
Overview of Condition Variables

- CVs are often used when *mutual exclusion* alone is inadequate.
Overview of Condition Variables

- CVs are often used when *mutual exclusion* alone is inadequate, e.g.
- Inefficient use of resources
  - e.g., due to excessive “busy waiting”

See [en.wikipedia.org/wiki/Busy_waiting](http://en.wikipedia.org/wiki/Busy_waiting)
Overview of Condition Variables

• CVs are often used when *mutual exclusion* alone is inadequate, e.g.
  • Inefficient use of resources
  • Insufficient to ensure *coordination*
    • e.g., what to do when a thread encounters shared state that it can't do any work upon (yet)
Overview of Condition Variables

- CVs are often used when *mutual exclusion* alone is inadequate, e.g.
  - Inefficient use of resources
  - Insufficient to ensure *coordination*
    - e.g., what to do when a thread encounters shared state that it can't do any work upon (yet)

**Waiting on an empty list**

Implementing Guarded Suspension with CVs
Implementing Guarded Suspension with CVs

- CVs are most often used to implement the *Guarded Suspension* pattern

See [en.wikipedia.org/wiki/Guarded_suspension](en.wikipedia.org/wiki/Guarded_suspension)
Implementing Guarded Suspension with CVs

- This pattern is applied to operations that can run only when a condition is satisfied

```java
Lock l = new Lock()
Condition cond = l.newCondition()
...
l.lock()
while (!conditionNotSatisfied())
    cond.await()
doOperationProcessing()
```
Implementing Guarded Suspension with CVs

- This pattern is applied to operations that can run only when a condition is satisfied, e.g.,
- a lock is acquired

```
Lock l = new Lock()
Condition cond = l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
    cond.await()
doOperationProcessing()
```
Implementing Guarded Suspension with CVs

- This pattern is applied to operations that can run only when a condition is satisfied, e.g.,
  - a lock is acquired
  - a precondition holds

```java
Lock l = new Lock();
Condition cond = l.newCondition();
...

l.lock();
while (conditionNotSatisfied()) {
    cond.await()
    doOperationProcessing()
}
```
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on may now be satisfied.

```java
Lock l = new Lock();
Condition cond = l.newCondition();
...

l.lock()
while (conditionNotSatisfied())
    cond.await()
doOperationProcessing()
```
Implementing Guarded Suspension with CVs

- In this example thread T\(_1\) uses a CV to suspend its execution until thread T\(_n\) notifies it that shared state it's waiting on *may* now be satisfied.

```
Lock l = new Lock()
Condition cond = l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
  cond.await()
doOperationProcessing()
```

Note the tentative nature of "may"..
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on may now be satisfied.

First, a lock must be acquired...

```
Lock l = new Lock()
Condition cond = l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
    cond.await()
doOperationProcessing()
```
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on $may$ now be satisfied

```
Lock l = new Lock()
Condition cond = l.newCondition()
...
  l.lock()
  while (conditionNotSatisfied())
    cond.await()
  doOperationProcessing()
```

Second, a condition is checked (in a loop) with the lock held..
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on *may* now be satisfied
- A condition can be arbitrarily complex

**ConditionVariable**

- `await()`
- `signal()`
- `signalAll()`

---

```java
Lock l = new Lock();
Condition cond = l.newCondition();
...

l.lock()
while (!conditionNotSatisfied())
cond.await()
doOperationProcessing()
```

---

*Any state shared between threads must be protected by a lock associated with the CV*

---

*e.g., a method call, an expression involving shared state, etc.*
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on may now be satisfied.
- A condition can be arbitrarily complex.

```
Lock l = new Lock();
Condition cond = l.newCondition();
...
l.lock()  
while (conditionNotSatisfied())
    cond.await()  
doOperationProcessing() 
```

The calling thread will block (possibly repeatedly) while the condition is not satisfied (await() atomically releases the lock).
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on *may* now be satisfied
- A condition can be arbitrarily complex

```
Lock l = new Lock()
Condition cond = l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
    cond.await()
    doOperationProcessing()
```

Another thread can signal the condition when shared state may now be true
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on *may* now be satisfied.

- A condition can be arbitrarily complex.

```java
Lock l = new Lock()
Condition cond = l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
    cond.await()
    doOperationProcessing()
```

*await() reacquires the lock & the condition is rechecked in the loop*
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on *may* now be satisfied
- A condition can be arbitrarily complex

```java
Lock l = new Lock();
Condition cond = l.newCondition();
...
l.lock();
while (!conditionNotSatisfied())
    cond.await();
doOperationProcessing();
```
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on *may* now be satisfied
- A condition can be arbitrarily complex
- Waiting on a CV releases the lock & suspends the thread *atomically*

```java
Lock l = new Lock()
Condition cond = l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
    cond.await()
doOperationProcessing()
```

*The lock is released when the thread is suspended on the CV*
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on may now be satisfied
  - A condition can be arbitrarily complex
  - Waiting on a CV releases the lock & suspends the thread atomically
  - Thread $T_1$ is suspended until thread $T_n$ signals the CV

```
Lock l = new Lock()
Condition cond = l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
    cond.await()
doOperationProcessing()
```
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on may now be satisfied.
  - A condition can be arbitrarily complex.
  - Waiting on a CV releases the lock & suspends the thread *atomically*.
  - Thread $T_1$ is suspended until thread $T_n$ signals the CV.

When a thread is signaled it wakes up & must re-acquire its associated lock.

```
cond.signal()
```
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on may now be satisfied.
- A condition can be arbitrarily complex.
- Waiting on a CV releases the lock & suspends the thread *atomically*.
- Thread $T_1$ is suspended until thread $T_n$ signals the CV.

```java
Lock l = new Lock();
Condition cond = l.newCondition();
...
l.lock();
while (!cond.conditionNotSatisfied())
    cond.await();
doOperationProcessing();
```

After lock is re-acquired the thread can reevaluate its condition to see if it’s satisfied.
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on *may* now be satisfied
  - A condition can be arbitrarily complex
  - Waiting on a CV releases the lock & suspends the thread *atomically*
  - Thread $T_1$ is suspended until thread $T_n$ signals the CV

```java
Lock l = new Lock()
Condition cond = l.newCondition()
...
lock()
while (conditionNotSatisfied())
    cond.await()
doOperationProcessing()
```

*If condition is not satisfied the thread must wait (which releases the lock atomically)*
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on may now be satisfied
- A condition can be arbitrarily complex
- Waiting on a CV releases the lock & suspends the thread *atomically*
- Thread $T_1$ is suspended until thread $T_n$ signals the CV

```
Lock l = new Lock()
Condition cond = l.newCondition()
...
...
l.lock()
while (conditionNotSatisfied())
    cond.await()
    doOperationProcessing()
```

After the lock is re-acquired & the condition is satisfied the operation can proceed (with lock held)
End of Java ConditionObject (Part 1)