

```
Can't we just synchronize the two methods as we did previously?

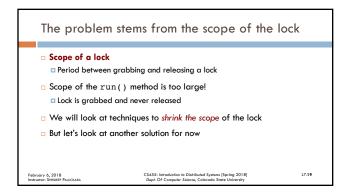
If we synchronized both run() and setDone()?

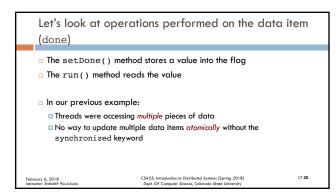
setDone() would never execute!

The run() method does not exit until the done flag is set

But the done flag cannot be set because setDone() cannot execute till run() completes

Uh oh ...
```





But Java specifies that the loading and storing of variables is atomic

Except for long and double variables

The setDone() should be atomic

The run() method has only one read operation of the data item

The race condition should not exist

But why is it there?

C5455: Introduction to Distributed Systems [Spring 2018]

Days Of Computer Science, Calorado State University

Threads are allowed to hold values of variables in registers

When one thread changes the value of the variable?
Another thread may not see the changed variable
This is particularly true in loops controlled by a variable

Copping thread loads value of variable in register and does not notice when value is changed by another thread

Two approaches to solving this

Providing setter and getter methods for variable and using the synchronized keyword

When lock is acquired, temporary values stored in registers are flushed to main memory

The volatile keyword

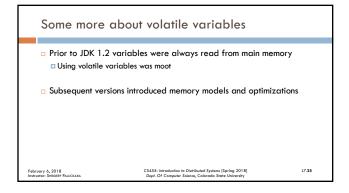
Much cleaner solution

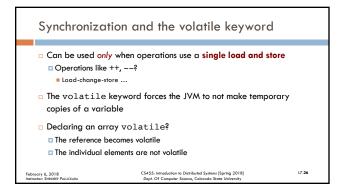
Petrocry 6, 2018

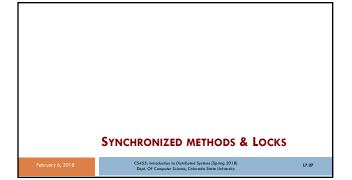
Entrocry 6, 2018

Days of Complete Sames, Colorado State University

12.23







Synchronizing methods

Not possible to execute the same method in one thread while ...

Method is running in another thread

If two different synchronized methods in an object are called?

They both require the lock of the same object

Two or more synchronized methods of the same object can never run in parallel in separate threads

A lock is based on a specific instance of an object

Not on a particular method or class

Suppose we have 2 objects: objectA and objectB with synchronized methods modifyData() and utilizeData()

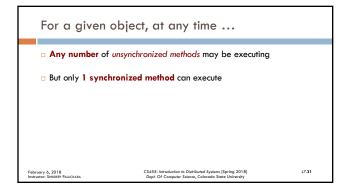
One thread can execute objectA.modifyData() while another executes objectB.utilizeData() in parallel

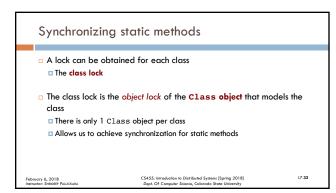
Two different locks are grabbed by two different threads
No need for threads to wait for each other

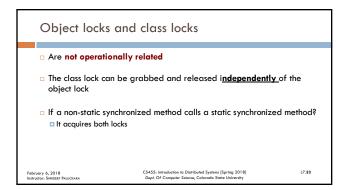
How does a synchronized method behave in conjunction with an unsynchronized one?

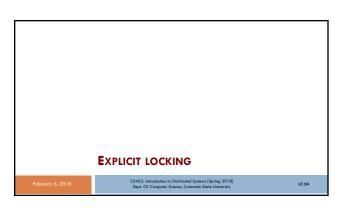
Synchronized methods try to grab the object lock
Only 1 synchronized method in a object can run at a time ... provides data protection

Unsynchronized methods
Don't grab the object lock
Can execute at any time ... by any thread
Regardless of whether a synchronized method is running









The synchronized keyword

Serializes accesses to synchronized methods in an object

Not suitable for controlling lock scope in certain situations

Can be too primitive in some cases

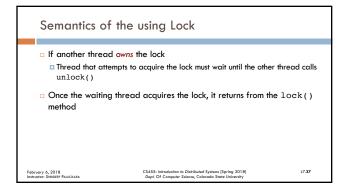
February 6, 2018
Dept Of Computer Senses, Califordia Synthesis (Spring 2018)
Dept Of Computer Senses, Califordia Sans University

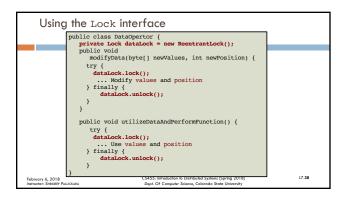
17.33

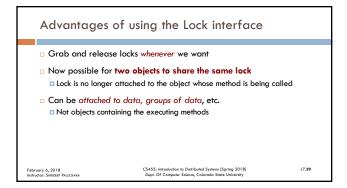
Many synchronization schemes in J2SE 5.0 onwards implement the Lock interface

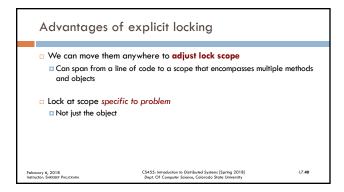
Two important methods
lock() and unlock()

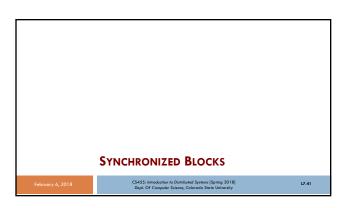
Similar to using the synchronized keyword
Call lock() at the start of the method
Call unlock() at the end of the method
Difference: we have an actual object that represents the lock
Store, pass around, or discard

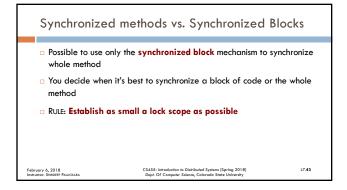


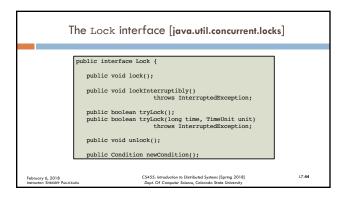


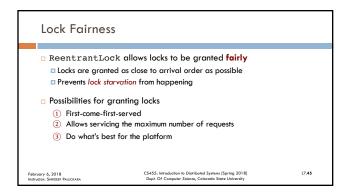


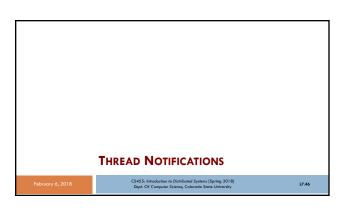










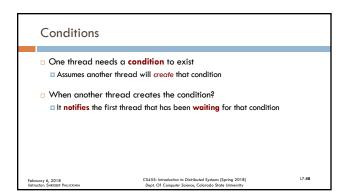


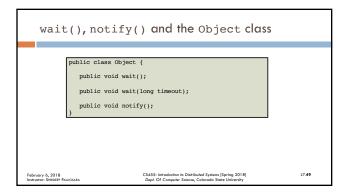
Objects and communications

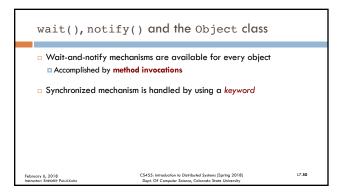
Every object has a lock
Every object also includes mechanisms that allow it to be a waiting area
Allows communication between threads

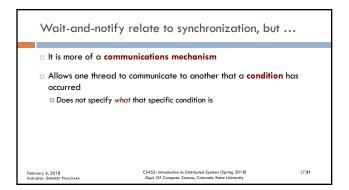
February 4, 2018
Dept of Compoter Saintes, Colorado State University

12-47
Dept of Compoter Saintes, Colorado State University









Can wait-and-notify replace the synchronized mechanism?

No

Does not solve the race condition that the synchronized mechanism solves

Must be used in conjunction with the synchronized lock
Prevents race condition that exists in the wait-notify mechanism itself

February 6, 2018

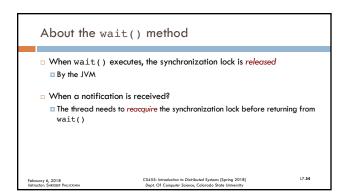
C5435: Introduction to Distributed Systems [Spring 2018]
Digit. Of Computer Science, Colorado State Utbernity

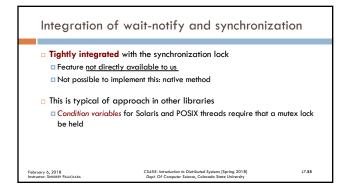
```
A code snippet that uses wait-notify to control the execution of the thread

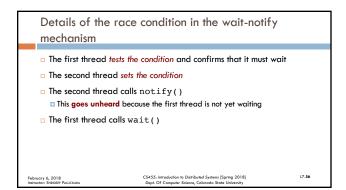
public class Tester implements Runnable {
    private boolean done = true;
    public synchronized run() {
        while (true) {
            if (done) wait();
            else { . . . Logic . . . wait(100);}
        }
    }
    public synchronized void setDone(boolean b) {
        done = b;
        if (idone) notify();
    }
}

February A, 2018

CS455 introduction to Distributed Syntems (Spring 2018)
    bayer Of Company Statems, Colorado State University
```







How does the potential race condition get resolved?

To call wait() or notify()

Obtain lock for the object on which this is being invoked

It seems as if the lock has been held for the entire wait() invocation, but ...

wait() releases lock prior to waiting

Reacquires the lock just before returning from wait()

February 4, 2018

Section 18 Distributed Systems (Spring 2018)

Days Of Computer Science, Colorado Store University

Is there a race condition during the time wait() releases and reacquires the lock?

wait() is tightly integrated with the lock mechanism

Object lock is not freed until the waiting thread is in a state in which it can receive notifications

System prevents race conditions from occurring here

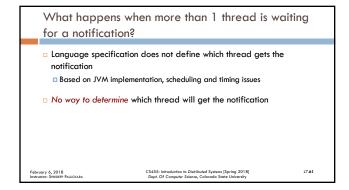
If a thread receives a notification is it guaranteed that condition is set?

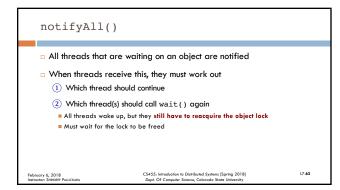
No
Prior to calling wait(), test condition while holding lock
Upon returning from wait() refest condition to see if you should wait() again

February 4, 2018
Dept of Computer Sames, Coloredo State Ubinerally
Dept of Computer Sames, Coloredo State Ubinerally

What if notify() is called and no thread is waiting?

Description:
Wait-and-notify mechanism has no knowledge about the condition about which it notifies
If notify() is called when no other thread is waiting?
The notification is lost





The contents of this slide-set are based on the following references

Java Threads. Scott Oaks and Henry Wong. 3rd Edition. O'Reilly Press. ISBN: 0-596-00782-5/978-0-596-00782-9. [Chapter 3, 4]
Operating Systems Principles and Practice. Thomas Anderson and Michael Dahlin. 2<sup>nd</sup> Edition. ISBN: 978-0985673529. [Chapter 5]