Introduction to Java
ReentrantLock

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 the concept of mutual exclusion in concurrent programs

See en.wikipedia.org/wiki/Mutual_exclusion
Learning Objectives in this Part of the Lesson

• Understand the concept of mutual exclusion in concurrent programs

• Note a human-known use of mutual exclusion
Overview of Mutual Exclusion Locks
Overview of Mutual Exclusion Locks

- A mutual exclusion lock (mutex) defines a “critical section”

A critical section is a group of instructions or region of code that must be executed atomically.
Overview of Mutual Exclusion Locks

- A mutual exclusion lock (mutex) defines a "critical section"
- Ensures only one thread can run in a block of code at a time
Overview of Mutual Exclusion Locks

• A mutual exclusion lock (mutex) defines a “critical section”
  • Ensures only one thread can run in a block of code at a time
  • The thread enters the critical section after acquiring the lock

Acquire lock

Critical Section
Overview of Mutual Exclusion Locks

- A mutual exclusion lock (mutex) defines a “critical section”
- Ensures only one thread can run in a block of code at a time
  - The thread enters the critical section after acquiring the lock
- This thread runs without fear of other threads corrupting shared mutable state
Overview of Mutual Exclusion Locks

- A mutual exclusion lock (mutex) defines a “critical section”
  - Ensures only one thread can run in a block of code at a time
  - Other threads are kept “at bay”
Overview of Mutual Exclusion Locks

- A mutual exclusion lock (mutex) defines a “critical section”
  - Ensures only one thread can run in a block of code at a time

- Other threads are kept “at bay”
  - Prevent corruption of shared mutable state that can be set/get by concurrent operations
Overview of Mutual Exclusion Locks

- A mutual exclusion lock (mutex) defines a “critical section”
  - Ensures only one thread can run in a block of code at a time
- Other threads are kept “at bay”
  - Prevent corruption of shared mutable state that can be set/get by concurrent operations

Other threads must obey the locking protocol or chaos will ensue!!
Overview of Mutual Exclusion Locks

- A mutual exclusion lock (mutex) defines a “critical section”
  - Ensures only one thread can run in a block of code at a time
- Other threads are kept “at bay”
  - Prevent corruption of shared mutable state that can be set/get by concurrent operations
- Race conditions could occur if multiple threads run within a critical section

Race conditions arise when a program depends on the sequence or timing of threads for it to operate properly

See [en.wikipedia.org/wiki/Race_condition](en.wikipedia.org/wiki/Race_condition)
Overview of Mutual Exclusion Locks

- A mutual exclusion lock (mutex) defines a “critical section”
  - Ensures only one thread can run in a block of code at a time
  - Other threads are kept “at bay”
  - After a thread leaves a critical section another thread can enter & start running
Overview of Mutual Exclusion Locks

- A mutual exclusion lock (mutex) defines a “critical section”
  - Ensures only one thread can run in a block of code at a time
  - Other threads are kept “at bay”
- After a thread leaves a critical section another thread can enter & start running
  - After acquiring the lock another thread enters the critical section
Overview of Mutual Exclusion Locks

• A mutual exclusion lock (mutex) defines a “critical section”
  • Ensures only one thread can run in a block of code at a time
  • Other threads are kept “at bay”
• After a thread leaves a critical section another thread can enter & start running
  • After acquiring the lock another thread enters the critical section
  • This thread has exclusive access to shared mutable state within critical section
Overview of Mutual Exclusion Locks

- A mutex is typically implemented in hardware via atomic operations

Atomic operations appear to occur instantaneously & either change the state of the system successful or have no effect

See [en.wikipedia.org/wiki/Linearizability](en.wikipedia.org/wiki/Linearizability)
Overview of Mutual Exclusion Locks

- A mutex is typically implemented in hardware via atomic operations.
- Implemented in Java via the `compareAndSwap*()` methods in the Unsafe class.

Concurrency

And few words about concurrency with `Unsafe.compareAndSwap` methods are atomic and can be used to implement high-performance lock-free data structures.

For example, consider the problem to increment value in the shared object using lot of threads.

First we define simple interface `Counter`:

```java
interface Counter {
    void increment();
    long getCount();
}
```

Then we define worker thread `CounterClient`, that uses `Counter`:

```java
class CounterClient implements Runnable {
    private Counter c;
    private int num;

    public CounterClient(Counter c, int num) {
        this.c = c;
        this.num = num;
    }

    @Override
    public void run() {
        for (int i = 0; i < num; i++) {
            c.increment();
        }
    }
}
```

See earlier discussion of "Java Atomic Classes & Operations"
Human Known Use of Mutual Exclusion Locks
Human Known Use of Mutual Exclusion Locks

- A human known use of mutual exclusion locks is an airplane restroom protocol.
Human Known Use of Mutual Exclusion Locks

- A human known use of mutual exclusion locks is an airplane restroom protocol

Person can only enter if the restroom is vacant
Human Known Use of Mutual Exclusion Locks

- A human known use of mutual exclusion locks is an airplane restroom protocol.
Human Known Use of Mutual Exclusion Locks

- A human known use of mutual exclusion locks is an airplane restroom protocol.

Other people who want to use the restroom must wait while it’s in use.
A human known use of mutual exclusion locks is an airplane restroom protocol. This protocol is "fully-bracketed," i.e., person who locks must be the same as the person who unlocks.
A human known use of mutual exclusion locks is an airplane restroom protocol.

Once the restroom is vacant another person can enter.
End of Introduction to Java ReentrantLock