Overview of Java Synchronizers

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 purpose of Java synchronizers
Overview of Java Synchronizers
A Java synchronizer is an object used to control the flow of cooperating threads based on its state.

See [en.wikipedia.org/wiki/Synchronization_(computer_science)](en.wikipedia.org/wiki/Synchronization_(computer_science))
Overview of Java Synchronizers

• Java synchronizers ensure interactions between threads obey certain properties
Overview of Java Synchronizers

- Java synchronizers ensure interactions between threads obey certain properties, e.g.
- Don’t corrupt shared mutable state

See henrikeichenhardt.blogspot.com/2013/06/why-shared-mutable-state-is-root-of-all.html
Overview of Java Synchronizers

- Java synchronizers ensure interactions between threads obey certain properties, e.g.
- Don’t corrupt shared mutable state

```java
class NonAtomicOps {
    long mCounter = 0;

    void increment() {
        // Thread T₁
        for (; ;) mCounter++;  
    }

    void decrement() {
        // Thread T₂
        for (; ;) mCounter--;  
    }

    ...
}
```

Running `increment()` & `decrement()` concurrently yields undefined behavior since `mCounter` is shared mutable data

See tutorials.jenkov.com/java-concurrency/race-conditions-and-critical-sections.html
Overview of Java Synchronizers

- Java synchronizers ensure interactions between threads obey certain properties, e.g.
  - Don’t corrupt shared mutable state

```java
class AtomicOps {
    long mCounter = 0;

    synchronized void increment() {
        // Thread T₁
        for (;;) mCounter++;  // Thread T₂
        for (;;) mCounter--;  // Thread T₃
    }

    synchronized void decrement() {
        // Thread T₄
        for (;;) mCounter--;  // Thread T₅
        for (;;) mCounter++;  // Thread T₆
    }
}
```

Running `increment()` & `decrement()` concurrently yields correct behavior since `mCounter` is shared mutable data synchronized at the (coarse-grained) method level.

See tutorials.jenkov.com/java-concurrency/synchronized.html
Overview of Java Synchronizers

- Java synchronizers ensure interactions between threads obey certain properties, e.g.
  - Don’t corrupt shared mutable state

```java
class AtomicOps {
    long mCounter = 0;

    void increment() { // Thread T₁
        for (;;) synchronized
        { mCounter++; }
    }

    void decrement() { // Thread T₂
        for (;;) synchronized
        { mCounter--; }
    }

    ...
}
```

Running `increment()` & `decrement()` concurrently yields correct behavior since `mCounter` is shared mutable data synchronized at the (fine-grained) statement level

See tutorials.jenkov.com/java-concurrency/synchronized.html
Overview of Java Synchronizers

• Java synchronizers ensure interactions between threads obey certain properties, e.g.
  • Don’t corrupt shared mutable state
  • Occur in the right order, at the right time, & under the right conditions
Overview of Java Synchronizers

- Java synchronizers ensure interactions between threads obey certain properties, e.g.
  - Don’t corrupt shared mutable state
  - Occur in the right order, at the right time, & under the right conditions

The unsynchronized version is buggy

See upcoming lesson on “Java Semaphore: Coordinating Threads”
Overview of Java Synchronizers

- Java synchronizers ensure interactions between threads obey certain properties, e.g.
  - Don’t corrupt shared mutable state
  - Occur in the right order, at the right time, & under the right conditions

```java
% java PlayPingPong
Ready...Set...Go!
Ping!(1)
Pong!(1)
Ping!(2)
Pong!(2)
Ping!(3)
Pong!(3)
Ping!(4)
Pong!(4)
Ping!(5)
Pong!(5)
Pong!(6)
Pong!(6)
Ping!(7)
Pong!(7)
Ping!(8)
Pong!(8)
Pong!(9)
Pong!(9)
Pong!(10)
Done!
```

The synchronized version coordinates the threads properly
Pervasiveness of Synchronizers in Java
Pervasiveness of Java Synchronizer Classes

- Multiple layers of synchronizers are provided on the Java platform
Pervasiveness of Java Synchronizer Classes

- Multiple layers of synchronizers are provided on the Java platform, e.g.
- The Java language contains some features that synchronize threads, e.g., volatile variables & built-in monitor objects

See [en.wikipedia.org/wiki/Java_programming_language](en.wikipedia.org/wiki/Java_programming_language)
### Pervasiveness of Java Synchronizer Classes

- Multiple layers of synchronizers are provided on the Java platform, e.g.
  - The Java language contains some features that synchronize threads
  - Other synchronizers are provided by the Java Class Library

- *E.g., Java atomics, locks, & other synchronizers*

---

#### Additional Frameworks & Languages

- Operating System Kernel
- Applications
- Threading & Synchronization Packages
- Java Execution Environment (e.g., JVM, ART, etc)
- System Libraries

---

#### See [en.wikipedia.org/wiki/Java_Class_Library](en.wikipedia.org/wiki/Java_Class_Library)
Pervasiveness of Java Synchronizer Classes

- We focus about equally on Java synchronization mechanisms & on Java threading mechanisms in this course.
Synchronization complexity arises from coordinating the interactions of entities that run concurrently.
Pervasiveness of Java Synchronizer Classes

- Synchronization complexity arises from coordinating the interactions of entities that run concurrently.

Java’s parallelism frameworks helps eliminate some of this complexity via “divide and conquer”
End of Overview of Java Synchronizers