Types of Java Synchronizer Capabilities

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

- Recognize the types of capabilities provided by Java synchronizers

<table>
<thead>
<tr>
<th>Category</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic operations</td>
<td>An action that effectively happens all at once or not at all</td>
</tr>
<tr>
<td>Mutual exclusion</td>
<td>Allows concurrent access &amp; updates to shared mutable data without race conditions</td>
</tr>
<tr>
<td>Coordination</td>
<td>Ensures computations run properly, e.g., in the right order, at the right time, under the right conditions, etc.</td>
</tr>
<tr>
<td>Barrier synchronization</td>
<td>Ensures that any thread(s) must stop at a certain point &amp; cannot proceed until all other thread(s) reach this barrier</td>
</tr>
</tbody>
</table>
Learning Objectives in this Part of the Lesson

- Recognize the types of capabilities provided by Java synchronizers

<table>
<thead>
<tr>
<th>Category</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic operations</td>
<td>An action that effectively happens all at once or not at all</td>
</tr>
<tr>
<td>Mutual exclusion</td>
<td>Allows concurrent access &amp; updates to shared mutable data without race conditions</td>
</tr>
<tr>
<td>Coordination</td>
<td>Ensures computations run properly, e.g., in the right order, at the right time, under the right conditions, etc.</td>
</tr>
<tr>
<td>Barrier synchronization</td>
<td>Ensures that any thread(s) must stop at a certain point &amp; cannot proceed until all other thread(s) reach this barrier</td>
</tr>
<tr>
<td>Category</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Atomic operations</td>
<td>An action that effectively happens all at once or not at all</td>
</tr>
<tr>
<td>Mutual exclusion</td>
<td>Allows concurrent access &amp; updates to shared mutable data without race conditions</td>
</tr>
<tr>
<td>Coordination</td>
<td>Ensures computations run properly, e.g., in the right order, at the right time, under the right conditions, etc.</td>
</tr>
<tr>
<td>Barrier synchronization</td>
<td>Ensures that any thread(s) must stop at a certain point &amp; cannot proceed until all other thread(s) reach this barrier</td>
</tr>
</tbody>
</table>
Learning Objectives in this Part of the Lesson

• Recognize the types of capabilities provided by Java synchronizers

<table>
<thead>
<tr>
<th>Category</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic operations</td>
<td>An action that effectively happens all at once or not at all</td>
</tr>
<tr>
<td>Mutual exclusion</td>
<td>Allows concurrent access &amp; updates to shared mutable data without race conditions</td>
</tr>
<tr>
<td>Coordination</td>
<td>Ensures computations run properly, e.g., in the right order, at the right time, under the right conditions, etc.</td>
</tr>
<tr>
<td>Barrier synchronization</td>
<td>Ensures that any thread(s) must stop at a certain point &amp; cannot proceed until all other thread(s) reach this barrier</td>
</tr>
</tbody>
</table>
## Learning Objectives in this Part of the Lesson

- Recognize the types of capabilities provided by Java synchronizers

<table>
<thead>
<tr>
<th>Category</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic operations</td>
<td>An action that effectively happens all at once or not at all</td>
</tr>
<tr>
<td>Mutual exclusion</td>
<td>Allows concurrent access &amp; updates to shared mutable data without race conditions</td>
</tr>
<tr>
<td>Coordination</td>
<td>Ensures computations run properly, e.g., in the right order, at the right time, under the right conditions, etc.</td>
</tr>
<tr>
<td>Barrier synchronization</td>
<td>Ensures that any thread(s) must stop at a certain point &amp; cannot proceed until all other thread(s) reach this barrier</td>
</tr>
</tbody>
</table>
Types of Java Synchronizer Capabilities
## Types of Java Synchronizer Capabilities

- Java synchronizers provide various types of capabilities

<table>
<thead>
<tr>
<th>Category</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic operations</td>
<td>An action that effectively happens all at once or not at all</td>
</tr>
<tr>
<td>Mutual exclusion</td>
<td>Allows concurrent access &amp; updates to shared mutable data without race conditions</td>
</tr>
<tr>
<td>Coordination</td>
<td>Ensures computations run properly, e.g., in the right order, at the right time, under the right conditions, etc.</td>
</tr>
<tr>
<td>Barrier synchronization</td>
<td>Ensures that any thread(s) must stop at a certain point &amp; cannot proceed until all other thread(s) reach this barrier</td>
</tr>
</tbody>
</table>
Java synchronizers provide various types of capabilities, e.g.

- **Atomic ordering**
  - Ensures an action happens all at once or not at all

See [en.wikipedia.org/wiki/Linearizability](en.wikipedia.org/wiki/Linearizability)
Types of Java Synchronizer Capabilities

- Java synchronizers provide various types of capabilities, e.g.
  - **Atomic ordering**
    - Ensures an action happens all at once or not at all
    - Operations on a field in thread\(_1\) occur all at once wrt operations on the field in thread\(_2\ldots n\)

<table>
<thead>
<tr>
<th></th>
<th>Thread(_1)</th>
<th>Thread(_2)</th>
<th>Long field</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialized</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>read field</td>
<td></td>
<td>← 0</td>
<td></td>
</tr>
<tr>
<td>increase field by 1</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>write back</td>
<td></td>
<td>→ 1</td>
<td></td>
</tr>
<tr>
<td>read field</td>
<td></td>
<td>← 1</td>
<td></td>
</tr>
<tr>
<td>increase field by 1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>write back</td>
<td></td>
<td>→ 2</td>
<td></td>
</tr>
</tbody>
</table>

*Atomicity does not occur on primitive Java data types without using synchronizers*

See [docs.oracle.com/javase/tutorial/essential/concurrency/atomic.html](https://docs.oracle.com/javase/tutorial/essential/concurrency/atomic.html)
Java synchronizers provide various types of capabilities, e.g.

- **Atomic ordering**
  - Ensures an action happens all at once or not at all
  - Operations on a field in thread\(_1\) occur all at once wrt operations on the field in thread\(_2\ldots n\)
  - Atomic ordering is supported by the Java atomic package

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/package-summary.html](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/package-summary.html)
Java synchronizers provide various types of capabilities, e.g.

- **Atomic ordering**
  - Ensures an action happens all at once or not at all
  - Operations on a field in thread\(_1\) occur all at once wrt operations on the field in thread\(_2\ldots n\)
  - Atomic ordering is supported by the Java atomic package
  - Atomic ordering is also supported by the Java volatile type qualifier

The volatile type qualifier ensures a variable is read from & written to main memory & not cached

See [en.wikipedia.org/wiki/Volatile_variable#In_Java](en.wikipedia.org/wiki/Volatile_variable#In_Java)
Types of Java Synchronizer Capabilities

- Java synchronizers provide various types of capabilities, e.g.
  - **Atomic ordering**
  - **Mutual exclusion**
    - Prevents simultaneous access to a shared resource in a critical section

See [en.wikipedia.org/wiki/Mutual_exclusion](en.wikipedia.org/wiki/Mutual_exclusion)
Race conditions occur when a program depends on the sequence or timing of threads for it to operate properly.

Types of Java Synchronizer Capabilities

- Atomic ordering
- Mutual exclusion
  - Prevents simultaneous access to a shared resource in a critical section

See en.wikipedia.org/wiki/Race_condition#Software
Java synchronizers provide various types of capabilities, e.g.

- **Atomic ordering**
- **Mutual exclusion**
  - Prevents simultaneous access to a shared resource in a critical section
- **Read/write conflicts**
  - If one thread reads while another thread writes concurrently, the field that’s read may be inconsistent

See [en.wikipedia.org/wiki/Read-write_conflict](http://en.wikipedia.org/wiki/Read-write_conflict)

<table>
<thead>
<tr>
<th>Time</th>
<th>Thread\textsubscript{1}</th>
<th>Thread\textsubscript{2}</th>
<th>Long field</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialized</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>read field</td>
<td></td>
<td>←</td>
<td>0</td>
</tr>
<tr>
<td>increase field by 1</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>write back</td>
<td></td>
<td></td>
<td>0 or 1?</td>
</tr>
<tr>
<td>read field</td>
<td></td>
<td>←</td>
<td>0 or 1?</td>
</tr>
</tbody>
</table>

*Two operations conflict if at least one is a write*
Java synchronizers provide various types of capabilities, e.g.

- **Atomic ordering**
- **Mutual exclusion**
  - Prevents simultaneous access to a shared resource in a critical section
  - Read/write conflicts
  - Write/write conflicts
    - If two threads try to write to same field concurrently, the result may be inconsistent

<table>
<thead>
<tr>
<th>Time</th>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Long field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>initialized</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>read field</td>
<td>← 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>increase field by 2</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>increase field by 1</td>
<td>↑ 0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>write back</td>
<td>write back</td>
<td>1 or 2?</td>
</tr>
</tbody>
</table>

This can yield a "lost update"

See [en.wikipedia.org/wiki/Write-write_conflict](en.wikipedia.org/wiki/Write-write_conflict)
Java synchronizers provide various types of capabilities, e.g.

- **Atomic ordering**
- **Mutual exclusion**
  - Prevents simultaneous access to a shared resource in a critical section
  - Read/write conflicts
  - Write/write conflicts

See [en.wikipedia.org/wiki/Memory_ordering](en.wikipedia.org/wiki/Memory_ordering)

These problems often occur in multi-core processors with "weak" memory ordering due to core caches that allow "out-of-order" load & store operations
Java synchronizers provide various types of capabilities, e.g.

- Atomic ordering
- **Mutual exclusion**
  - Prevents simultaneous access to a shared resource in a critical section
  - Read/write conflicts
  - Write/write conflicts
- Mutual exclusion is supported by the Java locks package
  - e.g., ReentrantLock, ReentrantReadWriteLock, StampedLock, etc.

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/package-summary.html](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/package-summary.html)
Java synchronizers provide various types of capabilities, e.g.

- **Atomic ordering**
- **Mutual exclusion**
  - Prevents simultaneous access to a shared resource in a critical section
  - Read/write conflicts
  - Write/write conflicts
  - Mutual exclusion is supported by the Java locks package
  - Mutual exclusion is also supported by the `synchronized` keyword in Java built-in monitor objects

Java synchronizers provide various types of capabilities, e.g.

- **Atomic ordering**
- **Mutual exclusion**

**Coordination**
- Ensures computations run properly
Types of Java Synchronizer Capabilities

- Java synchronizers provide various types of capabilities, e.g.
  - Atomic ordering
  - Mutual exclusion
  - Coordination
    - Ensures computations run properly, e.g.
      - In the right order

See [github.com/douglasraigschmidt/LiveLessons/tree/master/PingPongApplication](https://github.com/douglasraigschmidt/LiveLessons/tree/master/PingPongApplication)
Types of Java Synchronizer Capabilities

- Java synchronizers provide various types of capabilities, e.g.
  - Atomic ordering
  - Mutual exclusion
  - **Coordination**
    - Ensures computations run properly, e.g.
      - In the right order
      - At the right time

See [en.wikipedia.org/wiki/Real-time_computing](en.wikipedia.org/wiki/Real-time_computing)
Types of Java Synchronizer Capabilities

- Java synchronizers provide various types of capabilities, e.g.
  - Atomic ordering
  - Mutual exclusion
  - Coordination
    - Ensures computations run properly, e.g.
      - In the right order
      - At the right time
      - Under the right conditions

See github.com/douglascraigschmidt/LiveLessons/tree/master/PalantiriManagerApplication
Java synchronizers provide various types of capabilities, e.g.

- Atomic ordering
- Mutual exclusion
- **Coordination**
  - Ensures computations run properly
- Coordination is supported by the Java concurrent & locks packages
  - e.g., ConditionObject, Semaphore, etc.

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html)
Java synchronizers provide various types of capabilities, e.g.

- Atomic ordering
- Mutual exclusion
- **Coordination**
  - Ensures computations run properly
  - Coordination is supported by the Java concurrent & locks packages
  - Coordination is also supported by Java built-in monitor objects

Types of Java Synchronizer Capabilities

- Java synchronizers provide various types of capabilities, e.g.
  - Atomic ordering
  - Mutual exclusion
  - Coordination
  - **Barrier synchronization**
    - Ensures that any thread(s) must stop at a certain point & cannot proceed until all thread(s) reach the barrier

Barrier synchronization is a variant of coordination
Java synchronizers provide various types of capabilities, e.g.

- Atomic ordering
- Mutual exclusion
- Coordination

### Barrier synchronization

- Ensures that any thread(s) must stop at a certain point & cannot proceed until all thread(s) reach the barrier

- Barrier synchronization is supported by the Java concurrent package
  - e.g., CountDownLatch, CyclicBarrier, Phaser, etc.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html
We’ll cover all these types of Java synchronizers in this course!!

<table>
<thead>
<tr>
<th>Category</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic operations</td>
<td>An action that effectively happens all at once or not at all</td>
</tr>
<tr>
<td>Mutual exclusion</td>
<td>Allows concurrent access &amp; updates to shared mutable data without race conditions</td>
</tr>
<tr>
<td>Coordination</td>
<td>Ensures computations run properly, e.g., in the right order, at the right time, under the right conditions, etc.</td>
</tr>
<tr>
<td>Barrier synchronization</td>
<td>Ensures that any thread(s) must stop at a certain point &amp; cannot proceed until all other thread(s) reach this barrier</td>
</tr>
</tbody>
</table>
End of Types of Java Synchronizer Capabilities