Overview of Java Synchronized & Concurrent Collections

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 Lesson

• Be aware of Java’s synchronized & concurrent collections
Overview of Java Synchronized Collections
Overview of Java Synchronized Collections

• By default, Java collections are not synchronized

Note that this implementation is not synchronized. If multiple threads access an ArrayList instance concurrently, and at least one of the threads modifies the list structurally, it must be synchronized externally.

See docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html
Overview of Java Synchronized Collections

- By default, Java collections are not synchronized
- Thus, they are not thread-safe

Code is thread-safe if it only manipulates shared data structures in a manner that avoids race conditions by multiple concurrent threads

See en.wikipedia.org/wiki/Thread_safety
Overview of Java Synchronized Collections

- Java provides synchronized collection wrappers via static factory methods in the Collections class

<table>
<thead>
<tr>
<th>Collections Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>synchronizedCollection(coll)</td>
</tr>
<tr>
<td>synchronizedList(list)</td>
</tr>
<tr>
<td>synchronizedMap(map)</td>
</tr>
<tr>
<td>synchronizedSet(set)</td>
</tr>
</tbody>
</table>

See docs.oracle.com/javase/tutorial/collections/implementations/wrapper.html
Java provides synchronized collection wrappers via static factory methods in the Collections class, e.g.

- Ensure that method calls are thread-safe

```java
Map<Integer, String> mMap = new HashMap<>();

mMap = Collections.synchronizedMap(mMap);

Thread t1:
    mMap.put(1, "Newton");
    mMap.put(4, "Favre");
    mMap.put(7, "Elway");
    mMap.put(12, "Brady");
    mMap.put(13, "Warner");
    mMap.put(18, "Manning");

Thread t2:
    String s1 = mMap.get(12);

Thread t3:
    String s2 = mMap.get(13);

Thread t4:
    String s3 = mMap.get(18);
```
Overview of Java Synchronized Collections

- Java provides synchronized collection wrappers via static factory methods in the Collections class, e.g.
  ```java
  public class Collections {
    public static <K,V> Map<K,V> synchronizedMap (Map<K,V> m) {
      return new SynchronizedMap<>(m);
    }
  }
  ```

- Ensure that method calls are thread-safe
- Synchronized collections aren’t optimized for concurrent access

  e.g., the Map parameter is simply wrapped by a SynchronizedMap
Overview of Java Synchronized Collections

- Java provides synchronized collection wrappers via static factory methods in the Collections class, e.g.

  ```java
public class Collections {
    public static <K,V> Map<K,V> synchronizedMap
        (Map<K,V> m) {
        return new SynchronizedMap<>(m);
    }

    class SynchronizedMap<K,V> implements Map<K,V> {
        private final Map<K,V> m; // Backing Map
        final Object mutex; // Object on which to synchronize

        SynchronizedMap(Map<K,V> m)
        { this.m = Objects.requireNonNull(m); mutex = this; }

        public V get(Object key)
        { synchronized (mutex) { return m.get(key); } }
    }
  }
  ```

- Ensure that method calls are thread-safe

- Synchronized collections aren’t optimized for concurrent access

A synchronized collection is thread-safe & governed by a single exclusion lock
Overview of Java Synchronized Collections

- Java provides synchronized collection wrappers via static factory methods in the Collections class, e.g.
  ```java
class Collections {
    public static <K,V> Map<K,V> synchronizedMap(Map<K,V> m) {
      return new SynchronizedMap<>(m);
    }
}
```

- Ensure that method calls are thread-safe
- Synchronized collections aren’t optimized for concurrent access

```java
class SynchronizedMap<K,V> implements Map<K,V> ...
{
  private final Map<K,V> m; // Backing Map
  final Object mutex; // Object on which to synchronize

  SynchronizedMap(Map<K,V> m)
  {
    this.m = Objects.requireNonNull(m); mutex = this;
  }

  public V get(Object key)
  {
    synchronized (mutex) {
      return m.get(key);
    }
  }
}
```

Implemented by decorating each method in a synchronized block

See en.wikipedia.org/wiki/Decorator_pattern
Overview of Java Concurrent Collections
Overview of Java Concurrent Collections

- Java concurrent collections provide features that are optimized for the needs of concurrent programs

These are the concurrent-aware interfaces:

- BlockingQueue
- TransferQueue
- BlockingDeque
- ConcurrentHashMap
- ConcurrentMap
- ConcurrentNavigableMap

Concurrent-aware classes include:

- LinkedBlockingQueue
- ArrayBlockingQueue
- PriorityBlockingQueue
- DelayQueue
- SynchronousQueue
- LinkedBlockingDeque
- LinkedTransferQueue
- CopyOnWriteArrayList
- CopyOnWriteArraySet
- ConcurrentHashMap

See docs.oracle.com/javase/tutorial/essential/concurrency/collections.html
Overview of Java Concurrent Collections

• Java concurrent collections provide features that are optimized for the needs of concurrent programs
• A concurrent collection is thread-safe, but is not governed by just a single exclusion lock

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html
Overview of Java Concurrent Collections

- Java concurrent collections provide features that are optimized for the needs of concurrent programs
  - A concurrent collection is thread-safe, but is not governed by just a single exclusion lock
- They avoid *memory consistency errors* by defining a “happens-before” relationship
  - e.g., between a thread that adds an object to the collection with subsequent thread(s) that access or remove that object

See [docs.oracle.com/javase/tutorial/essential/concurrency/memconsist.html](docs.oracle.com/javase/tutorial/essential/concurrency/memconsist.html)
Overview of Java Concurrent Collections

- **ConcurrentHashMap**
  - Concurrent retrievals & adjustable expected concurrent updates via OO & functional programming APIs

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html)
Overview of Java Concurrent Collections

- **ConcurrentHashMap**
  - Concurrent retrievals & adjustable expected concurrent updates via OO & functional programming APIs
  - Optimized for multi-core CPUs

Overview of Java Concurrent Collections

- **ConcurrentHashMap**
  - Concurrent retrievals & adjustable expected concurrent updates via OO & functional programming APIs
  - Optimized for multi-core CPUs
  - Provides “atomic check-then-act” methods

See [dig.cs.illinois.edu/papers/checkThenAct.pdf](dig.cs.illinois.edu/papers/checkThenAct.pdf)
Overview of Java Concurrent Collections

- **ConcurrentHashMap**
  - Concurrent retrievals & adjustable expected concurrent updates via OO & functional programming APIs
  - Optimized for multi-core CPUs
  - Provides “atomic check-then-act” methods, e.g.
    - Replaces entry for a key only if currently mapped to some value

Instead of

```java
if (map.containsKey(key))
    return map.put(key, value);
```

use

```java
return map.replace(key, value);
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html#replace](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html#replace)
Overview of Java Concurrent Collections

- **ConcurrentHashMap**
  - Concurrent retrievals & adjustable expected concurrent updates via OO & functional programming APIs
  - Optimized for multi-core CPUs
  - Provides “atomic check-then-act” methods, e.g.
    - Replaces entry for a key only if currently mapped to some value
    - Replaces entry for a key only if currently mapped to a given value

Instead of

```java
if (map.containsKey(key) & oldValue == newValue)
    return map.put(key, newValue);
```

use

```java
return map.replace(key, oldValue, newValue);
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html#replace](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html#replace)
Overview of Java Concurrent Collections

- **ConcurrentHashMap**
  - Concurrent retrievals & adjustable expected concurrent updates via OO & functional programming APIs
  - Optimized for multi-core CPUs
  - Provides “atomic check-then-act” methods, e.g.
    - Replaces entry for a key only if currently mapped to some value
    - Replaces entry for a key only if currently mapped to a given value
    - If specified key isn’t already associated with a value, associate it with the given value

  Instead of
  ```java
  if (map.get(key) == null)
      return map.put(key, value);
  else
      return map.get(key);
  ```

  Use
  ```java
  return map.putIfAbsent(key, value);
  ```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html#putIfAbsent](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html#putIfAbsent)
Overview of Java Concurrent Collections

- **ConcurrentHashMap**
  - Concurrent retrievals & adjustable expected concurrent updates via OO & functional programming APIs
  - Optimized for multi-core CPUs
  - Provides “atomic check-then-act” methods, e.g.
    - Replaces entry for a key only if currently mapped to some value
    - Replaces entry for a key only if currently mapped to a given value
    - If specified key isn’t already associated with a value, associate it with the given value
    - If specified key isn’t already associated with a value, attempt to compute its value using the given mapping function & enter it into this map

Instead of

```java
if (map.get(key) == null) {
    V newValue = mappingFunc.apply(key);
    if (newValue != null)
        map.put(key, newValue);
}
```

use

```java
return map.computeIfAbsent(key, k -> new Value(f(k)));
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html#computeIfAbsent](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html#computeIfAbsent)
Overview of Java Concurrent Collections

*BlockingQueue*

- Define FIFO objects that block or timeout when adding to a full queue or retrieving from an empty queue

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html)
Overview of Java Concurrent Collections

*BlockingQueue*

- Define FIFO objects that block or timeout when adding to a full queue or retrieving from an empty queue.
End of Overview of Java Synchronized & Concurrent Collections