

Introduction to Java

Synchronized 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

- Recognize the capabilities & limits of Java's synchronized collections

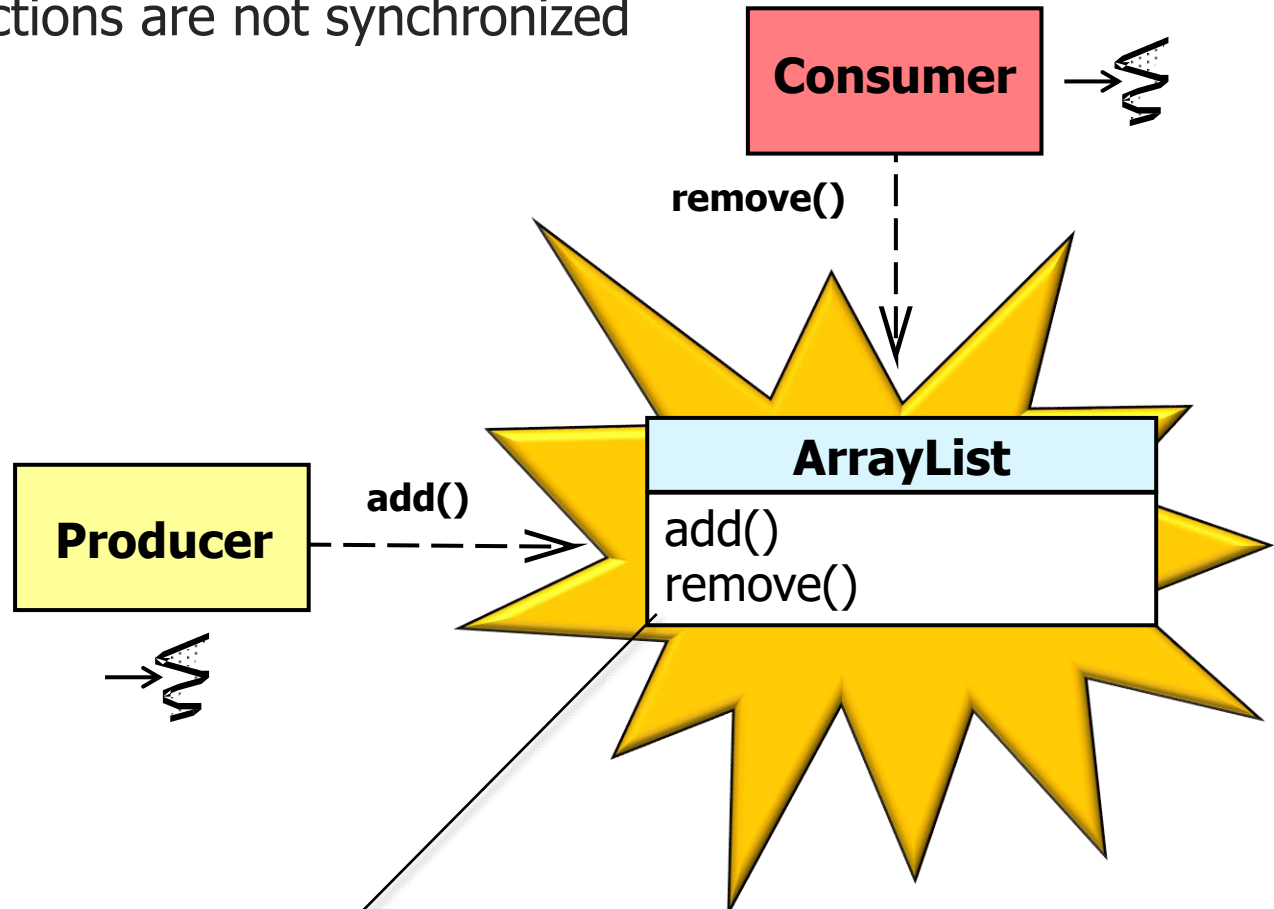
Collections Method
<code>synchronizedCollection(coll)</code>
<code>synchronizedList(list)</code>
<code>synchronizedMap(map)</code>
<code>synchronizedSet(set)</code>



Overview of Java Synchronized Collections

Overview of Java Synchronized Collections

- By default, Java collections are not synchronized

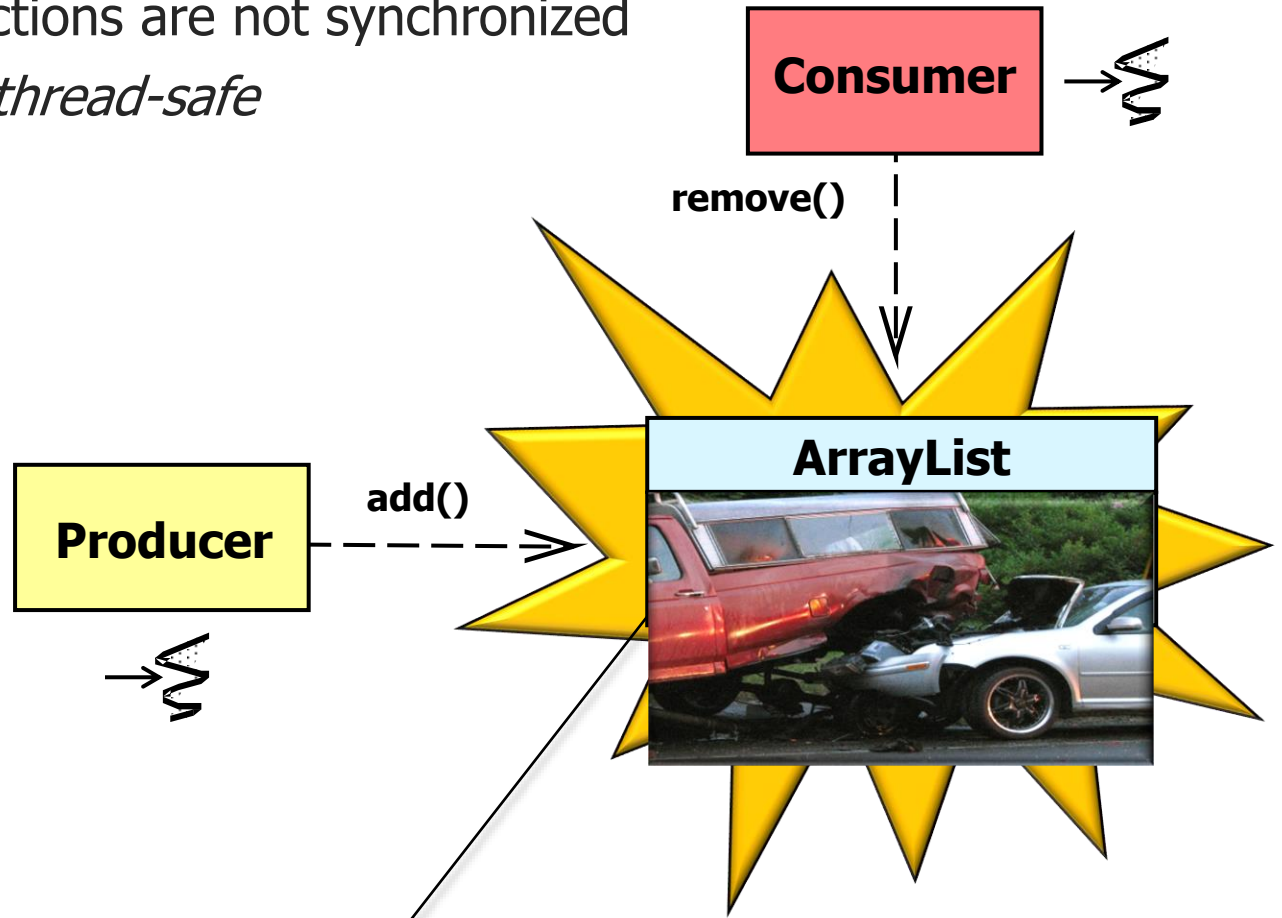


Note that this implementation is not synchronized. If multiple threads access an ArrayList instance concurrently, and at least one of the threads modifies the list structurally, it must be synchronized externally

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's synchronized collection *wrappers* are created via static factory methods

Collections Method
<code>synchronizedCollection(coll)</code>
<code>synchronizedList(list)</code>
<code>synchronizedMap(map)</code>
<code>synchronizedSet(set)</code>

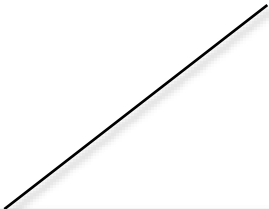


See docs.oracle.com/javase/tutorial/collections/implementations/wrapper.html

Overview of Java Synchronized Collections

- Java's synchronized collection *wrappers* are created via static factory methods, e.g.
- Ensure that method calls are *thread-safe*

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



e.g., the Map parameter is simply wrapped by a SynchronizedMap

Overview of Java Synchronized Collections

- Java's synchronized collection *wrappers* are created via static factory methods, e.g.
- Ensure that method calls are *thread-safe*

This factory method converts a non-thread-safe map into a thread-safe map via the synchronization wrapper

```
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's synchronized collection *wrappers* are created via static factory methods, e.g.
- Ensure that method calls are *thread-safe*

```
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);
```

*Multiple threads can thus
access & update the
synchronized collection*

Overview of Java Synchronized Collections

- Java's synchronized collection *wrappers* are created via static factory methods, e.g.
 - Ensure that method calls are *thread-safe*
 - Synchronized collections aren't optimized for concurrent access

A synchronized collection is thread-safe & governed by one mutual exclusion lock



```
class SynchronizedMap<K,V>
    implements Map<K,V> ... {
    // Backing Map
    private final Map<K,V> m;
    // Synchronizer object
    final Object mutex;

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

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

Overview of Java Synchronized Collections

- Java's synchronized collection *wrappers* are created via static factory methods, e.g.
 - Ensure that method calls are *thread-safe*
 - Synchronized collections aren't optimized for concurrent access

*Implemented by decorating each forwarded method in a **synchronized** statement*



```
class SynchronizedMap<K,V>
    implements Map<K,V> ... {
    // Backing Map
    private final Map<K,V> m;
    // Synchronizer object
    final Object mutex;

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

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

See en.wikipedia.org/wiki/Decorator_pattern

Overview of Java Synchronized Collections

- Java's synchronized collection *wrappers* are created via static factory methods, e.g.
 - Ensure that method calls are *thread-safe*
 - Synchronized collections aren't optimized for concurrent access

A single mutual exclusion lock can yield excessive contention



```
class SynchronizedMap<K,V>
    implements Map<K,V> ... {
    // Backing Map
    private final Map<K,V> m;
    // Synchronizer object
    final Object mutex;

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

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

See www.ibm.com/support/knowledgecenter/en/SS3KLZ/com.ibm.java.diagnostics.healthcenter.doc/topics/resolving.html

End of Introduction to Java Synchronized Collections