

#	Source		Pattern	
1	<pre>LeafQueue leafQueue = ...; -synchronized (leafQueue) {      57 LOC  }</pre>	<pre>LeafQueue leafQueue = ...; +try { +  leafQueue.getReadLock().lock();     57 LOC +} finally { +  leafQueue.getReadLock().unlock(); +}</pre>	<pre>synchronized (obj) {     ... }</pre>	<pre>try {     lock.lock();     ... } finally {     lock.unlock(); }</pre>
2	<pre>-Lock readlock = classLoaderContainerMapLock.readLock(); -try { -  readlock.lock(); -  result = classLoaderContainerMap.get(tccl); -} finally { -  readlock.unlock(); -} -if (result == null) { -  Lock writelock = classLoaderContainerMapLock.writeLock(); -  try { -    writeLock.lock(); -    result = classLoaderContainerMap.get(tccl); -    if (result == null) { -      result = new ServerContainerImpl(); -      classLoaderContainerMap.put(tccl, result); -    } -  } finally { -    writeLock.unlock(); -  } -}</pre>	<pre>+synchronized (classLoaderContainerMapLock) {      result = classLoaderContainerMap.get(tccl);     if (result == null) {         result = new ServerContainerImpl();         classLoaderContainerMap.put(tccl, result);     }  }</pre>	<pre>try {     readLock.lock();     read operations } finally {     readLock.unlock(); } try {     writeLock.lock();     write operations } finally {     writeLock.unlock(); }</pre>	<pre>synchronized {     all operations }</pre>
3	<pre>private static final Object lock = new Object(); private Map&lt;...&gt; count = new HashMap&lt;&gt;(); -synchronized (count) { -  Pair&lt;Job, String&gt; key = -    new ImmutablePair&lt;&gt;(jobID, name);  -  if (count.containsKey(key)) { -    count.put(key, count.get(key) + 1); -  } else { -    count.put(key, 1); -  } -}</pre>	<pre>private static final Object lock = new Object(); private Map&lt;...&gt; count = new HashMap&lt;&gt;(); +synchronized(lock) +  if (!jobCounts.containsKey(jobID)) { +    jobCounts.put(jobID, new HashMap&lt;&gt;()); +  } +  Map&lt;String, Integer&gt; count = jobCounts.get(jobID); +  if (count.containsKey(name)) { +    count.put(name, count.get(name) + 1); +  } else { +    count.put(name, 1); +  } +}</pre>	<pre>synchronized (obj1) {     ... }</pre>	<pre>synchronized (obj2) {     ... }</pre>
4	<pre>-public synchronized void reset() {      map.clear();     members = EMPTY_MEMBERS;  }</pre>	<pre>+private final Object membersLock = new Object(); +public void reset() { +  synchronized (membersLock) { +    map.clear(); +    members = EMPTY_MEMBERS; +  } +}</pre>	<pre>synchronized void foo() {     ... }</pre>	<pre>void foo() {     synchronized (obj) {         ...     } }</pre>
5	<pre>-public boolean isAccessed() { -  return this.accessed; -}</pre>	<pre>+public synchronized boolean isAccessed() { +  return this.accessed; +}</pre>	<pre>void foo() {     ... }</pre>	<pre>synchronized void foo() {     ... }</pre>
6	<pre>-synchronized (this.channelLookup) { -  try{ -    lookupResponse = AkkaUtils. -      &lt;JobManagerMessages.ConnectionInformation&gt;ask(channelLookup, -        new JobManagerMessages.LookupConnectionInformation( -          connectionInfo, jobID, sourceChannelID), timeout).response(); -  }catch(IOException ioe) { -    throw ioe; -  } -}</pre>	<pre>lookupResponse = AkkaUtils.   &lt;JobManagerMessages.ConnectionInformation&gt;ask(channelLookup,     new JobManagerMessages.LookupConnectionInformation(       connectionInfo, jobID, sourceChannelID), timeout).response();</pre>	<pre>synchronized (obj) {     ... }</pre>	<pre>...</pre>
7				
8	<pre>synchronized (buffers) {   if (...) { -    if (spillWriter != null) { -      spillWriter.close(); -    }     isFinished = true;   } }</pre>	<pre>synchronized (buffers) {   if (...) {     isFinished = true;   } +if (spillWriter != null) { +  spillWriter.close(); +}</pre>	<pre>synchronized (obj) {     statements1     statements2 }</pre>	<pre>synchronized (obj) {     statements1 } statements2</pre>
9				
10	<pre>-protected Membership membership = null; public boolean hasMembers() {   if (membership == null) setupMembership();   return membership.hasMembers(); } protected synchronized void setupMembership() {   if ( membership == null ) {     membership = new Membership(super.getLocalMember(true));   } }</pre>	<pre>+protected volatile Membership membership = null; public boolean hasMembers() {   if (membership == null) setupMembership();   return membership.hasMembers(); } protected synchronized void setupMembership() {   if ( membership == null ) {     membership = new Membership(super.getLocalMember(true));   } }</pre>	<pre>T foo;</pre>	<pre>volatile T foo;</pre>
11	<pre>-private volatile int requestCount;  -private volatile int errorCount;  -  requestCount++;</pre>	<pre>+private final AtomicInteger requestCount = +  new AtomicInteger(0); +private final AtomicInteger errorCount = new AtomicInteger(0);  +  requestCount.incrementAndGet();</pre>	<pre>volatile T foo;</pre>	<pre>TT foo;</pre>