

## Übung Java Lambda LockUtil

# Ausgangslage

Die folgenden Listings zeigen ein Lockable Interface, die Klasse LockUtil und ein Beispielpogramm. LockUtil bietet mit der Methode runLocked() die synchrone und damit concurrent Ausführung der implementierten Methode Lockable.runLocked(). Der synchrone Mechanismus basiert auf dem Concurrent Lock-Interface (e.g. ReentrantLock). Die Lock-Instanz wird durch die Methode

```

Lockable.getLock() zur Verfügung gestellt:package
ch.std.jp2.lambda.lockutil.classic;&#xA;&#xA;import
java.util.concurrent.locks.Lock;&#xA;&#xA;public interface Lockable&amp;lt;E&amp;gt; {&#xA;&#xA;
public abstract Lock getLock();&#xA; public abstract E runLocked() throws Exception;&#xA;
&#xA;}&#xA;}&#xA;package ch.std.jp2.lambda.lockutil.classic;&#xA;&#xA;public class LockException extends
Exception {&#xA;&#xA; private static final long serialVersionUID =
5619158779338557977L;&#xA;&#xA; public LockException() {&#xA; super();&#xA; }&#xA;&#xA;
public LockException(String message) {&#xA; super(message);&#xA; }&#xA;&#xA;}&#xA;}&#xA;package
ch.std.jp2.lambda.lockutil.classic;&#xA;&#xA;import java.util.concurrent.locks.Lock;&#xA;import
java.util.concurrent.locks.ReentrantLock;&#xA;&#xA;public class LockUtil {&#xA;&#xA; public static
Lock newLockInstance() {&#xA; return new ReentrantLock(); &#xA; }&#xA; &#xA; /**&#xA; * This
methods waits until given lock is free.&#xA; * &#xA; * There is no timeout support and a deadlock
may occure.&#xA; * &#xA; * @param lockable&#xA; */&#xA; public static &amp;lt;E&amp;gt; E
runLocked(Lockable&amp;lt;E&amp;gt; lockable) {&#xA; Lock lock = lockable.getLock();&#xA; if
(lock == null) {&#xA; throw new NullPointerException(&#34;lock is null&#34;);&#xA; }&#xA; E obj
= null;&#xA; try {&#xA; lock.lock();&#xA; obj = lockable.runLocked();&#xA; } catch (Exception e)
{&#xA; e.printStackTrace();&#xA; } finally {&#xA; lock.unlock();&#xA; }&#xA; return obj;&#xA;
}&#xA;&#xA;}&#xA;}&#xA;package ch.std.jp2.lambda.lockutil.classic;&#xA;&#xA;import
java.util.concurrent.locks.Lock;&#xA;&#xA;class TestLockable implements
Lockable&amp;lt;String&amp;gt; {&#xA;&#xA; private Lock testLock;&#xA; &#xA; public
TestLockable() {&#xA; this.testLock = LockUtil.newLockInstance();&#xA; }&#xA; &#xA;
@Override&#xA; public Lock getLock() {&#xA; return this.testLock;&#xA; }&#xA;&#xA;
@Override&#xA; public String runLocked() {&#xA;
System.out.println(&#34;TestLockable.runLocked()&#34;);&#xA; try {&#xA;
Thread.sleep(5000);&#xA; } catch (InterruptedException e) {&#xA; e.printStackTrace();&#xA;
}&#xA; return &#34;test&#34;;&#xA; }&#xA; &#xA;}&#xA;&#xA;}&#xA;&#xA;public class TestLockUtil
{&#xA;&#xA; public static void main(String[] args) {&#xA;
System.out.println(&#34;begin&#34;);&#xA; String result = LockUtil.runLocked(new
TestLockable());&#xA; System.out.println(&#34;result = &#34; + result);&#xA;
System.out.println(&#34;done&#34;);&#xA; }&#xA;}&#xA;}&#xA;

```

## Vorlage

Die Dateien der Vorlage finden Sie auch hier

## Aufgabe

Die Vorlage soll nun durch eine funktionale Lösung basierend auf Lambda Expressions umprogrammiert werden. Laden Sie die Klassen der Vorlage in Ihre Entwicklungsumgebung (Template) und führen Sie das Programm aus. Kopieren Sie das Package und planen Sie das Refactoring für die Lambda Lösung. Beachten Sie auch den möglichen Einsatz des Callable Interfaces.

## Lösung Lockable FunctionInterface

Eine mögliche Lösung mit Lockable FunctionalInterface finden Sie hier

## Lösung Callable FunctionInterface

Eine mögliche Lösung mit Callable FunctionalInterface finden Sie hier

## Kontakt

Simtech AG  
Finkenweg 23  
3110 Münsingen  
Schweiz

## Impressum

Das Copyright für sämtliche Inhalte dieser Website liegt bei Simtech AG, Schweiz.  
Beachten Sie auch unsere Hinweise zum Urheberrecht, Datenschutz und Haftungsausschluss.  
Jeder Hinweis auf Fehler nehmen wir gerne entgegen.

## Copyright

2024 Simtech AG, All rights reserved, Powered by stack.ch written in Golang by Daniel Schmutz

ldung-java-kurs-java-advanced-kurs-java-advanced---ressourcen-kurs-java-advanced---übungen-lamb