Übung Java Lambda LockUtil

Ausgangslage

Die folgenden Listings zeigen ein Lockable Interface, die Klasse LockUtil und ein Beispielprogramm. 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 gesellt:package ch.std.jpf2.lambda.lockutil.classic;

import java.util.concurrent.locks.Lock;

public interface Lockable<E> {

 public abstract Lock getLock();
 public abstract E runLocked() throws Exception;

}package ch.std.jpf2.lambda.lockutil.classic;

public class LockException extends Exception {

 private static final long serialVersionUID = 5619158779338557977L;

 public LockException() {
  super();
 }

 public LockException(String message) {
  super(message);
 }

}package ch.std.jpf2.lambda.lockutil.classic;

import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class LockUtil {

 public static Lock newLockInstance() {
  return new ReentrantLock();
 }

 /**
  * This methods waits until given lock is free.
  *
  * There is no timeout support and a deadlock may occure.
  *
  * @param lockable
  */
 public static <E> E runLocked(Lockable<E> lockable) {
  Lock lock = lockable.getLock();
  if (lock == null) {
  throw new NullPointerException("lock is null");
  }
  E obj = null;
  try {
  lock.lock();
  obj = lockable.runLocked();
  } catch (Exception e) {
  e.printStackTrace();
  } finally {
  lock.unlock();
  }
  return obj;
 }

}package ch.std.jpf2.lambda.lockutil.classic;

import java.util.concurrent.locks.Lock;

class TestLockable implements Lockable<String> {

 private Lock testLock;

 public TestLockable() {
  this.testLock = LockUtil.newLockInstance();
 }

 @Override
 public Lock getLock() {
  return this.testLock;
 }

 @Override
 public String runLocked() {
 System.out.println("TestLockable.runLocked()");
  try {
 Thread.sleep(5000);
  } catch (InterruptedException e) {
  e.printStackTrace();
 }
  return "test";
 }

}

public class TestLockUtil {

 public static void main(String[] args) {
 System.out.println("begin");
  String result = LockUtil.runLocked(new TestLockable());
  System.out.println("result = " + result);
 System.out.println("done");
 }
}

Vorlage

Die Dateien der Vorlage finden Sie auch hier

Aufgabe

Die Vorlage soll nun durch ein 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 Haftungsauschluss.
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

https://www.simtech-ag.ch/kurs-java-advanced-ressourcen-lockutil