

The LockManager App Case Study: Test Driver Implementation & Behavior

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

**Institute for Software
Integrated Systems**

**Vanderbilt University
Nashville, Tennessee, USA**



Learning Objectives in this Part of the Lesson

- Understand the implementation of the LockManagerTest class & associated client code that invoke synchronous methods on the LockManagerController

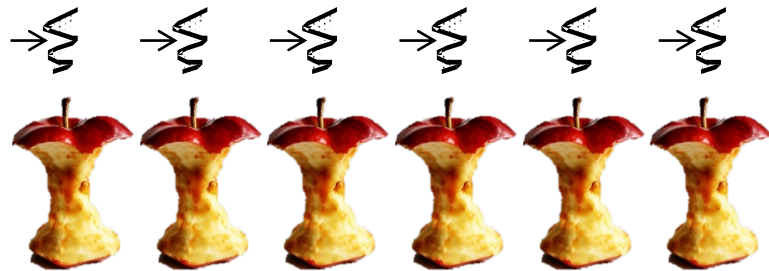
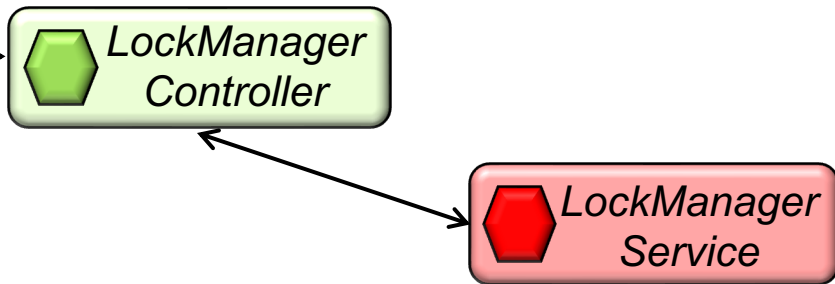
LockManagerTest

```
18 //see
19
20 //see
21 * This program tests the PrimeCheckClient and its ability to
22 * communicate with the PrimeCheckServerController.
23 //see
24
25 @SpringBootTest
26 @ContextConfiguration(classes = {
27     Components.class,
28     PrimeCheckClient.class,
29     PrimeCheckController.class
30 })
31 public class PrimeCheckTest {
32     //see
33     * Debugging tag used by the logger.
34     private final String TAG = getClass().getSimpleName();
35
36     //see
37     * This object connects to the TestClient. The @Autowired
38     * annotation ensures this field is initialized via Spring
39     * dependency injection, where an object receives another object
40     * it depends on (e.g., by creating a @Inject PrimeCheckClient).
41 }
```



*Synchronous
HTTP POST
requests/
responses*

LockManagerApplication



Overview of the LockManagerTest Driver

Overview of the LockManagerTest Driver

- This SpringBootTest class exercises all the features of the LockManager Application microservice

LockManagerTests		
f	🔒	mLockAPI LockAPI
f	🔒	sPERMIT_COUNT int
m	🔒	testMultipleAcquireAndRelease() void
m	🔒	testSingleAcquireAndRelease() void
m	🔒	testSingleLock(int, LockManager) void
m	🔒	testMultipleLocks(int, LockManager, int) void

See www.baeldung.com/spring-boot-testing

Overview of the LockManagerTest Driver

- This `SpringBootTest` class exercises all the features of the LockManager Application microservice
 - It loads the application context & tests the interaction between client & server components

LockManagerTests			
f	🔒	mLockAPI	LockAPI
f	🔒	sPERMIT_COUNT	int
m	🔒	testMultipleAcquireAndRelease()	void
m	🔒	testSingleAcquireAndRelease()	void
m	🔒	testSingleLock(int, LockManager)	void
m	🔒	testMultipleLocks(int, LockManager, int)	void

```
19 // ...
20 // ...
21 // This program tests the PrimeCheckClient and its ability to
22 // communicate with the PrimeCheckServerController.
23 // ...
24
25 @SpringBootTest
26 @ContextConfiguration(classes = {
27     Components.class,
28     PrimeCheckClient.class,
29     PrimeCheckController.class
30 })
31 public class PrimeCheckTest {
32     // Debugging tag used by the Logger.
33     // ...
34     private final String TAG = getClass().getSimpleName();
35
36     // ...
37     // This object connects to the TestClient. The @Autowired
38     // annotation ensures this field is initialized via Spring
39     // dependency injection, where an object receives another object
40     // it depends on (e.g., by creating a @Link PrimeCheckClient!).
41 }
```

**Synchronous
HTTP POST
requests/
responses**



LockManager
Controller

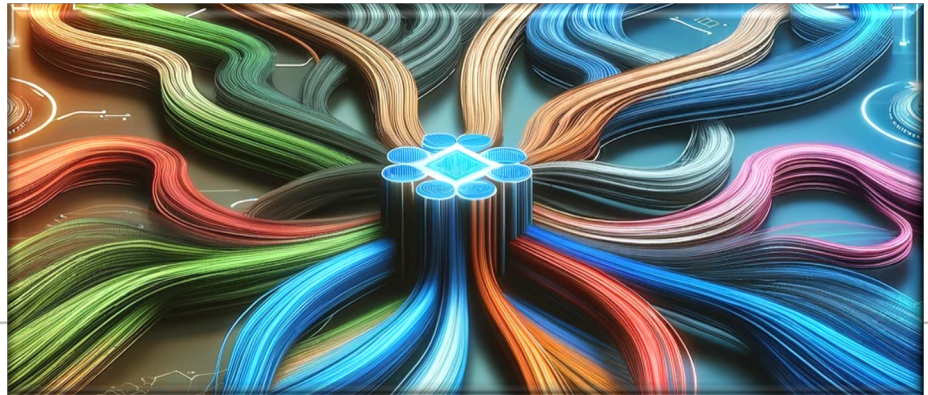


LockManager
Service

Overview of the LockManagerTest Driver

- This SpringBootTest class exercises all the features of the LockManager Application microservice
 - It loads the application context & tests the interaction between client & server components
- Its @Test-annotated methods validate the behavior of the LockManagerApplication under concurrent scenarios where multiple processes/threads interact via shared resources

LockManagerTests		
f	mLockAPI	LockAPI
f	sPERMIT_COUNT	int
m	testMultipleAcquireAndRelease()	void
m	testSingleAcquireAndRelease()	void
m	testSingleLock(int, LockManager)	void
m	testMultipleLocks(int, LockManager, int)	void



Overview of the LockManagerTest Driver

- The testSingleAcquireAndRelease() method
- Ensure multiple client threads can acquire & release a single lock concurrently from a LockManager

```
int maxLocks = 2;
int maxClients = 4;

var lockManager = mLockAPI
    .create(maxLocks);

IntStream
    .range(0, maxClients)
    .parallel()
    .forEach(client ->
        testSingleLock
            (client,
             lockManager));
```

Overview of the LockManagerTest Driver

- The testSingleAcquireAndRelease() method
- Ensure multiple client threads can acquire & release a single lock concurrently from a LockManager

*Create a LockManager
with a permit count of 2*

```
int maxLocks = 2;
int maxClients = 4;

var lockManager = mLockAPI
    .create(maxLocks);

IntStream
    .range(0, maxClients)
    .parallel()
    .forEach(client ->
        testSingleLock
            (client,
             lockManager));
```


Overview of the LockManagerTest Driver

- The `testSingleAcquireAndRelease()` method
- Ensure multiple client threads can acquire & release a single lock concurrently from a LockManager

```
int maxLocks = 2;
int maxClients = 4;

var lockManager = mLockAPI
    .create(maxLocks);

IntStream
    .range(0, maxClients)
    .parallel()
    .forEach(client ->
        testSingleLock
            (client,
             lockManager));
```

Execute test operations for 4 clients concurrently using parallel streams

Overview of the LockManagerTest Driver

- The `testSingleAcquireAndRelease()` method
- Ensure multiple client threads can acquire & release a single lock concurrently from a `LockManager`

```
int maxLocks = 2;
int maxClients = 4;

var lockManager = mLockAPI
    .create(maxLocks);

IntStream
    .range(0, maxClients)
    .parallel()
    .forEach(client ->
        testSingleLock
            (client,
             lockManager));
```

Each client calls the `testSingleLock()` method, which tests acquiring & releasing a single lock

Overview of the LockManagerTest Driver

- The testSingleLock() method
 - Acquire & release a single Lock object on a given client

```
void testSingleLock
(int client,
 LockManager lockManager) {

    var lock = mLockAPI
        .acquire(lockManager);

    var result = mLockAPI
        .release(lockManager,
            lock);

}
```

Overview of the LockManagerTest Driver

- The testSingleLock() method
 - Acquire & release a single Lock object on a given client

```
void testSingleLock
(int client,
 LockManager lockManager) {

    var lock = mLockAPI
        .acquire(lockManager);
    ...

    var result = mLockAPI
        .release(lockManager,
            lock);

}
```

Invoke remote acquire() & release() methods on the LockManagerController via the generated LockAPI proxy

Overview of the LockManagerTest Driver

- The testMultipleAcquireAndRelease() method
 - Ensure multiple client threads can acquire & release multiple locks concurrently from a LockManager

```
int maxLocks = 4;  
int maxClients = 8;  
int maxPermits = 2;
```

```
var lockManager = mLockAPI  
    .create(maxLocks);
```

```
IntStream
```

```
    .range(0, maxClients)  
    .parallel()  
    .forEach(client ->  
        testMultipleLocks  
            (client,  
             lockManager,  
             maxPermits));
```

Overview of the LockManagerTest Driver

- The testMultipleAcquireAndRelease() method
 - Ensure multiple client threads can acquire & release multiple locks concurrently from a LockManager

*Create a LockManager
with a permit count of 4*

```
int maxLocks = 4;  
int maxClients = 8;  
int maxPermits = 2;  
  
var lockManager = mLockAPI  
    .create(maxLocks);
```

```
IntStream  
    .range(0, maxClients)  
    .parallel()  
    .forEach(client ->  
        testMultipleLocks  
            (client,  
             lockManager,  
             maxPermits));
```

Overview of the LockManagerTest Driver

- The testMultipleAcquireAndRelease() method
 - Ensure multiple client threads can acquire & release multiple locks concurrently from a LockManager

```
int maxLocks = 4;  
int maxClients = 8;  
int maxPermits = 2;  
  
var lockManager = mLockAPI  
    .create(maxLocks);
```

IntStream

```
.range(0, maxClients)  
.parallel()  
.forEach(client ->  
    testMultipleLocks  
        (client,  
         lockManager,  
         maxPermits));
```

Execute test operations for 8 clients concurrently using parallel streams

Overview of the LockManagerTest Driver

- The testMultipleAcquireAndRelease() method
 - Ensure multiple client threads can acquire & release multiple locks concurrently from a LockManager

```
int maxLocks = 4;
int maxClients = 8;
int maxPermits = 2;

var lockManager = mLockAPI
    .create(maxLocks);

IntStream
    .range(0, maxClients)
    .parallel()
    .forEach(client ->
        testMultipleLocks
            (client,
             lockManager,
             maxPermits));
```

Each client calls the testMultipleLocks() method, which tests acquiring & releasing multiple permits

Overview of the LockManagerTest Driver

- The testMultipleLocks() method
 - Acquire & release multiple Lock objects on a given client

```
void testMultipleLocks
(int client,
 LockManager lockManager,
 int maxPermits) {

    var locks = mLockAPI
        .acquire(lockManager,
                maxPermits);
    ...

    var result = mLockAPI
        .release(lockManager,
                locks);

}
```

Overview of the LockManagerTest Driver

- The testMultipleLocks() method
 - Acquire & release multiple Lock objects on a given client

```
void testMultipleLocks
(int client,
 LockManager lockManager,
 int maxPermits) {

    var locks = mLockAPI
        .acquire(lockManager,
                 maxPermits);

    ...

    var result = mLockAPI
        .release(lockManager,
                 locks);

}
```

Invoke remote acquire() & release() methods on the LockManagerController via the generated LockAPI proxy

Implementing the LockManagerTest Driver

Implementing the LockManagerTest Driver

```
ex5 src test java edu vandy lockmanager LockManagerTests
Project
  ex5 ~/Dropbox/Documents/LiveLesson
    .gradle
    .idea
    build
    gradle
    src
      main
        java
        resources
          application.properties
      test
        java
          edu.vandy.lockmanager
            client
              ClientBeans
              LockAPI
            LockManagerTests
          resources
            application.properties
    .gitignore
    build.gradle
    gradlew
    gradlew.bat
    settings.gradle
  External Libraries
  Scratches and Consoles
Structure
17 /**
18  * This program tests the features of the {@link
19  * LockManagerApplication} microservice, which uses Spring WebMVC to
20  * provide a distributed lock manager for synchronous Spring client
21  * applications using quasi-asynchronous controller methods that
22  * return Spring {@link DeferredResult} objects. It also shows how to
23  * use the synchronous HTTP interface features in Spring framework 6,
24  * which enables the definition of declarative HTTP services using
25  * Java interfaces.
26  */
27 @SpringBootTest(classes = LockManagerApplication.class,
28                 webEnvironment = SpringBootTest
29                 .WebEnvironment.DEFINED_PORT)
30 class LockManagerTests {
31     /**
32     * The auto-wired {@link LockAPI} that accesses the {@link
33     * LockManagerApplication} microservice
```

See [WebMVC/ex5/src/test/java/edu/vandy/lockmanager/LockManagerTests.java](https://github.com/lockmanager/ex5/src/test/java/edu/vandy/lockmanager/LockManagerTests.java)

End of the LockManager App Case Study: Test Driver Implementation & Behavior