The LockManager App Case Study: Overview

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

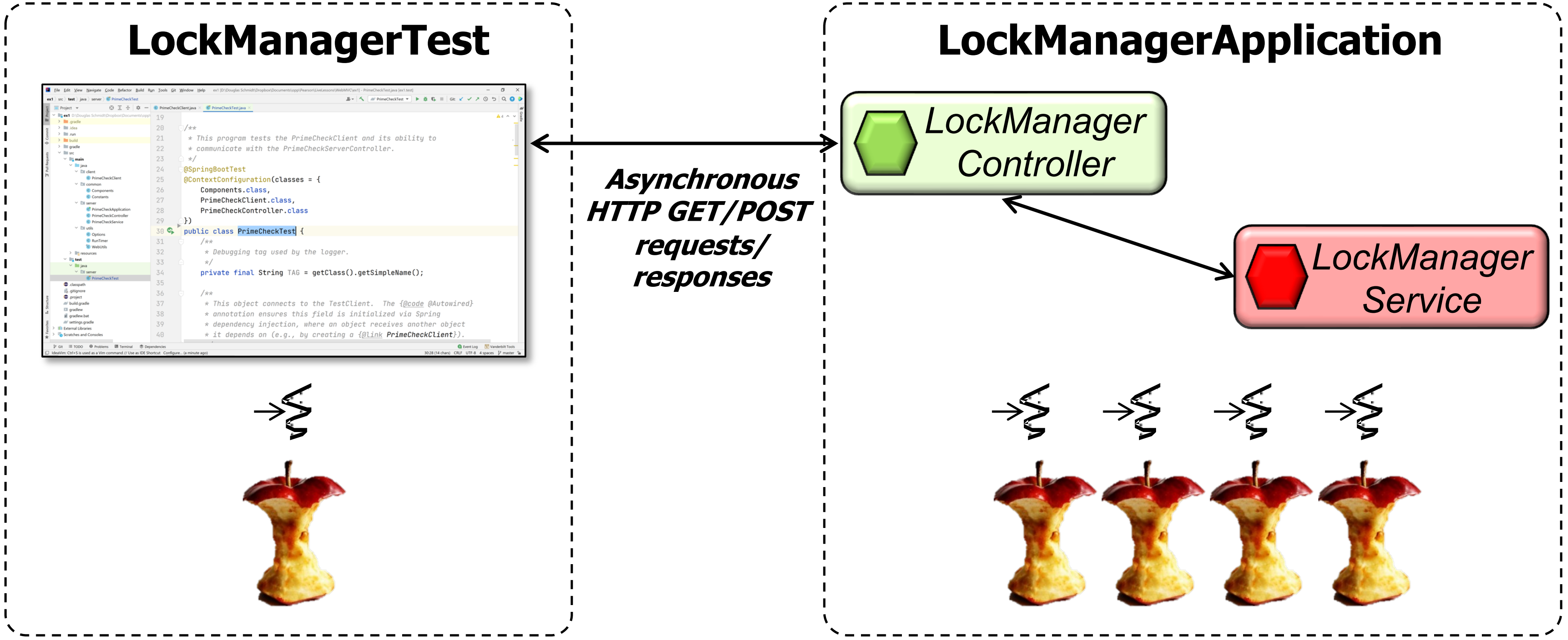
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand how Spring WebFlux sends/receives HTTP GET & POST requests asynchronously to/from a microservice that provides a distributed semaphore



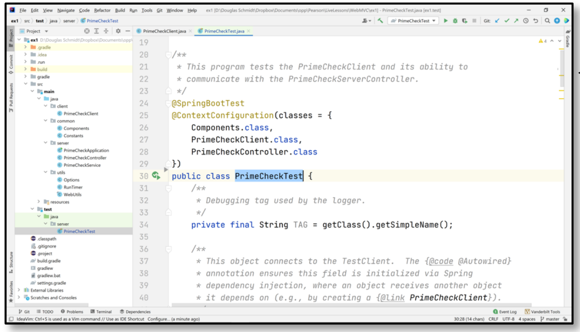
```
/**
 * This program tests the PrimeCheckClient and its ability to
 * communicate with the PrimeCheckServerController.
 */
@SpringBootTest
@ContextConfiguration(classes = {
    Components.class,
    PrimeCheckClient.class,
    PrimeCheckController.class
})
public class PrimeCheckTest {
    /**
     * Debugging tag used by the logger.
     */
    private final String TAG = getClass().getSimpleName();

    /**
     * This object connects to the TestClient.  The {@code @Autowired}
     * annotation ensures this field is initialized via Spring
     * dependency injection, where an object receives another object
     * it depends on (e.g., by creating a {@link PrimeCheckClient}).
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/WebFlux/ex1

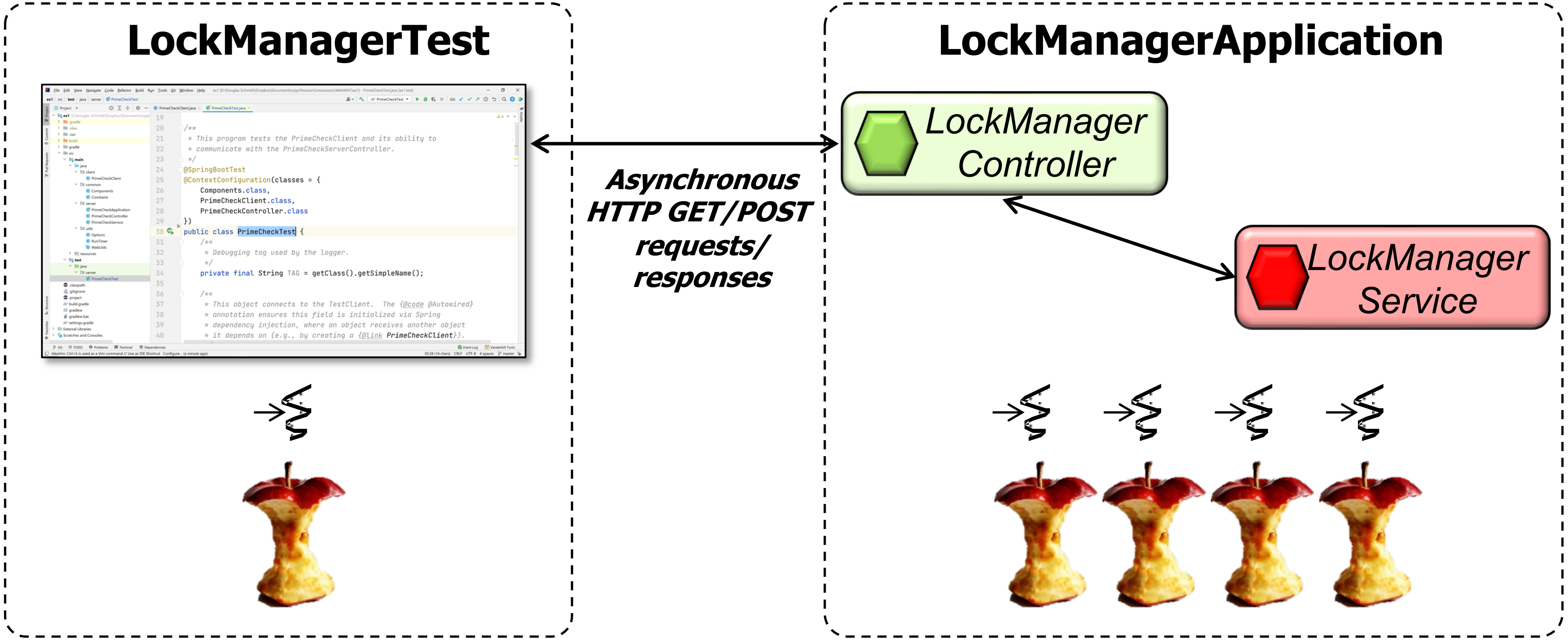
Overview of the Lock Manager App Case Study

Overview of the LockManager App Case Study

- This case study shows how Spring WebFlux sends & receives HTTP GET/POST requests asynchronously to/from a LockManager microservice



```java

/**
 * This program tests the PrimeCheckClient and its ability to
 * communicate with the PrimeCheckServerController.
 */
@SpringBootTest
@ContextConfiguration(classes = {
    Components.class,
    PrimeCheckClient.class,
    PrimeCheckController.class
})
public class PrimeCheckTest {
    /**
     * Debugging tag used by the logger.
     */
    private final String TAG = getClass().getSimpleName();

    /**
     * This object connects to the TestClient.  The {@code @Autowired}
     * annotation ensures this field is initialized via Spring
     * dependency injection, where an object receives another object
     * it depends on (e.g., by creating a {@link PrimeCheckClient}).
```

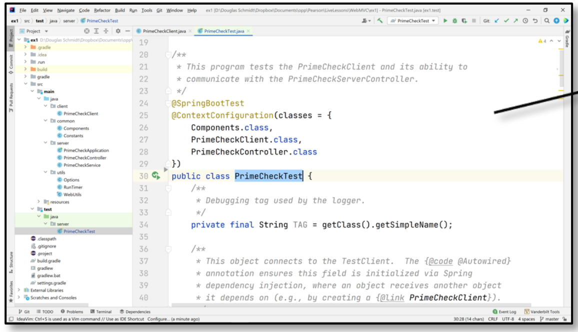


See hgithub.com/douglascraigschmidt/LiveLessons/tree/master/WebFlux/ex1

Overview of the LockManager App Case Study

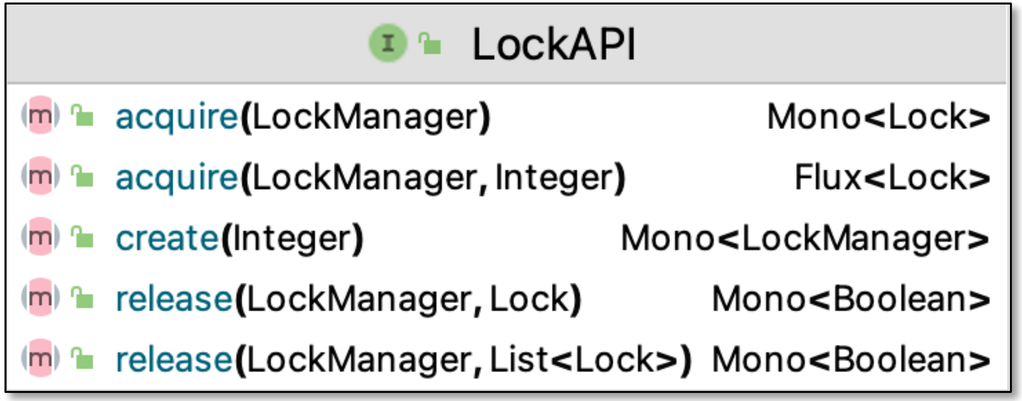
- This case study shows how Spring WebFlux sends & receives HTTP GET/POST requests asynchronously to/from a LockManager microservice

LockManagerTest



```
/**
 * This program tests the PrimeCheckClient and its ability to
 * communicate with the PrimeCheckServerController.
 */
@SpringBootTest
@ContextConfiguration(classes = {
    Components.class,
    PrimeCheckClient.class,
    PrimeCheckController.class
})
public class PrimeCheckTest {
    /**
     * Debugging tag used by the logger.
     */
    private final String TAG = getClass().getSimpleName();

    /**
     * This object connects to the TestClient.  The {@code @Autowired}
     * annotation ensures this field is initialized via Spring
     * dependency injection, where an object receives another object
     * it depends on (e.g., by creating a {@link PrimeCheckClient}).
```

The client can asynchronously acquire & release locks individually or in bulk via the declarative LockAPI interface



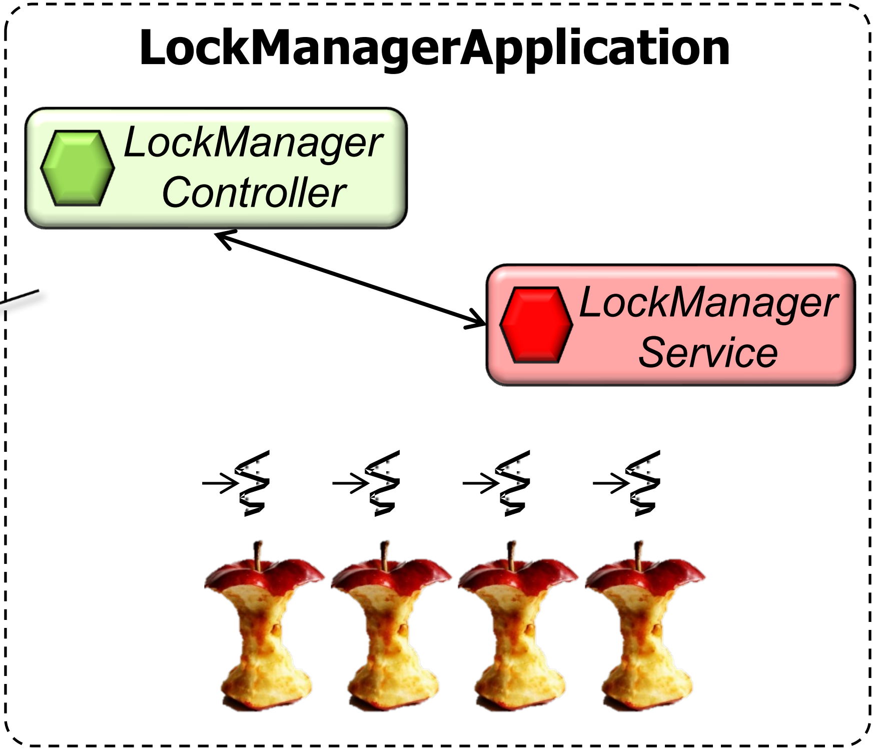
LockAPI

acquire(LockManager)	Mono<Lock>
acquire(LockManager, Integer)	Flux<Lock>
create(Integer)	Mono<LockManager>
release(LockManager, Lock)	Mono<Boolean>
release(LockManager, List<Lock>)	Mono<Boolean>

See WebFlux/ex1/src/test/java/edu/vandy/lockmanager/LockManagerTests.java

Overview of the LockManager App Case Study

- This case study shows how Spring WebFlux sends & receives HTTP GET/POST requests asynchronously to/from a LockManager microservice



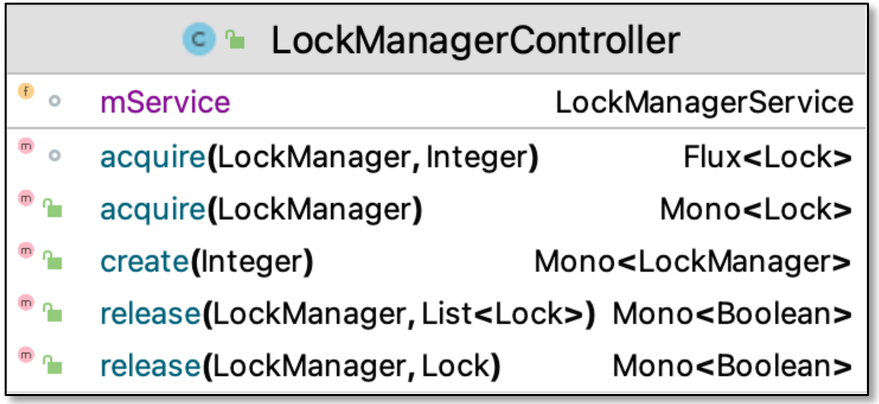
The server (microservice) can acquire & release locks individually or in bulk

See WebFlux/ex1/src/main/java/edu/vandy/lockmanager/server

Overview of the LockManager App Case Study

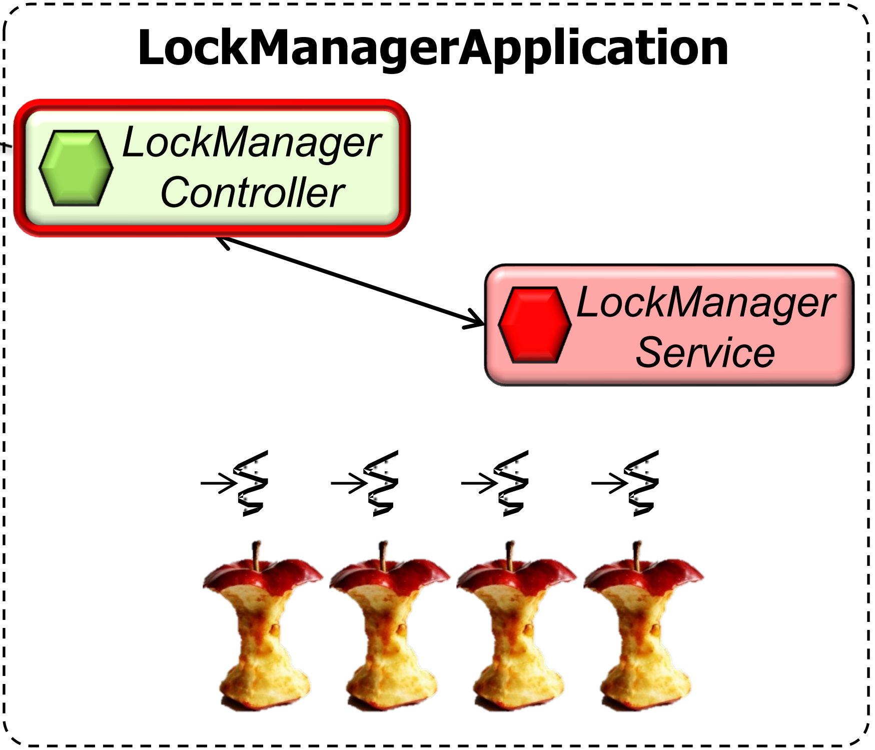
- This case study shows how Spring WebFlux sends & receives HTTP GET/POST requests asynchronously to/from a LockManager microservice

LockManagerController converts HTTP GET/POST requests into Java types & forwards them to LockManagerService



LockManagerController

mService	LockManagerService
acquire(LockManager, Integer)	Flux<Lock>
acquire(LockManager)	Mono<Lock>
create(Integer)	Mono<LockManager>
release(LockManager, List<Lock>)	Mono<Boolean>
release(LockManager, Lock)	Mono<Boolean>

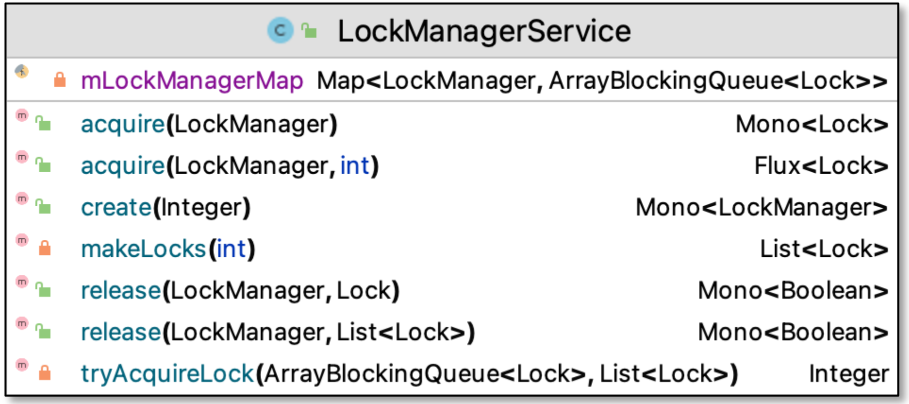


See WebFlux/ex1/src/main/java/edu/vandy/lockmanager/server/LockManagerController.java

Overview of the LockManager App Case Study

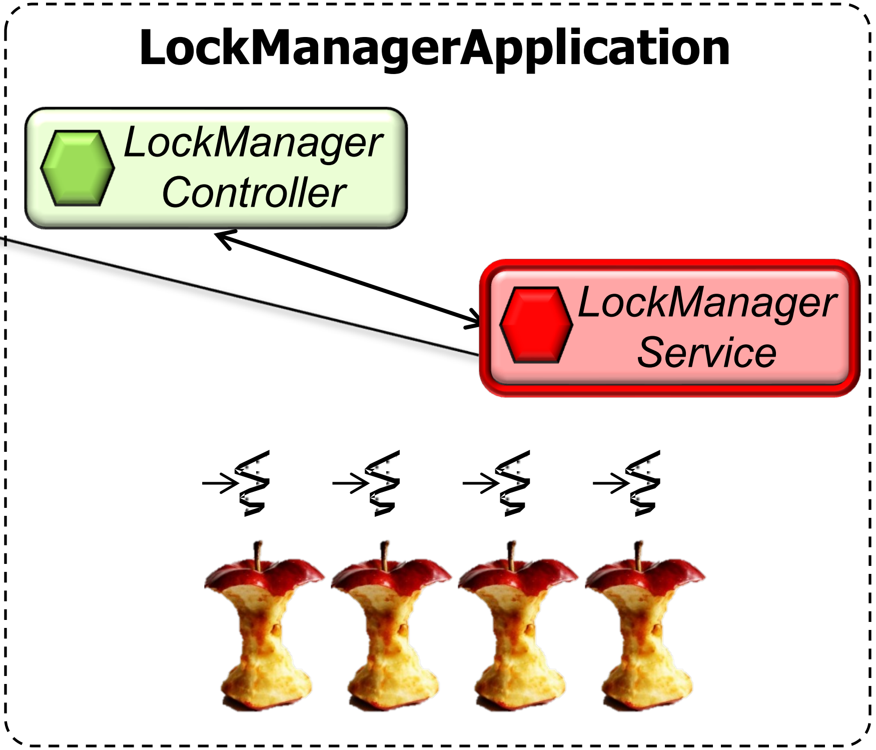
- This case study shows how Spring WebFlux sends & receives HTTP GET/POST requests asynchronously to/from a LockManager microservice

LockManagerService uses an ArrayBlockingQueue, reactive programming, & virtual threads to implement a distributed semaphore



LockManagerService

mLockManagerMap	Map<LockManager, ArrayBlockingQueue<Lock>>
acquire(LockManager)	Mono<Lock>
acquire(LockManager, int)	Flux<Lock>
create(Integer)	Mono<LockManager>
makeLocks(int)	List<Lock>
release(LockManager, Lock)	Mono<Boolean>
release(LockManager, List<Lock>)	Mono<Boolean>
tryAcquireLock(ArrayBlockingQueue<Lock>, List<Lock>)	Integer

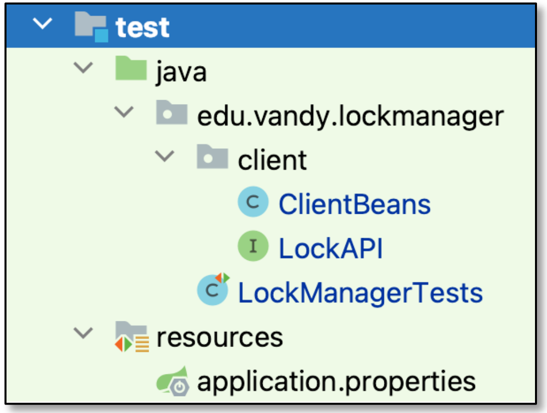


See WebFlux/ex1/src/main/java/edu/vandy/lockmanager/server/LockManagerService.java

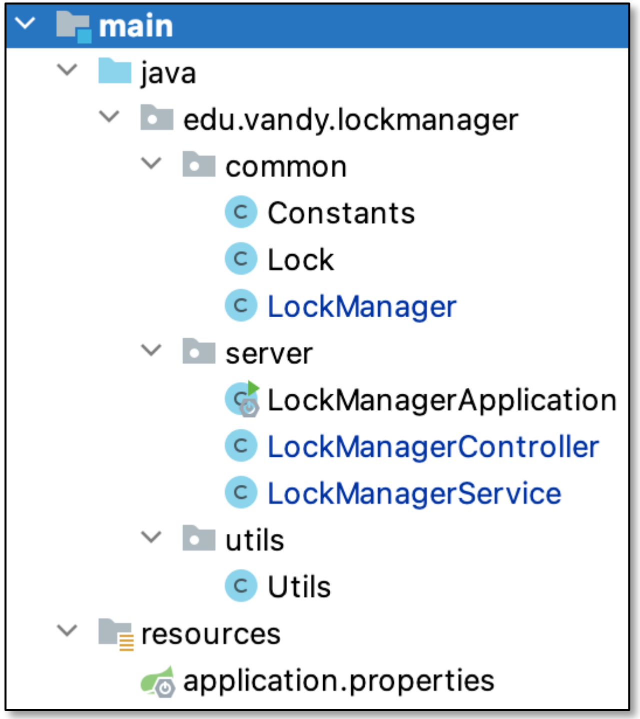
Structure of the Lock Manager App Project

Structure of the LockManager App Project

- The LockManager App project source code is organized into several packages



```
test
  java
    edu.vandy.lockmanager
      client
        ClientBeans
        LockAPI
      LockManagerTests
  resources
    application.properties
```

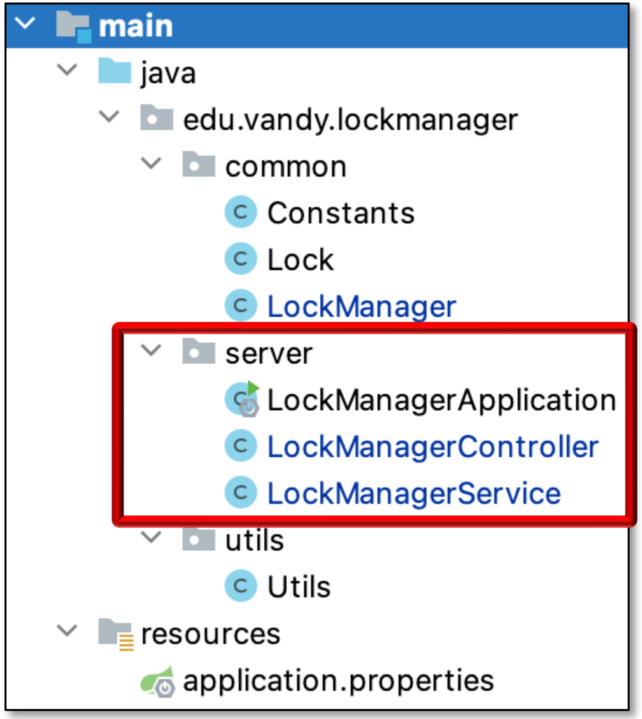


```
main
  java
    edu.vandy.lockmanager
      common
        Constants
        Lock
        LockManager
      server
        LockManagerApplication
        LockManagerController
        LockManagerService
      utils
        Utils
  resources
    application.properties
```

See hgithub.com/douglascraigschmidt/LiveLessons/tree/master/WebFlux/ex1

Structure of the LockManager App Project

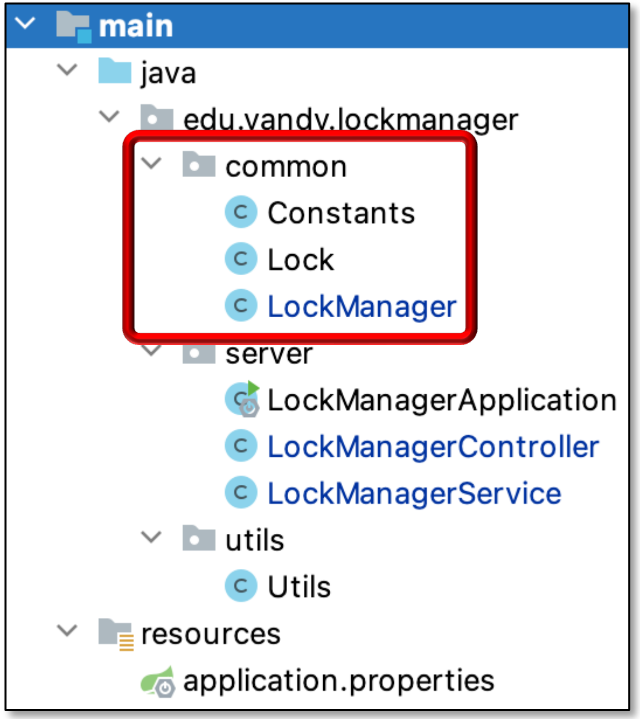
- The LockManager App project source code is organized into several packages
  - main
    - server
      - Contains the "app" entry point, the controller, & the service
        - This implementation uses reactive programming & reactive types



```
main
  java
    edu.vandy.lockmanager
      common
        Constants
        Lock
        LockManager
      server
        LockManagerApplication
        LockManagerController
        LockManagerService
      utils
        Utils
  resources
    application.properties
```

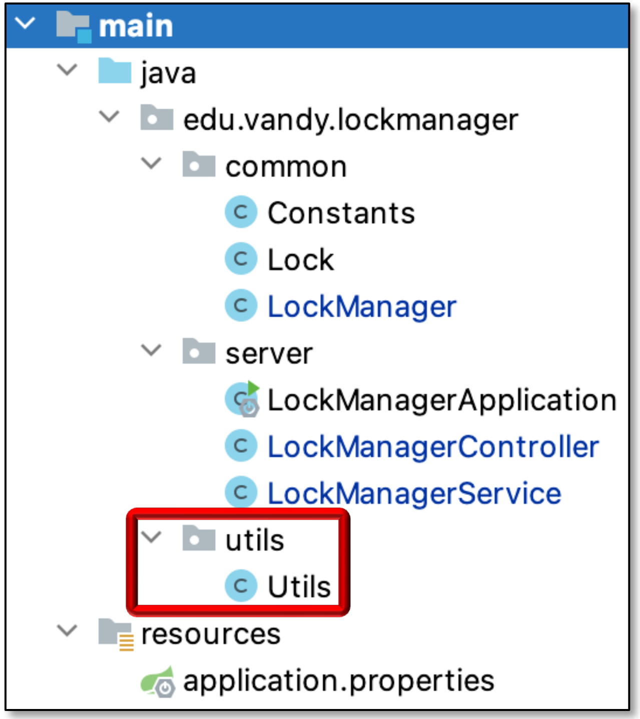
Structure of the LockManager App Project

- The LockManager App project source code is organized into several packages
  - main
    - server
    - common
      - Consolidates various project-specific helper classes, including the Lock object



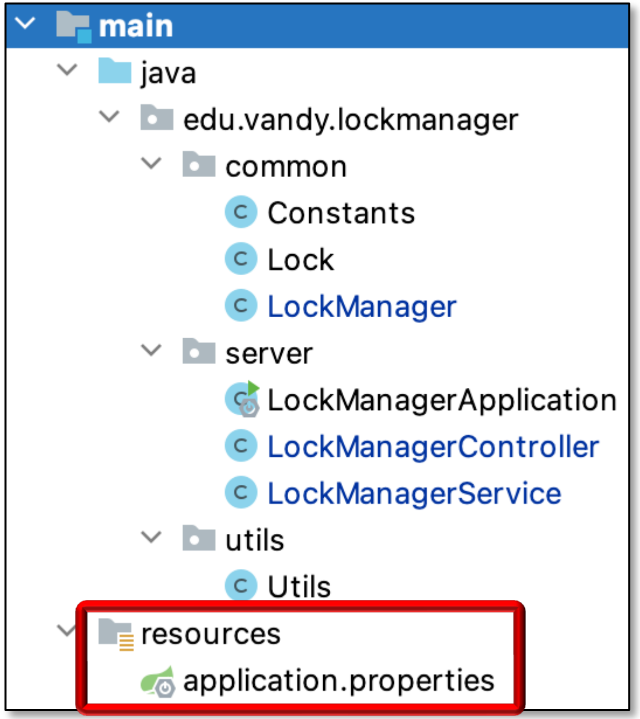
Structure of the LockManager App Project

- The LockManager App project source code is organized into several packages
  - main
    - server
    - common
    - utils
      - General-purpose utilities


main
java
edu.vandy.lockmanager
common
Constants
Lock
LockManager
server
LockManagerApplication
LockManagerController
LockManagerService
utils
Utils
resources
application.properties

Structure of the LockManager App Project

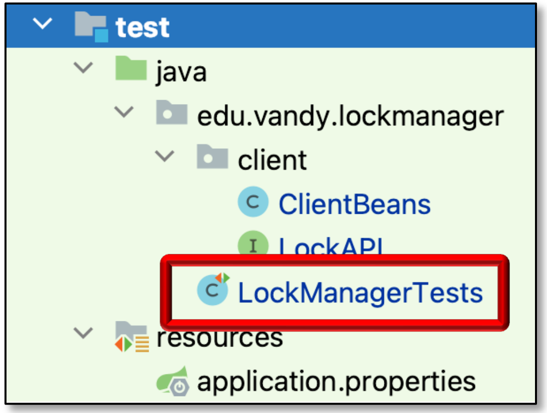
- The LockManager App project source code is organized into several packages
  - main
    - server
    - common
    - utils
    - resources
      - Defines various application properties
        - e.g., name & port number



```
main
  java
    edu.vandy.lockmanager
      common
        Constants
        Lock
        LockManager
      server
        LockManagerApplication
        LockManagerController
        LockManagerService
      utils
        Utils
  resources
    application.properties
```

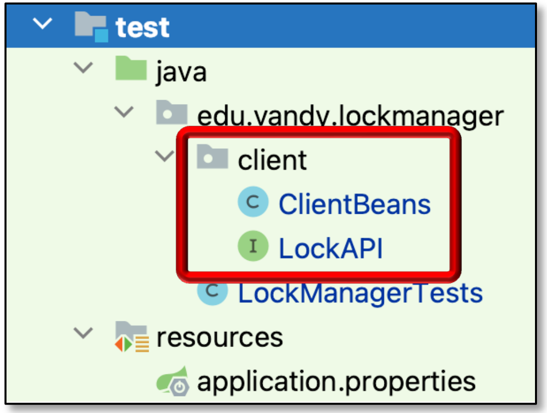
Structure of the LockManager App Project

- The LockManager App project source code is organized into several packages
  - test
    - LockManagerTest
      - This test driver initiates calls to the LockManager microservice



Structure of the LockManager App Project

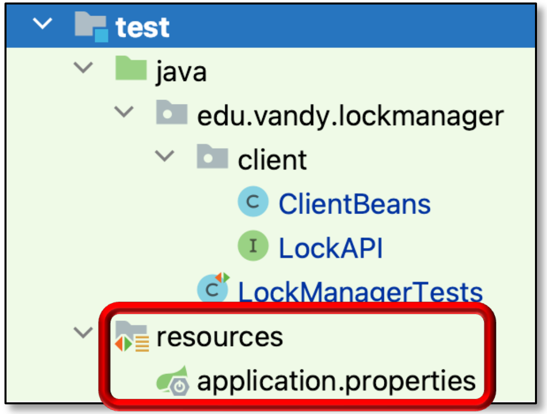
- The LockManager App project source code is organized into several packages
  - test
    - LockManagerTest
    - client
      - Sends/receives HTTP GET/POST requests to the LockManager microservice asynchronously
        - Uses the declarative HTTP interface features in Spring 6



```
test
  java
    edu.vandy.lockmanager
      client
        ClientBeans
        LockAPI
      LockManagerTests
  resources
    application.properties
```

Structure of the LockManager App Project

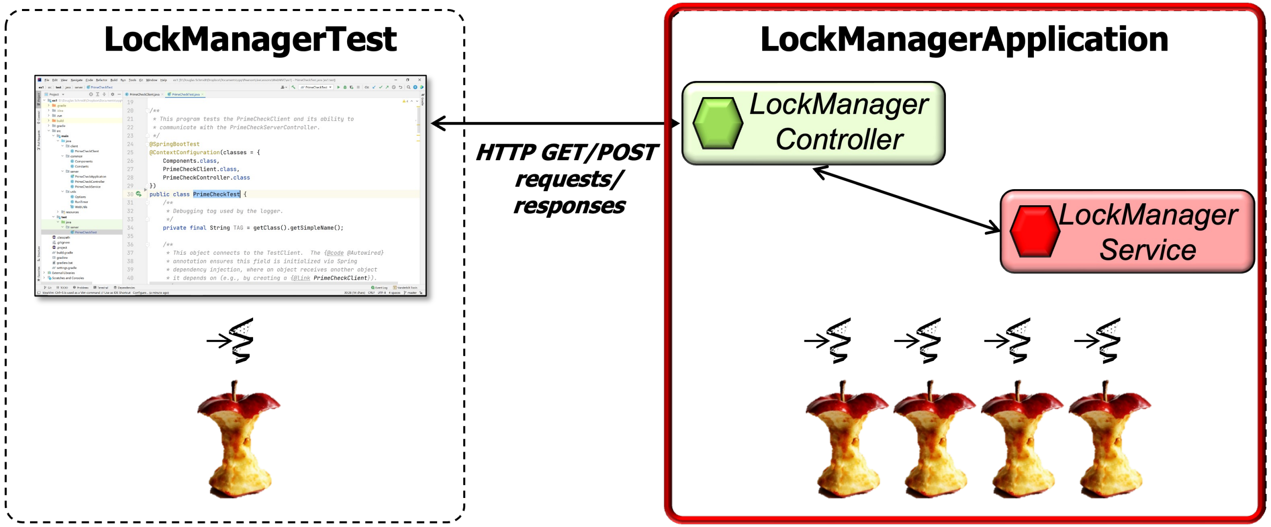
- The LockManager App project source code is organized into several packages
  - test
    - LockManagerTest
    - client
    - resources
      - Enables/disables Spring logging



Pros & Cons of the LockManager App

Pros & Cons of the LockManager App

- Pros
  - Virtual threads are used on the server to improve scalability

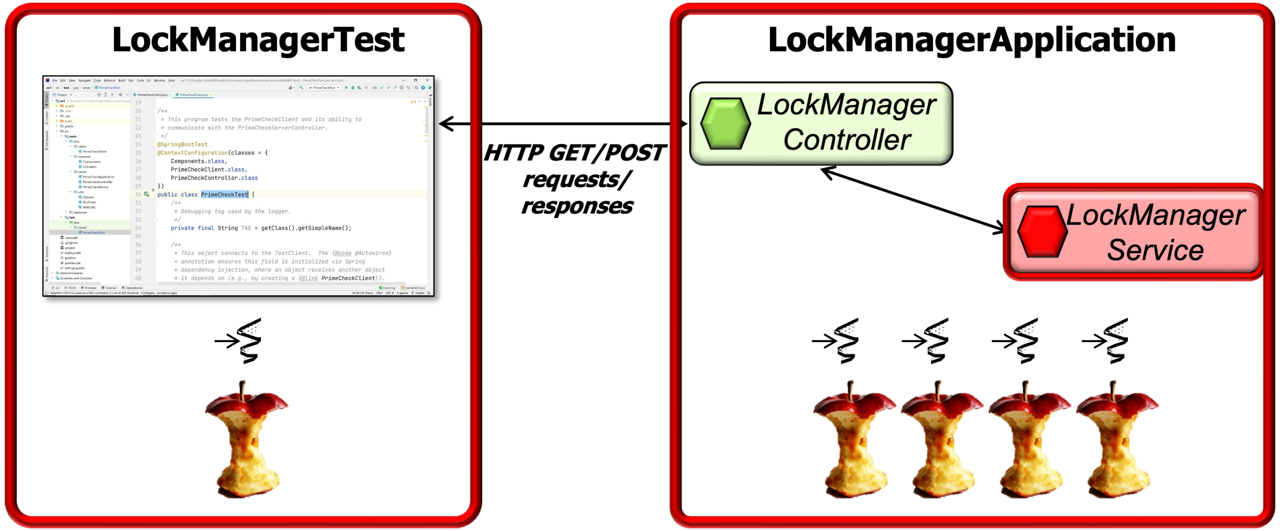


```java
/**
 * This program tests the PrimeCheckClient and its ability to
 * communicate with the PrimeCheckServerController.
 */
@SpringBootTest
@ContextConfiguration(classes = {
    Components.class,
    PrimeCheckClient.class,
    PrimeCheckController.class
})
public class PrimeCheckTest {
    /**
     * Debugging tag used by the logger.
     */
    private final String TAG = getClass().getSimpleName();

    /**
     * This object connects to the TestClient.  The {@code @Autowired}
     * annotation ensures this field is initialized via Spring
     * dependency injection, where an object receives another object
     * it depends on (e.g., by creating a {@link PrimeCheckClient}).
```

In contrast, the Spring WebMVC LockManager app used the servlet thread pool

Pros & Cons of the LockManager App

- Pros
  - Virtual threads are used on the server to improve scalability
  - The client (& server) are fully reactive & async

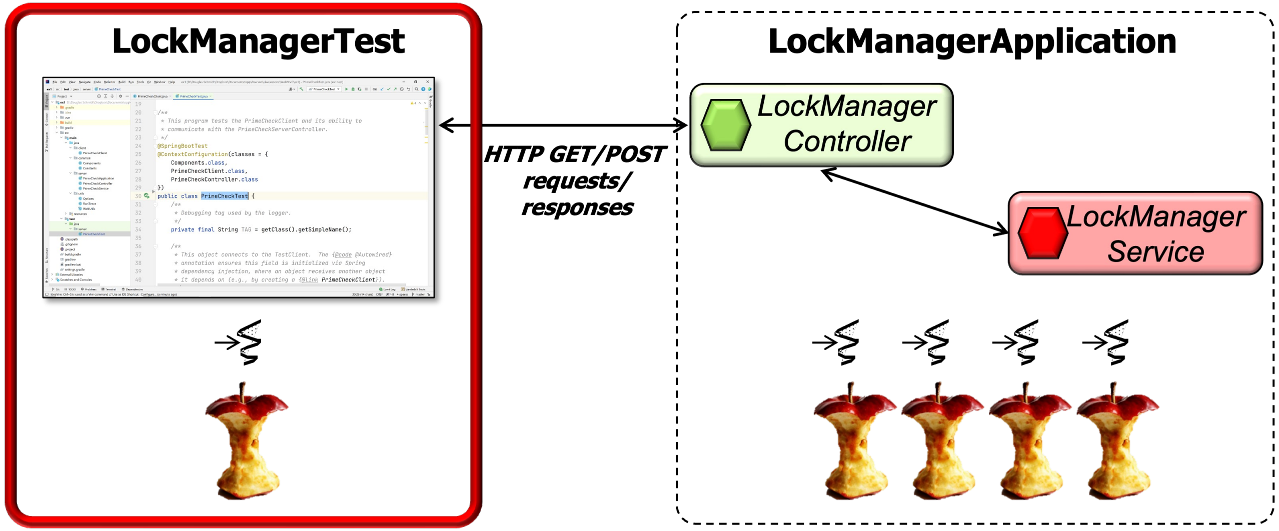


```java
/**
 * This program tests the PrimeCheckClient and its ability to
 * communicate with the PrimeCheckServerController.
 */
@SpringBootTest
@ContextConfiguration(classes = {
    Components.class,
    PrimeCheckClient.class,
    PrimeCheckController.class
})
public class PrimeCheckTest {
    /**
     * Debugging tag used by the logger.
     */
    private final String TAG = getClass().getSimpleName();

    /**
     * This object connects to the TestClient.  The {@code @Autowired}
     * annotation ensures this field is initialized via Spring
     * dependency injection, where an object receives another object
     * it depends on (e.g., by creating a {@link PrimeCheckClient}).
```

In contrast, the Spring WebMVC LockManager app wasn't fully reactive & async

Pros & Cons of the LockManager App

- Pros
  - Virtual threads are used on the server to improve scalability
  - The client (& server) are fully reactive & async
  - The client uses declarative Spring 6 HTTP interface asynchronous proxies

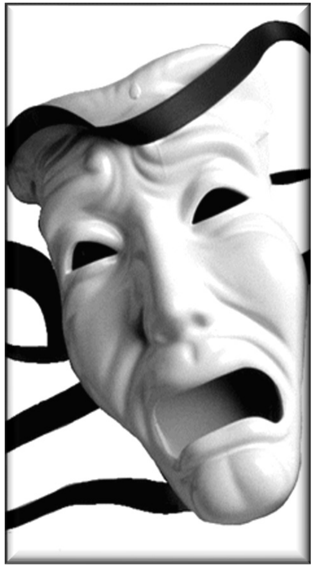


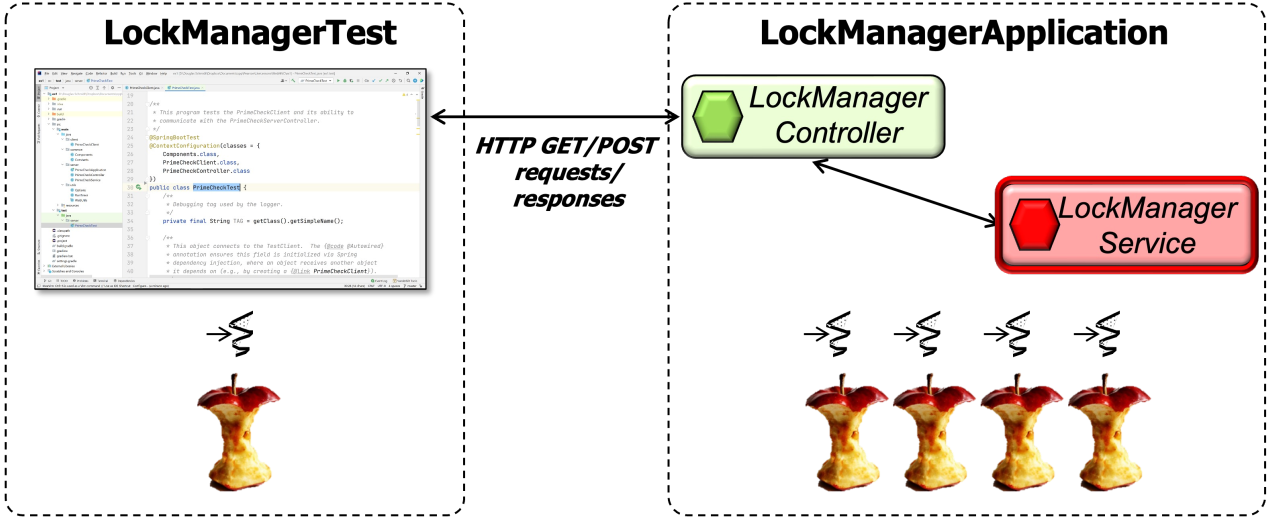
```
/**
 * This program tests the PrimeCheckClient and its ability to
 * communicate with the PrimeCheckServerController.
 */
@SpringBootTest
@ContextConfiguration(classes = {
    Components.class,
    PrimeCheckClient.class,
    PrimeCheckController.class
})
public class PrimeCheckTest {
    /**
     * Debugging tag used by the logger.
     */
    private final String TAG = getClass().getSimpleName();

    /**
     * This object connects to the TestClient.  The {@code @Autowired}
     * annotation ensures this field is initialized via Spring
     * dependency injection, where an object receives another object
     * it depends on (e.g., by creating a {@link PrimeCheckClient}).
```

See www.baeldung.com/spring-6-http-interface

Pros & Cons of the LockManager App

- Cons
  - While the ArrayBlockingQueue implementation is clever, it's not optimal





```
/**
 * This program tests the PrimeCheckClient and its ability to
 * communicate with the PrimeCheckServerController.
 */
@SpringBootTest
@ContextConfiguration(classes = {
    Components.class,
    PrimeCheckClient.class,
    PrimeCheckController.class
})
public class PrimeCheckTest {
    /**
     * Debugging tag used by the logger.
     */
    private final String TAG = getClass().getSimpleName();

    /**
     * This object connects to the TestClient.  The {@code @Autowired}
     * annotation ensures this field is initialized via Spring
     * dependency injection, where an object receives another object
     * it depends on (e.g., by creating a {@link PrimeCheckClient}).
```

There are far more optimal ways of implementing a semaphore!!

End of the LockManager App Case Study: Overview