The LockManager App Case Study: Overview

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

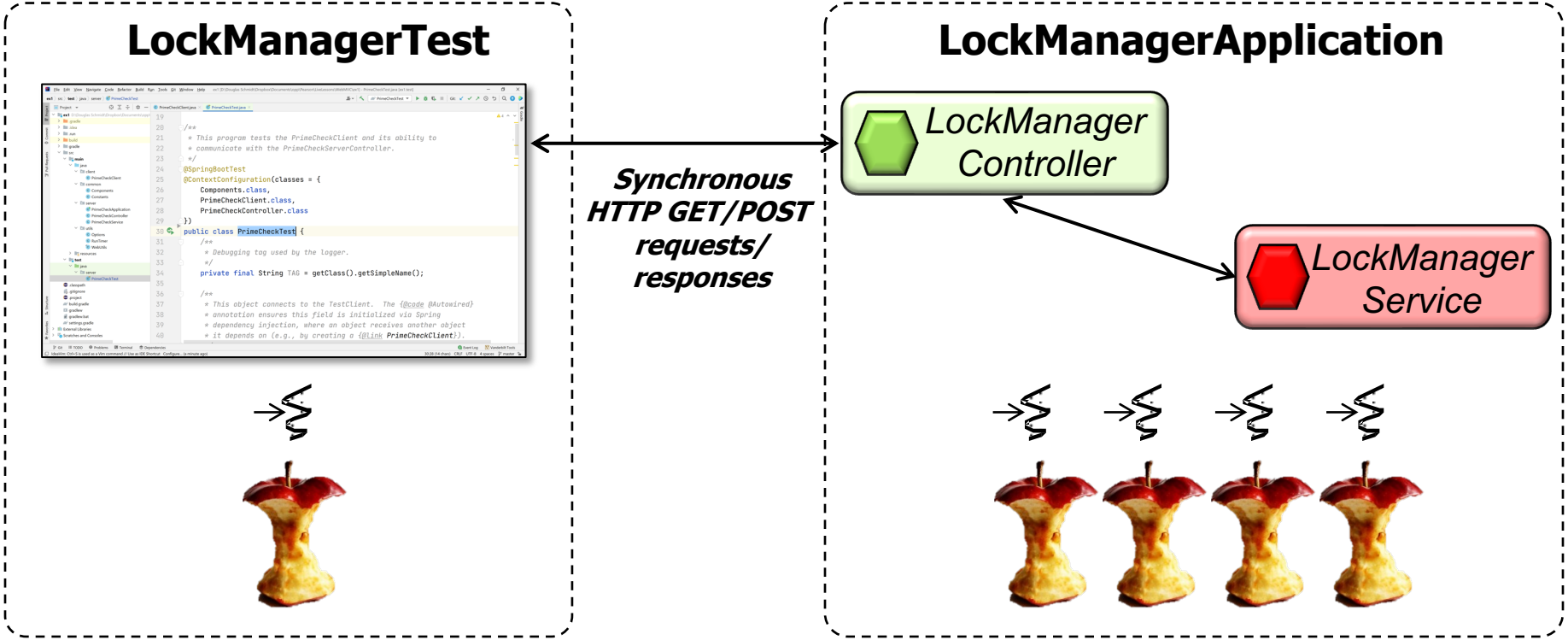
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

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

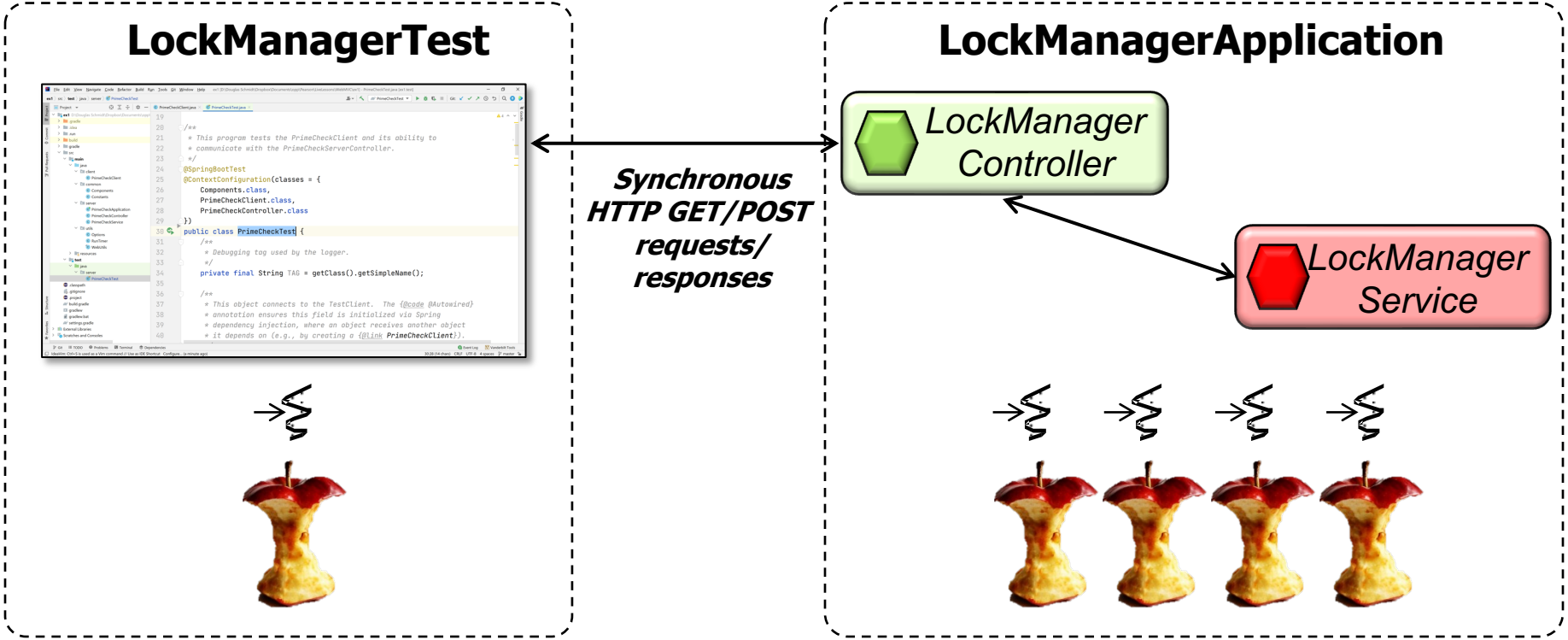


See github.com/douglascraigschmidt/LiveLessons/tree/master/WebMVC/ex5

Overview of the Lock Manager App Case Study

Overview of the LockManager App Case Study

- This case study shows how Spring WebMVC sends/receives HTTP GET/POST requests synchronously to/from a LockManager microservice

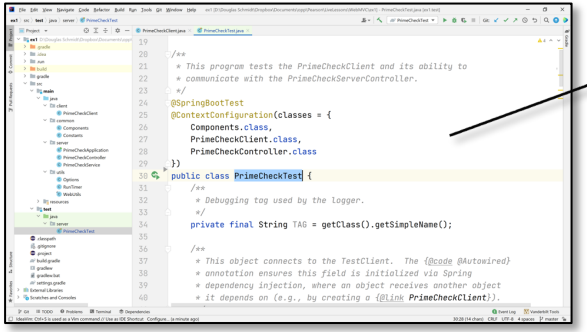


See github.com/douglascraigschmidt/LiveLessons/tree/master/WebMVC/ex5

Overview of the LockManager App Case Study

- This case study shows how Spring WebMVC sends/receives HTTP GET/POST requests synchronously to/from a LockManager microservice

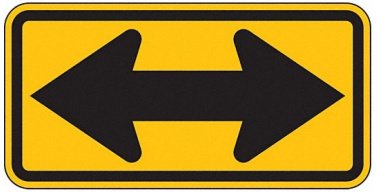
LockManagerTest



```
/**
 * 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 synchronously acquires & releases locks individually or in bulk using declarative LockAPI interface



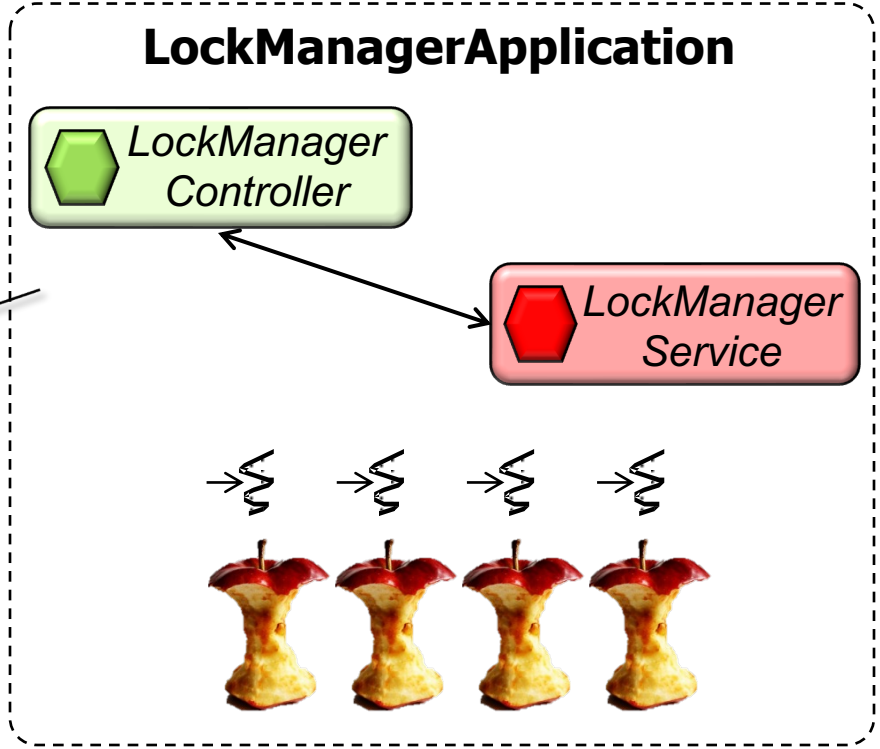
LockAPI

create(Integer)	Boolean
release(List<Lock>)	Boolean
acquire()	Lock
acquire(Integer)	List<Lock>
release(Lock)	Boolean

See WebMVC/ex5/src/test/java/edu/vandy/lockmanager/LockManagerTests.java

Overview of the LockManager App Case Study

- This case study shows how Spring WebMVC sends/receives HTTP GET/POST requests synchronously to/from a LockManager microservice



The server (microservice) acquires & releases locks individually or in bulk

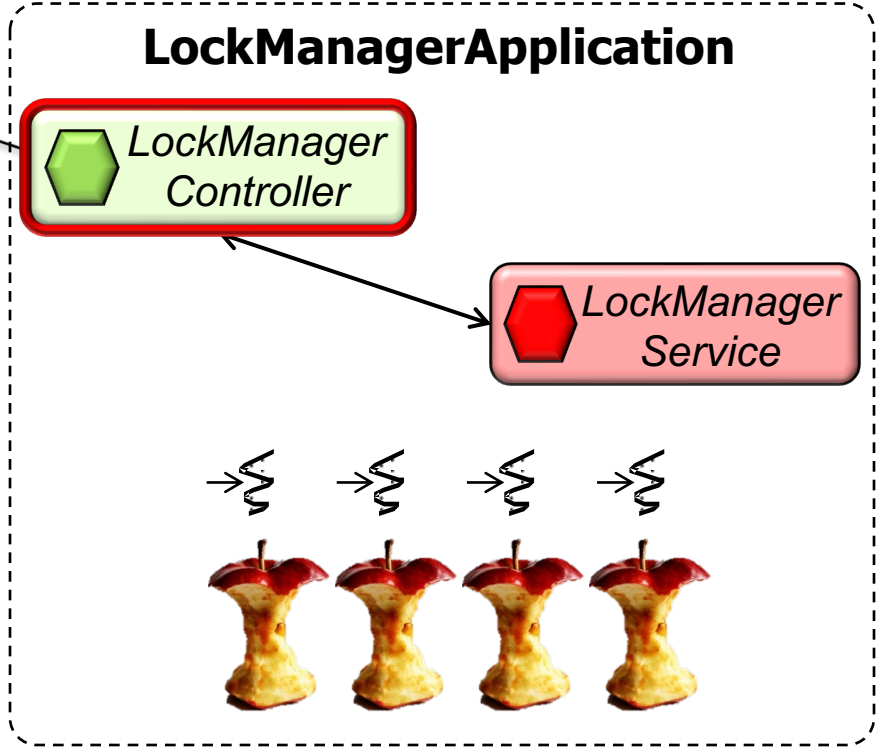
See WebMVC/ex5/src/main/java/edu/vandy/lockmanager/server

Overview of the LockManager App Case Study

- This case study shows how Spring WebMVC sends/receives HTTP GET/POST requests synchronously to/from a LockManager microservice

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

LockManagerController	
mService	LockManagerService
release(List<Lock>)	Boolean
create(Integer)	Boolean
acquire(Integer)	DeferredResult<List<Lock>>
release(Lock)	Boolean
acquire()	DeferredResult<Lock>
isAlive()	String



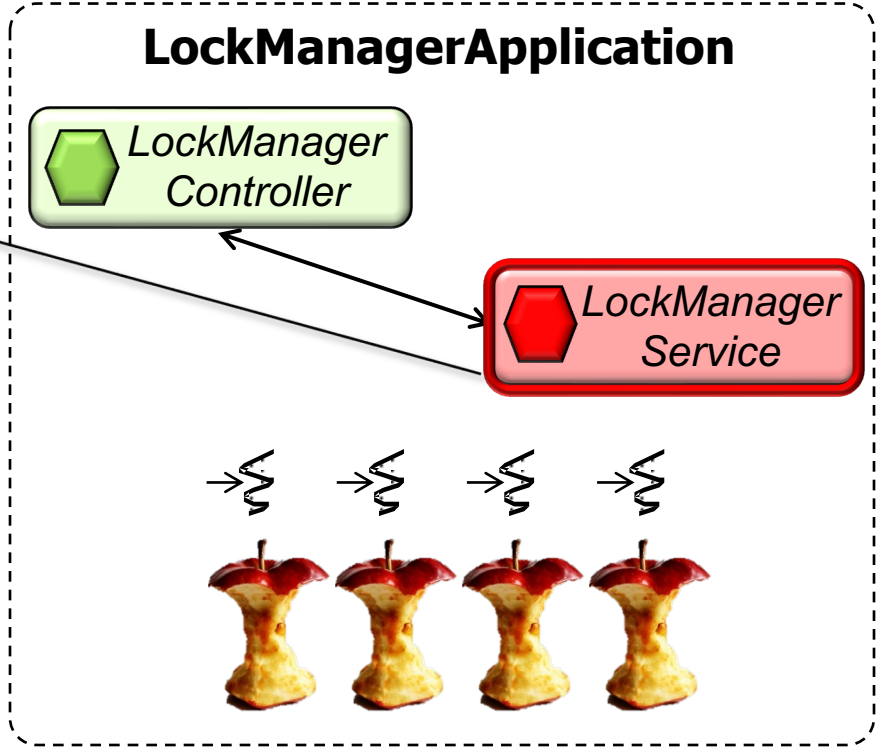
See WebMVC/ex5/src/main/java/edu/vandy/lockmanager/server/LockManagerController.java

Overview of the LockManager App Case Study

- This case study shows how Spring WebMVC sends/receives HTTP GET/POST requests synchronously to/from a LockManager microservice

The LockManagerService uses an Array BlockingQueue object & virtual threads to implement a distributed semaphore

LockManagerService	
mExecutor	ExecutorService
mAvailableLocks	ArrayBlockingQueue<Lock>
acquire()	DeferredResult<Lock>
create(Integer)	Boolean
release(List<Lock>)	Boolean
acquire(int)	DeferredResult<List<Lock>>
makeLocks(int)	List<Lock>
tryAcquireLock(List<Lock>)	Integer
release(Lock)	Boolean

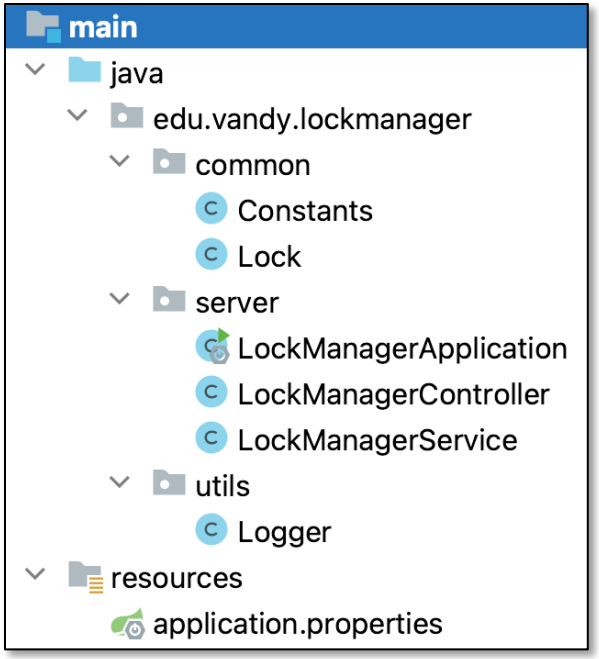


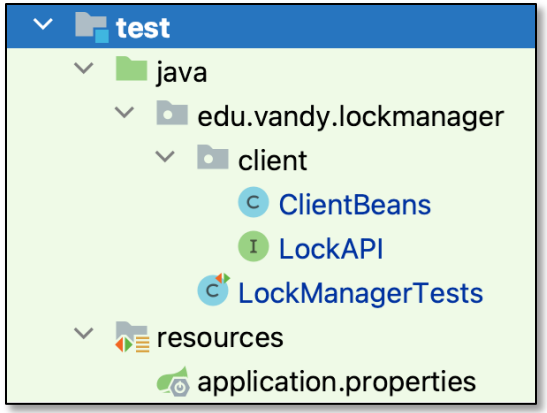
See WebMVC/ex5/src/main/java/edu/vandy/lockmanager/server/LockManagerService.java

Structure of the Lock Manager App Project

Structure of the LockManager App Project

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

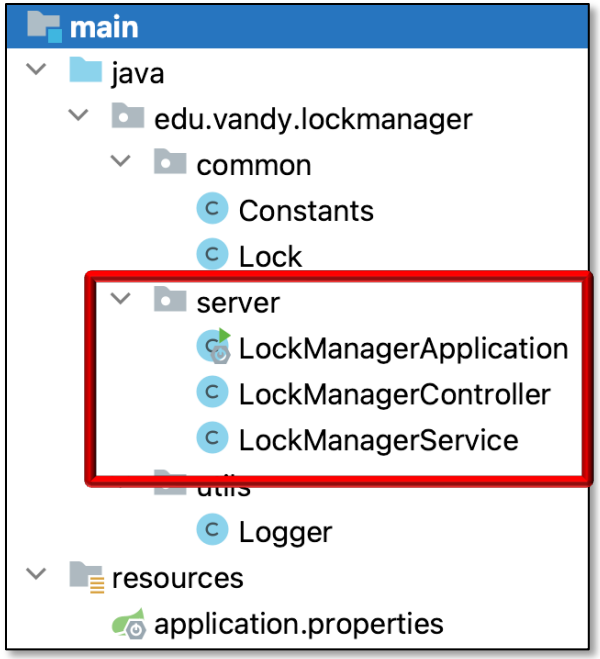




See github.com/douglascraigschmidt/LiveLessons/tree/master/WebMVC/ex5

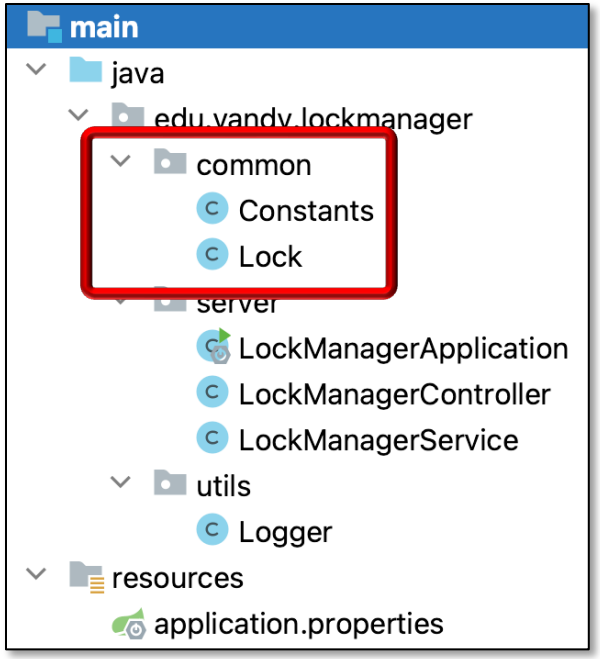
Structure of the LockManager App Project

- 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 conventional Java types



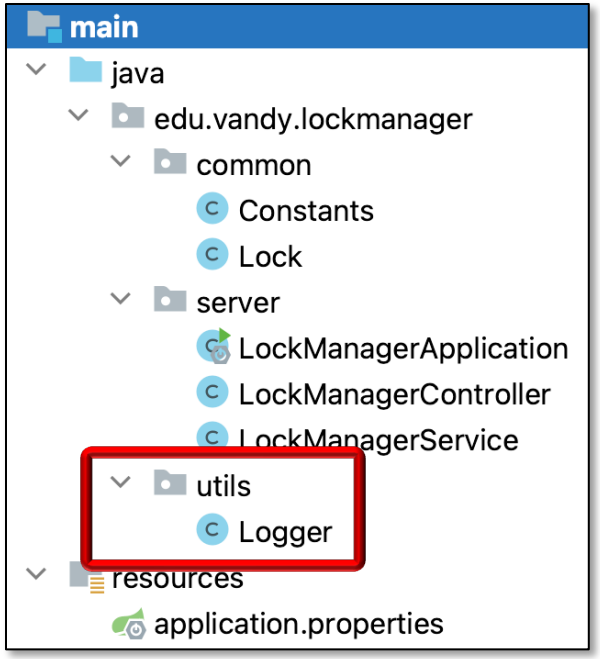
Structure of the LockManager App Project

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



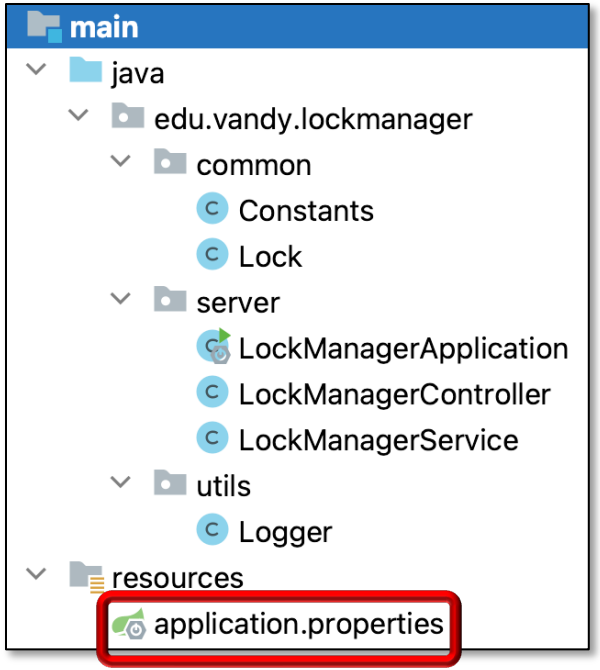
Structure of the LockManager App Project

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



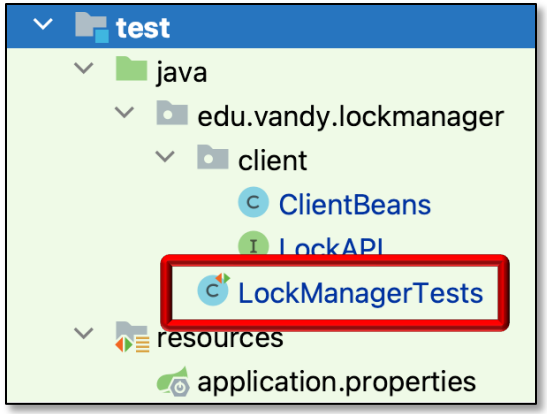
Structure of the LockManager App Project

- 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



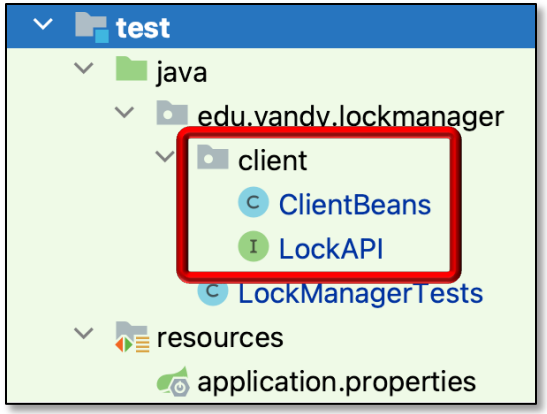
Structure of the LockManager App Project

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



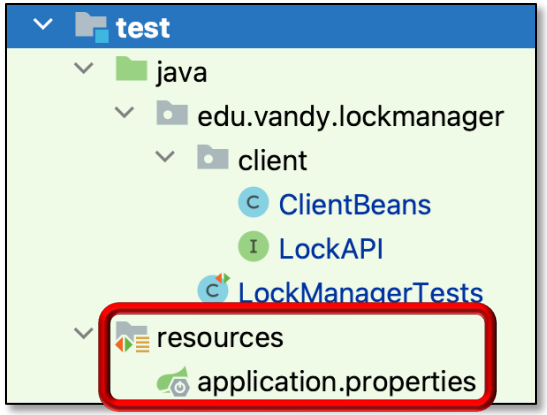
Structure of the LockManager App Project

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



Structure of the LockManager App Project

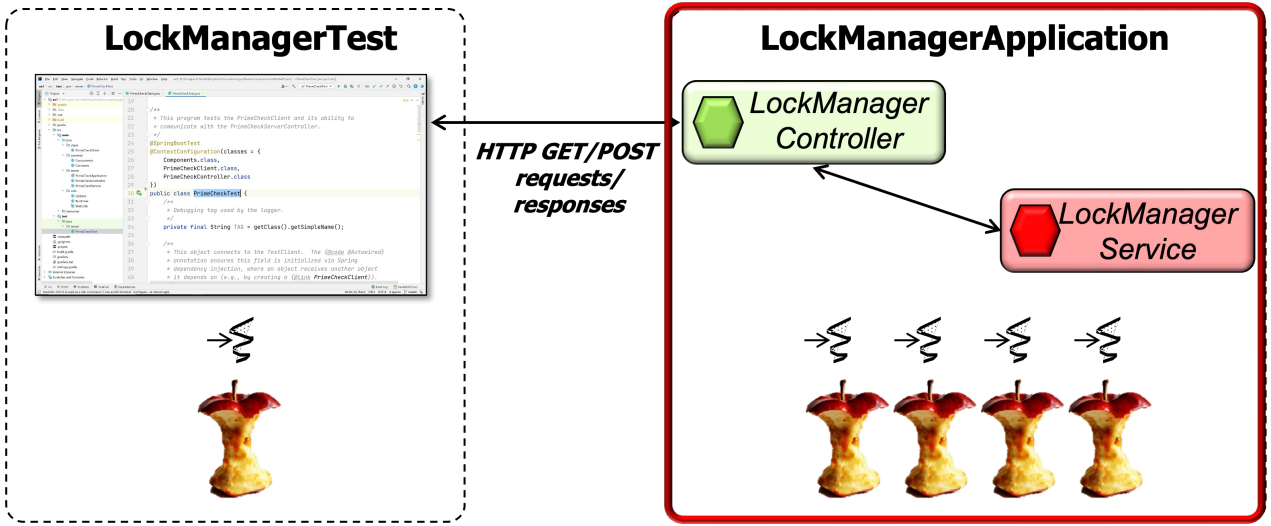
- The LockManager App project source code is organized into several packages
  - test
    - LockManagerTest
    - client
    - resources
      - Enables/disables Spring logging



Pros & Cons of the LockManager App

Pros & Cons of the LockManager App

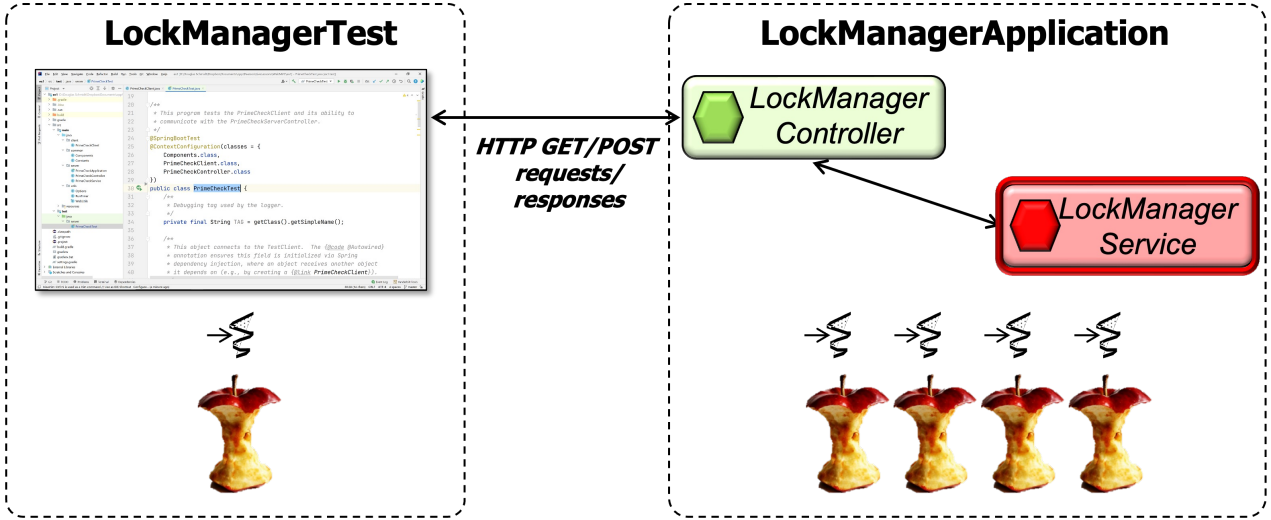
- Pros
  - Spring's DeferredRequest mechanism avoids blocking the servlet thread



Can improve system scalability in traditional (i.e., pre-JDK 19) Java execution environments

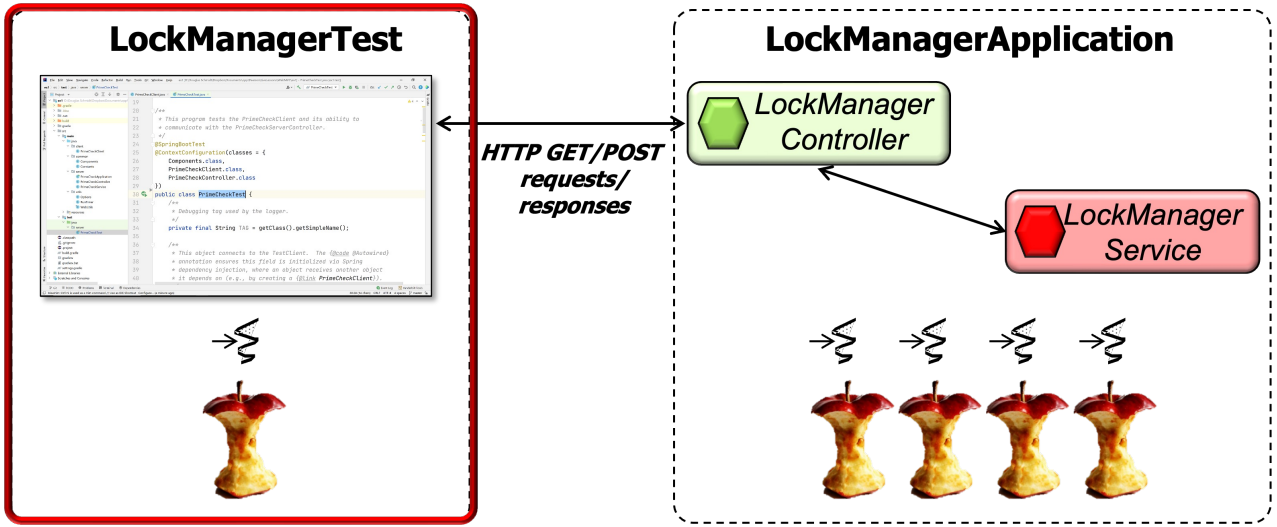
Pros & Cons of the LockManager App

- Pros
  - Spring's DeferredRequest mechanism avoids blocking the servlet thread
  - Clever use of ArrayBlockingQueue avoids having to know synchronizers



Pros & Cons of the LockManager App

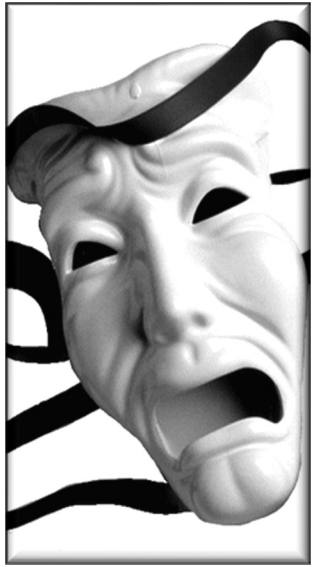
- Pros
  - Spring's DeferredRequest mechanism avoids blocking the servlet thread
  - Clever use of ArrayBlockingQueue avoids having to know synchronizers
  - The client uses declarative Spring 6 HTTP interface synchronous proxies

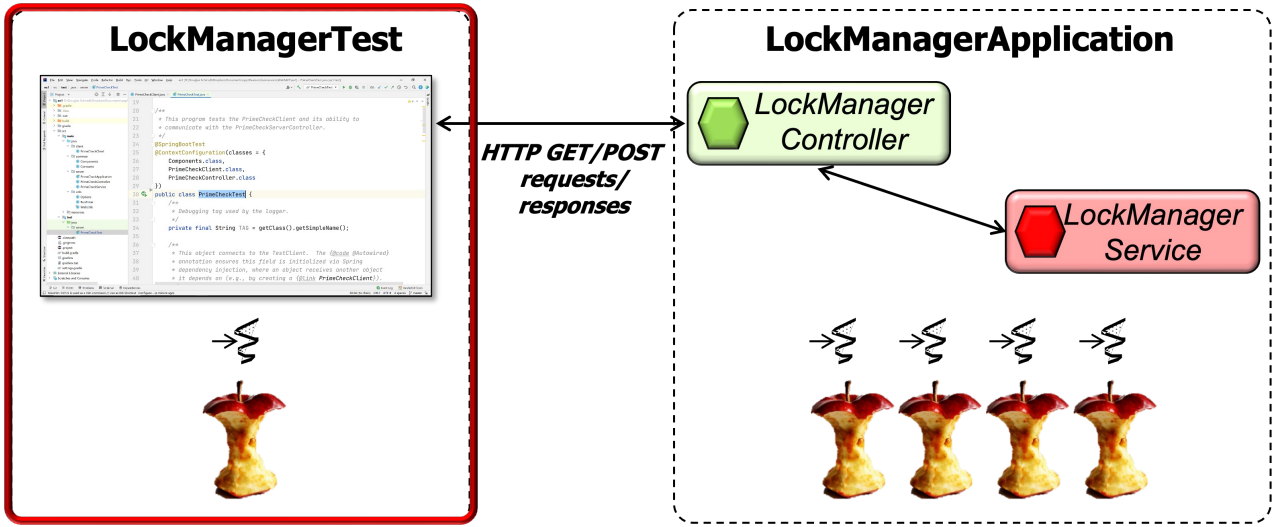


See www.baeldung.com/spring-6-http-interface

Pros & Cons of the LockManager App

- Cons
  - The client isn't actually asynchronous, only the server

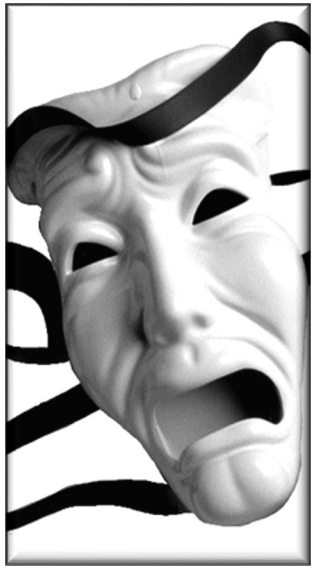


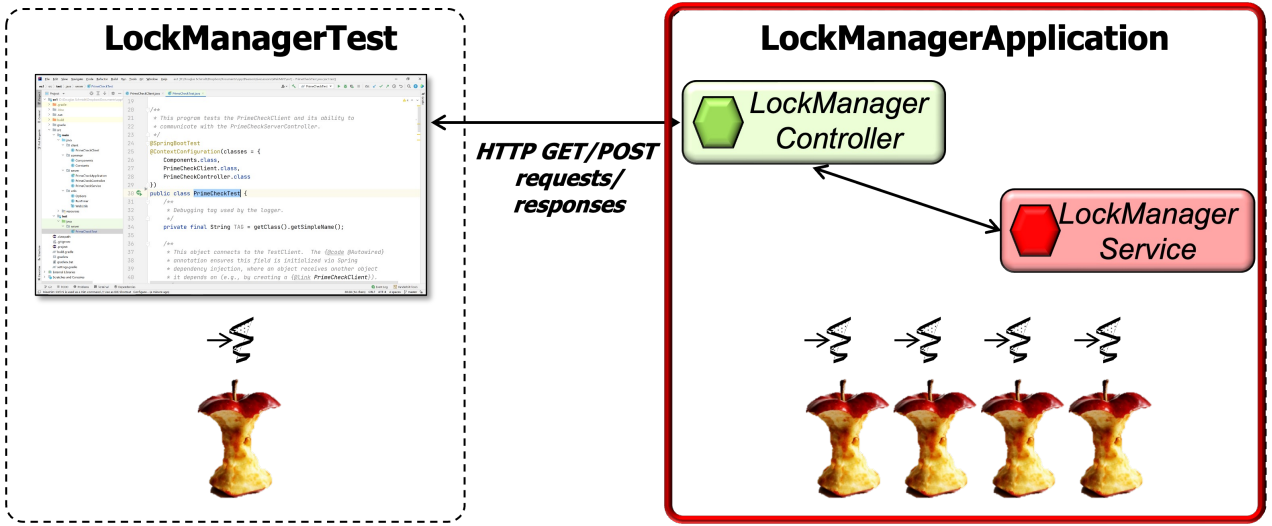


As a result client threads may block, which can cause timeout problems

Pros & Cons of the LockManager App

- Cons
  - The client isn't actually asynchronous, only the server
  - The server uses the Spring WebMVC thread pool model

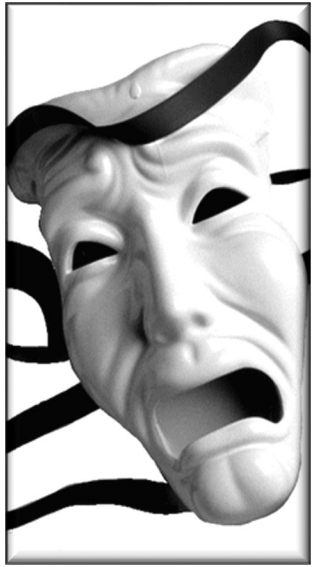


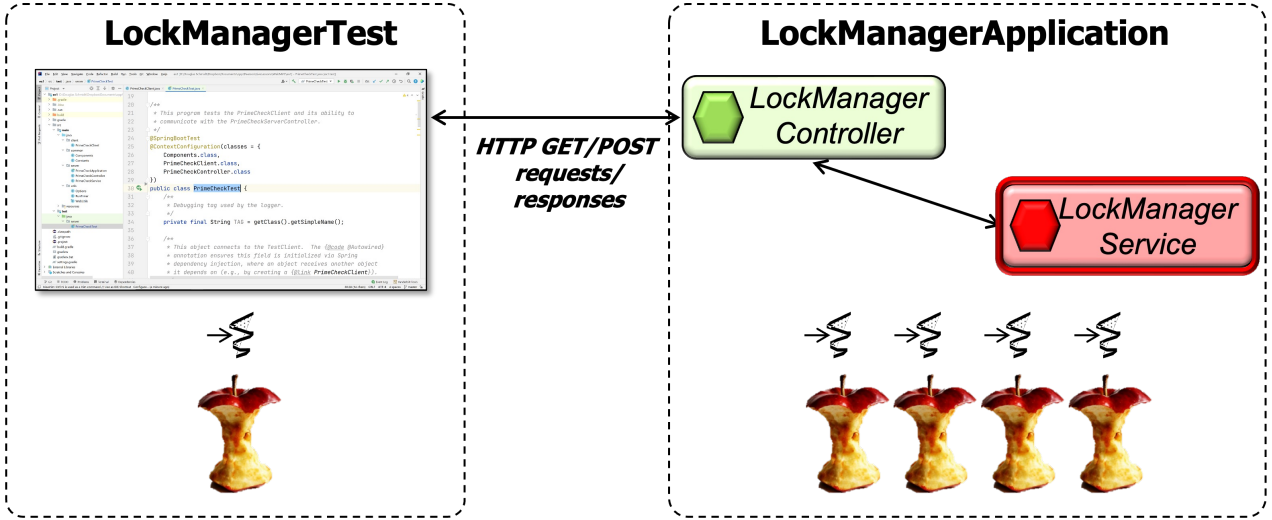


This doesn't take full advantages of Java virtual threads

Pros & Cons of the LockManager App

- Cons
  - The client isn't actually asynchronous, only the server
  - The server uses the Spring WebMVC thread pool model
  - The ArrayBlockingQueue implementation is not optimal

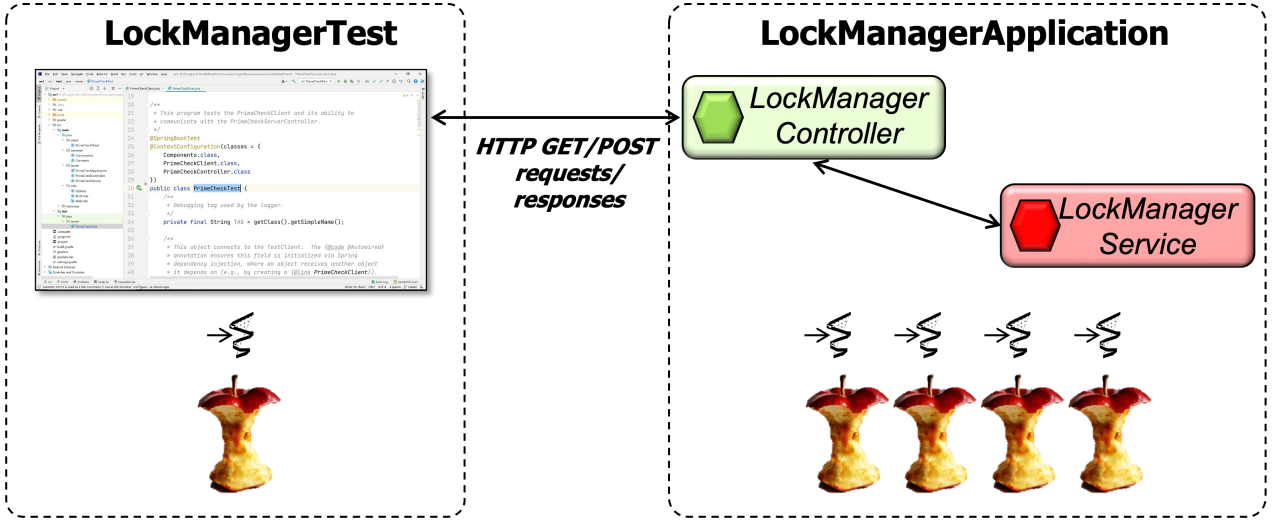




There are far more optimal ways of implementing a semaphore!!

Pros & Cons of the LockManager App

- Cons
  - The client isn't actually asynchronous, only the server
  - The server uses the Spring WebMVC thread pool mode
  - The ArrayBlockingQueue implementation is not optimal



We address some of these limitations by using WebFlux & Java virtual threads

End of the LockManager App Case Study: Overview