The MathServices App Case Study: Implementing the Primality Microservice

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

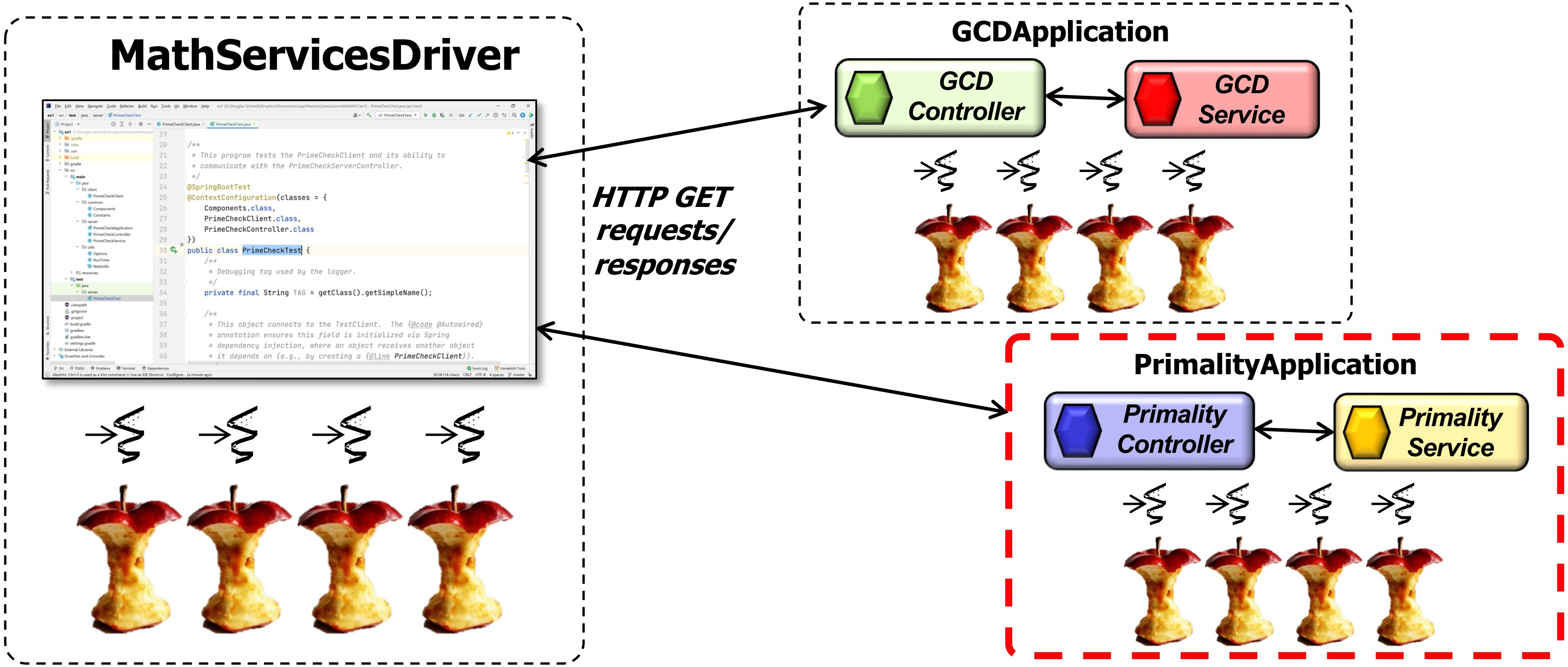
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand the concurrent implementation of the PrimalityController & PrimalityService classes that run in the PrimalityApplication microservice

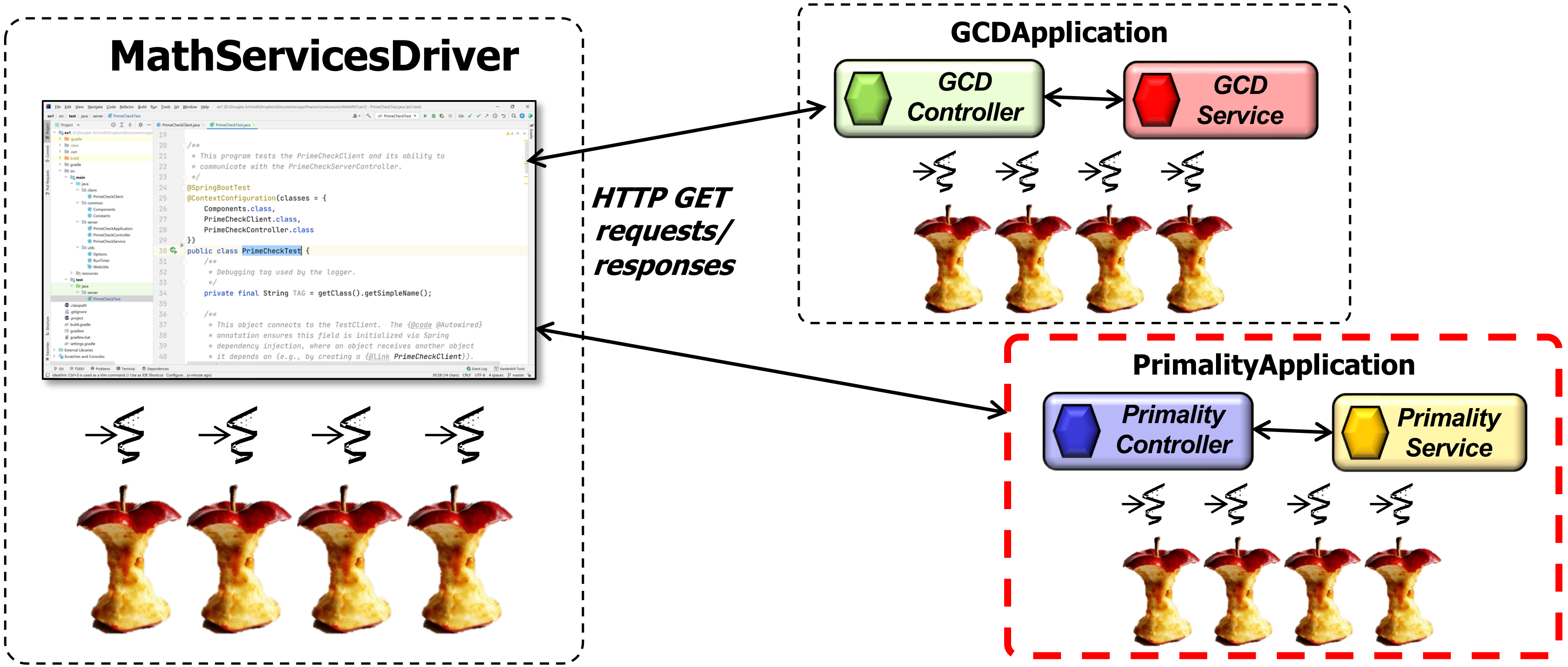


```
/**
 * This program tests the PrimeCheckClient and its ability to
 * communicate with the PrimeCheckServerController.
 */
@SpringBootTest
@ContextConfiguration(classes = {
    Components.class,
    PrimeCheckClient.class,
    PrimeCheckController.class
})
public class PrimeCheckTest {
    /**
     * 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}).
```

Learning Objectives in this Part of the Lesson

- Understand the implementation of the PrimalityController & PrimalityService classes that run in the PrimalityApplication microservice



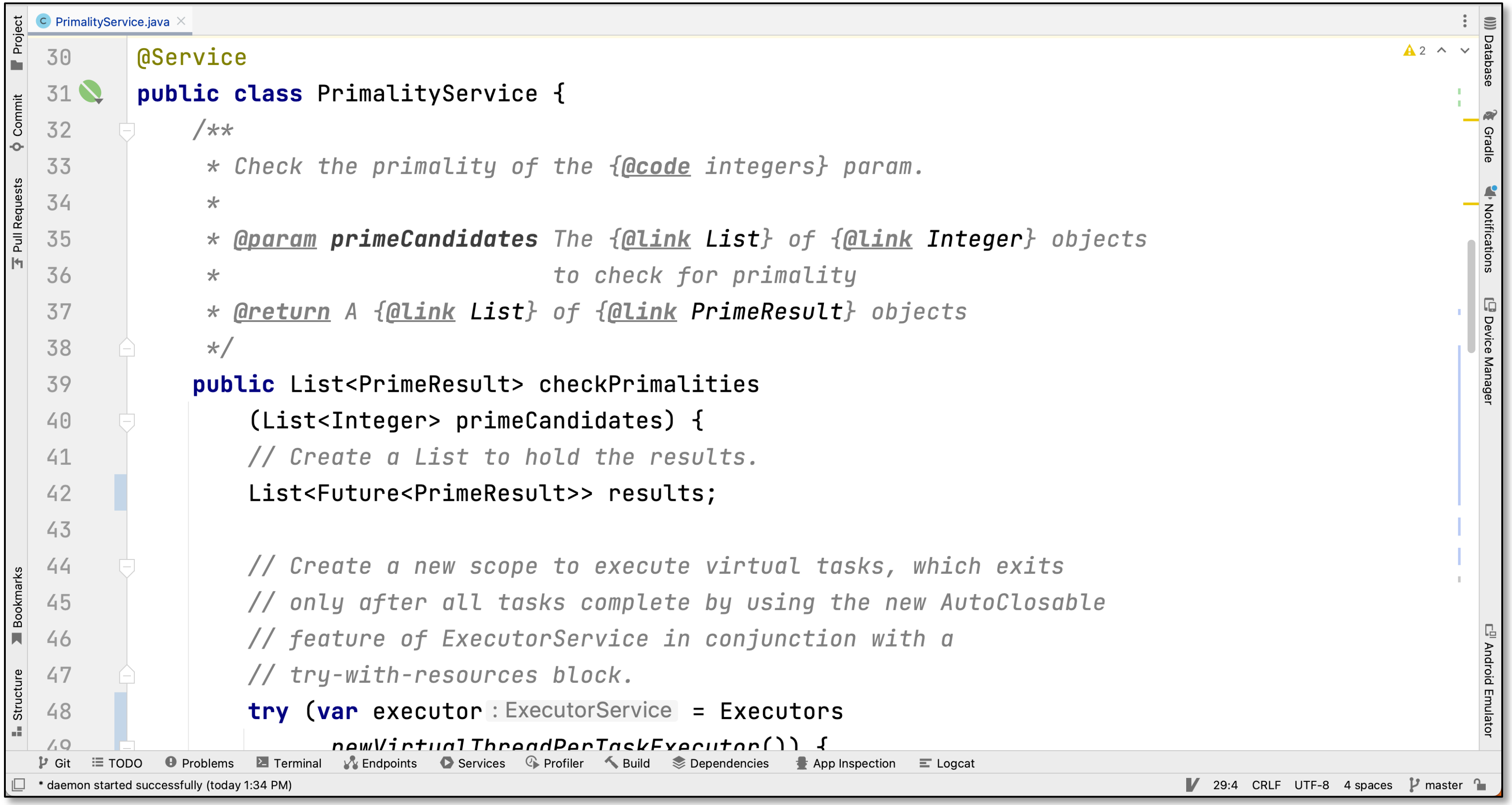
```
/**
 * This program tests the PrimeCheckClient and its ability to
 * communicate with the PrimeCheckServerController.
 */
@SpringBootTest
@ContextConfiguration(classes = {
    Components.class,
    PrimeCheckClient.class,
    PrimeCheckController.class
})
public class PrimeCheckTest {
    /**
     * 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 focus is on the Java Executors VirtualThreadPerTaskExecutor model

Implementing the Primality Application Microservice

Implementing the PrimalityApplication Microservice



PrimalityService.java

```
@Service
public class PrimalityService {
    /**
     * Check the primality of the {@code integers} param.
     *
     * @param primeCandidates The {@link List} of {@link Integer} objects
     *                        to check for primality
     * @return A {@link List} of {@link PrimeResult} objects
     */
    public List<PrimeResult> checkPrimalities
        (List<Integer> primeCandidates) {
        // Create a List to hold the results.
        List<Future<PrimeResult>> results;

        // Create a new scope to execute virtual tasks, which exits
        // only after all tasks complete by using the new AutoClosable
        // feature of ExecutorService in conjunction with a
        // try-with-resources block.
        try (var executor : ExecutorService = Executors
```

Git TODO Problems Terminal Endpoints Services Profiler Build Dependencies App Inspection Logcat

* daemon started successfully (today 1:34 PM)

29:4 CRLF UTF-8 4 spaces master

See github.com/douglascraigschmidt/LiveLessons/tree/master/WebMVC/ex3

End of the MathServices App Case Study: Implementing the Primality Microservice