

The PrimeCheckApp Case Study

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

**Institute for Software
Integrated Systems**

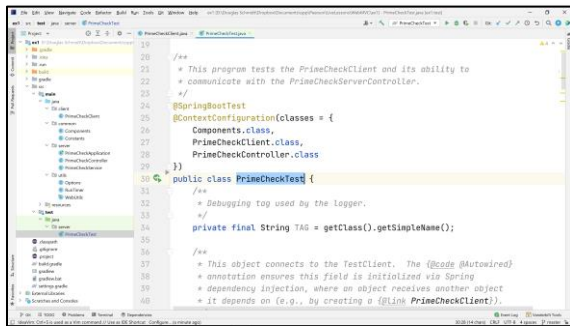
**Vanderbilt University
Nashville, Tennessee, USA**



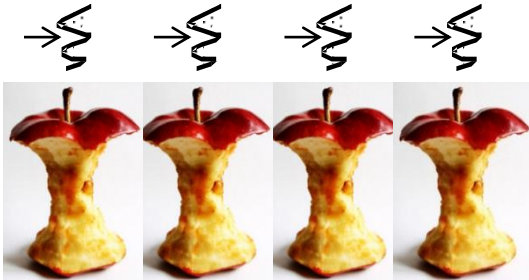
Learning Objectives in this Lesson

- Understand how functional programming & Java parallel streams are applied in a case study that uses Spring WebMVC to check primality of large integers

PrimeCheckTest

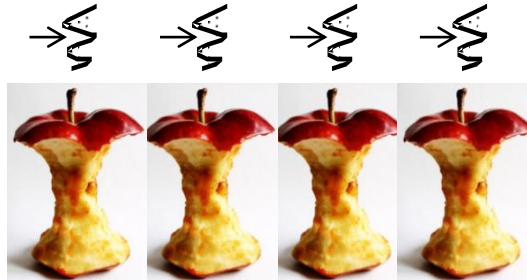


```
20 /**  
21  * This program tests the PrimeCheckClient and its ability to  
22  * communicate with the PrimeCheckServerController.  
23  */  
24  
25 @SpringBootTest  
26 @ContextConfiguration(classes = {  
27     Components.class,  
28     PrimeCheckClient.class,  
29     PrimeCheckServerController.class  
30 })  
31 public class PrimeCheckTest {  
32     /**  
33      * Debugging tag used by the logger.  
34      */  
35     private final String TAG = getClass().getSimpleName();  
36  
37     /**  
38      * This object connects to the testClient. The @Autowired  
39      * annotation ensures this field is initialized via Spring  
40      * dependency injection, where an object receives another object  
41      * it depends on (e.g., by creating a @Link PrimeCheckClient).  
42      */  
43     PrimeCheckClient testClient;  
44  
45     /**  
46      * This object connects to the testServer. The @Autowired  
47      * annotation ensures this field is initialized via Spring  
48      * dependency injection, where an object receives another object  
49      * it depends on (e.g., by creating a @Link PrimeCheckServerController).  
50      */  
51     PrimeCheckServerController testServer;  
52  
53     /**  
54      * This method tests the PrimeCheckClient and its ability to  
55      * communicate with the PrimeCheckServerController.  
56      */  
57     @Test  
58     public void testPrimeCheckClient() {  
59         // ...  
60     }  
61  
62     /**  
63      * This method tests the PrimeCheckServerController and its ability  
64      * to communicate with the PrimeCheckClient.  
65      */  
66     @Test  
67     public void testPrimeCheckServerController() {  
68         // ...  
69     }  
70 }  
71
```



*HTTP GET
requests/
responses*

PrimeCheckApp



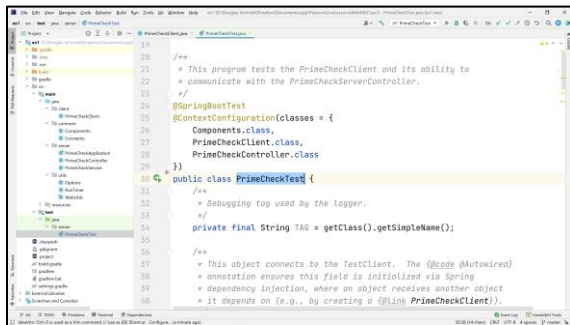
See github.com/douglasraigschmidt/LiveLessons/tree/master/WebMVC/ex1

Overview of the Prime CheckApp Case Study

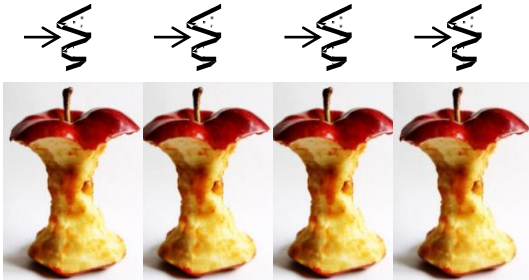
Overview of the PrimeCheckApp Case Study

- This case study shows how Spring WebMVC can be used to send & receive HTTP GET requests

PrimeCheckTest

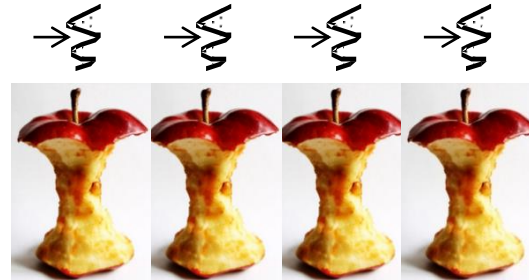


```
20 /**  
21  * This program tests the PrimeCheckClient and its ability to  
22  * communicate with the PrimeCheckServerController.  
23  */  
24 @SpringBootTest  
25 @ContextConfiguration(classes = {  
26     Components.class,  
27     PrimeCheckClient.class,  
28     PrimeCheckController.class  
29 })  
30 public class PrimeCheckTest {  
31     /**  
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      */  
42     @Autowired  
43     PrimeCheckClient client;  
44  
45     /**  
46      * This object connects to the TestServer. The @Autowired  
47      * annotation ensures this field is initialized via Spring  
48      * dependency injection, where an object receives another object  
49      * it depends on (e.g., by creating a @Link PrimeCheckController).  
50      */  
51     @Autowired  
52     PrimeCheckController controller;  
53  
54     /**  
55      * This method tests the PrimeCheckClient and its ability to  
56      * communicate with the PrimeCheckServerController.  
57      */  
58     @Test  
59     public void testPrimeCheckClient() {  
60         // ...  
61     }  
62  
63     /**  
64      * This method tests the PrimeCheckController and its ability to  
65      * communicate with the PrimeCheckServerController.  
66      */  
67     @Test  
68     public void testPrimeCheckController() {  
69         // ...  
70     }  
71 }
```



*HTTP GET
requests/
responses*

PrimeCheckApp



Overview of the PrimeCheckApp Case Study

- This case study shows how Spring WebMVC can be used to send & receive HTTP GET requests
- These requests are mapped to endpoint handler methods that determine the primality of large random integers

```
@RestController
public class PrimeCheckController {
    @GetMapping(CHECK_IF_PRIME)
    public Integer checkIfPrime
        (Integer primeCandidate) {...}

    @GetMapping(CHECK_IF_PRIME_LIST)
    public List<Integer>
        checkIfPrimeList
            (@RequestParam List<Integer>
             primeCandidates,
             Boolean parallel) {...}
}
```

Overview of the PrimeCheckApp Case Study

- This case study uses synchronous two-way calls & the Java parallel streams framework

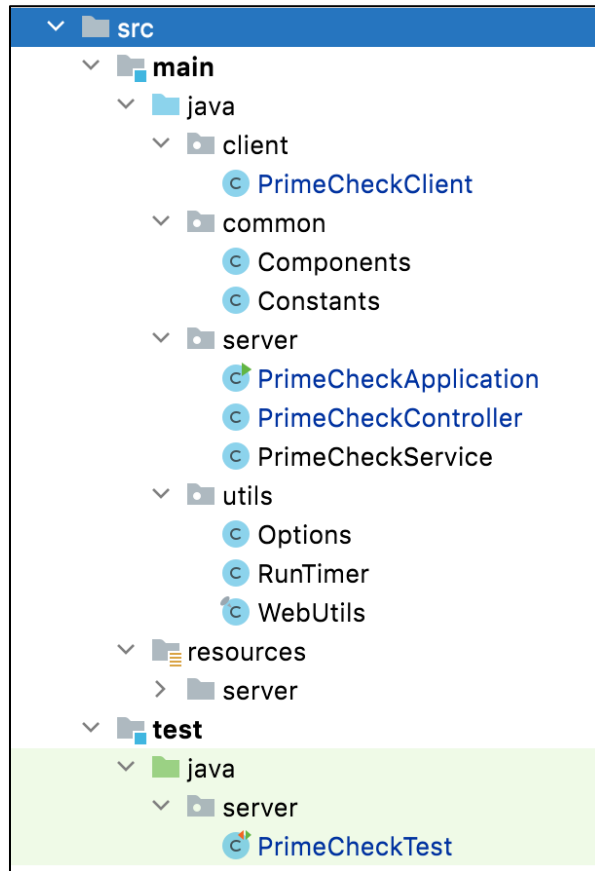
```
List<Integer> checkIfPrimeList
(List<Integer> primeCandidates,
 Boolean parallel) {
    var stream = primeCandidates
        .stream();

    if (parallel)
        stream.parallel();

    return stream
        .map(this::isPrime)
        .collect(toList());
}
```

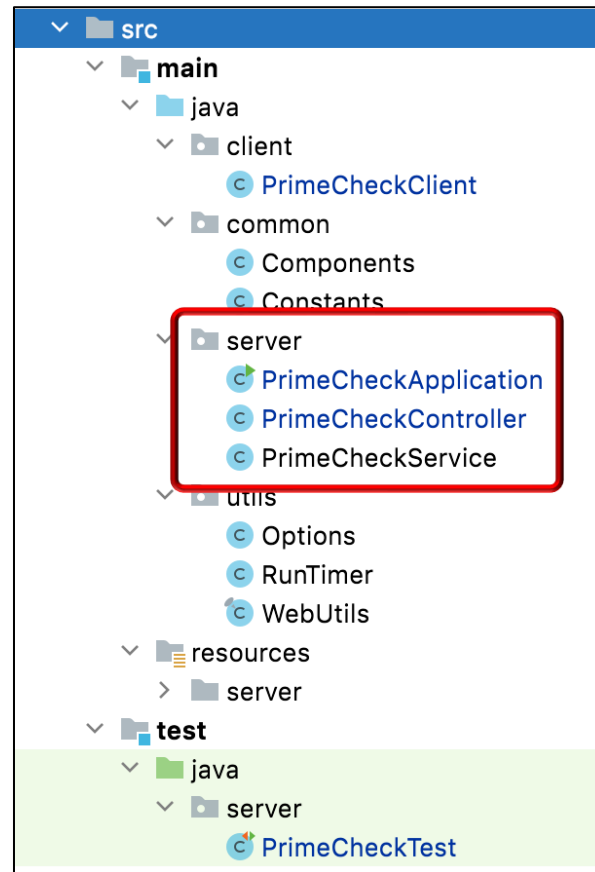
Overview of the PrimeCheckApp Case Study

- The source code is organized into several packages



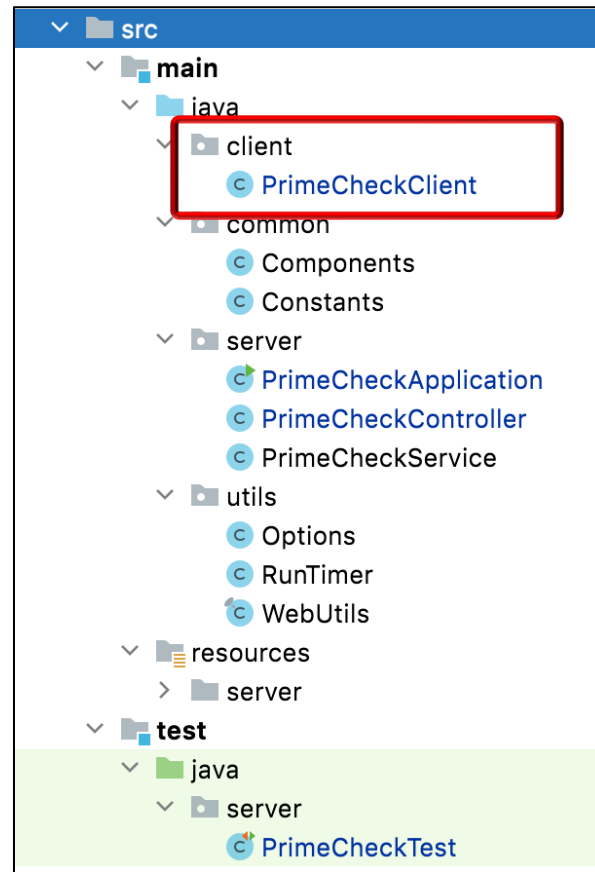
Overview of the PrimeCheckApp Case Study

- The source code is organized into several packages
 - Server



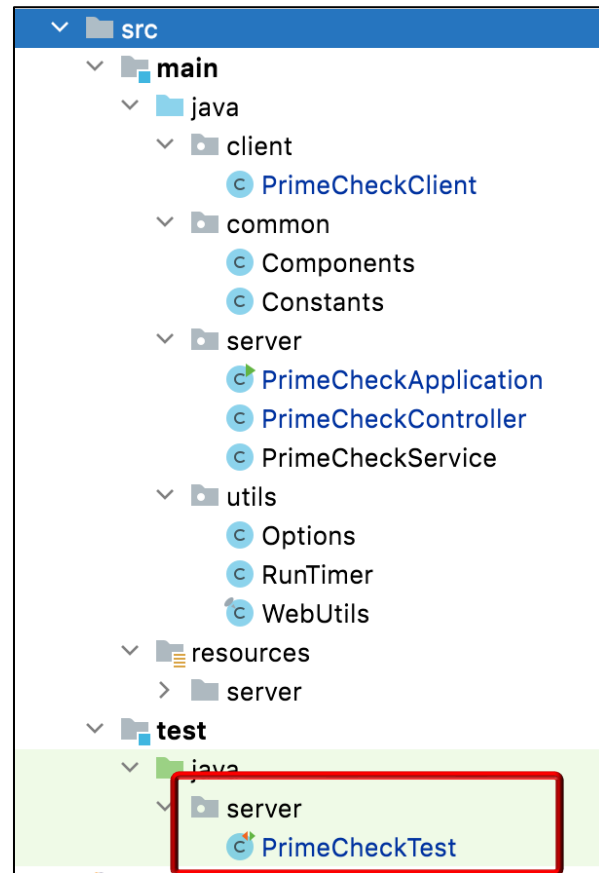
Overview of the PrimeCheckApp Case Study

- The source code is organized into several packages
 - Server
 - Client



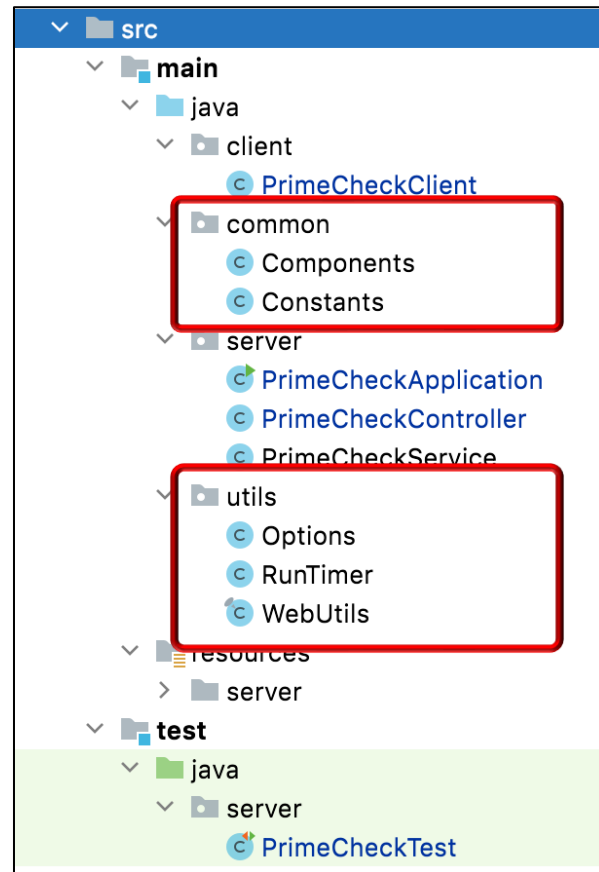
Overview of the PrimeCheckApp Case Study

- The source code is organized into several packages
 - Server
 - Client
 - Test driver

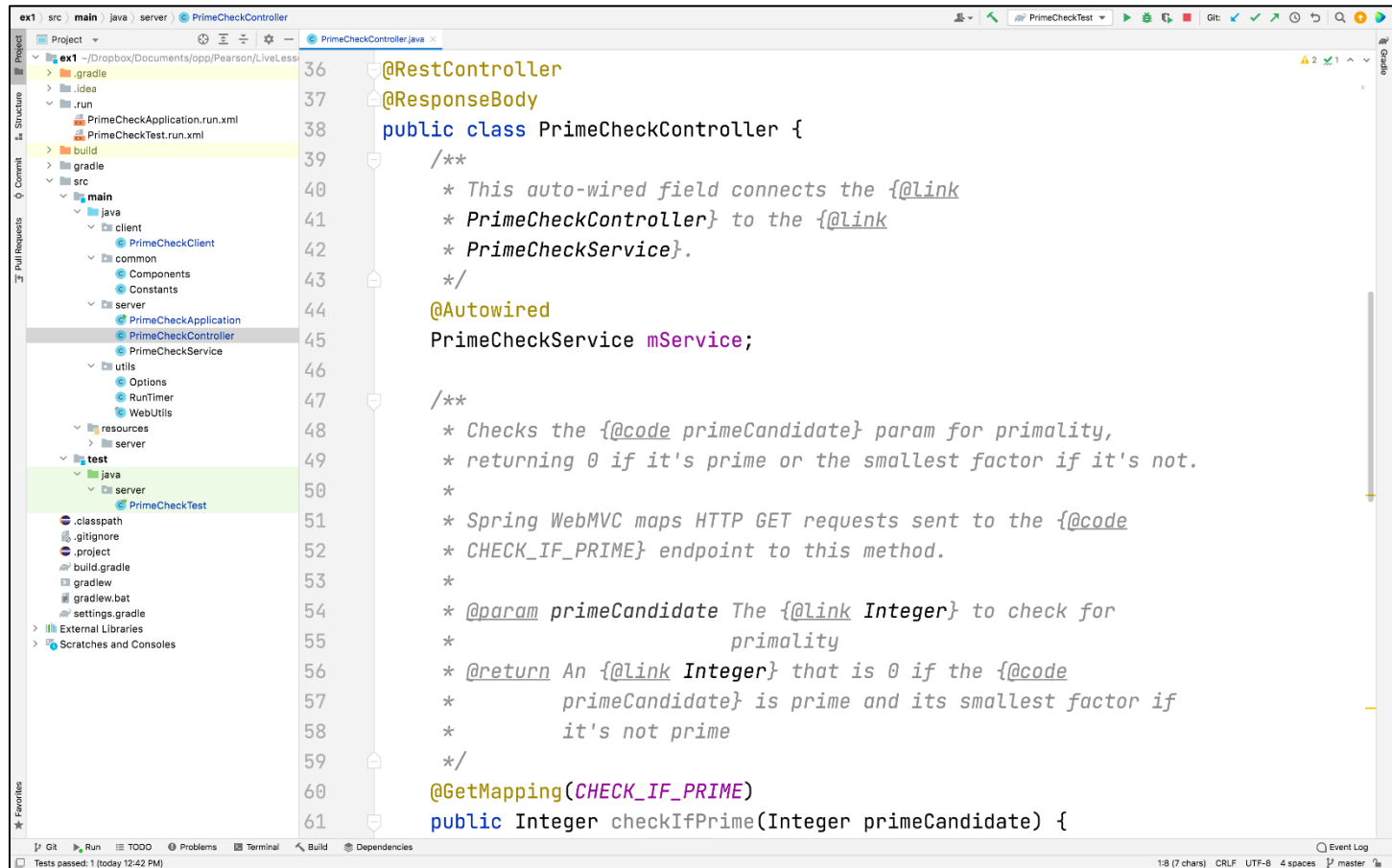


Overview of the PrimeCheckApp Case Study

- The source code is organized into several packages
 - Server
 - Client
 - Test driver
 - Common & utils



Overview of the PrimeCheckApp Case Study



```
36 @RestController
37 @ResponseBody
38 public class PrimeCheckController {
39     /**
40      * This auto-wired field connects the {@link
41      * PrimeCheckController} to the {@link
42      * PrimeCheckService}.
43      */
44     @Autowired
45     PrimeCheckService mService;
46
47     /**
48      * Checks the {@code primeCandidate} param for primality,
49      * returning 0 if it's prime or the smallest factor if it's not.
50      *
51      * Spring WebMvc maps HTTP GET requests sent to the {@code
52      * CHECK_IF_PRIME} endpoint to this method.
53      *
54      * @param primeCandidate The {@link Integer} to check for
55      * primality
56      * @return An {@link Integer} that is 0 if the {@code
57      * primeCandidate} is prime and its smallest factor if
58      * it's not prime
59      */
60     @GetMapping(CHECK_IF_PRIME)
61     public Integer checkIfPrime(Integer primeCandidate) {
```

See github.com/douglasraigschmidt/LiveLessons/tree/master/WebMVC/ex1

End of the PrimeCheck App Case Study