Spring Testing Exercises

Exercise 1: Basic Unit Test for a Service Method

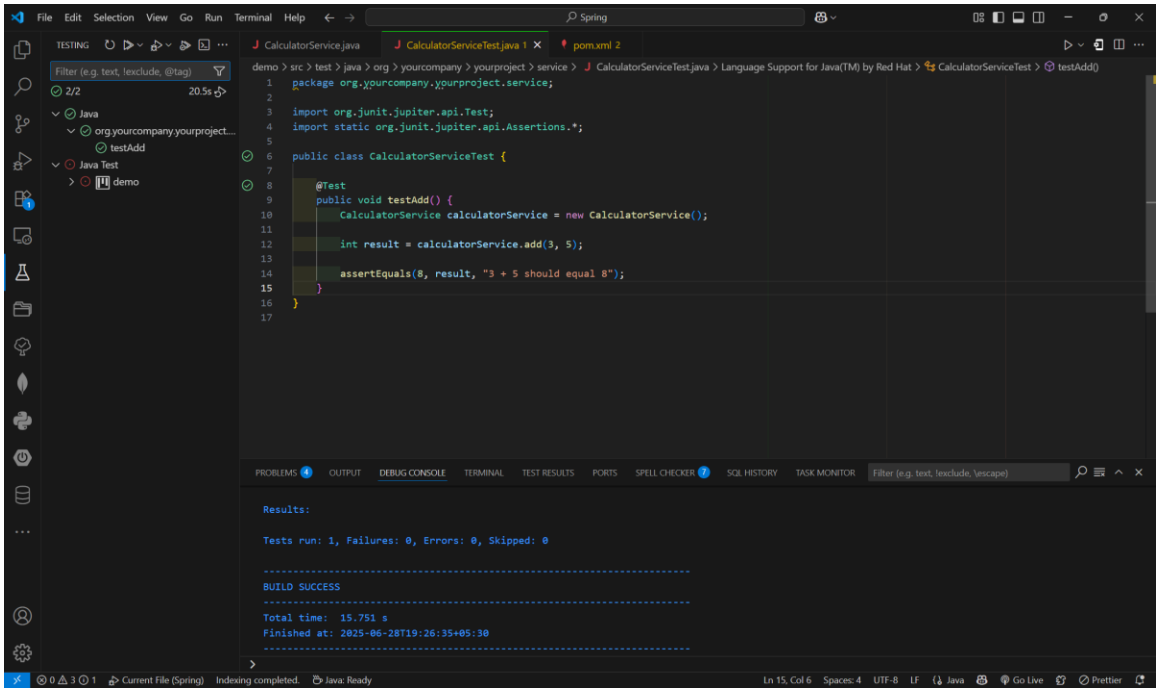
Task: Write a unit test for a service method that adds two numbers.

Service:

```
@Service
public class CalculatorService {
   public int add(int a, int b) {
      return a + b;
   }
}
```

Test:

```
public class CalculatorServiceTest {

   @Test
   public void testAdd() {
      CalculatorService calculatorService = new CalculatorService();

      int result = calculatorService.add(3, 5);

      assertEquals(8, result, "3 + 5 should equal 8");
   }
}
```



Exercise 2: Mocking a Repository in a Service Test

Task: Test a service that uses a repository to fetch data.

Entity:

```
@Entity
public class User {
   @Id
   private Long id;
   private String name;
   // getters and setters
}
```

Repository:

```
public interface UserRepository extends JpaRepository<User, Long> {
}
```

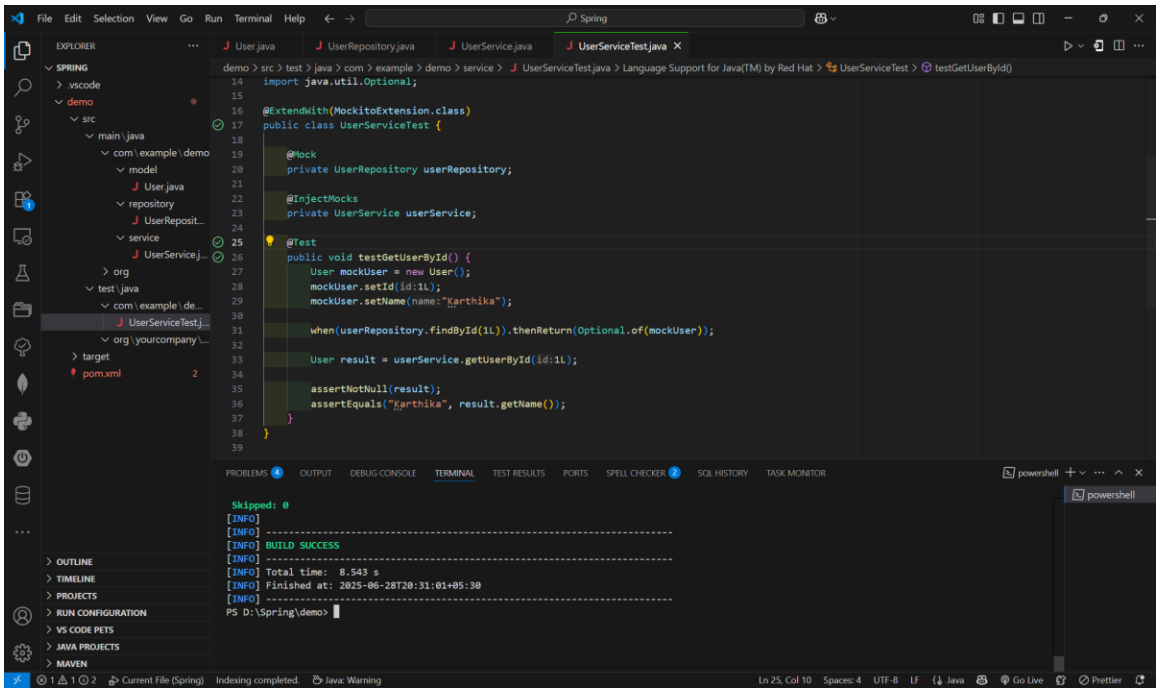
Service:

```
@Service
public class UserService {
   @Autowired
   private UserRepository userRepository;

   public User getUserById(Long id) {
      return userRepository.findById(id).orElse(null);
   }
}
```

Test:

```
package com.example.demo.service;

import com.example.demo.model.User;
import com.example.demo.repository.UserRepository;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.junit.jupiter.MockitoExtension;

import java.util.Optional;
```

```
@ExtendWith(MockitoExtension.class)
public class UserServiceTest {

   @Mock
   private UserRepository userRepository;

   @InjectMocks
   private UserService userService;

   @Test
   public void testGetUserById() {
      User mockUser = new User();
      mockUser.setId(1L);
      mockUser.setName("Karthika");

      when(userRepository.findById(1L)).thenReturn(Optional.of(mockUser));

      User result = userService.getUserById(1L);

      assertNotNull(result);
      assertEquals("Karthika", result.getName());
   }
}
```



Exercise 3: Testing a REST Controller with MockMvc

Task: Test a controller endpoint that returns a user.

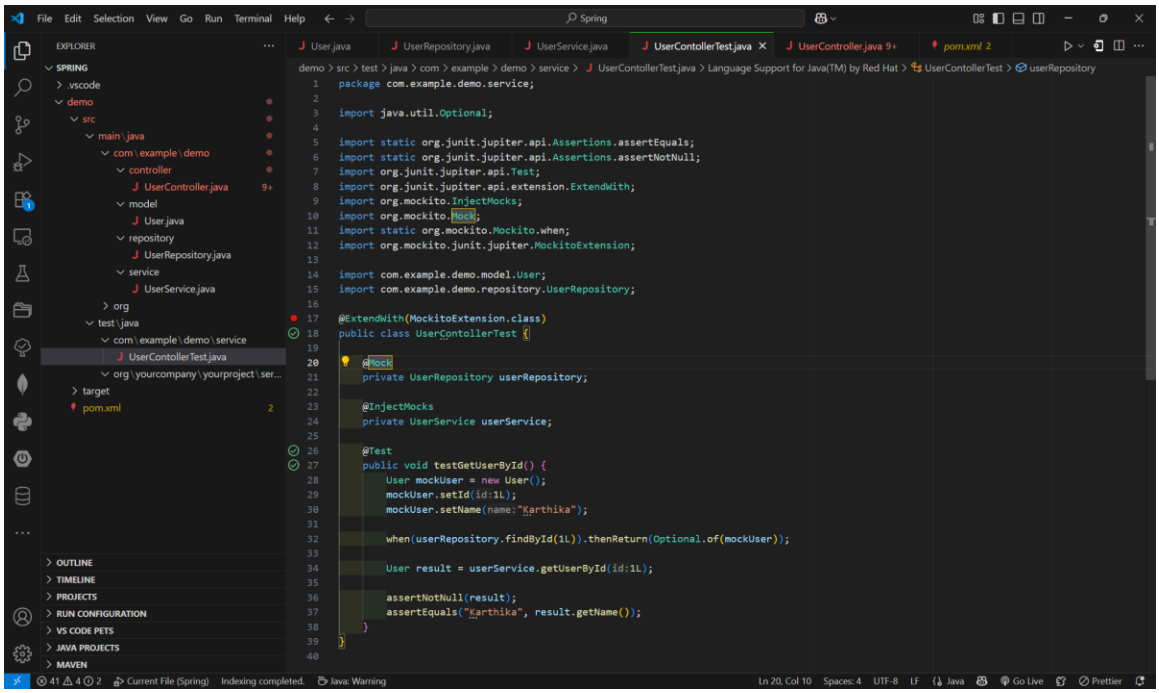
Controller:

```
@RestController
@RequestMapping("/users")
public class UserController {

   @Autowired
   private UserService userService;

   @GetMapping("/{id}")
   public ResponseEntity<User> getUser(@PathVariable Long id) {
      return ResponseEntity.ok(userService.getUserById(id));
   }
}
```

Test:

```
package com.example.demo.service;

import java.util.Optional;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import static org.mockito.Mockito.when;
import org.mockito.junit.jupiter.MockitoExtension;

import com.example.demo.model.User;
import com.example.demo.repository.UserRepository;

@ExtendWith(MockitoExtension.class)
public class UserContollerTest {

   @Mock
   private UserRepository userRepository;

   @InjectMocks
   private UserService userService;

   @Test
   public void testGetUserById() {
      User mockUser = new User();
      mockUser.setId(1L);
      mockUser.setName("Karthika");
```

```
    when(userRepository.findById(1L)).thenReturn(Optional.of(mockUser));

    User result = userService.getUserById(1L);

    assertNotNull(result);
    assertEquals("Karthika", result.getName());
  }
}
```



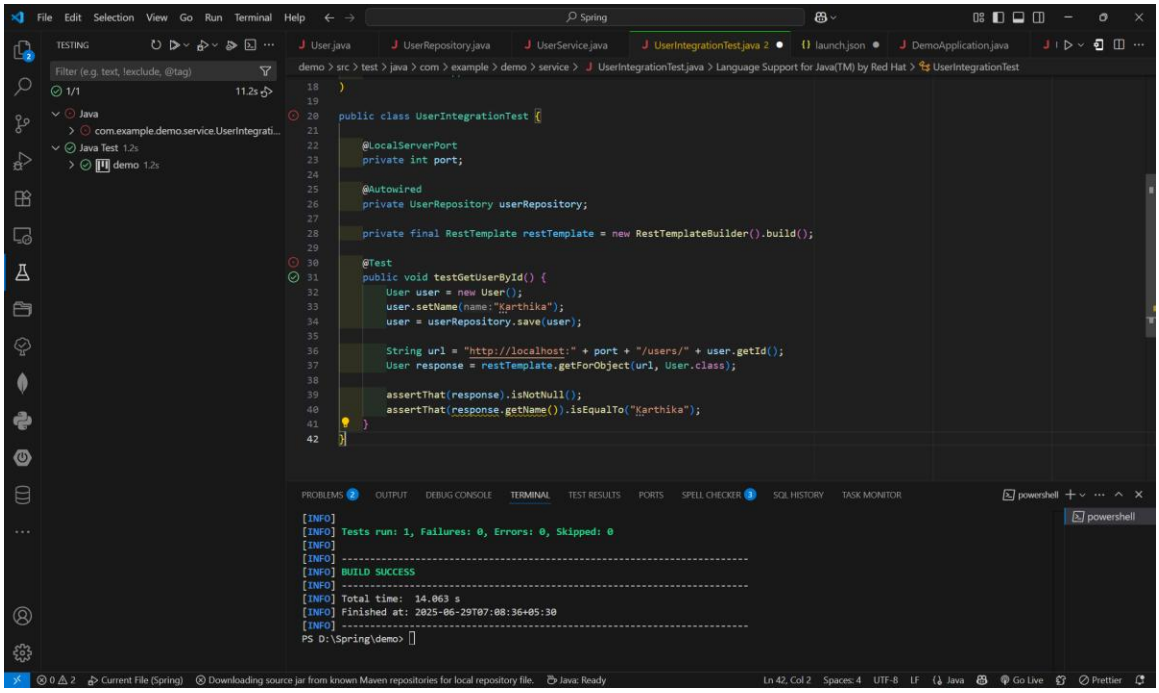
Exercise 4: Integration Test with Spring Boot

Task: Write an integration test that tests the full flow from controller to database.

Test:

```
public class UserIntegrationTest {

  @LocalServerPort
  private int port;

  @Autowired
  private UserRepository userRepository;

  private final RestTemplate restTemplate = new RestTemplateBuilder().build();

  @Test
  public void testGetUserById() {
    User user = new User();
    user.setName("Karthika");
```

```
    user = userRepository.save(user);

    String url = "http://localhost:" + port + "/users/" + user.getId();
    User response = restTemplate.getForObject(url, User.class);

    assertThat(response).isNotNull();
    assertThat(response.getName()).isEqualTo("Karthika");
  }
}
```



Exercise 5: Test Controller POST Endpoint

Task: Test a POST endpoint that creates a user.

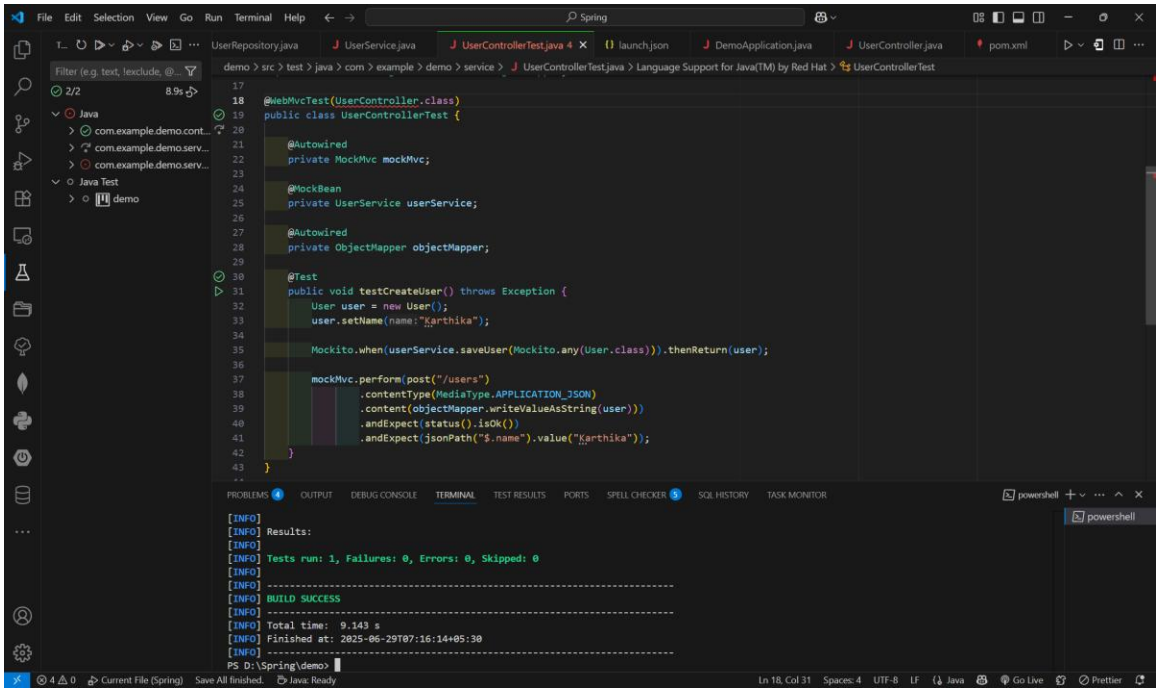
Controller:

```
@PostMapping
public ResponseEntity<User> createUser(@RequestBody User user) {
  return ResponseEntity.ok(userService.saveUser(user));
}
```

Test:

```
public class UserControllerTest {

  @Autowired
  private MockMvc mockMvc;
```

```
  @MockBean
  private UserService userService;

  @Autowired
  private ObjectMapper objectMapper;

  @Test
  public void testCreateUser() throws Exception {
    User user = new User();
    user.setName("Karthika");

    Mockito.when(userService.saveUser(Mockito.any(User.class))).thenReturn(user);

    mockMvc.perform(post("/users")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(user)))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.name").value("Karthika"));
  }
}
```



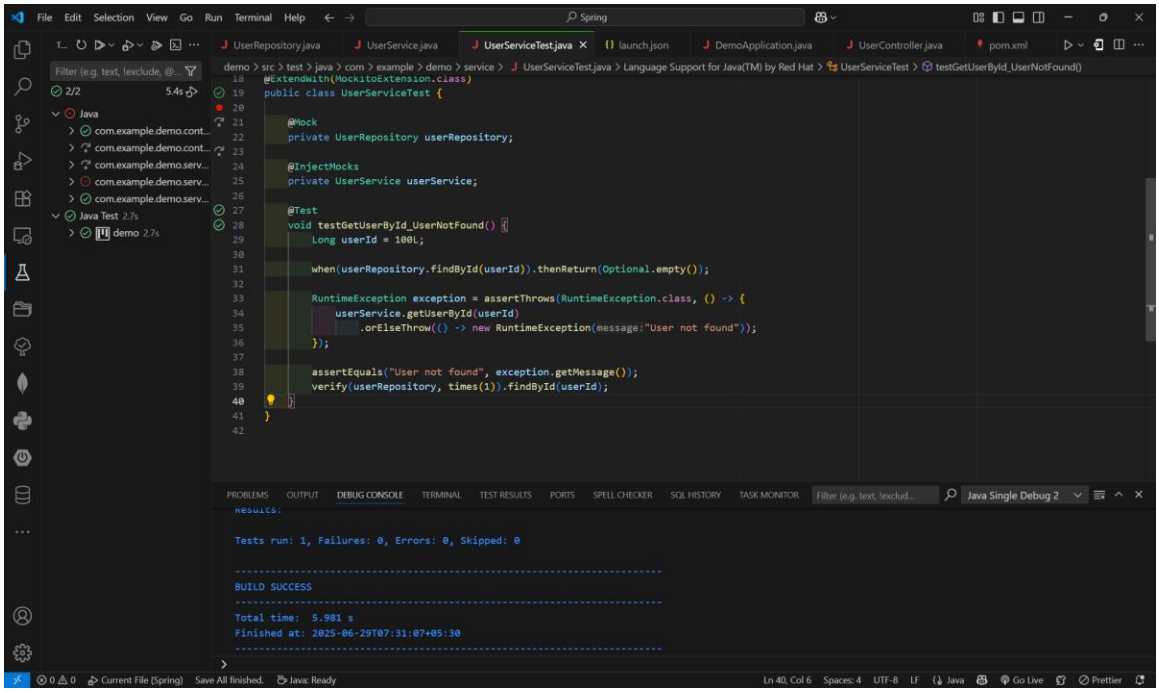
Exercise 6: Test Service Exception Handling

Task: Test how a service handles a missing user.

Test:

```
public class UserServiceTest {

  @Mock
  private UserRepository userRepository;

  @InjectMocks
```

```
  private UserService userService;

  @Test
  void testGetUserById_UserNotFound() {
    Long userId = 100L;

    when(userRepository.findById(userId)).thenReturn(Optional.empty());

    RuntimeException exception = assertThrows(RuntimeException.class, () -> {
      userService.getUserById(userId)
        .orElseThrow(() -> new RuntimeException("User not found"));
    });

    assertEquals("User not found", exception.getMessage());
    verify(userRepository, times(1)).findById(userId);
  }
}
```



Exercise 7: Test Custom Repository Query

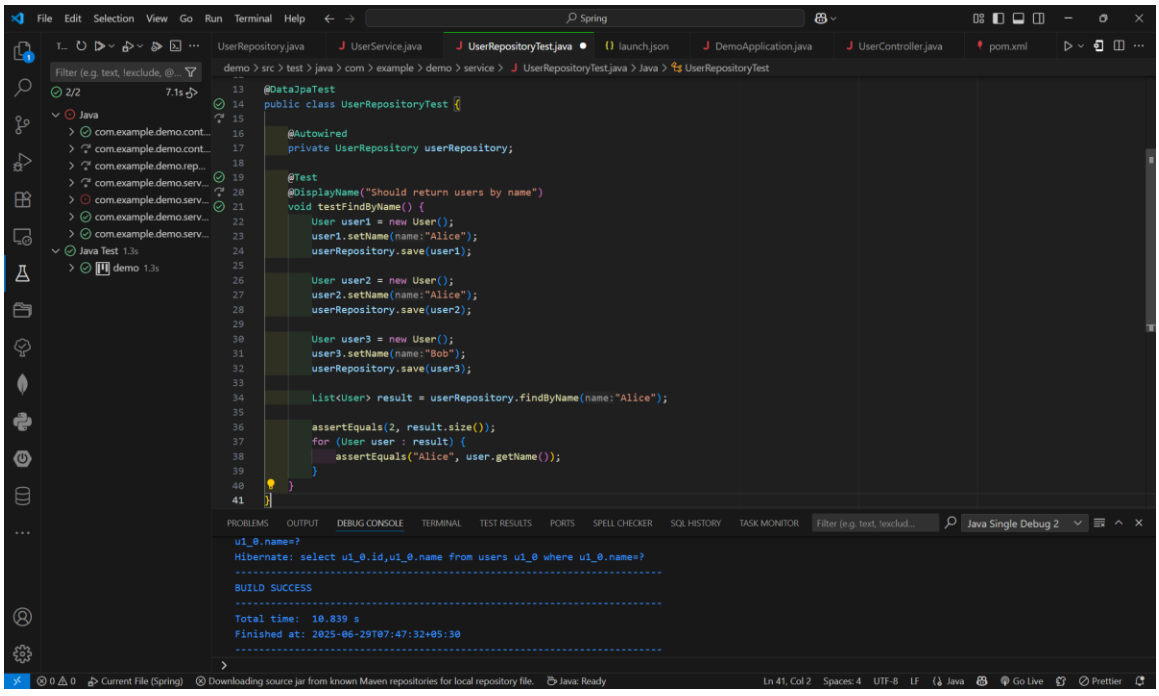
Task: Add and test a custom query method.

Repository:

```
public interface UserRepository extends JpaRepository<User, Long> {
  List<User> findByName(String name);
}
```

Test:

```
@DataJpaTest
public class UserRepositoryTest {

  @Autowired
  private UserRepository userRepository;

  @Test
  @DisplayName("Should return users by name")
  void testFindByName() {
    User user1 = new User();
    user1.setName("Alice");
    userRepository.save(user1);

    User user2 = new User();
    user2.setName("Alice");
    userRepository.save(user2);

    User user3 = new User();
    user3.setName("Bob");
    userRepository.save(user3);

    List<User> result = userRepository.findByName("Alice");

    assertEquals(2, result.size());
    for (User user : result) {
      assertEquals("Alice", user.getName());
    }
  }
}
```



Exercise 8: Test Controller Exception Handling

Task: Add and test a @ControllerAdvice for handling exceptions.

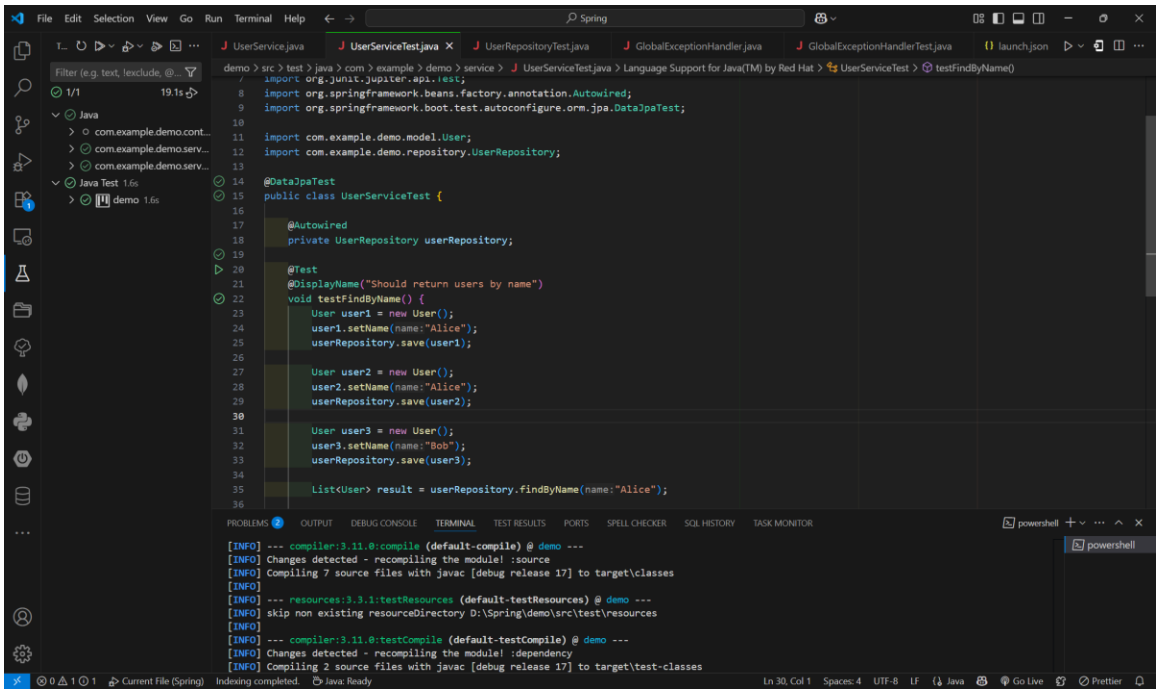
Exception Handler:

```
@ControllerAdvice
public class GlobalExceptionHandler {

   @ExceptionHandler(NoSuchElementException.class)

public ResponseEntity<String>

   handleNotFound(NoSuchElementException ex) { return

   ResponseEntity.status(HttpStatus.NOT_FOUND).body("User not

   found");

   }
}
```

Test:

```
@DataJpaTest
public class UserServiceTest {

   @Autowired
   private UserRepository userRepository;

   @Test
   @DisplayName("Should return users by name")
   void testFindByName() {
      User user1 = new User();
      user1.setName("Alice");
      userRepository.save(user1);

      User user2 = new User();
      user2.setName("Alice");
      userRepository.save(user2);

      User user3 = new User();
      user3.setName("Bob");
      userRepository.save(user3);

      List<User> result = userRepository.findByName("Alice");

      assertEquals(2, result.size());
      for (User user : result) {
```

```
        assertEquals("Alice", user.getName());
    }
  }
}
```

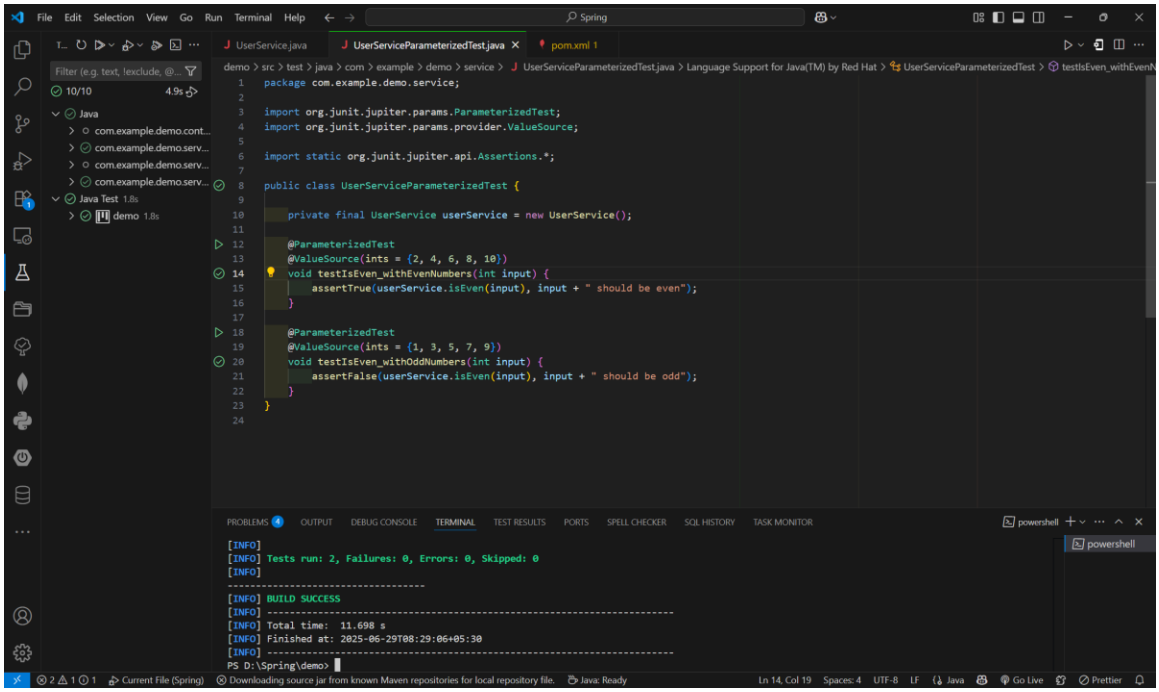


Exercise 9: Parameterized Test with JUnit

Task: Use @ParameterizedTest to test multiple inputs.

Test:

```
public class UserServiceParameterizedTest {

  private final UserService userService = new UserService();

  @ParameterizedTest
  @ValueSource(ints = {2, 4, 6, 8, 10})
  void testIsEven_withEvenNumbers(int input) {
    assertTrue(userService.isEven(input), input + " should be even");
  }

  @ParameterizedTest
  @ValueSource(ints = {1, 3, 5, 7, 9})
  void testIsEven_withOddNumbers(int input) {
    assertFalse(userService.isEven(input), input + " should be odd");
  }
}
```


File Edit Selection View Go Run Terminal Help
Spring
UserService.java
UserServiceParameterizedTest.java
pom.xml 1
demo > src > test > java > com > example > demo > service > UserServiceParameterizedTest.java > Language Support for Java(TM) by Red Hat > UserServiceParameterizedTest > testIsEven_withEvenN
10/10 4.9s
Java
com.example.demo.cont...
com.example.demo.serv...
com.example.demo.serv...
com.example.demo.serv...
Java Test 1.8s
demo 1.8s
1 package com.example.demo.service;
2
3 import org.junit.jupiter.params.ParameterizedTest;
4 import org.junit.jupiter.params.provider.ValueSource;
5
6 import static org.junit.jupiter.api.Assertions.*;
7
8 public class UserServiceParameterizedTest {
9
10 private final UserService userService = new UserService();
11
12 @ParameterizedTest
13 @ValueSource(ints = {2, 4, 6, 8, 10})
14 void testIsEven_withEvenNumbers(int input) {
15 assertTrue(userService.isEven(input), input + " should be even");
16 }
17
18 @ParameterizedTest
19 @ValueSource(ints = {1, 3, 5, 7, 9})
20 void testIsEven_withOddNumbers(int input) {
21 assertFalse(userService.isEven(input), input + " should be odd");
22 }
23 }
24
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL TEST RESULTS PORTS SPELL CHECKER SQL HISTORY TASK MONITOR
powershell
powershell
[INFO]
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] BUILD SUCCESS
[INFO] Total time: 11.698 s
[INFO] Finished at: 2025-06-29T08:29:06+05:30
PS D:\Spring\demo>
Current File (Spring)
Downloading source jar from known Maven repositories for local repository file.
Java: Ready
Ln 14, Col 19 Spaces: 4 UTF-8 LF Java Go Live Prettier