# Spring JUnit Testing Exercises - Solutions

## Exercise 1: Basic Unit Test for a Service Method

import static org.junit.jupiter.api.Assertions.\*;  
import org.junit.jupiter.api.Test;  
  
public class CalculatorServiceTest {  
 private CalculatorService calculatorService = new CalculatorService();  
  
 @Test  
 public void testAdd() {  
 assertEquals(5, calculatorService.add(2, 3));  
 }  
}

## Exercise 2: Mocking a Repository in a Service Test

import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
import org.junit.jupiter.api.Test;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.MockitoAnnotations;  
import java.util.Optional;  
  
public class UserServiceTest {  
 @Mock  
 private UserRepository userRepository;  
  
 @InjectMocks  
 private UserService userService;  
  
 public UserServiceTest() {  
 MockitoAnnotations.openMocks(this);  
 }  
  
 @Test  
 public void testGetUserById() {  
 User user = new User();  
 user.setId(1L);  
 user.setName("John");  
 when(userRepository.findById(1L)).thenReturn(Optional.of(user));  
  
 User result = userService.getUserById(1L);  
 assertNotNull(result);  
 assertEquals("John", result.getName());  
 }  
}

## Exercise 3: Testing a REST Controller with MockMvc

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;  
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;  
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;  
import static org.mockito.Mockito.\*;  
import org.junit.jupiter.api.Test;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;  
import org.springframework.boot.test.mock.mockito.MockBean;  
import org.springframework.test.web.servlet.MockMvc;  
  
@WebMvcTest(UserController.class)  
public class UserControllerTest {  
  
 @Autowired  
 private MockMvc mockMvc;  
  
 @MockBean  
 private UserService userService;  
  
 @Test  
 public void testGetUser() throws Exception {  
 User user = new User();  
 user.setId(1L);  
 user.setName("John");  
 when(userService.getUserById(1L)).thenReturn(user);  
  
 mockMvc.perform(get("/users/1"))  
 .andExpect(status().isOk())  
 .andExpect(jsonPath("$.name").value("John"));  
 }  
}

## Exercise 4: Integration Test with Spring Boot

import org.junit.jupiter.api.Test;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import static org.junit.jupiter.api.Assertions.\*;  
  
@SpringBootTest  
public class IntegrationTest {  
 @Autowired  
 private UserService userService;  
  
 @Test  
 public void testFullFlow() {  
 User user = new User();  
 user.setName("Alice");  
 User savedUser = userService.saveUser(user);  
 assertNotNull(savedUser.getId());  
 }  
}

## Exercise 5: Test Controller POST Endpoint

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;  
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;  
import com.fasterxml.jackson.databind.ObjectMapper;  
...  
@Test  
public void testCreateUser() throws Exception {  
 User user = new User();  
 user.setName("Jane");  
 when(userService.saveUser(any(User.class))).thenReturn(user);  
  
 mockMvc.perform(post("/users")  
 .contentType("application/json")  
 .content(new ObjectMapper().writeValueAsString(user)))  
 .andExpect(status().isOk());  
}

## Exercise 6: Test Service Exception Handling

@Test  
public void testUserNotFound() {  
 when(userRepository.findById(999L)).thenReturn(Optional.empty());  
 User result = userService.getUserById(999L);  
 assertNull(result);  
}

## Exercise 7: Test Custom Repository Query

@Test  
public void testFindByName() {  
 User user = new User();  
 user.setName("Sam");  
 when(userRepository.findByName("Sam")).thenReturn(List.of(user));  
 List<User> users = userRepository.findByName("Sam");  
 assertEquals(1, users.size());  
}

## Exercise 8: Test Controller Exception Handling

@Test  
public void testExceptionHandling() throws Exception {  
 when(userService.getUserById(99L)).thenThrow(new NoSuchElementException());  
  
 mockMvc.perform(get("/users/99"))  
 .andExpect(status().isNotFound())  
 .andExpect(content().string("User not found"));  
}

## Exercise 9: Parameterized Test with JUnit

import org.junit.jupiter.params.ParameterizedTest;  
import org.junit.jupiter.params.provider.CsvSource;  
  
@ParameterizedTest  
@CsvSource({ "2, true", "3, false", "4, true" })  
public void testIsEven(int number, boolean expected) {  
 CalculatorService service = new CalculatorService();  
 assertEquals(expected, service.isEven(number));  
}