SUPERSET ID-6416829

# Exercise 1: Setting Up JUnit

Test Code:  
@Test  
public void sampleTest() {  
 System.out.println("JUnit is set up correctly!");  
}

Output:

Running ExampleTest  
JUnit is set up correctly!  
Tests run: 1, Failures: 0

# Exercise 2: Writing Basic JUnit Tests

Main Code:

public class Calculator {  
 public int add(int a, int b) { return a + b; }  
 public int subtract(int a, int b) { return a - b; }  
}

Test Code:

@Test  
public void testAddition() {  
 assertEquals(10, calc.add(7, 3));  
}  
  
@Test  
public void testSubtraction() {  
 assertEquals(4, calc.subtract(10, 6));  
}

Output:

Running CalculatorTest  
Tests run: 2, Failures: 0

# Exercise 3: Assertions in JUnit

Test Code:

@Test  
public void testAssertions() {  
 assertEquals(5, 2 + 3); // Pass  
 assertTrue(5 > 3); // Pass  
 assertFalse(5 < 3); // Pass  
 assertNull(null); // Pass  
 assertNotNull(new Object()); // Pass  
}

Output:

Running AssertionsTest  
Tests run: 1, Failures: 0

# Exercise 4: AAA Pattern, Setup & Teardown

Test Code:

@Before  
public void setUp() {  
 System.out.println("Setting up Calculator object...");  
}  
  
@After  
public void tearDown() {  
 System.out.println("Tearing down...");  
}  
  
@Test  
public void testAddAAA() {  
 assertEquals(10, calc.add(4, 6));  
}  
  
@Test  
public void testSubtractAAA() {  
 assertEquals(7, calc.subtract(9, 2));  
}

Output:

Running CalculatorLifecycleTest  
Setting up Calculator object...  
Tearing down...  
Setting up Calculator object...  
Tearing down...  
Tests run: 2, Failures: 0

# Advanced JUnit Testing Exercises

**Exercise 1: Parameterized Tests**

**Code:**

**EvenChecker.java**

public class EvenChecker {  
 public boolean isEven(int number) {  
 return number % 2 == 0;  
 }  
}

**EvenCheckerTest.java**

import static org.junit.jupiter.api.Assertions.\*;  
import org.junit.jupiter.params.ParameterizedTest;  
import org.junit.jupiter.params.provider.ValueSource;  
  
public class EvenCheckerTest {  
  
 EvenChecker checker = new EvenChecker();  
  
 @ParameterizedTest  
 @ValueSource(ints = {2, 4, 6, 8, 10})  
 void testEvenNumbers(int number) {  
 assertTrue(checker.isEven(number));  
 }  
  
 @ParameterizedTest  
 @ValueSource(ints = {1, 3, 5, 7, 9})  
 void testOddNumbers(int number) {  
 assertFalse(checker.isEven(number));  
 }  
}

**Output:**

testEvenNumbers(2)  
 testEvenNumbers(4)  
 testEvenNumbers(6)  
 testEvenNumbers(8)  
 testEvenNumbers(10)  
 testOddNumbers(1)  
 testOddNumbers(3)  
 testOddNumbers(5)  
 testOddNumbers(7)  
 testOddNumbers(9)

**Exercise 2: Test Suites and Categories**

**Code:**

**MathTest.java**

import static org.junit.jupiter.api.Assertions.\*;  
import org.junit.jupiter.api.Test;  
  
public class MathTest {  
 @Test  
 void testAdd() {  
 assertEquals(5, 2 + 3);  
 }  
}

**StringTest.java**

import static org.junit.jupiter.api.Assertions.\*;  
import org.junit.jupiter.api.Test;  
  
public class StringTest {  
 @Test  
 void testLength() {  
 assertEquals(4, "test".length());  
 }  
}

**AllTests.java**

import org.junit.platform.suite.api.SelectClasses;  
import org.junit.platform.suite.api.Suite;  
  
@Suite  
@SelectClasses({ MathTest.class, StringTest.class })  
public class AllTests {  
}

**Output:**

testAdd()  
 testLength()

**Exercise 3: Test Execution Order**

**Code:**

**OrderedTests.java**

import static org.junit.jupiter.api.Assertions.\*;  
import org.junit.jupiter.api.\*;  
  
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)  
public class OrderedTests {  
  
 @Test  
 @Order(2)  
 void secondTest() {  
 System.out.println("Second test executed");  
 }  
  
 @Test  
 @Order(1)  
 void firstTest() {  
 System.out.println("First test executed");  
 }  
  
 @Test  
 @Order(3)  
 void thirdTest() {  
 System.out.println("Third test executed");  
 }  
}

**Output:**

First test executed  
Second test executed  
Third test executed

**Exercise 4: Exception Testing**

**Code:**

**ExceptionThrower.java**

public class ExceptionThrower {  
 public void throwException() {  
 throw new IllegalArgumentException("Invalid input");  
 }  
}

**ExceptionThrowerTest.java**

import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class ExceptionThrowerTest {  
  
 ExceptionThrower thrower = new ExceptionThrower();  
  
 @Test  
 void testException() {  
 assertThrows(IllegalArgumentException.class, () -> {  
 thrower.throwException();  
 });  
 }  
}

**Output:**

testException()

**Exercise 5: Timeout and Performance Testing**

**Code:**

**PerformanceTester.java**

public class PerformanceTester {  
 public void performTask() throws InterruptedException {  
 Thread.sleep(500); // 0.5 second delay  
 }  
}

**PerformanceTesterTest.java**

import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
  
import java.time.Duration;  
  
public class PerformanceTesterTest {  
  
 PerformanceTester tester = new PerformanceTester();  
  
 @Test  
 void testTimeout() {  
 assertTimeout(Duration.ofSeconds(1), () -> {  
 tester.performTask();  
 });  
 }  
}

**Output:**

testTimeout() (completed in < 1 second)

# Spring Testing Exercises

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

@Service  
public class CalculatorService {  
 public int add(int a, int b) {  
 return a + b;  
 }  
}  
@Test  
public void testAdd() {  
 assertEquals(5, calculatorService.add(2, 3));  
}  
 **Output:**  
Test passed: testAdd

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

@Mock  
private UserRepository userRepository;  
  
@InjectMocks  
private UserService userService;  
  
@Test  
public void testGetUserById() {  
 User user = new User();  
 user.setId(1L);  
 user.setName("Alice");  
  
 when(userRepository.findById(1L)).thenReturn(Optional.of(user));  
 User result = userService.getUserById(1L);  
 assertEquals("Alice", result.getName());  
}  
 **Output:**  
Test passed: testGetUserById

**Exercise 3: Testing a REST Controller with 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("Alice");  
  
 when(userService.getUserById(1L)).thenReturn(user);  
 mockMvc.perform(get("/users/1"))  
 .andExpect(status().isOk())  
 .andExpect(jsonPath("$.name").value("Alice"));  
 }  
}  
 **Output:**  
MockMvc test passed for /users/1

**Exercise 4: Integration Test with Spring Boot**

@SpringBootTest  
@AutoConfigureMockMvc  
public class UserIntegrationTest {  
  
 @Autowired  
 private MockMvc mockMvc;  
  
 @Autowired  
 private UserRepository userRepository;  
  
 @BeforeEach  
 public void setup() {  
 userRepository.deleteAll();  
 User user = new User();  
 user.setId(1L);  
 user.setName("Alice");  
 userRepository.save(user);  
 }  
  
 @Test  
 public void testGetUserIntegration() throws Exception {  
 mockMvc.perform(get("/users/1"))  
 .andExpect(status().isOk())  
 .andExpect(jsonPath("$.name").value("Alice"));  
 }  
}  
 **Output:**  
Integration test passed: /users/1 returns Alice

**Exercise 5: Test Controller POST Endpoint**

@WebMvcTest(UserController.class)  
public class CreateUserControllerTest {  
  
 @Autowired  
 private MockMvc mockMvc;  
  
 @MockBean  
 private UserService userService;  
  
 @Test  
 public void testCreateUser() throws Exception {  
 User user = new User(1L, "Bob");  
 when(userService.saveUser(any(User.class))).thenReturn(user);  
  
 mockMvc.perform(post("/users")  
 .contentType(MediaType.APPLICATION\_JSON)  
 .content("{\"id\":1,\"name\":\"Bob\"}"))  
 .andExpect(status().isOk())  
 .andExpect(jsonPath("$.name").value("Bob"));  
 }  
}  
 **Output:**  
POST /users test passed: returns Bob

**Exercise 6: Test Service Exception Handling**

@ExtendWith(MockitoExtension.class)  
public class UserServiceExceptionTest {  
  
 @Mock  
 private UserRepository userRepository;  
  
 @InjectMocks  
 private UserService userService;  
  
 @Test  
 public void testUserNotFound() {  
 when(userRepository.findById(100L)).thenReturn(Optional.empty());  
 User result = userService.getUserById(100L);  
 assertNull(result);  
 }  
}  
 **Output:**  
Test passed: user not found returns null

**Exercise 7: Test Custom Repository Query**

@DataJpaTest  
public class UserRepositoryTest {  
  
 @Autowired  
 private UserRepository userRepository;  
  
 @BeforeEach  
 public void setup() {  
 userRepository.save(new User(1L, "John"));  
 userRepository.save(new User(2L, "John"));  
 userRepository.save(new User(3L, "Alice"));  
 }  
  
 @Test  
 public void testFindByName() {  
 List<User> users = userRepository.findByName("John");  
 assertEquals(2, users.size());  
 }  
}  
 **Output:**  
Test passed: 2 users found with name John

**Exercise 8: Test Controller Exception Handling**

@WebMvcTest(UserController.class)  
public class ExceptionHandlerTest {  
  
 @Autowired  
 private MockMvc mockMvc;  
  
 @MockBean  
 private UserService userService;  
  
 @Test  
 public void testUserNotFoundException() throws Exception {  
 when(userService.getUserById(1L)).thenThrow(new NoSuchElementException());  
  
 mockMvc.perform(get("/users/1"))  
 .andExpect(status().isNotFound())  
 .andExpect(content().string("User not found"));  
 }  
}  
**Output:**  
Exception test passed: NoSuchElementException handled with 404

**Exercise 9: Parameterized Test with JUnit**

class CalculatorServiceParamTest {  
  
 private final CalculatorService service = new CalculatorService();  
  
 @ParameterizedTest  
 @CsvSource({  
 "1,2,3",  
 "2,3,5",  
 "-1,-1,-2",  
 "0,0,0"  
 })  
 void testAdd(int a, int b, int expected) {  
 assertEquals(expected, service.add(a, b));  
 }  
}  
**Output:**  
Test passed: All add() combinations