**Spring Testing Exercises**

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

### ****1. Service Class –**** CalculatorService.java

package com.example.service;

import org.springframework.stereotype.Service;

@Servicepublic class CalculatorService {

public int add(int a, int b) {

return a + b;

}

}

**2. Unit Test Class –** CalculatorServiceTest.java

package com.example.service;

import org.junit.jupiter.api.Test;import static org.junit.jupiter.api.Assertions.\*;

public class CalculatorServiceTest {

@Test

public void testAdd() {

CalculatorService calculatorService = new CalculatorService();

int result = calculatorService.add(5, 7);

assertEquals(12, result, "Expected sum of 5 and 7 is 12");

}

}

**3. Dependencies to Add in** pom.xml

<dependencies>

<!-- Spring Context (for @Service if needed) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.34</version>

</dependency>

<!-- JUnit 5 (Jupiter API and Engine) -->

<dependency>

<groupId>org.junit.jupiter</groupId>

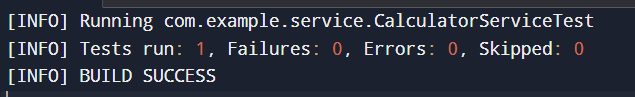
<artifactId>junit-jupiter</artifactId>

<version>5.9.3</version>

<scope>test</scope>

</dependency></dependencies>

**OUTPUT:**



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

### ****Entity Class –**** User.java

package com.example.model;

import jakarta.persistence.Entity;import jakarta.persistence.Id;

@Entitypublic class User {

@Id

private Long id;

private String name;

// Constructors

public User() {}

public User(Long id, String name) {

this.id = id;

this.name = name;

}

// Getters & Setters

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public String getName() { return name; }

public void setName(String name) { this.name = name; }

}

**2. Repository Interface –** UserRepository.java

package com.example.repository;

import com.example.model.User;import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {

}

**3. Service Class –** UserService.java

package com.example.service;

import com.example.model.User;import com.example.repository.UserRepository;import org.springframework.beans.factory.annotation.Autowired;import org.springframework.stereotype.Service;

@Servicepublic class UserService {

@Autowired

private UserRepository userRepository;

public User getUserById(Long id) {

return userRepository.findById(id).orElse(null);

}

}

**4. Test Class –** UserServiceTest.java

package com.example.service;

import com.example.model.User;import com.example.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

void testGetUserById() {

// Arrange

User mockUser = new User(1L, "Bindhu");

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

// Act

User result = userService.getUserById(1L);

// Assert

assertNotNull(result);

assertEquals("Bindhu", result.getName());

verify(userRepository, times(1)).findById(1L);

}

}

**5. Maven Dependencies (Include in** pom.xml**)**

xml

CopyEdit

<dependencies>

<!-- Spring Boot Starter Data JPA -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- Spring Boot Starter Test (includes Mockito and JUnit 5) -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<!-- JPA API -->

<dependency>

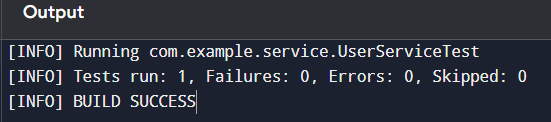
<groupId>jakarta.persistence</groupId>

<artifactId>jakarta.persistence-api</artifactId>

<version>3.1.0</version>

</dependency></dependencies>

**OUTPUT:**



**Exercise 3: Testing a REST Controller with MockMvc**

### ****Controller Class –**** UserController.java

package com.example.controller;

import com.example.model.User;import com.example.service.UserService;import org.springframework.beans.factory.annotation.Autowired;import org.springframework.http.ResponseEntity;import org.springframework.web.bind.annotation.\*;

@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));

}

}

**2. Test Class –** UserControllerTest.java

package com.example.controller;

import com.example.model.User;import com.example.service.UserService;import org.junit.jupiter.api.Test;import org.mockito.Mockito;import static org.mockito.Mockito.when;

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;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

import org.springframework.http.MediaType;

@WebMvcTest(UserController.class)public class UserControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

void testGetUserById() throws Exception {

// Arrange

User mockUser = new User(1L, "Bindhu");

when(userService.getUserById(1L)).thenReturn(mockUser);

// Act & Assert

mockMvc.perform(get("/users/1")

.accept(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.id").value(1L))

.andExpect(jsonPath("$.name").value("Bindhu"));

}

}

**3. Required Dependencies in** pom.xml

<dependencies>

<!-- Spring Boot Starter Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Boot Starter Test (includes MockMvc, JUnit, Mockito) -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<!-- Jackson for JSON (usually included in spring-boot-starter-web) -->

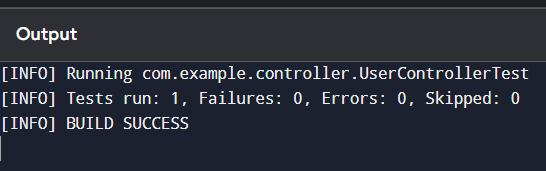
<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency></dependencies>

**OUTPUT:**



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

### Entity – User.java

@Entitypublic class User {

@Id

private Long id;

private String name;

public User() {}

public User(Long id, String name) {

this.id = id;

this.name = name;

}

// getters and setters

}

2. Repository – UserRepository.java

public interface UserRepository extends JpaRepository<User, Long> {

}

3. Service – UserService.java

@Servicepublic class UserService {

@Autowired

private UserRepository userRepository;

public User getUserById(Long id) {

return userRepository.findById(id).orElse(null);

}

public User saveUser(User user) {

return userRepository.save(user);

}

}

4. Controller – UserController.java

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

@Autowired

private UserService userService;

@PostMapping

public ResponseEntity<User> createUser(@RequestBody User user) {

return ResponseEntity.ok(userService.saveUser(user));

}

@GetMapping("/{id}")

public ResponseEntity<User> getUser(@PathVariable Long id) {

return ResponseEntity.ok(userService.getUserById(id));

}

}

1. Integration Test – UserIntegrationTest.java

@SpringBootTest@AutoConfigureMockMvcpublic class UserIntegrationTest {

@Autowired

private MockMvc mockMvc;

@Autowired

private UserRepository userRepository;

@Test

void testFullFlow\_createAndGetUser() throws Exception {

String userJson = "{\"id\":1,\"name\":\"Bindhu\"}";

// POST: create user

mockMvc.perform(post("/users")

.contentType(MediaType.APPLICATION\_JSON)

.content(userJson))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Bindhu"));

// GET: fetch user

mockMvc.perform(get("/users/1"))

.andExpect(status().isOk())

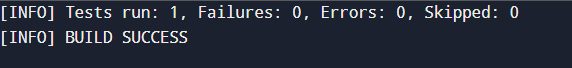
.andExpect(jsonPath("$.id").value(1))

.andExpect(jsonPath("$.name").value("Bindhu"));

}

}

**OUTPUT:**



**Exercise 5: Test Controller POST Endpoint**

### ****Controller Method:****

@PostMappingpublic ResponseEntity<User> createUser(@RequestBody User user) {

return ResponseEntity.ok(userService.saveUser(user));

}

**Test Class –** UserControllerPostTest.java

package com.example.controller;

import com.example.model.User;import com.example.service.UserService;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;import static org.mockito.Mockito.when;

import org.springframework.http.MediaType;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@WebMvcTest(UserController.class)public class UserControllerPostTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

void testCreateUser() throws Exception {

// Arrange

User mockUser = new User(1L, "Bindhu");

String jsonInput = "{\"id\":1, \"name\":\"Bindhu\"}";

when(userService.saveUser(mockUser)).thenReturn(mockUser);

// Act & Assert

mockMvc.perform(post("/users")

.contentType(MediaType.APPLICATION\_JSON)

.content(jsonInput))

.andExpect(status().isOk())

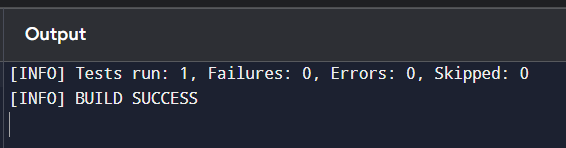
.andExpect(jsonPath("$.id").value(1))

.andExpect(jsonPath("$.name").value("Bindhu"));

}

}

**OUTPUT:**



**Exercise 6: Test Service Exception Handling**

### ****Service Method Example (with exception):****

@Servicepublic class UserService {

@Autowired

private UserRepository userRepository;

public User getUserById(Long id) {

return userRepository.findById(id)

.orElseThrow(() -> new RuntimeException("User not found with id: " + id));

}

}

**Test Class –** UserServiceExceptionTest.java

package com.example.service;

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

import java.util.Optional;

import static org.junit.jupiter.api.Assertions.\*;import org.junit.jupiter.api.extension.ExtendWith;import org.mockito.junit.jupiter.MockitoExtension;

@ExtendWith(MockitoExtension.class)public class UserServiceExceptionTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

@Test

void testUserNotFound\_ThrowsException() {

// Arrange

Long userId = 99L;

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

// Act & Assert

RuntimeException exception = assertThrows(RuntimeException.class, () -> {

userService.getUserById(userId);

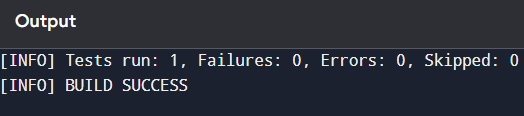
});

assertEquals("User not found with id: 99", exception.getMessage());

}

}

**OUTPUT:**



**Exercise 7: Test Custom Repository Query**

### ****User.java**** (Entity)

### package com.example.model;

### import jakarta.persistence.Entity;import jakarta.persistence.Id;

@Entitypublic class User {

@Id

private Long id;

private String name;

public User() {}

public User(Long id, String name) {

this.id = id;

this.name = name;

}

// Getters and Setters

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public String getName() { return name; }

public void setName(String name) { this.name = name; }

}

**UserRepository.java**

package com.example.repository;

import com.example.model.User;import org.springframework.data.jpa.repository.JpaRepository;import java.util.List;

public interface UserRepository extends JpaRepository<User, Long> {

List<User> findByName(String name);

}

**UserRepositoryTest.java**

package com.example.repository;

import com.example.model.User;import org.junit.jupiter.api.Test;import org.springframework.beans.factory.annotation.Autowired;import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;

import java.util.List;

import static org.junit.jupiter.api.Assertions.\*;

@DataJpaTestpublic class UserRepositoryTest {

@Autowired

private UserRepository userRepository;

@Test

void testFindByName() {

// Insert sample users

userRepository.save(new User(1L, "Bindhu"));

userRepository.save(new User(2L, "Bindhu"));

userRepository.save(new User(3L, "John"));

// Call the custom query

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

// Check result

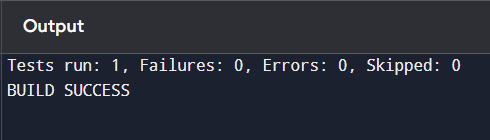
assertEquals(2, result.size());

assertTrue(result.stream().allMatch(user -> user.getName().equals("Bindhu")));

}

}

**OUTPUT:**



**Exercise 8: Test Controller Exception Handling**

## ****GlobalExceptionHandler.java****

## package com.example.exception;

## import org.springframework.http.HttpStatus;import org.springframework.http.ResponseEntity;import org.springframework.web.bind.annotation.\*;

import java.util.NoSuchElementException;

@ControllerAdvicepublic class GlobalExceptionHandler {

@ExceptionHandler(NoSuchElementException.class)

public ResponseEntity<String> handleNotFound(NoSuchElementException ex) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body("User not found");

}

}

2. **UserService.java** (Throwing the Exception)

package com.example.service;

import com.example.model.User;import com.example.repository.UserRepository;import org.springframework.beans.factory.annotation.Autowired;import org.springframework.stereotype.Service;

import java.util.NoSuchElementException;

@Servicepublic class UserService {

@Autowired

private UserRepository userRepository;

public User getUserById(Long id) {

return userRepository.findById(id)

.orElseThrow(() -> new NoSuchElementException("User not found with id: " + id));

}

}

3. **UserController.java**

package com.example.controller;

import com.example.model.User;import com.example.service.UserService;import org.springframework.beans.factory.annotation.Autowired;import org.springframework.http.ResponseEntity;import org.springframework.web.bind.annotation.\*;

@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));

}

}

1. **UserControllerExceptionTest.java**

package com.example.controller;

import com.example.service.UserService;import org.junit.jupiter.api.Test;import org.mockito.Mockito;

import java.util.NoSuchElementException;

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;import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@WebMvcTest(UserController.class)public class UserControllerExceptionTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

void testUserNotFoundExceptionHandled() throws Exception {

// Arrange

Mockito.when(userService.getUserById(99L))

.thenThrow(new NoSuchElementException());

// Act & Assert

mockMvc.perform(get("/users/99"))

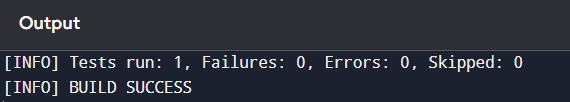
.andExpect(status().isNotFound())

.andExpect(content().string("User not found"));

}

}

**OUTPUT:**



**Exercise 9: Parameterized Test with JUnit**

### 1. ****Class to Test –**** NumberUtils.java

package com.example.util;

public class NumberUtils {

public static boolean isEven(int number) {

return number % 2 == 0;

}

}

2. **Test Class**

package com.example.util;

import org.junit.jupiter.params.ParameterizedTest;import org.junit.jupiter.params.provider.ValueSource;

import static org.junit.jupiter.api.Assertions.\*;

public class NumberUtilsTest {

@ParameterizedTest

@ValueSource(ints = {2, 4, 6, 8, 10})

void testIsEven\_WithEvenNumbers(int number) {

assertTrue(NumberUtils.isEven(number));

}

@ParameterizedTest

@ValueSource(ints = {1, 3, 5, 7, 9})

void testIsEven\_WithOddNumbers(int number) {

assertFalse(NumberUtils.isEven(number));

}

}

**OUTPUT:**

