**Spring Testing Exercises**

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

package com.example.junitdemo;

import org.springframework.stereotype.Service;

*@Service*

public class CalculatorService {

public int add(int a, int b) {

return a + b;

}

}

package com.example.junitdemo;

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

import org.junit.jupiter.api.Test;

public class CalculatorServiceTest {

private final CalculatorService calculatorService = new CalculatorService();

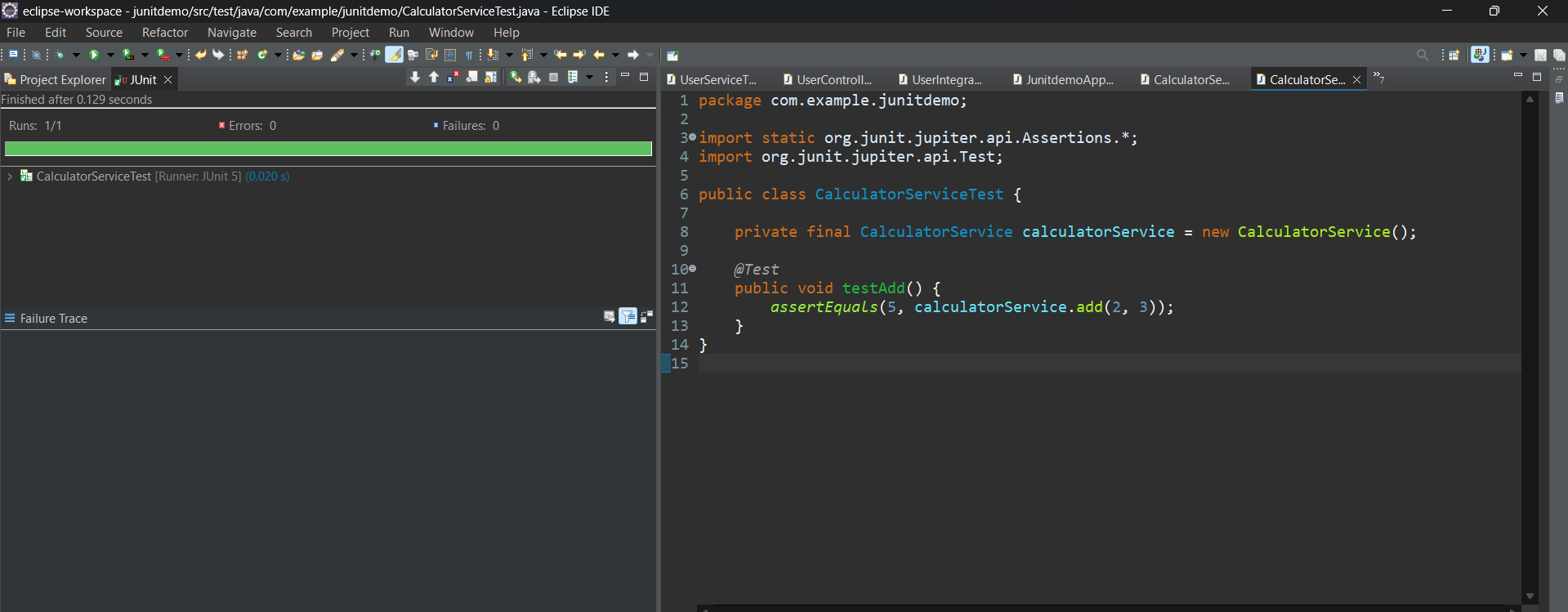
*@Test*

public void testAdd() {

*assertEquals*(5, calculatorService.add(2, 3));

}

}



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

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

import static org.mockito.Mockito.\*;

import java.util.Optional;

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

public class UserServiceTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

public UserServiceTest() {

MockitoAnnotations.openMocks(this);

}

@Test

public void testGetUserById() {

User user = new User(1L, "Alice");

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

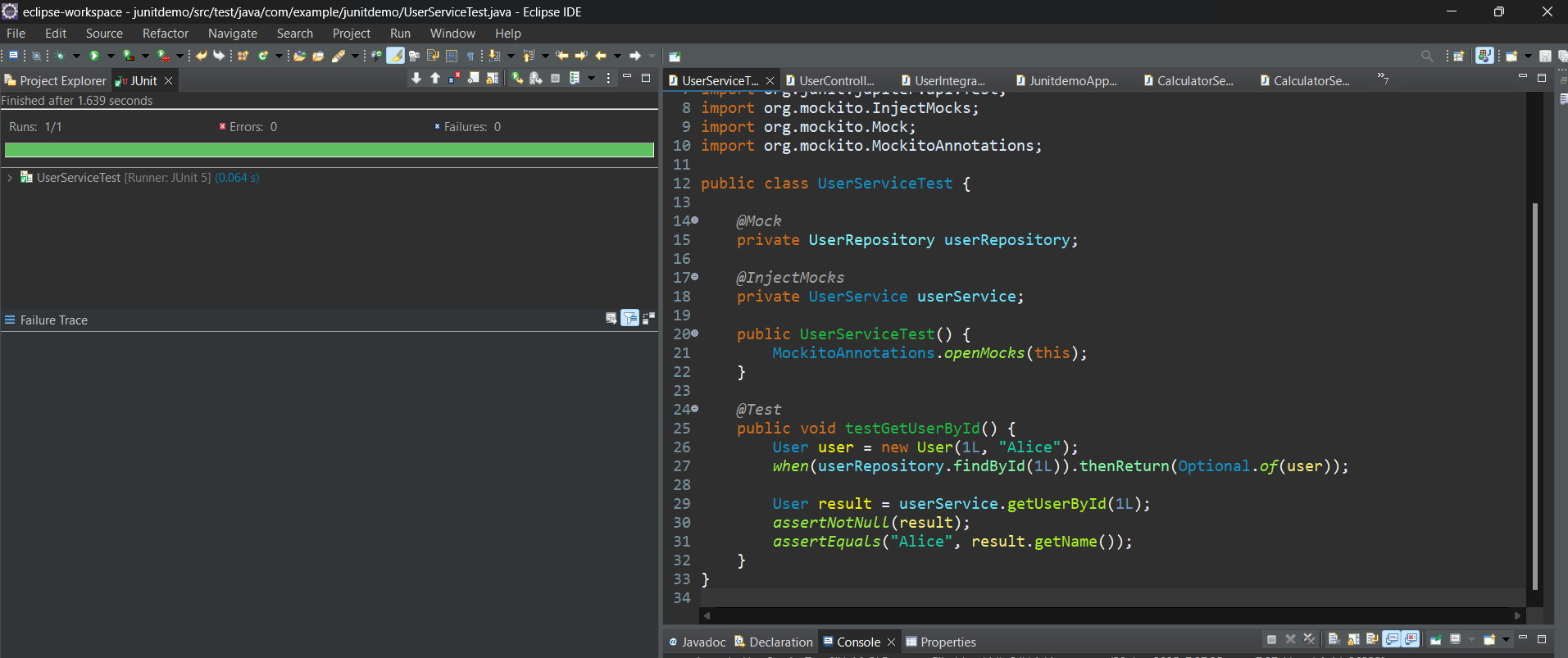
User result = userService.getUserById(1L);

assertNotNull(result);

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

}

}



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

package com.example.junitdemo;

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

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

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.\*;

*@WebMvcTest*(UserController.class)

public class UserControllerTest {

*@Autowired*

private MockMvc mockMvc;

*@MockBean*

private UserService userService;

*@Test*

public void testGetUser() throws Exception {

User user = new User(1L, "Bob");

*when*(userService.getUserById(1L)).thenReturn(user);

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

.andExpect(*status*().isOk())

.andExpect(*jsonPath*("$.name").value("Bob"));

}

}

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

package com.example.junitdemo;

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

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

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;

import org.springframework.test.web.servlet.MockMvc;

*@SpringBootTest*

*@AutoConfigureMockMvc*

public class UserIntegrationTest {

*@Autowired*

private MockMvc mockMvc;

*@Test*

public void testFullFlow() throws Exception {

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

.andExpect(*status*().isOk())

.andExpect(*jsonPath*("$.name").exists());

}

}

**Exercise 5: Test Controller POST Endpoint**

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

import static org.mockito.Mockito.\*;

import java.util.Optional;

import org.junit.jupiter.api.Test;

public class MissingUserTest {

@Test

public void testUserNotFound() {

UserRepository mockRepo = mock(UserRepository.class);

when(mockRepo.findById(99L)).thenReturn(Optional.empty());

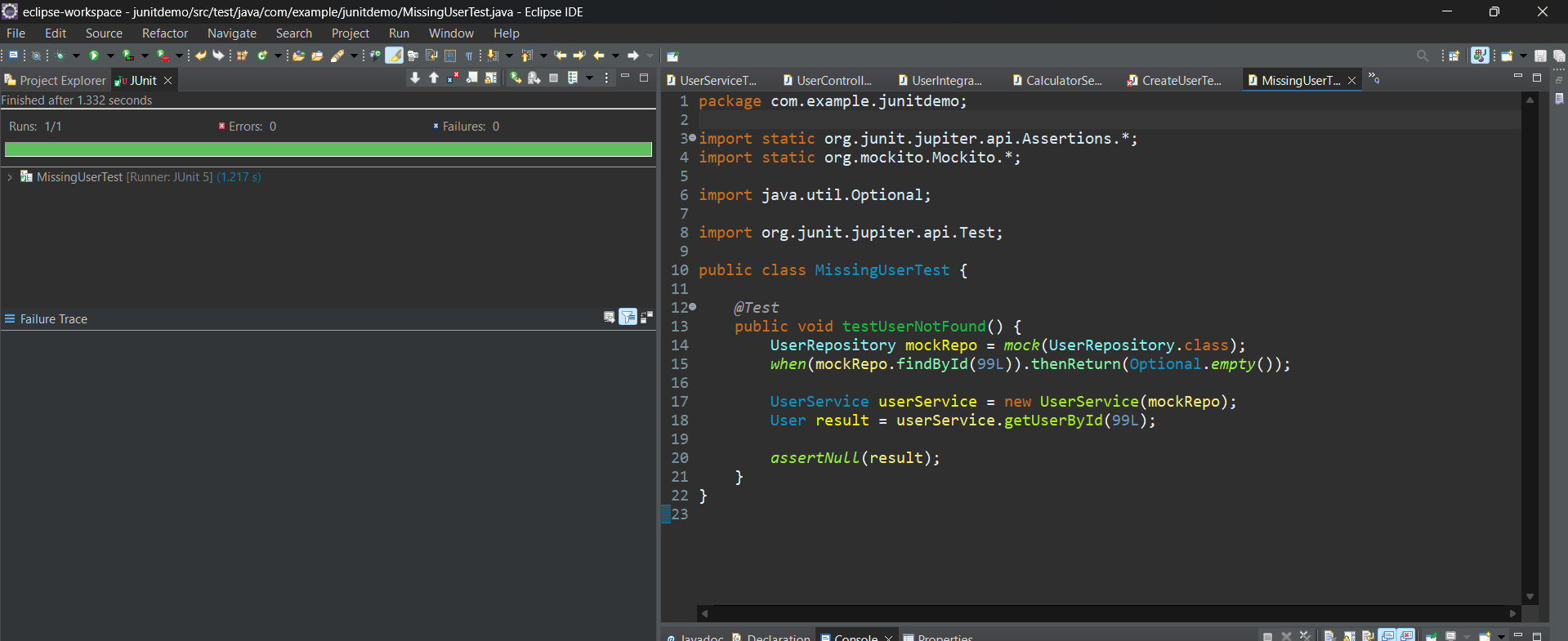
UserService userService = new UserService(mockRepo);

User result = userService.getUserById(99L);

assertNull(result);

}

}



**Exercise 7: Test Custom Repository Query**

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

import static org.mockito.Mockito.\*;

import java.util.List;

import org.junit.jupiter.api.Test;

public class CustomQueryTest {

@Test

public void testFindByName() {

UserRepository mockRepo = mock(UserRepository.class);

when(mockRepo.findByName("Tom")).thenReturn(List.of(new User(1L, "Tom")));

List<User> users = mockRepo.findByName("Tom");

assertEquals(1, users.size());

assertEquals("Tom", users.get(0).getName());

}

}

@ControllerAdvice

public class GlobalExceptionHandler {

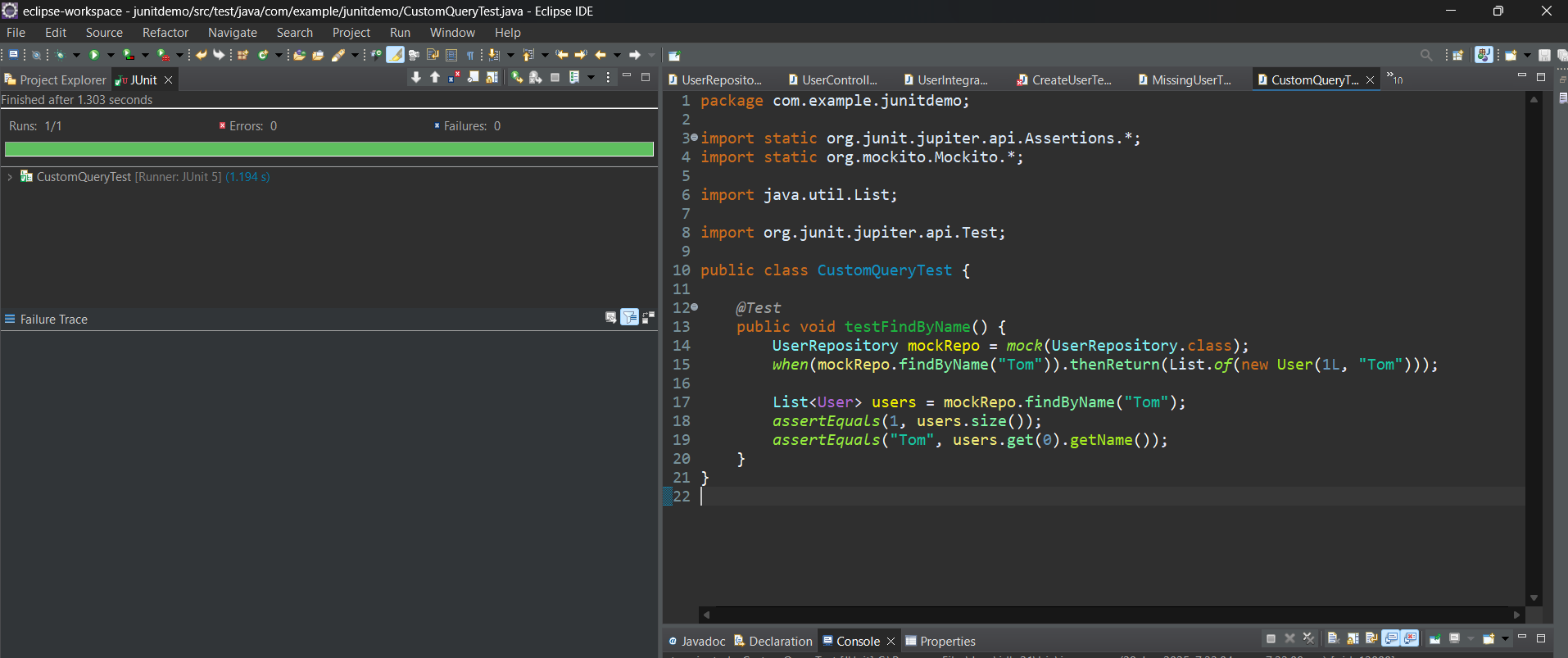
@ExceptionHandler(NoSuchElementException.class)

public ResponseEntity<String> handleNotFound(NoSuchElementException ex) {

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

}

}



**Exercise 8: Test Controller Exception Handling**

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

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

import java.util.NoSuchElementException;

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.\*;

@WebMvcTest(UserController.class)

public class ExceptionHandlerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

public void testUserNotFoundException() throws Exception {

when(userService.getUserById(100L)).thenThrow(new NoSuchElementException());

mockMvc.perform(get("/users/100"))

.andExpect(status().isNotFound())

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

}

}

**Exercise 9: Parameterized Test with JUnit**

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

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.CsvSource;

public class CalculatorParameterizedTest {

private final CalculatorService calculatorService = new CalculatorService();

@ParameterizedTest

@CsvSource({

"1, 2, 3",

"5, 5, 10",

"10, -5, 5"

})

public void testAdd(int a, int b, int expected) {

assertEquals(expected, calculatorService.add(a, b));

}

}

