**1.Write a program to demonstrate standard tests.**

Source code:

import org.junit.jupiter.api.Test;

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

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

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

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

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

import org.springframework.test.web.servlet.result.MockMvcResultMatchers;

@SpringBootTest

@AutoConfigureMockMvc

public class StandardTests {

@Autowired

private MockMvc mockMvc;

@Test

public void testGetRequest() throws Exception {

mockMvc.perform(MockMvcRequestBuilders.get("/api/example"))

.andExpect(MockMvcResultMatchers.status().isOk());

}

@Test

public void testPostRequest() throws Exception {

mockMvc.perform(MockMvcRequestBuilders.post("/api/example")

.content("Some data"))

.andExpect(MockMvcResultMatchers.status().isCreated());

}

}

**2. Write a program to demonstrate assertions?**

import org.junit.jupiter.api.Test;

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

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

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

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

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

import org.springframework.test.web.servlet.result.MockMvcResultMatchers;

import org.springframework.util.Assert;

@SpringBootTest

@AutoConfigureMockMvc

public class AssertionTest {

@Autowired

private MockMvc mockMvc;

@Test

public void testAssertEquals() {

int expected = 5;

int actual = 2 + 3;

Assert.assertEquals(expected, actual);

}

@Test

public void testAssertTrue() {

boolean condition = true;

Assert.isTrue(condition, "The condition is false");

}

@Test

public void testAssertNotNull() {

String value = "Hello";

Assert.notNull(value, "The value is null");

}

@Test

public void testGetRequest() throws Exception {

mockMvc.perform(MockMvcRequestBuilders.get("/api/example"))

.andExpect(MockMvcResultMatchers.status().isOk());

}

}

**3. Write a program to demonstrate conditional test executions?**

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.condition.EnabledIfSystemProperty;

import org.junit.jupiter.api.condition.EnabledOnOs;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class ConditionalTestExecution {

@Test

@EnabledOnOs(org.junit.jupiter.api.condition.OS.WINDOWS)

void onlyOnWindows() {

// Test code specific to Windows OS

assertEquals(2, 1 + 1);

}

@Test

@EnabledIfSystemProperty(named = "environment", matches = "dev")

void onlyOnDevEnvironment() {

// Test code specific to the "dev" environment

assertEquals("dev", System.getProperty("environment"));

}

@Test

@EnabledIfSystemProperty(named = "env", matches = "prod", disabledReason = "Disabled for prod environment")

void disabledOnProdEnvironment() {

// This test will not be executed in the "prod" environment

}

}

**4. Write a program to demonstrate nested and repeated tests?**

Source code:

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.Nested;

import org.junit.jupiter.api.RepeatedTest;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class NestedAndRepeatedTests {

@Nested

@DisplayName("Math operations")

class MathOperations {

@Test

@DisplayName("Addition test")

void additionTest() {

assertEquals(5, 2 + 3);

}

@Test

@DisplayName("Subtraction test")

void subtractionTest() {

assertEquals(3, 5 - 2);

}

@Nested

@DisplayName("Multiplication tests")

class MultiplicationTests {

@Test

@DisplayName("Multiplication with zero test")

void multiplicationWithZeroTest() {

assertEquals(0, 5 \* 0);

}

@RepeatedTest(3)

@DisplayName("Random multiplication test")

void randomMultiplicationTest() {

int operand1 = (int) (Math.random() \* 10);

int operand2 = (int) (Math.random() \* 10);

int result = operand1 \* operand2;

assertEquals(result, operand1 \* operand2);

}

}

}

}

**5. Write a program to demonstrate dynamic tests?**

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.DynamicTest;

import org.junit.jupiter.api.TestFactory;

import java.util.Arrays;

import java.util.Collection;

import java.util.stream.Stream;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class DynamicTests {

@TestFactory

@DisplayName("Dynamic addition tests")

Collection<DynamicTest> dynamicAdditionTests() {

return Arrays.asList(

DynamicTest.dynamicTest("Test 1", () -> {

int result = 2 + 3;

assertEquals(5, result);

}),

DynamicTest.dynamicTest("Test 2", () -> {

int result = 5 + 7;

assertEquals(12, result);

}),

DynamicTest.dynamicTest("Test 3", () -> {

int result = 0 + 0;

assertEquals(0, result);

})

);

}

@TestFactory

@DisplayName("Dynamic multiplication tests")

Stream<DynamicTest> dynamicMultiplicationTests() {

return Stream.of(2, 3, 4)

.map(operand -> DynamicTest.dynamicTest("Test " + operand, () -> {

int result = operand \* 5;

assertEquals(operand \* 5, result);

}));

}

}

**6. Write a program to demonstrate a dependency injection?**

Scode:

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

import org.springframework.stereotype.Component;

@Component

class EmailService {

public void sendEmail(String recipient, String message) {

System.out.println("Sending email to " + recipient + ": " + message);

}

}

@Component

class NotificationService {

private final EmailService emailService;

@Autowired

public NotificationService(EmailService emailService) {

this.emailService = emailService;

}

public void sendNotification(String recipient, String message) {

emailService.sendEmail(recipient, message);

}

}

@Component

public class MyApp {

private final NotificationService notificationService;

@Autowired

public MyApp(NotificationService notificationService) {

this.notificationService = notificationService;

}

public void run() {

String recipient = "example@example.com";

String message = "Hello, world!";

notificationService.sendNotification(recipient, message);

}

public static void main(String[] args) {

MyApp app = new MyApp(); // Instantiate the main application

app.run();

}

}

**7. You are given a project to demonstrate RESTful with Spring Boot?**

Scode:

@RestController

@RequestMapping("/api/books")

public class BookController {

private List<Book> books = new ArrayList<>();

@GetMapping

public List<Book> getAllBooks() {

return books;

}

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

// Find book by id and return it

}

@PostMapping

public ResponseEntity<Book> createBook(@RequestBody Book book) {

// Create a new book

// Set the book's id (e.g., using an auto-generated value)

// Add the book to the list of books

// Return the created book with a HTTP 201 Created status

}

@PutMapping("/{id}")

public ResponseEntity<Book> updateBook(@PathVariable Long id, @RequestBody Book updatedBook) {

// Find the book by id and update its attributes with the values from updatedBook

// Return the updated book with a HTTP 200 OK status

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteBook(@PathVariable Long id) {

// Find the book by id and remove it from the list of books

// Return a HTTP 204 No Content response

}

}