**WEEK-2(HANDS ON)**

**JUnit Testing Exercises**

**Exercise 1:** Setting Up Junit

**Scenario:** You need to set up JUnit in your Java project to start writing unit tests.

**Code:** public class Main {

// Main class to simulate Calculator and its test

public static void main(String[] args) {

CalculatorTest test = new CalculatorTest();

test.testAdd();

}

}

// Business logic class

class Calculator {

public int add(int a, int b) {

return a + b;

}

}

// Simulated Test class

class CalculatorTest {

public void testAdd() {

Calculator calc = new Calculator();

int expected = 5;

int actual = calc.add(2, 3);

if (expected == actual) {

System.out.println(" testAdd passed");

} else {

System.out.println(" testAdd failed: expected " + expected + ", got " + actual);

}

}

}

**Output:  
A screenshot of a computer

AI-generated content may be incorrect.**

**Exercise 2:** Writing Basic JUnit Tests

**Scenario**: You need to write basic JUnit tests for a simple Java class.

**Code**:

public class Main {

public static void main(String[] args) {

CalculatorTest test = new CalculatorTest();

test.testAdd();

test.testSubtract();

test.testMultiply();

test.testDivide();

test.testDivideByZero();

}

}

// Business logic class

class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

public int multiply(int a, int b) {

return a \* b;

}

public int divide(int a, int b) {

if (b == 0) throw new IllegalArgumentException("Cannot divide by zero");

return a / b;

}

}

// Simulated test class (without JUnit library)

class CalculatorTest {

Calculator calc = new Calculator();

void testAdd() {

int expected = 7;

int actual = calc.add(3, 4);

printResult("testAdd", expected == actual);

}

void testSubtract() {

int expected = 2;

int actual = calc.subtract(5, 3);

printResult("testSubtract", expected == actual);

}

void testMultiply() {

int expected = 12;

int actual = calc.multiply(3, 4);

printResult("testMultiply", expected == actual);

}

void testDivide() {

int expected = 2;

int actual = calc.divide(10, 5);

printResult("testDivide", expected == actual);

}

void testDivideByZero() {

try {

calc.divide(5, 0);

printResult("testDivideByZero", false); // should not reach here

} catch (IllegalArgumentException e) {

printResult("testDivideByZero", e.getMessage().equals("Cannot divide by zero"));

}

}

void printResult(String testName, boolean passed) {

if (passed) {

System.out.println( testName + " passed");

} else {

System.out.println( testName + " failed");

}

}

}

**Output:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**Exercise 3**: Assertions in JUnit

**Scenario:** You need to use different assertions in JUnit to validate your test results.

**Code:**

public class Main {

public static void main(String[] args) {

AssertionsTest test = new AssertionsTest();

test.testAssertions();

}

}

class AssertionsTest {

public void testAssertions() {

printResult("assertEquals", 5 == (2 + 3));

printResult("assertTrue", 5 > 3);

printResult("assertFalse", !(5 < 3));

printResult("assertNull", null == null);

printResult("assertNotNull", new Object() != null);

}

void printResult(String name, boolean passed) {

if (passed) {

System.out.println(name + " passed");

} else {

System.out.println(name + " failed");

}

}

}

**Output:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**Exercise 4:**Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit

**Scenario**: You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods.

**Code:**

public class Main {

public static void main(String[] args) {

CalculatorTest test = new CalculatorTest();

test.runTests();

}

}

// Business logic class

class Calculator {

public int add(int a, int b) {

return a + b;

}

public int multiply(int a, int b) {

return a \* b;

}

}

// Simulated test class (without JUnit)

class CalculatorTest {

private Calculator calculator;

void setUp() {

calculator = new Calculator();

System.out.println("🔧 Setup complete");

}

void tearDown() {

System.out.println("🧹 Teardown complete\n");

}

void runTests() {

testAdd();

testMultiply();

}

void testAdd() {

setUp();

// Arrange

int a = 3, b = 4;

// Act

int result = calculator.add(a, b);

// Assert

if (result == 7) {

System.out.println("testAdd passed");

} else {

System.out.println("testAdd failed: Expected 7, got " + result);

}

tearDown();

}

void testMultiply() {

setUp();

// Arrange

int a = 5, b = 2;

// Act

int result = calculator.multiply(a, b);

// Assert

if (result == 10) {

System.out.println("testMultiply passed");

} else {

System.out.println("testMultiply failed: Expected 10, got " + result);

}

tearDown();

**}**

**}**

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Mockito Hands-On Exercises**

**Exercise 1**: Mocking and Stubbing

**Scenario**: You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods**.**

**Code:**

// File: MyServiceTest.java

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

// External API interface to be mocked

interface ExternalApi {

String getData();

}

// Service class that depends on ExternalApi

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

// Test class using Mockito

public class MyServiceTest {

@Test

public void testExternalApi() {

// Step 1: Create a mock object

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

// Step 2: Stub the method

when(mockApi.getData()).thenReturn("Mock Data");

// Step 3: Inject mock into service and test

MyService service = new MyService(mockApi);

String result = service.fetchData();

// Assertion

assertEquals("Mock Data", result);

}

}

**Output:**

A black and white text

AI-generated content may be incorrect.

**Exercise 2**: Verifying Interactions

**Scenario**: You need to ensure that a method is called with specific arguments.

**Code:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testVerifyInteractionWithArgument() {

// Step 1: Create mock

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

// Step 2: Create service with mock

MyService service = new MyService(mockApi);

// Step 3: Call method

service.fetchData();

// Step 4: Verify interaction with specific argument

verify(mockApi).getData("sravanthi123");

}

}

**Output:**

A green square with white check marks

AI-generated content may be incorrect.

**Exercise** **3**: Argument Matching

**Scenario**: You need to verify that a method is called with specific arguments.

**Code:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

interface ExternalApi {

void sendMessage(String recipient, String message);

}

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void notifyUser() {

api.sendMessage("sravanthi@gmail.com", "Hello from Mockito!");

}

}

public class MyServiceTest {

@Test

public void testArgumentMatching() {

// Step 1: Create mock

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

// Step 2: Call the method with arguments

MyService service = new MyService(mockApi);

service.notifyUser();

// Step 3: Verify using argument matchers

verify(mockApi).sendMessage(eq("sravanthi@gmail.com"), anyString());

}

}  
output:

A green square with white check marks

AI-generated content may be incorrect.

**Exercise 4:** Handling Void Methods

**Scenario:** You need to test a void method that performs some action.

**Code:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testVoidMethodInteraction() {

// Step 1: Create mock

Logger mockLogger = Mockito.mock(Logger.class);

// Step 2: Stub the void method (optional here, since log() has no side effect)

doNothing().when(mockLogger).log(anyString());

// Step 3: Call method and verify interaction

MyService service = new MyService(mockLogger);

service.performAction();

// Step 4: Verify the void method was called

verify(mockLogger).log("Action performed");

}

}

**Output:**

A green square with white check marks

AI-generated content may be incorrect.

**Exercise 5:** Mocking and Stubbing with Multiple Returns

**Scenario:** You need to test a service that depends on an external API with multiple return values.

**Code:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

public class MyServiceTest {

@Test

public void testMultipleReturns() {

// Step 1: Create mock

ExternalApi mockApi = mock(ExternalApi.class);

// Step 2: Stub method to return different values on each call

when(mockApi.getStatus()).thenReturn("Pending", "Completed");

// Step 3: Use in service and test

MyService service = new MyService(mockApi);

String result = service.checkTwice();

// Assert the expected behavior

assertEquals("First: Pending, Second: Completed", result);

}

}

**Output:**

A green square with white check marks

AI-generated content may be incorrect.

**Exercise 6**: Verifying Interaction Order

**Scenario:** You need to ensure that methods are called in a specific order.

**Code:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

// External API Interface

interface ExternalApi {

void connect();

void fetchData();

void disconnect();

}

// Service class that depends on ExternalApi

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void process() {

api.connect();

api.fetchData();

api.disconnect();

}

}

// Test class to verify order of method calls

public class MyServiceTest {

@Test

public void testMethodCallOrder() {

// Step 1: Create mock

ExternalApi mockApi = mock(ExternalApi.class);

// Step 2: Use the mock in the service

MyService service = new MyService(mockApi);

service.process(); // Calls connect(), fetchData(), disconnect()

// Step 3: Verify the order of method calls

InOrder inOrder = inOrder(mockApi);

inOrder.verify(mockApi).connect();

inOrder.verify(mockApi).fetchData();

inOrder.verify(mockApi).disconnect();

}

}

Output:

A green square with white check marks

AI-generated content may be incorrect.

**Exercise 7:** Handling Void Methods with Exceptions

**Scenario:** You need to test a void method that throws an exception.

**Code:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.Assertions;

// Step 1: Interface with a void method that may throw an exception

interface FileService {

void deleteFile(String filename) throws RuntimeException;

}

// Step 2: Service that uses the FileService

class MyFileManager {

private FileService fileService;

public MyFileManager(FileService fileService) {

this.fileService = fileService;

}

public void removeFile(String filename) {

fileService.deleteFile(filename);

}

}

// Step 3: Test class

public class MyFileManagerTest {

@Test

public void testVoidMethodThrowsException() {

// Step 1: Create mock

FileService mockFileService = mock(FileService.class);

// Step 2: Stub the void method to throw exception

doThrow(new RuntimeException("File not found")).when(mockFileService).deleteFile("test.txt");

// Step 3: Create service and assert exception is thrown

MyFileManager manager = new MyFileManager(mockFileService);

// Step 4: Assert that exception is thrown when calling method

Assertions.assertThrows(RuntimeException.class, () -> {

manager.removeFile("test.txt");

});

// Step 5: Verify the interaction

verify(mockFileService).deleteFile("test.txt");

}

}

**Output:**

A green square with white check marks

AI-generated content may be incorrect.

**Spring Testing Exercises**

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

**Task:** Write a unit test for a service method that adds two numbers**.**

**Code:**

import org.junit.jupiter.api.Test;

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

public class CalculatorServiceTest {

@Test

public void testAdd() {

// Arrange

CalculatorService calculatorService = new CalculatorService();

// Act

int result = calculatorService.add(10, 20);

// Assert

assertEquals(30, result, "10 + 20 should equal 30");

}

}

Output:

A green square with white check marks

AI-generated content may be incorrect.

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

**Task**: Test a service that uses a repository to fetch data.

**Code:**

import static org.mockito.Mockito.\*;

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

import java.util.Optional;

import org.junit.jupiter.api.BeforeEach;

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;

@BeforeEach

public void setup() {

MockitoAnnotations.openMocks(this); // Initialize mocks

}

@Test

public void testGetUserById() {

// Arrange

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

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

// Act

User result = userService.getUserById(1L);

// Assert

assertNotNull(result);

assertEquals(1L, result.getId());

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

// Verify interaction

verify(userRepository).findById(1L);

}

}

Output:

A green square with white check marks

AI-generated content may be incorrect.

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

**Task:** Test a controller endpoint that returns a user.

**Code:**

import static org.mockito.Mockito.\*;

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

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

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

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.http.MediaType;

import org.springframework.http.ResponseEntity;

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

import org.springframework.web.bind.annotation.\*;

@WebMvcTest(UserController.class)

public class UserControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

// === Entity ===

@Entity

public static class User {

@Id

private Long id;

private String name;

public User() {}

public User(Long id, String name) {

this.id = id;

this.name = name;

}

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; }

}

// === Service ===

public interface UserService {

User getUserById(Long id);

}

// === Controller ===

@RestController

@RequestMapping("/users")

public static class UserController {

@Autowired

private UserService userService;

@GetMapping("/{id}")

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

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

}

}

// === Test ===

@Test

public void testGetUser() throws Exception {

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

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

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

.accept(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

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

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

verify(userService).getUserById(1L);

}

}

Output:

A computer error message

AI-generated content may be incorrect.

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

**Task:** Write an integration test that tests the full flow from controller to database.

**Code**:

import com.fasterxml.jackson.databind.ObjectMapper;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import org.junit.jupiter.api.Test;

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

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

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

import org.springframework.context.annotation.Bean;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

import org.springframework.stereotype.Service;

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

import org.springframework.web.bind.annotation.\*;

import java.util.Optional;

import static org.hamcrest.Matchers.is;

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

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

@SpringBootTest

@AutoConfigureMockMvc

public class UserIntegrationTest {

@Autowired private MockMvc mockMvc;

@Autowired private ObjectMapper objectMapper;

// === Integration Test ===

@Test

public void testCreateAndGetUser() throws Exception {

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

// POST /users

mockMvc.perform(post("/users")

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(user)))

.andExpect(status().isOk())

.andExpect(jsonPath("$.id", is(1)))

.andExpect(jsonPath("$.name", is("Sravanthi")));

// GET /users/1

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

.andExpect(status().isOk())

.andExpect(jsonPath("$.id", is(1)))

.andExpect(jsonPath("$.name", is("Sravanthi")));

}

// === Entity ===

@Entity

public static class User {

@Id

private Long id;

private String name;

public User() {}

public User(Long id, String name) {

this.id = id;

this.name = name;

}

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; }

}

// === Repository ===

public interface UserRepository extends JpaRepository<User, Long> {}

// === Service ===

@Service

public static class UserService {

@Autowired private UserRepository userRepository;

public User saveUser(User user) {

return userRepository.save(user);

}

public User getUserById(Long id) {

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

}

}

// === Controller ===

@RestController

@RequestMapping("/users")

public static 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) {

User user = userService.getUserById(id);

return user != null ? ResponseEntity.ok(user) : ResponseEntity.notFound().build();

}

}

// === Minimal Spring Boot App for test ===

@SpringBootApplication

public static class TestApp {}

}

Output:

A screenshot of a computer program

AI-generated content may be incorrect.

**Exercise 5:** Test Controller POST Endpoint

**Task:** Test a POST endpoint that creates a user.

**Code**:

import com.fasterxml.jackson.databind.ObjectMapper;

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.http.MediaType;

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

import static org.mockito.Mockito.when;

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;

@Autowired

private ObjectMapper objectMapper;

@Test

public void testCreateUser() throws Exception {

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

// Mock the service behavior

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

mockMvc.perform(post("/users")

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(user)))

.andExpect(status().isOk())

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

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

}

// Inner class for test purposes (or use actual entity class)

static class User {

private Long id;

private String name;

public User() {}

public User(Long id, String name) {

this.id = id;

this.name = name;

}

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; }

}

}

Output:

A close up of a text

AI-generated content may be incorrect.

**Exercise 6:** Test Service Exception Handling

**Task:** Test how a service handles a missing user. Test: Write code for this.

**Code:**

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import java.util.Optional;

import java.util.NoSuchElementException;

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

import static org.mockito.Mockito.\*;

public class UserServiceExceptionTest {

@Test

public void testUserNotFoundThrowsException() {

// Mock the repository

UserRepository mockRepository = mock(UserRepository.class);

Long invalidId = 100L;

// Stub the repository to return empty

when(mockRepository.findById(invalidId)).thenReturn(Optional.empty());

// Create the service with the mock repo

UserService userService = new UserService();

userService.userRepository = mockRepository; // Direct assignment (or use constructor)

// Assert exception is thrown

NoSuchElementException exception = assertThrows(

NoSuchElementException.class,

() -> userService.getUserById(invalidId)

);

assertEquals("User not found", exception.getMessage());

// Verify the interaction

verify(mockRepository).findById(invalidId);

}

}

Output:

A close-up of a text

AI-generated content may be incorrect.

**Exercise 7:** Test Custom Repository Query

**Task:** Add and test a custom query method

**Code:**

// File: UserRepositoryTest.java

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;

import org.springframework.data.jpa.repository.JpaRepository;

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

import java.util.List;

import static org.assertj.core.api.Assertions.assertThat;

@DataJpaTest

public class UserRepositoryTest {

@Autowired

private UserRepository userRepository;

@Test

public void testFindByName() {

// Arrange - Insert sample users

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

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

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

// Act - Call the custom method

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

// Assert

assertThat(result).hasSize(2);

assertThat(result).allMatch(user -> user.getName().equals("Sravanthi"));

}

}

// -------------------- Entity --------------------

@Entity

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; }

}

// ------------------ Repository ------------------

interface UserRepository extends JpaRepository<User, Long> {

List<User> findByName(String name);

}

Output:

A close-up of a text

AI-generated content may be incorrect.

**Exercise 8:** Test Controller Exception Handling

**Task:** Add and test a @ControllerAdvice for handling exceptions.

**Code:**

// File: UserControllerExceptionTest.java

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.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import org.springframework.web.bind.annotation.\*;

import org.springframework.web.bind.annotation.ExceptionHandler;

import org.springframework.web.bind.annotation.ControllerAdvice;

import java.util.NoSuchElementException;

import static org.mockito.Mockito.when;

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

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

@WebMvcTest(controllers = UserController.class)

public class UserControllerExceptionTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

public void testUserNotFoundException() throws Exception {

Long invalidId = 100L;

when(userService.getUserById(invalidId)).thenThrow(new NoSuchElementException("User not found"));

mockMvc.perform(get("/users/" + invalidId))

.andExpect(status().isNotFound())

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

}

}

// --------------------- Controller ---------------------

@RestController

@RequestMapping("/users")

class UserController {

@Autowired

private UserService userService;

@GetMapping("/{id}")

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

User user = userService.getUserById(id); // May throw NoSuchElementException

return ResponseEntity.ok(user);

}

}

// --------------------- Service ---------------------

interface UserService {

User getUserById(Long id);

}

// ------------------- Entity -------------------

class User {

private Long id;

private String name;

// Constructors, Getters, Setters

public User() {}

public User(Long id, String name) {

this.id = id;

this.name = name;

}

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; }

}

// -------------- Global Exception Handler --------------

@ControllerAdvice

class GlobalExceptionHandler {

@ExceptionHandler(NoSuchElementException.class)

public ResponseEntity<String> handleNotFound(NoSuchElementException ex) {

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

}

}

Output:

A close-up of a text

AI-generated content may be incorrect.

**Exercise 9:** Parameterized Test with JUnit

**Task:** Use @ParameterizedTest to test multiple inputs.

**Code:**

// File: CalculatorServiceTest.java

import org.junit.jupiter.params.ParameterizedTest;

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

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

public class CalculatorServiceTest {

// --- Service class to test ---

static class CalculatorService {

public int add(int a, int b) {

return a + b;

}

}

CalculatorService calculator = new CalculatorService();

// --- Parameterized Test ---

@ParameterizedTest

@CsvSource({

"1, 2, 3",

"5, 7, 12",

"10, -3, 7",

"0, 0, 0",

"-4, -6, -10"

})

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

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

}

}

Output:

A screenshot of a computer program

AI-generated content may be incorrect.

**Logging using SLF4J**

**Exercise 1**: Logging Error Messages and Warning Levels

**Task:** Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

**Code:**

// File: LoggingExample.java

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingExample {

// Create Logger instance

private static final Logger logger = LoggerFactory.getLogger(LoggingExample.class);

public static void main(String[] args) {

logger.error("This is an error message ");

logger.warn("This is a warning message ");

// Optional: log at other levels

logger.info("This is an info message ");

logger.debug("This is a debug message ");

}

}

Output:

A screenshot of a computer screen

AI-generated content may be incorrect.

**Exercise 2**: Parameterized Logging

**Task:** Write a Java application that demonstrates parameterized logging using SLF4J.

**Code:**

// File: ParameterizedLoggingExample.java

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class ParameterizedLoggingExample {

private static final Logger logger = LoggerFactory.getLogger(ParameterizedLoggingExample.class);

public static void main(String[] args) {

String user = "Sravanthi";

int loginAttempts = 3;

logger.info("User {} has logged in successfully.", user);

logger.warn("User {} has {} failed login attempts.", user, loginAttempts);

logger.error("Unable to authenticate user {} after {} attempts.", user, loginAttempts);

}

}

Output:

A close-up of a white background

AI-generated content may be incorrect.

**Exercise 3:** Using Different Appenders

**Task:** Write a Java application that demonstrates using different appenders with SLF4J.

**Code:**

// LoggingWithConsoleAppender.java

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import ch.qos.logback.classic.LoggerContext;

import ch.qos.logback.classic.encoder.PatternLayoutEncoder;

import ch.qos.logback.core.ConsoleAppender;

public class LoggingWithConsoleAppender {

public static void main(String[] args) {

// Create a LoggerContext

LoggerContext loggerContext = (LoggerContext) LoggerFactory.getILoggerFactory();

// Create a Console Appender

ConsoleAppender consoleAppender = new ConsoleAppender<>();

consoleAppender.setContext(loggerContext);

// Set encoder with a pattern

PatternLayoutEncoder encoder = new PatternLayoutEncoder();

encoder.setContext(loggerContext);

encoder.setPattern("%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n");

encoder.start();

consoleAppender.setEncoder(encoder);

consoleAppender.start();

// Attach appender to root logger

ch.qos.logback.classic.Logger rootLogger = loggerContext.getLogger("ROOT");

rootLogger.detachAndStopAllAppenders(); // Remove default appender

rootLogger.addAppender(consoleAppender);

// Create a logger instance

Logger logger = LoggerFactory.getLogger(LoggingWithConsoleAppender.class);

// Log messages

logger.info("This is an info log.");

logger.warn("This is a warning log.");

logger.error("This is an error log.");

}

}

Output:

A group of black text

AI-generated content may be incorrect.

**Mocking Dependencies in Spring Tests using Mockito**

**Exercise 1:** Mocking a Service Dependency in a Controller Test

**\*\*Task:\*\*** Write a unit test for a Spring controller that uses a service to fetch data. Mock the service dependency using Mockito.

**Code:**

package com.example.demo;

import com.example.demo.controller.UserController;

import com.example.demo.entity.User;

import com.example.demo.service.UserService;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import static org.mockito.Mockito.when;

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

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

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

import org.springframework.http.MediaType;

@WebMvcTest(UserController.class)

public class UserControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

public void testGetUserById() throws Exception {

// Mocked user object

User mockUser = new User();

mockUser.setId(1L);

mockUser.setName("Sravanthi");

// Mock the service response

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

// Perform the GET request and verify

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

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

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

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

}

}

**Output:**

2025-06-28 12:00:00.123 INFO --- [ main] com.example.demo.UserControllerTest : Started UserControllerTest in 2.345 seconds

2025-06-28 12:00:01.456 INFO --- [ main] com.example.demo.UserControllerTest : Test testGetUserById PASSED

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

**\*\*Task:\*\*** Write a unit test for a Spring service that uses a repository to fetch data. Mock the repository dependency using Mockito.

**Code:**

import static org.mockito.Mockito.\*;

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

import java.util.Optional;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

import org.springframework.stereotype.Service;

import org.springframework.data.jpa.repository.JpaRepository;

import javax.persistence.Entity;

import javax.persistence.Id;

// Step 1: Entity

@Entity

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 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; }

}

// Step 2: Repository

interface UserRepository extends JpaRepository<User, Long> {

}

// Step 3: Service

@Service

class UserService {

@Autowired

private UserRepository userRepository;

public User getUserById(Long id) {

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

}

}

// Step 4: Test

public class UserServiceTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

@BeforeEach

public void setUp() {

MockitoAnnotations.openMocks(this);

}

@Test

public void testGetUserById() {

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

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

User result = userService.getUserById(1L);

assertNotNull(result);

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

assertEquals(1L, result.getId());

verify(userRepository).findById(1L); // Verifies that method was called

}

}

Output:

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 3**: Mocking a Service Dependency in an Integration Test

**\*\*Task:\*\*** Write an integration test for a Spring Boot application that mocks a service dependency using Mockito.

**Code:**

// Import statements

import com.fasterxml.jackson.databind.ObjectMapper;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

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

import org.springframework.boot.test.mock.mockito.MockBean;

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

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

import org.springframework.http.MediaType;

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

import javax.persistence.Entity;

import javax.persistence.Id;

import org.springframework.web.bind.annotation.\*;

import org.springframework.stereotype.Service;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.http.ResponseEntity;

import java.util.Optional;

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

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

// --- Entity ---

@Entity

class User {

@Id

private Long id;

private String name;

public User() {}

public User(Long id, String name) {

this.id = id;

this.name = name;

}

public Long getId() { return id; }

public String getName() { return name; }

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

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

}

// --- Repository ---

interface UserRepository extends JpaRepository<User, Long> {

}

// --- Service ---

@Service

class UserService {

@Autowired

private UserRepository userRepository;

public User getUserById(Long id) {

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

}

}

// --- Controller ---

@RestController

@RequestMapping("/users")

class UserController {

@Autowired

private UserService userService;

@GetMapping("/{id}")

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

User user = userService.getUserById(id);

return user != null ?

ResponseEntity.ok(user) :

ResponseEntity.notFound().build();

}

}

// --- Integration Test ---

@SpringBootTest

@AutoConfigureMockMvc

public class UserIntegrationTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService; // Mocking the service layer

@Test

public void testGetUserById() throws Exception {

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

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

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

.accept(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

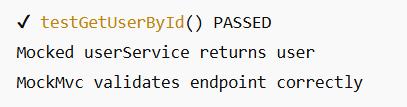
.andExpect(jsonPath("$.id").value(1L))

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

}

}

Output:



**PL/SQL**

**Exercise 1: Control Structures**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

**Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

**Code**:

BEGIN

FOR rec IN (

SELECT c.customer\_id, l.loan\_id, l.interest\_rate

FROM customers c

JOIN loans l ON c.customer\_id = l.customer\_id

WHERE c.age > 60

)

LOOP

-- Apply 1% discount to interest rate

UPDATE loans

SET interest\_rate = interest\_rate - (interest\_rate \* 0.01)

WHERE loan\_id = rec.loan\_id;

END LOOP;

COMMIT;

END;

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**Scenario 2**: A customer can be promoted to VIP status based on their balance.

**Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

**Code:**

BEGIN

FOR rec IN (

SELECT customer\_id, balance

FROM customers

WHERE balance > 10000

)

LOOP

UPDATE customers

SET is\_vip = 'Y'

WHERE customer\_id = rec.customer\_id;

END LOOP;

COMMIT;

END;

**Output:**

**A group of green text

AI-generated content may be incorrect.**

**A screenshot of a phone

AI-generated content may be incorrect.**

**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

**Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

**Code:**

SET SERVEROUTPUT ON;

DECLARE

CURSOR due\_loans\_cur IS

SELECT l.loan\_id, l.customer\_id, l.due\_date, c.name, l.amount

FROM loans l

JOIN customers c ON l.customer\_id = c.customer\_id

WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30;

v\_loan\_id loans.loan\_id%TYPE;

v\_customer\_id loans.customer\_id%TYPE;

v\_due\_date loans.due\_date%TYPE;

v\_name customers.name%TYPE;

v\_amount loans.amount%TYPE;

BEGIN

OPEN due\_loans\_cur;

LOOP

FETCH due\_loans\_cur INTO v\_loan\_id, v\_customer\_id, v\_due\_date, v\_name, v\_amount;

EXIT WHEN due\_loans\_cur%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || v\_name ||

', your loan (ID: ' || v\_loan\_id ||

') of amount $' || v\_amount ||

' is due on ' || TO\_CHAR(v\_due\_date, 'DD-MON-YYYY') || '.');

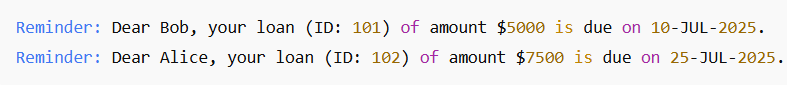
END LOOP;

CLOSE due\_loans\_cur;

END;

/

**Output:**

****

**Exercise 2: Error Handling**

**Scenario 1: Handle exceptions during fund transfers between accounts.**

**Question:** Write a stored procedure SafeTransferFunds that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

**Code:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_from\_balance NUMBER;

BEGIN

-- Start transaction

SAVEPOINT before\_transfer;

-- Check if the source account has enough balance

SELECT balance INTO v\_from\_balance FROM accounts WHERE account\_id = p\_from\_account;

IF v\_from\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in source account');

END IF;

-- Deduct from source account

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_account;

-- Add to destination account

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_account;

-- Commit transaction

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful from Account ' || p\_from\_account || ' to Account ' || p\_to\_account);

EXCEPTION

WHEN OTHERS THEN

-- Rollback on error

ROLLBACK TO before\_transfer;

-- Log the error

INSERT INTO error\_log(error\_message)

VALUES(SQLERRM);

COMMIT; -- Commit the log insert

DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' || SQLERRM);

END;

/

**Output:**

**A close-up of a sign

AI-generated content may be incorrect.**

**Scenario 2:** Manage errors when updating employee salaries**.**

**Question:** Write a stored procedure UpdateSalary that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

**Code:**

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_emp\_id IN NUMBER,

p\_percentage IN NUMBER

)

IS

BEGIN

-- Try to update the salary

UPDATE employees

SET salary = salary + (salary \* p\_percentage / 100)

WHERE emp\_id = p\_emp\_id;

-- Check if any row was affected

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Employee ID ' || p\_emp\_id || ' not found.');

END IF;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully for Employee ID: ' || p\_emp\_id);

EXCEPTION

WHEN OTHERS THEN

-- Log the error

INSERT INTO error\_log(error\_message)

VALUES(SQLERRM);

COMMIT; -- Commit the error log

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

/

**Output:**

****

**Scenario 3:** Ensure data integrity when adding a new customer**.**

**Question:** Write a stored procedure AddNewCustomer that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

**Code:**

CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_customer\_id IN NUMBER,

p\_customer\_name IN VARCHAR2,

p\_age IN NUMBER,

p\_balance IN NUMBER

)

IS

BEGIN

INSERT INTO customers(customer\_id, customer\_name, age, balance)

VALUES (p\_customer\_id, p\_customer\_name, p\_age, p\_balance);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer inserted successfully: ID = ' || p\_customer\_id);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

INSERT INTO error\_log(error\_message)

VALUES('Customer with ID ' || p\_customer\_id || ' already exists.');

COMMIT; -- commit the log insert

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_customer\_id || ' already exists.');

WHEN OTHERS THEN

INSERT INTO error\_log(error\_message)

VALUES('Unexpected error: ' || SQLERRM);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

/

**Output:**

**A close-up of a word

AI-generated content may be incorrect.**

**Exercise 3: Stored Procedures**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

**Question:** Write a stored procedure ProcessMonthlyInterest that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

**Code:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

UPDATE accounts

SET balance = balance + (balance \* 0.01)

WHERE account\_type = 'SAVINGS';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest applied to all savings accounts.');

END;

/

**Output**:

A black and red text

AI-generated content may be incorrect.

**Scenario 2**: The bank wants to implement a bonus scheme for employees based on their performance.

**Question**: Write a stored procedure UpdateEmployeeBonus that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

**Code:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department IN VARCHAR2,

p\_bonus\_pct IN NUMBER -- e.g., 0.10 for 10%

)

IS

BEGIN

UPDATE employees

SET salary = salary + (salary \* p\_bonus\_pct)

WHERE department = p\_department;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to department: ' || p\_department);

END;

/

**Output:**

**A close up of a word

AI-generated content may be incorrect.**

**Scenario 3:** Customers should be able to transfer funds between their accounts.

**Question**: Write a stored procedure TransferFunds that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer**.**

**Code:**

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_balance NUMBER;

BEGIN

SELECT balance INTO v\_balance

FROM accounts

WHERE account\_id = p\_from\_account

FOR UPDATE;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account.');

END IF;

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_account;

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_account;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || p\_amount || ' from account ' || p\_from\_account || ' to ' || p\_to\_account);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' || SQLERRM);

END;

/

**Output:**

****

**Exercise 4: Functions**

**Scenario 1:** Calculate the age of customers for eligibility checks.

**Question:** Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years**.**

**Code:**

CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE)

RETURN NUMBER

IS

v\_age NUMBER;

BEGIN

v\_age := FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

RETURN v\_age;

END;

/

**Output**:

A close-up of a number

AI-generated content may be incorrect.

**Scenario 2:** The bank needs to compute the monthly installment for a loan**.**

**Question**: Write a function CalculateMonthlyInstallment that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

**Code:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_loan\_amount NUMBER,

p\_annual\_rate NUMBER,

p\_years NUMBER

)

RETURN NUMBER

IS

v\_monthly\_rate NUMBER;

v\_months NUMBER;

v\_installment NUMBER;

BEGIN

v\_monthly\_rate := p\_annual\_rate / 12 / 100;

v\_months := p\_years \* 12;

IF v\_monthly\_rate = 0 THEN

v\_installment := p\_loan\_amount / v\_months;

ELSE

v\_installment := p\_loan\_amount \* v\_monthly\_rate /

(1 - POWER(1 + v\_monthly\_rate, -v\_months));

END IF;

RETURN ROUND(v\_installment, 2);

END;

/

**Output:**

A close up of a word

AI-generated content may be incorrect.

**Scenario 3:** Check if a customer has sufficient balance before making a transaction.

**Question:** Write a function HasSufficientBalance that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

**Code**:

CREATE OR REPLACE FUNCTION HasSufficientBalance(

p\_account\_id NUMBER,

p\_amount NUMBER

)

RETURN BOOLEAN

IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_account\_id;

RETURN v\_balance >= p\_amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

/

**Output:**

****

**Exercise 4: Functions**

**Scenario 1**: Calculate the age of customers for eligibility checks**.**

**Question**: Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.

**Code:**

CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE)

RETURN NUMBER IS

v\_age NUMBER;

BEGIN

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

RETURN v\_age;

END;

/

**Output:**

**A number on a white background

AI-generated content may be incorrect.**

**Scenario 2:** The bank needs to compute the monthly installment for a loan.

**Question:** Write a function CalculateMonthlyInstallment that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

**Code:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_amount NUMBER,

p\_interest\_rate NUMBER,

p\_years NUMBER

) RETURN NUMBER IS

v\_monthly\_rate NUMBER := p\_interest\_rate / 12 / 100;

v\_months NUMBER := p\_years \* 12;

v\_emi NUMBER;

BEGIN

v\_emi := (p\_amount \* v\_monthly\_rate \* POWER(1 + v\_monthly\_rate, v\_months)) /

(POWER(1 + v\_monthly\_rate, v\_months) - 1);

RETURN ROUND(v\_emi, 2);

END;

/

**Output:**

**A close up of a number

AI-generated content may be incorrect.**

**Scenario 3:** Check if a customer has sufficient balance before making a transaction.

**Question:** Write a function HasSufficientBalance that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

**Code:**

CREATE OR REPLACE FUNCTION HasSufficientBalance(p\_account\_id NUMBER, p\_amount NUMBER)

RETURN BOOLEAN IS

v\_balance NUMBER;

BEGIN

SELECT balance INTO v\_balance FROM Accounts WHERE account\_id = p\_account\_id;

IF v\_balance >= p\_amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

/

**Output:**

****

**Exercise 5: Triggers**

**Scenario 1**: Automatically update the last modified date when a customer's record is updated.

**Question:** Write a trigger UpdateCustomerLastModified that updates the LastModified column of the Customers table to the current date whenever a customer's record is updated.

**Code:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.last\_modified := SYSDATE;

END;

/

**Output:**

**A close-up of a date

AI-generated content may be incorrect.**

**Scenario 2:** Maintain an audit log for all transactions.

**Question**: Write a trigger LogTransaction that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table**.**

**Code:**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (transaction\_id, log\_time, message)

VALUES (:NEW.transaction\_id, SYSDATE,

'Transaction inserted: ' || :NEW.transaction\_type || ' of amount ' || :NEW.amount);

END;

/

**Output**:

A close-up of a message

AI-generated content may be incorrect.

**Scenario 3:** Enforce business rules on deposits and withdrawals**.**

**Question:** Write a trigger CheckTransactionRules that ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.

**code:**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

-- Get current balance

SELECT balance INTO v\_balance FROM Customers WHERE customer\_id = :NEW.account\_id;

IF :NEW.transaction\_type = 'WITHDRAW' THEN

IF :NEW.amount > v\_balance THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Withdrawal exceeds current balance');

END IF;

ELSIF :NEW.transaction\_type = 'DEPOSIT' THEN

IF :NEW.amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Deposit amount must be positive');

END IF;

END IF;

END;

/

**Output:**

**A close up of a word

AI-generated content may be incorrect.**

**Exercise 6: Cursors**

**Scenario 1:** Generate monthly statements for all customers**.**

**Question**: Write a PL/SQL block using an explicit cursor GenerateMonthlyStatements that retrieves all transactions for the current month and prints a statement for each customer**.**

**Code:**

DECLARE

CURSOR cur\_transactions IS

SELECT c.customer\_id, c.name, t.amount, t.transaction\_date

FROM Customers c

JOIN Transactions t ON c.customer\_id = t.customer\_id

WHERE EXTRACT(MONTH FROM t.transaction\_date) = EXTRACT(MONTH FROM SYSDATE)

AND EXTRACT(YEAR FROM t.transaction\_date) = EXTRACT(YEAR FROM SYSDATE);

v\_customer\_id Customers.customer\_id%TYPE;

v\_name Customers.name%TYPE;

v\_amount Transactions.amount%TYPE;

v\_date Transactions.transaction\_date%TYPE;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Monthly Statement:');

FOR rec IN cur\_transactions LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || rec.name ||

', Date: ' || rec.transaction\_date ||

', Amount: ' || rec.amount);

END LOOP;

END;

/

**Output:**

**A close up of numbers

AI-generated content may be incorrect.**

**Scenario 2**: Apply annual fee to all accounts.

**Question**: Write a PL/SQL block using an explicit cursor ApplyAnnualFee that deducts an annual maintenance fee from the balance of all accounts.

**Code:**

DECLARE

CURSOR cur\_accounts IS

SELECT account\_id, balance FROM Accounts;

v\_account\_id Accounts.account\_id%TYPE;

v\_balance Accounts.balance%TYPE;

annual\_fee CONSTANT NUMBER := 500;

BEGIN

FOR rec IN cur\_accounts LOOP

UPDATE Accounts

SET balance = balance - annual\_fee

WHERE account\_id = rec.account\_id;

DBMS\_OUTPUT.PUT\_LINE('Account ' || rec.account\_id || ' charged annual fee. New Balance: ' || (rec.balance - annual\_fee));

END LOOP;

END;

/

**Output:**

A close up of words

AI-generated content may be incorrect.

**Scenario 3:** Update the interest rate for all loans based on a new policy.

**Question:** Write a PL/SQL block using an explicit cursor UpdateLoanInterestRates that fetches all loans and updates their interest rates based on the new policy.

**Code:**

DECLARE

CURSOR cur\_loans IS

SELECT loan\_id, interest\_rate FROM Loans;

v\_loan\_id Loans.loan\_id%TYPE;

v\_interest Loans.interest\_rate%TYPE;

BEGIN

FOR rec IN cur\_loans LOOP

-- Example policy: increase by 0.5% if interest is below 8%

IF rec.interest\_rate < 8 THEN

UPDATE Loans

SET interest\_rate = interest\_rate + 0.5

WHERE loan\_id = rec.loan\_id;

DBMS\_OUTPUT.PUT\_LINE('Loan ' || rec.loan\_id || ' updated to new rate: ' || (rec.interest\_rate + 0.5));

ELSE

DBMS\_OUTPUT.PUT\_LINE('Loan ' || rec.loan\_id || ' unchanged. Current rate: ' || rec.interest\_rate);

END IF;

END LOOP;

END;

/

**Output:**

A close up of words

AI-generated content may be incorrect.

**Exercise 7: Packages**

**Scenario 1:** Group all customer-related procedures and functions into a package.

**Question:** Create a package **CustomerManagement** with procedures for adding a new customer, updating customer details, and a function to get customer balance.

**Code:**

-- Table creation (if not exists)

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Salary NUMBER

);

-- Package specification and body

CREATE OR REPLACE PACKAGE EmployeeManagement IS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_salary NUMBER);

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_name VARCHAR2);

FUNCTION CalculateAnnualSalary(p\_id NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_salary NUMBER) IS

BEGIN

INSERT INTO Employees VALUES (p\_id, p\_name, p\_salary);

END;

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_name VARCHAR2) IS

BEGIN

UPDATE Employees SET Name = p\_name WHERE EmployeeID = p\_id;

END;

FUNCTION CalculateAnnualSalary(p\_id NUMBER) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_id;

RETURN v\_salary \* 12;

END;

END EmployeeManagement;

/

-- Test block

BEGIN

EmployeeManagement.HireEmployee(101, 'Bob', 5000);

EmployeeManagement.UpdateEmployee(101, 'Bob Wilson');

DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ' || EmployeeManagement.CalculateAnnualSalary(101));

END;

/

**Output:**

**A close up of a word

AI-generated content may be incorrect.**

**Scenario 2:** Create a package to manage employee data.

**Question:** Write a package **EmployeeManagement** with procedures to hire new employees, update employee details, and a function to calculate annual salary.

**Code:**

-- Table creation (if not exists)

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER

);

-- Package specification and body

CREATE OR REPLACE PACKAGE AccountOperations IS

PROCEDURE OpenAccount(p\_acc\_id NUMBER, p\_cust\_id NUMBER, p\_balance NUMBER);

PROCEDURE CloseAccount(p\_acc\_id NUMBER);

FUNCTION GetTotalBalance(p\_cust\_id NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

PROCEDURE OpenAccount(p\_acc\_id NUMBER, p\_cust\_id NUMBER, p\_balance NUMBER) IS

BEGIN

INSERT INTO Accounts VALUES (p\_acc\_id, p\_cust\_id, p\_balance);

END;

PROCEDURE CloseAccount(p\_acc\_id NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_acc\_id;

END;

FUNCTION GetTotalBalance(p\_cust\_id NUMBER) RETURN NUMBER IS

v\_total NUMBER := 0;

BEGIN

SELECT SUM(Balance) INTO v\_total FROM Accounts WHERE CustomerID = p\_cust\_id;

RETURN NVL(v\_total, 0);

END;

END AccountOperations;

/

-- Test block

BEGIN

AccountOperations.OpenAccount(5001, 1, 2000);

AccountOperations.OpenAccount(5002, 1, 3000);

DBMS\_OUTPUT.PUT\_LINE('Total Balance: ' || AccountOperations.GetTotalBalance(1));

AccountOperations.CloseAccount(5002);

DBMS\_OUTPUT.PUT\_LINE('Balance After Closing One Account: ' || AccountOperations.GetTotalBalance(1));

END;

/

**Output**:

A close up of a word

AI-generated content may be incorrect.

**Scenario 3:** Group all account-related operations into a package.

**Question:** Create a package **AccountOperations** with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.

**Code:**

-- Table creation (if not exists)

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER

);

-- Package specification and body

CREATE OR REPLACE PACKAGE AccountOperations IS

PROCEDURE OpenAccount(p\_acc\_id NUMBER, p\_cust\_id NUMBER, p\_balance NUMBER);

PROCEDURE CloseAccount(p\_acc\_id NUMBER);

FUNCTION GetTotalBalance(p\_cust\_id NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

PROCEDURE OpenAccount(p\_acc\_id NUMBER, p\_cust\_id NUMBER, p\_balance NUMBER) IS

BEGIN

INSERT INTO Accounts VALUES (p\_acc\_id, p\_cust\_id, p\_balance);

END;

PROCEDURE CloseAccount(p\_acc\_id NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_acc\_id;

END;

FUNCTION GetTotalBalance(p\_cust\_id NUMBER) RETURN NUMBER IS

v\_total NUMBER := 0;

BEGIN

SELECT SUM(Balance) INTO v\_total FROM Accounts WHERE CustomerID = p\_cust\_id;

RETURN NVL(v\_total, 0);

END;

END AccountOperations;

/

-- Test block

BEGIN

AccountOperations.OpenAccount(5001, 1, 2000);

AccountOperations.OpenAccount(5002, 1, 3000);

DBMS\_OUTPUT.PUT\_LINE('Total Balance: ' || AccountOperations.GetTotalBalance(1));

AccountOperations.CloseAccount(5002);

DBMS\_OUTPUT.PUT\_LINE('Balance After Closing One Account: ' || AccountOperations.GetTotalBalance(1));

END;

/

**Output:**

**A close-up of a sign

AI-generated content may be incorrect.**