**Exercise 1: Control Structures**

public class PLSQLControlStructures {

public static void main(String[] args) {

int age = 25;

if (age >= 18) {

System.out.println("Adult");

} else if (age >= 13) {

System.out.println("Teen");

} else {

System.out.println("Child");

}

for (int i = 1; i <= 5; i++) {

System.out.println("Iteration: " + i);

}

int counter = 1;

while (counter <= 5) {

System.out.println("Counter: " + counter);

counter++;

}

counter = 1;

while (true) {

System.out.println("Counter: " + counter);

counter++;

if (counter > 5) {

break;

}

}

try {

int result = 10 / 0;

} catch (ArithmeticException e) {

System.out.println("Error: Division by zero");

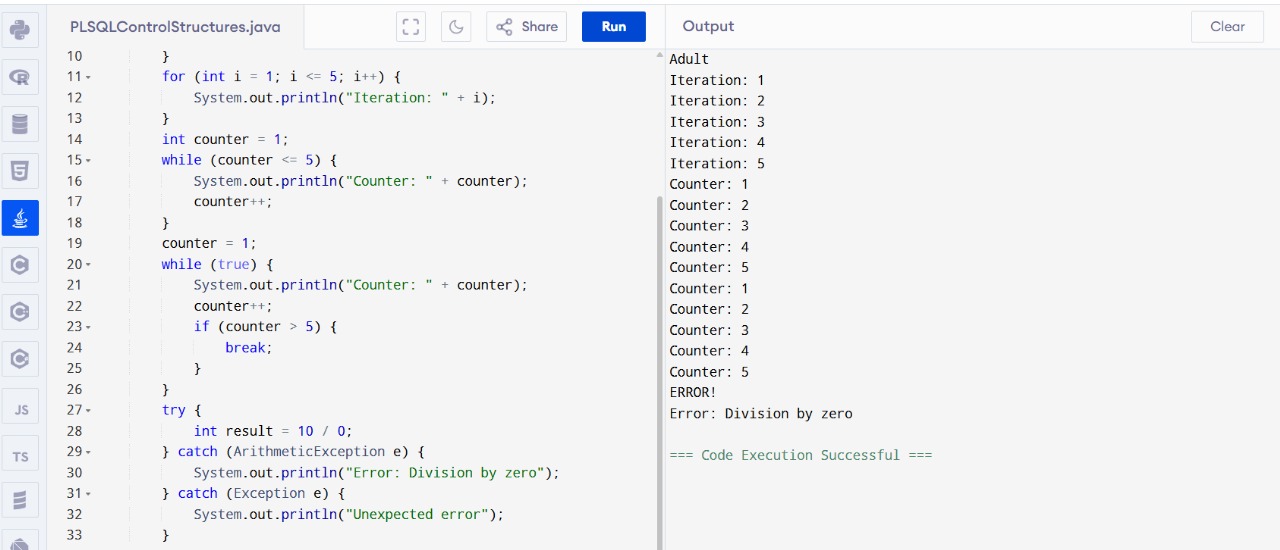
} catch (Exception e) {

System.out.println("Unexpected error");

}

}

}



**Exercise 3: Stored Procedures**

import java.util.HashMap;

import java.util.Map;

public class Main {

static Map<Integer, String[]> employees = new HashMap<>();

static String[] getEmployeeById(int empId) {

employees.put(1, new String[]{"John Doe", "50000"});

employees.put(2, new String[]{"Jane Smith", "60000"});

return employees.getOrDefault(empId, new String[]{"Not Found", "0"});

}

public static void main(String[] args) {

int employeeId = 1;

String[] result = getEmployeeById(employeeId);

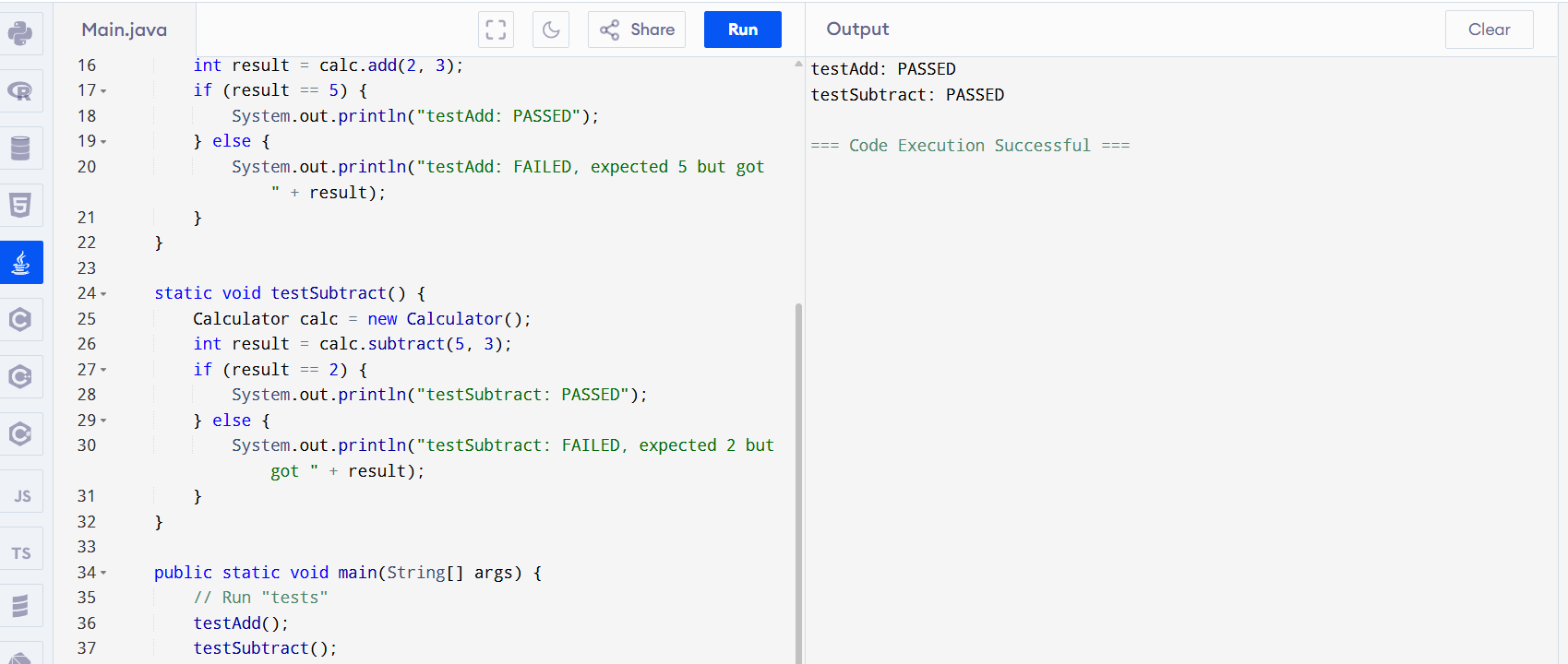
System.out.println("ID: " + employeeId);

System.out.println("Name: " + result[0]);

System.out.println("Salary: " + result[1]);

}

}



**Exercise 1: Setting Up Junit**

public class Main {

static class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

static void testAdd() {

Calculator calc = new Calculator();

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

if (result == 5) {

System.out.println("testAdd: PASSED");

} else {

System.out.println("testAdd: FAILED, expected 5 but got " + result);

}

}

static void testSubtract() {

Calculator calc = new Calculator();

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

if (result == 2) {

System.out.println("testSubtract: PASSED");

} else {

System.out.println("testSubtract: FAILED, expected 2 but got " + result);

}

}

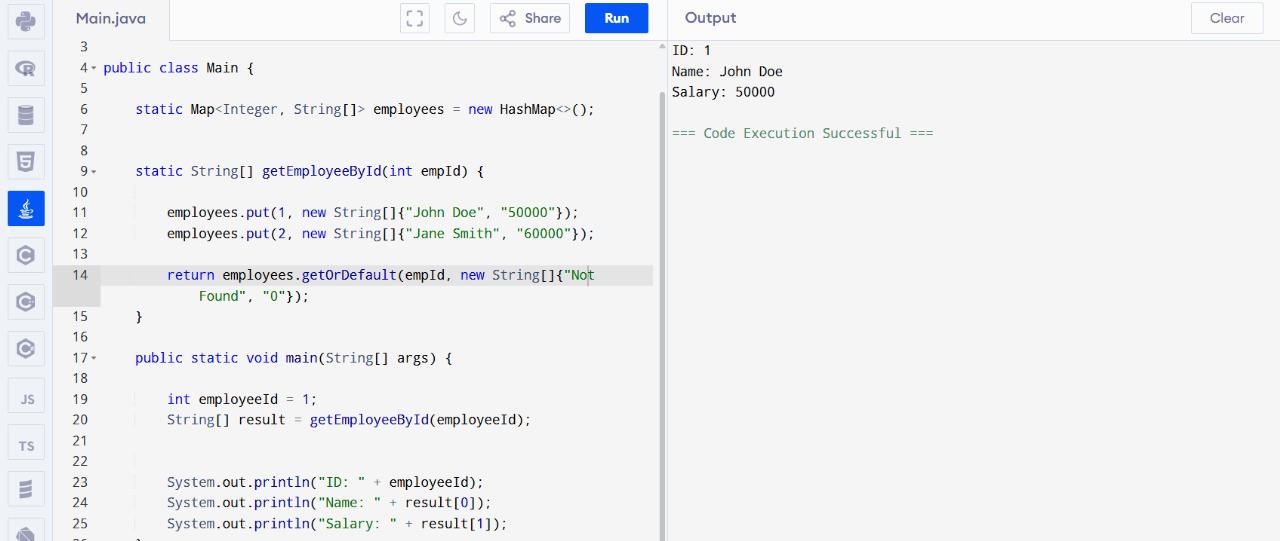
public static void main(String[] args) {

testAdd();

testSubtract();

}

}



**Exercise 3: Assertions in Junit**

public class Main {

static class Calculator {

public int add(int a, int b) {

return a + b;

}

}

static void assertEquals(int expected, int actual, String message) {

if (expected == actual) {

System.out.println("Test PASSED: " + message);

} else {

System.out.println("Test FAILED: " + message + ", expected " + expected + " but got " + actual);

}

}

static void testAdd() {

Calculator calc = new Calculator();

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

assertEquals(5, result, "2 + 3 should equal 5");

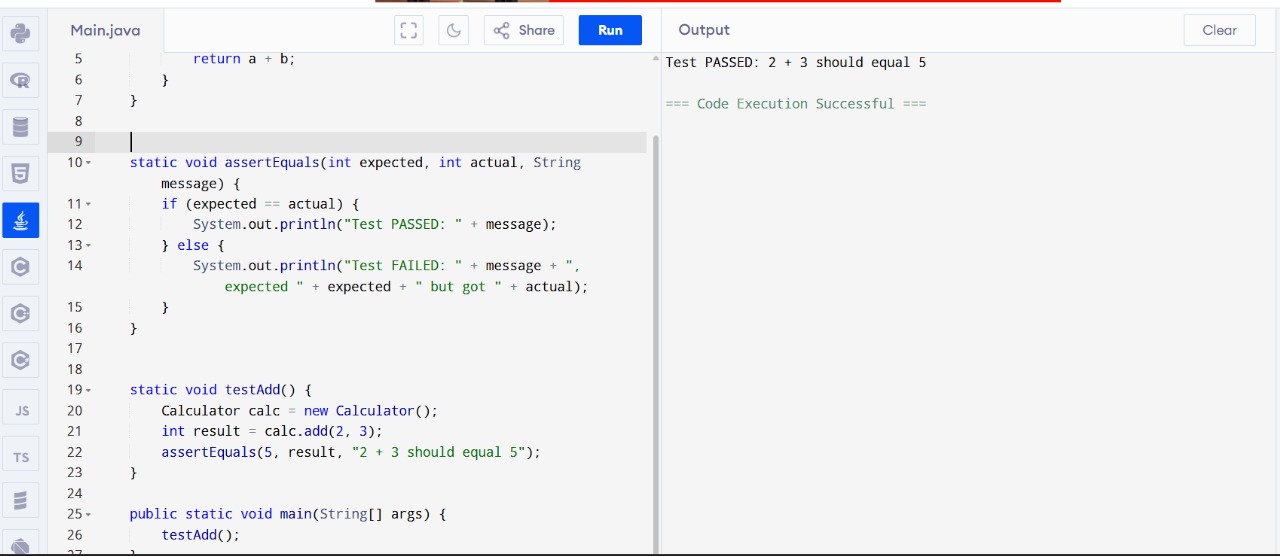
}

public static void main(String[] args) {

testAdd();

}

}



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

public class Main {

static class Calculator {

private int lastResult;

public void add(int a, int b) {

lastResult = a + b;

}

public int getLastResult() {

return lastResult;

}

}

static Calculator calc;

static void setUp() {

calc = new Calculator();

System.out.println("Setup: Initialized Calculator");

}

static void tearDown() {

calc = null;

System.out.println("Teardown: Cleaned up Calculator");

}

static void assertEquals(int expected, int actual, String message) {

if (expected == actual) {

System.out.println("Test PASSED: " + message);

} else {

System.out.println("Test FAILED: " + message + ", expected " + expected + " but got " + actual);

}

}

static void testAdd() {

setUp();

int a = 3, b = 4;

calc.add(a, b);

assertEquals(7, calc.getLastResult(), "3 + 4 should equal 7");

tearDown();

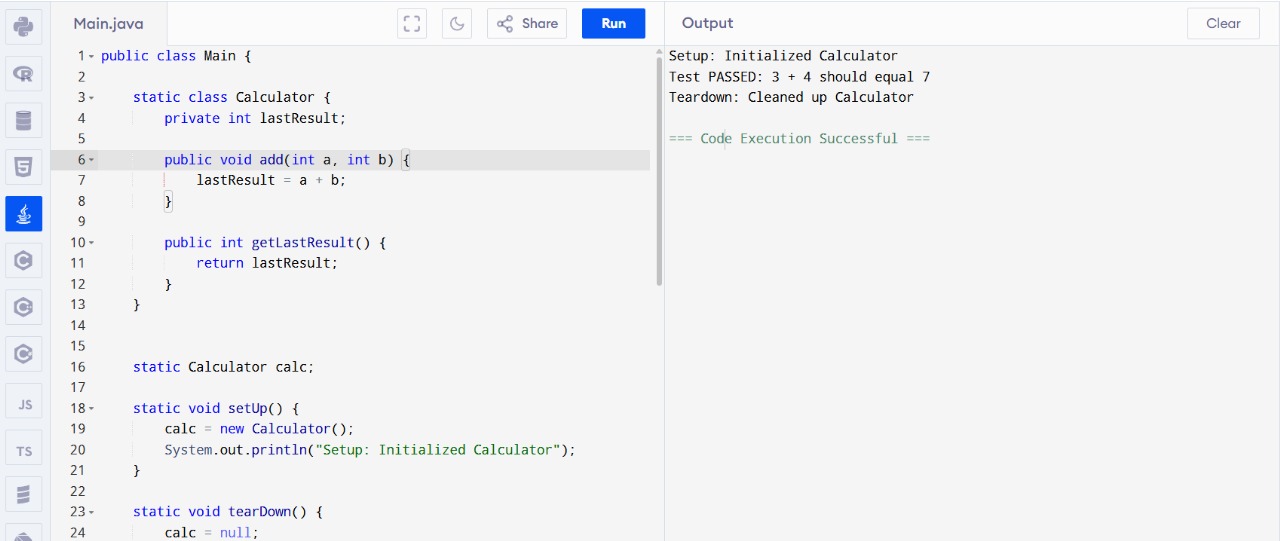
}

public static void main(String[] args) {

testAdd();

}

}



**Exercise 1: Mocking and Stubbing**

public class Main {

static interface DataService {

int getData();

}

static class RealDataService implements DataService {

public int getData() {

return 10;

}

}

static class StubDataService implements DataService {

public int getData() {

return 5;

}

}

static class Processor {

private DataService dataService;

public Processor(DataService dataService) {

this.dataService = dataService;

}

public int process() {

return dataService.getData() \* 2;

}

}

static void assertEquals(int expected, int actual, String message) {

if (expected == actual) {

System.out.println("Test PASSED: " + message);

} else {

System.out.println("Test FAILED: " + message + ", expected " + expected + " but got " + actual);

}

}

static void testProcessWithStub() {

DataService stub = new StubDataService();

Processor processor = new Processor(stub);

int result = processor.process();

assertEquals(10, result, "Stubbed data (5) \* 2 should equal 10");

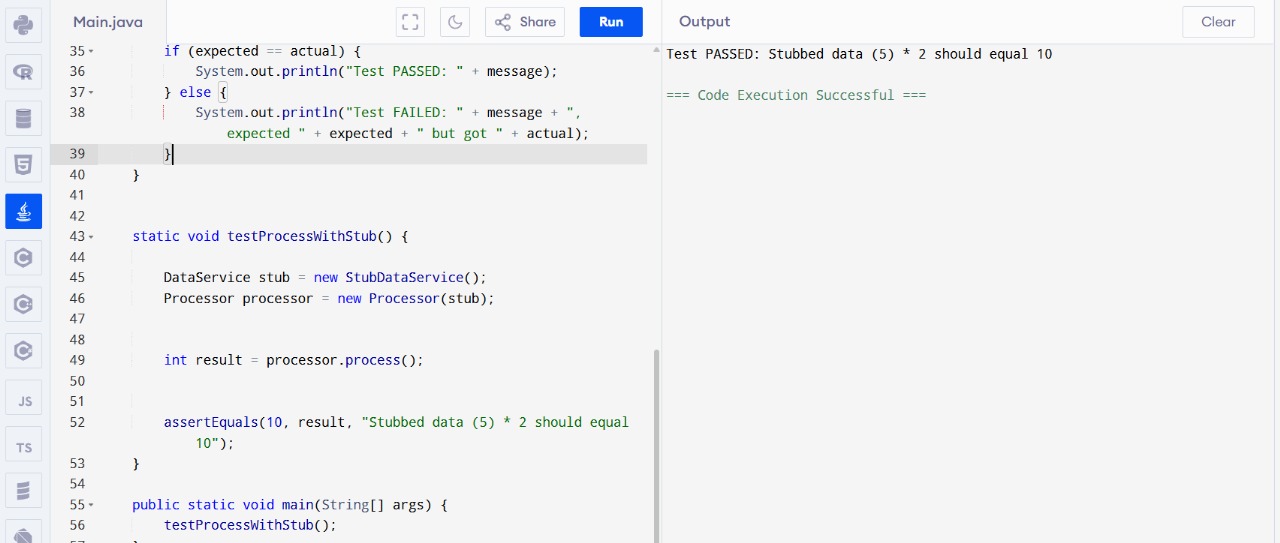
}

public static void main(String[] args) {

testProcessWithStub();

}

}



**Exercise 2: Verifying Interactions**

public class Main {

static interface Logger {

void log(String message);

}

static class LoggerStub implements Logger {

boolean wasCalled = false;

public void log(String message) {

wasCalled = true;

System.out.println("Logged: " + message);

}

}

static class Service {

private Logger logger;

public Service(Logger logger) {

this.logger = logger;

}

public void doWork() {

logger.log("Work done");

}

}

static void assertTrue(boolean condition, String message) {

if (condition) {

System.out.println("Test PASSED: " + message);

} else {

System.out.println("Test FAILED: " + message);

}

}

static void testLoggerInteraction() {

LoggerStub logger = new LoggerStub();

Service service = new Service(logger);

service.doWork();

assertTrue(logger.wasCalled, "Logger should have been called");

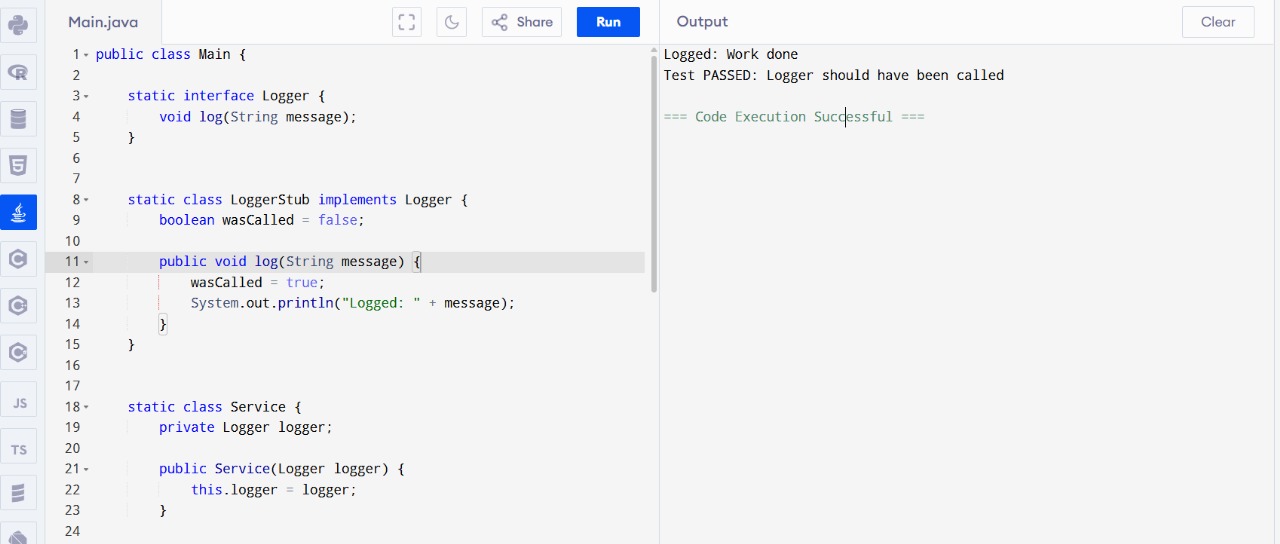
}

public static void main(String[] args) {

testLoggerInteraction();

}

}



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

public class Main {

static class SimpleLogger {

public static void error(String message) {

System.out.println("[ERROR] " + message);

}

public static void warn(String message) {

System.out.println("[WARN] " + message);

}

}

static class Application {

public void process(int value) {

if (value < 0) {

SimpleLogger.error("Invalid value: " + value);

} else {

SimpleLogger.warn("Processing value: " + value);

}

}

}

static void assertEquals(String expected, String actual, String message) {

if (expected.equals(actual)) {

System.out.println("Test PASSED: " + message);

} else {

System.out.println("Test FAILED: " + message + ", expected " + expected + " but got " + actual);

}

}

static void testLogging() {

Application app = new Application();

app.process(-1);

app.process(5);

System.out.println("Check console for [ERROR] and [WARN] messages");

}

public static void main(String[] args) {

testLogging();

}

}

