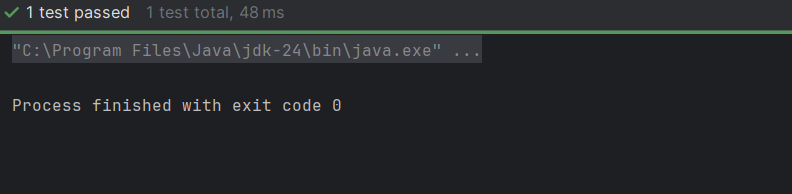
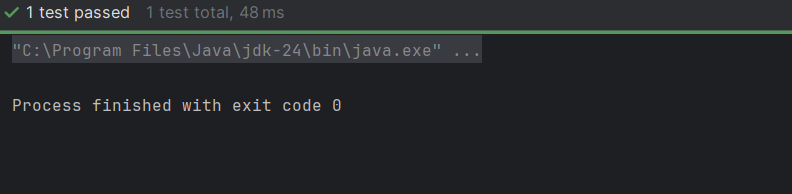
**Exercise 1: Setting Up JUnit**

import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
  
public class Main {  
  
 @Test  
 void testAddition() {  
 *assertEquals*(4, 2 + 2); // expected, actual  
 }  
}



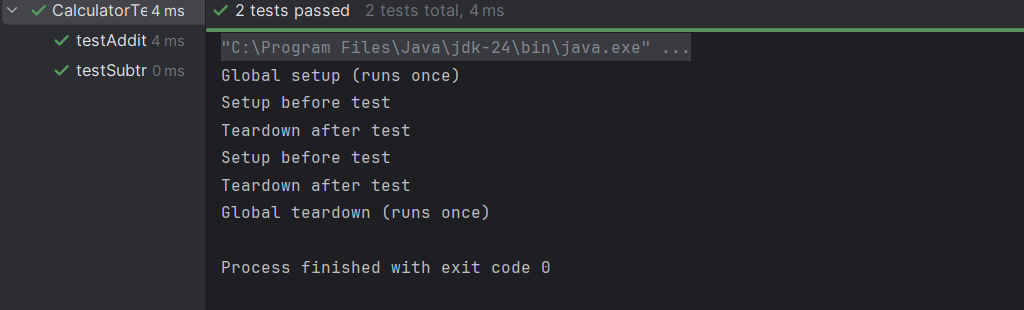
**Exercise 3: Assertions in JUnit**

import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class AssertionsTest {  
  
 @Test  
 public void testAssertions() {  
 *assertEquals*(5, 2 + 3);  
 *assertTrue*(5 > 3);  
 *assertFalse*(5 < 3);  
 *assertNull*(null);  
 *assertNotNull*(new Object());  
 }  
}



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

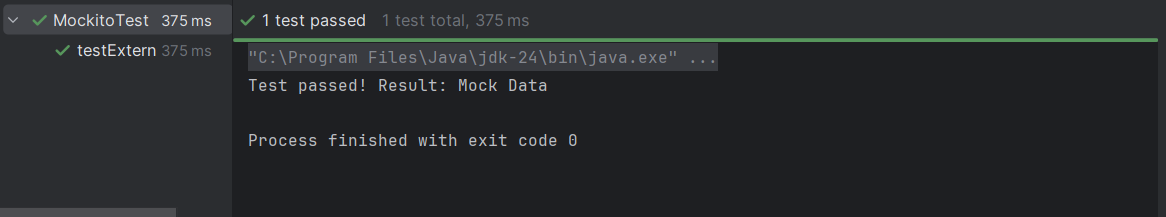
import org.junit.\*;  
import static org.junit.Assert.\*;  
  
public class CalculatorTest {  
  
 private Calculator calculator;  
  
 @Before  
 public void setUp() {  
 calculator = new Calculator();  
 System.*out*.println("Setup before test");  
 }  
  
 @After  
 public void tearDown() {  
 calculator = null;  
 System.*out*.println("Teardown after test");  
 }  
  
 @BeforeClass  
 public static void initAll() {  
 System.*out*.println("Global setup (runs once)");  
 }  
  
 @AfterClass  
 public static void tearDownAll() {  
 System.*out*.println("Global teardown (runs once)");  
 }  
  
 @Test  
 public void testAddition() {  
 // Arrange  
 int a = 5;  
 int b = 3;  
  
 // Act  
 int result = calculator.add(a, b);  
  
 // Assert  
 *assertEquals*(8, result);  
 }  
  
 @Test  
 public void testSubtraction() {  
 // Arrange  
 int a = 10;  
 int b = 4;  
 int result = calculator.subtract(a, b);  
  
 // Assert  
 *assertEquals*(6, result);  
 }  
}  
  
class Calculator {  
 public int add(int a, int b) {  
 return a + b;  
 }  
 public int subtract(int a, int b) {  
 return a - b;  
 }  
}



**Mockito exercises**

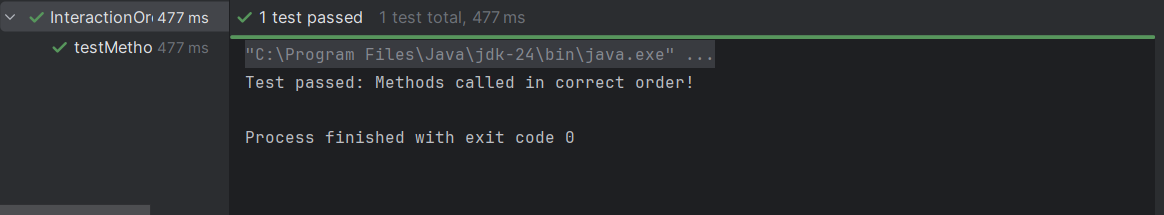
**Exercise 1: Mocking and Stubbing**

import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
import static org.mockito.Mockito.\*;  
  
interface ExternalApi {  
 String getData();  
}  
  
class MyService {  
 private final ExternalApi api;  
 public MyService(ExternalApi api) { this.api = api; }  
 public String fetchData() { return api.getData(); }  
}  
  
public class MockitoTest {  
 @Test  
 public void testExternalApi() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 *when*(mockApi.getData()).thenReturn("Mock Data");  
  
 MyService service = new MyService(mockApi);  
 String result = service.fetchData();  
  
 *assertEquals*("Mock Data", result);  
 System.*out*.println("Test passed! Result: " + result);  
 }  
}



**Exercise 2: Verifying Interactions**

import org.junit.jupiter.api.Test;  
import org.mockito.InOrder;  
import static org.mockito.Mockito.\*;  
  
  
class OrderDependentService {  
 private final Dependency dependency;  
 public OrderDependentService(Dependency dependency) {  
 this.dependency = dependency;  
 }  
 public void executeProcess() {  
 dependency.start();  
 dependency.process();  
 dependency.end();  
 }  
}  
  
interface Dependency {  
 void start();  
 void process();  
 void end();  
}  
  
  
public class InteractionOrderTest {  
 @Test  
 public void testMethodCallOrder() {  
 // 1. Create mock  
 Dependency mockDependency = *mock*(Dependency.class);  
  
 OrderDependentService service = new OrderDependentService(mockDependency);  
 service.executeProcess();  
  
   
 InOrder inOrder = *inOrder*(mockDependency);  
 inOrder.verify(mockDependency).start();  
 inOrder.verify(mockDependency).process();  
 inOrder.verify(mockDependency).end();  
  
 System.*out*.println("Test passed: Methods called in correct order!");  
 }  
}



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

import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
public class LoggingExample {  
 // Initialize logger  
 private static final Logger *logger* = LoggerFactory.*getLogger*(LoggingExample.class);  
  
 public static void main(String[] args) {  
 // Log messages at different levels  
 *logger*.error("This is an ERROR message - something went wrong!");  
 *logger*.warn("This is a WARNING message - potential issue detected");  
 *logger*.info("This is an INFO message - normal operation");  
 *logger*.debug("This is a DEBUG message - detailed information");  
  
 // Example with variable  
 int userId = 42;  
 *logger*.info("User with ID {} logged in", userId);  
 }  
}

