# **JUnit Tutorial | Testing Framework for Java**

JUnit tutorial provides basic and advanced concepts of **unit testing in java** with examples. Our junit tutorial is designed for beginners and professionals.

It is an *open-source testing framework* for java programmers. The java programmer can create test cases and test his/her own code.

It is one of the unit testing framework. Current version is junit 4.

To perform unit testing, we need to create test cases. The **unit test case** is a code which ensures that the program logic works as expected.

The **org.junit** package contains many interfaces and classes for junit testing such as Assert, Test, Before, After etc.

## **Types of unit testing**

There are two ways to perform unit testing: 1) manual testing 2) automated testing.

#### **1) Manual Testing**

If you execute the test cases manually without any tool support, it is known as manual testing. It is time consuming and less reliable.

#### **2) Automated Testing**

If you execute the test cases by tool support, it is known as automated testing. It is fast and more reliable.

## **1. Introduction to JUnit**

JUnit is a popular **testing framework** for Java. It allows developers to write and run repeatable tests to ensure that their code functions correctly.

### ✅ ****Why Use JUnit?****

* Automates testing
* Improves code reliability
* Supports Test-Driven Development (TDD)
* Provides assertions to verify expected results

### ✅ ****JUnit Annotations****

| **Annotation** | **Description** |
| --- | --- |
| @Test | Marks a method as a test case |
| @BeforeEach | Runs before each test |
| @AfterEach | Runs after each test |
| @BeforeAll | Runs once before all tests |
| @AfterAll | Runs once after all tests |
| @Disabled | Skips a test case |
| @RepeatedTest | Runs a test multiple times |

## 📌 **2. Setting Up JUnit in Java (JUnit 5)**

### ✅ ****Step 1: Create a Java Project****

Use **Maven** or **Gradle** to add dependencies.

#### **Using Maven**

Add this dependency to pom.xml:

xml

CopyEdit

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<version>5.9.2</version>

<scope>test</scope>

</dependency>

</dependencies>

## 📌 **3. Writing Basic Test Cases**

Let's write a simple test case using JUnit.

#### ✅ **Example: Testing Addition in Calculator**

java

CopyEdit

import org.junit.jupiter.api.Test;

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

class CalculatorTest {

@Test

void testAddition() {

int result = 2 + 3;

assertEquals(5, result);

}

}

### ✅ ****Explanation****

1. @Test marks the method as a test case.
2. assertEquals(5, result); verifies that the result is 5.

## 📌 **4. Assertions in JUnit**

Assertions help in verifying expected vs. actual results.

### ✅ ****Common Assertions****

| **Assertion** | **Description** |
| --- | --- |
| assertEquals(a, b) | Checks if a equals b |
| assertNotEquals(a, b) | Checks if a is not equal to b |
| assertTrue(condition) | Checks if condition is true |
| assertFalse(condition) | Checks if condition is false |
| assertNull(object) | Checks if object is null |
| assertNotNull(object) | Checks if object is not null |

#### ✅ **Example: Using Assertions**

java

CopyEdit

import org.junit.jupiter.api.Test;

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

class AssertionTest {

@Test

void testAssertions() {

String str = "JUnit";

assertNotNull(str);

assertEquals("JUnit", str);

assertTrue(5 > 2);

}

}

## 📌 **5. Test Fixtures (**@BeforeEach **and** @AfterEach**)**

Fixtures initialize objects before running tests.

#### ✅ **Example: Using** @BeforeEach **and** @AfterEach

java

CopyEdit

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

class FixtureTest {

@BeforeEach

void setup() {

System.out.println("Setup before test");

}

@Test

void test1() {

System.out.println("Executing Test 1");

}

@AfterEach

void tearDown() {

System.out.println("Cleanup after test");

}

}

### ✅ ****Explanation****

1. @BeforeEach runs **before** each test.
2. @AfterEach runs **after** each test.

## 📌 **6. Parameterized Tests**

Parameterized tests allow running the same test with different inputs.

#### ✅ **Example: Running a Test with Multiple Values**

java

CopyEdit

import org.junit.jupiter.params.ParameterizedTest;

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

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

class ParameterizedTestExample {

@ParameterizedTest

@ValueSource(ints = {2, 4, 6, 8})

void testEvenNumbers(int number) {

assertTrue(number % 2 == 0);

}

}

### ✅ ****Explanation****

* @ValueSource(ints = {2, 4, 6, 8}) provides multiple test values.
* assertTrue(number % 2 == 0); verifies if the number is even.

## 📌 **7. Test Suites**

A **test suite** runs multiple test classes together.

#### ✅ **Example: Running Multiple Tests**

java

CopyEdit

import org.junit.platform.suite.api.\*;

@Suite

@SelectClasses({ CalculatorTest.class, AssertionTest.class })

class TestSuiteExample { }

## 📌 **8. Mockito (Mocking Dependencies)**

Mockito is used to **mock dependencies** in unit tests.

### ✅ ****Example: Mocking a Service Layer****

java

CopyEdit

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

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

class ServiceTest {

@Test

void testMocking() {

MyService mockService = Mockito.mock(MyService.class);

Mockito.when(mockService.getData()).thenReturn("Mock Data");

assertEquals("Mock Data", mockService.getData());

}

}

class MyService {

String getData() {

return "Real Data";

}

}

## 📌 **9. Rest Assured (Testing REST APIs)**

Rest Assured is used for **API testing**.

### ✅ ****Example: Testing an API****

java

CopyEdit

import io.restassured.RestAssured;

import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.\*;

import static org.hamcrest.Matchers.\*;

class RestAssuredTest {

@Test

void testGetRequest() {

RestAssured.baseURI = "https://jsonplaceholder.typicode.com";

given().

when().

get("/posts/1").

then().

statusCode(200).

body("id", equalTo(1));

}

}

### ✅ ****Explanation****

1. given().when().get("/posts/1") makes a GET request.
2. then().statusCode(200) checks the response status.
3. body("id", equalTo(1)) verifies the response body.

## 📌 **10. Code Coverage**

Code coverage measures how much of your code is tested.

### ✅ ****Using Jacoco for Code Coverage****

1. Add this to pom.xml:

xml

CopyEdit

<build>

<plugins>

<plugin>

<groupId>org.jacoco</groupId>

<artifactId>jacoco-maven-plugin</artifactId>

<version>0.8.7</version>

<executions>

<execution>

<goals>

<goal>prepare-agent</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

1. Run:

sh

CopyEdit

mvn clean test

mvn jacoco:report

1. Open target/site/jacoco/index.html to view coverage.

## 🎯 **Conclusion**

We covered: ✅ **JUnit Basics**  
✅ **Assertions & Test Fixtures**  
✅ **Parameterized Tests & Test Suites**  
✅ **Mockito for Mocking**  
✅ **Rest Assured for API Testing**  
✅ **Code Coverage with Jacoco**

ere’s a simple **Java class** and a corresponding **JUnit 5 test case**.

### ****Step 1: Simple Java Class (MathUtils.java)****

java

CopyEdit

public class MathUtils {

public int add(int a, int b) {

return a + b;

}

}

### ****Step 2: JUnit 5 Test Class (MathUtilsTest.java)****

java

CopyEdit

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

import org.junit.jupiter.api.Test;

class MathUtilsTest {

@Test

void testAdd() {

MathUtils mathUtils = new MathUtils();

int result = mathUtils.add(5, 3);

assertEquals(8, result, "5 + 3 should be 8");

}

}

### ****Explanation:****

1. **MathUtils** - A simple class with an add() method.
2. **MathUtilsTest** - A JUnit 5 test class.
3. **@Test** - Marks the method as a test case.
4. **assertEquals(8, result, "5 + 3 should be 8")** - Checks if the result is correct.

🚀 **Now you can run this test using JUnit 5!**

4o

##### You said:

some more examples like above

###### ChatGPT said:

ChatGPT

Here are a few more simple Java methods along with their corresponding **JUnit 5 test cases**:

### ****Example 1: Subtraction Method****

#### **Step 1: Java Class (**MathUtils.java**)**

java

CopyEdit

public class MathUtils {

public int subtract(int a, int b) {

return a - b;

}

}

#### **Step 2: JUnit 5 Test (**MathUtilsTest.java**)**

java

CopyEdit

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

import org.junit.jupiter.api.Test;

class MathUtilsTest {

@Test

void testSubtract() {

MathUtils mathUtils = new MathUtils();

int result = mathUtils.subtract(10, 3);

assertEquals(7, result, "10 - 3 should be 7");

}

}

### ****Example 2: Multiply Method****

#### **Step 1: Java Class (**MathUtils.java**)**

java

CopyEdit

public class MathUtils {

public int multiply(int a, int b) {

return a \* b;

}

}

#### **Step 2: JUnit 5 Test (**MathUtilsTest.java**)**

java

CopyEdit

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

import org.junit.jupiter.api.Test;

class MathUtilsTest {

@Test

void testMultiply() {

MathUtils mathUtils = new MathUtils();

int result = mathUtils.multiply(4, 5);

assertEquals(20, result, "4 \* 5 should be 20");

}

}

### ****Example 3: Divide Method****

#### **Step 1: Java Class (**MathUtils.java**)**

java

CopyEdit

public class MathUtils {

public int divide(int a, int b) {

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

return a / b;

}

}

#### **Step 2: JUnit 5 Test (**MathUtilsTest.java**)**

java

CopyEdit

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

import org.junit.jupiter.api.Test;

class MathUtilsTest {

@Test

void testDivide() {

MathUtils mathUtils = new MathUtils();

int result = mathUtils.divide(10, 2);

assertEquals(5, result, "10 / 2 should be 5");

}

@Test

void testDivideByZero() {

MathUtils mathUtils = new MathUtils();

assertThrows(IllegalArgumentException.class, () -> mathUtils.divide(10, 0), "Cannot divide by zero");

}

}

### ****Example 4: Square Root Method****

#### **Step 1: Java Class (**MathUtils.java**)**

java

CopyEdit

public class MathUtils {

public double sqrt(double a) {

if (a < 0) throw new IllegalArgumentException("Cannot take square root of negative number");

return Math.sqrt(a);

}

}

#### **Step 2: JUnit 5 Test (**MathUtilsTest.java**)**

java

CopyEdit

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

import org.junit.jupiter.api.Test;

class MathUtilsTest {

@Test

void testSqrt() {

MathUtils mathUtils = new MathUtils();

double result = mathUtils.sqrt(16);

assertEquals(4, result, "Square root of 16 should be 4");

}

@Test

void testSqrtNegativeNumber() {

MathUtils mathUtils = new MathUtils();

assertThrows(IllegalArgumentException.class, () -> mathUtils.sqrt(-4), "Cannot take square root of negative number");

}

}

### ****Example 5: Power Method****

#### **Step 1: Java Class (**MathUtils.java**)**

java

CopyEdit

public class MathUtils {

public double power(double base, double exponent) {

return Math.pow(base, exponent);

}

}

#### **Step 2: JUnit 5 Test (**MathUtilsTest.java**)**

java

CopyEdit

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

import org.junit.jupiter.api.Test;

class MathUtilsTest {

@Test

void testPower() {

MathUtils mathUtils = new MathUtils();

double result = mathUtils.power(2, 3);

assertEquals(8, result, "2^3 should be 8");

}

}

### ****Explanation:****

1. **Simple Java methods**: These methods perform basic mathematical operations like subtraction, multiplication, division, square root, and power.
2. **JUnit 5 Tests**: Each test checks whether the method's output is correct using assertEquals or assertThrows (for exceptions).
3. **Edge cases**: For division, we handle division by zero; for square root, we handle negative inputs.

These are great beginner-level examples to practice writing JUnit tests in Java. 😊

4o mini

**Steps:**

1. **Use @Mock** to mock customerService.
2. **Use @InjectMocks** to inject the mock into CustomerController.
3. **Use when() from Mockito** to define the behavior of the mock.
4. **Use assertEquals** to verify the response.

Java:

@Autowired

**private** CustomerService customerService=**new** CustomerService();

@GetMapping

**public** List<Customer> getAllCustomers() {

**return** customerService.getAllCustomers();

}

**JUnit Test Code:**

java

CopyEdit

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

import static org.mockito.Mockito.\*;

import java.util.Arrays;

import java.util.List;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.extension.ExtendWith;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.junit.jupiter.MockitoExtension;

import org.springframework.http.ResponseEntity;

class CustomerControllerTest {

@ExtendWith(MockitoExtension.class) // Enable Mockito in JUnit 5

class CustomerControllerTest {

@Mock

private CustomerService customerService;

@InjectMocks

private CustomerController customerController;

private List<Customer> customerList;

@BeforeEach

void setUp() {

customerList = Arrays.asList(

new Customer(1L, "John Doe"),

new Customer(2L, "Jane Smith")

);

}

@Test

void testGetAllCustomers() {

// Mocking the service response

when(customerService.getAllCustomers()).thenReturn(customerList);

// Calling the actual controller method

List<Customer> result = customerController.getAllCustomers();

// Verifying the response

assertEquals(2, result.size());

assertEquals("John Doe", result.get(0).getName());

assertEquals("Jane Smith", result.get(1).getName());

// Verify that the service method was called once

verify(customerService, times(1)).getAllCustomers();

}

}

}

**Explanation:**

1. **@Mock** - Mocks CustomerService.
2. **@InjectMocks** - Injects customerService into CustomerController.
3. **when(customerService.getAllCustomers()).thenReturn(customerList);** - Defines the mock behavior.
4. **assertEquals()** - Checks if the response matches expectations.
5. **verify(customerService, times(1)).getAllCustomers();** - Ensures getAllCustomers() is called once.

# **1. Writing More Test Cases with Assertions**

### ✅ ****Example: Testing String Operations****

java

CopyEdit

import org.junit.jupiter.api.Test;

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

class StringTest {

@Test

void testStringOperations() {

String str = "JUnit Testing";

// Verify the string is not null

assertNotNull(str);

// Check if string starts with "JUnit"

assertTrue(str.startsWith("JUnit"));

// Verify length of string

assertEquals(14, str.length());

// Check if two strings are equal (ignoring case)

assertEquals("junit testing", str.toLowerCase());

// Verify that the string does not contain "ABC"

assertFalse(str.contains("ABC"));

}

}

### ✅ ****Step-by-Step Explanation****

1. **Create a string str** with "JUnit Testing".
2. **Use assertNotNull(str);** to check that str is not null.
3. **Check if it starts with "JUnit"** using assertTrue(str.startsWith("JUnit"));.
4. **Verify string length** using assertEquals(14, str.length());.
5. **Check case-insensitive equality** using assertEquals("junit testing", str.toLowerCase());.
6. **Ensure the string does not contain "ABC"** with assertFalse(str.contains("ABC"));.

# 📌 **2. Using** @BeforeEach **and** @AfterEach

### ✅ ****Example: Setting Up Before and After Each Test****

java

CopyEdit

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

class LifecycleTest {

@BeforeEach

void setup() {

System.out.println("Setup before each test");

}

@Test

void testMethod1() {

System.out.println("Executing Test 1");

}

@Test

void testMethod2() {

System.out.println("Executing Test 2");

}

@AfterEach

void tearDown() {

System.out.println("Cleanup after each test");

}

}

### ✅ ****Step-by-Step Explanation****

1. **@BeforeEach setup()** runs **before each test** and prints "Setup before each test".
2. **Two test methods** (testMethod1() and testMethod2()) run separately.
3. **@AfterEach tearDown()** runs **after each test** and prints "Cleanup after each test".

📌 **Output Order:**

bash

CopyEdit

Setup before each test

Executing Test 1

Cleanup after each test

Setup before each test

Executing Test 2

Cleanup after each test

# 📌 **3. Running Tests with Different Inputs (Parameterized Tests)**

### ✅ ****Example: Checking Even Numbers****

java

CopyEdit

import org.junit.jupiter.params.ParameterizedTest;

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

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

class ParameterizedExample {

@ParameterizedTest

@ValueSource(ints = {2, 4, 6, 8, 10})

void testEvenNumbers(int number) {

assertTrue(number % 2 == 0);

}

}

### ✅ ****Step-by-Step Explanation****

1. **@ParameterizedTest** makes it a parameterized test.
2. **@ValueSource(ints = {2, 4, 6, 8, 10})** provides test values.
3. **assertTrue(number % 2 == 0);** checks if each number is even.

📌 **Test Runs 5 Times with Different Values**

# 📌 **4. Running Multiple Test Classes in a Suite**

### ✅ ****Example: Creating a Test Suite****

java

CopyEdit

import org.junit.platform.suite.api.\*;

@Suite

@SelectClasses({ StringTest.class, LifecycleTest.class })

class TestSuiteExample { }

### ✅ ****Step-by-Step Explanation****

1. **@Suite** marks it as a test suite.
2. **@SelectClasses({StringTest.class, LifecycleTest.class})** runs both test classes together.

📌 **When you run TestSuiteExample, it will execute all tests from StringTest and LifecycleTest.**

# 📌 **5. Mocking Dependencies with Mockito**

### ✅ ****Example: Mocking a Service Class****

java

CopyEdit

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

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

class MockTest {

@Test

void testMocking() {

// Create a mock of the service class

MyService mockService = Mockito.mock(MyService.class);

// Define behavior for mock

Mockito.when(mockService.getData()).thenReturn("Mocked Data");

// Check if mock works as expected

assertEquals("Mocked Data", mockService.getData());

}

}

// Service class

class MyService {

String getData() {

return "Real Data";

}

}

### ✅ ****Step-by-Step Explanation****

1. **Mockito.mock(MyService.class);** creates a fake version of MyService.
2. **Mockito.when(mockService.getData()).thenReturn("Mocked Data");** sets expected behavior.
3. **Check if the mock returns "Mocked Data"** instead of "Real Data".

📌 **This is useful for testing when you don't want to call real database methods.**

# 📌 **6. Testing REST APIs with Rest Assured**

### ✅ ****Example: Testing a REST API GET Request****

java

CopyEdit

import io.restassured.RestAssured;

import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.\*;

import static org.hamcrest.Matchers.\*;

class RestApiTest {

@Test

void testGetRequest() {

RestAssured.baseURI = "https://jsonplaceholder.typicode.com";

given().

when().

get("/posts/1").

then().

statusCode(200).

body("id", equalTo(1));

}

}

### ✅ ****Step-by-Step Explanation****

1. **Set API Base URL** → RestAssured.baseURI = "https://jsonplaceholder.typicode.com";
2. **Make GET Request** → get("/posts/1")
3. **Verify Response Status is 200** → statusCode(200);
4. **Check Response Body Contains "id": 1** → body("id", equalTo(1));

📌 **This tests if the API correctly returns post #1.**

# 📌 **7. Checking Code Coverage with Jacoco**

### ✅ ****Example: Running Code Coverage****

1. **Add This to pom.xml**

xml

CopyEdit

<build>

<plugins>

<plugin>

<groupId>org.jacoco</groupId>

<artifactId>jacoco-maven-plugin</artifactId>

<version>0.8.7</version>

<executions>

<execution>

<goals>

<goal>prepare-agent</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

1. **Run the Following Commands**

sh

CopyEdit

mvn clean test

mvn jacoco:report

1. **Check Code Coverage Report**  
   Open:

bash

CopyEdit

target/site/jacoco/index.html

📌 **This will generate a report showing how much of your code is tested.**

# 🎯 **Conclusion**

We covered:  
✅ **More Assertion Examples**  
✅ **Lifecycle Methods (@BeforeEach, @AfterEach)**  
✅ **Parameterized Tests**  
✅ **Test Suites**  
✅ **Mockito for Mocking Services**  
✅ **Rest Assured for API Testing**  
✅ **Jacoco for Code Coverage**