# Complete JUnit Testing Guide with Examples

## Table of Contents

1. [Introduction to JUnit](#introduction-to-junit)
2. [Setting Up JUnit](#setting-up-junit)
3. [Writing Test Cases](#writing-test-cases)
4. [Assertions](#assertions)
5. [Test Fixtures](#test-fixtures)
6. [Parameterized Tests](#parameterized-tests)
7. [Test Suites](#test-suites)
8. [Mockito](#mockito)
9. [Rest Assured](#rest-assured)
10. [Code Coverage](#code-coverage)

## 1. Introduction to JUnit

JUnit is a popular testing framework for Java applications. It provides annotations and assertions to write and run tests effectively.

### Key Benefits:
- Easy to write and maintain tests
- Integration with IDEs and build tools
- Annotations for test lifecycle management
- Rich assertion library
- Support for parameterized and dynamic tests

## 2. Setting Up JUnit

### Maven Setup (pom.xml)
```xml
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.9.2</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-core</artifactId>
        <version>5.1.1</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>io.rest-assured</groupId>
        <artifactId>rest-assured</artifactId>
        <version>5.3.0</version>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>3.0.0-M9</version>
        </plugin>
        <plugin>
            <groupId>org.jacoco</groupId>
            <artifactId>jacoco-maven-plugin</artifactId>
            <version>0.8.8</version>
            <executions>
                <execution>
                    <goals>
                        <goal>prepare-agent</goal>
                    </goals>
                </execution>
                <execution>
                    <id>report</id>
                    <phase>test</phase>
                    <goals>
                        <goal>report</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

### Gradle Setup (build.gradle)
```gradle
dependencies {
    testImplementation 'org.junit.jupiter:junit-jupiter:5.9.2'
    testImplementation 'org.mockito:mockito-core:5.1.1'
    testImplementation 'io.rest-assured:rest-assured:5.3.0'
}

test {
    useJUnitPlatform()
}

apply plugin: 'jacoco'

jacoco {
    toolVersion = "0.8.8"
}
```

How to Run:
- Maven: `mvn test`
- Gradle: `./gradlew test`
- IDE: Right-click on test class and select "Run Tests"

## 3. Writing Test Cases

### Example 1: Basic Calculator Test

File: `src/main/java/Calculator.java`
```java
public 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;
    }
}
```

**File: `src/test/java/CalculatorTest.java`**
```java
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.DisplayName;
import static org.junit.jupiter.api.Assertions.*;

public class CalculatorTest {

    @Test
    @DisplayName("Test addition of two positive numbers")
    public void testAddition() {
        Calculator calculator = new Calculator();
        int result = calculator.add(5, 3);
        assertEquals(8, result, "5 + 3 should equal 8");
    }

    @Test
    @DisplayName("Test division by zero throws exception")
    public void testDivisionByZero() {
        Calculator calculator = new Calculator();
        assertThrows(IllegalArgumentException.class,
            () -> calculator.divide(10, 0),
            "Division by zero should throw IllegalArgumentException");
    }
}
```

### Example 2: String Utility Test

**File: `src/main/java/StringUtils.java`**
```java
public class StringUtils {
    public static boolean isPalindrome(String str) {
        if (str == null) return false;
        String cleaned = str.replaceAll("[^a-zA-Z0-9]", "").toLowerCase();
        return cleaned.equals(new StringBuilder(cleaned).reverse().toString());
    }

    public static String capitalizeWords(String str) {
        if (str == null || str.isEmpty()) return str;
        String[] words = str.split(" ");
~~~

```java
        StringBuilder result = new StringBuilder();
        for (String word : words) {
            if (!word.isEmpty()) {
                result.append(Character.toUpperCase(word.charAt(0)))
                      .append(word.substring(1).toLowerCase())
                      .append(" ");
            }
        }
        return result.toString().trim();
    }
}
```

File: `src/test/java/StringUtilsTest.java`
```java
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.DisplayName;
import static org.junit.jupiter.api.Assertions.*;

public class StringUtilsTest {

    @Test
    @DisplayName("Test palindrome detection")
    public void testIsPalindrome() {
        assertTrue(StringUtils.isPalindrome("A man a plan a canal Panama"));
        assertTrue(StringUtils.isPalindrome("racecar"));
        assertFalse(StringUtils.isPalindrome("hello"));
        assertFalse(StringUtils.isPalindrome(null));
    }

    @Test
    @DisplayName("Test word capitalization")
    public void testCapitalizeWords() {
        assertEquals("Hello World", StringUtils.capitalizeWords("hello world"));
        assertEquals("Java Programming", StringUtils.capitalizeWords("JAVA programming"));
        assertEquals("", StringUtils.capitalizeWords(""));
        assertNull(StringUtils.capitalizeWords(null));
    }
}
```

How to Run:
```bash
# Maven
mvn test -Dtest=CalculatorTest
mvn test -Dtest=StringUtilsTest

# Gradle
./gradlew test --tests CalculatorTest
./gradlew test --tests StringUtilsTest
```

## 4. Assertions

### Example 1: Basic Assertions

File: `src/test/java/BasicAssertionsTest.java`
```java
```

```java
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.DisplayName;
import static org.junit.jupiter.api.Assertions.*;
import java.time.Duration;
import java.util.Arrays;
import java.util.List;

public class BasicAssertionsTest {

    @Test
    @DisplayName("Test various assertion methods")
    public void testBasicAssertions() {
        // Equality assertions
        assertEquals(4, 2 + 2, "Simple addition should work");
        assertNotEquals(5, 2 + 2, "2 + 2 should not equal 5");

        // Boolean assertions
        assertTrue(5 > 3, "5 should be greater than 3");
        assertFalse(3 > 5, "3 should not be greater than 5");

        // Null assertions
        String nullString = null;
        String nonNullString = "Hello";
        assertNull(nullString, "String should be null");
        assertNotNull(nonNullString, "String should not be null");

        // Array assertions
        int[] expected = {1, 2, 3};
        int[] actual = {1, 2, 3};
        assertArrayEquals(expected, actual, "Arrays should be equal");

        // Collection assertions
        List<String> expectedList = Arrays.asList("apple", "banana");
        List<String> actualList = Arrays.asList("apple", "banana");
        assertEquals(expectedList, actualList, "Lists should be equal");
    }

    @Test
    @DisplayName("Test timeout assertion")
    public void testTimeout() {
        assertTimeout(Duration.ofMillis(100), () -> {
            Thread.sleep(50);
            return "Completed within timeout";
        }, "Operation should complete within 100ms");
    }
}
```

### Example 2: Advanced Assertions

File: `src/main/java/Person.java`
```java
import java.util.Objects;

public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() { return name; }
    public int getAge() { return age; }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Person person = (Person) obj;
        return age == person.age && Objects.equals(name, person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }

    @Override
    public String toString() {
        return "Person{name='" + name + "', age=" + age + "}";
    }
}
```

File: `src/test/java/AdvancedAssertionsTest.java`
```java
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.DisplayName;
import static org.junit.jupiter.api.Assertions.*;
import java.util.Arrays;
import java.util.List;

public class AdvancedAssertionsTest {

    @Test
    @DisplayName("Test assertAll for grouped assertions")
    public void testGroupedAssertions() {
        Person person = new Person("John Doe", 30);

        assertAll("Person properties",
            () -> assertEquals("John Doe", person.getName(), "Name should match"),
            () -> assertEquals(30, person.getAge(), "Age should match"),
            () -> assertNotNull(person.toString(), "ToString should not be null")
        );
    }

    @Test
    @DisplayName("Test exception assertions with message validation")
    public void testExceptionWithMessage() {
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            throw new IllegalArgumentException("Invalid input provided");
        });

        assertEquals("Invalid input provided", exception.getMessage());
        assertTrue(exception.getMessage().contains("Invalid"));
    }

    @Test
    @DisplayName("Test iterable assertions")
    public void testIterableAssertions() {
        List<String> fruits = Arrays.asList("apple", "banana", "cherry");

        assertAll("Fruit list validations",
            () -> assertEquals(3, fruits.size(), "Should have 3 fruits"),
            () -> assertTrue(fruits.contains("apple"), "Should contain apple"),
            () -> assertFalse(fruits.contains("orange"), "Should not contain orange"),
            () -> assertEquals("apple", fruits.get(0), "First fruit should be apple")
        );
    }
}
```

## 5. Test Fixtures

### Example 1: BeforeEach and AfterEach

File: `src/main/java/DatabaseConnection.java`
```java
public class DatabaseConnection {
    private boolean connected = false;
    private String connectionString;

    public void connect(String connectionString) {
        this.connectionString = connectionString;
        this.connected = true;
        System.out.println("Connected to: " + connectionString);
    }

    public void disconnect() {
        this.connected = false;
        this.connectionString = null;
        System.out.println("Disconnected from database");
    }

    public boolean isConnected() {
        return connected;
    }

    public String executeQuery(String query) {
        if (!connected) {
            throw new IllegalStateException("Not connected to database");
        }
        return "Result for: " + query;
    }
}
```

File: `src/test/java/DatabaseConnectionTest.java`
```java
import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;

public class DatabaseConnectionTest {

    private DatabaseConnection dbConnection;
    @BeforeEach
    @DisplayName("Set up database connection before each test")
    public void setUp() {
        System.out.println("Setting up test...");
        dbConnection = new DatabaseConnection();
        dbConnection.connect("jdbc:h2:mem:testdb");
    }

    @AfterEach
    @DisplayName("Clean up after each test")
    public void tearDown() {
        System.out.println("Cleaning up test...");
        if (dbConnection != null && dbConnection.isConnected()) {
            dbConnection.disconnect();
        }
    }

    @Test
    @DisplayName("Test database connection is established")
    public void testConnectionEstablished() {
        assertTrue(dbConnection.isConnected(), "Database should be connected");
    }

    @Test
    @DisplayName("Test query execution")
    public void testQueryExecution() {
        String result = dbConnection.executeQuery("SELECT * FROM users");
        assertEquals("Result for: SELECT * FROM users", result);
    }

    @Test
    @DisplayName("Test query execution when not connected throws exception")
    public void testQueryExecutionWhenDisconnected() {
        dbConnection.disconnect();

        assertThrows(IllegalStateException.class,
            () -> dbConnection.executeQuery("SELECT * FROM users"),
            "Should throw exception when not connected");
    }
}
```

### Example 2: BeforeAll and AfterAll

File: `src/main/java/FileManager.java`
```java
import java.io.*;
import java.nio.file.*;
import java.util.List;

public class FileManager {
    public void writeToFile(String fileName, String content) throws IOException {
        Files.write(Paths.get(fileName), content.getBytes());
    }

    public String readFromFile(String fileName) throws IOException {
        return new String(Files.readAllBytes(Paths.get(fileName)));
    }

    public boolean fileExists(String fileName) {
        return Files.exists(Paths.get(fileName));
    }

    public void deleteFile(String fileName) throws IOException {
        Files.deleteIfExists(Paths.get(fileName));
    }
}
```

File: `src/test/java/FileManagerTest.java`
```java
import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class FileManagerTest {

    private static final String TEST_DIR = "test-files";
    private static final String TEST_FILE = TEST_DIR + "/test.txt";
    private FileManager fileManager;

    @BeforeAll
    @DisplayName("Create test directory")
    public static void setUpClass() throws IOException {
        System.out.println("Creating test directory...");
        Files.createDirectories(Paths.get(TEST_DIR));
    }

    @AfterAll
    @DisplayName("Clean up test directory")
    public static void tearDownClass() throws IOException {
        System.out.println("Cleaning up test directory...");
        Files.deleteIfExists(Paths.get(TEST_FILE));
        Files.deleteIfExists(Paths.get(TEST_DIR));
    }

    @BeforeEach
    public void setUp() {
        fileManager = new FileManager();
    }

    @Test
    @DisplayName("Test file creation and reading")
    public void testFileCreationAndReading() throws IOException {
        String content = "Hello, JUnit 5!";

        fileManager.writeToFile(TEST_FILE, content);
        assertTrue(fileManager.fileExists(TEST_FILE), "File should exist after writing");

        String readContent = fileManager.readFromFile(TEST_FILE);
        assertEquals(content, readContent, "Read content should match written content");
    }

    @Test
    @DisplayName("Test file deletion")
    public void testFileDeletion() throws IOException {
        String content = "Test content for deletion";
        fileManager.writeToFile(TEST_FILE, content);
        assertTrue(fileManager.fileExists(TEST_FILE), "File should exist before deletion");

        fileManager.deleteFile(TEST_FILE);
        assertFalse(fileManager.fileExists(TEST_FILE), "File should not exist after deletion");
    }
}
```

## 6. Parameterized Tests

### Example 1: ValueSource and CsvSource

File: `src/main/java/MathUtils.java`
```java
public class MathUtils {
    public static boolean isPrime(int number) {
        if (number <= 1) return false;
        if (number <= 3) return true;
        if (number % 2 == 0 || number % 3 == 0) return false;

        for (int i = 5; i * i <= number; i += 6) {
            if (number % i == 0 || number % (i + 2) == 0) {
                return false;
            }
        }
        return true;
    }

    public static int factorial(int n) {
        if (n < 0) throw new IllegalArgumentException("Factorial is not defined for negative numbers");
        if (n == 0 || n == 1) return 1;
        return n * factorial(n - 1);
    }

    public static double calculateBMI(double weight, double height) {
        if (weight <= 0 || height <= 0) {
            throw new IllegalArgumentException("Weight and height must be positive");
        }
        return weight / (height * height);
    }
}
```

File: `src/test/java/ParameterizedMathUtilsTest.java`
```java
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.*;
import static org.junit.jupiter.api.Assertions.*;

public class ParameterizedMathUtilsTest {

    @ParameterizedTest(name = "Test isPrime with value: {0}")
    @DisplayName("Test prime number detection")
    @ValueSource(ints = {2, 3, 5, 7, 11, 13, 17, 19, 23})
    public void testIsPrimeWithPrimes(int number) {
        assertTrue(MathUtils.isPrime(number), number + " should be prime");
    }

    @ParameterizedTest(name = "Test non-prime with value: {0}")
    @DisplayName("Test non-prime number detection")
    @ValueSource(ints = {1, 4, 6, 8, 9, 10, 12, 14, 15})
    public void testIsPrimeWithNonPrimes(int number) {
        assertFalse(MathUtils.isPrime(number), number + " should not be prime");
    }

    @ParameterizedTest(name = "Test factorial: {0}! = {1}")
    @DisplayName("Test factorial calculation")
    @CsvSource({
        "0, 1",
        "1, 1",
        "2, 2",
        "3, 6",
        "4, 24",
        "5, 120"
    })
    public void testFactorial(int input, int expected) {
        assertEquals(expected, MathUtils.factorial(input),
            "Factorial of " + input + " should be " + expected);
    }
}
```

### Example 2: MethodSource and Custom ArgumentsProvider

File: `src/test/java/AdvancedParameterizedTest.java`
```java
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.*;
import static org.junit.jupiter.api.Assertions.*;
import java.util.stream.Stream;
import org.junit.jupiter.api.extension.ExtensionContext;
import java.util.stream.Arguments;

public class AdvancedParameterizedTest {

    @ParameterizedTest(name = "BMI calculation: weight={0}kg, height={1}m, expected={2}")
    @DisplayName("Test BMI calculation with method source")
    @MethodSource("provideBMIData")
    public void testBMICalculation(double weight, double height, double expected) {
        double actual = MathUtils.calculateBMI(weight, height);
        assertEquals(expected, actual, 0.01,
            "BMI calculation should be accurate within 0.01");
    }

    static Stream<Arguments> provideBMIData() {
        return Stream.of(
            Arguments.of(70.0, 1.75, 22.86),
            Arguments.of(80.0, 1.80, 24.69),
            Arguments.of(60.0, 1.65, 22.04),
            Arguments.of(90.0, 1.85, 26.30)
        );
    }

    @ParameterizedTest(name = "String validation: {0}")
    @DisplayName("Test string validation with enum source")
    @EnumSource(StringTestCase.class)
    public void testStringValidation(StringTestCase testCase) {
        assertEquals(testCase.expectedResult,
                     StringUtils.isPalindrome(testCase.input),
                     "Palindrome test for: " + testCase.input);
    }

    enum StringTestCase {
        SIMPLE_PALINDROME("racecar", true),
        PHRASE_PALINDROME("A man a plan a canal Panama", true),
        NOT_PALINDROME("hello", false),
        EMPTY_STRING("", true),
        SINGLE_CHAR("a", true);

        final String input;
        final boolean expectedResult;

        StringTestCase(String input, boolean expectedResult) {
            this.input = input;
            this.expectedResult = expectedResult;
        }
    }
}
```

How to Run:
```bash
# Maven - run all parameterized tests
mvn test -Dtest=ParameterizedMathUtilsTest
mvn test -Dtest=AdvancedParameterizedTest

# Gradle
./gradlew test --tests ParameterizedMathUtilsTest
./gradlew test --tests AdvancedParameterizedTest
```

## 7. Test Suites

### Example 1: Basic Test Suite

File: `src/test/java/AllCalculatorTests.java`
```java
import org.junit.platform.suite.api.SelectClasses;
import org.junit.platform.suite.api.Suite;
import org.junit.platform.suite.api.SuiteDisplayName;

@Suite
@SuiteDisplayName("All Calculator Related Tests")
@SelectClasses({
    CalculatorTest.class,
    ParameterizedMathUtilsTest.class,
    AdvancedParameterizedTest.class
})
public class AllCalculatorTests {
    // This class remains empty, it is used only as a holder for the above annotations
}
```

### Example 2: Package-based Test Suite

File: `src/test/java/AllUtilityTests.java`
```java
import org.junit.platform.suite.api.*;

@Suite
@SuiteDisplayName("All Utility Classes Test Suite")
@SelectPackages({
    "com.example.utils",
    "com.example.math"
})
@IncludeClassNamePatterns(".*Test.*")
@ExcludeClassNamePatterns(".*IntegrationTest.*")
public class AllUtilityTests {
    // Empty class used as test suite holder
}
```

File: `src/test/java/QuickTestSuite.java`
```java
import org.junit.platform.suite.api.*;

@Suite
@SuiteDisplayName("Quick Test Suite")
@SelectClasses({
    BasicAssertionsTest.class,
    StringUtilsTest.class
})
@IncludeTags("quick")
@ExcludeTags("slow")
public class QuickTestSuite {
    // Test suite for quick running tests
}
```

How to Run Test Suites:
```bash
# Maven
mvn test -Dtest=AllCalculatorTests
mvn test -Dtest=AllUtilityTests

# Gradle
./gradlew test --tests AllCalculatorTests
./gradlew test --tests AllUtilityTests
```

## 8. Mockito

### Example 1: Basic Mocking

File: `src/main/java/UserService.java`
```java
import java.util.List;

public class UserService {
    private UserRepository userRepository;
    private EmailService emailService;

    public UserService(UserRepository userRepository, EmailService emailService) {
        this.userRepository = userRepository;
        this.emailService = emailService;
    }

    public User createUser(String name, String email) {
        if (userRepository.existsByEmail(email)) {
            throw new IllegalArgumentException("User with email already exists");
        }

        User user = new User(name, email);
        User savedUser = userRepository.save(user);
        emailService.sendWelcomeEmail(savedUser.getEmail());
        return savedUser;
    }

    public List<User> getAllActiveUsers() {
        return userRepository.findByStatus("ACTIVE");
    }

    public void deactivateUser(Long userId) {
        User user = userRepository.findById(userId);
        if (user == null) {
            throw new IllegalArgumentException("User not found");
        }
        user.setStatus("INACTIVE");
        userRepository.save(user);
        emailService.sendDeactivationEmail(user.getEmail());
    }
}

class User {
    private Long id;
    private String name;
    private String email;
    private String status;

    public User(String name, String email) {
        this.name = name;
        this.email = email;
        this.status = "ACTIVE";
    }

    // 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; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public String getStatus() { return status; }
    public void setStatus(String status) { this.status = status; }
}

interface UserRepository {
    User save(User user);
    User findById(Long id);
    List<User> findByStatus(String status);
    boolean existsByEmail(String email);
}

interface EmailService {
    void sendWelcomeEmail(String email);
    void sendDeactivationEmail(String email);
}
```

File: `src/test/java/UserServiceMockitoTest.java`
```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;
import java.util.Arrays;
import java.util.List;

@ExtendWith(MockitoExtension.class)
public class UserServiceMockitoTest {

    @Mock
    private UserRepository userRepository;

    @Mock
    private EmailService emailService;

    private UserService userService;

    @BeforeEach
    public void setUp() {
        userService = new UserService(userRepository, emailService);
    }

    @Test
    @DisplayName("Test successful user creation")
    public void testCreateUserSuccess() {
        // Arrange
        String name = "John Doe";
        String email = "john@example.com";
        User expectedUser = new User(name, email);
        expectedUser.setId(1L);

        when(userRepository.existsByEmail(email)).thenReturn(false);
        when(userRepository.save(any(User.class))).thenReturn(expectedUser);

        // Act
        User createdUser = userService.createUser(name, email);

        // Assert
        assertNotNull(createdUser);
        assertEquals(name, createdUser.getName());
        assertEquals(email, createdUser.getEmail());
        assertEquals(1L, createdUser.getId());

        // Verify interactions
        verify(userRepository).existsByEmail(email);
        verify(userRepository).save(any(User.class));
        verify(emailService).sendWelcomeEmail(email);
    }

    @Test
    @DisplayName("Test user creation fails when email already exists")
    public void testCreateUserEmailAlreadyExists() {
        // Arrange
        String name = "John Doe";
        String email = "existing@example.com";

        when(userRepository.existsByEmail(email)).thenReturn(true);

        // Act & Assert
        assertThrows(IllegalArgumentException.class,
            () -> userService.createUser(name, email),
            "Should throw exception when email already exists");

        // Verify that save and email service were never called
        verify(userRepository, never()).save(any(User.class));
        verify(emailService, never()).sendWelcomeEmail(anyString());
    }
}
```

### Example 2: Advanced Mocking with Argument Captors and Spies

File: `src/test/java/AdvancedMockitoTest.java`
```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.*;
import org.mockito.junit.jupiter.MockitoExtension;
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;
import java.util.Arrays;
import java.util.List;

@ExtendWith(MockitoExtension.class)
public class AdvancedMockitoTest {

    @Mock
    private UserRepository userRepository;

    @Mock
    private EmailService emailService;

    @Spy
    private UserService userService;

    @Captor
    private ArgumentCaptor<String> stringCaptor;

    @BeforeEach
    public void setUp() {
        userService = new UserService(userRepository, emailService);
    }
```

```
@Test
@DisplayName("Test get all active users with argument captor")
public void testGetAllActiveUsersWithCaptor() {
    // Arrange
    List<User> expectedUsers = Arrays.asList(
        new User("John", "john@example.com"),
        new User("Jane", "jane@example.com")
    );

    when(userRepository.findByStatus(anyString())).thenReturn(expectedUsers);

    // Act
    List<User> actualUsers = userService.getAllActiveUsers();

    // Assert
    assertEquals(2, actualUsers.size());

    // Capture and verify the argument passed to findByStatus
    verify(userRepository).findByStatus(stringCaptor.capture());
    assertEquals("ACTIVE", stringCaptor.getValue());
}

@Test
@DisplayName("Test user deactivation with multiple verifications")
public void testDeactivateUserWithMultipleVerifications() {
    // Arrange
    Long userId = 1L;
    User user = new User("John Doe", "john@example.com");
    user.setId(userId);
    user.setStatus("ACTIVE");

    when(userRepository.findById(userId)).thenReturn(user);
    when(userRepository.save(any(User.class))).thenReturn(user);

    // Act
    userService.deactivateUser(userId);

    // Assert and verify
    verify(userRepository).findById(userId);
    verify(userRepository).save(userCaptor.capture());
    verify(emailService).sendDeactivationEmail(stringCaptor.capture());

    // Verify the captured user has correct status
    User capturedUser = userCaptor.getValue();
    assertEquals("INACTIVE", capturedUser.getStatus());

    // Verify the correct email was captured
    String capturedEmail = stringCaptor.getValue();
    assertEquals("john@example.com", capturedEmail);
}

@Test
@DisplayName("Test deactivate user throws exception when user not found")
public void testDeactivateUserNotFound() {
    // Arrange
    Long userId = 999L;
    when(userRepository.findById(userId)).thenReturn(null);

    // Act & Assert
```

```
        assertThrows(IllegalArgumentException.class,
            () -> userService.deactivateUser(userId),
            "Should throw exception when user not found");

        // Verify that save and email service were never called
        verify(userRepository, never()).save(any(User.class));
        verify(emailService, never()).sendDeactivationEmail(anyString());
    }

    @Test
    @DisplayName("Test with custom answer and doAnswer")
    public void testWithCustomAnswer() {
        // Arrange - Custom behavior for save method
        doAnswer(invocation -> {
            User user = invocation.getArgument(0);
            user.setId(100L); // Simulate auto-generated ID
            return user;
        }).when(userRepository).save(any(User.class));

        when(userRepository.existsByEmail(anyString())).thenReturn(false);

        // Act
        User createdUser = userService.createUser("Test User", "test@example.com");

        // Assert
        assertEquals(100L, createdUser.getId());
        assertEquals("Test User", createdUser.getName());
    }
}
```

How to Run Mockito Tests:
```bash
# Maven
mvn test -Dtest=UserServiceMockitoTest
mvn test -Dtest=AdvancedMockitoTest

# Gradle
./gradlew test --tests UserServiceMockitoTest
./gradlew test --tests AdvancedMockitoTest
```

## 9. Rest Assured

### Example 1: Basic REST API Testing

File: `src/test/java/BasicRestApiTest.java`
```java
import io.restassured.RestAssured;
import io.restassured.response.Response;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.DisplayName;
import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;
import static org.junit.jupiter.api.Assertions.*;

public class BasicRestApiTest {
```

```
@BeforeAll
public static void setUp() {
    // Base URI for JSONPlaceholder API (free testing API)
    RestAssured.baseURI = "https://jsonplaceholder.typicode.com";
    RestAssured.enableLoggingOfRequestAndResponseIfValidationFails();
}

@Test
@DisplayName("Test GET request - Fetch all posts")
public void testGetAllPosts() {
    given()
        .when()
            .get("/posts")
        .then()
            .statusCode(200)
            .body("size()", greaterThan(0))
            .body("[0].userId", notNullValue())
            .body("[0].title", notNullValue())
            .time(lessThan(5000L)); // Response time less than 5 seconds
}

@Test
@DisplayName("Test GET request - Fetch specific post")
public void testGetSpecificPost() {
    int postId = 1;

    Response response = given()
        .pathParam("postId", postId)
        .when()
            .get("/posts/{postId}")
        .then()
            .statusCode(200)
            .body("id", equalTo(postId))
            .body("userId", equalTo(1))
            .body("title", notNullValue())
            .body("body", notNullValue())
            .extract()
            .response();

    // Additional assertions using response object
    assertEquals(postId, response.jsonPath().getInt("id"));
    assertTrue(response.jsonPath().getString("title").length() > 0);
}

@Test
@DisplayName("Test POST request - Create new post")
public void testCreatePost() {
    String requestBody = """
        {
            "title": "JUnit 5 Testing",
            "body": "This is a test post created using Rest Assured",
            "userId": 1
        }
        """;

    given()
        .header("Content-Type", "application/json")
        .body(requestBody)
        .when()
```

```
                .post("/posts")
            .then()
                .statusCode(201)
                .body("title", equalTo("JUnit 5 Testing"))
                .body("body", containsString("Rest Assured"))
                .body("userId", equalTo(1))
                .body("id", notNullValue());
    }

    @Test
    @DisplayName("Test PUT request - Update existing post")
    public void testUpdatePost() {
        int postId = 1;
        String updateBody = """
            {
                "id": 1,
                "title": "Updated Title",
                "body": "Updated body content",
                "userId": 1
            }
            """;

        given()
            .header("Content-Type", "application/json")
            .pathParam("postId", postId)
            .body(updateBody)
            .when()
                .put("/posts/{postId}")
            .then()
                .statusCode(200)
                .body("title", equalTo("Updated Title"))
                .body("body", equalTo("Updated body content"))
                .body("id", equalTo(postId));
    }
}
```

### Example 2: Advanced REST API Testing

File: `src/test/java/AdvancedRestApiTest.java`
```java
import io.restassured.RestAssured;
import io.restassured.response.Response;
import io.restassured.specification.RequestSpecification;
import io.restassured.specification.ResponseSpecification;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.DisplayName;
import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;
import java.util.List;
import java.util.Map;

public class AdvancedRestApiTest {

    private static RequestSpecification requestSpec;
    private static ResponseSpecification responseSpec;

    @BeforeAll
    public static void setUp() {
        RestAssured.baseURI = "https://jsonplaceholder.typicode.com";

        // Request specification
        requestSpec = given()
            .header("Content-Type", "application/json")
            .header("Accept", "application/json");

        // Response specification
        responseSpec = expect()
            .statusCode(200)
            .time(lessThan(5000L));
    }

    @Test
    @DisplayName("Test with query parameters and response validation")
    public void testGetPostsWithQueryParams() {
        given(requestSpec)
            .queryParam("userId", 1)
            .when()
                .get("/posts")
            .then(responseSpec)
                .body("size()", greaterThan(0))
                .body("findAll { it.userId == 1 }.size()", greaterThan(0))
                .body("userId", everyItem(equalTo(1)));
    }

    @Test
    @DisplayName("Test response extraction and complex validation")
    public void testResponseExtractionAndValidation() {
        Response response = given(requestSpec)
            .when()
                .get("/users")
            .then()
                .statusCode(200)
                .extract()
                .response();

        // Extract specific data from response
        List<Map<String, Object>> users = response.jsonPath().getList("$");
        assertEquals(10, users.size(), "Should have 10 users");

        // Validate first user structure
        Map<String, Object> firstUser = users.get(0);
        assertTrue(firstUser.containsKey("id"), "User should have id field");
        assertTrue(firstUser.containsKey("name"), "User should have name field");
        assertTrue(firstUser.containsKey("email"), "User should have email field");

        // Extract and validate email format
        String email = response.jsonPath().getString("[0].email");
        assertTrue(email.contains("@"), "Email should contain @ symbol");

        // Extract nested data (address)
        String city = response.jsonPath().getString("[0].address.city");
        assertNotNull(city, "City should not be null");

        // Extract list of all names
        List<String> names = response.jsonPath().getList("name");
        assertEquals(10, names.size(), "Should extract 10 names");
    }
    @Test
    @DisplayName("Test DELETE request and error handling")
    public void testDeletePost() {
        int postId = 1;

        // Delete the post
        given(requestSpec)
            .pathParam("postId", postId)
            .when()
                .delete("/posts/{postId}")
            .then()
                .statusCode(200);

        // Verify post still exists (JSONPlaceholder doesn't actually delete)
        given(requestSpec)
            .pathParam("postId", postId)
            .when()
                .get("/posts/{postId}")
            .then()
                .statusCode(200)
                .body("id", equalTo(postId));
    }

    @Test
    @DisplayName("Test error scenarios and status codes")
    public void testErrorScenarios() {
        // Test 404 - Not Found
        given(requestSpec)
            .pathParam("postId", 999)
            .when()
                .get("/posts/{postId}")
            .then()
                .statusCode(404);

        // Test invalid JSON in POST request
        given()
            .header("Content-Type", "application/json")
            .body("invalid json")
            .when()
                .post("/posts")
            .then()
                .statusCode(anyOf(equalTo(400), equalTo(500))); // Could be 400 or 500 depending on API
    }

    @Test
    @DisplayName("Test JSON schema validation")
    public void testJsonSchemaValidation() {
        given(requestSpec)
            .when()
                .get("/posts/1")
            .then()
                .statusCode(200)
                .body(matchesJsonSchemaInClasspath("post-schema.json"));
    }
}
```

File: `src/test/resources/post-schema.json`
```json
```

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "required": ["userId", "id", "title", "body"],
  "properties": {
    "userId": {
      "type": "integer"
    },
    "id": {
      "type": "integer"
    },
    "title": {
      "type": "string"
    },
    "body": {
      "type": "string"
    }
  }
}
```

How to Run REST API Tests:
```bash
# Maven
mvn test -Dtest=BasicRestApiTest
mvn test -Dtest=AdvancedRestApiTest

# Gradle
./gradlew test --tests BasicRestApiTest
./gradlew test --tests AdvancedRestApiTest
```

## 10. Code Coverage

### Example 1: JaCoCo Configuration and Basic Coverage

File: `src/main/java/ShoppingCart.java`
```java
import java.util.ArrayList;
import java.util.List;

public class ShoppingCart {
    private List<Item> items;
    private double discountPercentage;

    public ShoppingCart() {
        this.items = new ArrayList<>();
        this.discountPercentage = 0.0;
    }

    public void addItem(Item item) {
        if (item == null) {
            throw new IllegalArgumentException("Item cannot be null");
        }
        items.add(item);
    }

    public void removeItem(Item item) {
        items.remove(item);
    }

    public double calculateTotal() {
        double total = items.stream()
            .mapToDouble(item -> item.getPrice() * item.getQuantity())
            .sum();

        if (discountPercentage > 0) {
            total = total * (1 - discountPercentage / 100);
        }

        return total;
    }

    public int getItemCount() {
        return items.size();
    }

    public void applyDiscount(double percentage) {
        if (percentage < 0 || percentage > 100) {
            throw new IllegalArgumentException("Discount percentage must be between 0 and 100");
        }
        this.discountPercentage = percentage;
    }

    public boolean isEmpty() {
        return items.isEmpty();
    }

    public void clearCart() {
        items.clear();
        discountPercentage = 0.0;
    }

    // Getters
    public List<Item> getItems() { return new ArrayList<>(items); }
    public double getDiscountPercentage() { return discountPercentage; }
}

class Item {
    private String name;
    private double price;
    private int quantity;

    public Item(String name, double price, int quantity) {
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }

    // Getters and setters
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public double getPrice() { return price; }
    public void setPrice(double price) { this.price = price; }
    public int getQuantity() { return quantity; }
    public void setQuantity(int quantity) { this.quantity = quantity; }

    @Override
```

```java
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Item item = (Item) obj;
        return Double.compare(item.price, price) == 0 &&
               quantity == item.quantity &&
               name.equals(item.name);
    }
}
```

File: `src/test/java/ShoppingCartCoverageTest.java`

```java
import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;

public class ShoppingCartCoverageTest {

    private ShoppingCart cart;

    @BeforeEach
    public void setUp() {
        cart = new ShoppingCart();
    }

    @Test
    @DisplayName("Test add item functionality")
    public void testAddItem() {
        Item item = new Item("Apple", 1.50, 3);
        cart.addItem(item);

        assertEquals(1, cart.getItemCount());
        assertFalse(cart.isEmpty());
        assertTrue(cart.getItems().contains(item));
    }

    @Test
    @DisplayName("Test add null item throws exception")
    public void testAddNullItem() {
        assertThrows(IllegalArgumentException.class,
            () -> cart.addItem(null),
            "Adding null item should throw exception");
    }

    @Test
    @DisplayName("Test remove item functionality")
    public void testRemoveItem() {
        Item item = new Item("Banana", 0.75, 2);
        cart.addItem(item);
        assertEquals(1, cart.getItemCount());

        cart.removeItem(item);
        assertEquals(0, cart.getItemCount());
        assertTrue(cart.isEmpty());
    }

    @Test
    @DisplayName("Test calculate total without discount")
    public void testCalculateTotalWithoutDiscount() {
        cart.addItem(new Item("Apple", 1.50, 2)); // 3.00
        cart.addItem(new Item("Banana", 0.75, 3)); // 2.25

        assertEquals(5.25, cart.calculateTotal(), 0.01);
    }

    @Test
    @DisplayName("Test calculate total with discount")
    public void testCalculateTotalWithDiscount() {
        cart.addItem(new Item("Apple", 2.00, 5)); // 10.00
        cart.applyDiscount(20); // 20% discount

        assertEquals(8.00, cart.calculateTotal(), 0.01);
        assertEquals(20.0, cart.getDiscountPercentage());
    }

    @Test
    @DisplayName("Test apply invalid discount throws exception")
    public void testApplyInvalidDiscount() {
        assertAll("Invalid discount scenarios",
            () -> assertThrows(IllegalArgumentException.class,
                () -> cart.applyDiscount(-5),
                "Negative discount should throw exception"),
            () -> assertThrows(IllegalArgumentException.class,
                () -> cart.applyDiscount(105),
                "Discount over 100% should throw exception")
        );
    }

    @Test
    @DisplayName("Test clear cart functionality")
    public void testClearCart() {
        cart.addItem(new Item("Orange", 1.25, 4));
        cart.applyDiscount(15);

        assertFalse(cart.isEmpty());
        assertEquals(15.0, cart.getDiscountPercentage());

        cart.clearCart();

        assertTrue(cart.isEmpty());
        assertEquals(0, cart.getItemCount());
        assertEquals(0.0, cart.getDiscountPercentage());
    }

    @Test
    @DisplayName("Test empty cart total")
    public void testEmptyCartTotal() {
        assertEquals(0.0, cart.calculateTotal());
        assertTrue(cart.isEmpty());
    }
}
```

### Example 2: Coverage Analysis and Reporting

File: `src/test/java/ComprehensiveCoverageTest.java`
```java
import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;
```

```
public class ComprehensiveCoverageTest {

    @Test
    @DisplayName("Test Item class coverage")
    public void testItemClassCoverage() {
        // Test constructor and getters
        Item item = new Item("Test Item", 5.99, 2);
        assertEquals("Test Item", item.getName());
        assertEquals(5.99, item.getPrice());
        assertEquals(2, item.getQuantity());

        // Test setters
        item.setName("Updated Item");
        item.setPrice(7.99);
        item.setQuantity(3);

        assertEquals("Updated Item", item.getName());
        assertEquals(7.99, item.getPrice());
        assertEquals(3, item.getQuantity());

        // Test equals method
        Item item1 = new Item("Test", 1.0, 1);
        Item item2 = new Item("Test", 1.0, 1);
        Item item3 = new Item("Different", 1.0, 1);

        assertTrue(item1.equals(item2), "Identical items should be equal");
        assertFalse(item1.equals(item3), "Different items should not be equal");
        assertFalse(item1.equals(null), "Item should not equal null");
        assertFalse(item1.equals("String"), "Item should not equal different type");
        assertTrue(item1.equals(item1), "Item should equal itself");
    }

    @Test
    @DisplayName("Test edge cases for full coverage")
    public void testEdgeCases() {
        ShoppingCart cart = new ShoppingCart();

        // Test with zero price items
        Item freeItem = new Item("Free Sample", 0.0, 1);
        cart.addItem(freeItem);
        assertEquals(0.0, cart.calculateTotal());

        // Test with zero quantity items
        Item zeroQuantityItem = new Item("Zero Quantity", 5.0, 0);
        cart.addItem(zeroQuantityItem);
        assertEquals(0.0, cart.calculateTotal());

        // Test discount edge cases
        cart.clearCart();
        cart.addItem(new Item("Test", 10.0, 1));

        // Test 0% discount
        cart.applyDiscount(0);
        assertEquals(10.0, cart.calculateTotal());

        // Test 100% discount
        cart.applyDiscount(100);
        assertEquals(0.0, cart.calculateTotal());

        // Test boundary values for discount
        cart.applyDiscount(0.01); // Just above 0
        cart.applyDiscount(99.99); // Just below 100

        // No assertions needed, just ensuring branches are covered
    }

    @Test
    @DisplayName("Test collections and defensive copying")
    public void testCollectionsHandling() {
        ShoppingCart cart = new ShoppingCart();
        Item item1 = new Item("Item1", 5.0, 1);
        Item item2 = new Item("Item2", 3.0, 2);

        cart.addItem(item1);
        cart.addItem(item2);

        // Test that getItems returns a defensive copy
        List<Item> items = cart.getItems();
        items.clear(); // This should not affect the original cart

        assertEquals(2, cart.getItemCount(), "Original cart should be unaffected");

        // Test removing non-existent item
        Item nonExistentItem = new Item("Non-existent", 1.0, 1);
        cart.removeItem(nonExistentItem); // Should not throw exception
        assertEquals(2, cart.getItemCount(), "Item count should remain the same");
    }
}
```

File: `pom.xml` (Additional JaCoCo configuration for detailed reporting)
```xml
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.8</version>
    <executions>
        <execution>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
        <execution>
            <id>report</id>
            <phase>test</phase>
            <goals>
                <goal>report</goal>
            </goals>
        </execution>
        <execution>
            <id>check</id>
            <goals>
                <goal>check</goal>
            </goals>
            <configuration>
                <rules>
                    <rule>
                        <element>CLASS</element>
                        <limits>
                            <limit>
                <counter>LINE</counter>
                <value>COVEREDRATIO</value>
                <minimum>0.80</minimum>
              </limit>
              <limit>
                <counter>BRANCH</counter>
                <value>COVEREDRATIO</value>
                <minimum>0.75</minimum>
              </limit>
            </limits>
          </rule>
        </rules>
      </configuration>
    </execution>
  </executions>
</plugin>
```

How to Generate and View Coverage Reports:

### Maven Commands:
```bash
# Run tests and generate coverage report
mvn clean test

# Generate HTML coverage report
mvn jacoco:report

# Check coverage thresholds
mvn jacoco:check

# View report (will be generated in target/site/jacoco/index.html)
# Open target/site/jacoco/index.html in a web browser
```

### Gradle Commands:
```bash
# Run tests and generate coverage
./gradlew test jacocoTestReport

# Check coverage thresholds (if configured)
./gradlew jacocoTestCoverageVerification

# View report (will be in build/reports/jacoco/test/html/index.html)
```

### Coverage Report Interpretation:

The JaCoCo report will show:
- **Line Coverage**: Percentage of code lines executed
- **Branch Coverage**: Percentage of if/else branches tested
- **Cyclomatic Complexity**: Code complexity metrics
- **Method Coverage**: Percentage of methods executed
- **Class Coverage**: Percentage of classes tested

### File Structure:
```
project/
├── src/
│   ├── main/java/
│   │   ├── Calculator.java
│   │   ├── StringUtils.java
│   │   ├── DatabaseConnection.java
│   │   ├── FileManager.java
│   │   ├── MathUtils.java
│   │   ├── Person.java
│   │   ├── UserService.java
│   │   └── ShoppingCart.java
│   └── test/java/
│       ├── CalculatorTest.java
│       ├── StringUtilsTest.java
│       ├── BasicAssertionsTest.java
│       ├── AdvancedAssertionsTest.java
│       ├── DatabaseConnectionTest.java
│       ├── FileManagerTest.java
│       ├── ParameterizedMathUtilsTest.java
│       ├── AdvancedParameterizedTest.java
│       ├── AllCalculatorTests.java
│       ├── AllUtilityTests.java
│       ├── QuickTestSuite.java
│       ├── UserServiceMockitoTest.java
│       ├── AdvancedMockitoTest.java
│       ├── BasicRestApiTest.java
│       ├── AdvancedRestApiTest.java
│       ├── ShoppingCartCoverageTest.java
│       └── ComprehensiveCoverageTest.java
│   └── test/resources/
│       └── post-schema.json
├── target/ (Maven) or build/ (Gradle)
│   └── site/jacoco/ (coverage reports)
├── pom.xml (Maven) or build.gradle (Gradle)
└── README.md
```

This comprehensive guide covers all major aspects of JUnit testing with practical, executable examples that demonstrate real-world testing scenarios.tor<User> userCaptor;

```
    @Captor
    private ArgumentCap
```

Connecting the remote repository

```
$ mkdir GitRepo
$ cd GitRepo
```

Method 1:

```
GitRepo $ git clone https://github.com/Bhuvaneswari-tech/Material.git

GitRepo $ cd Material

Material $ git checkout -b feature/version1.0
Switched to a new branch 'feature/version1.0'

Material $ vi index.html
Material $ vi style.css

Material $ git add .

Material $ git commit -m “Initial Commit”
create mode

Material $ git push origin feature/version1.0
```

Method 2:

```
GitRepo $ git remote add origin https://github.com/Bhuvaneswari-tech/Material.git

Material $ git checkout -b feature/version1.0
```

✅ Task 1: Initialize a Git Repository

🧩 Scenario:

You just created a new project folder called `PortfolioWebsite`. You want to start tracking it with Git.

✅ Task 2: Create a New Branch for Feature Development

🧩 Scenario:

You’re adding a "Contact Us" form to your site. Instead of working on `main`, you want to isolate changes.

✅ Task 3: Commit Code with Message

🧩 Scenario:

You created the HTML form and want to save it with a descriptive message.

✅ Task 4: Pull Before Push to Avoid Conflicts

🧩 Scenario:

You and your teammate are working on the same branch. Before pushing, you want to ensure you have the latest code.

✅ Task 5: Resolve Merge Conflicts and Push

🧩 Scenario:

After pulling, you hit a conflict in `style.css` that you and your teammate edited. You fix it manually.