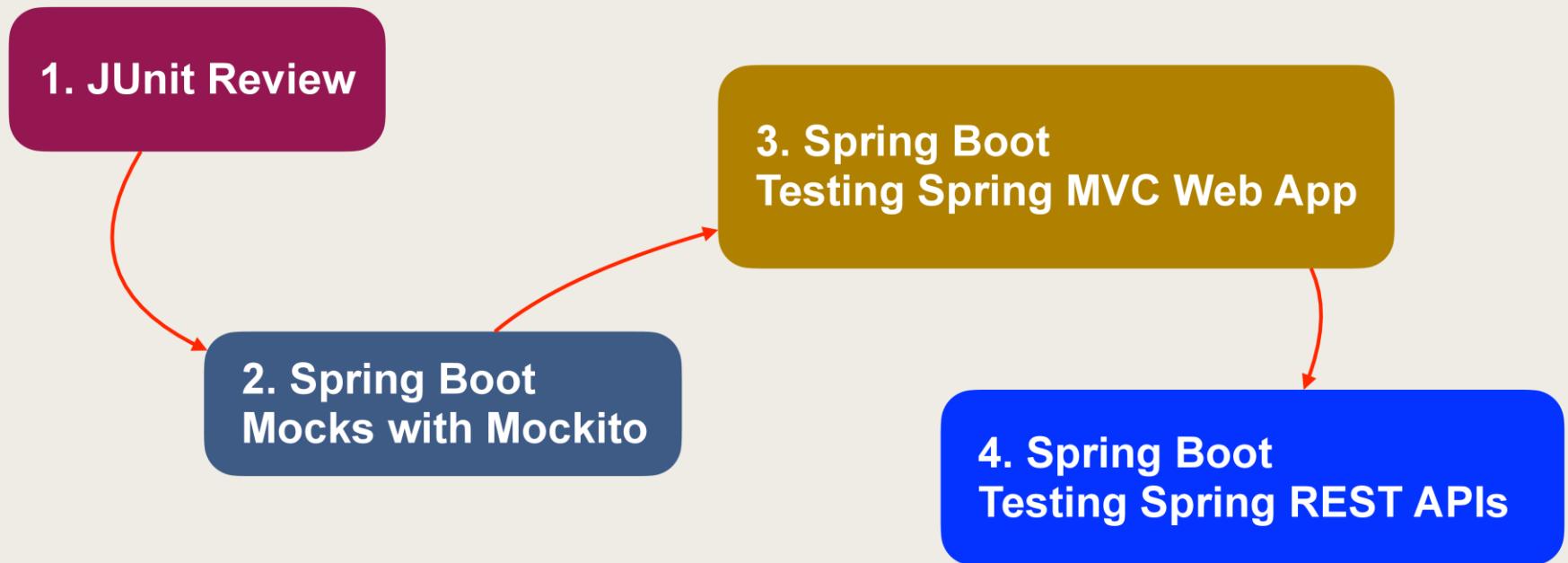


Spring Boot Unit Testing

Spring Boot Unit Testing

- Spring Boot supports developing unit tests and integration tests using JUnit and Mockito.
- By developing tests, you can create applications with better code design, fewer bugs, and higher reliability.
- This course shows you how to take advantage of Spring Boot's testing support.

Course Road Map



Prerequisites

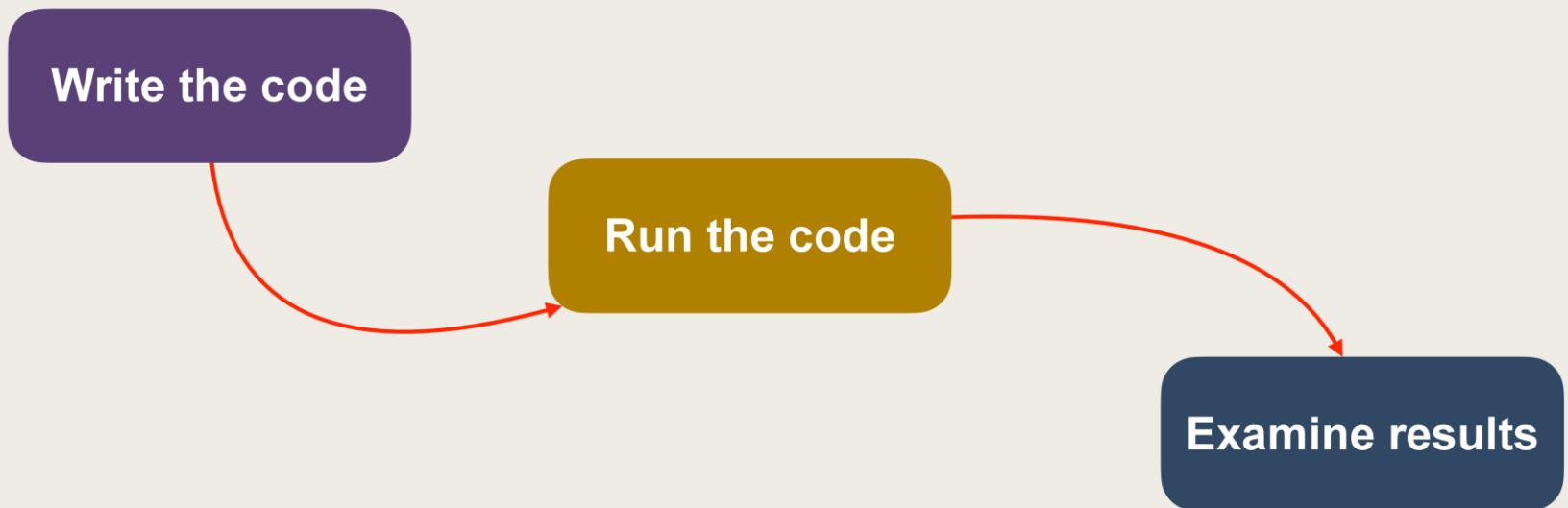
- You should have prior experience with
 - Java
 - Spring Boot
 - Maven

Spring & Hibernate for Beginners

Spring Boot - Testing

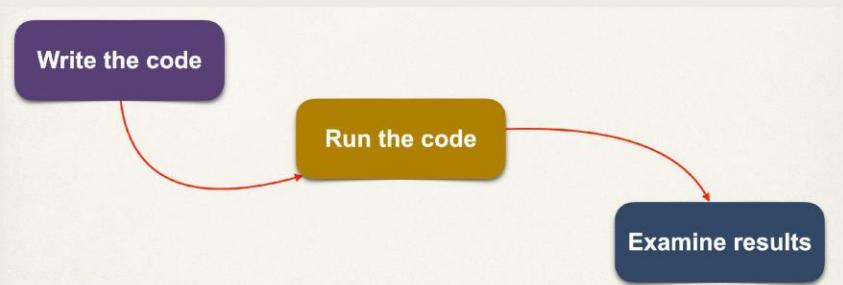
How Do We Test Code?

- Manual testing ... maybe this???



What's Wrong?

- Requires human interaction and analysis
- Test case is not reproducible
- No way to automate the test
- What if I wanted to
 - Run a battery of tests for each code commit
 - Produce a report of test results



What is Unit Testing?

- Testing an individual unit of code for correctness
- Provide fixed inputs
- Expect known output

Test Cases

`add(5, 2)`

7

`add(-3, -8)`

-11

`add(-3, 1)`

-2

`add(0, 0)`

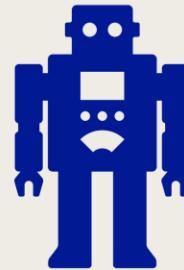
0

Calculator

`add(int x, int y) : int`

Benefits of Unit Testing

- Automated tests
- Better code design
- Fewer bugs and higher reliability
- Increases confidence for code refactoring ... did I break anything??
- Basic requirement for
 - DevOps and Build Pipelines
 - Continuous Integration / Continuous Deployment (CI/CD)



Integration Testing

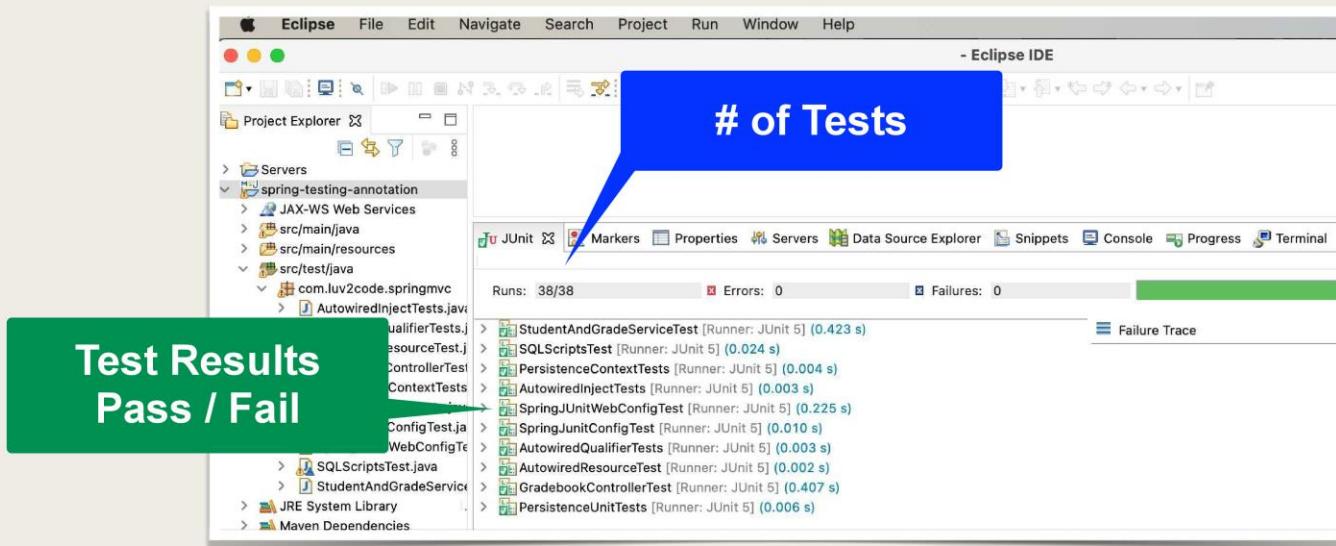
- Test multiple components together as part of a test plan
- Determine if software units work together as expected
- Identify any negative side effects due to integration
- Can test using mocks / stubs
- Can also test using live integrations (database, file system)

Unit Testing Frameworks

- JUnit
 - Supports creating test cases
 - Automation of the test cases with pass / fail
 - Utilities for test setup, teardown and assertions
- Mockito
 - Create mocks and stubs
 - Minimize dependencies on external components

Unit Testing Tooling Support

- Eclipse



Spring Boot - Testing

Getting Started

- Let's start with some simple examples for unit testing
- At the moment, we will focus on the JUnit fundamentals for testing
 - How to define a test
 - JUnit Assertions
 - Running tests
- Later we'll discuss other techniques such as Test-Driven-Development (TDD)

Test Cases

`add(2, 4)`

6

`add(1, 9)`

10

DemoUtils

`add(int x, int y) : int`

Code to Test

DemoUtils.java

```
package com.luv2code.junitdemo;

public class DemoUtils {

    public int add(int a, int b) {
        return a + b;
    }

}
```

Development Process

Step-By-Step

1. Add Maven dependencies for JUnit
2. Create test package
3. Create unit test
4. Run unit test

Step 1: Add Maven dependencies for JUnit

pom.xml

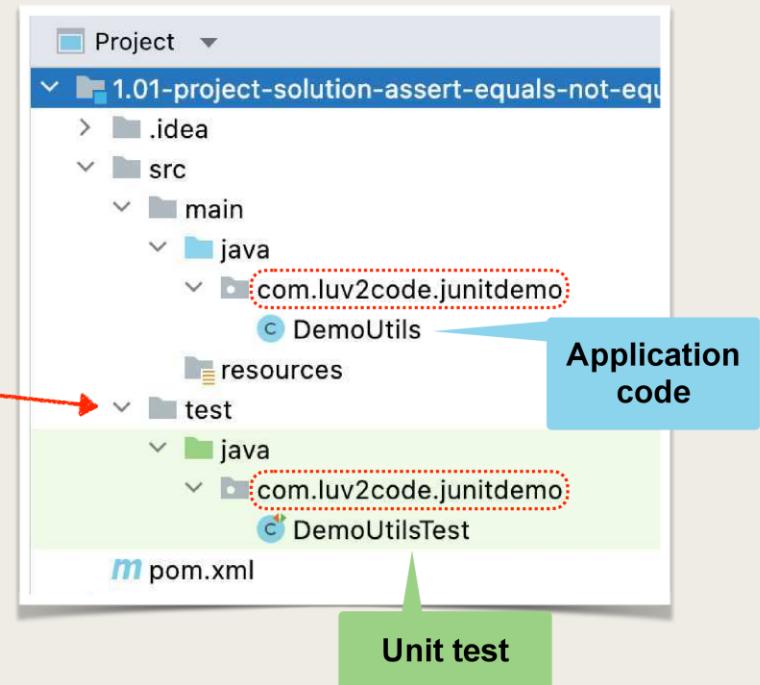
...

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.8.2</version>
    <scope>test</scope>
</dependency>
```

...

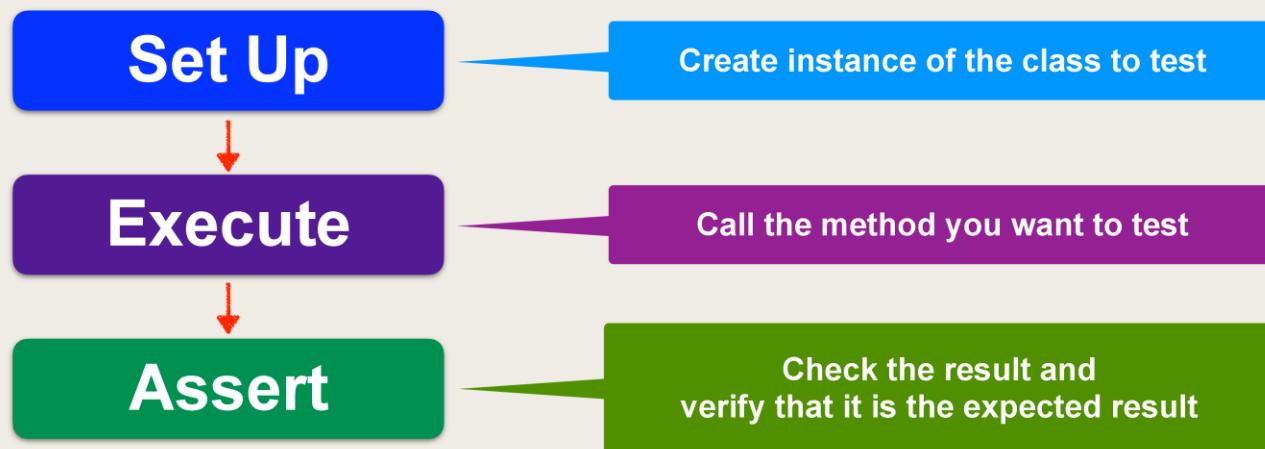
Step 2: Create test package

- The code we are testing is located in package:
 - `com.luv2code.junitdemo`
- A convention is to create test classes in similar package structure under `/test`
- Not a hard requirement, merely a convention
 - Helps with organization of test classes
 - Easy to find test classes when joining new projects
 - Special edge cases for accessing protected class members



Step 3: Create unit test

- Unit tests have the following structure



Step 3: Create unit test

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Assertions;

class DemoUtilsTest {

    @Test
    void testEqualsAndHashCode() {
        // set up
        DemoUtils demoUtils = new DemoUtils();

        int expected = 6;

        // execute
        int actual = demoUtils.add(2, 4);

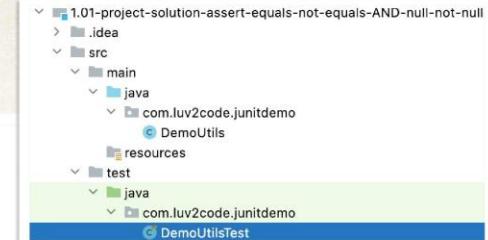
        // assert
        Assertions.assertEquals(expected, actual, "2+4 must be 6");
    }
}
```

Annotate the method
as a test method

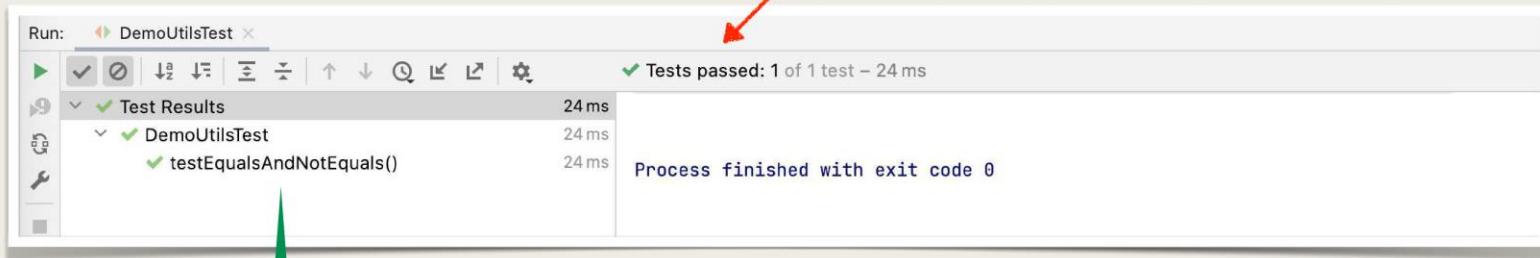
Create instance of the class to test

Call the method you want to test

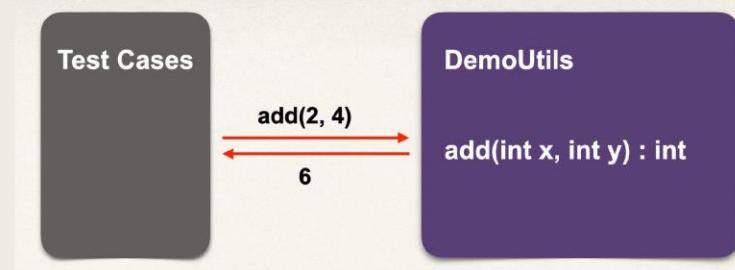
Check the *actual* result and
verify that it is the *expected* result



Step 4: Run unit test



Test passed



JUnit Assertions

- JUnit has a collection of assertions
- Defined in class: `org.junit.jupiter.api.Assertions`

Method name	Description
<code>assertEquals</code>	Assert that the values are equal
<code>assertNotEquals</code>	Assert that the values are not equal
<code>assertNull</code>	Assert that the value is null
<code>assertNotNull</code>	Assert that the value is not null
...	...

assertEquals

Expected value

Optional message
if test fails

```
Assertions.assertEquals(expected, actual, "2+4 must be 6");
```

Actual value
after executing method
under test

assertNotEquals

Unexpected value

Optional message
if test fails

```
Assertions.assertEquals(unexpected, actual, "2+4 must not be 8");
```

Actual value
after executing method
under test

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Assertions;

class DemoUtilsTest {

    @Test
    void testEqualsAndHashCode() {

        // set up
        DemoUtils demoUtils = new DemoUtils();

        int expected = 6;
        int unexpected = 8;

        // execute
        int actual = demoUtils.add(2, 4);

        // assert
        Assertions.assertEquals(expected, actual, "2+4 must be 6");
        Assertions.assertNotEquals(unexpected, actual, "2+4 must not be 8");

    }
}
```

Create instance of the class to test

Call the method you want to test

Check the *actual* result and
verify that it is NOT the *unexpected* result

Static Import

- Static import: a short-cut for referencing static methods & fields in a class

BEFORE

```
import org.junit.jupiter.api.Assertions;  
  
...  
  
Assertions.assertEquals(expected, actual, "2+4 must be 6");  
Assertions.assertNotEquals(unexpected, actual, "2+4 must not be 8");
```

Class name

Method name

Short-cut for
static methods & fields

Static import

Just give
method name

AFTER

```
import static org.junit.jupiter.api.Assertions.assertEquals;  
import static org.junit.jupiter.api.Assertions.assertNotEquals;  
  
...  
  
assertEquals(expected, actual, "2+4 must be 6");  
assertNotEquals(unexpected, actual, "2+4 must not be 8");
```

Static Import

- Can also use wildcard for static import

BEFORE

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

...
assertEquals(expected, actual, "2+4 must be 6");
assertNotEquals(unexpected, actual, "2+4 must not be 8");
```

Wildcard

AFTER

Just give
method name

```
import static org.junit.jupiter.api.Assertions.*;
...
assertEquals(expected, actual, "2+4 must be 6");
assertNotEquals(unexpected, actual, "2+4 must not be 8");
```

Code to Test

DemoUtils.java

```
package com.luv2code.junitdemo;

public class DemoUtils {

    public Object checkNull(Object obj) {

        if (obj != null) {
            return obj;
        }
        return null;
    }
}
```

Assertions: Null and NotNull

Method name	Description
<code>assertNull</code>	Assert that the value is null
<code>assertNotNull</code>	Assert that the value is not null
...	...

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

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

class DemoUtilsTest {

    @Test
    void testNullAndNotNull() {

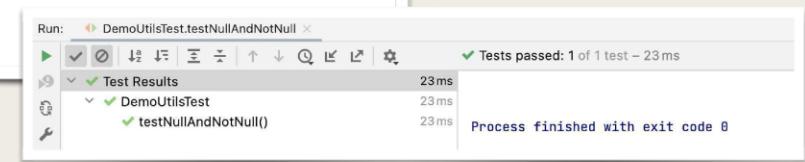
        DemoUtils demoUtils = new DemoUtils();

        String str1 = null;
        String str2 = "luv2code";

        assertNull(demoUtils.checkNotNull(str1), "Object should be null");
        assertNotNull(demoUtils.checkNotNull(str2), "Object should not be null");
    }
}
```

Actual value
after executing method
under test

Optional message
if test fails



Multiple Tests in One Class

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

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

class DemoUtilsTest {

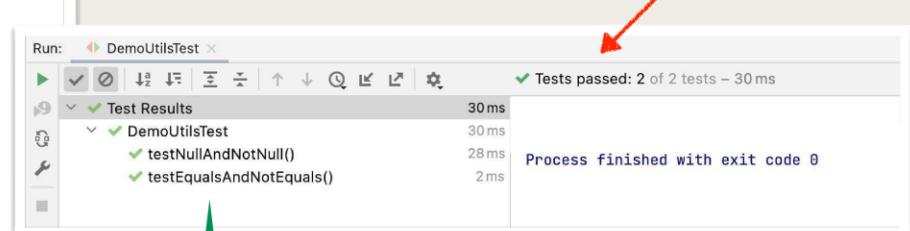
    @Test
    void testEqualsAndNotEquals() {
        // set up
        DemoUtils demoUtils = new DemoUtils();

        // execute and assert
        assertEquals(6, demoUtils.add(2, 4), "2+4 must be 6");
        assertNotEquals(8, demoUtils.add(1, 9), "1+9 must not be 8");
    }

    @Test
    void testNullAndNotNull() {
        DemoUtils demoUtils = new DemoUtils();

        String str1 = null;
        String str2 = "luv2code";

        assertNull(demoUtils.checkNull(str1), "Object should be null");
        assertNotNull(demoUtils.checkNull(str2), "Object should not be null");
    }
}
```



Both tests passed

Lifecycle Methods

- When developing tests, we may need to perform common operations
- Before each test
 - Create objects, set up test data
- After each test
 - Release resources, clean up test data

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

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

class DemoUtilsTest {

    @Test
    void testEqualsAndNotEquals() {
        // set up
        DemoUtils demoUtils = new DemoUtils();

        // execute and assert
        assertEquals(6, demoUtils.add(2, 4), "2+4 must be 6");
        assertNotEquals(8, demoUtils.add(1, 9), "1+9 must not be 8");
    }

    @Test
    void testNullAndNotNull() {
        // set up
        DemoUtils demoUtils = new DemoUtils();

        String str1 = null;
        String str2 = "luv2code";

        assertNull(demoUtils.checkNotNull(str1), "Object should be null");
        assertNotNull(demoUtils.checkNotNull(str2), "Object should not be null");
    }
}
```

BEFORE

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import static org.junit.jupiter.api.Assertions.*;

class DemoUtilsTest {

    DemoUtils demoUtils;

    @BeforeEach
    void setupBeforeEach() {
        // set up
        demoUtils = new DemoUtils();
        System.out.println("@BeforeEach executes before the execution of each test method");
    }

    @Test
    void testEqualsAndNotEquals() {
        // execute and assert
        assertEquals(6, demoUtils.add(2, 4), "2+4 must be 6");
        assertNotEquals(8, demoUtils.add(1, 9), "1+9 must not be 8");
    }

    @Test
    void testNullAndNotNull() {
        String str1 = null;
        String str2 = "luv2code";

        assertNull(demoUtils.checkNotNull(str1), "Object should be null");
        assertNotNull(demoUtils.checkNotNull(str2), "Object should not be null");
    }
}
```

AFTER

Create object before each test

No need to create object ... handled by @BeforeEach

No need to create object ... handled by @BeforeEach

Spring Boot - Testing

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.AfterEach;
import static org.junit.jupiter.api.Assertions.*;

class DemoUtilsTest {

    DemoUtils demoUtils;

    @BeforeEach
    void setupBeforeEach() {
        // set up
        demoUtils = new DemoUtils();
        System.out.println("@BeforeEach executes before the execution of each test method");
    }

    @AfterEach
    void tearDownAfterEach() {
        System.out.println("Running @AfterEach\n");
    }

    @Test
    void testEqualsAndHashCode() {

        // execute and assert
        assertEquals(6, demoUtils.add(2, 4), "2+4 must be 6");
        assertNotEquals(8, demoUtils.add(1, 9), "1+9 must not be 8");
    }

    @Test
    void testNullAndNotNull() {
        String str1 = null;
        String str2 = "luv2code";

        assertNull(demoUtils.checkNotNull(str1), "Object should be null");
        assertNotNull(demoUtils.checkNotNull(str2), "Object should not be null");
    }
}
```

Run after each test

Execution Sequence

@BeforeEach

@Test Method One

@AfterEach

@BeforeEach

@Test Method Two

@AfterEach

Don't worry about order of test methods.
We'll cover that later.



```
@BeforeEach
void setupBeforeEach() {
    // set up
    demoUtils = new DemoUtils();
    System.out.println("@BeforeEach executes before the execution of each test method");
}

@AfterEach
void tearDownAfterEach() {
    System.out.println("Running @AfterEach\n");
}

@Test
void testNullAndNotNull() {
    System.out.println("Running test: testNullAndNotNull");

    String str1 = null;
    String str2 = "luv2code";

    assertNull(demoUtils.checkNotNull(str1), "Object should be null");
    assertNotNull(demoUtils.checkNotNull(str2), "Object should not be null");
}
...
```

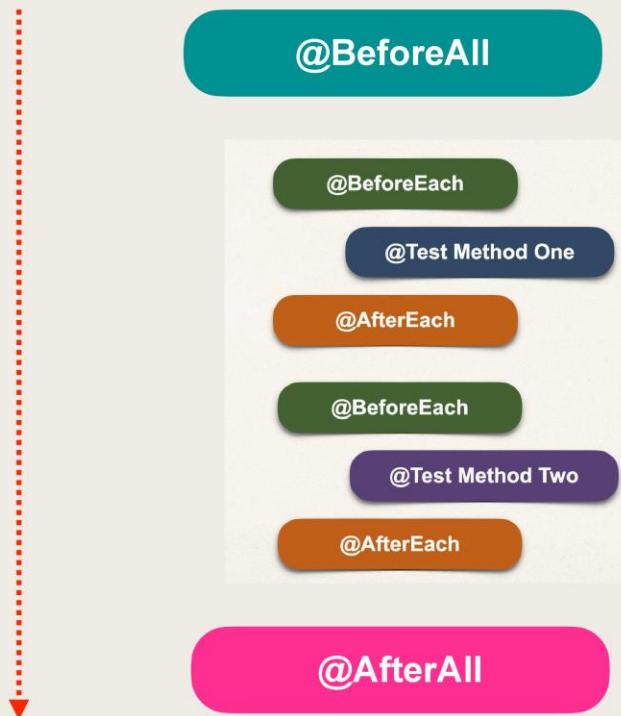
Lifecycle Methods

- When developing tests, we may need to perform one-time operations
- One-time set up before all tests
 - Get database connections, connect to remote servers ...
- One-time clean up after all tests
 - Release database connections, disconnect from remote servers ...

Lifecycle Method Annotations

Annotation	Description
@BeforeAll	Method is executed only once, before all test methods. <i>Useful for getting database connections, connecting to servers ...</i>
@AfterAll	Method is executed only once, after all test methods. <i>Useful for releasing database connections, disconnecting from servers ...</i>
...	...

Execution Sequence



DemoUtilsTest.java

```
package com.luv2code.junitdemo;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.AfterAll;
import static org.junit.jupiter.api.Assertions.*;

class DemoUtilsTest {

    DemoUtils demoUtils;

    @BeforeEach
    void setupBeforeEach() {
        // set up
        demoUtils = new DemoUtils();
        System.out.println("@BeforeEach executes before the execution of each test method");
    }

    @AfterEach
    void tearDownAfterEach() {
        System.out.println("Running @AfterEach\n");
    }

    @BeforeAll
    static void setupBeforeEachClass() {
        System.out.println("@BeforeAll executes only once before all test methods execution in the class\n");
    }

    @AfterAll
    static void tearDownAfterAll() {
        System.out.println("@AfterAll executes only once after all test methods execution in the class");
    }

    @Test
    void testEqualsAndNotEquals() {
        System.out.println("Running test: testEqualsAndNotEquals");
        ...
    }

    @Test
    void testNullAndNotNull() {
        System.out.println("Running test: testNullAndNotNull");
        ...
    }
}
```

Executed only once,
before all test methods

Executed only once,
after all test methods

By default, methods
must be static

Execution Sequence

@BeforeAll

@BeforeEach

@Test Method One

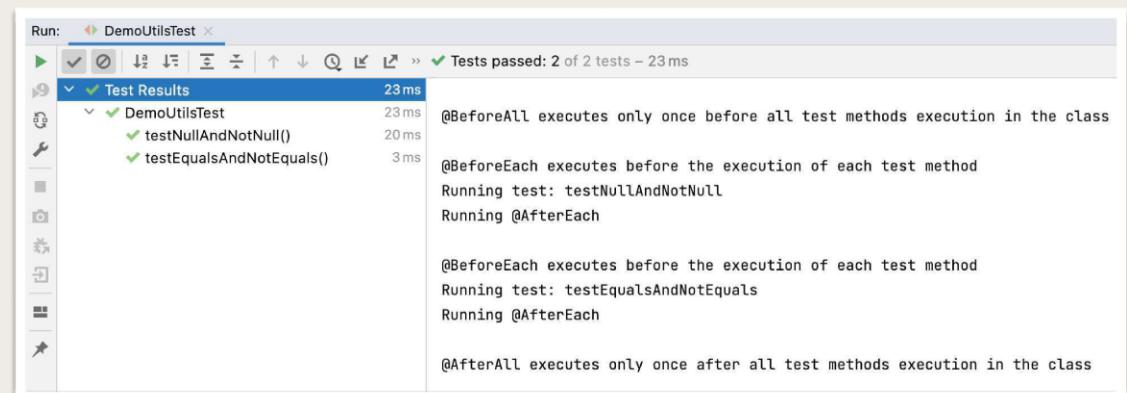
@AfterEach

@BeforeEach

@Test Method Two

@AfterEach

@AfterAll

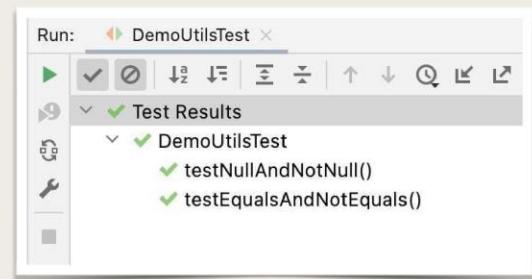


```
@BeforeAll
static void setupBeforeEachClass() {
    System.out.println("@BeforeAll executes only once before all test methods execution in the class\n");
}

@AfterAll
static void tearDownAfterAll() {
    System.out.println("@AfterAll executes only once after all test methods execution in the class");
}
```

Custom Display Names

- Currently the method names are listed in test results
- We'd like to give custom display names
 - Provide a more descriptive name for the test
 - Include spaces, special characters: *Test for Equality to support JIRA #123*
 - Useful for sharing test reports with project management and non-techies



@DisplayName Annotation

Annotation	Description
@DisplayName	Custom display name with spaces, special characters and emojis. <i>Useful for test reports in IDE or external test runner</i>

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import static org.junit.jupiter.api.Assertions.*;

class DemoUtilsTest {

    DemoUtils demoUtils;

    @BeforeEach
    void setupBeforeEach() {
        // set up
        demoUtils = new DemoUtils();
    }

    @Test
    @DisplayName("Null and Not Null")
    void testNullAndNotNull() {
        String str1 = null;
        String str2 = "luv2code";

        assertNull(demoUtils.checkNotNull(str1), "Object should be null");
        assertNotEqual(demoUtils.checkNotNull(str2), "Object should not be null");
    }

    @Test
    @DisplayName("Equals and Not Equals")
    void testEqualsAndNotEquals() {

        // execute and assert
        assertEquals(6, demoUtils.add(2, 4), "2+4 must be 6");
        assertNotEqual(8, demoUtils.add(1, 9), "1+9 must not be 8");
    }
}
```

Run: All in junitdemo	
	✓
	✖
	↓ ^a
	⟳
	☰
	↑
	↓
	⟳
	☰
	⟳
	☰
	✓ <default package>
	▼
	✓ DemoUtilsTest
	▼
	✓ Null and Not Null
	✓ Equals and Not Equals
	29ms
	29ms
	25ms
	4ms

Display Name Generators

- JUnit can generate display names for you

Name	Description
Simple	Removes trailing parentheses from test method name
ReplaceUnderscores	Replaces underscores in test method name with spaces
IndicativeSentences	Generate sentence based on test class name and test method name

Simple Generator

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayNameGeneration;
import org.junit.jupiter.api.DisplayNameGenerator;
import static org.junit.jupiter.api.Assertions.*;

@DisplayNameGeneration(DisplayNameGenerator.Simple.class)
class DemoUtilsTest {

    DemoUtils demoUtils;

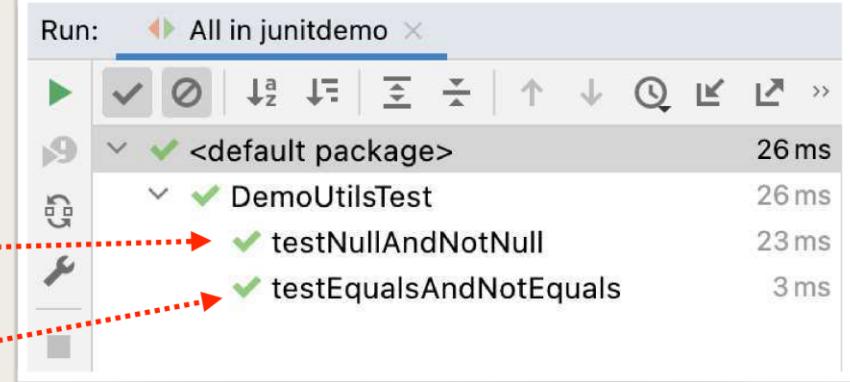
    @BeforeEach
    void setupBeforeEach() {
        // set up
        demoUtils = new DemoUtils();
    }

    @Test
    void testNullAndNotNull() {
        String str1 = null;
        String str2 = "luv2code";

        assertNull(demoUtils.checkNull(str1), "Object should be null");
        assertNotEquals(demoUtils.checkNull(str2), "Object should not be null");
    }

    @Test
    void testEqualsAndNotEquals() {
        // execute and assert
        assertEquals(6, demoUtils.add(2, 4), "2+4 must be 6");
        assertNotEquals(8, demoUtils.add(1, 9), "1+9 must not be 8");
    }
}
```

Removes trailing parentheses from test method name



Replace Underscores Generator

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayNameGeneration;
import org.junit.jupiter.api.DisplayNameGenerator;
import static org.junit.jupiter.api.Assertions.*;

@DisplayNameGeneration(DisplayNameGenerator.ReplaceUnderscores.class)
class DemoUtilsTest {

    DemoUtils demoUtils;

    @BeforeEach
    void setupBeforeEach() {
        // set up
        demoUtils = new DemoUtils();
    }

    @Test
    void test_Equals_And_Not_Equals() {
        // execute and assert
        assertEquals(6, demoUtils.add(2, 4), "2+4 must be 6");
        assertNotEquals(8, demoUtils.add(1, 9), "1+9 must not be 8");
    }

    @Test
    void test_Null_And_Not_Null() {
        String str1 = null;
        String str2 = "luv2code";

        assertNull(demoUtils.checkNotNull(str1), "Object should be null");
        assertNotNull(demoUtils.checkNotNull(str2), "Object should not be null");
    }
}
```

Replaces underscores with spaces

Run: All in junitdemo	
	✓
	All in junitdemo
	<default package>
	DemoUtilsTest
	✓ test Equals And Not Equals
	✓ test Null And Not Null

Indicative Sentences Generator

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayNameGeneration;
import org.junit.jupiter.api.DisplayNameGenerator;
import static org.junit.jupiter.api.Assertions.*;

@DisplayNameGeneration(DisplayNameGenerator.IndicativeSentences.class)
class DemoUtilsTest {

    DemoUtils demoUtils;

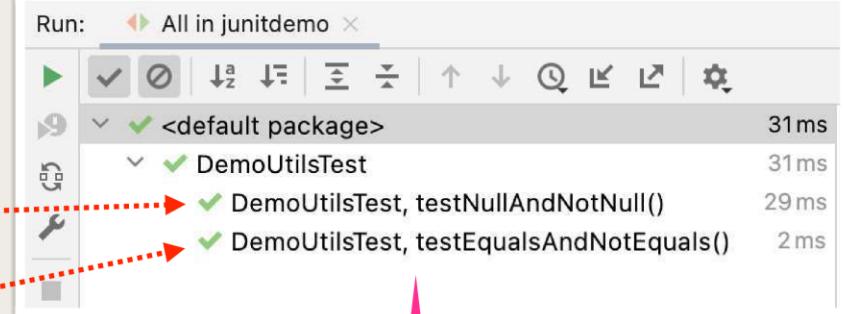
    @BeforeEach
    void setupBeforeEach() {
        // set up
        demoUtils = new DemoUtils();
    }

    @Test
    void testNullAndNotNull() {
        String str1 = null;
        String str2 = "luv2code";

        assertNull(demoUtils.checkNull(str1), "Object should be null");
        assertNotEquals(demoUtils.checkNull(str2), "Object should not be null");
    }

    @Test
    void testEqualsAndNotEquals() {
        // execute and assert
        assertEquals(6, demoUtils.add(2, 4), "2+4 must be 6");
        assertNotEquals(8, demoUtils.add(1, 9), "1+9 must not be 8");
    }
}
```

Generate sentence based on
test class name and test method name



<testClassName>, <methodName>

Assertions

- Test for Same and NotSame

Method name	Description
<code>assertSame</code>	Assert that items refer to same object
<code>assertNotSame</code>	Assert that items do not refer to same object

Code to Test

DemoUtils.java

```
package com.luv2code.junitdemo;

public class DemoUtils {

    private String academy = "Luv2Code Academy";
    private String academyDuplicate = academy;

    public String getAcademy() {
        return academy;
    }

    public String getAcademyDuplicate() {
        return academyDuplicate;
    }
}
```

Check if these refer to the same object

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

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

class DemoUtilsTest {

    DemoUtils demoUtils;
    ...
    @DisplayName("Same and Not Same")
    @Test
    void testSameAndNotSame() {
        String str = "luv2code";
        assertEquals(demoUtils.getAcademy(), demoUtils.getAcademyDuplicate(), "Objects should refer to same object");
        assertNotEquals(str, demoUtils.getAcademy(), "Objects should not refer to same object");
    }
}
```

Object1

Object2

Object1

Object2

DemoUtils.java

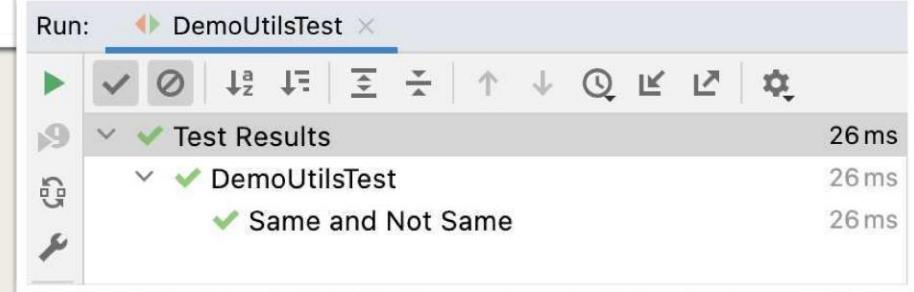
```
package com.luv2code.junitdemo;

public class DemoUtils {

    private String academy = "Luv2Code Academy";
    private String academyDuplicate = academy;

    public String getAcademy() {
        return academy;
    }

    public String getAcademyDuplicate() {
        return academyDuplicate;
    }
}
```



Assertions

- Test for True and False

Method name	Description
<code>assertTrue</code>	Assert that condition is true
<code>assertFalse</code>	Assert that condition is false

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

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

class DemoUtilsTest {

    DemoUtils demoUtils;

    ...

    @DisplayName("True and False")
    @Test
    void testTrueFalse() {
        int gradeOne = 10;
        int gradeTwo = 5;

        assertTrue(demoUtils.isGreater(gradeOne, gradeTwo), "This should return true");
        assertFalse(demoUtils.isGreater(gradeTwo, gradeOne), "This should return false");
    }
}
```

Boolean condition

Boolean condition

DemoUtils.java

```
package com.luv2code.junitdemo;

public class DemoUtils {

    public Boolean isGreater(int n1, int n2) {
        return n1 > n2;
    }
}
```

Run: DemoUtilsTest

		30 ms
		30 ms
		30 ms
		30 ms

Assertions

Method name	Description
assertArrayEquals	Assert that both object arrays are deeply equal

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

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

class DemoUtilsTest {

    DemoUtils demoUtils;
    ...
    @DisplayName("Array Equals")
    @Test
    void testArrayEquals() {
        String[] stringArray = {"A", "B", "C"};
        assertArrayEquals(stringArray, demoUtils.getFirstThreeLettersOfAlphabet(), "Arrays should be the same");
    }
}
```

DemoUtils.java

```
package com.luv2code.junitdemo;

public class DemoUtils {

    private String[] firstThreeLettersOfAlphabet = {"A", "B", "C"};

    public String[] getFirstThreeLettersOfAlphabet() {
        return firstThreeLettersOfAlphabet;
    }
}
```

Run: DemoUtilsTest.testArrayEquals

Test Results 57 ms

DemoUtilsTest 57 ms

 Array Equals 57 ms

Assertions

Method name	Description
assertIterableEquals	Assert that both object iterables are deeply equal

An "iterable" is an instance of a class
that implements the `java.lang.Iterable` interface

Examples: `ArrayList`, `LinkedList`, `HashSet`, `TreeSet` ...

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;
import java.util.List;

class DemoUtilsTest {

    DemoUtils demoUtils;

    ...

    @DisplayName("Iterable equals")
    @Test
    void testIterableEquals() {
        List<String> theList = List.of("luv", "2", "code");

        assertIterableEquals(theList, demoUtils.getAcademyInList(), "Expected list should be same as actual list");
    }
}
```

```
DemoUtils.java

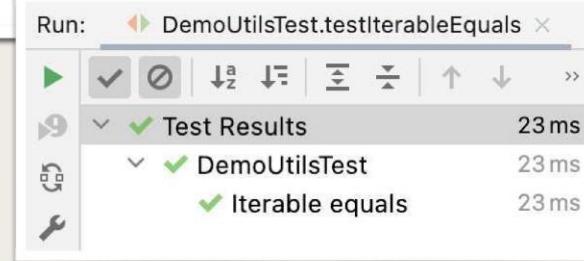
package com.luv2code.junitdemo;

import java.util.List;

public class DemoUtils {

    private List<String> academyInList = List.of("luv", "2", "code");

    public List<String> getAcademyInList() {
        return academyInList;
    }
}
```



Assertions

Method name	Description
assertLinesMatch	Assert that both lists of strings match

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;
import java.util.List;
}

class DemoUtilsTest {

    DemoUtils demoUtils;

    ...

    @DisplayName("Lines match")
    @Test
    void testLinesMatch() {
        List<String> theList = List.of("luv", "2", "code");

        assertLinesMatch(theList, demoUtils.getAcademyInList(), "Lines should match");
    }
}
```

```
DemoUtils.java

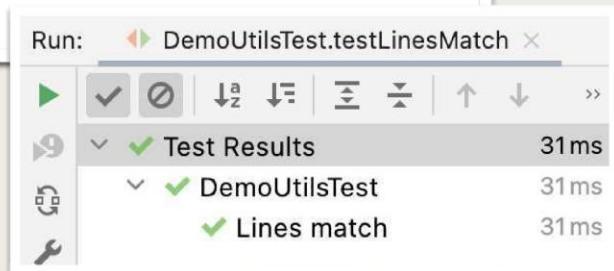
package com.luv2code.junitdemo;

import java.util.List;

public class DemoUtils {

    private List<String> academyInList = List.of("luv", "2", "code");

    public List<String> getAcademyInList() {
        return academyInList;
    }
}
```



Assertions

Method name	Description
<code>assertThrows</code>	Assert that an executable throws an exception of expected type

Code to Test

DemoUtils.java

```
package com.luv2code.junitdemo;

public class DemoUtils {

    public String throwException(int a) throws Exception {
        if (a < 0) {
            throw new Exception("Value should be greater than or equal to 0");
        }
        return "Value is greater than or equal to 0";
    }
}
```

Verify this method
throws an exception for values < 0

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

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

class DemoUtilsTest {

    DemoUtils demoUtils;

    ...

    @DisplayName("Throws and Does Not Throw")
    @Test
    void testThrowsAndDoesNotThrow() {
        assertThrows(Exception.class, () -> { demoUtils.throwException(-1); }, "Should throw exception");

        assertDoesNotThrow(() -> { demoUtils.throwException(5); }, "Should not throw exception");
    }
}
```

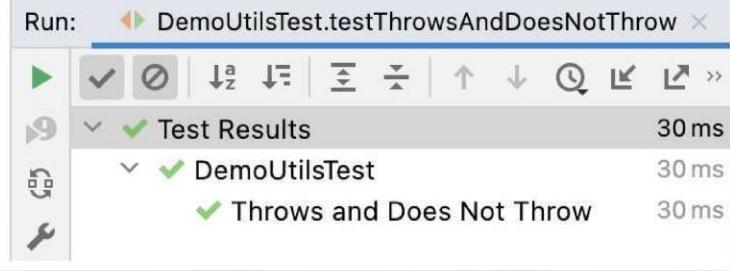
Lambda expression

DemoUtils.java

```
package com.luv2code.junitdemo;

public class DemoUtils {

    public String throwException(int a) throws Exception {
        if (a < 0) {
            throw new Exception("Value should be greater than or equal to 0");
        }
        return "Value is greater than or equal to 0";
    }
}
```



Assertions

Method name	Description
<code>assertTimeoutPreemptively</code>	Assert that an executable completes before given timeout is exceeded

Execution is preemptively aborted
if timeout is exceeded

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;
import java.time.Duration;

class DemoUtilsTest {

    DemoUtils demoUtils;
    ...
    @DisplayName("Timeout")
    @Test
    void testTimeout() {
        assertTimeoutPreemptively(Duration.ofSeconds(3), () -> { demoUtils.checkTimeout(); },
            "Method should execute in 3 seconds");
    }
}
```

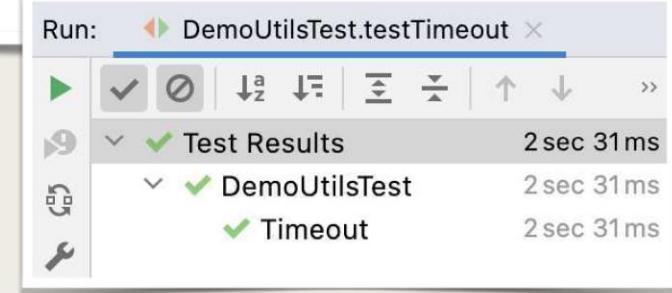
Timeout duration

Lambda expression

```
DemoUtils.java
package com.luv2code.junitdemo;

public class DemoUtils {

    public void checkTimeout() throws InterruptedException {
        System.out.println("I am going to sleep");
        Thread.sleep(2000);
        System.out.println("Sleeping over");
    }
}
```



Remember this?



Order???

- In general
 - Order should not be a factor in unit tests
 - There should be no dependency between tests
 - All tests should pass regardless of the order in which they are run

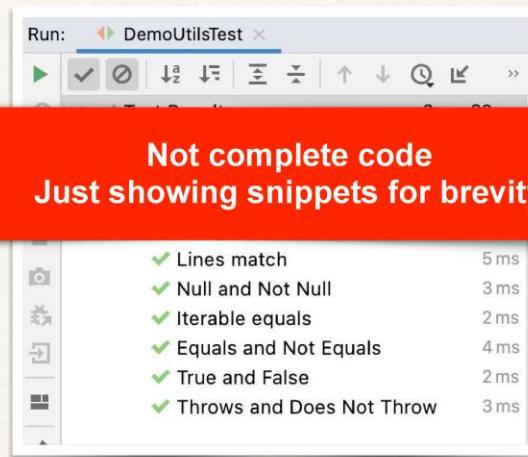
Use Cases

- However, there are some uses cases when you want to control the order
 - You want tests to appear in alphabetical order for reporting purposes
 - Sharing reports with project management, QA team etc...
 - Group tests based on functionality or requirements

Existing ... before any changes

DemoUtilsTest.java

```
class DemoUtilsTest {  
    ...  
  
    @DisplayName("Equals and Not Equals")  
    void testEqualsAndNotEquals()  
  
    @DisplayName("Null and Not Null")  
    void testNullAndNotNull()  
  
    @DisplayName("Same and Not Same")  
    void testSameAndNotSame()  
  
    @DisplayName("True and False")  
    void testTrueFalse()  
  
    @DisplayName("Array Equals")  
    void testArrayEquals()  
  
    @DisplayName("Iterable equals")  
    void testIterableEquals()  
  
    @DisplayName("Lines match")  
    void testLinesMatch()  
  
    @DisplayName("Throws and Does Not Throw")  
    void testThrowsAndDoesNotThrow()  
  
    @DisplayName("Timeout")  
    void testTimeout()  
}
```



JUnit Docs

By default, test classes and methods will be ordered using an algorithm that is deterministic but intentionally nonobvious.

<https://junit.org/junit5/docs/current/user-guide/>

Spring Boot - Testing

Annotation

Annotation	Description
@TestMethodOrder	Configures the order/sort algorithm for the test methods

Specify Method Order

Name	Description
<code>MethodOrderer.DisplayName</code>	Sorts test methods alphanumerically based on display names
<code>MethodOrderer.MethodName</code>	Sorts test methods alphanumerically based on method names
<code>MethodOrderer.Random</code>	Pseudo-random order based on method names
<code>MethodOrderer.OrderAnnotation</code>	Sorts test methods numerically based on @Order annotation

Order by Display Name

Sorted alphanumerically

DemoUtilsTest.java

```
@TestMethodOrder(MethodOrderer.DisplayName.class)
class DemoUtilsTest {
    ...

    @DisplayName("Equals and Not Equals")
    void testEqualsAndNotEquals()

    @DisplayName("Null and Not Null")
    void testNullAndNotNull()

    @DisplayName("Same and Not Same")
    void testSameAndNotSame()

    @DisplayName("True and False")
    void testTrueFalse()

    @DisplayName("Array Equals")
    void testArrayEquals()

    @DisplayName("Iterable equals")
    void testIterableEquals()

    @DisplayName("Lines match")
    void testLinesMatch()

    @DisplayName("Throws and Does Not Throw")
    void testThrowsAndDoesNotThrow()

    @DisplayName("Timeout")
    void testTimeout()
}
```

Run: DemoUtilsTest	
	✓
	✗
	↓ ^a
	✗
	2 sec 42 ms
▼	Test Results
▼	▼ DemoUtilsTest 2 sec 42 ms
✓	Array Equals 17 ms
✓	Equals and Not Equals 2 ms
✓	Iterable equals 1 ms
✓	Lines match 4 ms
✓	Null and Not Null 2 ms
✓	Same and Not Same 2 ms
✓	Throws and Does Not Throw 3 ms
✓	Timeout 2 sec 9 ms
✓	True and False 2 ms

Spring Boot - Testing

Order by Method Name

Sorted alphanumerically

DemoUtilsTest.java

```
@TestMethodOrder(MethodOrderer.MethodName.class)
class DemoUtilsTest {
    ...
    // @DisplayName("Equals and Not Equals")
    void testEqualsAndNotEquals()

    void testNullAndNotNull()

    void testSameAndNotSame()

    void testTrueFalse()

    void testArrayEquals()

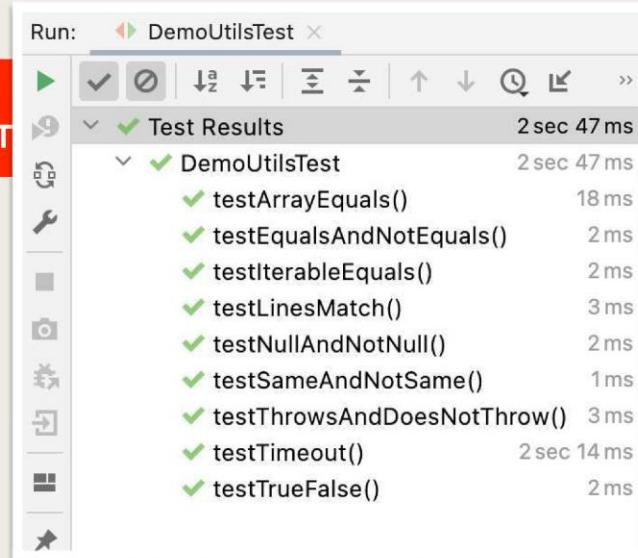
    void testIterableEquals()

    void testLinesMatch()

    void testThrowsAndDoesNotThrow()

    void testTimeout()
}
```

NOT



Random by Method Name

DemoUtilsTest.java

```
@TestMethodOrder(MethodOrderer.Random.class)
class DemoUtilsTest {
    ...
    void testEqualsAndNotEquals()
    void testNullAndNotNull()
    void testSameAndNotSame()
    void testTrueFalse()
    void testArrayEquals()
    void testIterableEquals()
    void testLinesMatch()
    ...
}
```

Can also use DisplayName
since everything is random
... order doesn't matter

Run: DemoUtilsTest	
Test Results 2 sec 47 ms	
✓	DemoUtilsTest
✓	testTrueFalse() 18 ms
✓	testLinesMatch() 4 ms
✓	testArrayEquals() 2 ms
✓	testTimeout() 2 sec 9 ms
✓	testThrowsAndDoesNotThrow() 3 ms
✓	testEqualsAndNotEquals() 3 ms
✓	testNullAndNotNull() 3 ms
✓	testIterableEquals() 3 ms
✓	testSameAndNotSame() 2 ms

Run: DemoUtilsTest	
Test Results 2 sec 58 ms	
✓	DemoUtilsTest
✓	testLinesMatch() 29 ms
✓	testTimeout() 2 sec 12 ms
✓	testNullAndNotNull() 3 ms
✓	testThrowsAndDoesNotThrow() 4 ms
✓	testIterableEquals() 2 ms
✓	testArrayEquals() 2 ms
✓	testEqualsAndNotEquals() 2 ms
✓	testTrueFalse() 2 ms
✓	testSameAndNotSame() 2 ms

Great scenario!!!!

Make sure all of your tests pass regardless of order
Confirms no dependencies between tests

Annotation

Annotation	Description
@Order	<p>Manually specify the order with an <code>int</code> number</p> <ul style="list-style-type: none">- Order with lowest number has highest priority- Negative numbers are allowed

@Order

Remember
Lowest number has highest priority

DemoUtilsTest.java

```
① @TestMethodOrder(MethodOrderer.OrderAnnotation.class)
class DemoUtilsTest {
    ...
    ② @DisplayName("Equals and Not Equals")
    @Order(3)
    void testEqualsAndNotEquals()

    ③ @DisplayName("Null and Not Null")
    @Order(1)
    void testNullAndNotNull()

    @DisplayName("Same and Not Same")
    void testSameAndNotSame()

    @DisplayName("True and False")
    void testTrueFalse()

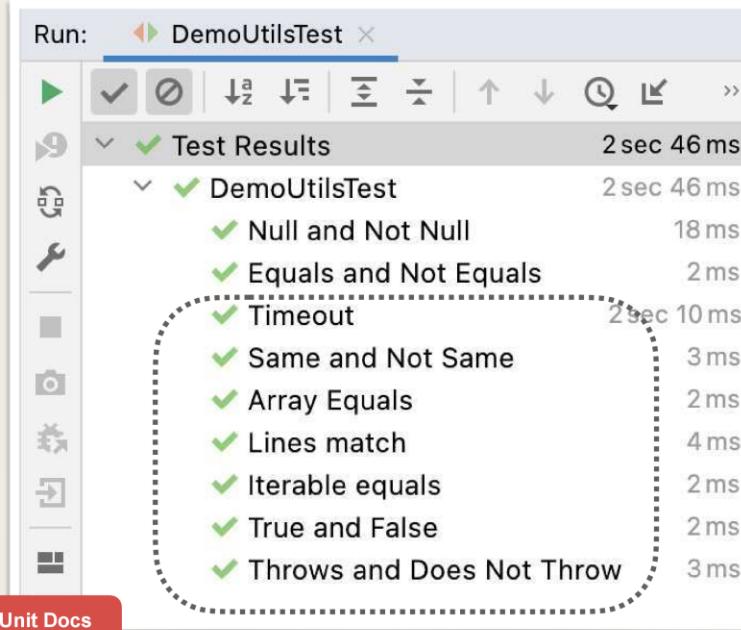
    @DisplayName("Array Equals")
    void testArrayEquals()

    @DisplayName("Iterable equals")
    void testIterableEquals()

    @DisplayName("Lines match")
    void testLinesMatch()

    @DisplayName("Throws and Does Not Throw")
    void testThrowsAndDoesNotThrow()

    @DisplayName("Timeout")
    void testTimeout()
}
```



JUnit Docs

By default, test classes and methods will be ordered using an algorithm that is deterministic but intentionally nonobvious.

Spring Boot - Testing

@Order - Negative Numbers

Remember
Lowest number has highest priority

DemoUtilsTest.java

```
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
class DemoUtilsTest {
    ...
    @DisplayName("Equals and Not Equals")
    @Order(3) ③
    void testEqualsAndNotEquals()

    @DisplayName("Null and Not Null")
    @Order(1) ①
    void testNullAndNotNull()

    @DisplayName("Same and Not Same")
    void testSameAndNotSame()

    @DisplayName("True and False")
    void testTrueFalse()

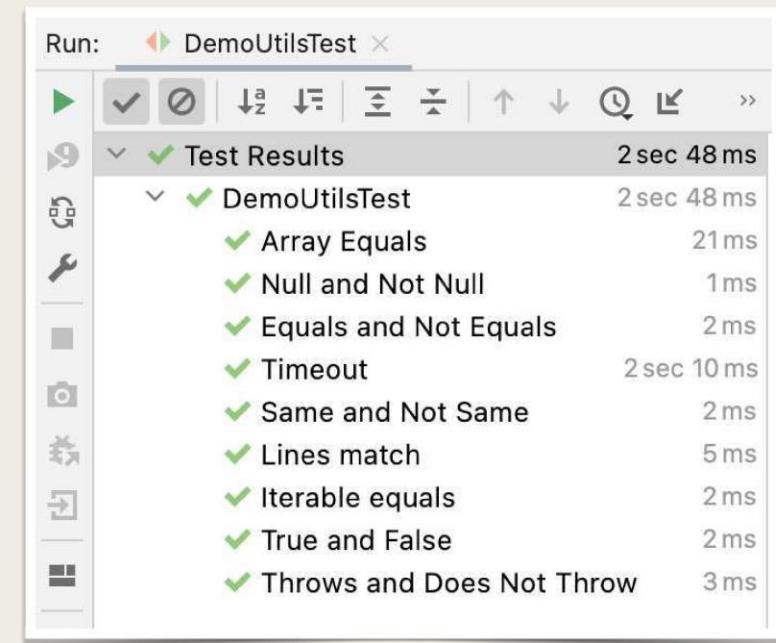
    @DisplayName("Array Equals")
    @Order(-7) ④ -7
    void testArrayEquals()

    @DisplayName("Iterable equals")
    void testIterableEquals()

    @DisplayName("Lines match")
    void testLinesMatch()

    @DisplayName("Throws and Does Not Throw")
    void testThrowsAndDoesNotThrow()

    @DisplayName("Timeout")
    void testTimeout()
}
```



@Order - Duplicate Order Values

If duplicate @Order values
then ...

JUnit Docs

By default, test classes and methods will be ordered using an algorithm that is deterministic but intentionally nonobvious.

Conditional Tests - Use Cases

- Don't run a test because the method to test is broken ... and we are waiting on dev team to fix it
- A test should only run for a specific version of Java (Java 18) or range of versions (13 - 18)
- A test should only run on a given operating system: MS Windows, Mac, Linux
- A test should only run if specific environment variables or system properties are set

Why not just comment the code???

- We could do that ... but then the tests will not be displayed in reports
- Easy to forget about broken tests
- Manual process to enable / disable tests for a given operating system etc ...
- Let's report the tests ... so that management and QA are aware of the issue

Annotations

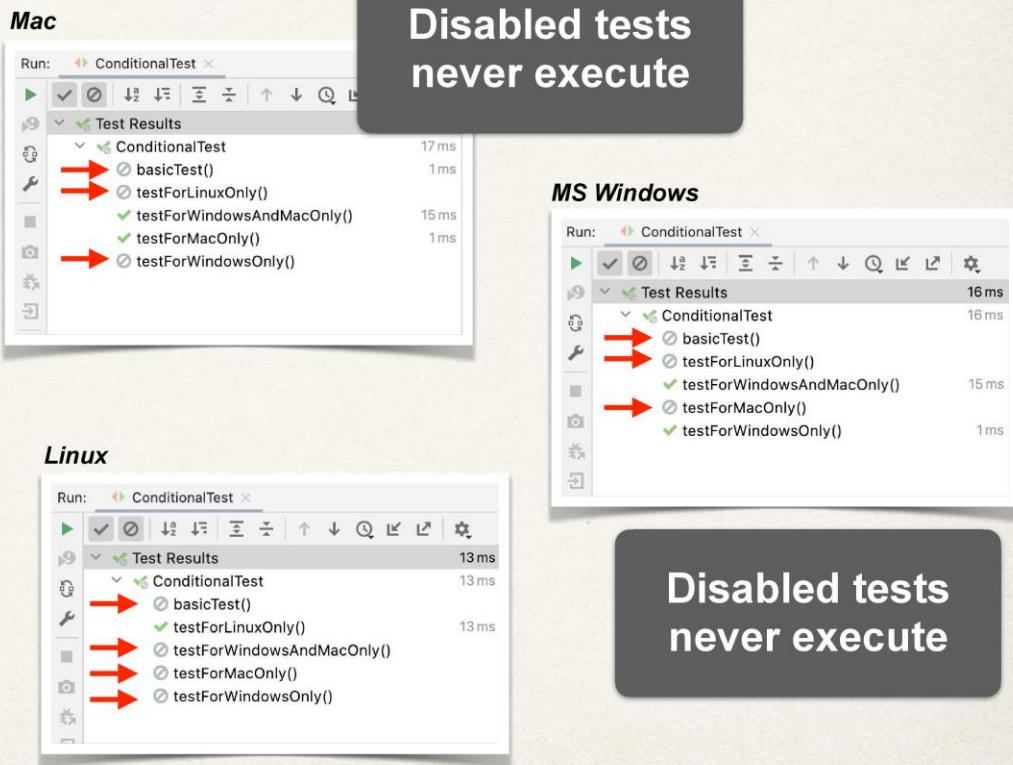
Name	Description
@Disabled	Disable a test method
@EnabledOnOs	Enable test when running on a given operating system
...	...

**Annotations can be applied at
the class level or method level**

@Disabled and @EnabledOnOs

ConditionalTest.java

```
class ConditionalTest {  
  
    @Test  
    @Disabled("Don't run until JIRA #123 is resolved")  
    void basicTest() {  
        // execute method and perform assertions  
    }  
  
    @Test  
    @EnabledOnOs(OS.WINDOWS)  
    void testForWindowsOnly() {  
        // execute method and perform assertions  
    }  
  
    @Test  
    @EnabledOnOs(OS.MAC)  
    void testForMacOnly() {  
        // execute method and perform assertions  
    }  
  
    @Test  
    @EnabledOnOs({OS.WINDOWS, OS.MAC})  
    void testForWindowsAndMacOnly() {  
        // execute method and perform assertions  
    }  
  
    @Test  
    @EnabledOnOs(OS.LINUX)  
    void testForLinuxOnly() {  
        // execute method and perform assertions  
    }  
}
```



Annotations

Name	Description
@EnabledOnJre	Enable test for a given Java version
@EnabledForJreRange	Enable test for a given Java version range
...	...

@EnabledOnJre @EnabledForJreRange

ConditionalTest.java

```
class ConditionalTest {

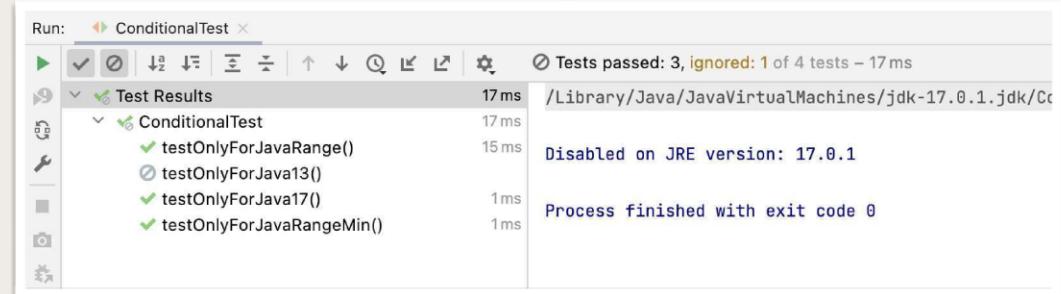
    @Test
    @EnabledOnJre(JRE.JAVA_17)
    void testOnlyForJava17() {
        // execute method and perform assertions
    }

    @Test
    @EnabledOnJre(JRE.JAVA_13)
    void testOnlyForJava13() {
        // execute method and perform assertions
    }

    @Test
    @EnabledForJreRange(min=JRE.JAVA_13, max=JRE.JAVA_18)
    void testOnlyForJavaRange() {
        // execute method and perform assertions
    }

    @Test
    @EnabledForJreRange(min=JRE.JAVA_11)
    void testOnlyForJavaRangeMin() {
        // execute method and perform assertions
    }
}
```

Running on Java 17

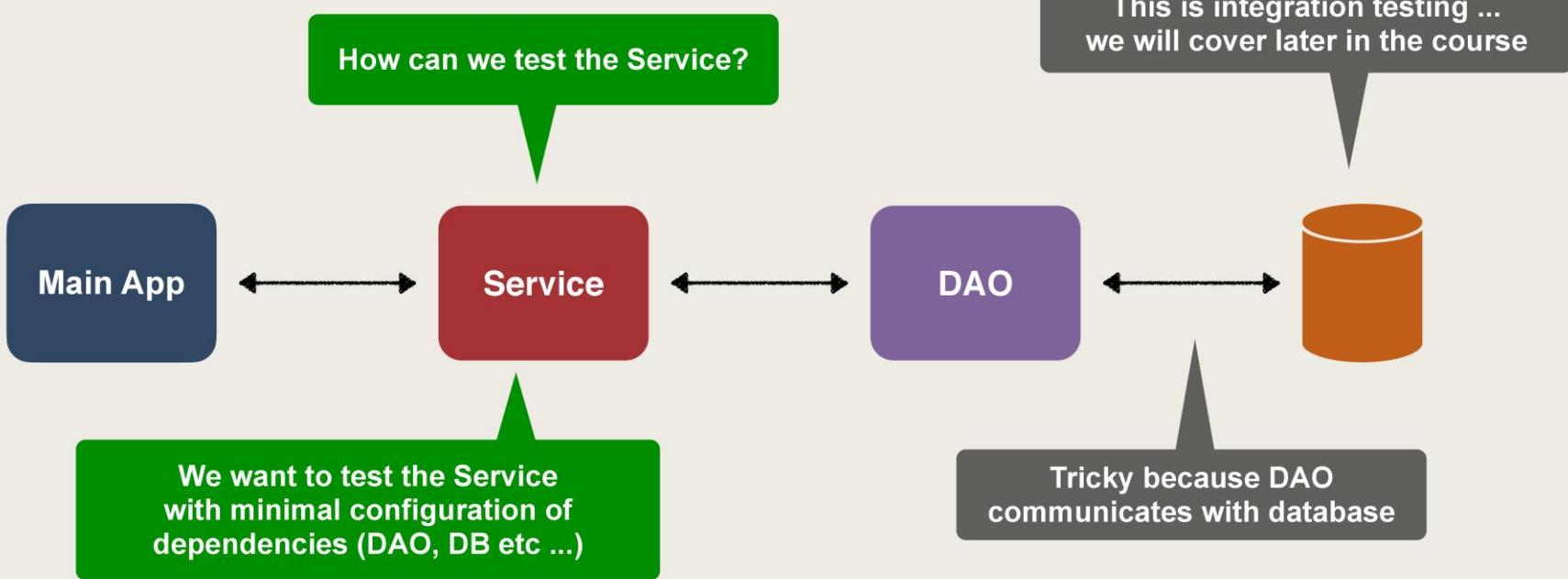


Annotations

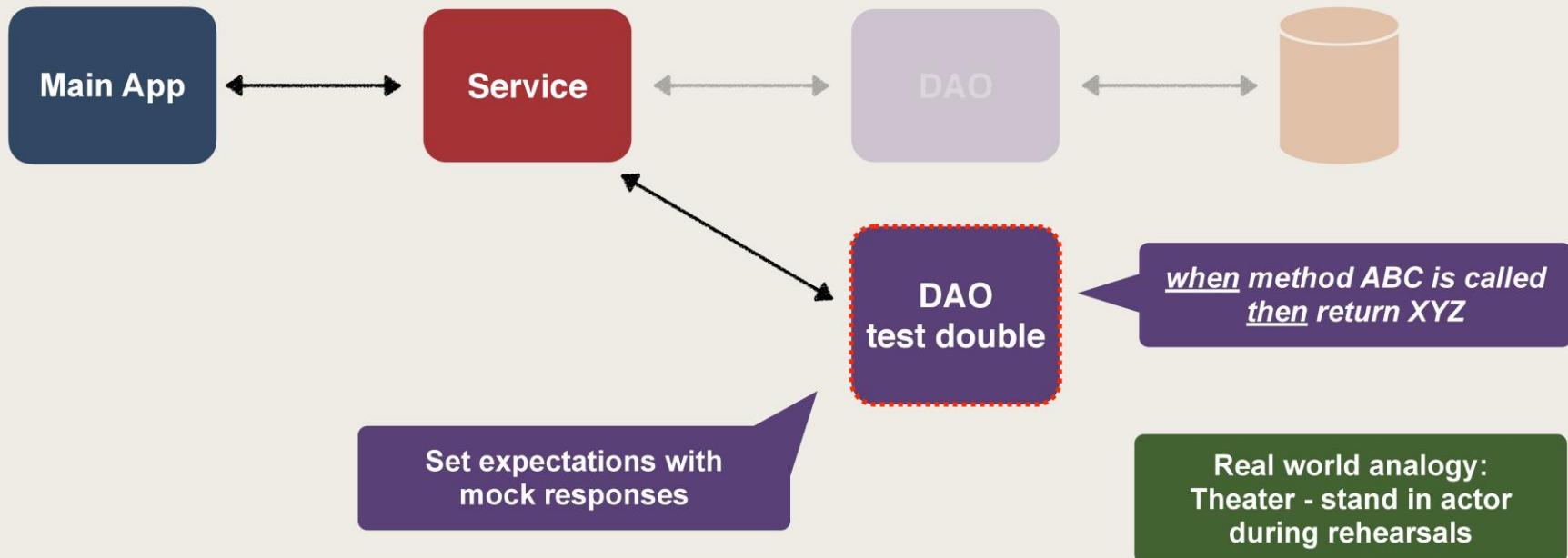
Name	Description
@EnabledIfSystemProperty	Enable test based on system property
@EnabledIfEnvironmentVariable	Enable test based on environment variable
...	...

Mockito and Spring Boot

Typical Application Architecture



Using a Test Double



**The technique of using test doubles
is known as "mocking"**

Benefits of Mocking

- Allows us to test a given class in isolation
- Test interaction between given class and its dependencies
- Minimizes configuration / availability of dependencies
- For example DAO, DB, REST API etc
 - We can mock the DAO to give a response
 - We can mock a REST API to give a response

Real world analogy:
Theater - stand in actor
during rehearsals

Mocking Frameworks

- The Java ecosystem includes a number of Mocking frameworks
- The Mocking frameworks provide following features:
 - Minimize hand-coding of mocks ... leverage annotations
 - Set expectations for mock responses
 - Verify the calls to methods including the number of calls
 - Programmatic support for throwing exceptions

Mocking Frameworks



Name	Website
Mockito	site.mockito.org
EasyMock	www.easymock.org
JMockit	jmockit.github.io
...	

We will use Mockito since
Spring Boot has built-in support for Mockito

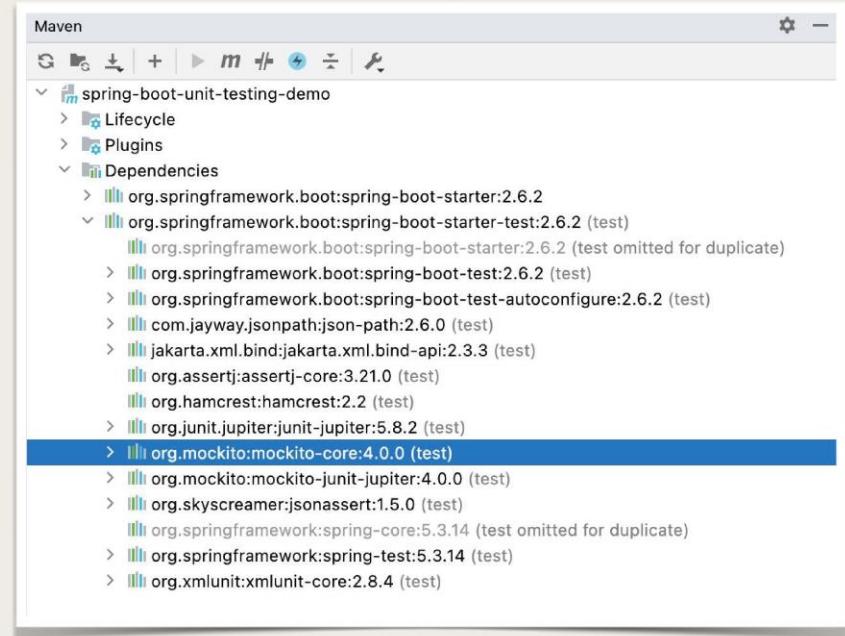
Spring Boot Starter - Transitive Dependency for Mockito

pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
```

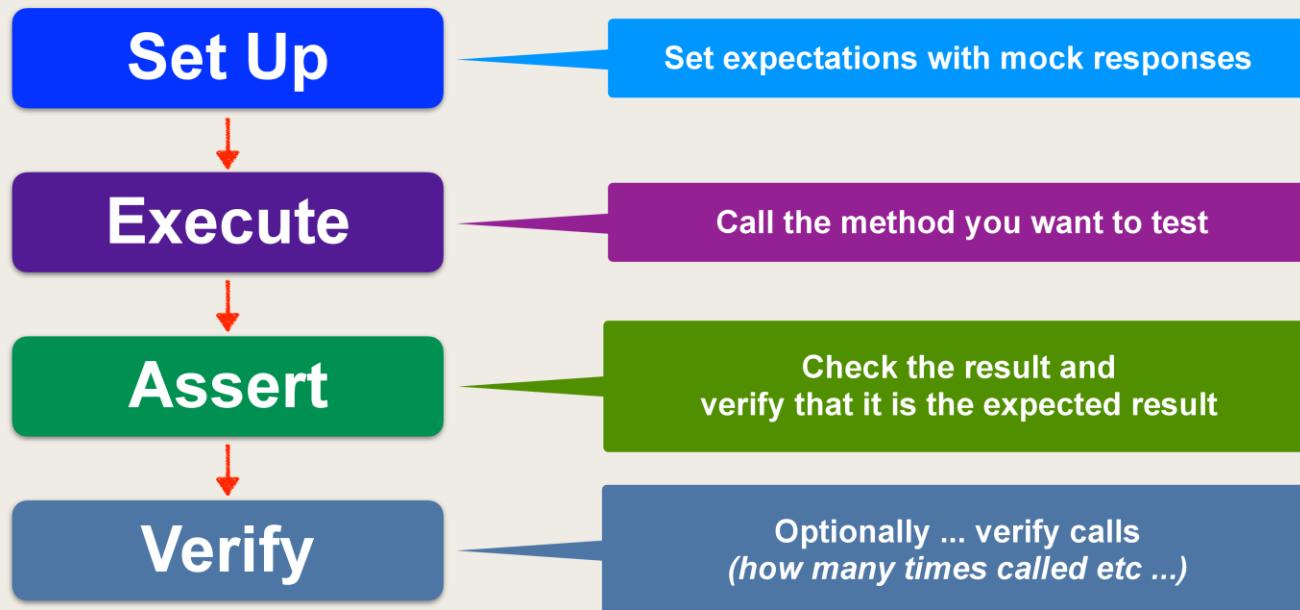
Starter includes a transitive dependency on Mockito

We get it for free :-)

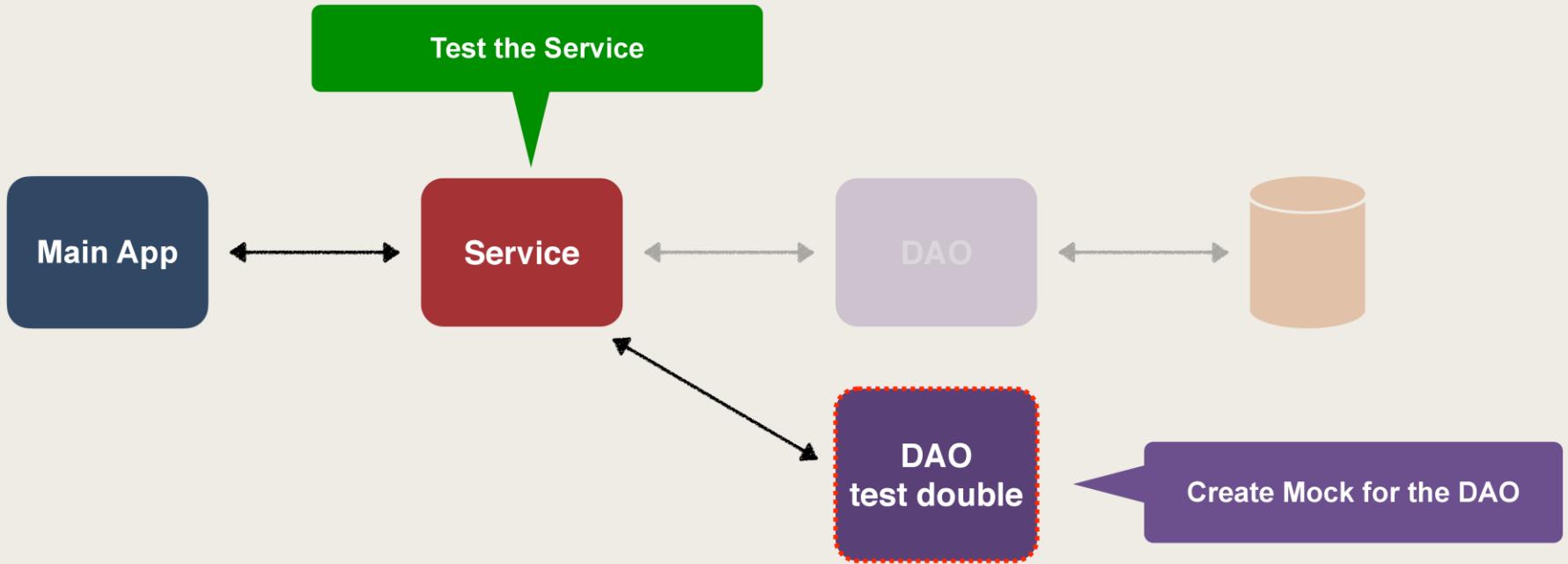


Unit testing with Mocks

- Unit tests with Mocks have the following structure



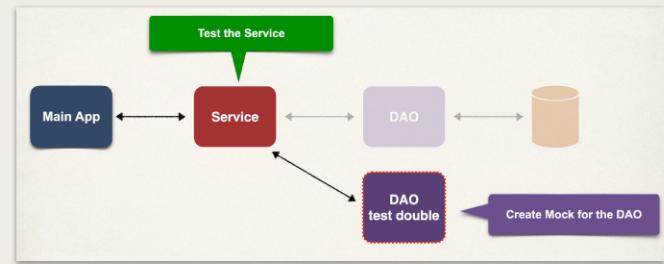
Testing Plan



Development Process

Step-By-Step

1. Create Mock for DAO
2. Inject mock into Service
3. Set up expectations
4. Call method under test and assert results
5. Verify method calls

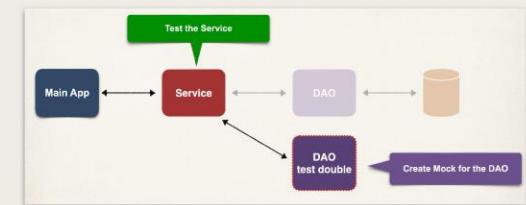


Step 1: Create Mock for the DAO

MockAnnotationTest.java

```
import org.mockito.Mock;  
...  
  
@SpringBootTest(classes=MvcTestingExampleApplication.class)  
public class MockAnnotationTest {  
  
    @Mock  
    private ApplicationDao applicationDao;  
  
    ...  
}
```

Create Mock for the DAO



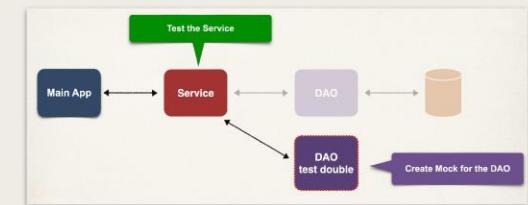
Step 2: Inject mock into Service

MockAnnotationTest.java

```
import org.mockito.Mock;
import org.mockito.InjectMocks;
...
@SpringBootTest(classes=MvcTestingExampleApplication.class)
public class MockAnnotationTest {

    @Mock
    private ApplicationDao applicationDao;

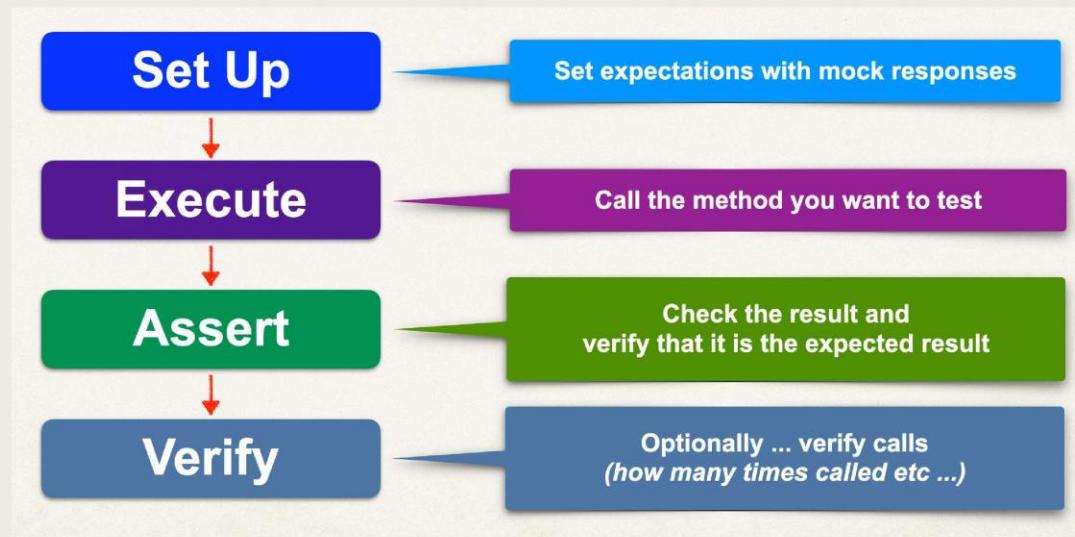
    @InjectMocks
    private ApplicationService applicationService;
    ...
}
```



Inject mock dependencies

Note: Will only inject dependencies annotated with `@Mock` or `@Spy`

Step 3: Set up expectations



Step 3: Set up expectations

*when method
doSomeWork(...)
is called
then return "I am finished"*

```
import static org.mockito.Mockito.when;  
...  
String aResponse = "I am finished";  
  
when( doSomeWork() ).thenReturn( aResponse );
```

response

method

Real world analogy:
Theater - just read the script

Step 3: Set up expectations

MockAnnotationTest.java

```
import static org.mockito.Mockito.when;
import static org.junit.jupiter.api.Assertions.assertEquals;
...

@SpringBootTest(classes=MvcTestingExampleApplication.class)
public class MockAnnotationTest {

    @Mock
    private ApplicationDao applicationDao;

    @InjectMocks
    private ApplicationService applicationService;

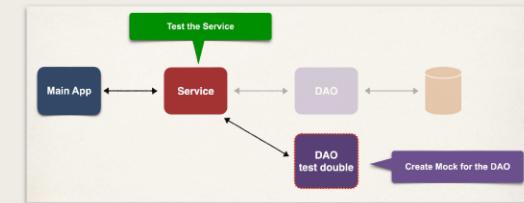
    @Autowired
    private CollegeStudent studentOne;

    @Autowired
    private StudentGrades studentGrades;

    @DisplayName("When & Verify")
    @Test
    public void assertEqualsTestAddGrades() {
        when(applicationDao.addGradeResultsForSingleClass(
            studentGrades.getMathGradeResults())).thenReturn(100.0);

        ...
    }
}
```

Set up expectations
for mock response



when method
addGradeResultsForSingleClass(...)
is called
then return 100.0

Step 4: Call method under test and assert results

MockAnnotationTest.java

```
import static org.mockito.Mockito.when;
import static org.junit.jupiter.api.Assertions.assertEquals;
...

@SpringBootTest(classes= MvcTestingExampleApplication.class)
public class MockAnnotationTest {

    @Mock
    private ApplicationDao applicationDao;

    @InjectMocks
    private ApplicationService applicationService;

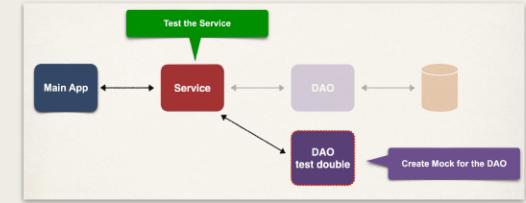
    @Autowired
    private CollegeStudent studentOne;

    @Autowired
    private StudentGrades studentGrades;

    @DisplayName("When & Verify")
    @Test
    public void assertEqualsTestAddGrades() {
        when(applicationDao.addGradeResultsForSingleClass(
            studentGrades.getMathGradeResults())).thenReturn(100.0);

        assertEquals(100.0, applicationService.addGradeResultsForSingleClass(
            studentOne.getStudentGrades().getMathGradeResults()));

    }
}
```



Assert results

*The service uses the DAO ...
that has been set up to return 100.0*

Step 5: Verify method calls

MockAnnotationTest.java

```
import static org.mockito.Mockito.when;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.times;
...
@SpringBootTest(classes= MvcTestingExampleApplication.class)
public class MockAnnotationTest {

    @Mock
    private ApplicationDao applicationDao;

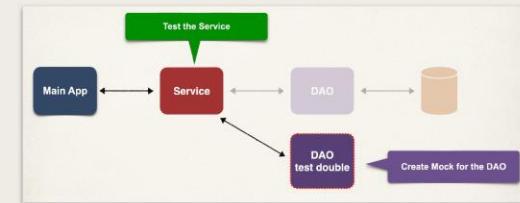
    @InjectMocks
    private ApplicationService applicationService;

    ...

    @DisplayName("When & Verify")
    @Test
    public void assertEqualsTestAddGrades() {
        when(applicationDao.addGradeResultsForSingleClass(
                studentGrades.getMathGradeResults())).thenReturn(100.0);

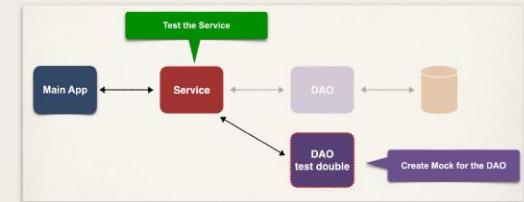
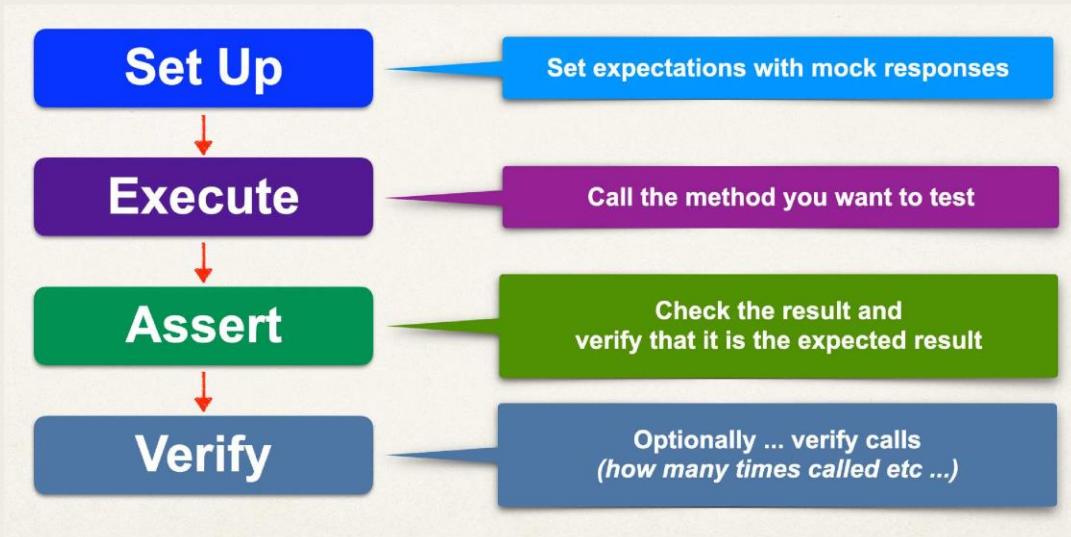
        assertEquals(100.0, applicationService.addGradeResultsForSingleClass(
                studentOne.getStudentGrades().getMathGradeResults()));

        verify(applicationDao,
            times(1)).addGradeResultsForSingleClass(studentGrades.getMathGradeResults());
    }
}
```



Verify the DAO method was called 1 time

Recap



Mockito Resources

- Additional features
- Stubs, spies
- Argument matchers, Answers
- ...

site.mockito.org

MockAnnotationTest.java

```
import org.mockito.Mock;
import org.mockito.InjectMocks;
...
@SpringBootTest(classes=MvcTestingExampleApplication.class)
public class MockAnnotationTest {

    @Mock
    private ApplicationDao applicationDao;

    @InjectMocks
    private ApplicationService applicationService;
    ...
}
```

Inject mock dependencies

Note: Will only inject dependencies annotated with @Mock or @Spy

Spring Boot @MockBean

- Instead of using Mockito: `@Mock` and `@InjectMocks`
- Use Spring Boot support: `@MockBean` and `@Autowired`
- `@MockBean`
 - includes Mockito `@Mock` functionality
 - also adds mock bean to Spring ApplicationContext
 - if existing bean is there, the mock bean will replace it
 - thus making the mock bean available for injection with `@Autowired`

Use Spring Boot `@MockBean`
when you need to inject mocks
AND
inject regular beans from app context

BEFORE

MockAnnotationTest.java

```
import org.mockito.Mock;
import org.mockito.InjectMocks;
...
@SpringBootTest(classes=MvcTestingExampleApplication.class)
public class MockAnnotationTest {

    @Mock
    private ApplicationDao applicationDao;

    @InjectMocks
    private ApplicationService applicationService;

    ...
}
```

AFTER

MockAnnotationTest.java

```
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.beans.factory.annotation.Autowired;
...
@SpringBootTest(classes=MvcTestingExampleApplication.class)
public class MockAnnotationTest {

    @MockBean
    private ApplicationDao applicationDao;

    @Autowired
    private ApplicationService applicationService;

    ...
}
```

Create Mock for the DAO

Inject dependencies

No longer a limitation ...
can inject any dependency ...
mock or regular bean

Spring Boot support for Unit Testing

What do you need for Spring Boot unit testing?

- Access to the Spring Application Context
- Support for Spring dependency injection
- Retrieve data from Spring application.properties
- Mock object support for web, data, REST APIs etc ...

Unit Testing support in Spring Boot

- Spring Boot provides rich testing support
- **@SpringBootTest**
 - Loads the application context
 - Support for Spring dependency injection
 - You can access data from Spring application.properties
 - ...

Spring Boot Starter - for Testing support

- Add Maven dependency

pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
```

Starter includes a transitive
dependency on JUnit 5

We get it for free :-)

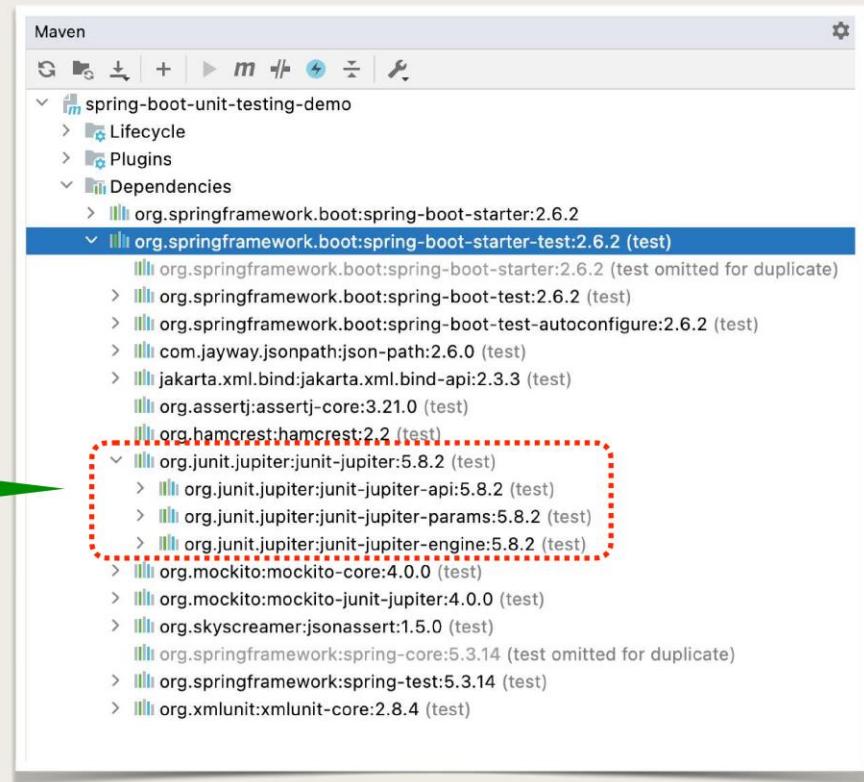
Spring Boot Starter - Transitive Dependency for JUnit 5

pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
```

Starter includes a transitive dependency on JUnit 5

We get it for free :-)



Spring Boot - Testing

Spring Boot Starter - Transitive Dependency for JUnit 5

At command-line, type:

Starter includes a transitive dependency on JUnit 5

We get it for free :-)

```
mvn dependency:tree
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.luv2code:spring-boot-unit-testing-demo >-----
[INFO] Building spring-boot-unit-testing-demo 1.0.0
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-dependency-plugin:3.2.0:tree (default-cli) @ spring-boot-unit-testing-demo ---
[INFO] com.luv2code:spring-boot-unit-testing-demo:jar:1.0.0
[INFO] ...
[INFO]   \- org.springframework.boot:spring-boot-starter-test:jar:2.6.2:test
[INFO]     \- org.junit.jupiter:junit-jupiter:jar:5.8.2:test
[INFO]       +- org.junit.jupiter:junit-jupiter-api:jar:5.8.2:test
[INFO]       +- org.junit.jupiter:junit-jupiter-params:jar:5.8.2:test
[INFO]       \- org.junit.jupiter:junit-jupiter-engine:jar:5.8.2:test
[INFO] ...
[INFO]
```

Spring Boot Test

ApplicationExampleTest.java

```
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
public class ApplicationExampleTest {

    @Test
    void basicTest() {
        // ...
    }

}
```

Loads Spring Application Context

Inject Spring Beans

ApplicationExampleTest.java

```
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.beans.factory.annotation.Autowired;

@SpringBootTest
public class ApplicationExampleTest {

    @Autowired
    StudentGrades studentGrades;

    @Test
    void basicTest() {
        // ...
    }
}
```

Injection

```
@Component
public class StudentGrades {
    ...
}
```

Access Application Properties

ApplicationExampleTest.java

```
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.beans.factory.annotation.Value;

@SpringBootTest
public class ApplicationExampleTest {

    @Value("${info.school.name}")
    private String schoolName;

    @Value("${info.app.name}")
    private String appInfo;

    @Test
    void basicTest() {
        // ...
    }
}
```

application.properties

```
info.school.name=luv2code
info.app.name=My Super Cool Gradebook
```

Access data from
application.properties

Access Application Context

ApplicationExampleTest.java

```
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;

@SpringBootTest
public class ApplicationExampleTest {

    @Autowired
    ApplicationContext context;

    @Test
    void basicTest() {
        // ...
    }
}
```



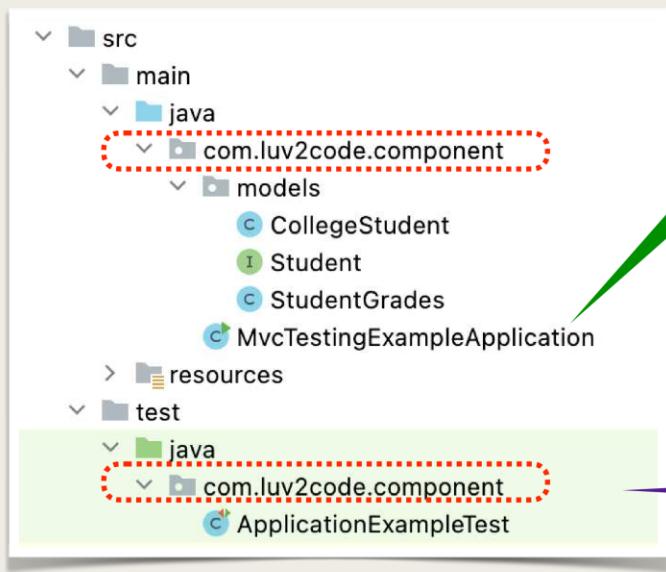
Access Spring Application Context

@SpringBootTest configuration

**Place your test class in test package
same as your main package**

- This implicitly defines a base search
 - Allows you to leverage default configuration
 - No need to explicitly reference the main Spring Boot application class

More on Configuration



Main Spring Boot application class

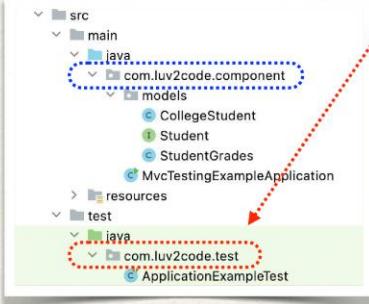
Automatically component scans sub-packages

Same package names

... no additional config required ...

More on Configuration

- Default configuration is fine if everything is under
 - **com.luv2code.component**
- But what if test class is in a different package???
- **com.luv2code.test**



```
package com.luv2code.test;  
...  
@SpringBootTest(classes = MvcTestingExampleApplication.class)  
public class ApplicationExampleTest {  
    ...  
}
```

Explicitly reference
main SpringBoot class

REST API Testing - Overview

Technical Stack

- Spring Boot
- Spring Data JPA
- Spring @RestController

Spring Boot - Testing

Existing Code (partial)

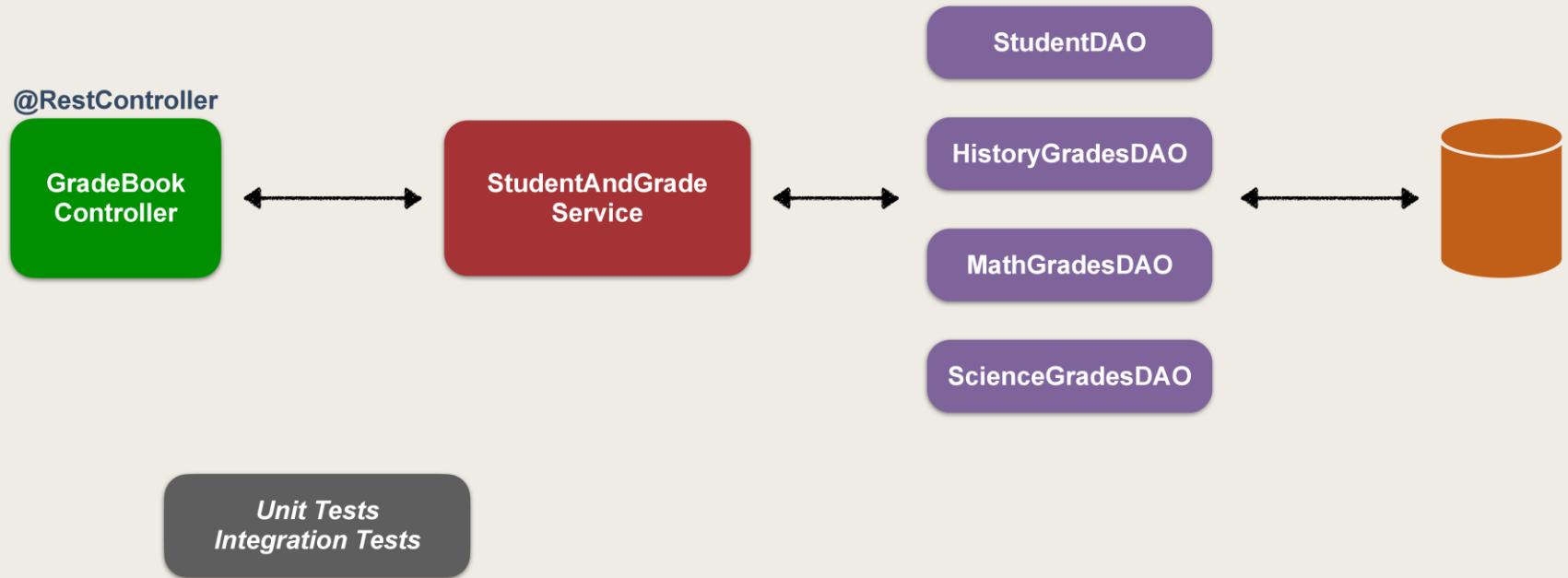
@RestController

GradeBookController.java

Model

CollegeStudent.java
Grade.java
Gradebook.java
GradebookCollegeStudent.java
HistoryGrade.java
MathGrade.java
ScienceGrade.java
Student.java
StudentGrades.java

Application Architecture



Spring Boot - Testing

Problem

- How can we test REST API developed with Spring REST Controllers?
- How can we create HTTP requests and send to the Spring REST controller?
- How can we verify HTTP response?
 - status code
 - content type
 - JSON response body

Spring Testing Support

- For testing Spring REST controllers, you can use `MockMvc`
- Provides Spring REST processing of request / response
- There is no need to run a server (embedded or external)

In general, the process is **VERY similar** to Spring MVC testing

Slight differences for content-type and checking JSON results

Preliminary Set Up

- For GradebookControllerTest
- Stub out the test class
- Define fields that we'll use later: MockMvc, Service, DAOs etcs
- @BeforeAll, @BeforeEach, @AfterEach

GradebookControllerTest.java

```
...
@TestPropertySource("/application-test.properties")
@AutoConfigureMockMvc
@SpringBootTest
public class GradebookControllerTest {

    // inject support utils

    // inject Service and DAOs

    // inject SQL strings

    // @BeforeAll, @BeforeEach and @AfterEach

}
```

Get Students Endpoint

- Get a list of students as a JSON array

http://localhost:1500/

Web browser will send a GET request

```
[  
  {  
    "id": 10,  
    "firstname": "David",  
    "lastname": "Adams",  
    "emailAddress": "david@luv2code.com",  
    "studentGrades": {  
      "mathGradeResults": [],  
      "scienceGradeResults": [],  
      "historyGradeResults": []  
    },  
    "fullName": "David Adams"  
  },  
  {  
    "id": 11,  
    "firstname": "John",  
    "lastname": "Doe",  
    "emailAddress": "john@luv2code.com",  
    "studentGrades": {  
      "mathGradeResults": [],  
      "scienceGradeResults": [],  
      "historyGradeResults": []  
    },  
    "fullName": "John Doe"  
  },  
  ...  
]
```

GradebookController

```
@RestController
public class GradebookController {

    @Autowired
    private StudentAndGradeService studentService;

    @Autowired
    private Gradebook gradebook;

    @RequestMapping(value = "/", method = RequestMethod.GET)
    public List<GradebookCollegeStudent> getStudents() {
        gradebook = studentService.getGradebook();
        return gradebook.getStudents();
    }

    ...
}
```



```
[
  {
    "id": 10,
    "firstname": "David",
    "lastname": "Adams",
    "emailAddress": "david@luv2code.com",
    "studentGrades": [
      "mathGradeResults": [],
      "scienceGradeResults": [],
      "historyGradeResults": []
    ],
    "fullName": "David Adams"
  },
  {
    "id": 11,
    "firstname": "John",
    "lastname": "Doe",
    "emailAddress": "john@luv2code.com",
    "studentGrades": [
      "mathGradeResults": [],
      "scienceGradeResults": [],
      "historyGradeResults": []
    ],
    "fullName": "John Doe"
  },
  ...
]
```

Verifying the HTTP Response

- How can we verify HTTP response?
 - status code
 - content type
 - JSON response body

Verifying HTTP Status and Content Type

```
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
...
@TestPropertySource("/application-test.properties")
@AutoConfigureMockMvc
@SpringBootTest
public class GradebookControllerTest {
    ...
    @Test
    public void getStudentsHttpRequest() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.get("/"))
            .andExpect(status().isOk())
            .andExpect(content().contentType(APPLICATION_JSON_UTF8));
    }
    ...
}
```

HTTP status of 200 (OK)

application/json

Verifying JSON Response Body

- How can we verify JSON response body?
 - Access specific JSON element (even for nested elements)
 - Size of the JSON array

Solution: JsonPath

- **JsonPath** allows you to access elements of JSON
- Open-source project

<https://github.com/json-path/JsonPath>

JsonPath Examples

JsonPath	Result
\$	The root element to query. Starts all path expressions
\$.id	Access the id element of the JSON element
\$.firstname	Access the firstname element of the JSON element
...	...

```
{  
  "id": 10,  
  "firstname": "David",  
  "lastname": "Adams",  
  "emailAddress": "david@luv2code.com",  
  "studentGrades": {  
    "mathGradeResults": [],  
    "scienceGradeResults": [],  
    "historyGradeResults": []  
  },  
  "fullName": "David Adams"  
}
```

<https://github.com/json-path/JsonPath>

JsonPath - Verifying Array Size

```
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;  
import static org.hamcrest.Matchers.hasSize;  
  
@  
@  
@  
public class GradebookControllerTest {  
  
    ...  
  
    @Test  
    public void getStudentsHttpRequest() throws Exception {  
  
        mockMvc.perform(MockMvcRequestBuilders.get("/"))  
            .andExpect(status().isOk())  
            .andExpect(content().contentType(APPLICATION_JSON_UTF8))  
            .andExpect(jsonPath("$", hasSize(2));  
    }  
    ...  
}
```

Root element

Verify JSON array size is 2

```
[  
  {  
    "id": 10,  
    "firstname": "David",  
    "lastname": "Adams",  
    "emailAddress": "david@luv2code.com",  
    "studentGrades": {  
      "mathGradeResults": [],  
      "scienceGradeResults": [],  
      "historyGradeResults": []  
    },  
    "fullName": "David Adams"  
  },  
  {  
    "id": 11,  
    "firstname": "John",  
    "lastname": "Doe",  
    "emailAddress": "john@luv2code.com",  
    "studentGrades": {  
      "mathGradeResults": [],  
      "scienceGradeResults": [],  
      "historyGradeResults": []  
    },  
    "fullName": "John Doe"  
  }]  
]
```

Thank you

All the best