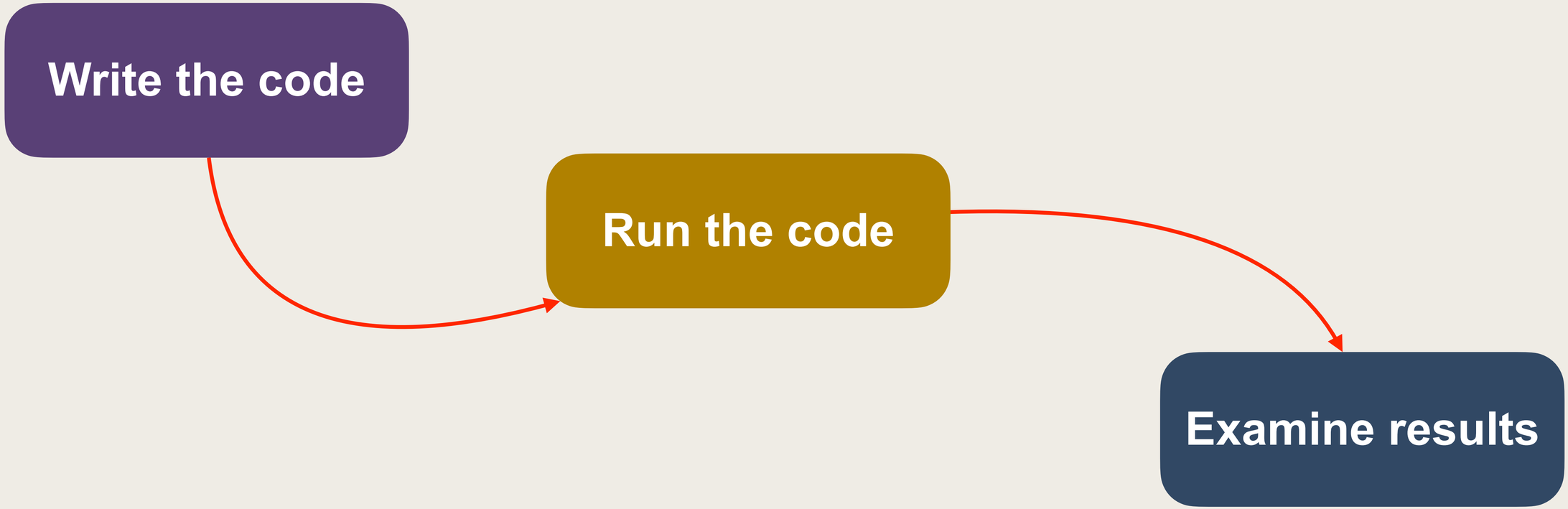
luv2code

Unit Testing Overview

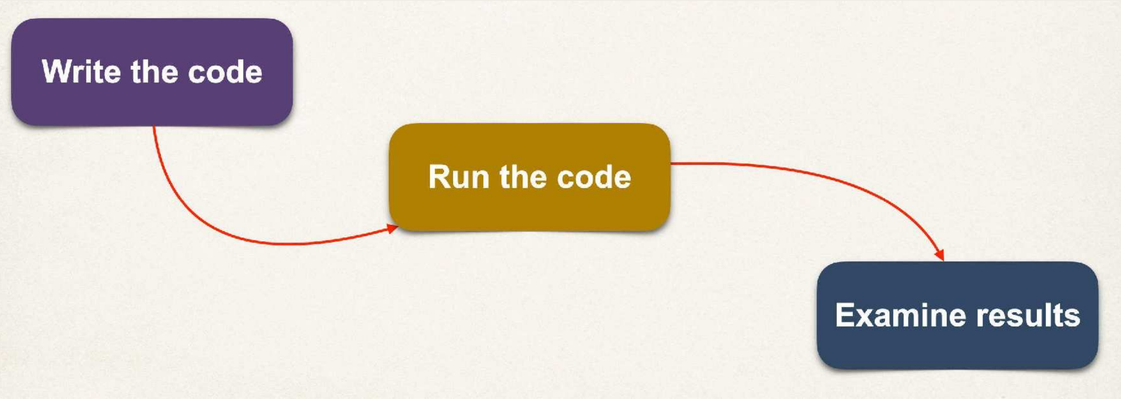
How Do We Test Code?

- Manual testing ... maybe this???



What's Wrong?

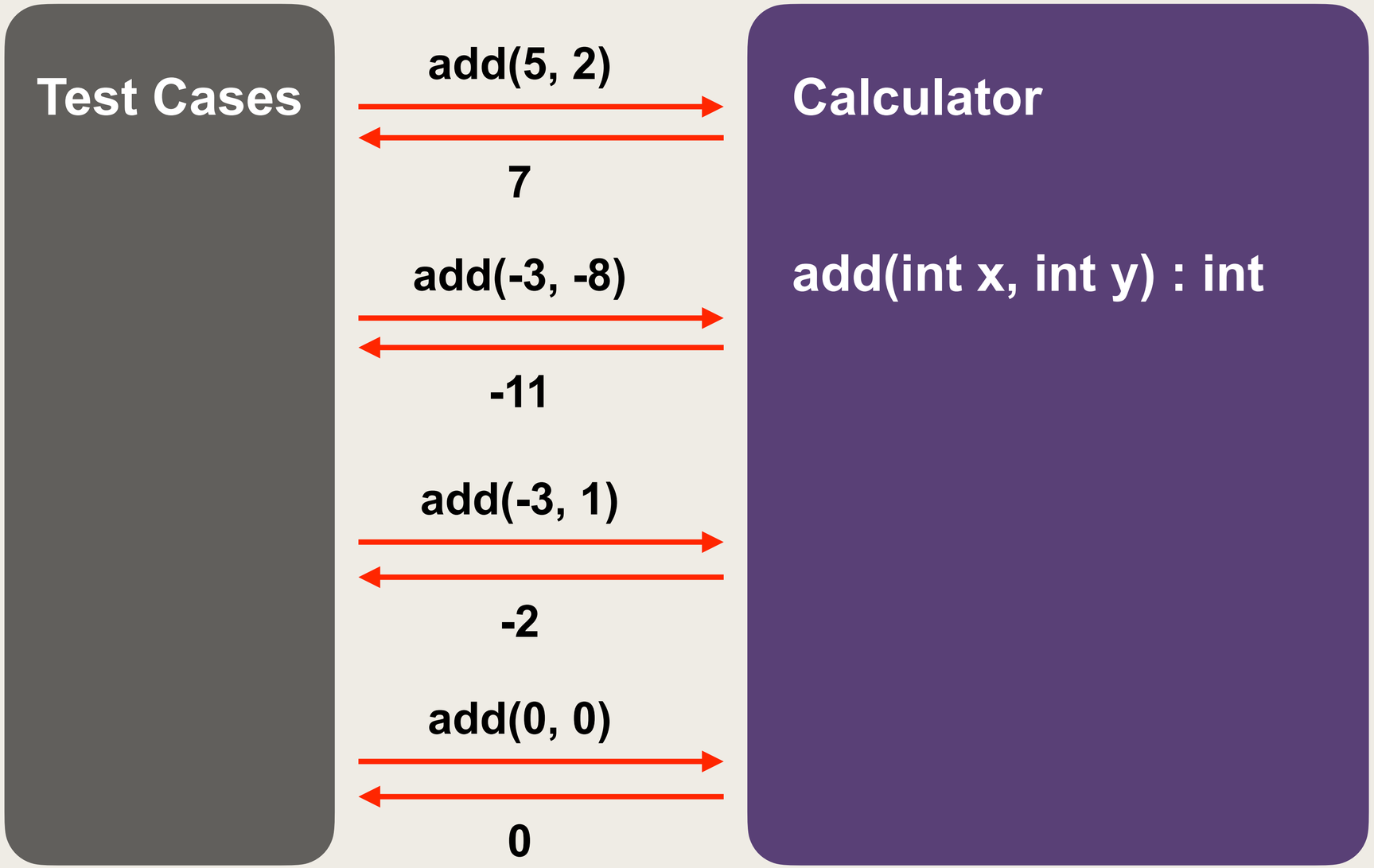
- Requires human interaction and analysis
- Test case is not reproducible
- No way to automate the test

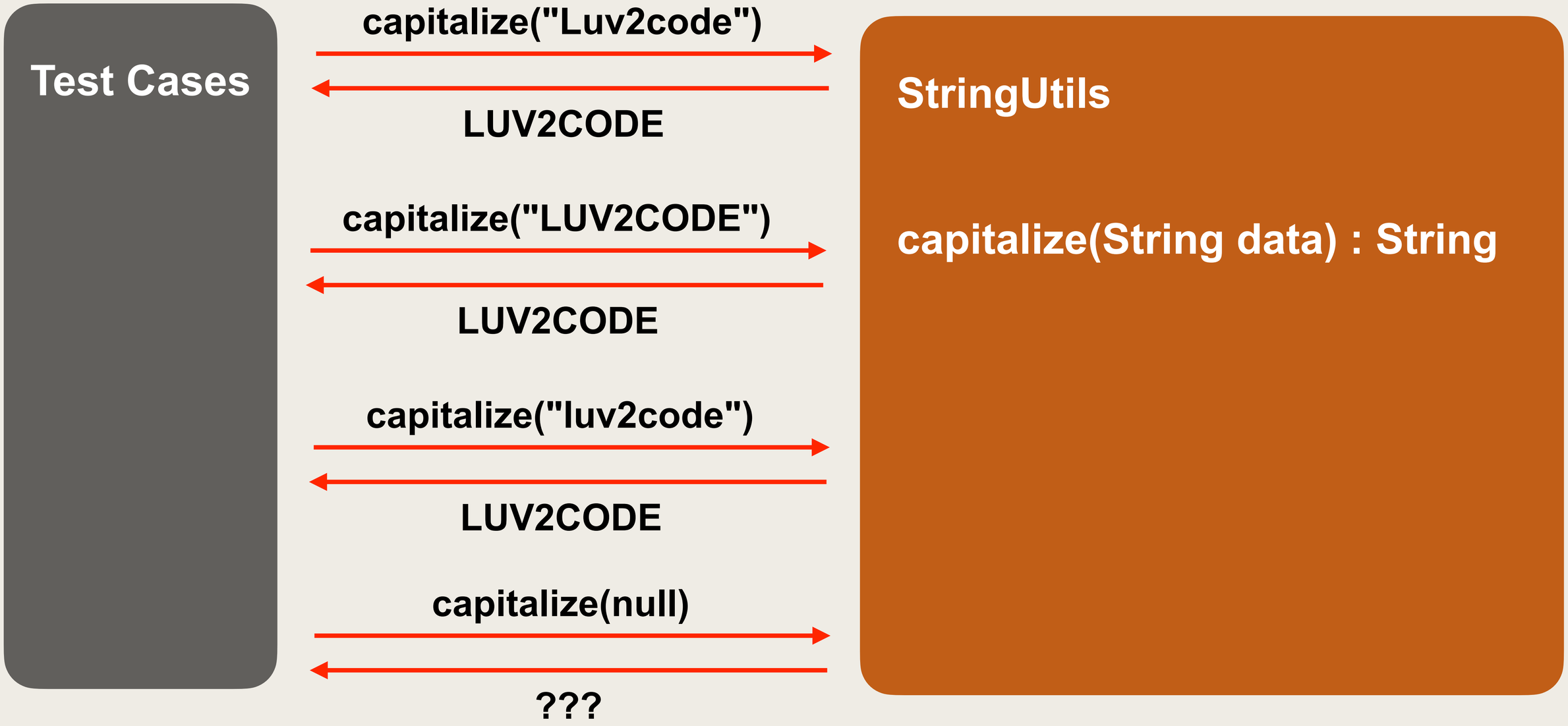


- What if I wanted to
  - Run a battery of tests for each code commit
  - Produce a report of test results

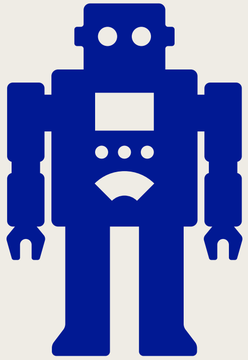
What is Unit Testing?

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





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)

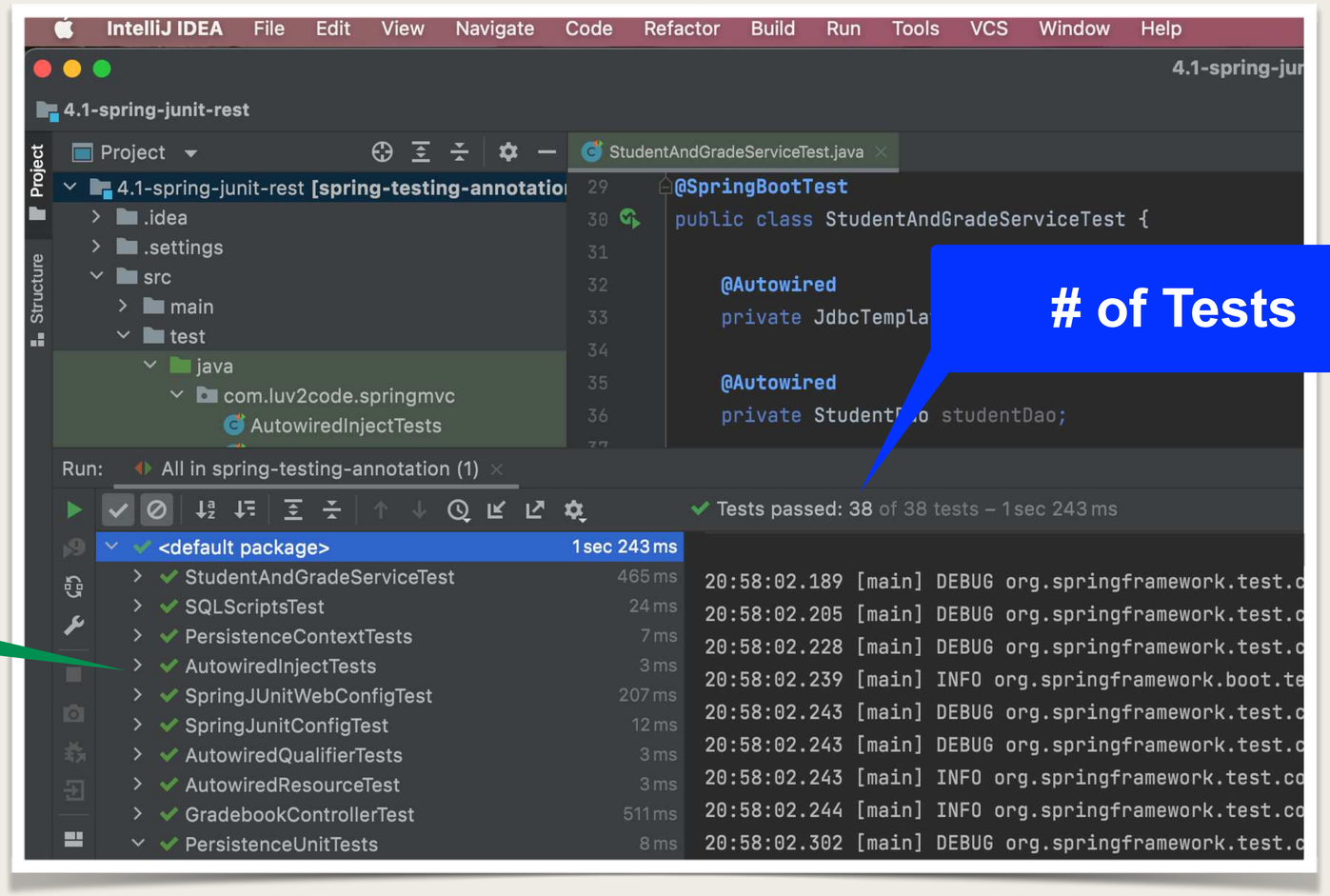
Unit Testing Frameworks

- 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

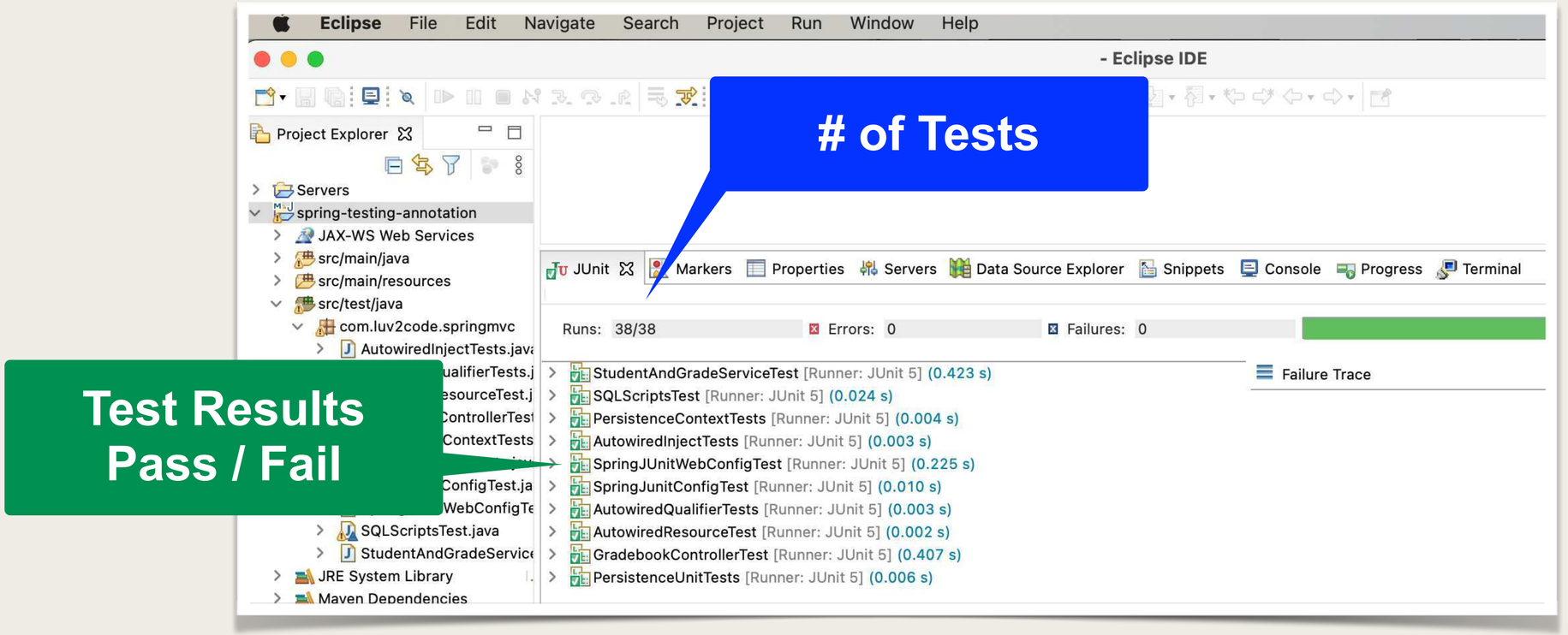
Unit Testing Tooling Support

- IntelliJ



Unit Testing Tooling Support

- Eclipse

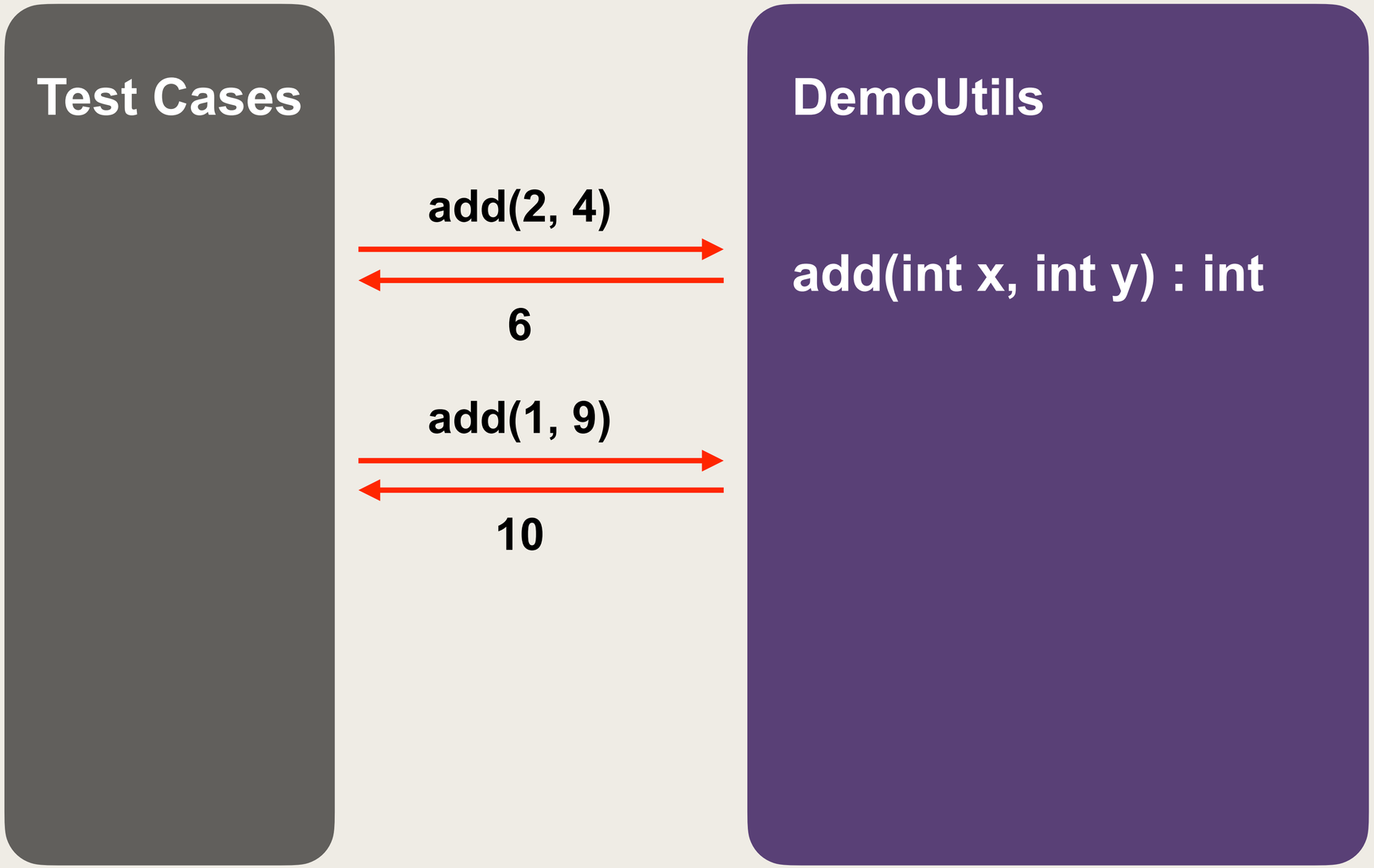


luv2code

Getting Started with JUnit

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)



Code to Test

DemoUtils.java

```
package com.luv2code.junitdemo;

public class DemoUtils {

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

}
```

Development Process

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

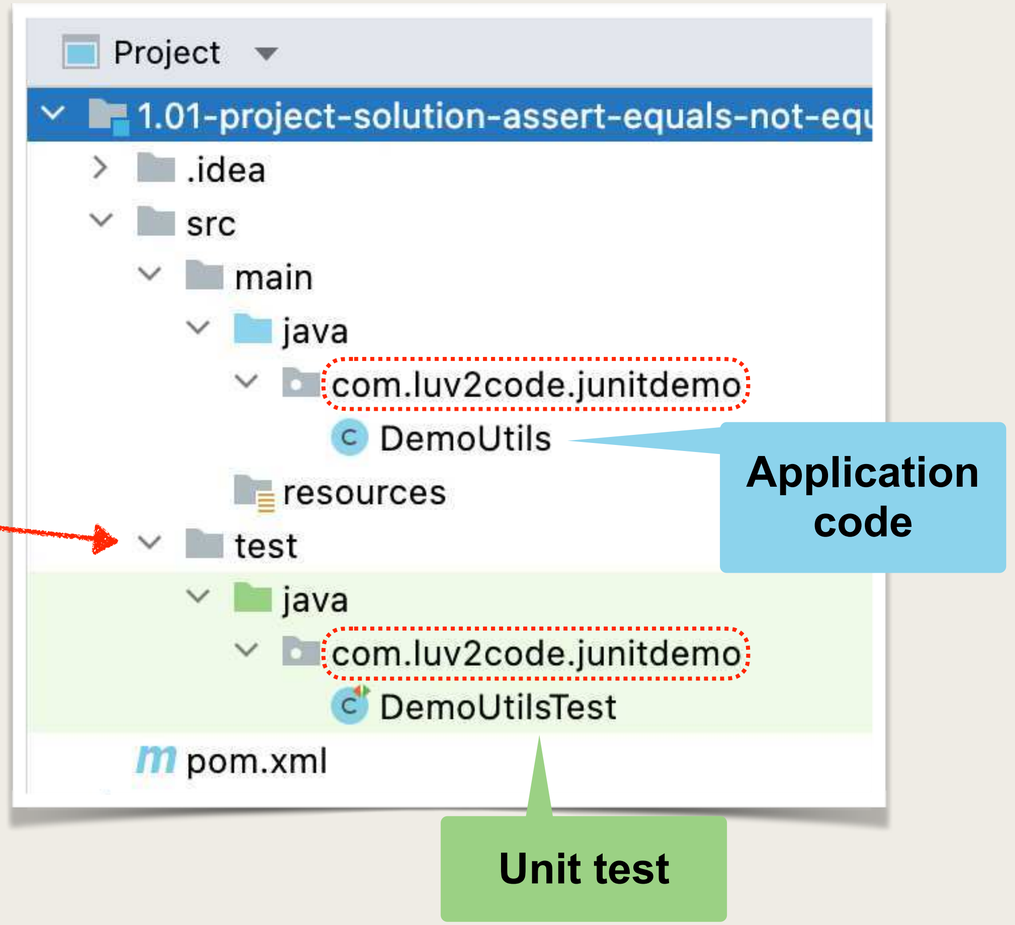
Step 1: Add Maven dependencies for JUnit

pom.xml

```
...

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

...
```

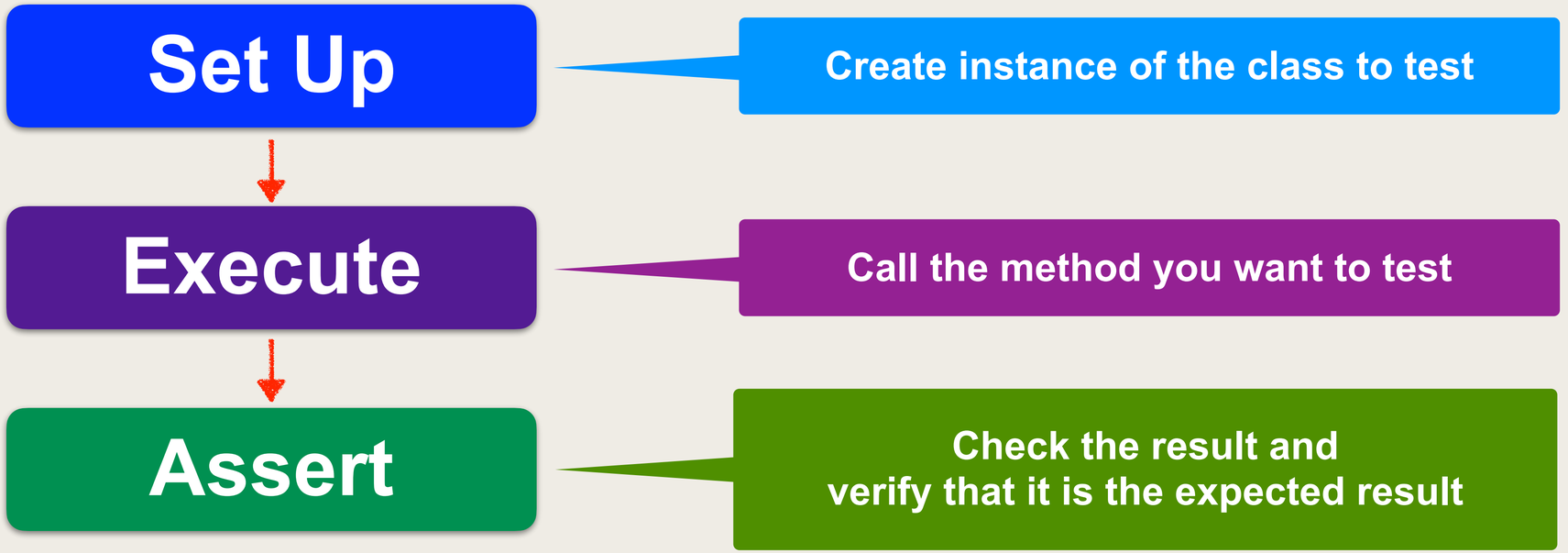
Step 2: Create test package

- 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

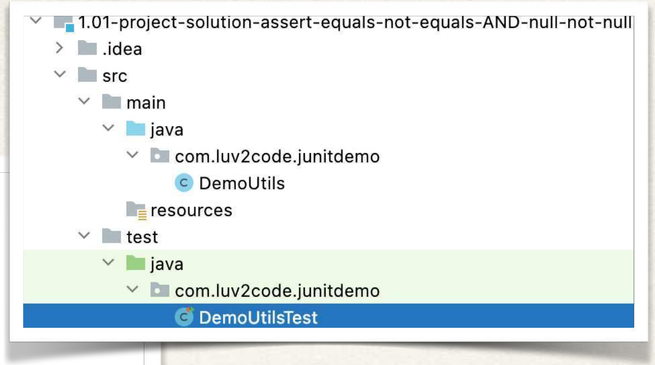


Step 3: Create unit test

- 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 testEqualsAndNotEquals() {

        // 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");
    }

}
```

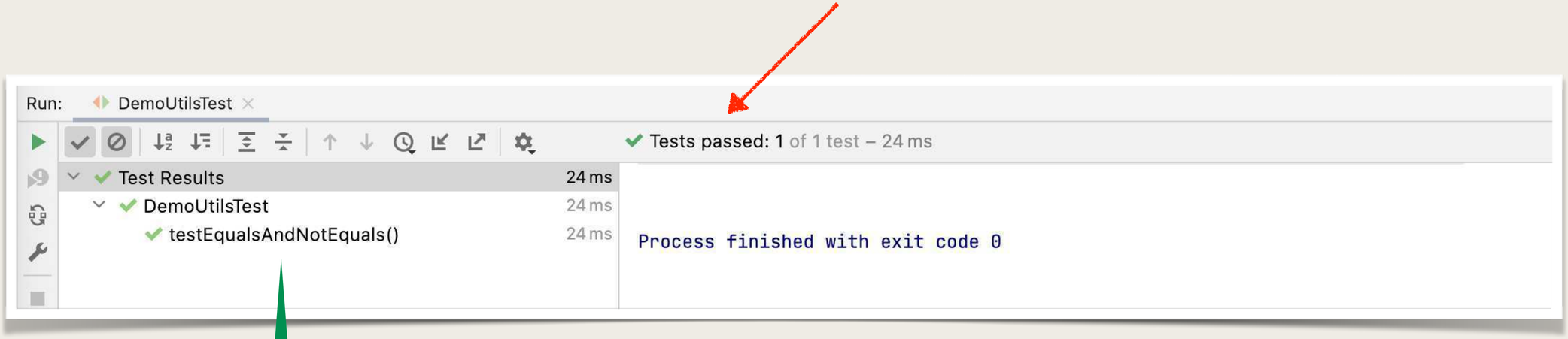
Annotate the method as a test method

Create instance of the class to test

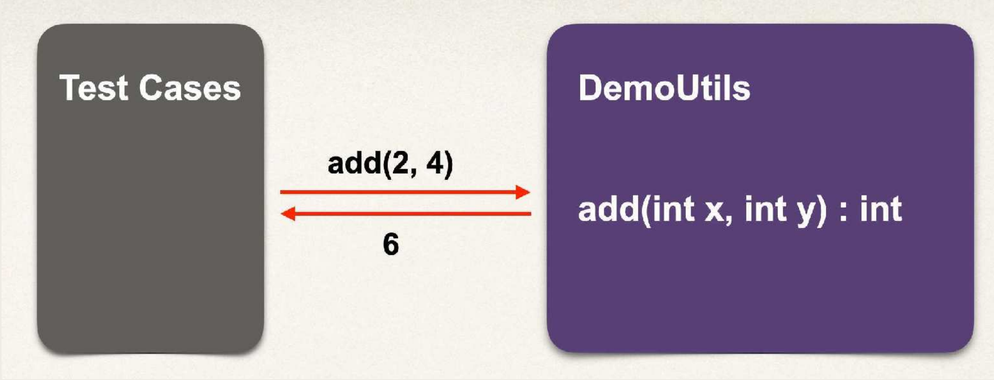
Call the method you want to test

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

Step 4: Run unit test



Test passed



IntelliJ Unit Testing Support

- IntelliJ provides additional unit testing support
  - Build dependency management: Maven and Gradle
  - Auto creation of test classes and test method stubs
  - ...

https://www.jetbrains.com/help/idea/junit.html

luv2code

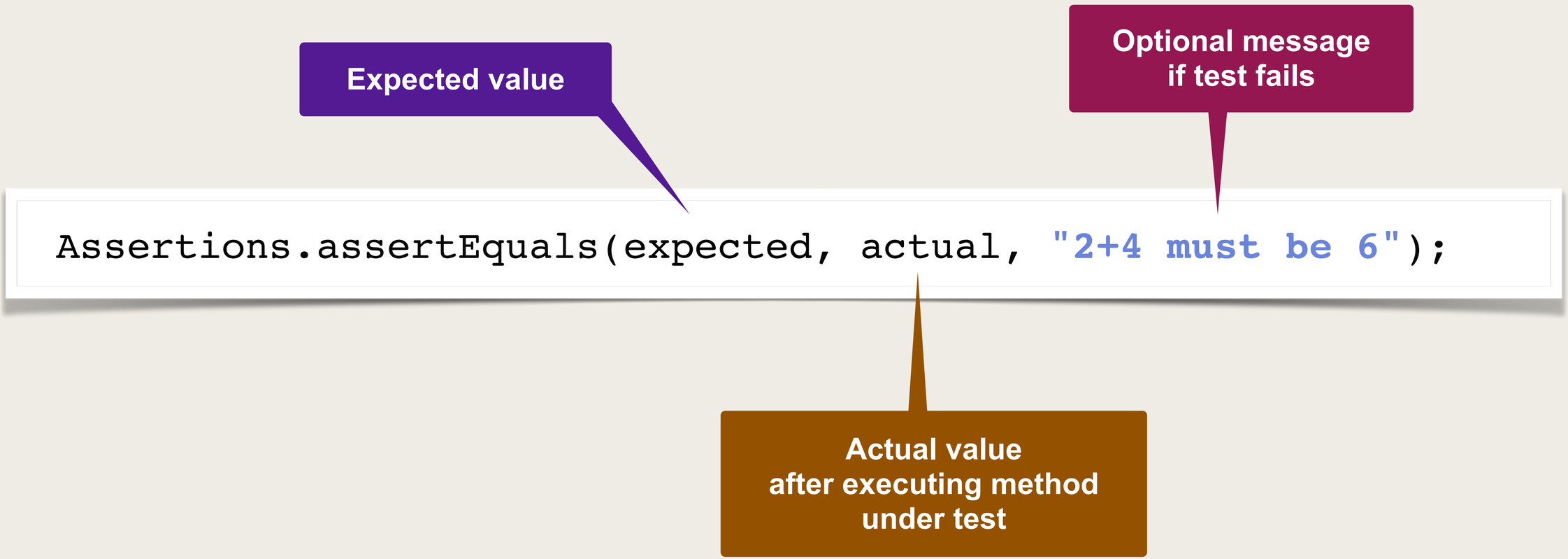
JUnit Assertions

JUnit Assertions

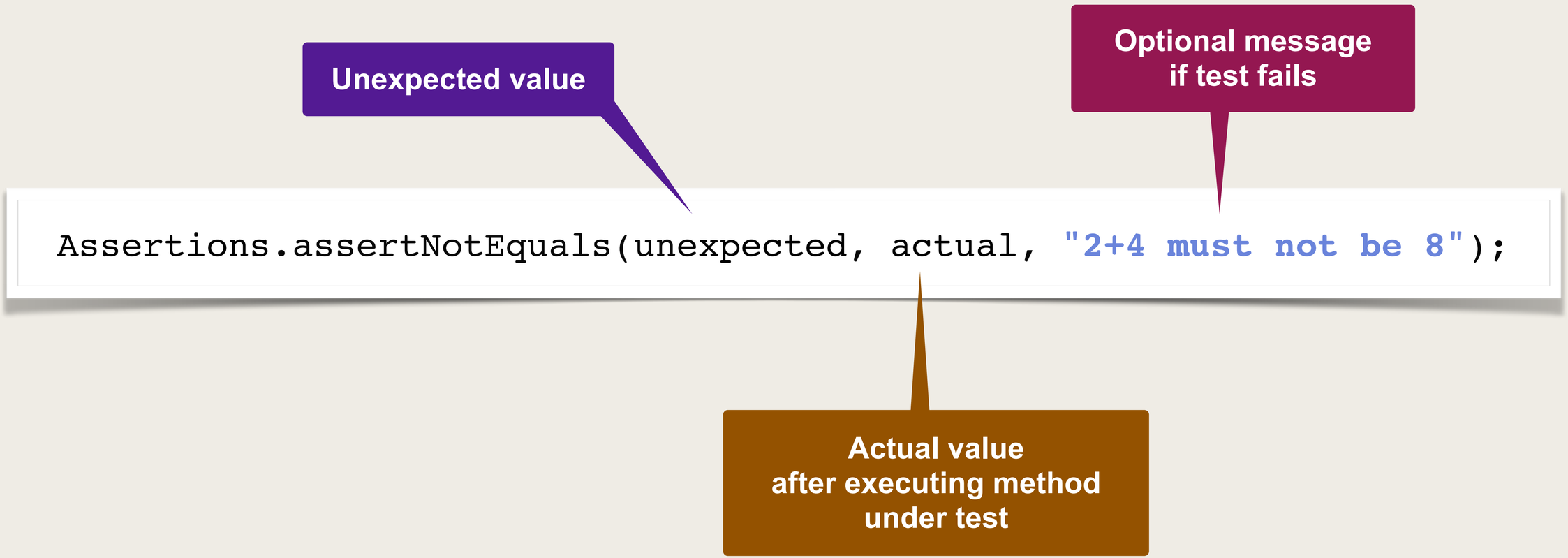
- JUnit has a collection of assertions
- Defined in class: `org.junit.jupiter.api.Assertions`

Method name	Description
`assertEquals`	Assert that the values are equal
`assertNotEquals`	Assert that the values are not equal
`assertNull`	Assert that the value is null
`assertNotNull`	Assert that the value is not null
...	...

assertEquals



assertNotEquals



DemoUtilsTest.java

```
package com.luv2code.junitdemo;

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

class DemoUtilsTest {

   @Test
   void testEqualsAndNotEquals() {

       // 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

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");
```

Just give method name

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");
```

AFTER

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

   ...

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

Wildcard

Just give method name

Short-cut for
static methods & fields

Static import

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();

       int expected = 6;
       int unexpected = 8;

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

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

   }

}
```

Just give
method name

Restructuring code

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

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

class DemoUtilsTest {

    @Test
    void testEqualsAndNotEquals() {

        // setup
        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");

    }

}
```

Expected value

Optional message if test fails

Actual value after executing method under test

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

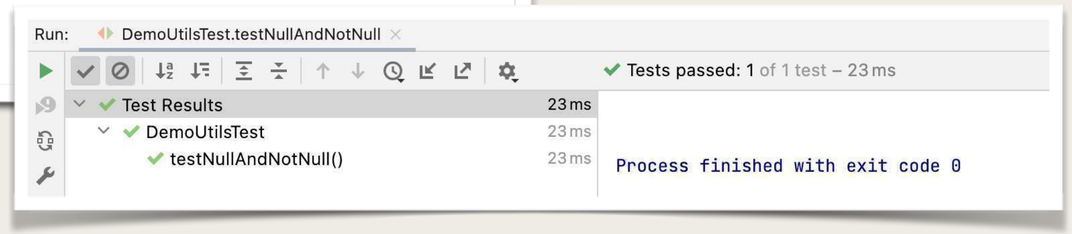
Method name	Description
`assertNull`	Assert that the value is null
`assertNotNull`	Assert that the value is not null
...	...

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

    }

}
```

Optional message if test fails

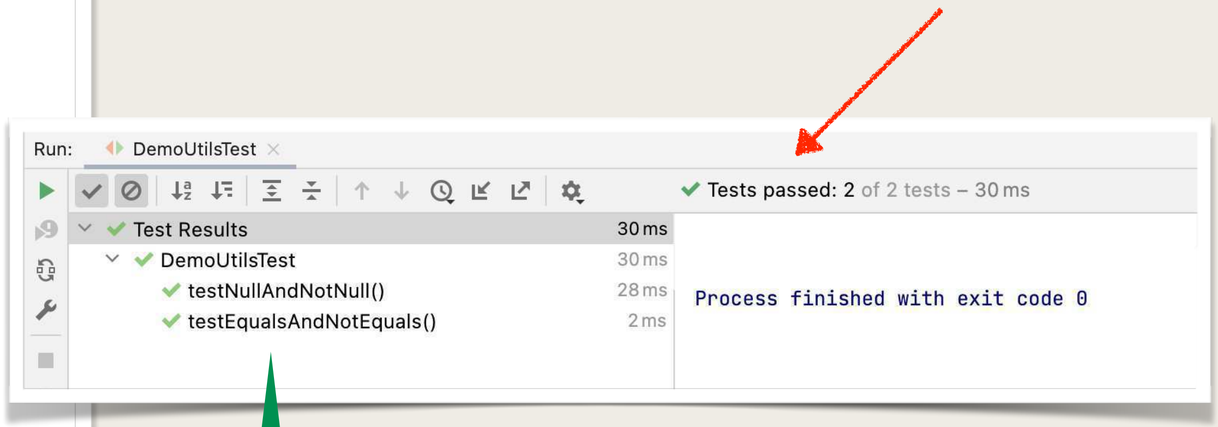
Actual value after executing method under test



Multiple Tests in One Class

DemoUtilsTest.java

```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

More Details

- JUnit User Guide

https://junit.org/junit5/docs/current/user-guide/#writing-tests

luv2code

JUnit Lifecycle Methods

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

BEFORE

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

}
```

DemoUtilsTest.java

AFTER

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

Create object before each test

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

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

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.AfterEeach;
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 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.checkNull(str1), "Object should be null");
        assertNotNull(demoUtils.checkNull(str2), "Object should not be null");
    }
}
```

Run after each test

DemoUtilsTest.java

```java
package com.luv2code.junitdemo;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.AfterEeach;
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 testEqualsAndNotEquals() {

        System.out.println("Running test: 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() {

        System.out.println("Running test: testNullAndNotNull");


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

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

Adding println(...) for diagnostics

Adding println(...) for diagnostics

Execution Sequence

@BeforeEach

@Test Method One

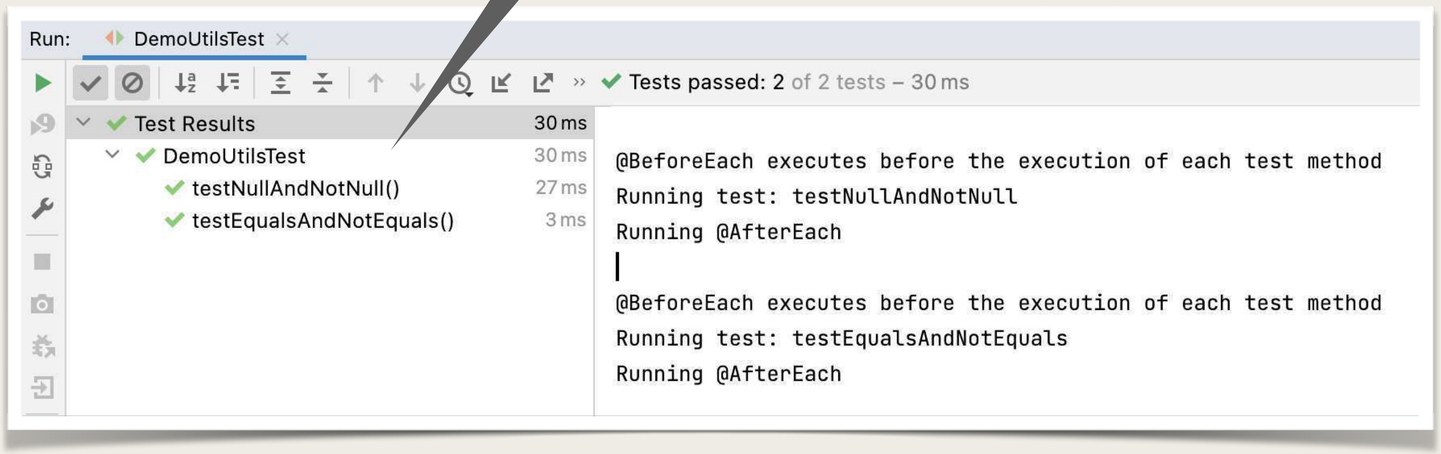
@AfterEach

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

...
```

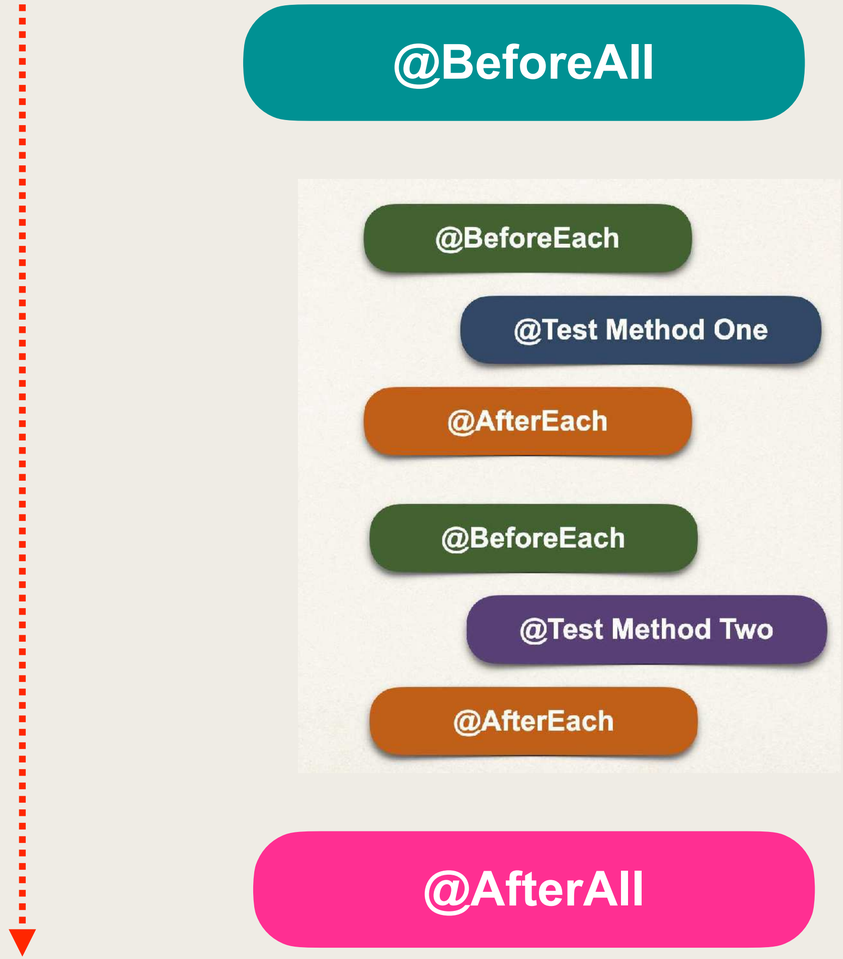
Lifecycle Methods

- 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. *Useful for getting database connections, connecting to servers ...*
`@AfterAll`	Method is executed only once, after all test methods. *Useful for releasing database connections, disconnecting from servers ...*
...	...

Execution Sequence



DemoUtilsTest.java

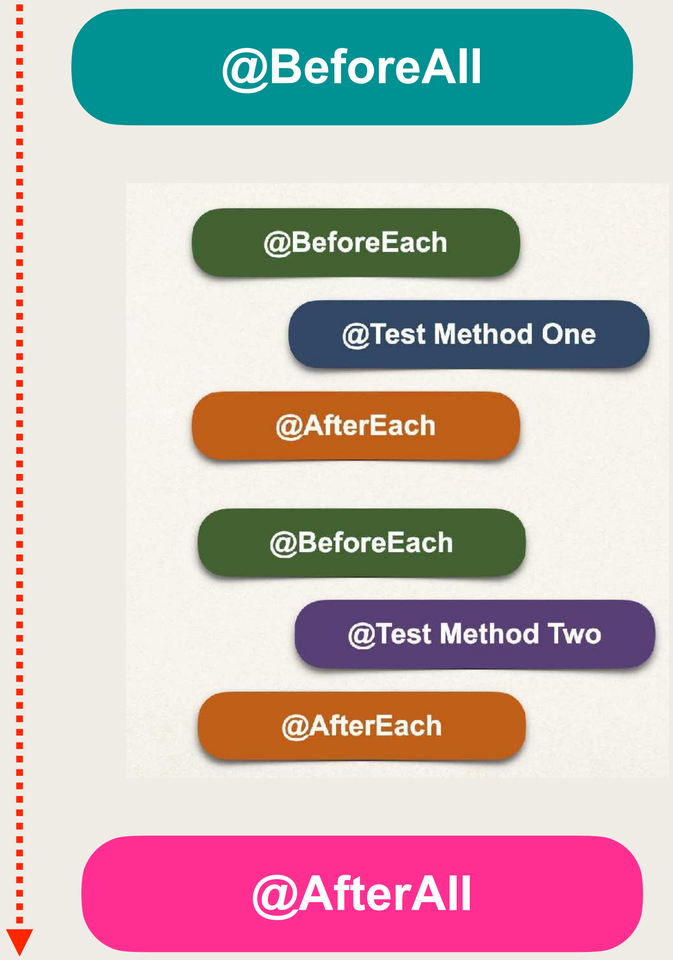
```java
package com.luv2code.junitdemo;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.AfterEeach;
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");
        ...
    }
}
```

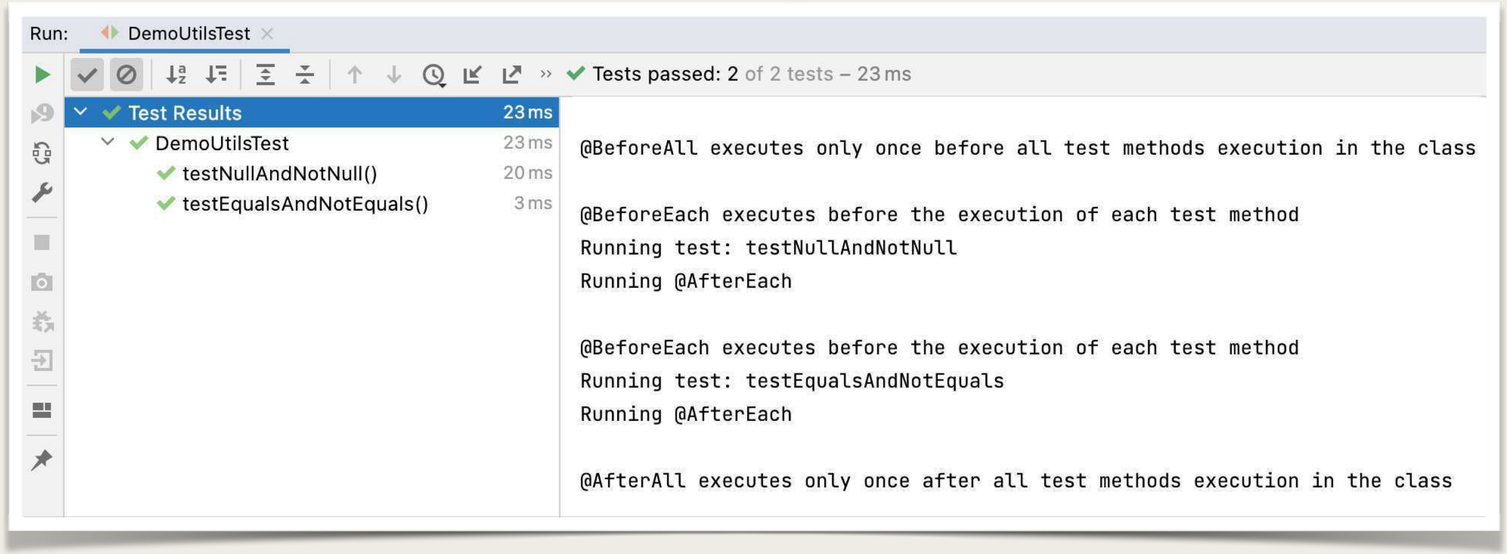
Executed only once, before all test methods

Executed only once, after all test methods

By default, methods must be static

Execution Sequence

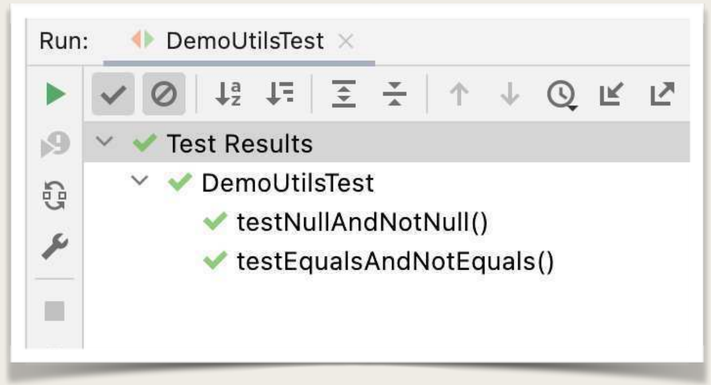





```java
@BeforeAll
static void setupBeforeEachClass() {
    System.out.println("@BeforeAll executes only once before all test methods execution in the class\n");
}

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

JUnit Custom Display Names

Custom Display Names



- Currently the method names are listed in test results

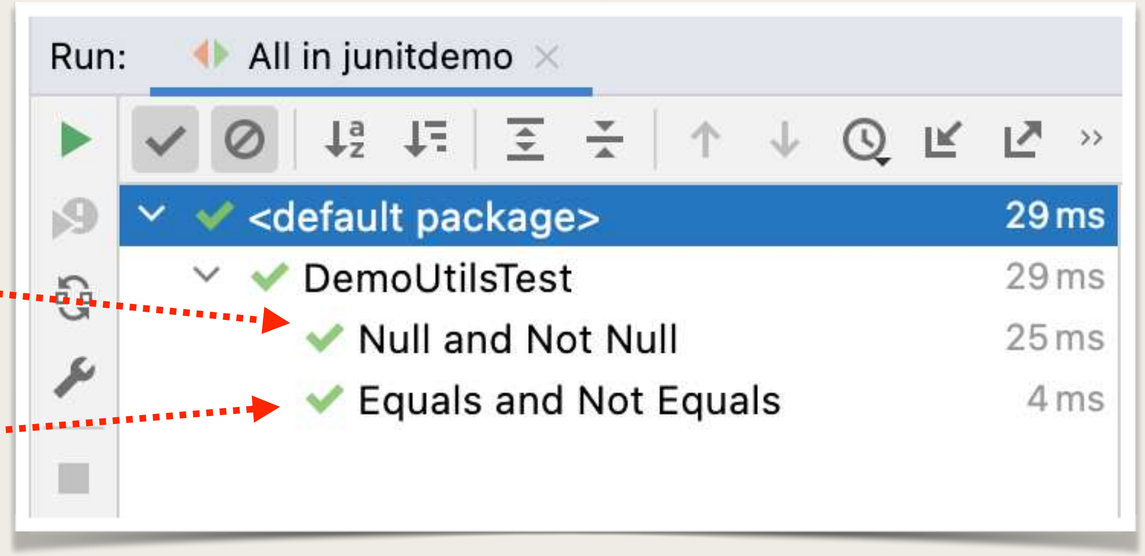
- We'd like to give custom display names
  - Provide a more descriptive name for the test
  - Include spaces, special characters: *Test for Equality to support JIRA #123*
  - Useful for sharing test reports with project management and non-techies

@DisplayName Annotation

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

DemoUtilsTest.java

```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.checkNull(str1), "Object should be null");
        assertNotNull(demoUtils.checkNull(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");
        assertNotEquals(8, demoUtils.add(1, 9), "1+9 must not be 8");
    }
}
```



Display Name Generators

- JUnit can generate display names for you

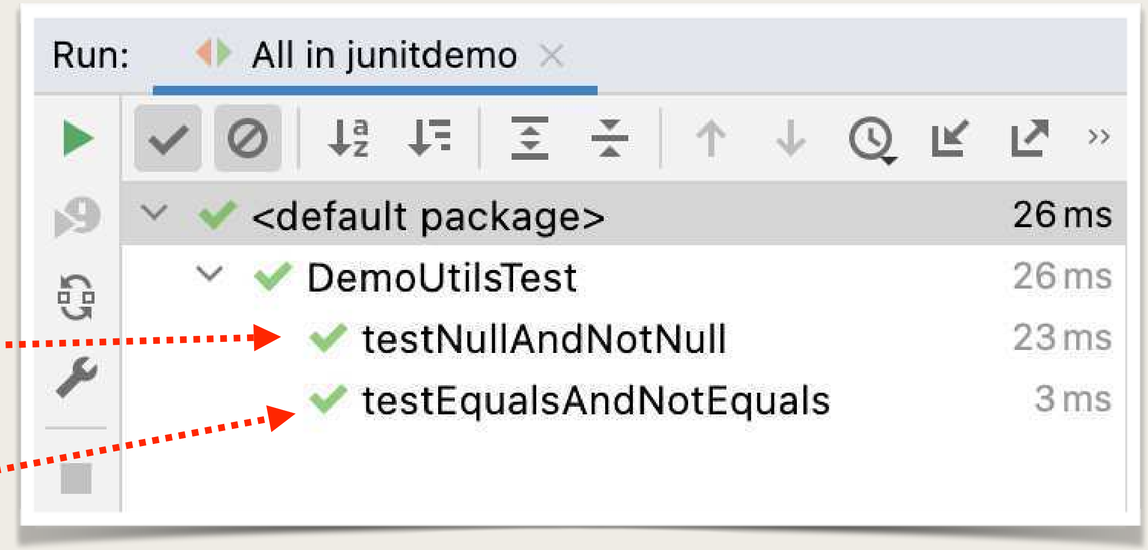
Name	Description
`Simple`	Removes trailing parentheses from test method name
`ReplaceUnderscores`	Replaces underscores in test method name with spaces
`IndicativeSentences`	Generate sentence based on test class name and test method name

Simple Generator

DemoUtilsTest.java

```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");
        assertNotNull(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");
    }
}
```

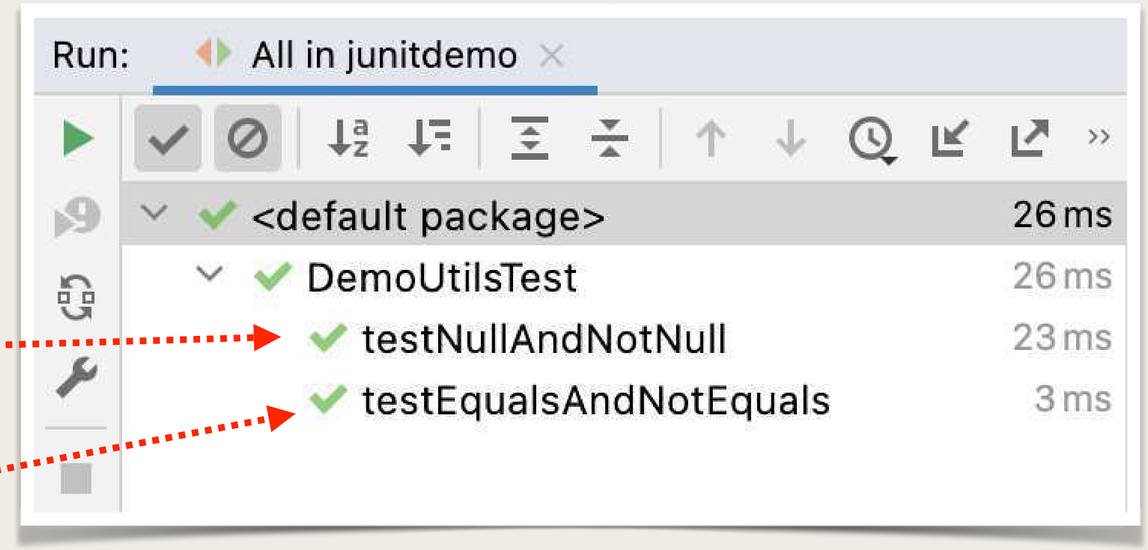
Removes trailing parentheses from test method name



Simple Generator

DemoUtilsTest.java

```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");
        assertNotNull(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");
    }
}
```

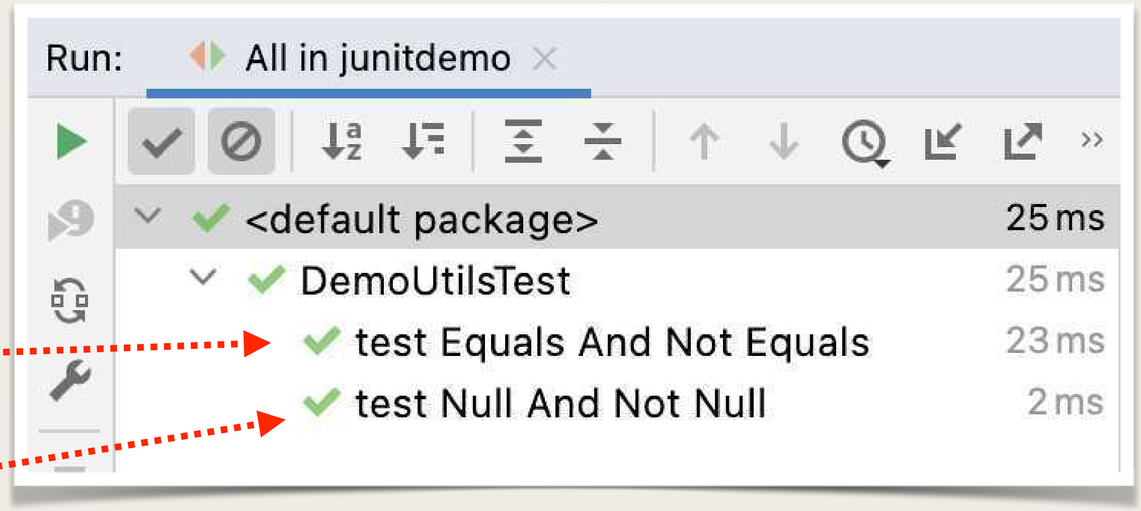


Replace Underscores Generator

DemoUtilsTest.java

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

Replaces underscores with spaces

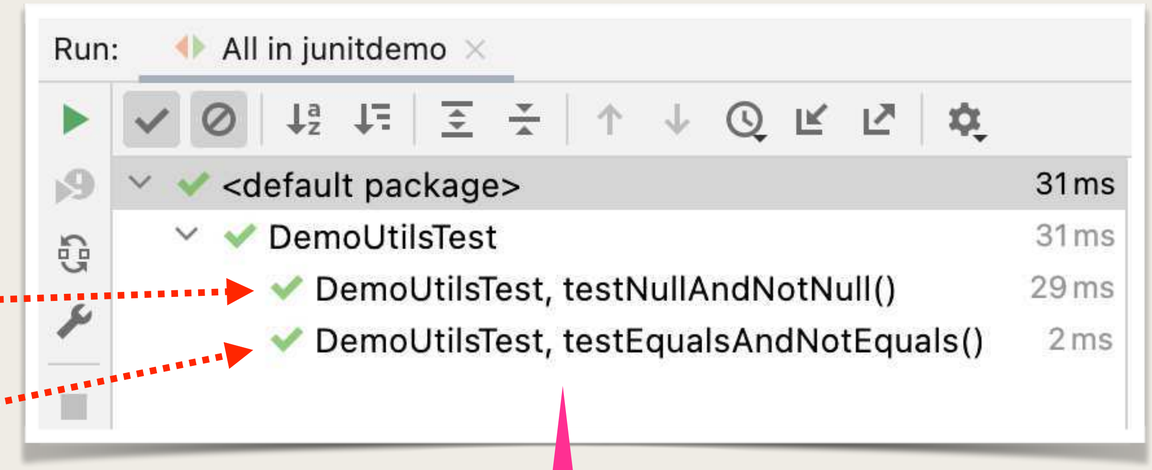


Indicative Sentences Generator

DemoUtilsTest.java

```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");
        assertNotNull(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: Same and True

Assertions

- Test for Same and NotSame

Method name	Description
`assertSame`	Assert that items refer to same object
`assertNotSame`	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

DemoUtils.java

```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;
   }
}
```

DemoUtilsTest.java

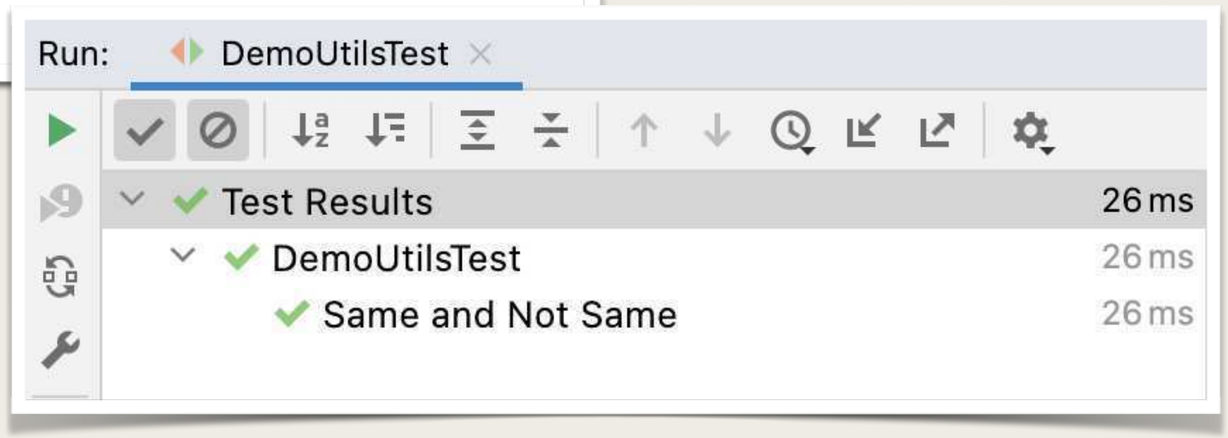
```java
package com.luv2code.junitdemo;

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";

        assertSame(demoUtils.getAcademy(), demoUtils.getAcademyDuplicate(), "Objects should refer to same object");
        assertNotSame(str, demoUtils.getAcademy(), "Objects should not refer to same object");
    }

}
```

Object1

Object2

Object1

Object2



Assertions

- Test for True and False

Method name	Description
`assertTrue`	Assert that condition is true
`assertFalse`	Assert that condition is false

Code to Test

DemoUtils.java

```
package com.luv2code.junitdemo;

public class DemoUtils {

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

For academic purposes only :-)
We can refactor accordingly

Code to Test

Refactored version

DemoUtils.java

```
package com.luv2code.junitdemo;

public class DemoUtils {

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

DemoUtilsTest.java

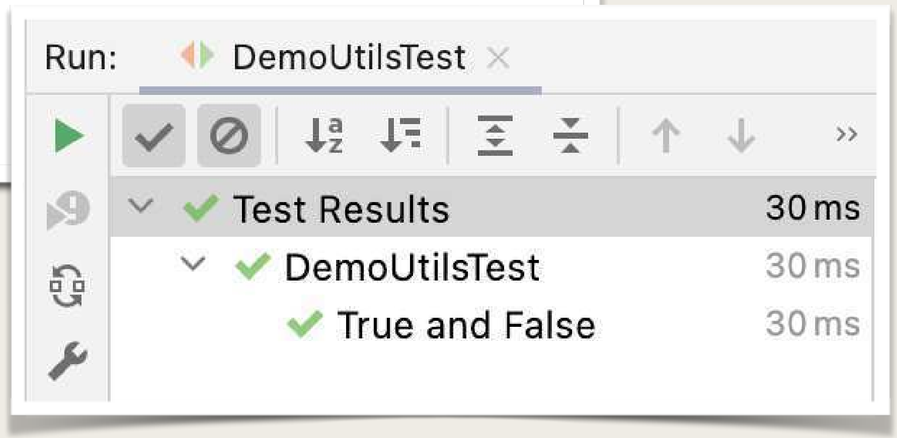
```java
package com.luv2code.junitdemo;

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 true");
    }
}
```

Boolean condition

Boolean condition

DemoUtils.java

```java
package com.luv2code.junitdemo;

public class DemoUtils {

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



Assertions: Arrays, Iterables and Lines

Assertions

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

Code to Test

DemoUtils.java

```
package com.luv2code.junitdemo;

public class DemoUtils {

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

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

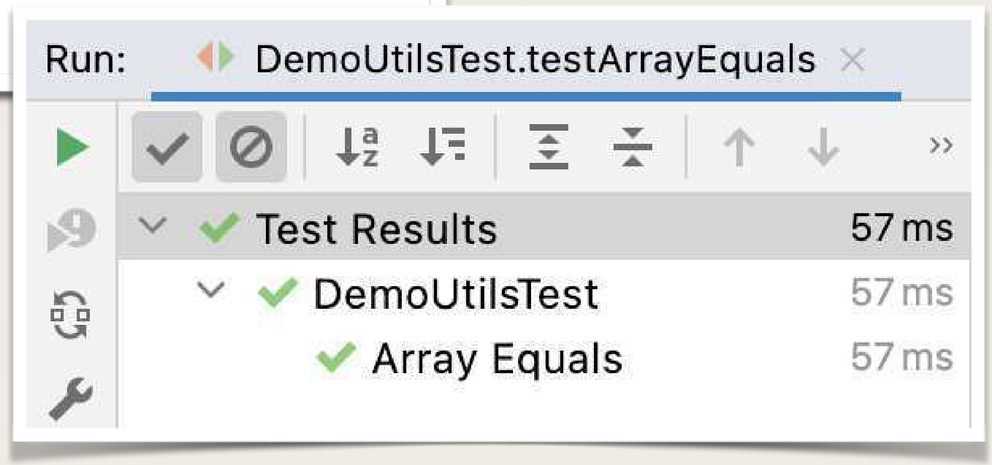
Make comparisons against this array

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;
    }
}
```



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` ...

Code to Test

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;
    }
}
```

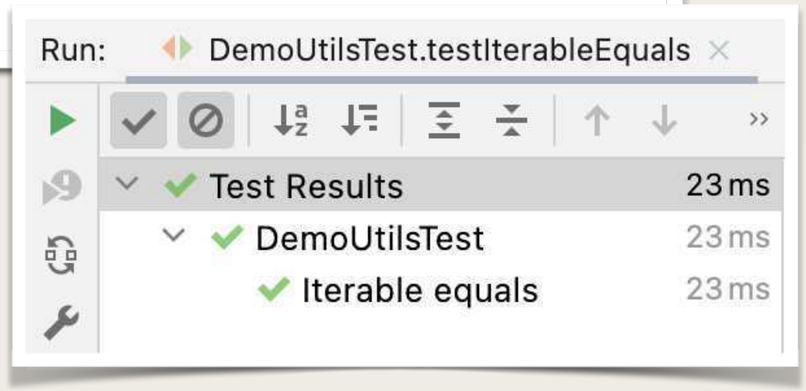
Make comparisons against this List

DemoUtilsTest.java

```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

```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

Code to Test

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;
    }
}
```

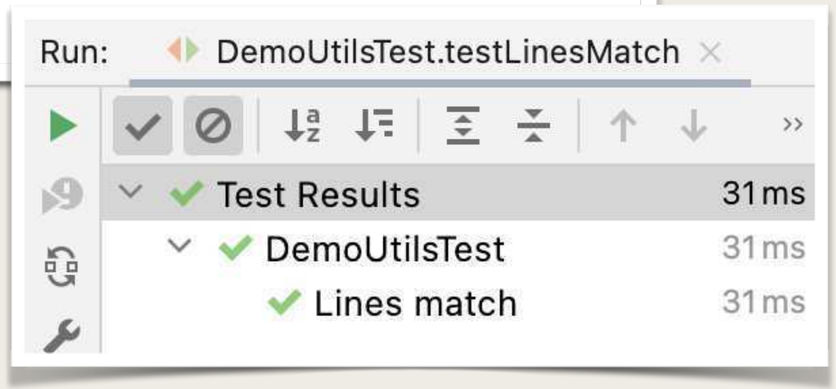
Make comparisons against this List

DemoUtils.java

```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;
    }
}
```

DemoUtilsTest.java

```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");
    }
```



luv2code

Assertions: Throws and Timeout

Assertions

Method name	Description
`assertThrows`	Assert that an executable throws an exception of expected type

Code to Test

Verify this method
throws an exception for values < 0

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";
    }

}
```

DemoUtilsTest.java

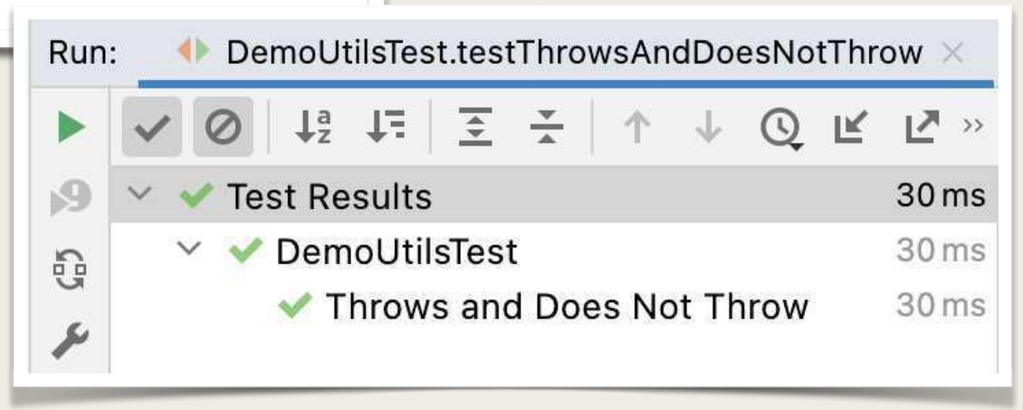
```java
package com.luv2code.junitdemo;

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

Lambda expression

DemoUtils.java

```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
`assertTimeoutPreemptively`	Assert that an executable completes before given timeout is exceeded

Execution is preemptively aborted if timeout is exceeded

Code to Test

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");
    }
}
```

Make sure method doesn't oversleep. :-)

DemoUtilsTest.java

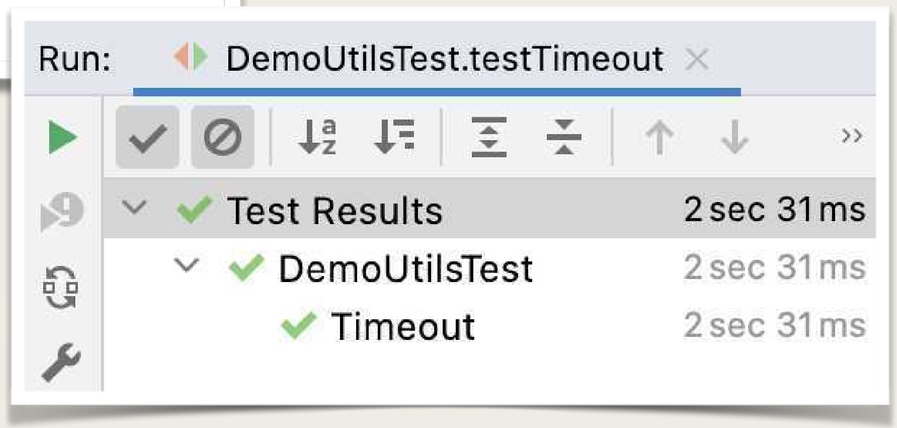
```
package com.luv2code.junitdemo;

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");
    }
}
```



luv2code

Running Tests in Order

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

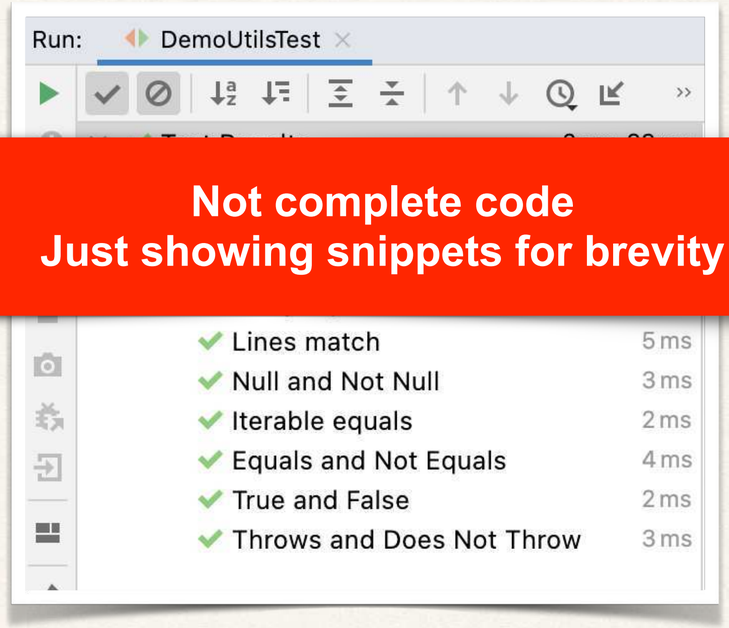
Use Cases

- 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()
}
```



Not complete code
Just showing snippets for brevity

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/

Annotation

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

Specify Method Order

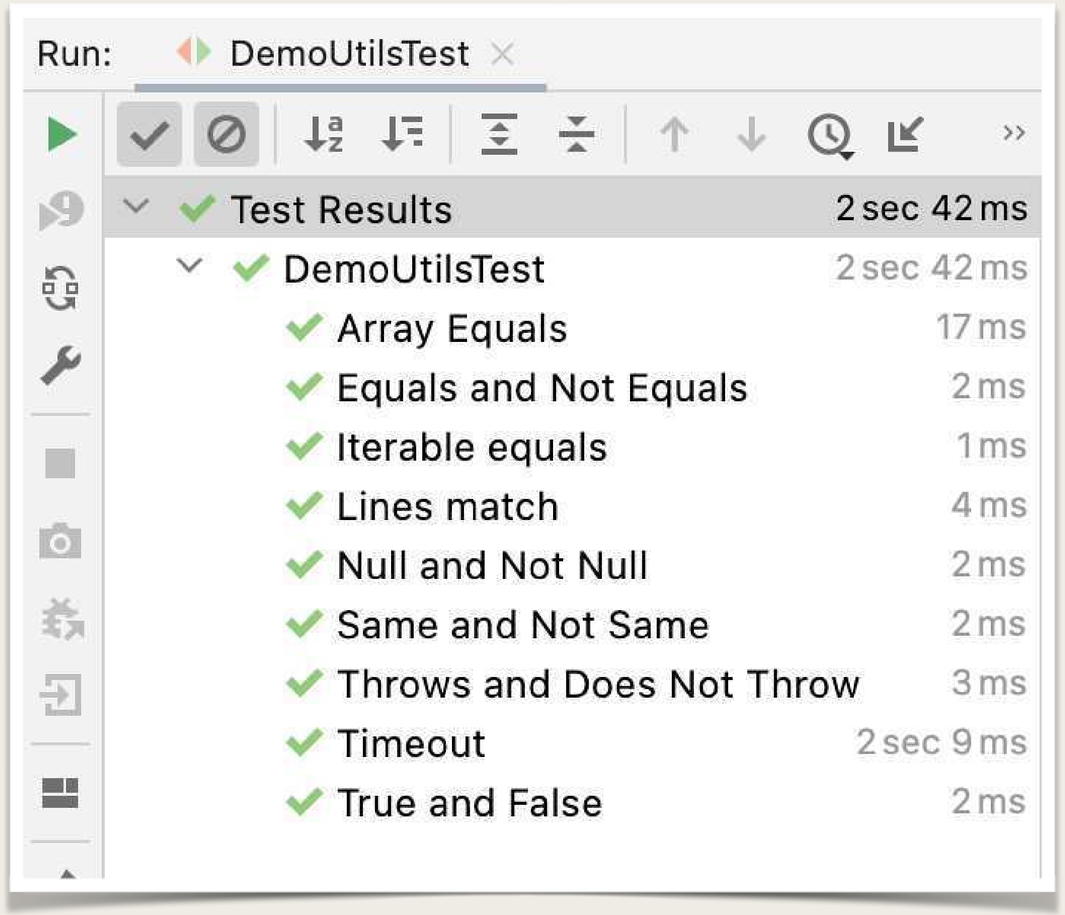
Name	Description
`MethodOrderer.DisplayName`	Sorts test methods alphanumerically based on display names
`MethodOrderer.MethodName`	Sorts test methods alphanumerically based on method names
`MethodOrderer.Random`	Pseudo-random order based on method names
`MethodOrderer.OrderAnnotation`	Sorts test methods 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()
}
```



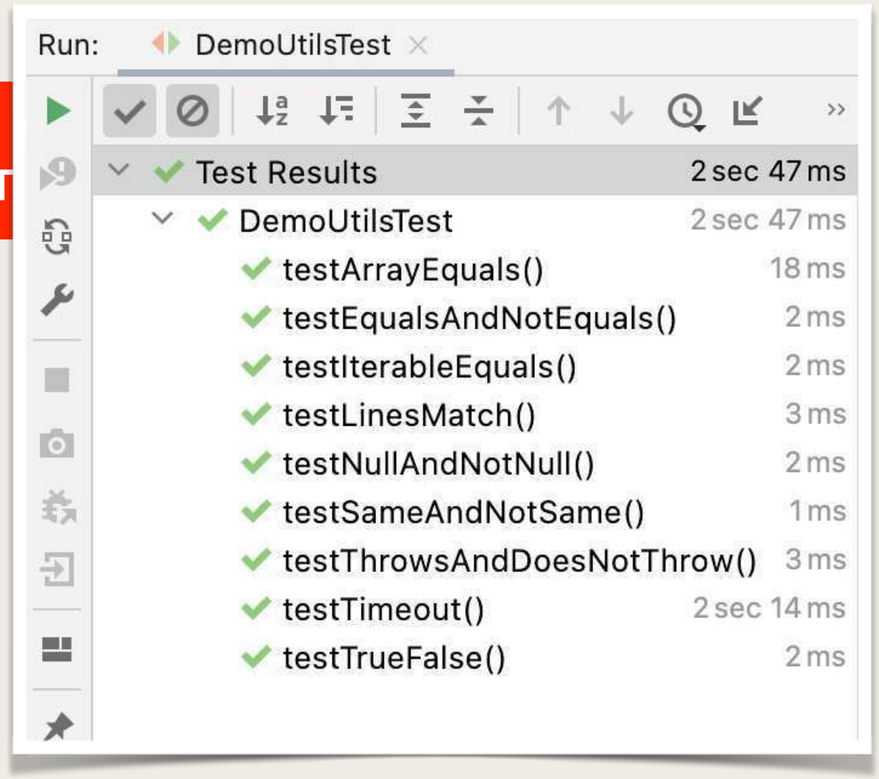
Order by Method Name

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

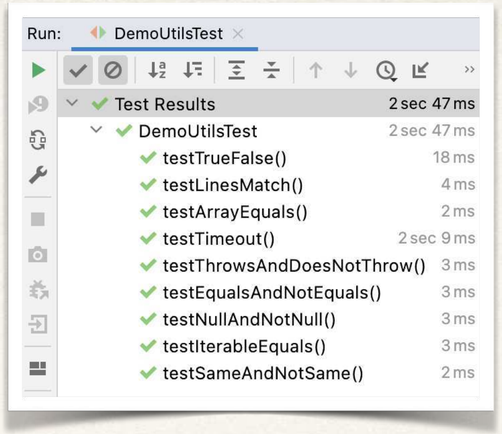


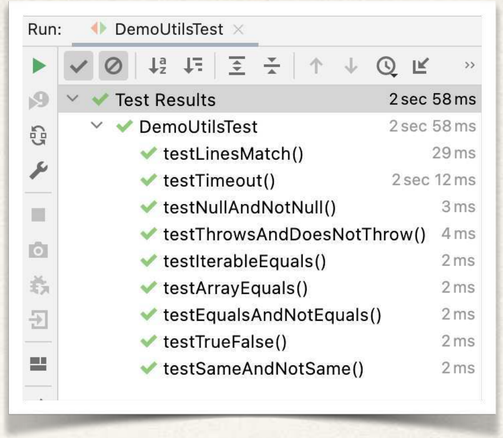
Random by Method Name

DemoUtilsTest.java

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

    void testEqualsAndNotEquals()

    void testNullAndNotNull()

    void testSameAndNotSame()

    void testTrueFalse()

    void testArrayEquals()

    void testIterableEquals()

    void testLinesMatch()

    voi

    voi
}
```

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





Great scenario!!!!

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

Annotation

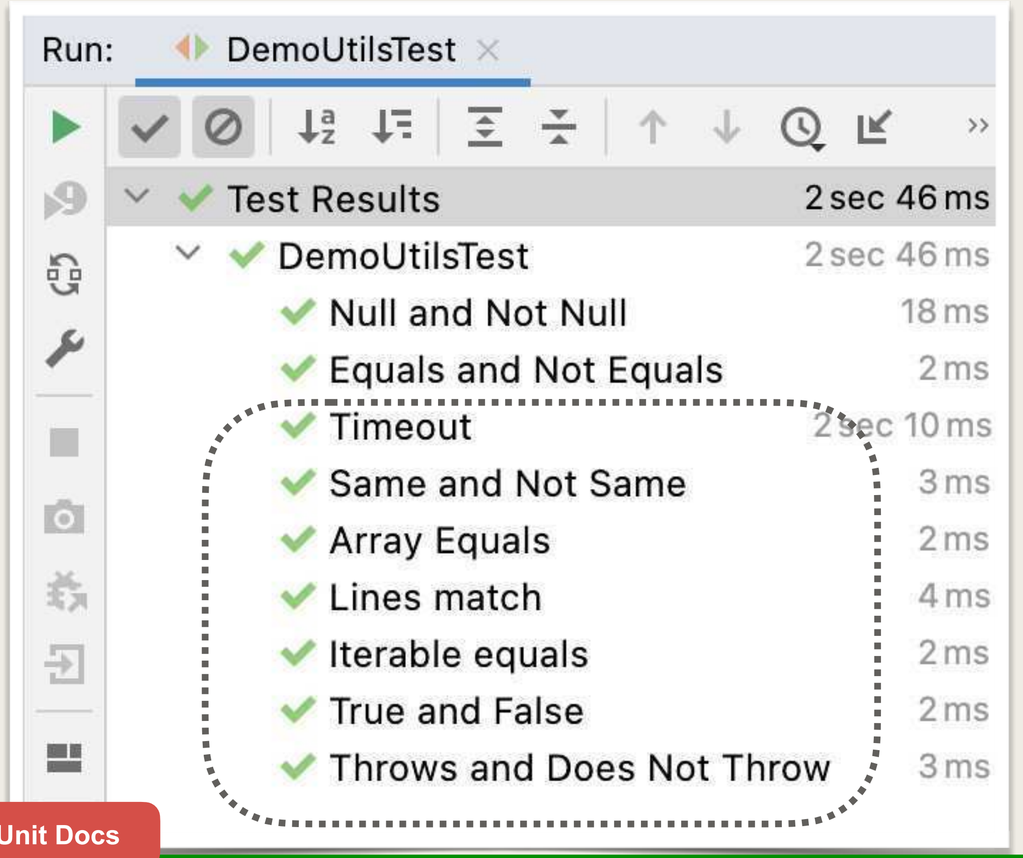
Annotation	Description
`@Order`	Manually specify the order with an `int` number - Order with lowest number has highest priority - Negative numbers are allowed

@Order

Remember
Lowest number has highest priority

DemoUtilsTest.java

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

    @DisplayName("Equals and Not Equals")
    @Order(3)
    void testEqualsAndNotEquals()

    @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

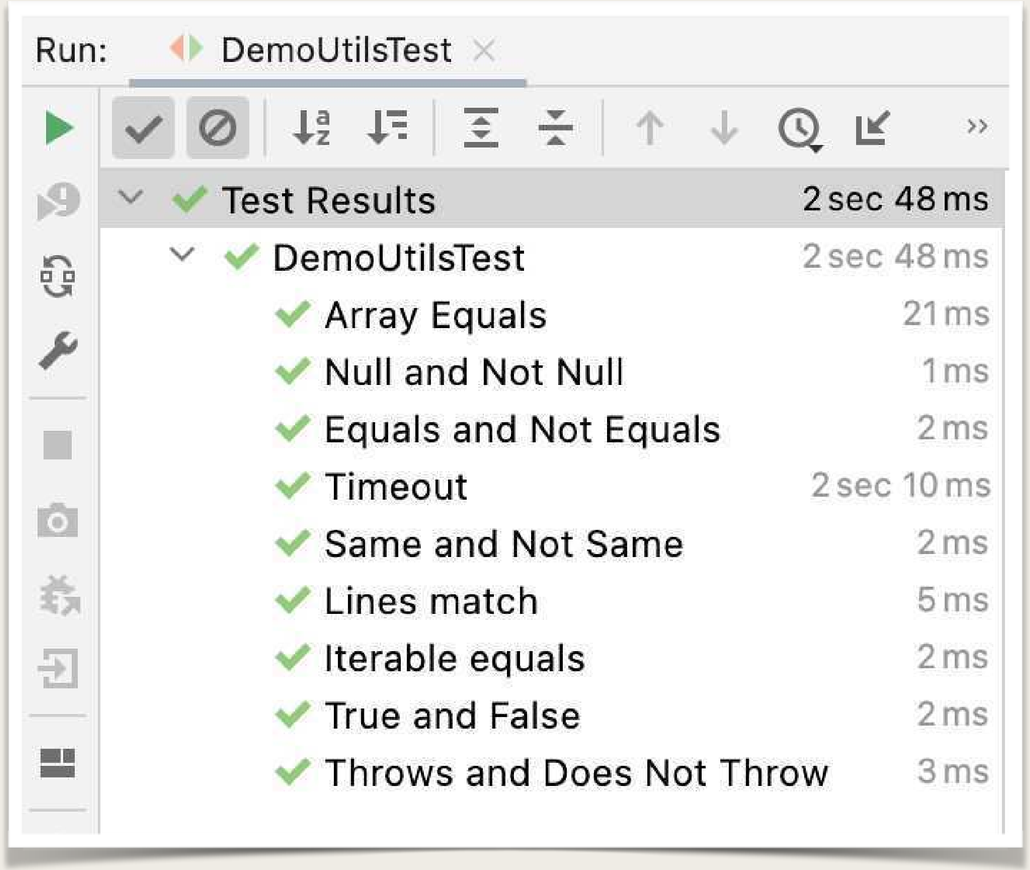
By default, test classes and methods will be ordered using an algorithm that is deterministic but intentionally nonobvious.

@Order - Negative Numbers

Remember
Lowest number has highest priority

DemoUtilsTest.java

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

    @DisplayName("Equals and Not Equals")
    @Order(3)
    void testEqualsAndNotEquals()

    @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")
    @Order(-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.

Additional Features

- If you have multiple test classes, you can order the classes
- Define custom order implementation
- Configure default order in properties file

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

luv2code

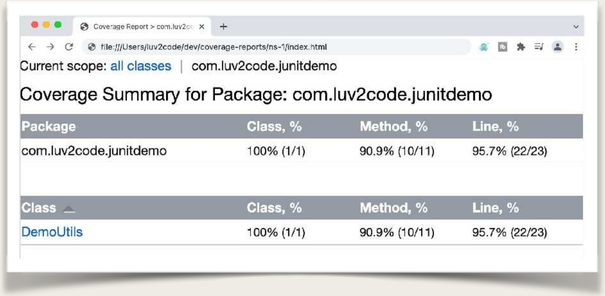
Code Coverage and Test Reports with IntelliJ

Code Coverage

- Code coverage measures how many methods/lines are called by your tests
- Coverage is represented as a percentage: 50% coverage etc ...
- In general, the higher the coverage the better
  - However, 100% is not always attainable
  - On most teams, 70%-80% is acceptable
- Code coverage is only a metric and can be easily tricked with bad tests
- Use the metric as simply one data point in your software process

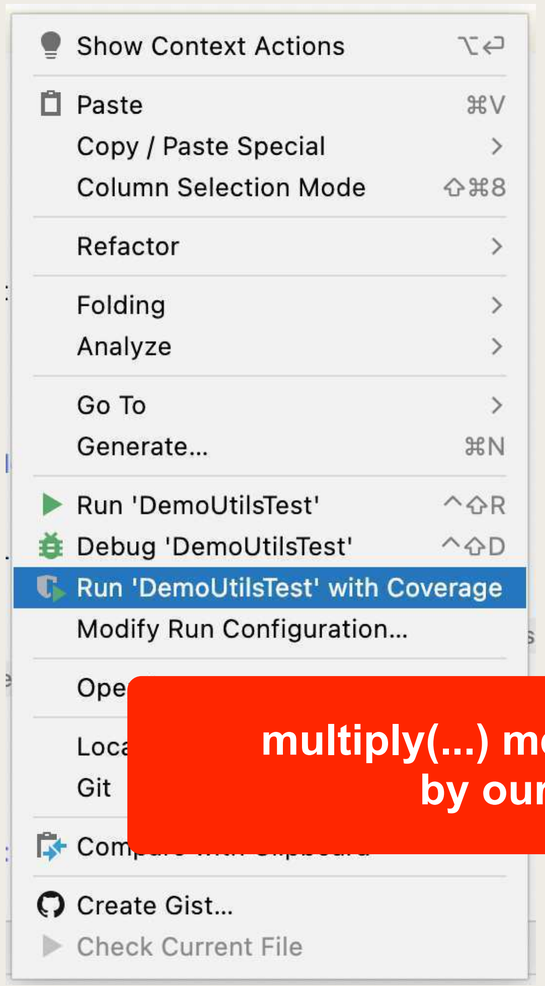
IntelliJ Support for Code Coverage

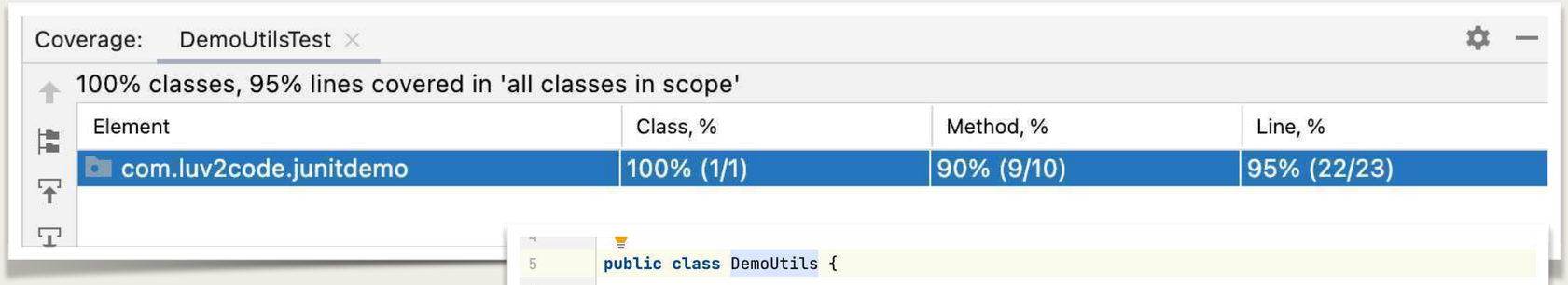
- IntelliJ has built-in support for code coverage
- Can generate coverage reports in the IDE
- Also can generate HTML output for viewing in web browser

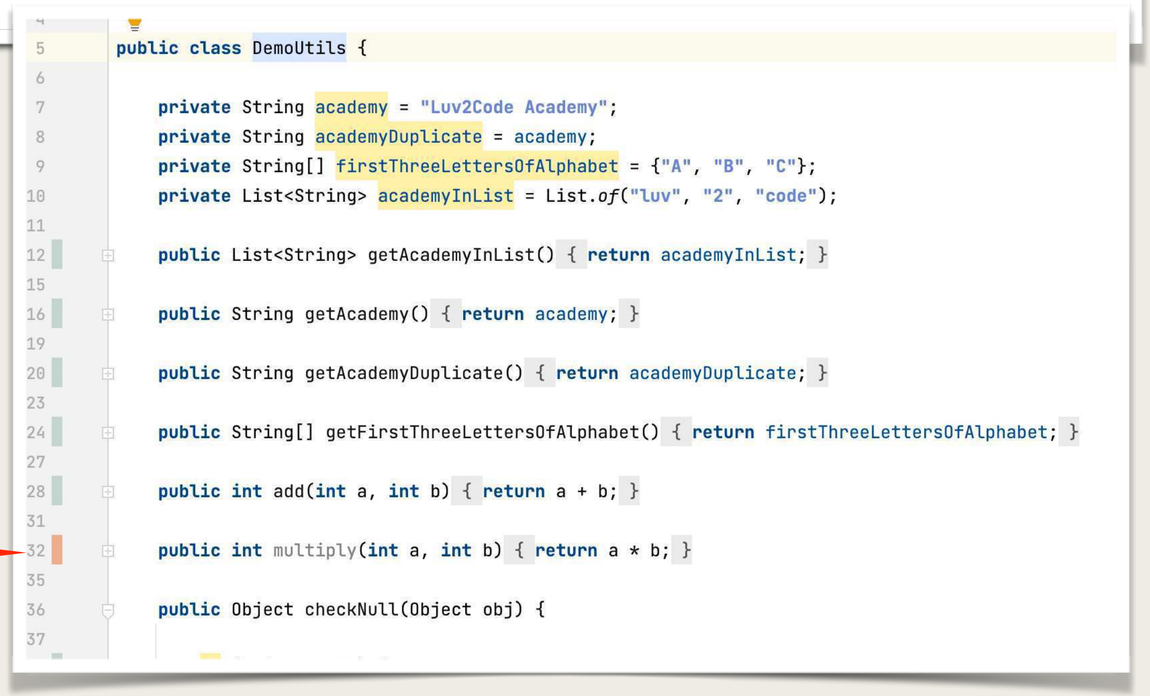


- If you are not using IntelliJ, don't worry
  - In later videos, I'll show you how to do the same using Maven commands
  - No IDE is required ... run Maven from command-line

IntelliJ Support for Code Coverage

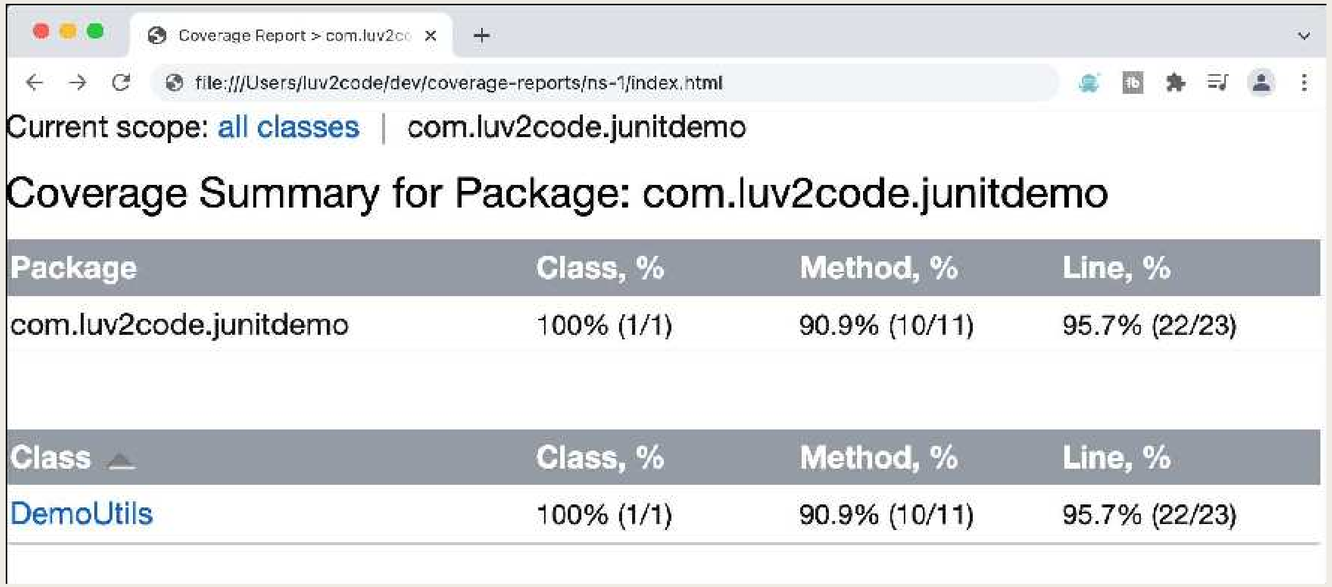




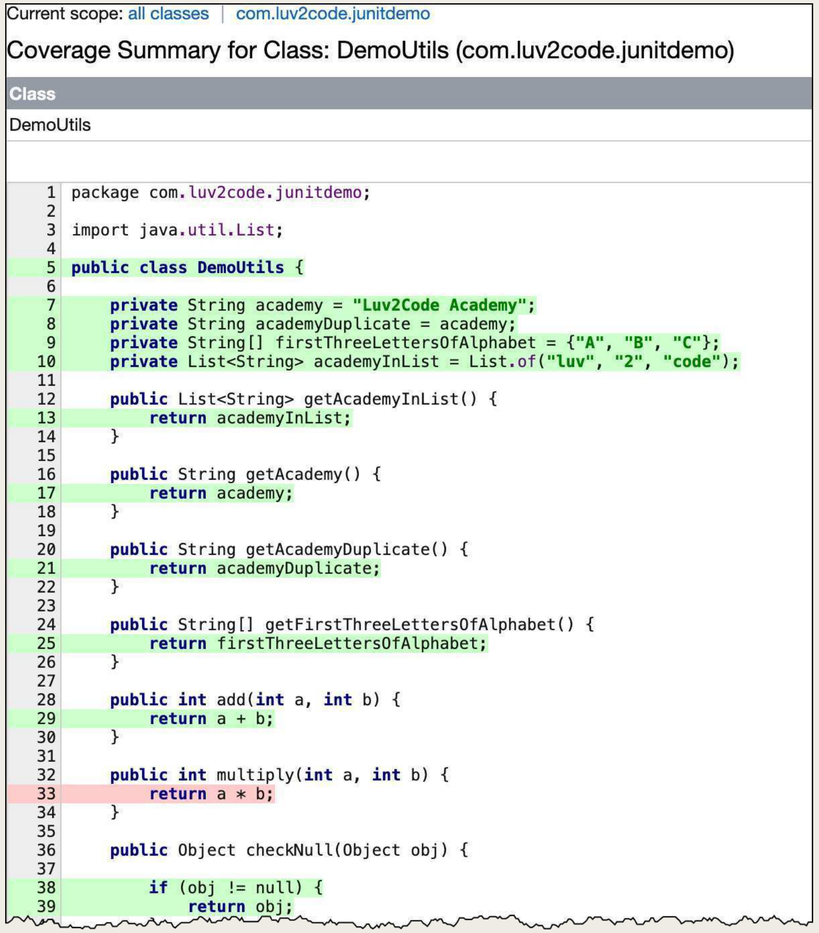


multiply(...) method not covered by our unit tests

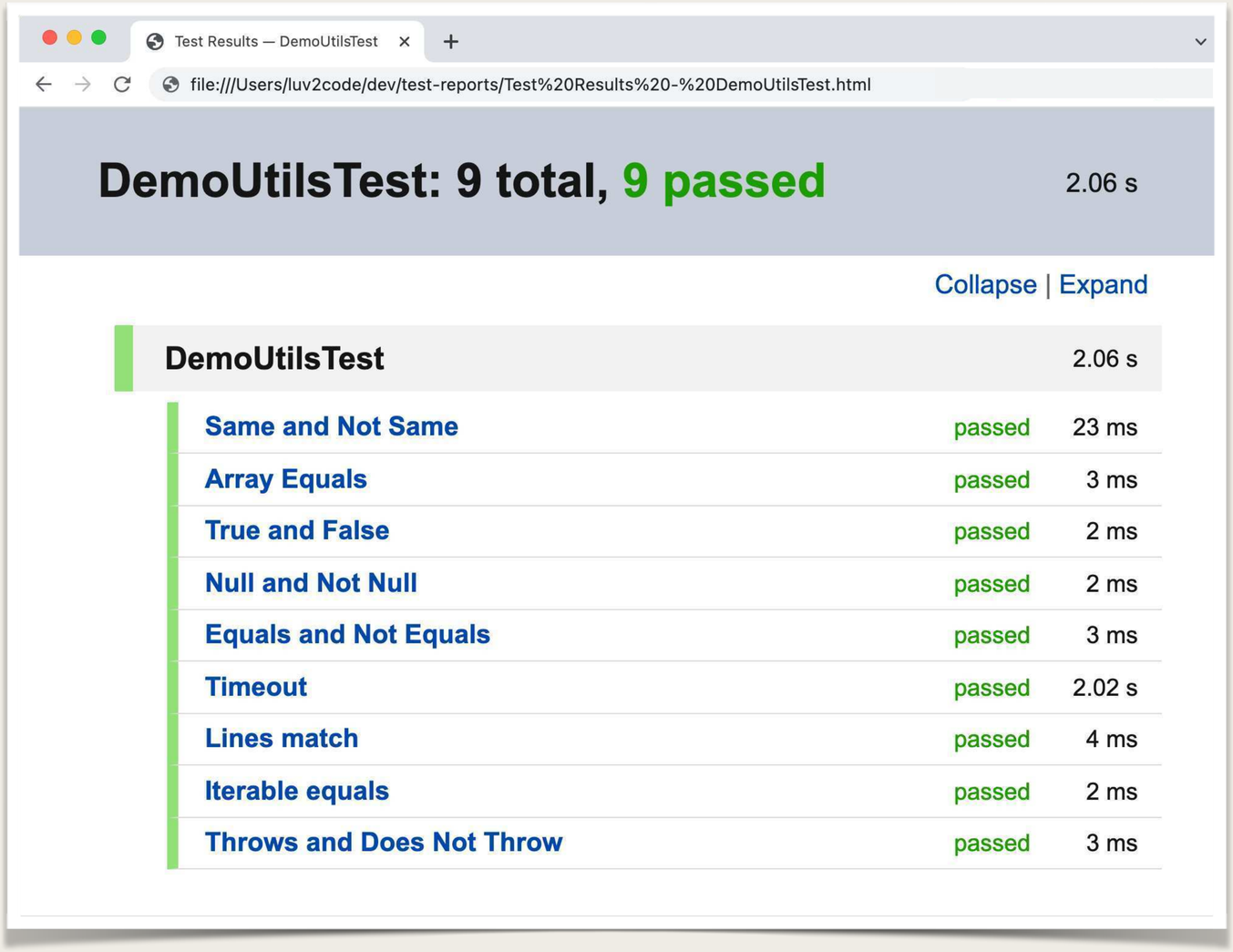
IntelliJ - Generate Code Coverage Report



multiply(...) method not covered by our unit tests



IntelliJ - Generate Test Reports



IntelliJ Documentation

- Additional features and support for testing

https://www.jetbrains.com/help/idea/tests-in-ide.html

luv2code

Code Coverage and Test Reports with Maven

Code Coverage Reports

- If you are not using IntelliJ, don't worry
  - In this video, I'll show you how to do the same using Maven commands
  - No IDE is required ... run Maven from command-line
  - Useful when running as part of DevOps build process
  - Continuous Integration / Continuous Deployment (CI/CD) environments

System Requirements

- To complete these steps, you must have Maven installed (not with IDE)
- If you don't have Maven installed, follow steps at: `https://maven.apache.org`
- Once installed, verify your Maven installation
- Open a terminal window and type: **`mvn -version`**

```
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcdc97d260186937)
Maven home: ...
...
```

Development Process

1. Run unit tests
2. Generate unit test reports
3. Generate code coverage reports

Step 1: Run unit tests

- At command-line, type: **`mvn clean test`**

```
...

[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.luv2code.junitdemo.DemoUtilsTest
I am going to sleep
Sleeping over
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.077 s - in com.luv2code.junitdemo.DemoUtilsTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.673 s

...
```

Step 2: Generate unit test reports

- Maven **SureFire-Report** plugin can generate HTML unit test report

```
...

<reporting>

    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-report-plugin</artifactId>
            <version>3.5.2</version>
       </plugin>
    </plugins>

</reporting>

...
```

Step 2: Generate unit test reports

- Reference **Maven-Site** plugin for supporting HTML content (css, images etc)

```
...

<build>

    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-site-plugin</artifactId>
            <version>3.21.0</version>
        </plugin>

    </plugins>

</build>
...
```

Step 2: Generate unit test reports

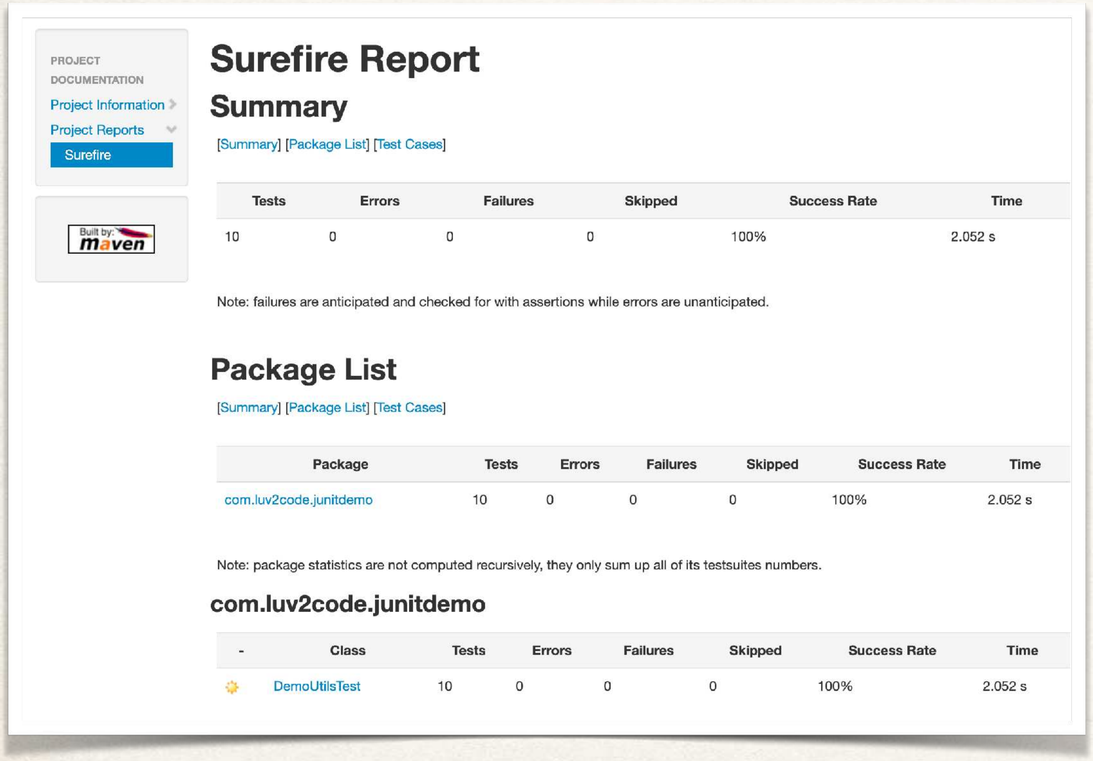
- At command-line, type:

```
mvn site
```

Generates HTML content and unit test reports

View unit test reports

File: target/site/surefire.html



PROJECT DOCUMENTATION

Project Information

Project Reports

Surefire

Built by Maven

Surefire Report

Summary

[Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
10	0	0	0	100%	2.052 s

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

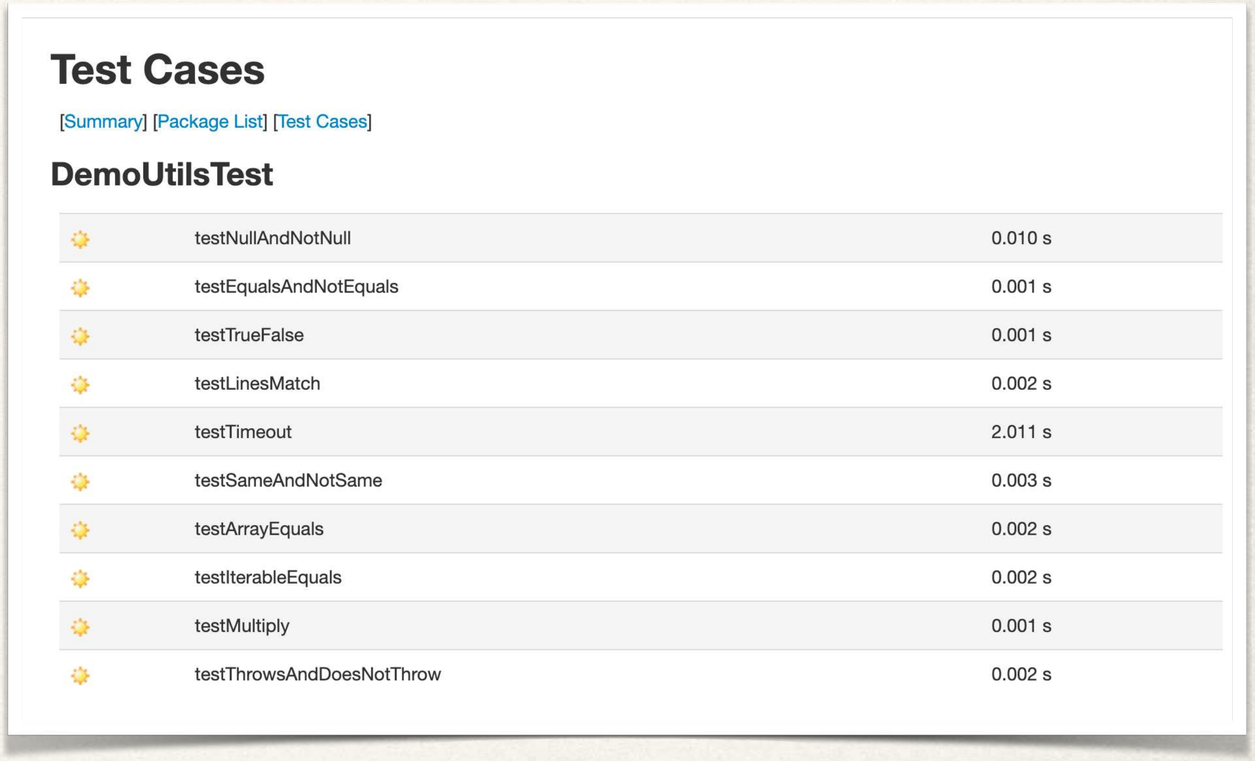
[Summary] [Package List] [Test Cases]

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
com.luv2code.junitdemo	10	0	0	0	100%	2.052 s

Note: package statistics are not computed recursively, they only sum up all of its testsuites numbers.

com.luv2code.junitdemo

-	Class	Tests	Errors	Failures	Skipped	Success Rate	Time
	DemoUtilsTest	10	0	0	0	100%	2.052 s



Test Cases

[Summary] [Package List] [Test Cases]

DemoUtilsTest

	testNullAndNotNull	0.010 s
	testEqualsAndNotEquals	0.001 s
	testTrueFalse	0.001 s
	testLinesMatch	0.002 s
	testTimeout	2.011 s
	testSameAndNotSame	0.003 s
	testArrayEquals	0.002 s
	testIterableEquals	0.002 s
	testMultiply	0.001 s
	testThrowsAndDoesNotThrow	0.002 s

Show Only Failed Tests

- By default, Maven **Surefire-Report** plugin shows full test results
  - Successful tests and failed tests
- To generate reports for only failed tests

```
<reporting>

    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-report-plugin</artifactId>
            <version>3.5.2</version>
            <configuration>
                <showSuccess>false</showSuccess>
            </configuration>
       </plugin>
    </plugins>

</reporting>
```

generate reports for only failed tests

Show Only Failed Tests

File: target/site/surefire.html

Surefire Report

Summary

[Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
10	0	1	0	90.0%	2.056 s

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

[Summary] [Package List] [Test Cases]

Package	Tests	Errors	Failures	Skipped	Success Rate
com.luv2code.junitdemo	10	0	1	0	90.0%

Test Cases

[Summary] [Package List] [Test Cases]

DemoUtilsTest

testMultiply + [Detail]		0.005 s
-	4*3 must be 12 ==> expected: <13> but was: <12>	-

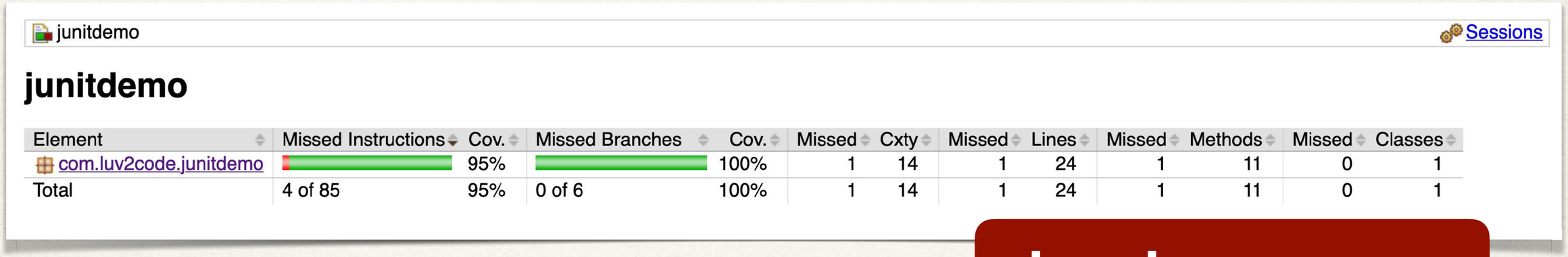
Failure Details

[Summary] [Package List] [Test Cases]

testMultiply	
-	org.opentest4j.AssertionFailedError: 4*3 must be 12 ==> expected: <13> but was: <12>
-	com.luv2code.junitdemo.DemoUtilsTest:24

Step 3: Generate code coverage reports

- **JaCoCo** is a free code coverage library
- JaCoCo provides a Maven plugin to generate code coverage reports


junitdemo

Sessions

junitdemo

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.luv2code.junitdemo		95%		100%	1	14	1	24	1	11	0	1
Total	4 of 85	95%	0 of 6	100%	1	14	1	24	1	11	0	1

www.jacoco.org

Ja: Java
Co: Code
Co: Coverage

Step 3: Generate code coverage reports

pom.xml

```
...
   <build>
      <plugin>
          <groupId>org.jacoco</groupId>
          <artifactId>jacoco-maven-plugin</artifactId>
          <version>0.8.12</version>

          <executions>
              <execution>
                  <id>jacoco-prepare</id>
                  <goals>
                      <goal>prepare-agent</goal>
                  </goals>
              </execution>

              <execution>
                 <id>jacoco-report</id>
                 <phase>test</phase>
                  <goals>
                      <goal>report</goal>
                  </goals>
              </execution>
          </executions>

      </plugin>
   </build>
...
```

Prepare JaCoCo agent

This goal is bound by default to Maven's `initialize` phase

During Maven's `test` phase,

execute the plugin goal

`jacoco-maven-plugin:report`

Step 3: Generate code coverage reports

- At command-line, type: **`mvn clean test`**

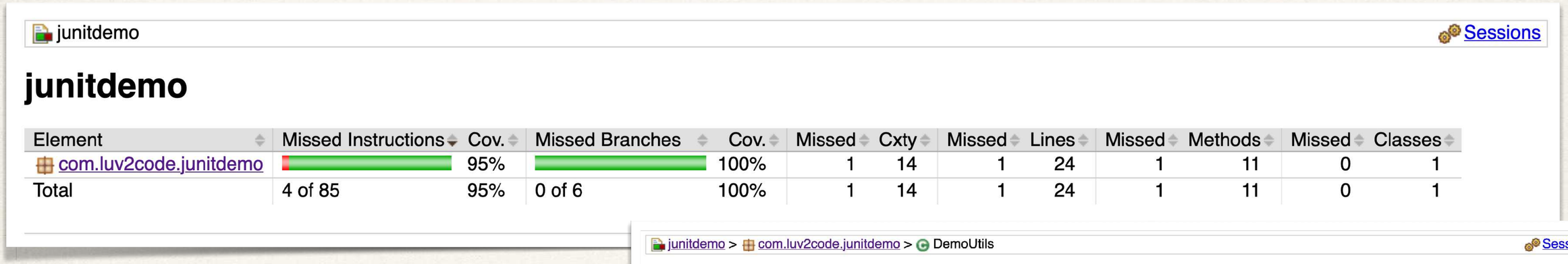
```
...

[INFO] ------------------------< com.luv2code:junitdemo >------------------------
[INFO] Building junitdemo 1.0
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- clean:3.2.0:clean (default-clean) @ junitdemo ---
[INFO] ...
[INFO]
[INFO] --- jacoco:0.8.12:report (jacoco-report) @ junitdemo ---
[INFO] ...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------

...
```

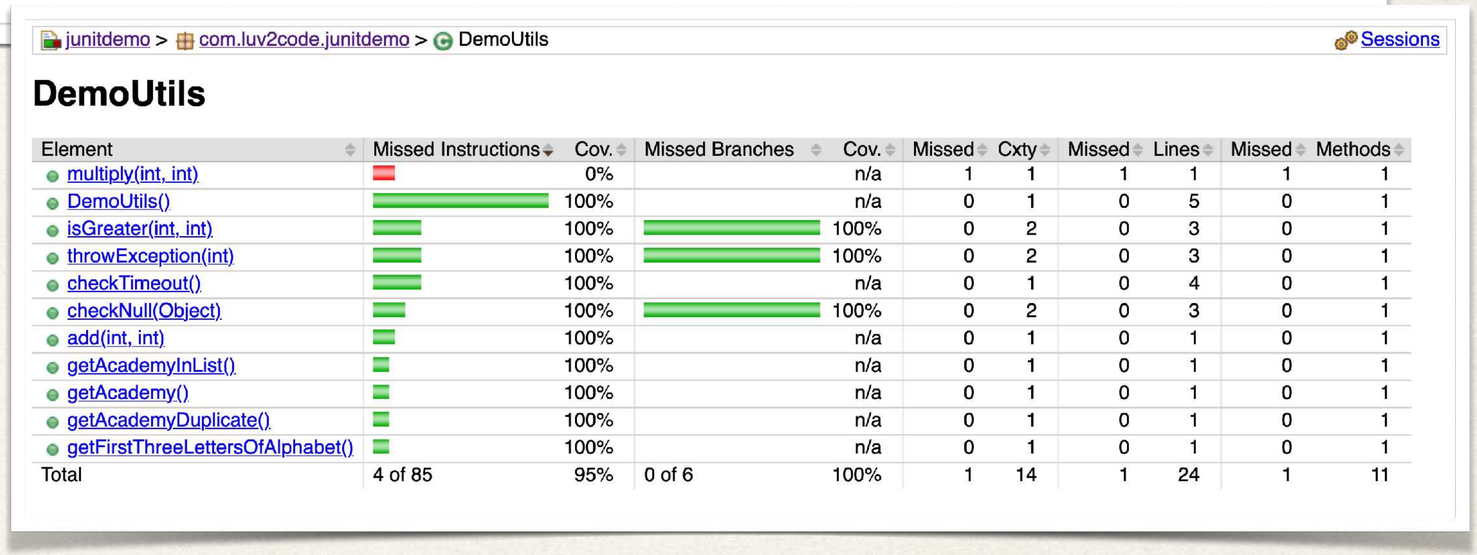
Jacoco report generation is attached to `test` phase.

Reports are automatically generated when unit tests are executed

View code coverage reports

File: target/site/jacoco/index.html





Maven Plugin Documentation

https://maven.apache.org/plugins

https://www.jacoco.org/jacoco/trunk/doc/

luv2code

Conditional Tests

Conditional Tests

We may not need to run all of the tests ....

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
...	...

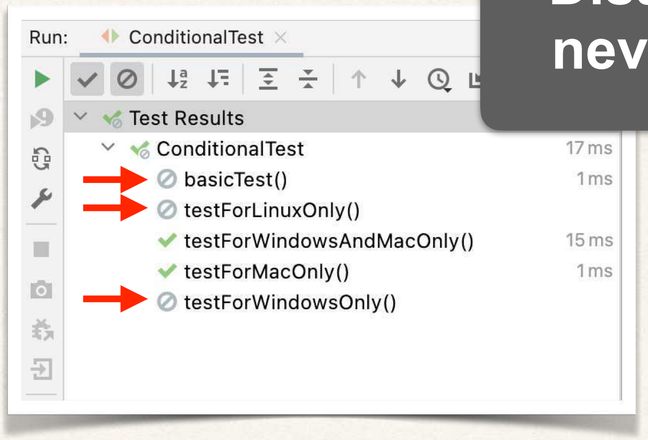
Annotations can be applied at the class level or method level

@Disabled and @EnabledOnOs

ConditionalTest.java

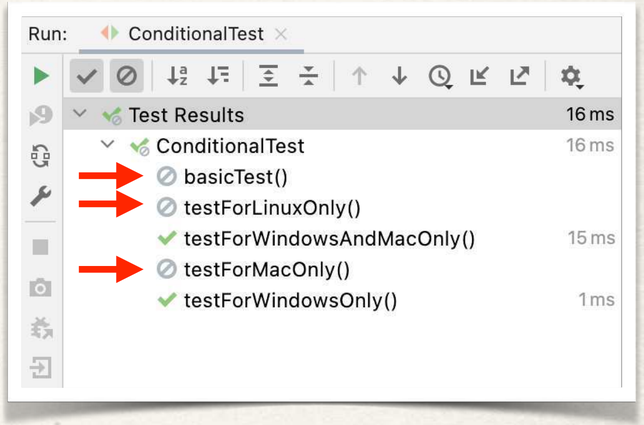
```java
class ConditionalTest {

    @Test
    @Disabled("Don't run until JIRA #123 is resolved")
    void basicTest() {
        // execute method 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
    }

}
```

Mac

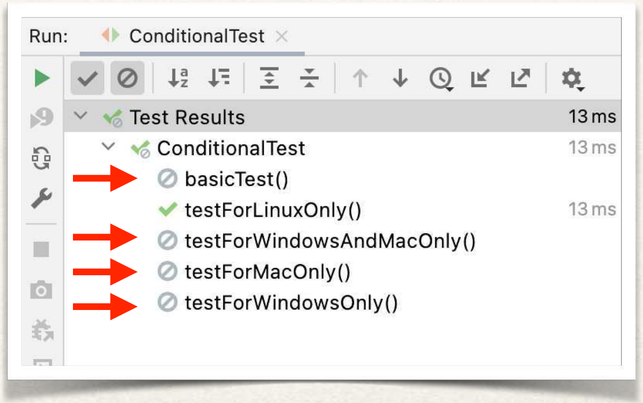


Disabled tests never execute

MS Windows



Linux



Disabled tests never execute

Annotations

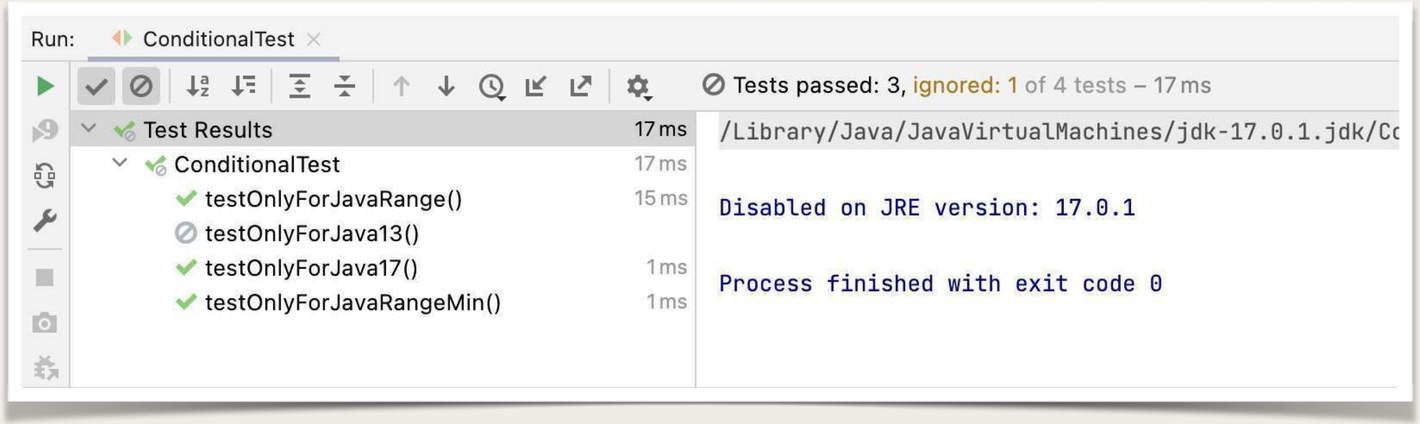
Name	Description
`@EnabledOnJre`	Enable test for a given Java version
`@EnabledForJreRange`	Enable test for a given Java version range
...	...

@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
...	...

@EnabledIfSystemProperty @EnabledIfEnvironmentVariable

ConditionalTest.java

Any name

Regular expression

```
class ConditionalTest {

    @Test
    @EnabledIfSystemProperty(named="LUV2CODE_SYS_PROP", matches="CI_CD_DEPLOY")
    void testOnlyForSystemProperty() {
        // execute method and perform assertions
    }

    @Test
    @EnabledIfEnvironmentVariable(named="LUV2CODE_ENV", matches="DEV")
    void testOnlyForDevEnvironment() {
        // execute method and perform assertions
    }

}
```