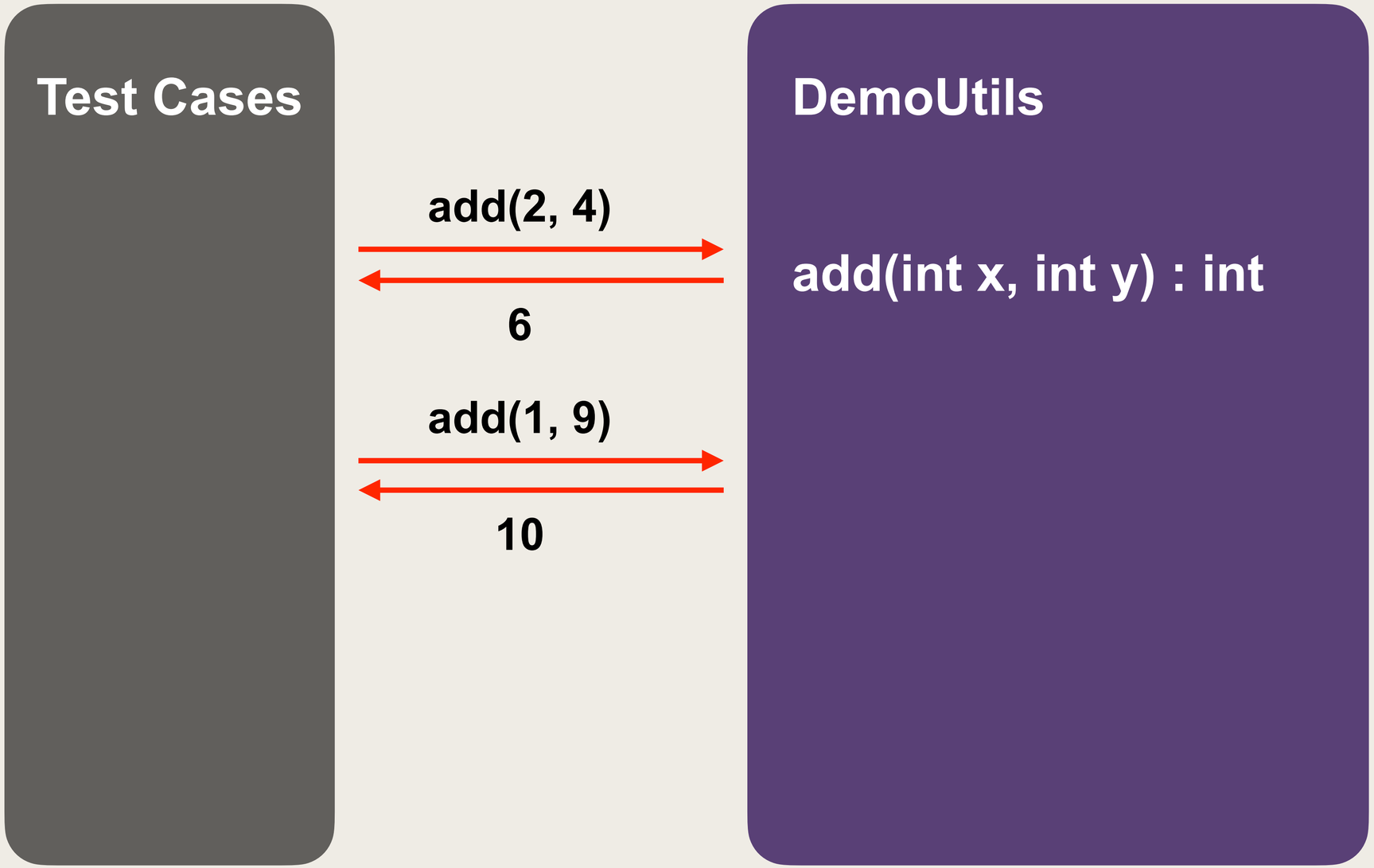
luv2code

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

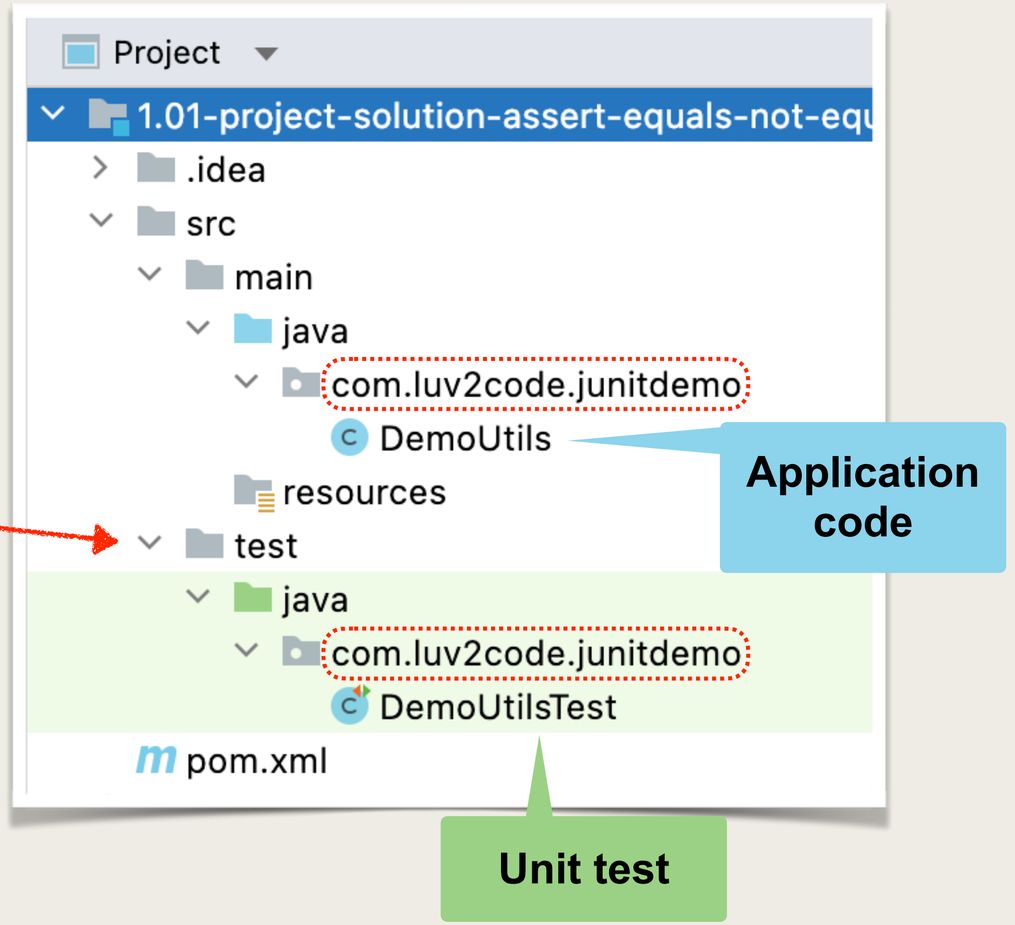
```
...

    <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`

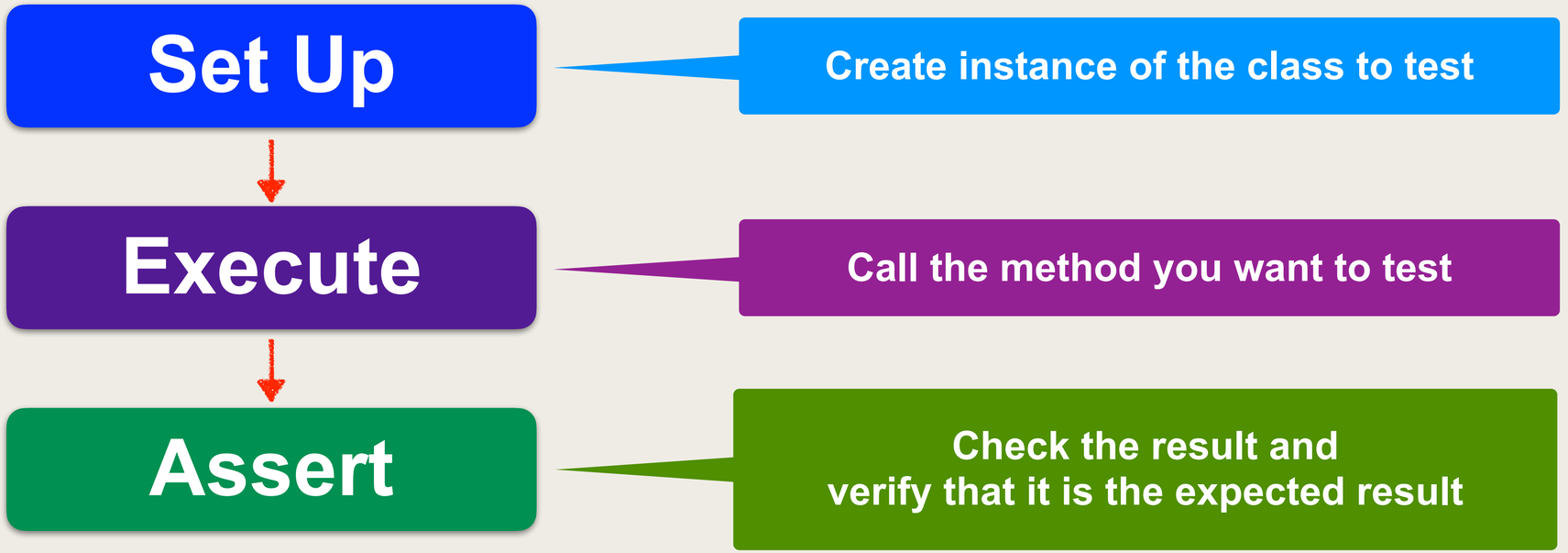
- A convention is to create test classes
  in similar package structure under `/test`

- 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



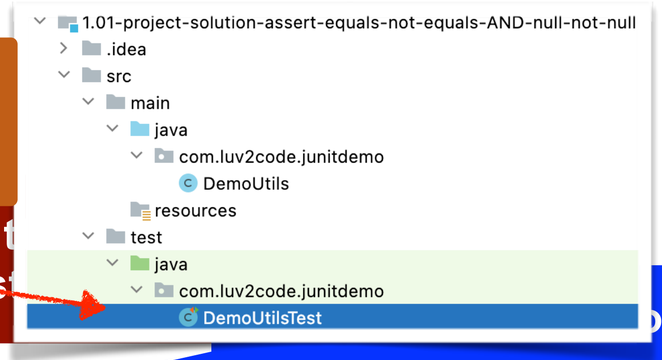
Step 3: Create unit test

DemoUtilsTest.java

```
package com.luv2code.junitdemo;

import org.junit.jupiter.
import org.junit.jupiter.

class DemoUtilsTest {

    @Test
    void testEqualsAndNotEquals() {

        // s
        Demo                    w DemoUtils();

        int

        // e
        int                     dd(2, 4);

        // a
        Asse                    (expected, actual, "2+4 must be 6");
    }

}
```

Test can have any method name

Annotate t… as a test



…f the class to test

Set Up

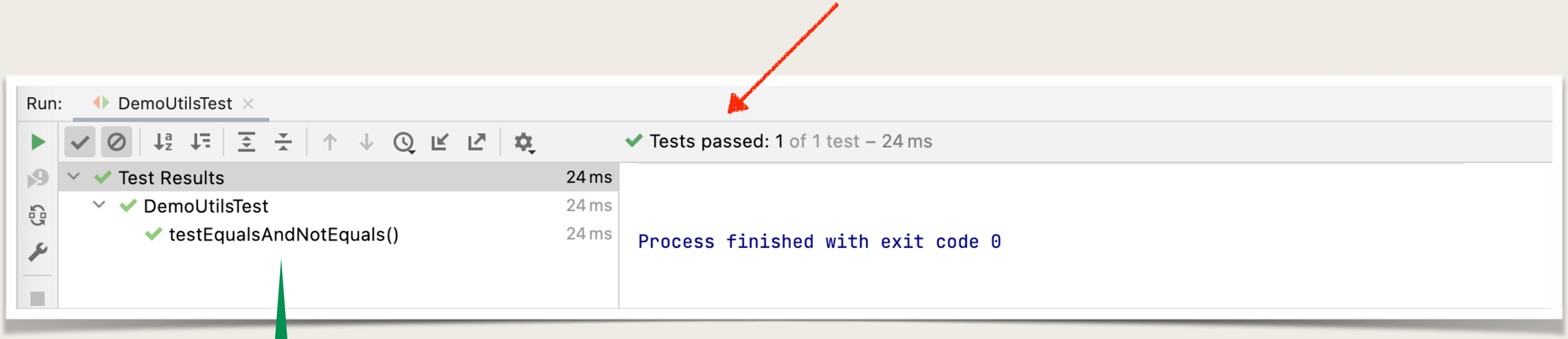
Execute

Assert

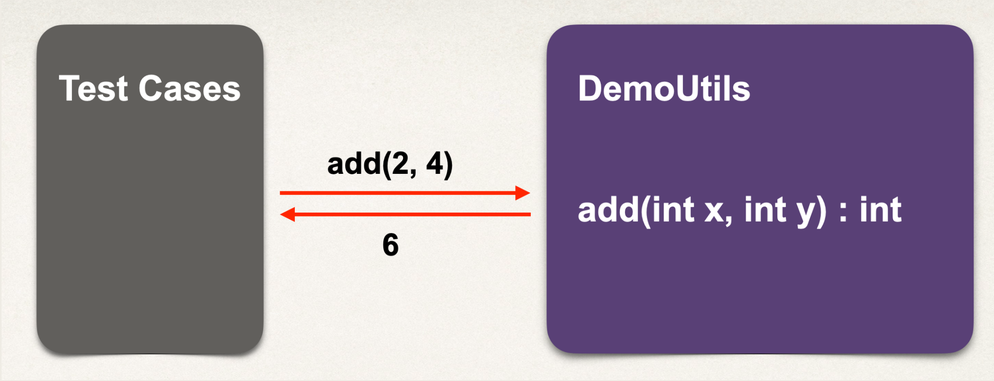
Call the method you want to test

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