



University of
Sheffield



COM3529 Software Testing and Analysis

Background Information for the Practical Sessions

Professor Phil McMinn

COM3529 GitHub Repository

The materials for this module will be pushed to a GitHub repository:

<https://github.com/UoS-COM3529/com3529-2025>

You will need to do a “pull” each week to obtain the latest content.

Java

All the code examples are in Java, and the tests are in JUnit.

To use the Java examples in the repository, you will need to have at **Java 11 or better** installed on your machine.

Gradle

Java code examples are in a **Gradle** library.

Once you have cloned the repository, you can compile and run tests at the terminal from the **code** directory.

These commands should work on Mac/Linux/WSL.

(For the Windows Prompt , replace the initial “./” with “.\”)

<code>./gradlew build</code>	----->	compile all code		
<code>./gradlew test</code>	----->	run all tests		
<code>./gradlew test --tests uk.ac.shef.com3529.TriangleTest</code>			run all tests in a specific class	run a specific test
<code>./gradlew test --tests uk.ac.shef.com3529.TriangleTest.shouldClassifyEquilateral</code>			↑	↑

See the Gradle website and documentation for more information:

<https://gradle.org>

Use of Integrated Developer Environments (IDEs)

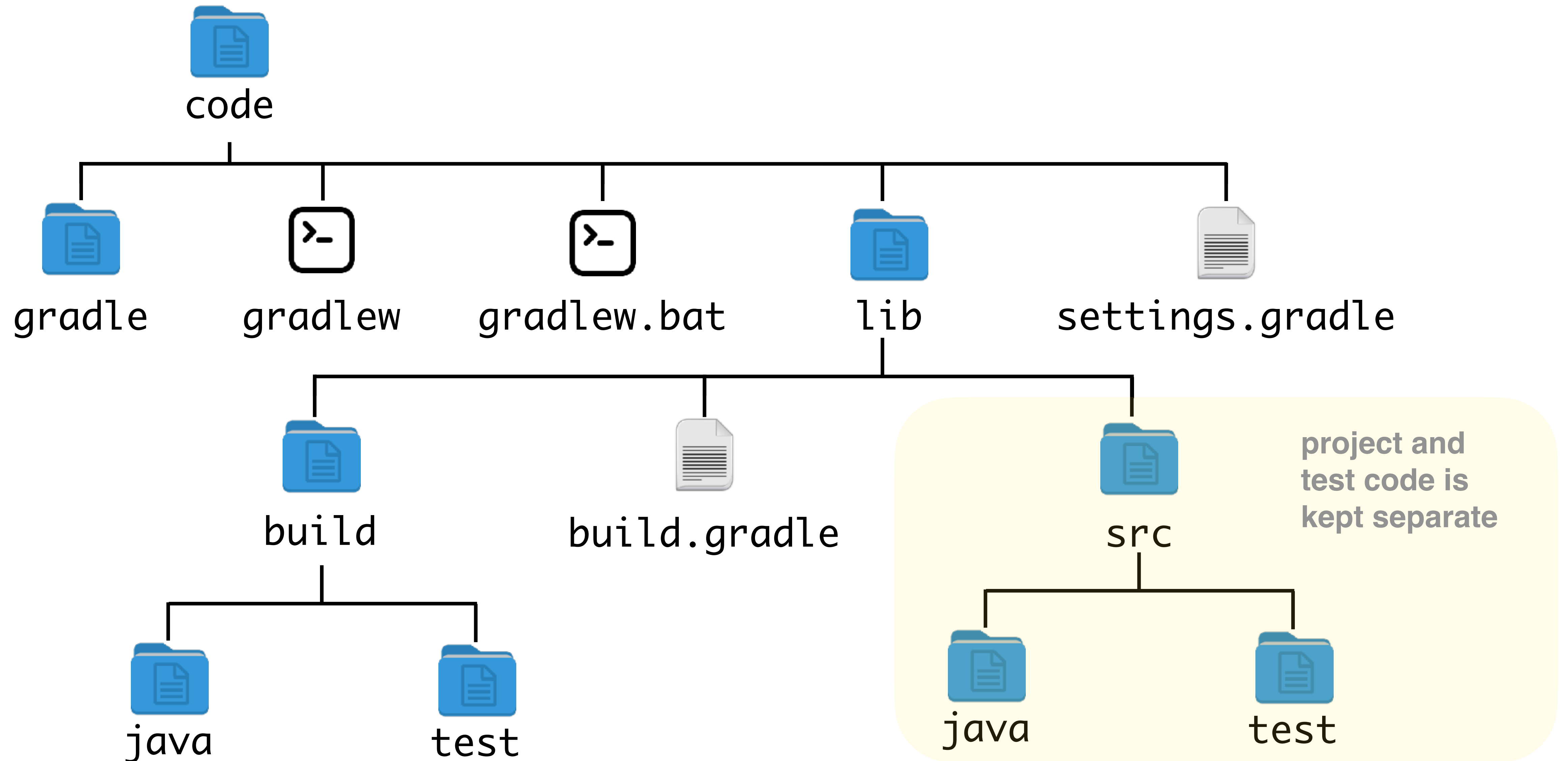


Most modern IDEs support Gradle, e.g. **IntelliJ IDEA (recommended)**

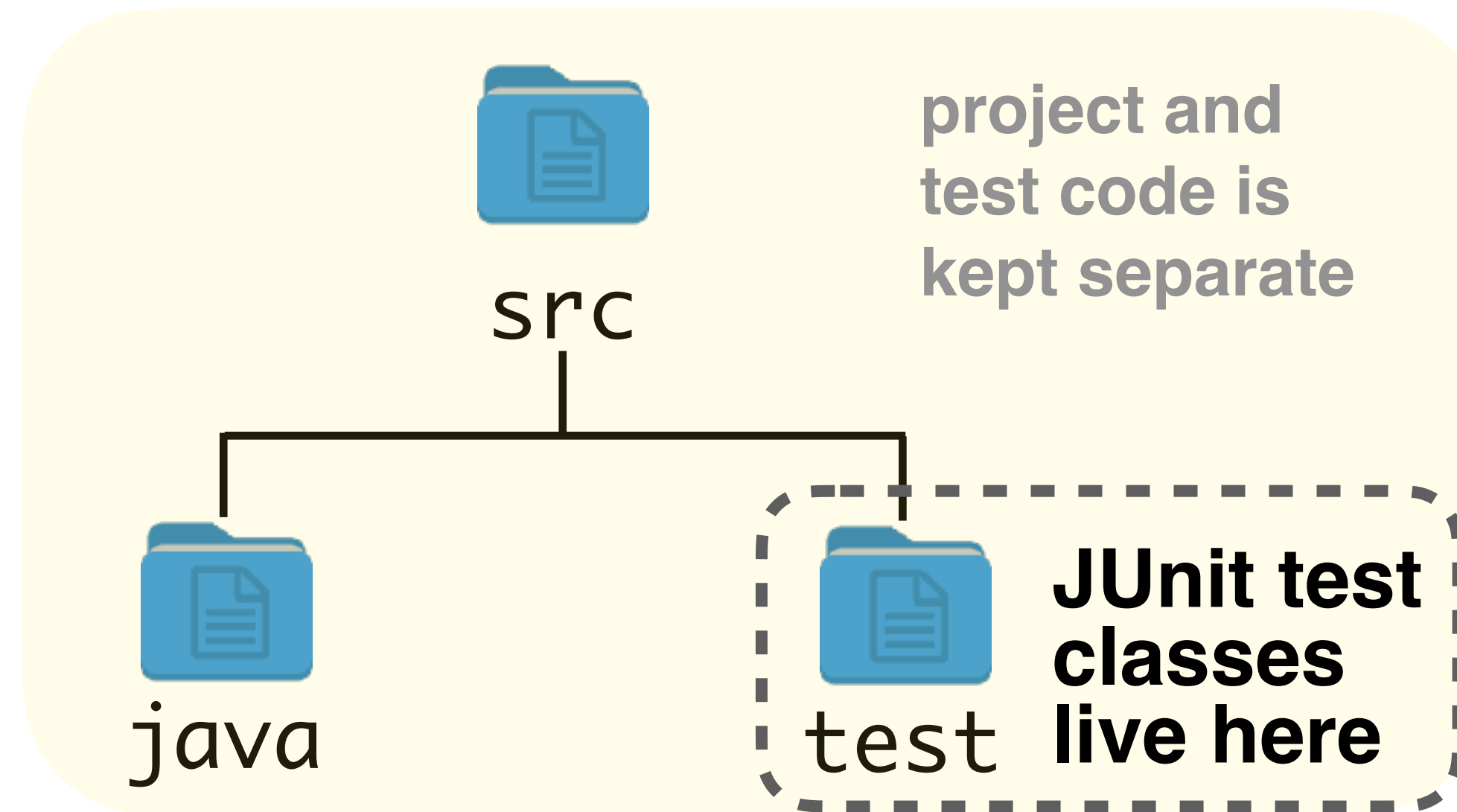
Just create a new project in the code directory and it should find the Gradle configuration.

From here, code will compile automatically and you can run specific tests through the IDE.

Gradle Project Structure



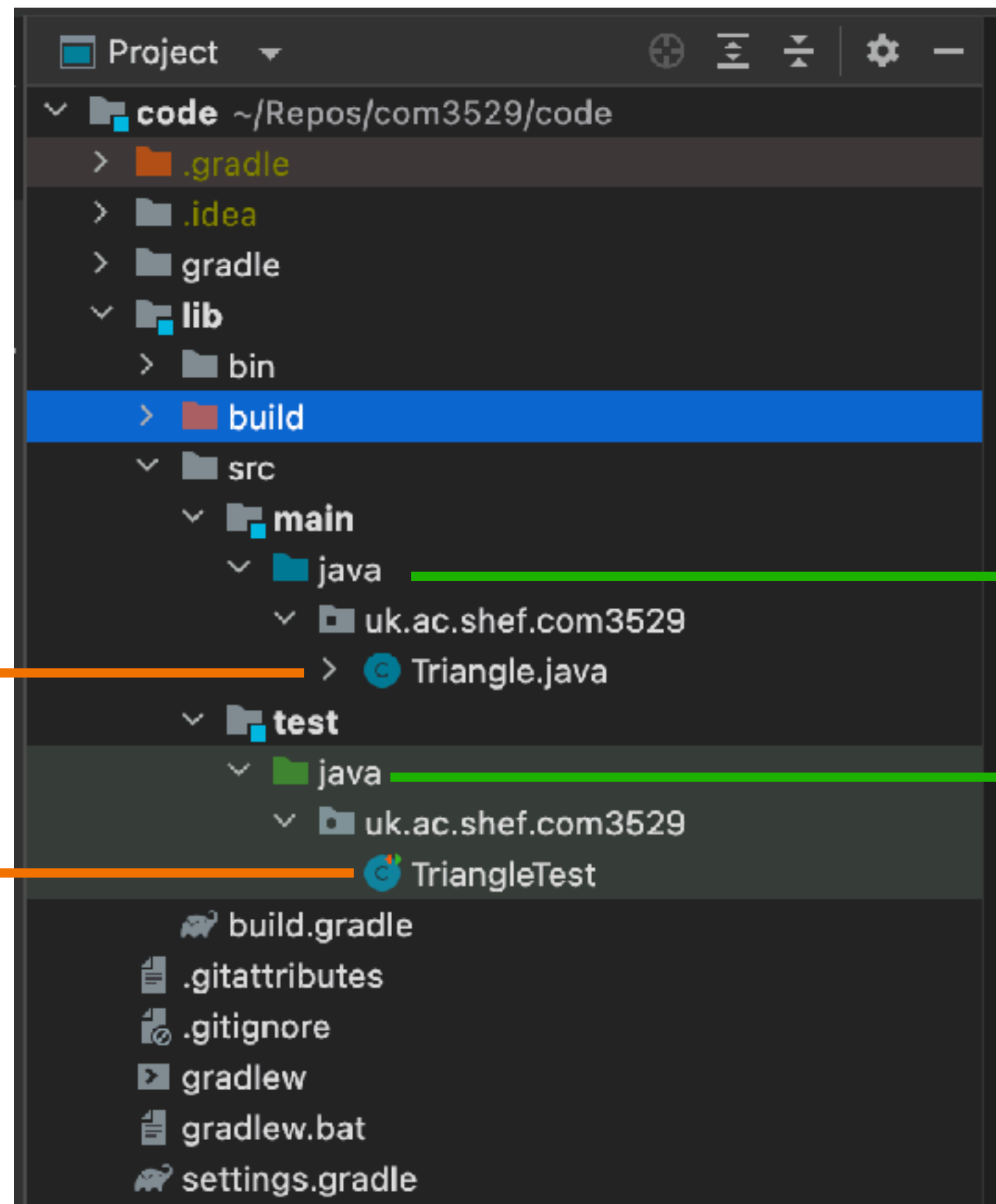
JUnit



Throughout this module, we'll be using JUnit 5

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

Let's Test!



Production code goes in `src/main/java`

Test code goes in `src/test/java`

We're going to test the `Triangle.java` class with a JUnit test class called `TriangleTest`

A JUnit Test Class and a Test

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

public class TriangleTest {

    @Test
    public void shouldClassifyEquilateral() {
        // Test code goes here...
    }

    // ...
}
```

Tests are annotated with `@Test`
JUnit then knows which methods
are test methods and which are
helper methods

The Ingredients of a Test

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

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

public class TriangleTest {

    @Test
    public void shouldClassifyEquilateral() {
        Triangle.Type result = Triangle.classify(10, 10, 10);
        assertEquals(Triangle.Type.EQUILATERAL, result);
    }

    // ...
}
```

We start by making method call(s) to set up the test and to the part of the system we want to test.

The Ingredients of a Test

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

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

public class TriangleTest {

    @Test
    public void shouldClassifyEquilateral() {
        Triangle.Type result = Triangle.classify(10, 10, 10);
        assertEquals(Triangle.Type.EQUILATERAL, result);
    }

    // ...
}
```

We then write
assertion
statements to check
the **actual result** is
the one we
expected.

JUnit Assertions

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

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

public class TriangleTest {

    @Test
    public void shouldClassifyEquilateral() {
        Triangle.Type result = Triangle.classify(10, 10, 10);
        assertEquals(Triangle.Type.EQUILATERAL, result);
    }

    // ...
}
```

The `assertEquals` method is a part of JUnit and specifically checks that some **expected value** is **equal** to the **actual one** returned from the unit being tested.

JUnit has a plethora of assertion types for checking relationships between actual and expected outputs.

These include `assertTrue(booleanVariable)`, `assertNull(reference)`, assertions on arrays and more. See:

<https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/Assertions.html>

Checking for Exceptions with `assertThrows`

```
@Test
public void shouldThrowExceptionWithInvalidTriangle() {
    assertThrows(InvalidTriangleException.class, () -> {
        Triangle.classify(0, 0, 0);
    });
}
```

```
@Test
public void shouldThrowExceptionWithInvalidTriangle() {
    Exception e = assertThrows(InvalidTriangleException.class, () -> {
        Triangle.classify(0, 0, 0);
    });
    assertEquals("(0, 0, 0) is not a valid triangle", e.getMessage());
}
```

A more elaborate version that also checks the exception message.

Arguably such checks make the test more **brittle**.

“**import static**” means importing a static method from another class and using it as if it were in the current class

Each test method is annotated with **@Test**

Assert that a method's return value is as expected with **assertEquals**

Assert that an exception is thrown as expected with **assertThrows**

```
package uk.ac.shef.com3529;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;

public class TriangleTest {

    @Test
    public void shouldClassifyEquilateral() {
        Triangle.Type result = Triangle.classify(10, 10, 10);
        assertEquals(Triangle.Type.EQUILATERAL, result);
    }

    @Test
    public void shouldClassifyIsocenes() {
        Triangle.Type result = Triangle.classify(5, 10, 10);
        assertEquals(Triangle.Type.ISOSCELES, result);
    }

    @Test
    public void shouldClassifyIsocenesWhenSidesAreOutOfOrder() {
        Triangle.Type result = Triangle.classify(10, 10, 5);
        assertEquals(Triangle.Type.ISOSCELES, result);
    }

    @Test
    public void shouldThrowExceptionWithInvalidTriangle() {
        assertThrows(InvalidTriangleException.class, () -> {
            Triangle.classify(0, 0, 0);
        });
    }
}
```

Task

Locate the implementation of the `daysBetweenTwoDates` method discussed in lectures. Its class lives in the repository:

`code/lib/src/main/java/Calendar.java`

Where should the test class live in the Gradle organisation of a Java project?

Create the class and write some initial tests for it in JUnit.

When thinking of tests to write, don't forget to test for corner cases like leap years.