COM3529 Software Testing and Analysis

Getting Started

Professor Phil McMinn

COM3529 GitHub Repository

The first half of this module is accompanied by a GitHub repository.

Each week, I will push the lecture slides, with code examples, and the weekly practical material:

https://github.com/philmcminn/com3529

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 command work on Mac/Linux. (For Windows, remove the initial "`./`")

```
./gradlew build
./gradlew test
./gradlew test --tests uk.ac.shef.com3529.TriangleTest
./gradlew test --tests uk.ac.shef.com3529.TriangleTest.shouldClassifyEquilateral
```

compile all code

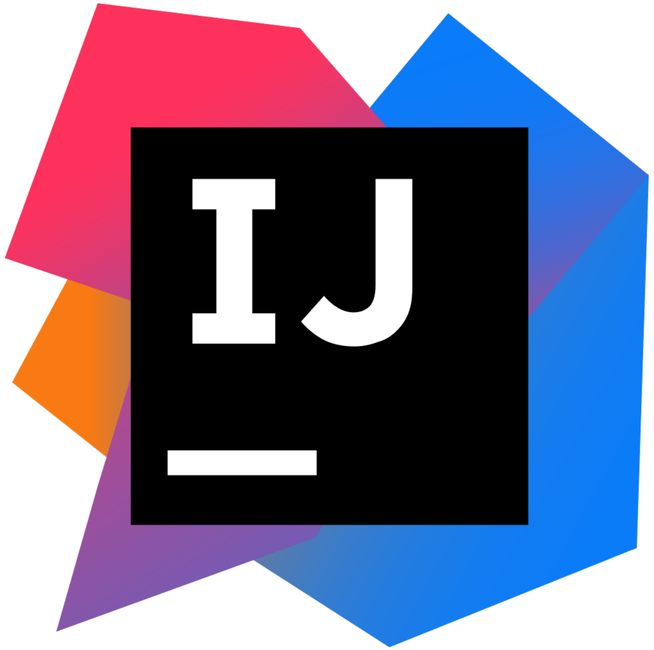
run all tests

run all tests in a specific class

run a specific test

See the Gradle website and documentation for more information:
`https://gradle.org`

Use of Integrated Developer Environments (IDEs)

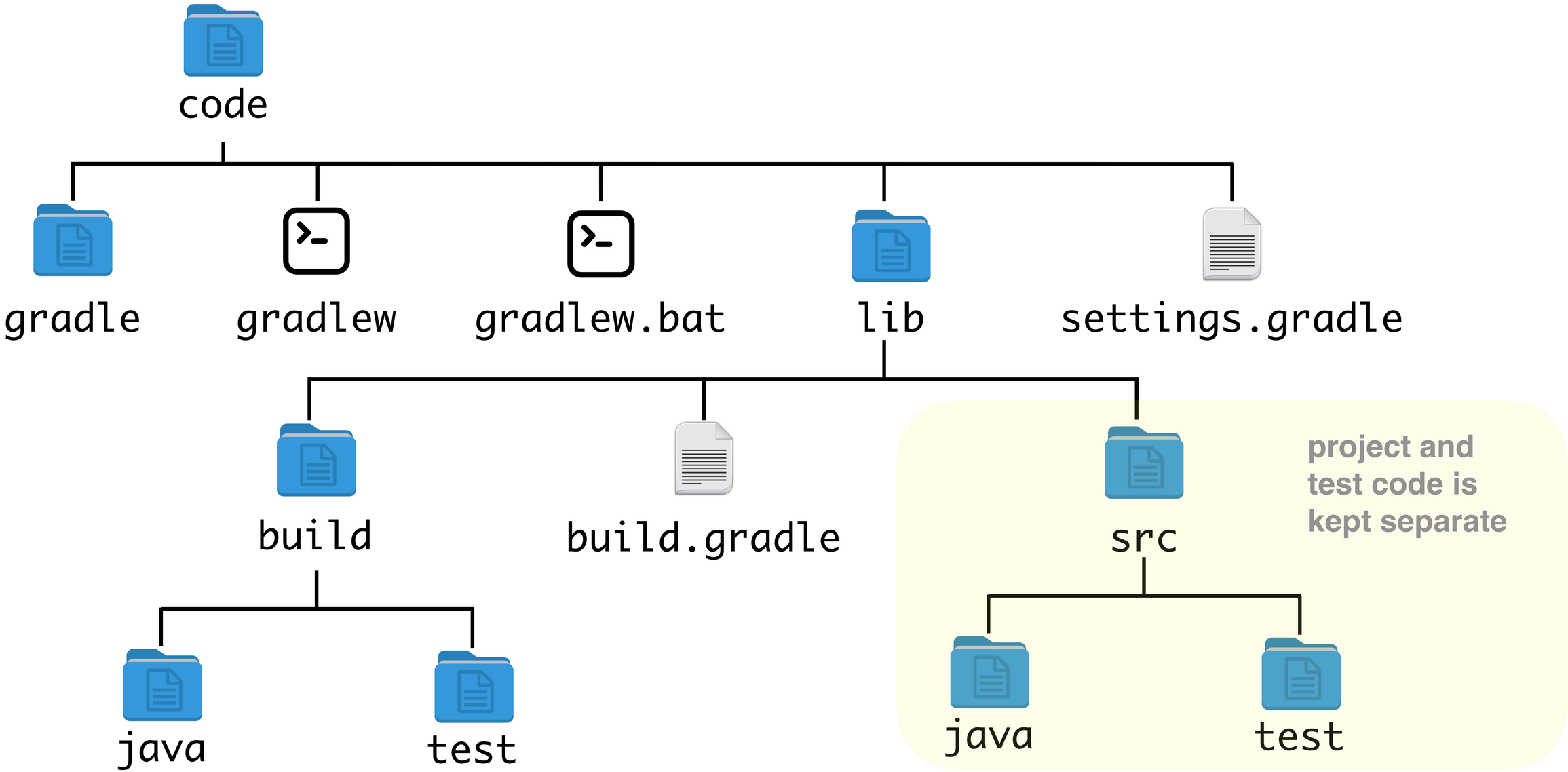


Most modern IDEs support Gradle, e.g. **IntelliJ IDEA** (recommended)

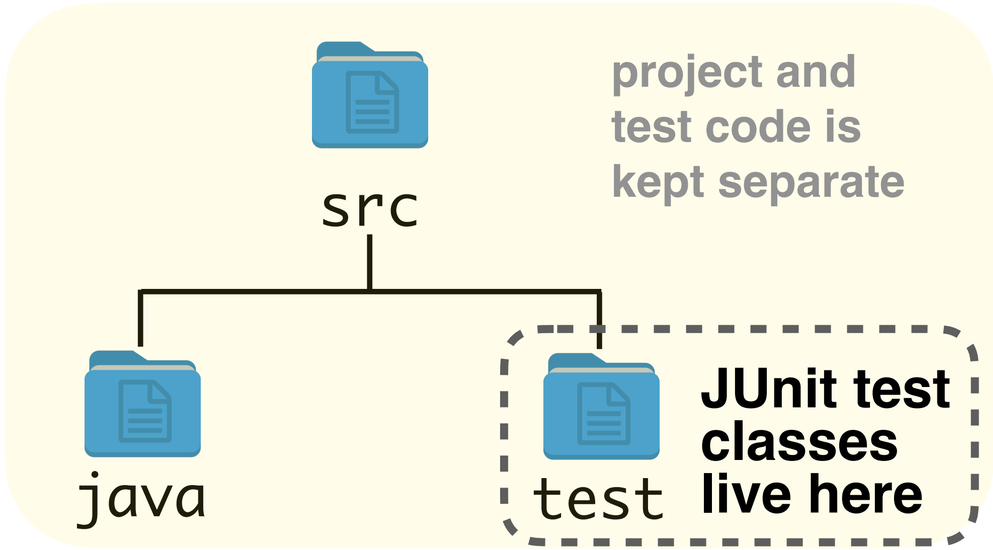
Just create a new project in the code directory and it should find the Gradle configuration.

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

Gradle Project Structure



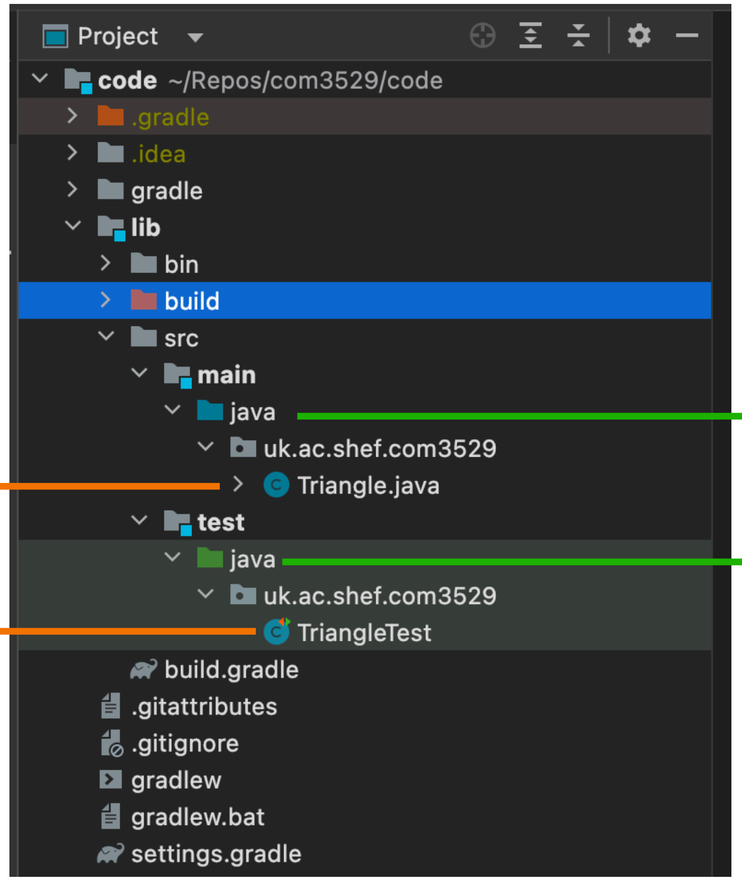
JUnit



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

```java
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

```java
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

```java
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

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

```java
@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`

```java
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 shouldClassifyIsoceles() {
        Triangle.Type result = Triangle.classify(5, 10, 10);
        assertEquals(Triangle.Type.ISOSCELES, result);
    }

    @Test
    public void shouldClassifyIsocelesWhenSidesAreOutOfOrder() {
        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);
        });
    }
}
```

Assignment 1

Details of assignment 1 can be found in the `practicals` directory of the GitHub repository – `practicals/assignment1.md`

COM3529 Assignment 1

This assignment involves the `Assignment1.java` class that is part of the Gradle project that lives in the code directory of this repository. See `code/lib/src/main/java/uk/ac/shef/com3529/Assignment1.java`.

The class contains four methods: `findLast`, `countPositive`, `lastZero`, and `oddOrPos`.

Each method has a defect. You will need to write JUnit tests for each defect and establish a fix.

Specifically, for each method in `Assignment1.java`, you will need to answer the following questions, which you'll need to submit as a PDF report. The tests you write should be added to a class called `Assignment1Test.java` (NB: When naming tests, `[methodName]` should be replaced by the name of the method you are writing the test for.)

1. (a) What and where is the defect? *[1 mark]*

   (b) Under what condition(s) do inputs to the method cause it to fail? *[1 mark]*

   (c) Write ONE JUnit test case that causes the method to fail. This should be called `[methodName]_failure`. (Note that the test should also fail - i.e., the test should have an assertion for the *correct* behaviour of the method.) *[1 mark]*