Practical Session Briefing

COMSM0086

Dr Simon Lock

Overview

Last week you encountered Maven for the first time

For each exercise we'll provide a Maven template

Avoids you having to do all the boring set up work

Means you can get on and focus on programming

There are more features we haven't covered yet

Worth spending time exploring them in more detail

Maven Recap

Maven is a cross-language build environment
It's a little bit like "make"...
But a whole lot more sophisticated !

It can be used to manage dependencies
Not only define required SW, but also INSTALL it !

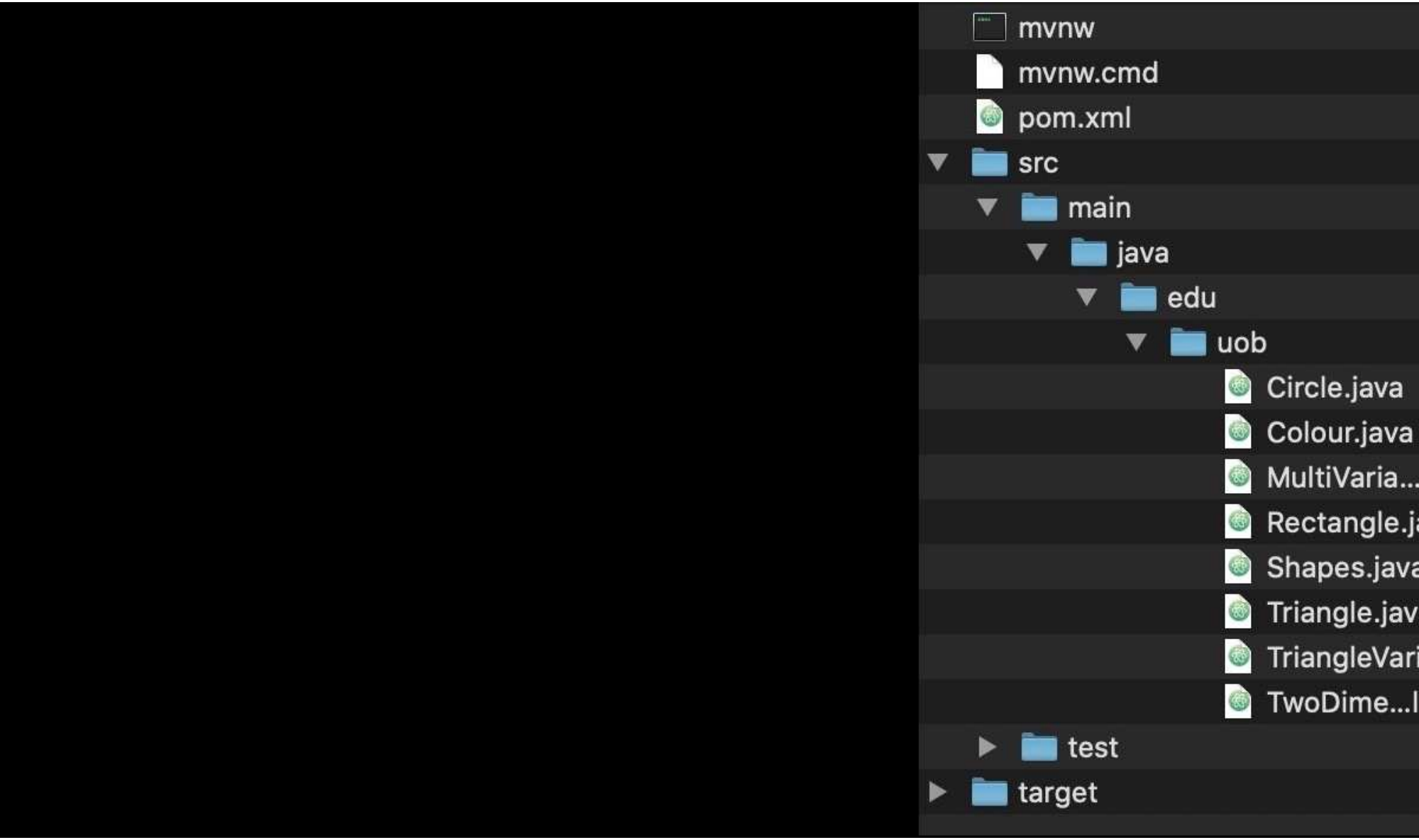
You may have noticed a lot of text scrolling
(Especially the first time you ran it)
That was Maven installing various libs and plugins

Maven in More Detail

Core to Maven is a set of configuration files
These describe a project and its dependencies
POM

Most IDEs support Maven - including IntelliJ !
So you can import/open a Maven project seamlessly

Also defines a number of standards & conventions
Including structure and content of project folder...


mvnw
mvnw.cmd
pom.xml
src
main
java
edu
uob
Circle.java
Colour.java
MultiVaria...
Rectangle.j
Shapes.java
Triangle.jav
TriangleVari
TwoDime...
test
target

Testing and Reporting

There's more to Maven than compiling & running

Maven supports various development activities

`Code Analysis, Unit & Integration Testing, Reporting...`

In this unit we make extensive use of testing tools

Test Driven Development (TDD) is a key activity

So much so that it is touched on in all three units !

This Week's Workbook

Let's take a look at this week's workbook
It's a fairly easy and straightforward workbook
You may have completed a lot of it already !

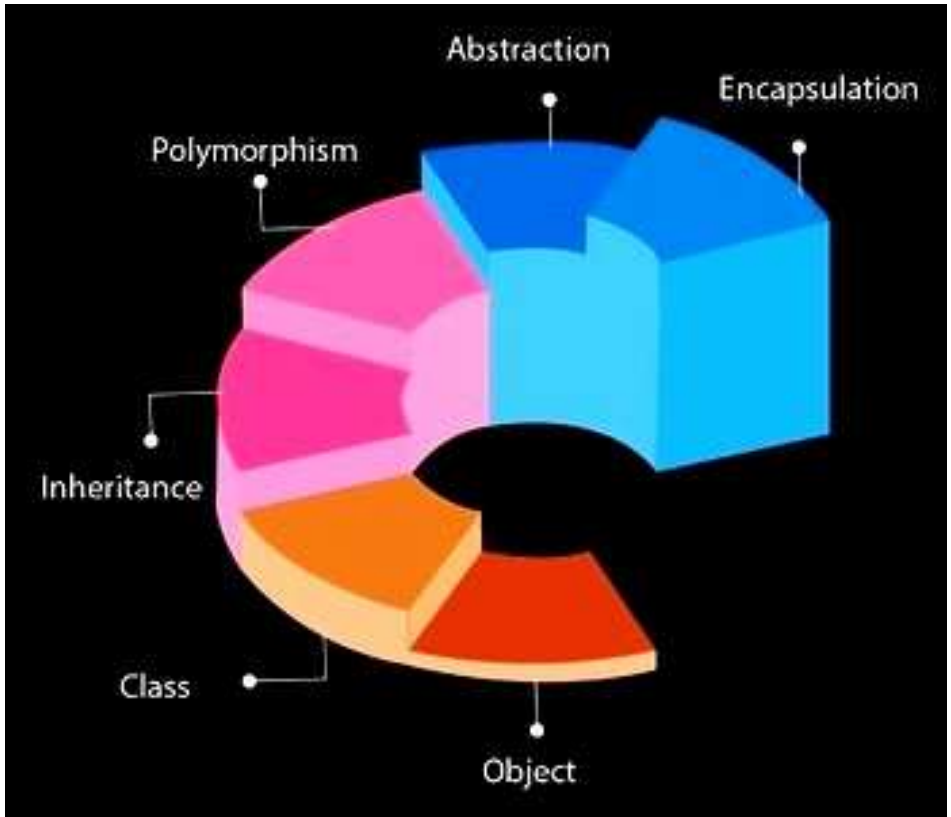
This won't be always the case for future workbooks
Unit starts off slowly, so no-one gets left behind

Let's briefly consider each task from workbook
Doesn't hurt to recap if you have already done it !

Workbook: Task 1

This task just introduces the workbook

Makes reference to a number of key topics:



Workbook: Task 2

Slides/video to refresh memory on Object & Classes

Some gentle practical tasks to achieve:

- Add a constructor method to the Triangle class
- Add 3 parameters to constructor (side lengths)
- Store side lengths as int variables ('attributes')
- Write a method to return the longest side

Add a fragment of code to test out all of the above

Create some triangles to make sure they all work

Workbook: Task 2 cont.

Add a toString method that describes the triangle:

```
This is a Triangle with sides of length 4, 5, 7
```

Note that ALL Java Objects have a toString method !

It's good practice to override default with your own

You should always try to return something descriptive

If you don't provide a toString method, you just get:

```
edu.uob.Triangle@754dd69e
```

Workbook: Task 3

Explores the topics of Inheritance and Polymorphism
Slides and Video fragments from previous lecture

Introduce Triangle into hierarchy (using 'extends')
Use polymorphism to store different shapes...
All in the same `TwoDimensionalShape` variable !

Some additional "PRO" material for deeper insight
(referencing, avoiding duplication, inheritance)

Workbook: Task 4

Explores the topics of Abstraction and Encapsulation
Again, slides/video fragments from previous lecture

Add a `Colour` variable to `TwoDimensionalShape`
Important that `Colour` is private (hidden inside)

Again, some optional "PRO" materials available
Find out more about subtleties of public/private

Workbook: Task 5

Refresher slides on "getters" and "setters"
('accessors' and 'mutators')
Extended public/private/projected PRO slides

Add setColour & getColour methods to shape class
Add colour details to string returned from toString
Where is the best place to add this code ?
(HINT: Which shapes can have a colour ?)
(HINT: Could you use overriding and chaining ?)

Workbook: Task 6

New video and slides on enumerations !

We have provided an enum for variants of triangle:

`EQUILATERAL, ISOSCELES, SCALENE, RIGHT, FLAT etc.`

Add code to your constructor to work out which it is
Implement simple variants first (e.g. EQUILATERAL)
Move on to checks for more difficult variants later

Order you check for them in the code is important
Check for "bad" variants first (ILLEGAL, IMPOSSIBLE)

Workbook: Task 6 cont.

In order to allow other objects to access the variant
You will need to add a `getVariant` method

Many other shapes may also have variants
To help, a `MultiVariantShape` interface is provided
You must 'implement' this in your Triangle class
(which involves writing a `getVariant` method)

Workbook: Task 7

Rather than manually adding/removing test code

We are going to do something more systematic !

We have provided you with a JUnit test script:

TriangleTests

Use this in IntelliJ to test your variant checker code

IntelliJ IDEA

Workbook: Task 7 cont.

Just reading test script doesn't provide much insight

So I wrote a graphical test visualiser (just for fun):

TriangleTestViewer

Created using a platform called "Processing"

(Popular Java-based audio/visual framework)

Final Few Tests

Final test method uses some very large triangles

We must remember that data types are constrained

There's a limit to range of numbers an int can store

Also, float variables have limited precision (~7 DP)

HINTS

There is a variety of primitive data types in Java

Be selective about calculations that you use

Also think about the order that you perform them

Workbook: Task 8

At some point you'll need to test via command line

Terminal
project-folder

Key Commands:

```
./mvnw clean
./mvnw compile
./mvnw exec:java
./mvnw test
```

Broken Projects !

Terminal

project-folder

Need to ensure ./mvnw script is executable

Switching platforms can corrupt ./mvnw script

The "hidden" .mvn folder can be missing

Mismatch between installed and POM version of java

Be aware of difference between "mvn" and "mvnw"

To work !