OXO Exercise Briefing

COMSM0086

Dr Simon Lock & Dr Sion Hannuna

Before we begin: Shapes Recap

A few people asked to see sample solution of Shapes
A simple exercise, but worth spending a bit of time

We won't be looking at every single line of code
A lot of it is fairly straightforward and uninteresting

A few interesting features & notable characteristics
Let's focus on just those (in no particular order)...

Overriding and Chaining

All `TwoDimensionalShapes` will have a colour

Specific shapes however have specific geometry

Where should we implement `toString` method ?

```
abstract class TwoDimensionalShape {

  private Colour shapeColour;

  public TwoDimensionalShape() {}

  public String toString() {
      return "This shape is " + shapeColour;
  }
```

```
public String toString() {
  return "Triangle with sides " + first + "," + second + "," + third + ". " + super.toString();
}
```

Multiple Constructors: With Chaining !

```
class Triangle extends TwoDimensionalShape implements MultiVariantShape {
  private int first;
  private int second;
  private int third;
  private TriangleVariant variant;
  static int population;

  public Triangle(int first, int second, int third) {
    this.first = first;
    this.second = second;
    this.third = third;
    variant = identifyTriangleVariant();
    population++;
  }

  public Triangle(int first, int second, int third, Colour colour) {
    this(first, second, third);
    super.setColour(colour);
  }
```

Variant Identification: One line each !

```
private TriangleVariant identifyTriangleVariant() {
  int longestSide = getLongestSide(first, second, third);
  // sumOfAllSides needs to be long to avoid int overflow problems
  long sumOfAllSides = ((long)first) + ((long)second) + ((long)third);
  // Check for all the "bad" variants first (otherwise we might classify as a "good" triangle first)
  if((first<=0) || (second<=0) || (third<=0)) return TriangleVariant.ILLEGAL;
  // This could have been `sumOfAllSides == longestSide*2` but risks overflowing due to large numbers
  else if(sumOfAllSides/2 == longestSide) return TriangleVariant.FLAT;
  // This could have been `sumOfAllSides < longestSide*2` but risks overflowing due to large numbers
  else if(sumOfAllSides/2 < longestSide) return TriangleVariant.IMPOSSIBLE;
  // Relatively straight-forward (and reliable) to check that all sides are equal
  else if((first==second) && (second==third)) return TriangleVariant.EQUILATERAL;
  // Isosceles check appears late - a triangle might have two identical length sides, but be illegal
  else if((first==second) || (second==third) || (third==first)) return TriangleVariant.ISOSCELES;
  // Checking for right angle triangles is a bit complex, so farm it out to a separate function
  else if(isRightAngle(first,second,third)) return TriangleVariant.RIGHT;
  // Scalenes should be near the bottom because the test for them is a bit "loose"
  else if((first!=second) && (second!=third) && (third!=first)) return TriangleVariant.SCALENE;
  // Otherwise, we don't know what type of triangle it is (theoretically, we should never get here !)
  else return null;
}
```

Right Angles: Avoid square roots & overflow

```
private boolean isRightAngle(int first, int second, int third) {
  int longestSide = getLongestSide(first, second, third);
  long firstSquared = ((long)first) * ((long)first);
  long secondSquared = ((long)second) * ((long)second);
  long thirdSquared = ((long)third) * ((long)third);
  long longestSquared = ((long)longestSide) * ((long)longestSide);
  return ( - longestSquared + firstSquared + secondSquared + thirdSquared) == longestSquared;
}

private int getLongestSide(int a, int b, int c) {
  int largest = a;
  if(b>largest) largest = b;
  if(c>largest) largest = c;
  return largest;
}
```

Random shapes: "randomInt" class method

```
public static void main(String[] args) {
    TwoDimensionalShape[] shapes = new TwoDimensionalShape[100];
    for(int i=0; i<shapes.length ;i++) {
        int randomNumber = randomInt(0,3);
        if(randomNumber==0) shapes[i] = new Triangle(randomInt(1,20),randomInt(1,20),randomInt(1,20));
        if(randomNumber==1) shapes[i] = new Circle(randomInt(1,20));
        if(randomNumber==2) shapes[i] = new Rectangle(randomInt(1,20),randomInt(1,20));
    }
    int loopTriangleCounter = 0;
    for(int i=0; i<shapes.length ;i++) {
        if(shapes[i] instanceof Triangle) loopTriangleCounter++;
    }
    System.out.println("-".repeat(30));
    System.out.println("Loop counter population is: " + loopTriangleCounter);
    System.out.println("Class variable population is: " + Triangle.getPolulationSize());
}

public static int randomInt(int lowerLimit, int upperLimit) {
    return lowerLimit + (int)(Math.random()*(upperLimit-lowerLimit));
}
```

Moving on to this week's exercise...

OXO

Next two workbooks focus on a single application

Aim is to build a noughts-and-crosses (OXO) game

It's a fairly easy activity (easier than later ones ;o)

This is NOT an assessed exercise

Let's take a look at the key features...

Model View Controller

In this unit, we often talk about 'Design Patterns'

First pattern we encounter is 'Model-View-Controller'

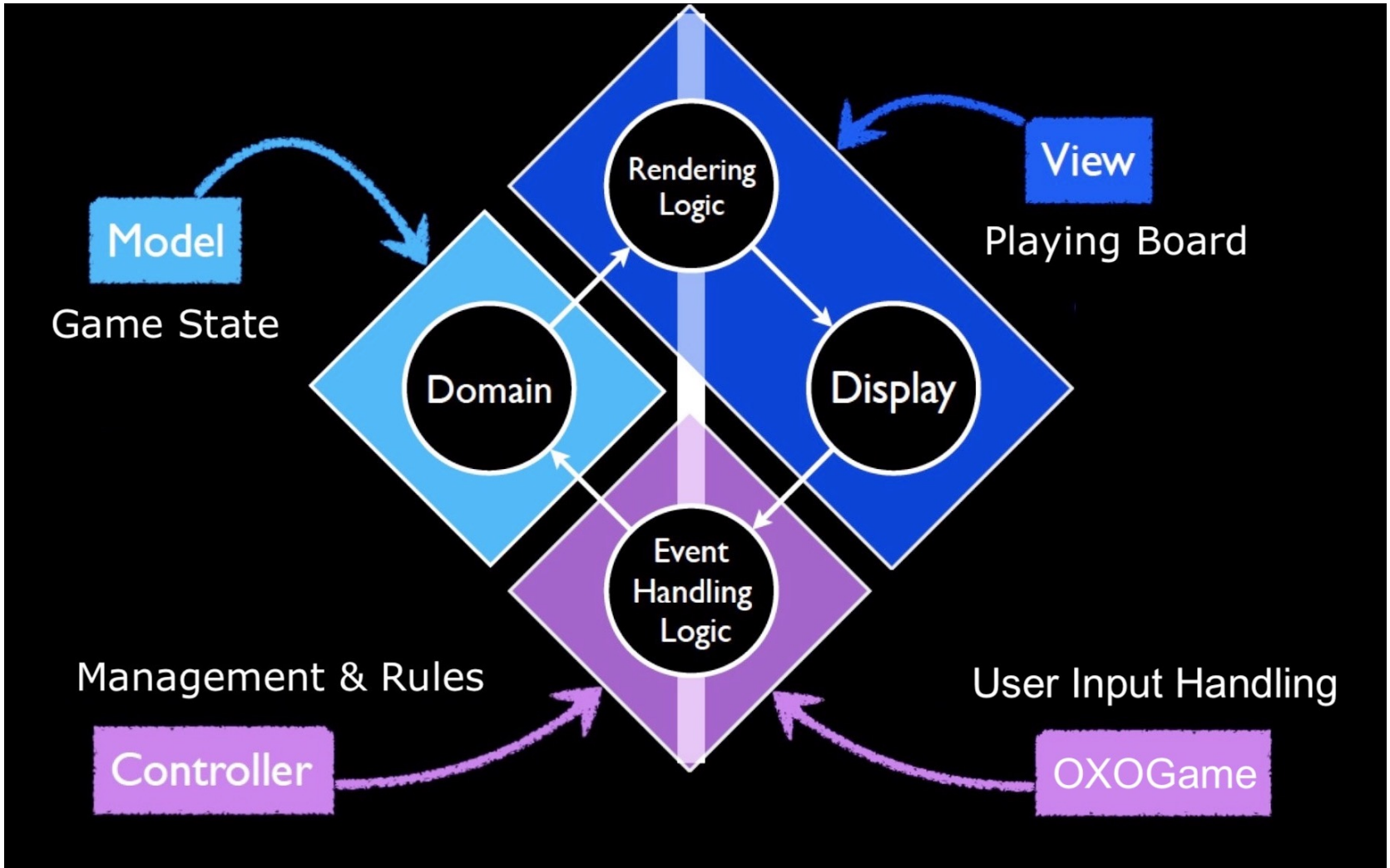
Useful for structuring interactive UI applications

Approach is to split application into 3 'components':

- Model: the core application 'state' data
- View: the bit that the user sees (GUI or text)
- Controller: interprets events and makes decisions

This separation makes change and evolution easier

MVC for OXO


Model
Game State
Rendering Logic
View
Playing Board
Domain
Display
Event Handling Logic
Management & Rules
User Input Handling
Controller
OXOGame

Fill in the gaps

A Maven template project has been provided for you
Already implemented are:

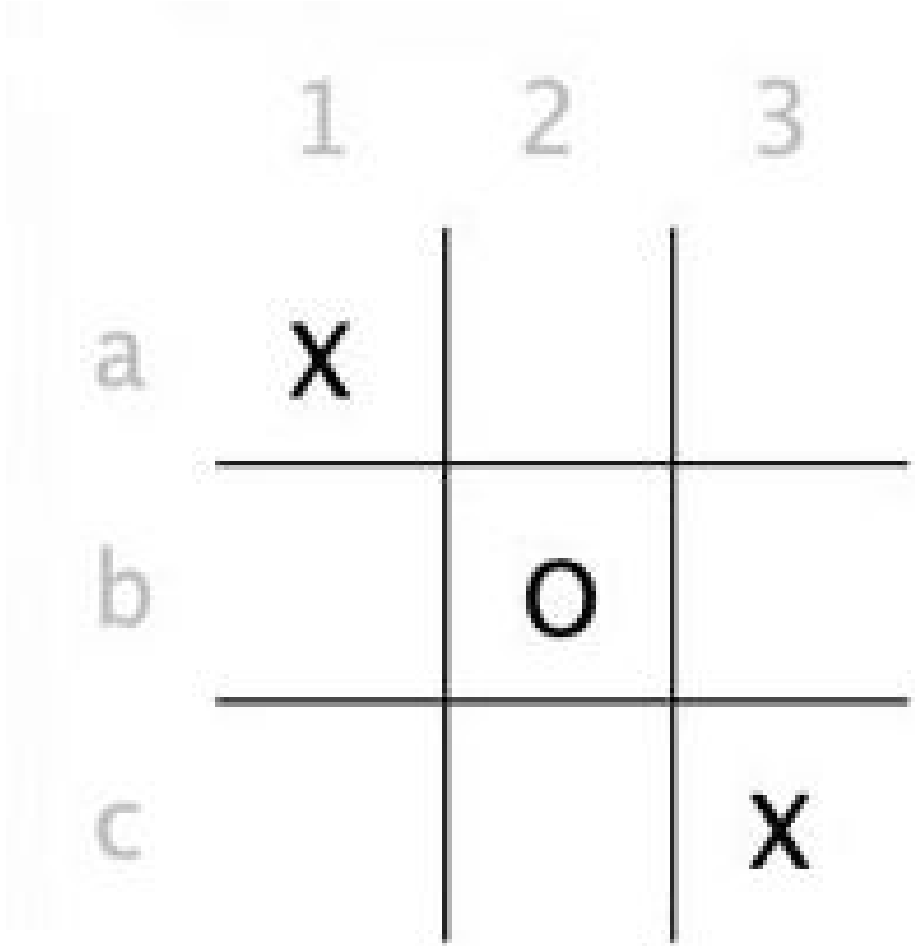
- OXOGame: Main window (that receivies user input)
- OXOView: Provides 'Rendering Logic' and 'Display'
- OXOModel: Game State from the OXO domain

OXOController is empty: your task is to complete it !
You must call OXOModel methods to change state
You don't need to interface directly with OXOView
(it monitors OXOModel and updates automatically)

What does Game State 'Model' consist of ?

- The number of cells in a row required to win
- The set of players currently playing the game
- The player whose turn it currently is
- The "owner" of each cell in the game grid
- The winner of the game (when the game ends)
- Whether or not the game has been drawn

OXOView



OXOGame

Key feature of Java: "Write Once, Run Everywhere"

Main window looks _similar_ on different platforms

User Input

Players take it in turns to make a move

Enter cell identifier into the OXOGame GUI window

Consists of row letter and column number (e.g. b2)

Inputted identifier then passed OXOController via:

```
public void handleIncomingCommand(String command)
```

Your task: interpret identifier & update game state

run-demo

Key Game Features

Win Detection

No point playing game if no one actually wins !

It's Controller's job to detect when a win is achieved

Must be able to check for wins in all directions

(horizontally / vertically / diagonally)

Note: win detection must work on grids of ANY size

Not just the standard 3x3 board !

The reason for this is...

Dynamic Board Size

It would be nice to alter board size during a game

If it became clear a game was going to be a draw

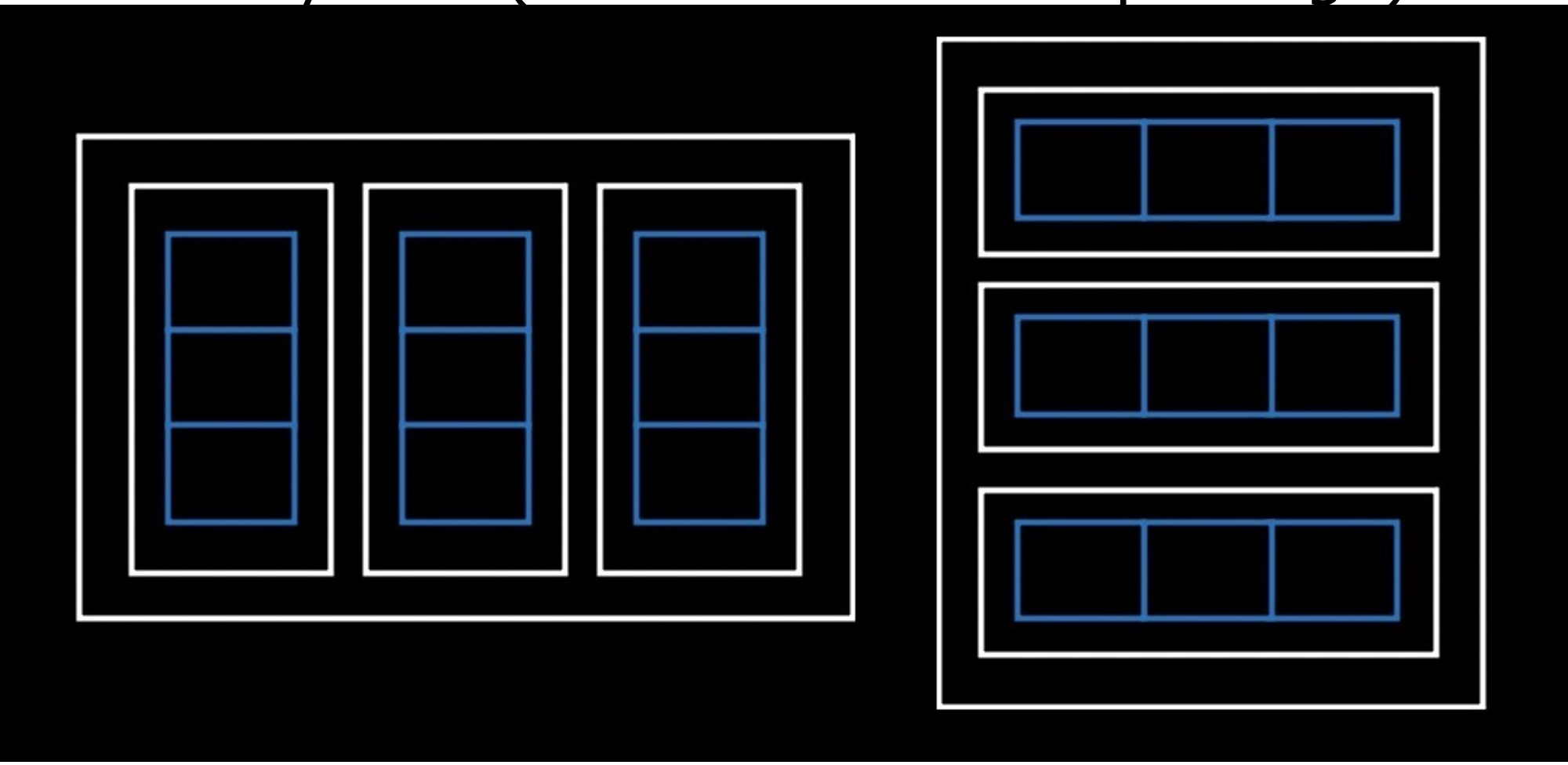
(all blank cells are used but no one has won)

Would be nice to be able to increase the board size

in order to allow play to continue

This influences data structures used in OXOModel...

ArrayLists (from 'Collections' package)



For a 3x3 game you must instantiate FOUR ArrayLists

Automated Testing

Game complex enough to need automated testing
(Manual testing soon becomes a major pain)

As "developers" it is your job to write tests scripts
Focus on testing OXOController (the bit you built)

You have been provided with a skeleton test script
Populate this with a comprehensive set of test cases

ExampleControllerTests

Warning: There are Lambdas !

Example test script contains some unusual syntax:

```
assert(Exception.class, ()-> sendCommand(), comment);
```

Lambda operator "->" is something
we have not actually covered yet
Operates like an inline function
Allows us to pass CODE "into"
an already existing method

More on this NEXT week !

Questions ?