

# Thinking in Objects

COMSM0086

Dr Simon Lock & Dr Sion Hannuna

# Aim of this Session

Our main aim is to provide some guidance on:  
"How to Think in Objects"  
(and stop thinking in functions ;o)

Whilst at the same time highlighting:  
"How to Harness the Power of Object Orientation"

# Why this ? Why now ?

You should all be familiar with basic programming  
(from your previous studies in the C unit from TB1)

Challenge this term is to think in terms of Objects  
This is new, unfamiliar and challenging to master

Previous lectures introduced building blocks of Java  
In this session we aim to put those blocks together

Going from an abstract description of a problem...  
to a suitable Object Oriented solution is not easy !

# Identifying Classes

First challenge we face is identifying some Classes  
In order to partition our code into various class files

Sometimes students will ask:

"How do I identify GOOD classes ?"

The *\*real\** answer to this question is probably:

"Practice and Experience"

This answer feels a bit like the old joke...

# Carnegie Concert Hall - New York

Lost Tourist: How do I get to Carnegie Hall ?



# Carnegie Concert Hall - New York

Lost Tourist: How do I get to Carnegie Hall ?

Passing Pianist: Practice !



# Some Real Help ?

But that answer isn't much help in learning OOP !

Identifying a set of suitable classes is not easy  
A difficult task, involving knowledge and creativity

There is no "right answer"...

But there are "good choices" & "not so good choices"

Here are a few simple tips for identifying classes...

# Identifying Suitable Classes

Collect key entities ('nouns') from 'problem domain'

Student, Triangle, Colour, Board, Player, Location

Collect key entities ('nouns') from 'solution domain'

LookupTable, NameValuePair, AddressServer, LinkedList

Collect key tasks/roles from the 'solution domain'

Include these as "doer" classes (my word ;o)

(they "do" things & their name often ends in "er")

FileParser, DataLoader, CommandHandler, ReportGenerator



# Incremental Identification

Don't expect to identify ALL classes in advance  
Some classes "emerge" during iterative development

Be prepared to refactor project as code expands:

- Splitting classes when they get complex/incoherent
- Creating additional class when new feature won't fit
- Merging classes when commonalities arise
- Wholesale restructuring when things turn ugly !!!

Design patterns can help provide a good structure  
(but we haven't really covered these yet ;o)

# Inheritance

Look out for opportunities to use inheritance !  
(But don't try overuse it !!!)

Unit assignments are designed to involve inheritance  
Real world opportunity for use are much less common

You'll often need to extend existing class hierarchies  
The need to create your own hierarchy is less common

# Illustrative Example

Let us consider a simple example application...

A graphical visualisation of GitHub groupwork

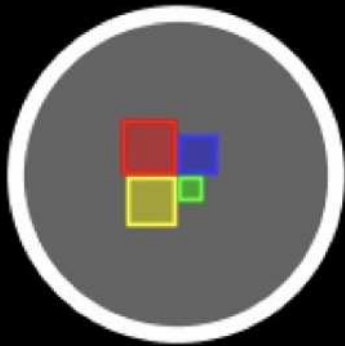
Each project involves 3-5 student contributors

We can generate reports on contributor activity  
(code commits, pull requests, change reviews etc.)

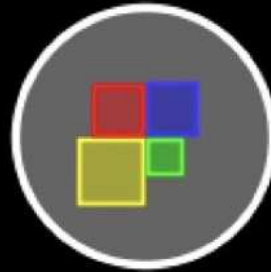
Reports are exported as separate JSON documents

Which are then be imported into our application

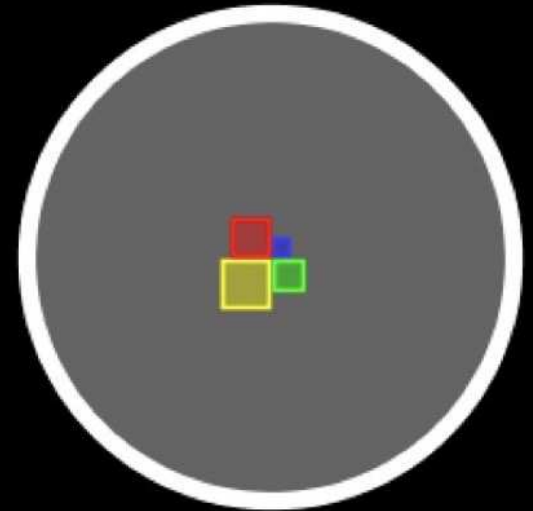
This then generates graphical representations...



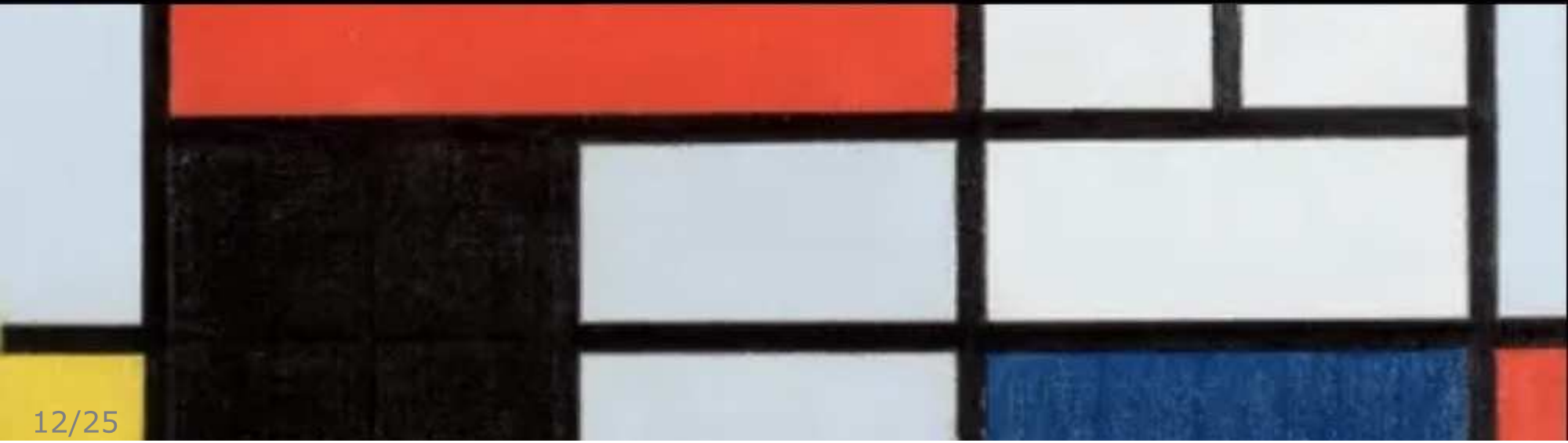
Viv (vv12345)



Una (uu12345)



Tom (tt12345)



# Implementation

First let's see a function-oriented implementation

This was written using the Processing platform  
Makes good use of Processing graphics libraries !

YES !

It IS perfectly possible to write valid Java code,  
that totally avoids Object Oriented constructs

MondrianFunctional

# Summary of Function Calling Patterns

```
setup >> scanForProjectReports
```

```
setup >> loadNextReport
```

```
draw >> drawAllStudentsInGroup...
```

```
    >> drawSingleStudent...
```

```
        >> getValueOfMetric
```

```
keyPressed >> loadPreviousReport >> loadJSONObject
```

```
keyPressed >> loadNextReport >> loadJSONObject
```

# Reflection on the code

This code works fine and does the job...

But it is not Object Oriented in structure:

- Centralised control
- Use of globals and much flow of data
- Tight linkages between functions

Let's refactor it so that it is more Object Oriented  
(which, in the end, is what we are here to learn)

Also an opportunity to explore some OO concepts  
Make use of powerful Java features along the way

# Analysis: Proposed Classes

There are two key entities from application domain:

- Project: Encapsulates details of a specific repository
- Student: Encapsulates details of an individual student

A data structure for ALL projects would be useful:

- ProjectList: Implemented by extending `ArrayList`

Also a "doer" class to scan for available reports:

- ProjectScanner: returns all reports in specified folder

MondrianObjectOriented



# Main Class

Main class is relatively simple, it just deals with:

- Setting up key parameters (window size, font etc.)
- Kicking off the scanner to look for project reports
- Filling the `ProjectList` data structure with reports
- Drawing of graphics (delegated to other classes)
- Handling of key presses from user (next/previous)

# OO Concepts: Student and Project Classes

## Delegated Responsibility

Both classes are responsible for drawing themselves

Both load JSON and populate their internal variables

## Abstraction and Encapsulation

Advanced features "hidden away" inside classes

For example: just-in-time loading and caching of data

# OO Concepts: ProjectList Class

## Abstraction and Encapsulation

Internally deals with creating & managing iterators

Internally handles checking existence of next/prev

## Inheritance and Reuse

Extends ArrayList, adding project specific features

Reuses methods from ArrayList and Java Iterators

Adds "getCurrent" method (not provided by ArrayList)

## Extension and Method Augmentation

Augments existing methods with safety features

Methods with more descriptive domain-specific names  
(e.g. "next" becomes "moveToNextProject")

# OO Concepts: ProjectScanner Class

A "doer" utility class to scan for project reports  
Returns a list of projects loaded in from filesystem

## Abstraction and Encapsulation

Messy details of File IO "hidden away" inside class

## Cohesion

ProjectScanner class focuses only on Low-level File IO  
Could be expanded with other File IO features later

# A Few Additional Notes

Function-style methods are still useful in OO code (e.g. ``loadStudentData`` and ``getValueOfMetric``)

We have combined "key entities" with "doers":  
The Student class represents the Student entity...  
...but also implements drawing & JSON data parsing

In a more complex application, we might choose to split these out into a number of separate classes

# Complexity

You could argue that we've increased complexity of the code by creating multiple additional class files (some classes are small and contain little code)

On the positive side, code is logically organised...  
It is clear where various features should be located

The whole application is currently quite simplistic  
(we removed some features from the *\*real\** code)

After a few more iterations it could become complex  
We'd soon be grateful for a well-organised structure

# Evolution

Carpenters often say: "Measure twice, cut once"  
Why can't we plan and design code in advance ?

That's a very 'waterfall' way of thinking  
Producing an entire design is time consuming  
It assumes certainty and perfect prior knowledge

Agile is more fluid and attempts to be flexible  
Encompasses exploration and "emerging features"

Better to embrace (and plan for) evolution  
Rather than enforce a rigid structure "set in stone"

## Expect Evolution

Don't feel bad when you have to refactor



Questions ?