Thinking in Objects

COMSM0086

Dr Simon Lock & Dr Sion Hannuna

Aim of this Session

The aim of this session is to provide guidance on:
"How to *Think* in Objects"
(and stop thinking in terms of functions ;o)

Whilst at the same time encouraging you to:
"Harness the Power of Object Orientation"

Why this ? Why now ?

You are familiar with fundamental programming
(from your previous studies on the C unit in TB1)

So far this TB we've implemented trivial exercises
Next workbook involves building a REAL application

Going from an abstract description of a problem...
to a GOOD *Object Oriented* solution is NOT easy !

This is where the guidance in this session will help

Identifying Classes

First challenge we face is identifying suitable Classes
In order to partition code into various source files

A question that students often ask is:

`"How do I identify GOOD classes ?"`

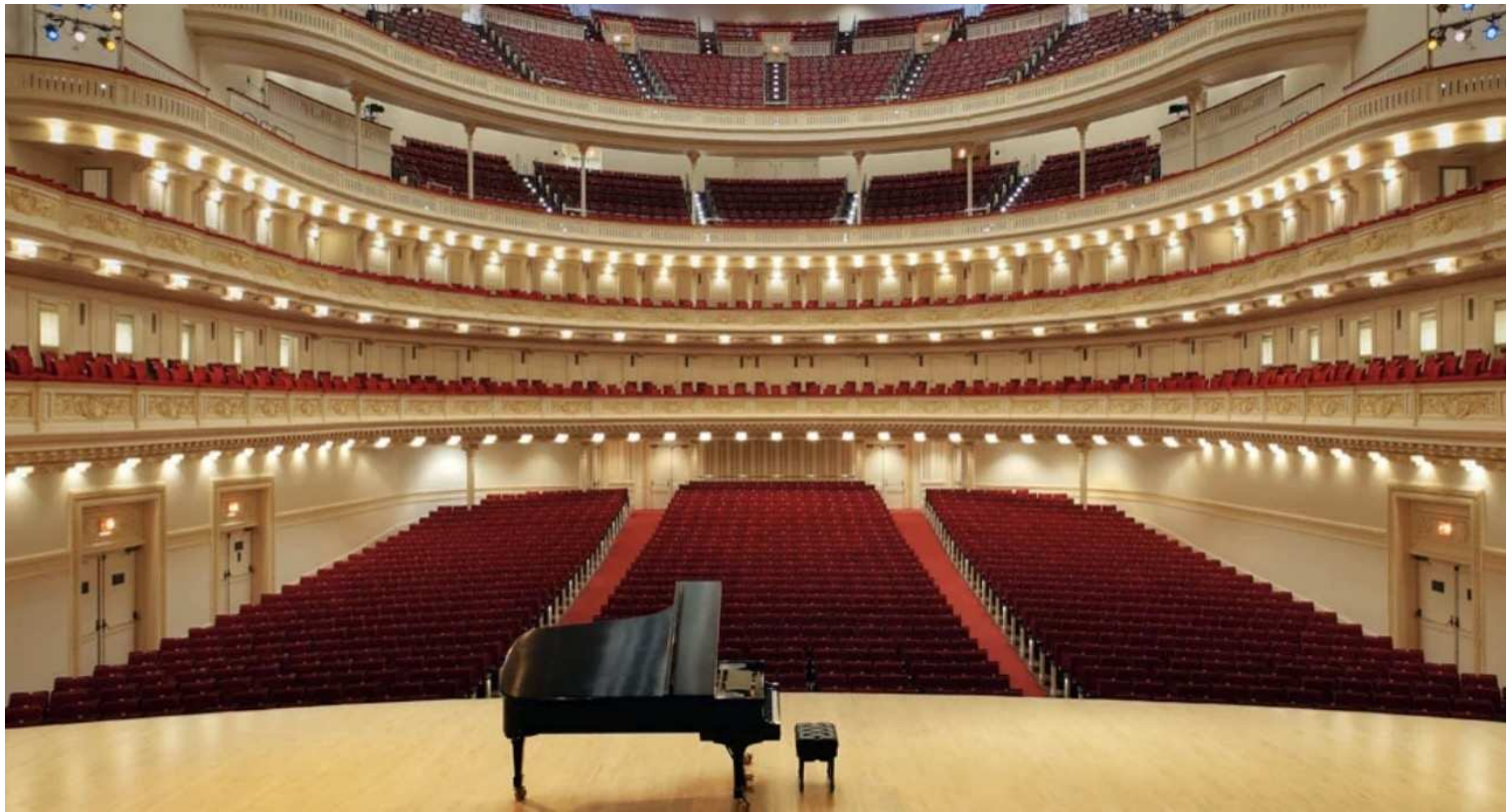
The truthful answer to this question is probably:

`"Practice and Experience"`

However, this answer feels a bit like the old joke...

Carnegie Concert Hall - New York

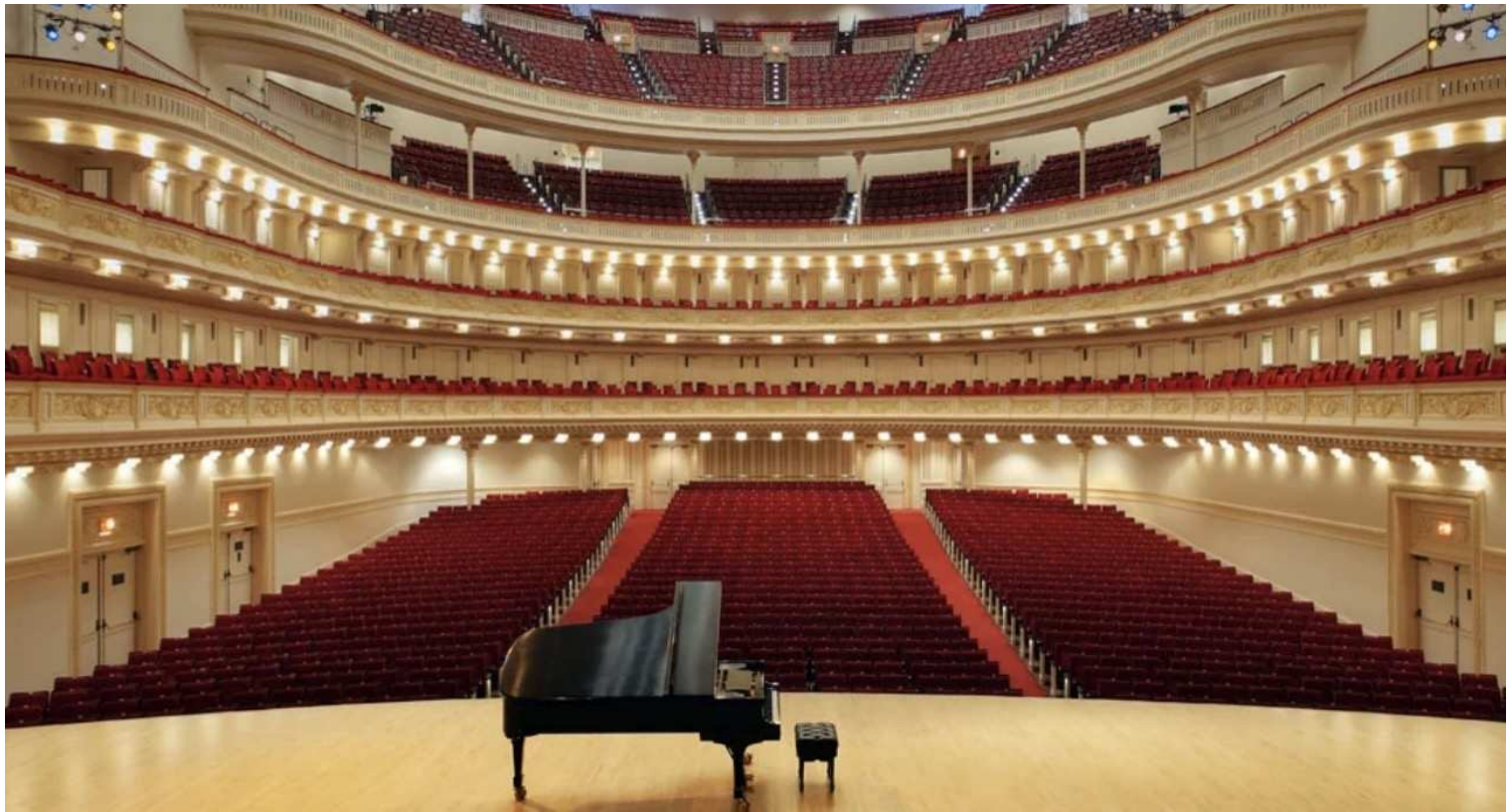
Lost Tourist: How do I get to Carnegie Hall ?



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 suitable classes is a DIFFICULT task
Involves knowledge, understanding and creativity

There is no "one 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 jobs/tasks/roles from BOTH domains
Include these as "doer" classes (my word ;o)
(they "do" things & their names often end 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
- Merging multiple classes when commonalities arise
- Creating additional class when new feature won't fit
- Wholesale restructuring when things turn ugly !!!

'Design Patterns' can help suggest suitable structure
(although we haven't really covered these yet ;o)

Inheritance

Look out for opportunities to use inheritance !
BUT be careful NOT to overuse it !!!

Unit assignments are designed to involve inheritance
(so you can gain experience applying the concept)
Real-world opportunities for use are far less common

You'll frequently need to extend existing hierarchies
The need to create your own hierarchy is less common
(unless you are building frameworks for others to use)

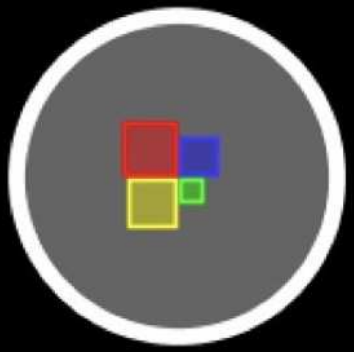
Illustrative Example

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.)

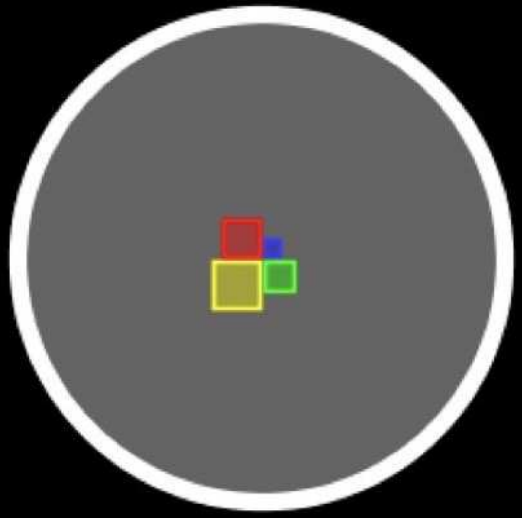
Reports are exported as separate JSON documents
These are then imported into our application
This then generates graphical representations

`GitHub Activity -> JSON Reports -> Graphic Visuals`



Viv (vv12345)

Una (uu12345)



Tom (tt12345)

Implementation

First let us explore a function-oriented implementation
This was written in Java using the Processing platform
Makes good use of Processing graphics libraries !

YES

It is perfectly possible to write valid Java code,
that TOTALLY ignores Object Oriented constructs

Don't try to understand all the code (not important)
Just try to appreciate organisational structure !

MondrianFunctional

Summary of Function Calling Patterns

```
setup >> scanForProjectReports

draw >> drawAllStudentsInGroup...
                    >> drawSingleStudent...
                                    >> getValueOfMetric

keyPressed >> loadPreviousReport >> loadJSONObject

keyPressed >> loadNextReport >> loadJSONObject
```

Reflection on the code

The code works fine and does the job as intended...
BUT it is NOT Object Oriented in structure:

- It has centralised control (one file controls all)
- There is limited delegation of responsibility
- It makes use of globals and much flow of data
- There are tight linkages between functions

Let's refactor this code to be 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 Java "power" 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 individual students

A data structure for ALL projects would be useful:

- ProjectList: "array" of project objects (not strings!)

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

- ProjectScanner: returns all reports in specified folder

MondrianObjectOriented

Main Class

A main class exists to start up the whole application

This 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
- Drawing graphics (actually delegated to other classes)
- Handling of key presses from user (next/previous)

A very short source file (only about 25 lines)

Nice and simple - pretty easy to understand !

OO Concepts: Student and Project Classes

Delegated Responsibility

Both load JSON and populate their internal variables

Both classes are responsible for drawing themselves

Abstraction and Encapsulation

Advanced features are "hidden away" inside classes

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

```
if (name==null) loadProjectData();
text(name, 7, 18);
```

OO Concepts: ProjectList Class

Abstraction and Encapsulation

Internally deals with creating & managing iterators

Internally handles checking existence of next/prev

Inheritance and Reuse

Extends and inherits methods from ArrayList

Adds 'getCurrent' method (not provided by ArrayList)

Extension and Method Augmentation

Methods "wrappered" within more descriptive names

(e.g. "next" becomes "moveToNextProject")

OO Concepts: ProjectScanner Class

A "doer" utility class to scan for project reports
Returns array of Project objects loaded 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')
Sometimes function-orientation is a good approach !

We have embedded "doers" inside "key entities":
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
Remember: Development is iterative and dynamic !

Complexity

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

Try to employ some of these ideas

when writing your own code !