Key Object Oriented Language Constructs

COMSM0086

Dr Simon Lock & Dr Sion Hannuna

Today's Aim

The aim of this session is to further explore some of the key OO concepts introduced previously

Note: for your convenience, fragments of this lecture will appear embedded within the workbook tasks
Allow you to refresh memory when attempting tasks

There are also additional slides/video in the workbooks
Be sure to view these: you'll not have seen them before

Warning

We are about to see a lot of terminology !

You won't be expected to remember it immediately
It will become familiar over the next few weeks

Any terms in 'single quotes' are official terminology
Anything in "double quotes" are my own informal words
See the next slide for some examples...

Classes

A 'Class' is a module of source code in Java
To start with, think of it like a "fancy" struct from C
We advise a 1-1 mapping between 'Class' and 'file'

```
class Counter
{
    // Code for class goes here !
}
```

Name of a class should match the name of the file
So the above would be in a file called Counter.java

Objects and Instances

'Classes' are static source code that you've written

A 'Class' describes a particular type of 'Object'

'Instances' of a class are created at run time

These live dynamic things are the 'Objects' in OOP

Process of creating live 'Objects' from 'Classes' is:

'Instantiation'

Java Types

The _majority_ of data types in Java are Classes
The exception to this are 'Primitive' types
(int, char, boolean, float etc. - note lower case !)

Primitives are just simple data (the same as in C)
Everything else in Java is a Class/Object !

Java provides the concept of an 'array' (just like C)
These can contain either 'Primitives' or 'Objects'
Arrays are homogenous - all elements of same type
(well, kinda ;o)

Attributes

A Class has a number of data fields or 'Attributes'

These are global to (accessible within) the class

Importantly NOT (usually) global to whole program

This is that notion of 'Encapsulation'

For example:

```
class Counter
{
   int count;
}
```

Methods

Classes have a bunch of functions called 'Methods'

```
class Counter
{
   int count;

   void increment() {
      count++;
   }
}
```

Such Methods are "attached" to that Class

They are called "on" a specific instance of that Class

```
Counter clubCounter;
clubCounter.increment();
```

Difference Between Functions & Methods

'Methods' are tied to a particular object
'Functions' are just "floating around"

In C you just call a function in isolation:

```
printf("Hello");
```

In Java you call a method ON a particular Object:

```
out.print("Hello");
serial.print("Hello");
file.print("Hello");
```

Standard Naming Conventions

Classes: camel case, starting with a capital

`String, RallyCar, FlyingRobot, WarmBloodedAnimal`

Objects: camel case, starting with lower case

`colour, carCounter, termTimeAddress, fluffyBunny`

Methods: camel case, starting with lower case

`draw, incrementCounter, getAddress, strokeBunny`

Don't use underscores (this isn't Python)

Constructor Methods

Special methods exist to create instances of a Class
These 'constructors' have same name as the Class

```
class Counter
{
   int count;

   public Counter() {
      count = 0;
   }
}

Counter myCounter = new Counter();
```

Notice: no return type defined for a constructor !
'public' so that it can be called from anywhere

Multiple Constructors

We can have simple constructor with default values:

```
public Counter() {
   count = 0;
}
```

Or complex ones, where we pass in some values:

```
public Counter(int startValue) {
   count = startValue;
}
```

Providing more than one method with the same name in this way is referred to as 'Overloading'

Example Class

Java has a String class to store & manipulate text
Inside there's some kind of array to store characters
(That's Abstraction and Encapsulation in action !)

A bunch of methods that "do things to" the text:
- length: gives the number of characters in the text
- charAt: gives you the char at a particular position
- contains: tells you if string contains a sequence
- toLowerCase: converts string into lower case
- substring: splits off a chunk of the string

Example Objects

We can create Objects (instances) of the String class
Each with a different character sequence inside:

```
String unitCode = new String("COMSM0103");
String unitCode = new String("COMSM0204");
```

There is (just for the String class) a shorthand:

```
String unitCode = "COMSM0305";
```

Provided because creating Strings is so common

Inheritance

Allows us to reuse & extend existing code

Take the String class for example...
Currently it is just plain text
But what if we wanted to add text styling ?
(Bold, Italics, Underline etc.)

We can 'extend' the basic String Class,
making use of all the existing methods,
but also adding in some of our own...

StyledString Class

```
class StyledString extends String
{
    void setUnderlined(boolean underline)
        // Some code in here !
    }

    void setItalics(boolean italic)
        // Some code in here !
    }
}
```

Overall Outcome

We have only added two new methods:

- `setUnderlined`
- `setItalics`

But, because we are 'extending' String, we also get:

- `length`
- `charAt`
- `contains`
- `toLowerCase`
- `etc.`

All for free !

Overriding

Adding features to a 'child' (the extending class)

Is often achieved by *replacing* an existing method

With a *new one* containing additional features

We write a new method, with a duplicate name

This replaces original method of the 'parent' class

This is called 'Overriding'

Method Chaining

If we are really clever...
We can *reuse* the method of 'superclass' (parent)
And then "tack on" the extra features at the end:

```
public String toString()
{
  String text = super.toString();
  if(isUnderlined) text = "\033[4m" + text + "\033[0m";
  if(isItalics) text = "\033[3m" + text + "\033[0m";
  return text;
}
```

Polymorphism

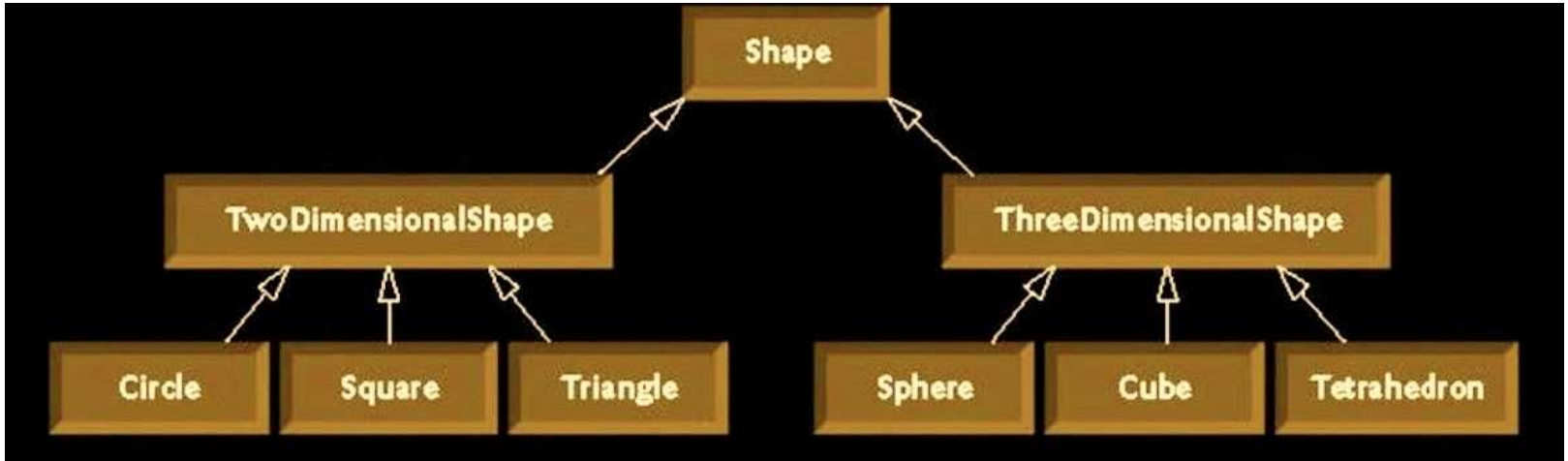
It is possible to "do things" to families of classes

Without caring exactly which class we are doing it to

For example, we can do things to ALL 3D shapes

(Such as getting volume, rotating in 3 axes etc.)

Without caring if it is a Sphere, Cube, Tetrahedron



Polymorphism in Action

What if we don't know what kind of String to expect ?

Might be a plain String, StyledString, ColourString...
Don't want to write a different method for each type

Luckily we don't have to !
We can just write a general purpose method:

```
public void spellCheck(String text) ...
```

And this will be able to operate on ANY kind of String
(Anything that is a String or 'subclass' of String)

Encapsulation

If this were C, we could just reach in
and set the style boolean attributes like so:

```
myString.isUnderlined = true;
myString.isItalics = true;
```

But this is very dangerous !
If we don't know how the object uses them...
How can we know it is safe to change them ?

DVD Collection Example

Imagine you had a collection of DVDs

Your friends can come and ask to borrow one

You can keep track of who has which disk

But what would happen if people could just walk in, and take them without asking !

There would be chaos !!!

You wouldn't know where anything was :o(

Accessor and Mutator methods

We may wish external objects to access internal data
But must ensure this is done in a controlled manner

Can be achieved by providing additional methods...
'Accessors' ("Getters") and 'Mutators' ("Setters"):

```
boolean isUnderlined = false;


void setUnderlined(boolean underline) ...
boolean getUnderlined() ...
boolean isUnderlined() ...
```

Controlling Access

We can define attributes and methods to be:

- 'public' access from any location (avoid variables)
- 'private' only inside class in which they're defined
- 'protected' inside the class OR any subclass
  (also from any class in the same package/folder)

If you don't specify any of the above, you will get:

"semi-protected" (lets not go there)

Just use 'public' or 'private' (as appropriate) for now

This probably doesn't make much sense at the moment !

Things should become clearer during the workbook

Review video fragments of this lecture if you need to

They are embedded in the workbook for this purpose !