Classes

A class is a "module" of a Java program

Typically 1-1 mapping between Class and file

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

A Class describes a particular type of Object

Objects of that type are then created at run time

Objects are "instances" of Class that describes them

Convention is that Class names start with a capital

Java Types

Sorry, but this next bit is a little confusing...
The _majority_ of data types in Java are Classes
The exception to this are "Primitive" types
(int, char, boolean, float etc)

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

Java provides the concept of an array (just like C)
These can contain either Primitives or Class types

Attributes

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

These are global to (accessible within) the class

But importantly NOT global to the whole program

(Remember the notion of "Encapsulation" ?)

For example:

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

Methods

Each Class has a bunch of functions: "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 carCounter;
carCounter.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");
```

Constructor Methods

Special methods exist to create instances of a Class

(Remember that such instances are called Objects)

Constructors have the same name as the Class:

```
class Counter
{
   int count;

   public Counter() {
      count = 0;
   }
}

Counter myCounter = new Counter();
```

Notice there is no return type with a constructor !

Multiple Constructors

We can have simple constructors with default values:

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

Or complex ones, where we passing in the settings:

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

Providing more than one method with the same name like this is referred to as "Overloading"

Problem

Sometimes you'll want to do something like this:

```
class Shape
{
    int x;
    int y;

    void setPostion(int x, int y)
    {
        x = x;
        y = y;
    }
}
```

Anything wrong with that ?

Solution

The solution is to use "this" to reference the Object:

```
class Shape
{
    int x;
    int y;

    void setPostion(int x, int y)
    {
       this.x = x;
       this.y = y;
    }
}
```

You can use "this" to access methods as well !

(call methods of an Object from inside that Object)

My Preference

Although that works, I prefer the following:

```
class Shape
{
    int x;
    int y;

    void setPostion(int xpos, int ypos)
    {
       x = xpos;
       y = ypos;
    }
}
```

I find it a bit clearer (and there is no name conflict)

Up to you which one you use !

Example Class

Java has a String class to store & manipulate text
Inside there is an array attribute to store characters
And a bunch of methods to "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 to lower case version
- substring: splits off a chunk of the string

All packaged up into a nice neat bundle - a Class !

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

Classes and Objects in Code

Methods of each Object operate on their own data
You don't want the "length" method of one String...
Telling you how long the text of another String is !

Similarly, with "substring":

```
String unitCode = new String("COMSM0103");
String unitName = "OOP with Java";
System.out.println(unitCode.substring(5,7));
```

What does the above code print out ?

Substring