```coffeescript
###

    Introduction to

        • Basic classes.

    We won't make classes very often, but we will use _existing_ classes a lot.
    Knowing a little about what a class is and how they are built will make it easier to understand how to use pre-made classes.

###


######################## P1
# A class is like a template for an object.
# A class definition describes a new category of objects.

class Cat               # The start of a "class definition" for the class 'Cat'. Note the Uppercase naming convention.
    hairballs:10        # All Cat objects will have the hairball property
    hork:->             # All Cat objects will have the hork() method
        print "HORK"

# An object based on a class is called an "instance". We create instances like this:
hobbes = new Cat() # Create an instance of 'Cat'
mittens = new Cat()
# Note camelCalse naming convention for variables that reference instances.

# An instance has all the methods and properties defined by its class.
hobbes.hork()
print hobbes.hairballs
print mittens.hairballs

# Tech Note: If you're familiar with JavaScript, you may be asking:
# "How does CoffeeScript have classes if it transpiles to JavaScript, which does not have classes?"
# Answer: Under the hood it's using JavaScript's prototype approach, it's just wrapped in key words
# that let us express things how we would in other class based languages.
```

```coffeescript
######################## P2
# Class definitions in CoffeeScript can include a special 'constructor' method.
# This method will be called, automatically, when a new instance is created.
# Other languages have similar mechanisms, though they might be called something like 'init'

class Robot
    constructor:->
        print "I'm Alive!"


r = new Robot()

######################## P3
# Constructor methods can accept arguments.
# This can be a handy way to configure an instance when we create one.

class Robot
    constructor:(givenName)->
        print "I'm Alive!"
        print "My name is " + givenName
        @name = givenName   # The '@' refers to the specific instance of Robot being constructed NOT the Robot class.
                            # This allows us to refer to the object being made when it is being made.

r = new Robot "Johnny"  # Note that we can omit the parenthesis like with normal function calls.
print r.name

# Technically, when we create an instance, we're calling a function that returns an object.


######################## P4
# When creating an instance, we can leave off the parenthesis even if we don't provide an argument.

class Derp
    isADerp:true

d = new Derp    # The 'new' keyword is enough of a context clue for CofeeScript to know what we want.
print d
```

```coffeescript
######################## P5
# Constructor methods can accept objects as arguments just like any other function.
# This allows us to configure complex objects without having to remember the exact order that arguments should go in.

class Box
	constructor:(size)->
		@w = size.width
		@h = size.height
		@d = size.depth

	getVolume:->
		return @w * @h * @d

b1 = new Box({width:10,height:10,depth:10})
b2 = new Box({height:20,depth:20,width:20})

print b1.getVolume()
print b2.getVolume()
```

```
######################## P6
# The same exact same program using object notation shortcuts.

class Box
    constructor:(size)->
        @w = size.width
        @h = size.height
        @d = size.depth

    getVolume:->
        @w * @h * @d    # The results of this math are automatically returned


b1 = new Box            # Create a Box instance, while passing the constructor a configuration object.
    width:10            # This approach is VERY common to CoffeeScript and Framer.
    height:10           # Study it carefully.
    depth:10

b2 = new Box
    width:20
    height:20
    depth:20

print b1.getVolume()
print b2.getVolume()

########################
######################## End / Note
########################
###

    Functions and Classes (in whatever form) allow developers to create complex and reusable building blocks.
    A collection of these building blocks is called a "library" or a "framework".
    FramerJS is itself such a "library".
    Other JavaScript libraries include: jQuery, D3, P5, React, Three.js. There are many more though.
    Framer is a little peculiar because it is written in CoffeeScript.
    That's a topic for another day though...

###
```