```
###

-   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 camelCase 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

# NOTE: Classes do not create visuals.
# Framer's Layer class creates visuals only because it has code in it that
# inserts DIV elements into the page and modifies their CSS. We will discuss
# how to extend this capability in the advanced section below.

# 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()

```

```coffeescript
######################## 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.
```

```
70  ######################## P4
71  # When creating an instance, we can leave off the parenthesis even if we don't provide an argument.
72
73  class Derp
74      isADerp:true
75
76  d = new Derp    # The 'new' keyword is enough of a context clue for CofeeScript to know what we want.
77  print d
78
79
```

```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()
```

```
99  ######################## P6
100 # The same exact same program using object notation shortcuts.
101
102 class Box
103     constructor:(size)->
104         @w = size.width
105         @h = size.height
106         @d = size.depth
107
108     getVolume:->
109         @w * @h * @d    # The results of this math are automatically returned
110
111
112 b1 = new Box            # Create a Box instance, while passing the constructor a configuration object.
113     width:10            # This approach is VERY common to CoffeeScript and Framer.
114     height:10           # Study it carefully.
115     depth:10
116
117 b2 = new Box
118     width:20
119     height:20
120     depth:20
121
122 print b1.getVolume()
123 print b2.getVolume()
124
```

```
########################
######################## 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...

###
```

```
########################
######################## Advanced : Extending Framer's Layer
########################

######################## P7
# You can create classes that have all the behaviors of Layer
# but with small tweaks of your choosing.

class Box extends Framer.Layer
    constructor: ->
        super()                     # Here we run the constructor function of the parent class.
        @width=10                   # Now, our Box instance has all the properties and behaviors of a Layer
        @height=10                  # By setting these properties here, every Box instance will have
        @backgroundColor="red"      # the same default values.

b1 = new Box
b2 = new Box
b2.x = 11

#b3 = new Box
#   rotation:45                    # NOTE: This will not work yet!
```

```coffeescript
######################### P8
# To make sure we can create our boxes just like we create layers our constructor
# must accept a configuration object and pass it to super()

class Box extends Framer.Layer
    constructor: (config)->
        super(config)                   # Here we run the constructor function of the parent class.
        @width=10                       # Now, our Box instance has all the properties and behaviors of a Layer
        @height=10                      # By setting these properties here, every Box instance will have
        @backgroundColor="red"          # the same default values.

b1 = new Box
    rotation:45

b2 = new Box
    rotation:45
    x:11
```

```coffee
######################## P8
# We can define behaviors in our classes too.

class Box extends Framer.Layer
    constructor: (config)->
        super(config)
        @onMouseDown ->          # As part of the constructor function we add event handlers.
            @animate
                rotation:45
        @onMouseUp ->
            @animate
                rotation:0

b1 = new Box
b1.center()
```