Method Lookup Path

Achieving inheritance with ruby classes and ruby modules is clear till now. Now let's put them both together to see how that affects the method lookup path. Method lookup path is the order in which classes are inspected when you call a method.

```
module Walk
  def walk
    "I can walk"
  end
end
module Swim
  def swim
    "I can swim"
  end
end
module Eat
  def eat
    "I'm eat"
  end
end

class Animal
  include Swim
  def speak
    "Hi I  can speak"
  end
end
```

```
puts  Animal .  ancestors
Animal
Swim
Object
Kernel
BasicObject
```

This means that when we call a method of any Animal object, first Ruby looks in the Animal class, then the Swim module, then the Object class, then the Kernel module,and finally the BasicObject class.

```
animal  =   Animal.new
animal.speak                   #   "Hi  I   can  speak"
```

Ruby found the speak method in the Animal class and looked no further.

```
animal.swim                    #   =>  "I  can  swim"
```

Ruby first looked for the swim instance method in Animal, and not finding it there, kept looking in the next place according to our list, which is the Swim module. It saw a swim method there, executed it, and stopped looking further.

```
animal.eat
 NoMethodError :   undefined  method  `eat'  for
#<Animal:0x00000000ab42f8>
from  (  irb ): 32
```

Ruby traversed all the classes and modules in the list, and didn't find an eat method, so it threw an error.

Let’s take another example:

```
class Dog < Animal
  include Walk
  include Eat
End
puts GoodDog.ancestors
Dog
Eat
Walk
Animal
Swim
Object
Kernel
BasicObject
```