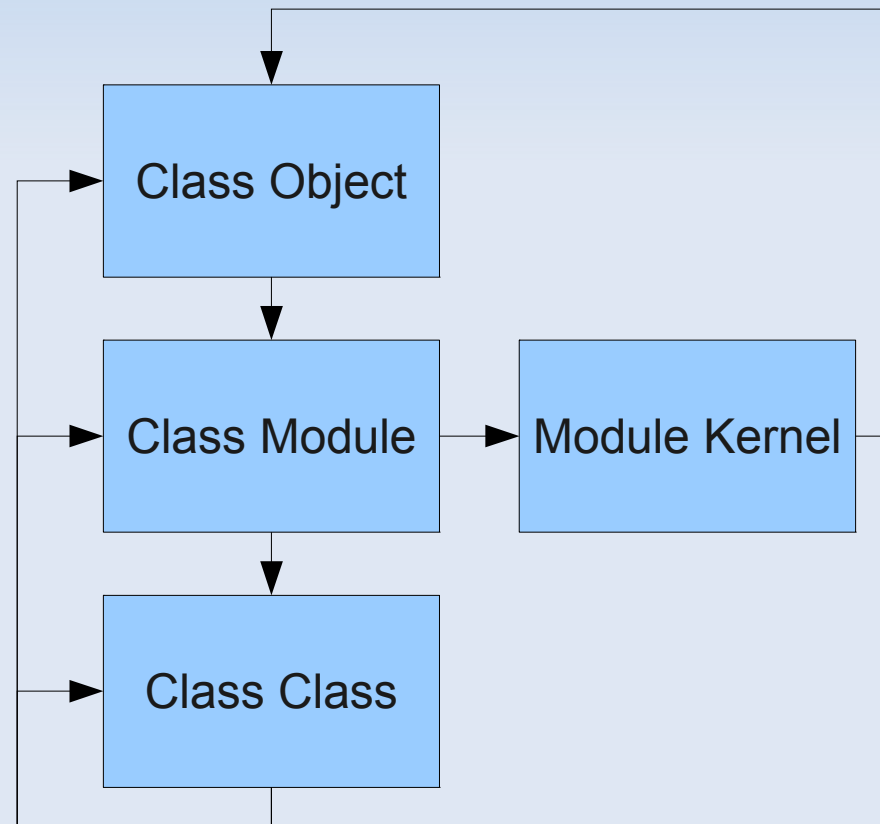


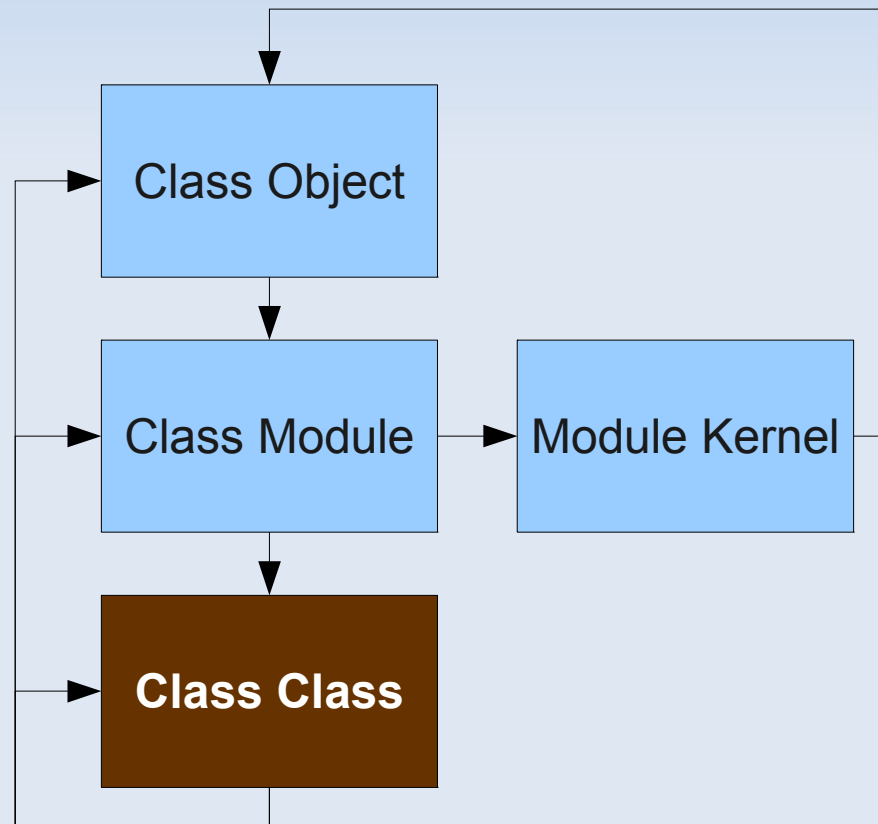
Core of the Core

A series on the **core** of the Ruby Object Model



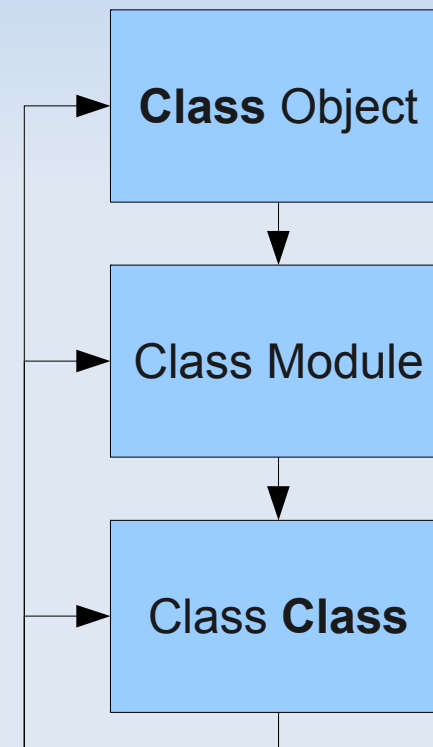
Part 1: Class Class

- A look into the secret identity of an Object



The Identity of an Object

- **Ruby is a pure object-oriented language**
- **Object is a Class**
(by essence)
- **Class is an Object**
(through inheritance)



The Identity of an Object

That's

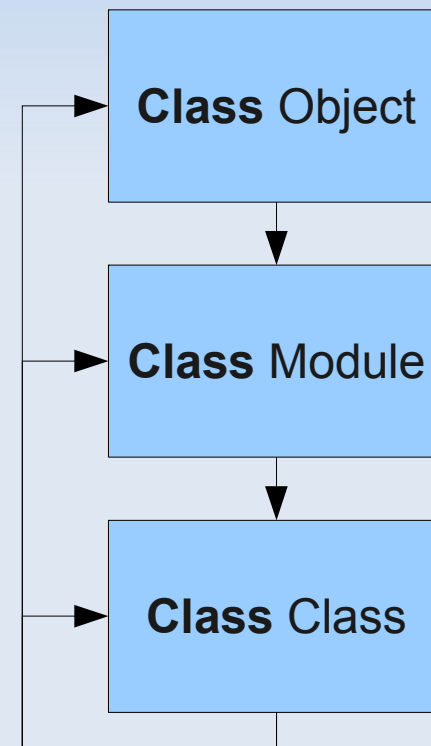
so

META!

The Identity of an Object

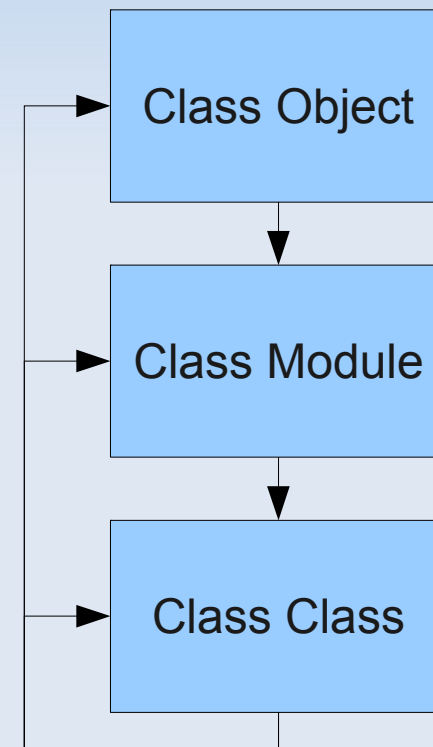
- Object is a **Class**
- Module is a **Class**
- Class is a **Class**

```
> Object.class  
] Class  
> Module.class  
] Class  
> Class.class  
] Class
```



The Identity of an Object

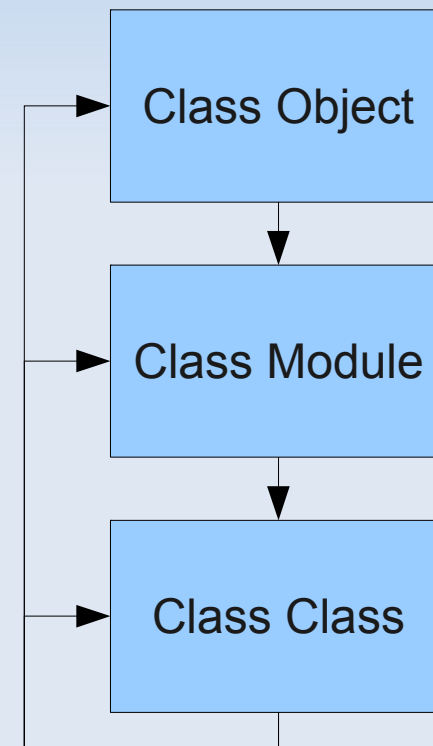
- It seems like Ruby is **Class-Oriented**.
- Why would Ruby be considered **Object-Oriented**?



The Identity of an Object

- Though everything is a Class (even Modules)
- Everything inherits from Object

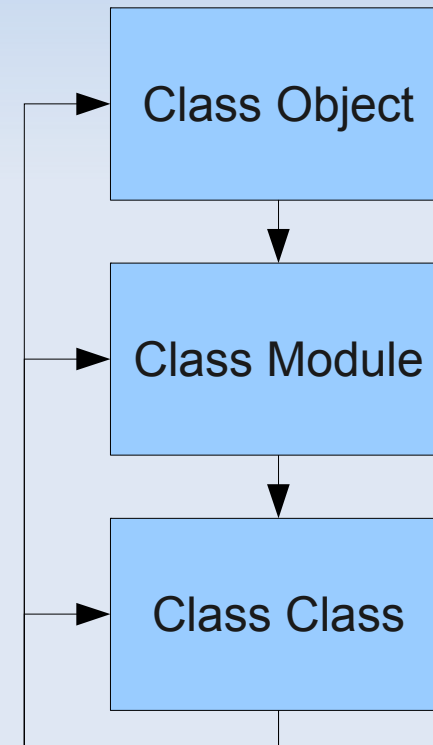
```
> Object.new.kind_of?(Class) ] false
> Module.new.kind_of?(Class) ] false
> Class.new.kind_of?(Class) ] true
> Object.new.kind_of?(Module) ] false
> Module.new.kind_of?(Module) ] true
> Class.new.kind_of?(Module) ] true
> Object.new.kind_of?(Object) ] true
> Module.new.kind_of?(Object) ] true
> Class.new.kind_of?(Object) ] true
```



The Identity of an Object

- To make things more confusing, everything in the core is somehow wrapped together.

```
> Class.kind_of?(Class)      ] true
> Class.kind_of?(Module)    ] true
> Class.kind_of?(Object)    ] true
> Object.kind_of?(Object)   ] true
> Object.kind_of?(Module)   ] true
> Object.kind_of?(Class)    ] true
> Module.kind_of?(Module)   ] true
> Module.kind_of?(Class)    ] true
> Module.kind_of?(Object)   ] true
```



Class Methods of Class Class

`Class.new` is the only non-inherited Class Method of the Class `Class`.

- `Class.new`

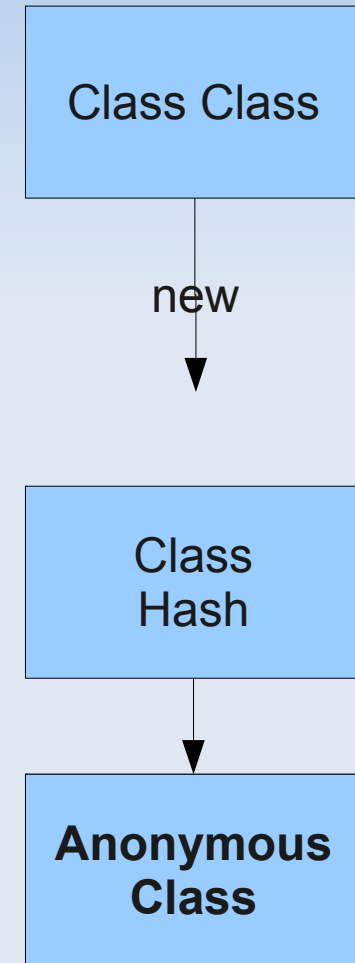
Class Methods of Class Class

Class.new

- Creates a new anonymous (unnamed) class with the given superclass (or Object if no parameter is given). You can optionally use a block to open up the class temporarily

```
> Class.new(Hash)
] #<Class:0xb772cb14>

> Class.new(Hash) do
>   def symbol_keys
>     keys.find_all {|k| k.is_a?(Symbol)}
>   end
> end
```

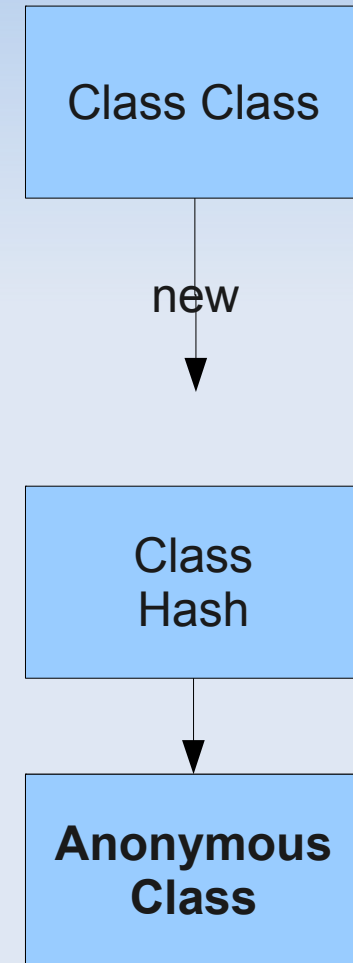


Class Methods of Class Class

More about Class.new

- If you want your anonymous class to be reusable you can assign it to a variable.

```
> my_hash = Class.new(Hash)
] #<Class:0xb7612904>
> my_hash.new
] {}
```



Class Methods of Class Class

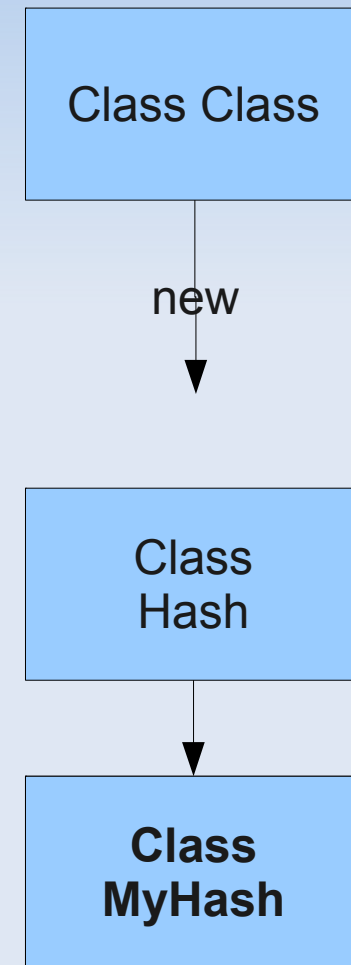
More about Class.new

- When you assign the class to a constant, the resulting Class object loses its anonymity.

```
> MyHash = Class.new(Hash)
] MyHash
> MyHash.new
] {}
```

- This is essentially the same as a standard class definition.

```
> class MyHash < Hash
] MyHash
> MyHash.new
] {}
```



Instance Methods of Class Class

An instance of the Class Class is a Class.

Instance Methods of the Class Class look like Class Methods in the instantiated Class.

- `class.allocate`
- `class.new`
- `class.superclass`
- `class.inherited`

Instance Methods of Class Class

class.allocate => an Object

- Allocates space for a new object of class's class. The returned object must be an instance of class.

```
> Class.allocate
```

```
] #<Class:0xb7780f34>
```

```
> Class.new.allocate
```

```
] #<#<Class:0xb776af2c>:0xb776a734>
```

```
> Module.allocate
```

```
] #<Module:0xb775c88c>
```

```
> Module.new.allocate
```

```
NoMethodError: undefined method `allocate' for #<Module:0xb774ae98>
```

```
> Object.allocate
```

```
] #<Object:0xb768f468>
```

```
> Object.new.allocate
```

```
NoMethodError: undefined method `allocate' for #<Object:0xb7679c44>
```

Instance Methods of Class Class

`class.new(args,...) => an Object`

- Calls `allocate` to create a new object of class's class, then invokes that object's `initialize` method, passing it `args`. This is the method that ends up getting called whenever an object is constructed using `.new`.

```
> class Hash
>   def initialize(*args)
>     args.each do |arg|
>       self[arg] = arg
>     end
>   end
> end
] nil

> Hash.new(:a, :b, :c)
] {:b=>:b, :c=>:c, :a=>:a}
```

Instance Methods of Class Class

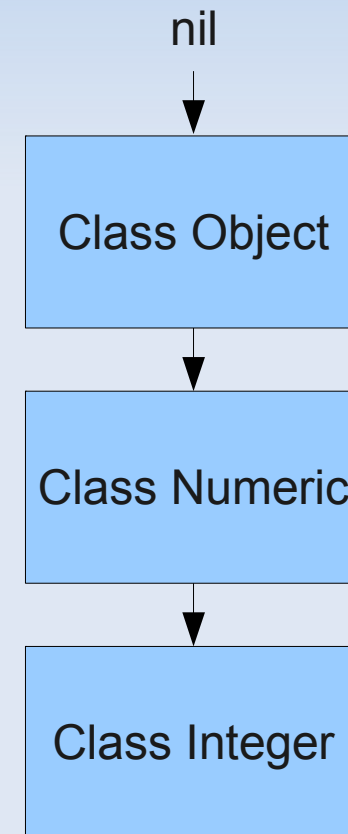
`class.superclass => a superclass or nil`

- Returns the superclass of class, or nil.

```
> Integer.superclass  
] Numeric
```

```
> Numeric.superclass  
] Object
```

```
> Object.superclass  
] nil
```



Instance Methods of Class Class

class.inherited(subclass)

- Callback invoked whenever a subclass of the current class is created.

```
class Object
  def self.inherited(subclass)
    puts "#{subclass} is a #{self}"
  end
end
```

```
class Animal; end
# Animal is a Object
```

```
class Bird < Animal; end
# Bird is a Animal
```

```
class Swallow < Bird; end
# Swallow is a Bird
```

Instance Methods of Class Class

More about `class.inherited(subclass)`

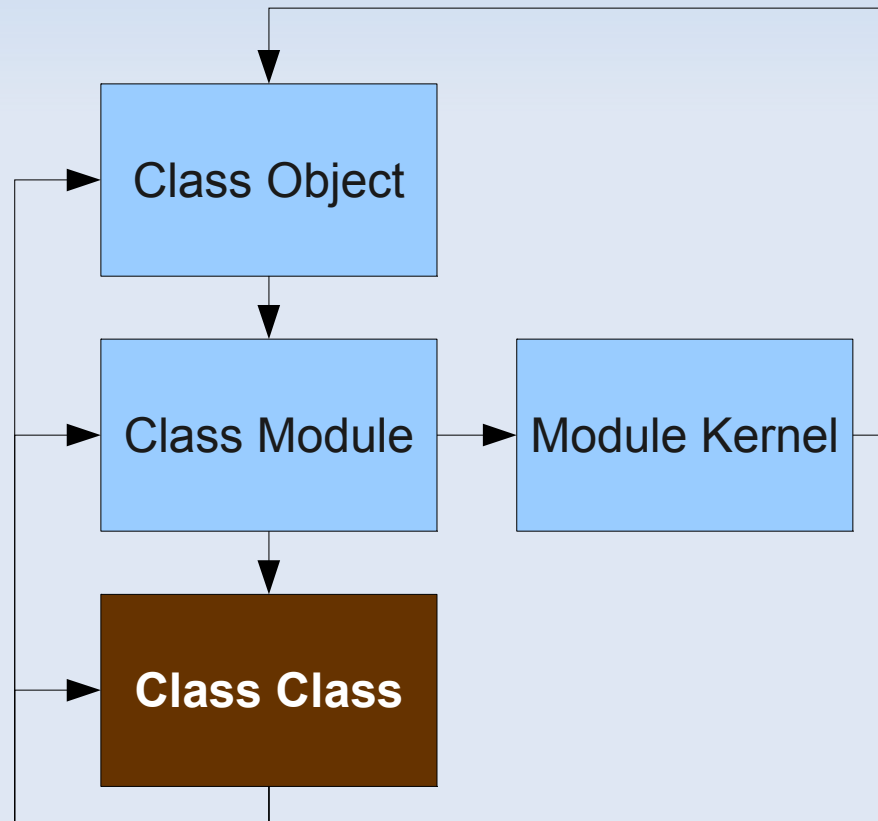
```
class Collector
  def self.inherited(subclass)
    @subclasses ||= []
    @subclasses << subclass
  end
end
```

```
class H < Collector; end
class I < Collector; end
```

```
> Collector.instance_variable_get(:@subclasses)
] [H, I]
```

Conclusion of Class Class

- Class Class is the most essential component to the Object-Oriented Ruby Object Model



Conclusion of Class Class

Essential Resources

- The Pickaxe
"Programming Ruby" by Dave Thomas
- Ruby RDoc

