

```
class JediSchool
  attr_reader :padawans, :parents, :grades

  #this 'mechanic' argument could be of any class
  def teach(jedi_master)
    jedi_master.teach_padawans(padawan)
  end

  ...
end
```



```
class JediMaster
  def teach_padawans(padawans)
    padawans.each { |padawan| teach_padawan(padawan) }
  end

  def teach_padawan(padawan)
    ...
  end
end
```

#Here you started with good intentions. Look, you don't
#care at all what kind of thing wants to teach your padawans,
#you just care that it can do teach_padawans

```
class JediSchool
  attr_reader :padawans, :parents, :grades

  #this 'mechanic' argument could be of any class
  def teach(trainers)
    trainers.each { |trainer|
      case trainers
      when Yoda
        trainer.meditate(padawans)
      when MaceWindu
        trainer.be_a_badass(padawans)
      when Anakin
        trainer.be_a_murder(padawans)
        trainer.join_sidious(sith_lord)
      end
    }
  end

  #...
end
```

```
class Yoda
  def meditate(padawans)
    ...
  end
end
```

```
class JediMaster
  def teach_padawan
    padawans.each {
    end

    def teach_padawan
      ...
    end
  end
end
```

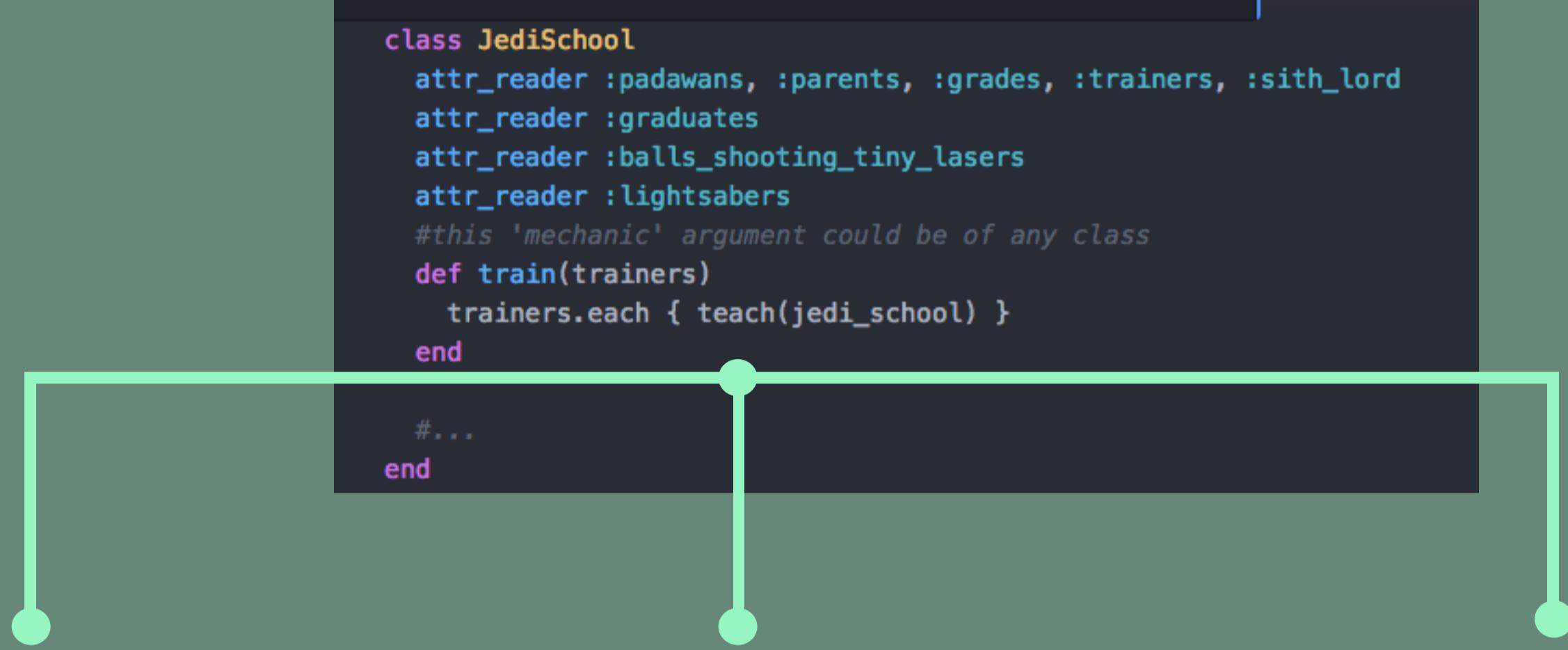
```
class MaceWindu
  def be_a_badass(padawans)
    ...
  end
end
```

```
class Anakin
  def be_a_murder(padawans)
    ...
  end

  def join_sidious(sith_lord)
    ...
  end
end
```

But then the plot thickens. Now there are multiple things
doing different methods, so you just decide to have a list
of conditionals accepting different types of objects, and
then acting based off the the class.

Sort of pretty, but not very reuseable. This means you have
to modify the train method and the JediSchool class
every time you want to add a method from a different
object. :-/



```
class Yoda
  def teach(jedi_school)
    padawans = jedi_school.padawans
    meditate(padawans)
  end

  def meditate(padawans)
    ...
  end
end
```

```
class MaceWindu
  def teach(jedi_school)
    padawans = jedi_school.padawans
    little_badasses = be_a_badass(padawans)
  end

  def be_a_badass(padawans)
    ...
  end

  private

  def force_lightning
    ...
  end

end
```

```
class Anakin
  def teach(jedi_school)
    sith_lord = jedi_school.sith_lord
    be_a_murderer(padawans)
    join_sidious(sith_lord)
  end

  def be_a_murder(padawans)
    ...
  end

  def join_sidious(sith_lord)
    ...
  end
end
```



Sandi Session

acquiring behavior thru inheritance





Modules

(that's next session)

acquiring behavior
thru inheritance **6**

```
# Here again you start with great intentions.

class StormTrooper
  attr_reader :height, :guns, :pistol_type

  def initialize(args)
    @height = args[:height]
    @guns = args[:guns]
    @pistol_type = args[:pistol_type]
  end

  def stuff
    { weapon_capacity: '2',
      grenade_capacity: '3',
      suit_color: 'white'
      pistol_type: pistol_type}
  end

  #...
end
```

```
trooper = StormTrooper.new(
  height: '6ft'
  guns: 'duh',
  pistol_type: 'blaster' )

trooper.height # => '6ft'
trooper.guns   # => 'duh'
```

...

Notes:
classical refers to an object-oriented class, not
meaning "archaic" ` ``

...

Sandi Quote:
"Inheritance is, at its core, a mechanism for automatic
messages delegation. It defines a forwarding path for
not-understood messages" ` ``

acquiring behavior
thru inheritance **6**

```

# Now we need a different kind of trooper
# Oh we'll just add arguments and if-else conditionals

class StormTrooper
  attr_reader :style, :height, :guns, :pistol_type

  def initialize(args)
    @style = args[:style]
    @height = args[:height]
    @guns = args[:guns]
    @sights = args[:sights]
    @shield = args[:shield]
    @pistol_type = args[:pistol_type]
  end

  def with_him
    if style == :elite_trooper
      { weapon_capacity: '2',
        grenade_capacity: '3',
        suit_color: 'white'
        guns: guns,
        sights: sights,
        shield: shield }
    else
      { weapon_capacity: '2',
        grenade_capacity: '3',
        suit_color: 'white'
        pistol_type: pistol_type
        guns: guns }
    end
  end

  #...
end

```

```

trooper = StormTrooper.new(
  style: :elite_trooper
  height: '6ft'
  guns: 'duh'
  sights: 'Sniper Sights woo!',
  shield: 'oh yeah we got those!'
  )

trooper.stuff
```
Notes:
classical refers to an object-oriented class, not
meaning "archaic"
```
Sandi Quote:
"Inheritance is, at its core, a mechanism for automatic
messages delegation. It defines a forwarding path for
not-understood messages"
```

```

```
Inheritance Strikes Back

class StormTrooper
 attr_reader :style, :height, :guns

 def initialize(args)
 @style = args[:style]
 @height = args[:height]
 @guns = args[:guns]
 @sights = args[:sights]
 @shield = args[:shield]
 @pistol_type = args[:pistol_type]
 end

 def with_him
 if style == :sniper
 { weapon_capacity: '2',
 grenade_capacity: '3',
 suit_color: 'white'
 guns: guns,
 sights: sights,
 shield: shield }
 else
 { weapon_capacity: '2',
 grenade_capacity: '3',
 suit_color: 'white',
 pistol_type: pistol_type
 guns: guns }
 end
 end

 ...
end
```

```
class Sniper < StormTrooper
 attr_reader :sights, :shield

 def initialize(args)
 @sights = args[:sights]
 @shield = args[:shield]
 super(args)
 end

 def with_him
 super.merge({
 sights: sights,
 shield: shield
 })
 end
end
```

```
trooper = Sniper.new(
 height: '6ft'
 sights: 'Sniper Sights woo!',
 shield: 'oh yeah we got those!'
)

trooper.stuff

...
```
Notes:  

The problem here is that Bicycle still describes a lot of specific information about the trooper you original designed for:  

the regular blaster pistol trooper.
```

```

acquiring behavior  
thru inheritance **6**

```

class StormTrooper
 attr_reader :height, :guns, :primary_gun_type

 def initialize(args={})
 @height = args[:size]
 @guns = args[:guns] || false
 @primary_gun_type = args[:primary_gun_type] || default_gun_type
 end

 def default_gun_type
 raise NotImplementedError,
 "This #{self.class} cannot respond to:"
 end
end

```

```

class BlasterTrooper < StormTrooper
 attr_reader :style, :height, :guns

 #...
 # def initialize(args)
 # @style = args[:style]
 # @height = args[:height]
 # @guns = args[:guns]
 # @sights = args[:sights]
 # @shield = args[:shield]
 # @pistol_type = args[:pistol_type]
 # end

 def default_gun_type
 'blaster'
 end

 #...
end

```

```

class Sniper < StormTrooper
 #...

 def default_gun_type
 'sniper rifle'
 end
end

blaster_trooper = BlasterTrooper.new(
 height: '6ft',
 pistol_type: 'blaster'
)

sniper_trooper = Sniper.new(
 height: '6ft',
 sights: 'Sniper Sights woo!',
 shield: 'oh yeah we got those!'
)

blaster_trooper.pistol_type # => 'blaster'
sniper_trooper.weapons_capacity # => '2'

```

acquiring behavior  
thru inheritance **6**

```

class StormTrooper
 attr_reader :height, :guns, :primary_gun_type

 def initialize(args={})
 @height = args[:size]
 @guns = args[:guns] || false
 @primary_gun_type = args[:primary_gun_type] || default_gun_type
 end

 def default_gun_type
 raise NotImplementedError,
 "This #{self.class} cannot respond to:"
 end
end

```

```

class BlasterTrooper < StormTrooper
 attr_reader :style, :height, :guns

 def initialize(args)
 @pistol_type = args[:pistol_type]
 super(args)
 end

 # ...
 def with_him
 super.merge({ pistol_type: pistol_flag })
 end

 def default_gun_type
 'blaster'
 end

 # ...
end

```

```

class Sniper < StormTrooper
 # ...
 def initialize(args)
 @sights = args[:sights]
 @shield = args[:shield]
 super(args)
 end

 def with_him
 super.merge({ sights: sights, shield: shield })
 end

 def default_gun_type
 'sniper rifle'
 end
end

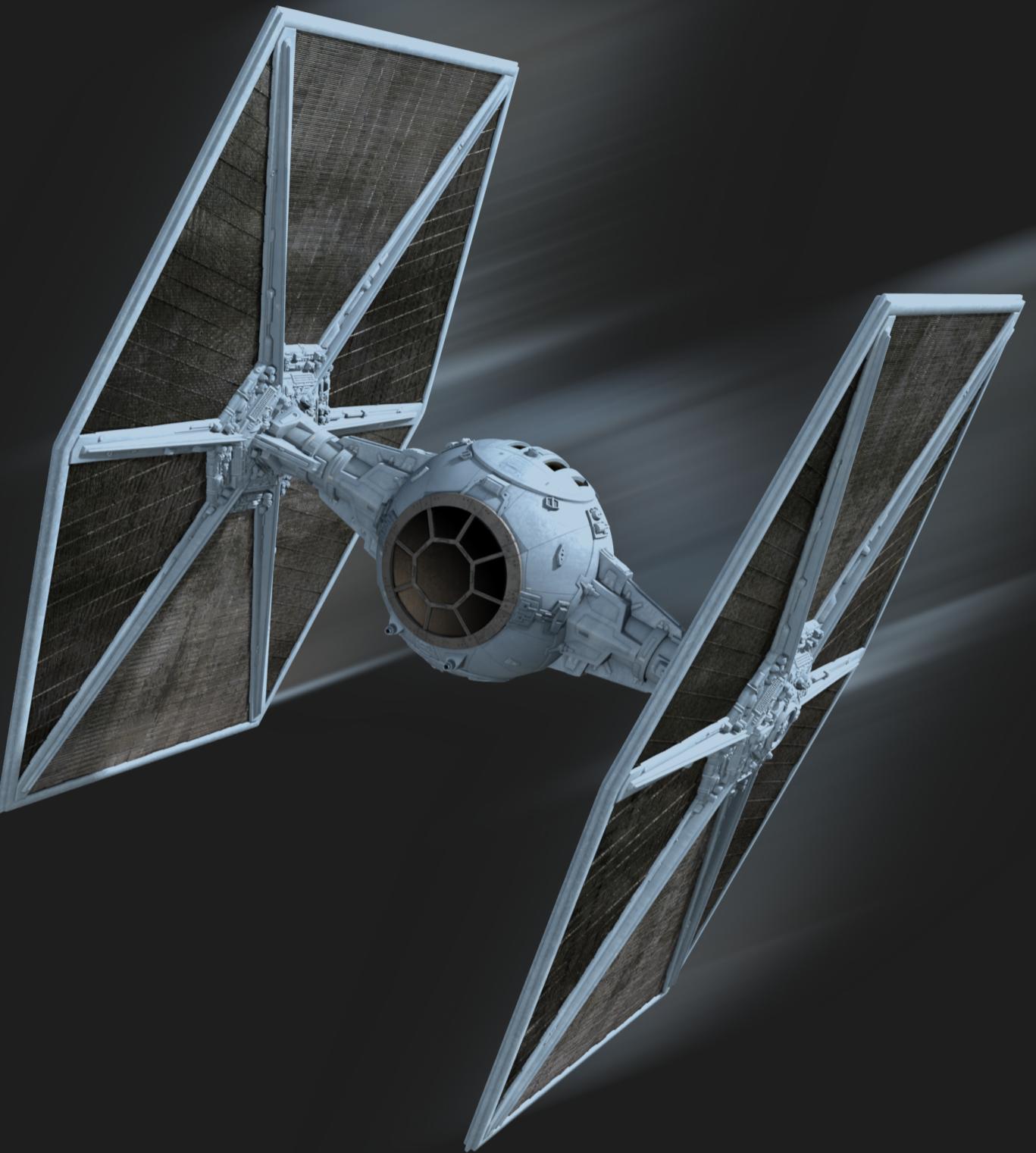
```

acquiring behavior  
thru inheritance **6**



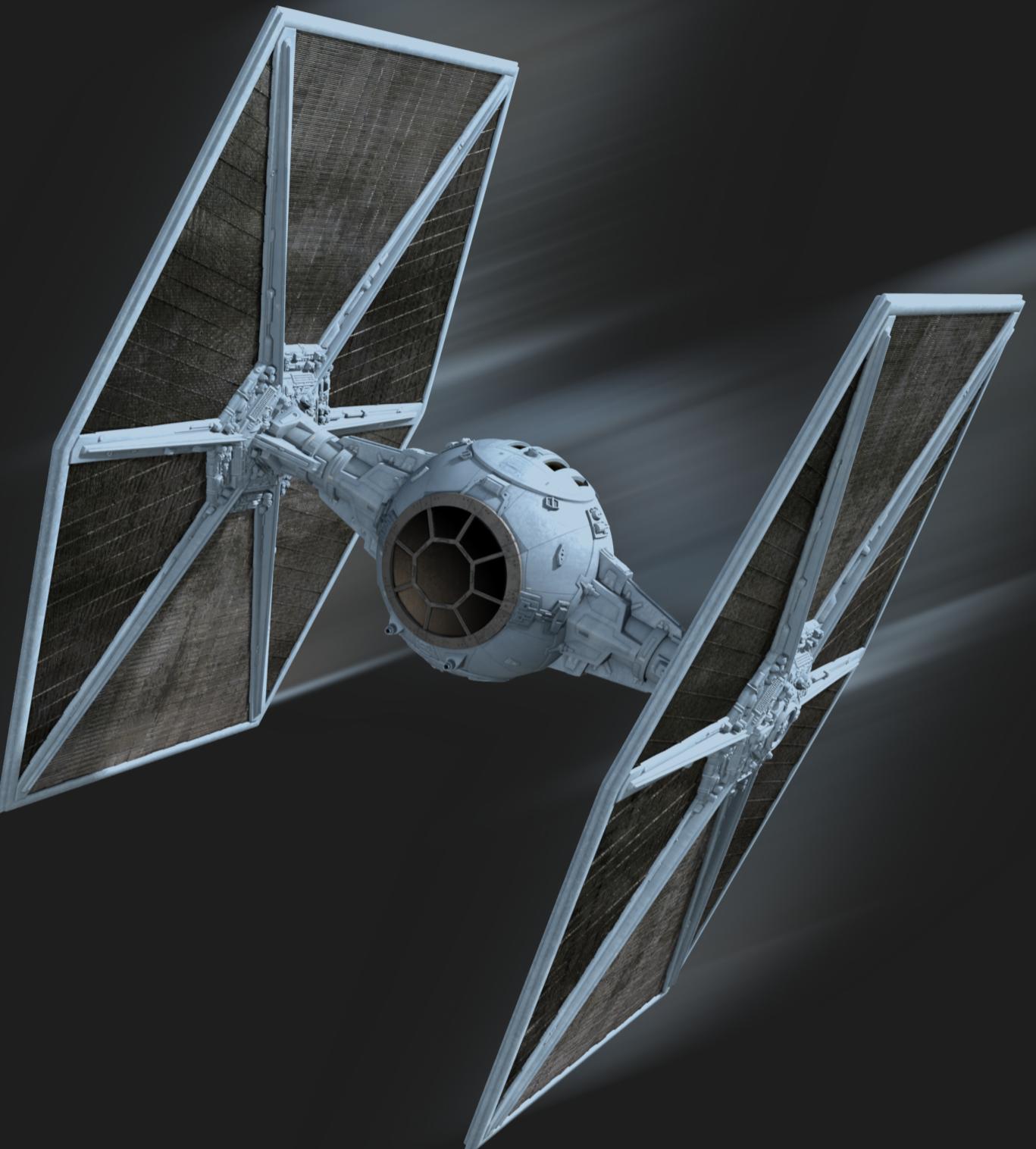
acquiring behavior  
thru inheritance

**6**



# Sandi Session

## sharing roles w/ modules



X

Clear

Start

Date

End

Date

S M T W T F S

February

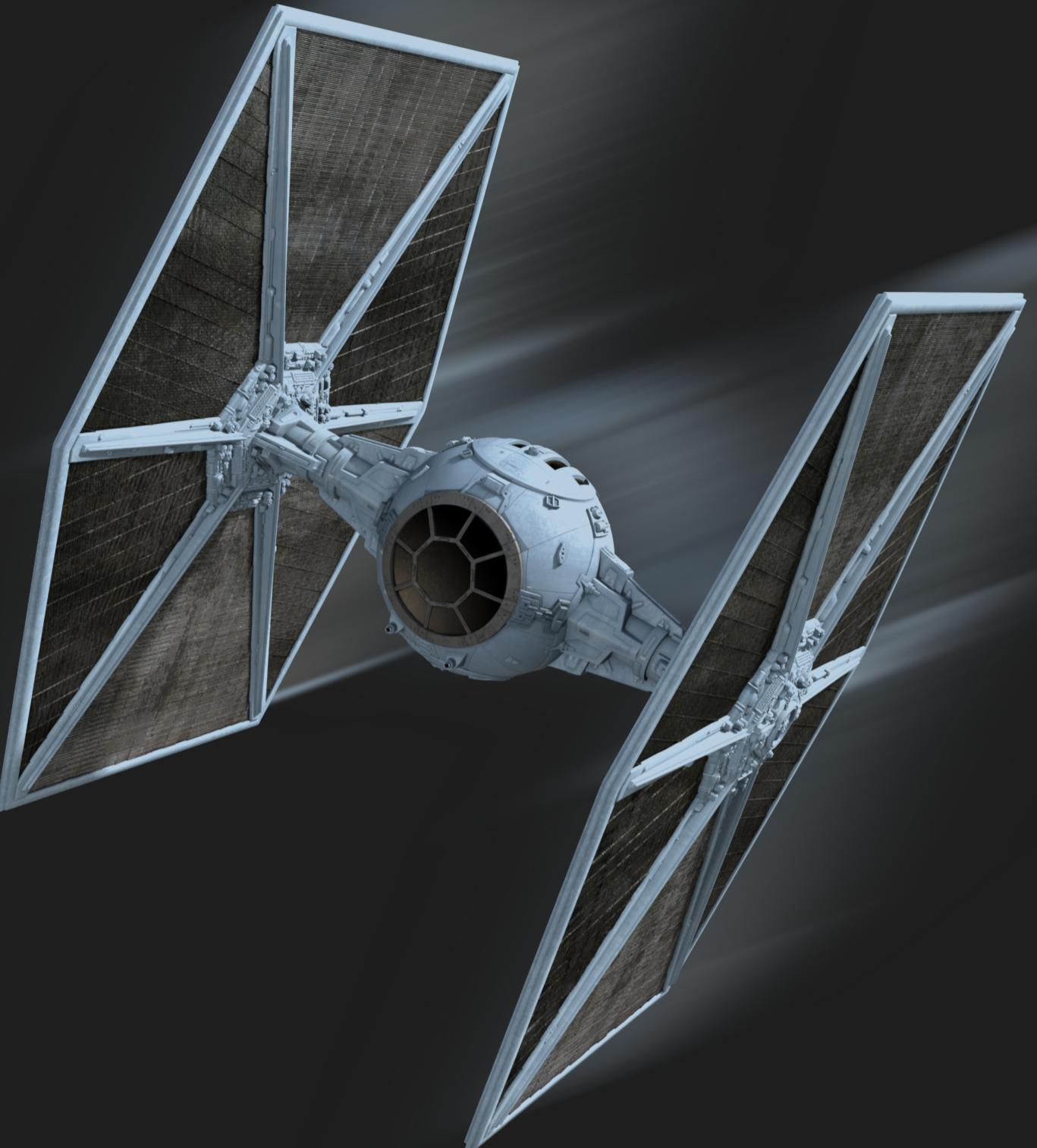
1 2 3 4

5 6 7 8 9 10 11

12 13 14 15 16 17 18

19 20 21 22 23 24 25

26 27 28



```
class Rental
 def rented?(rentable, start_date, end_date)
 puts "This #{claimable.class} " +
 "is not claimed\n" +
 " between #{start_date} and #{end_date}"
 false
 end
end

class TieFighter
 attr_reader :rent, :color, :hexagon_size

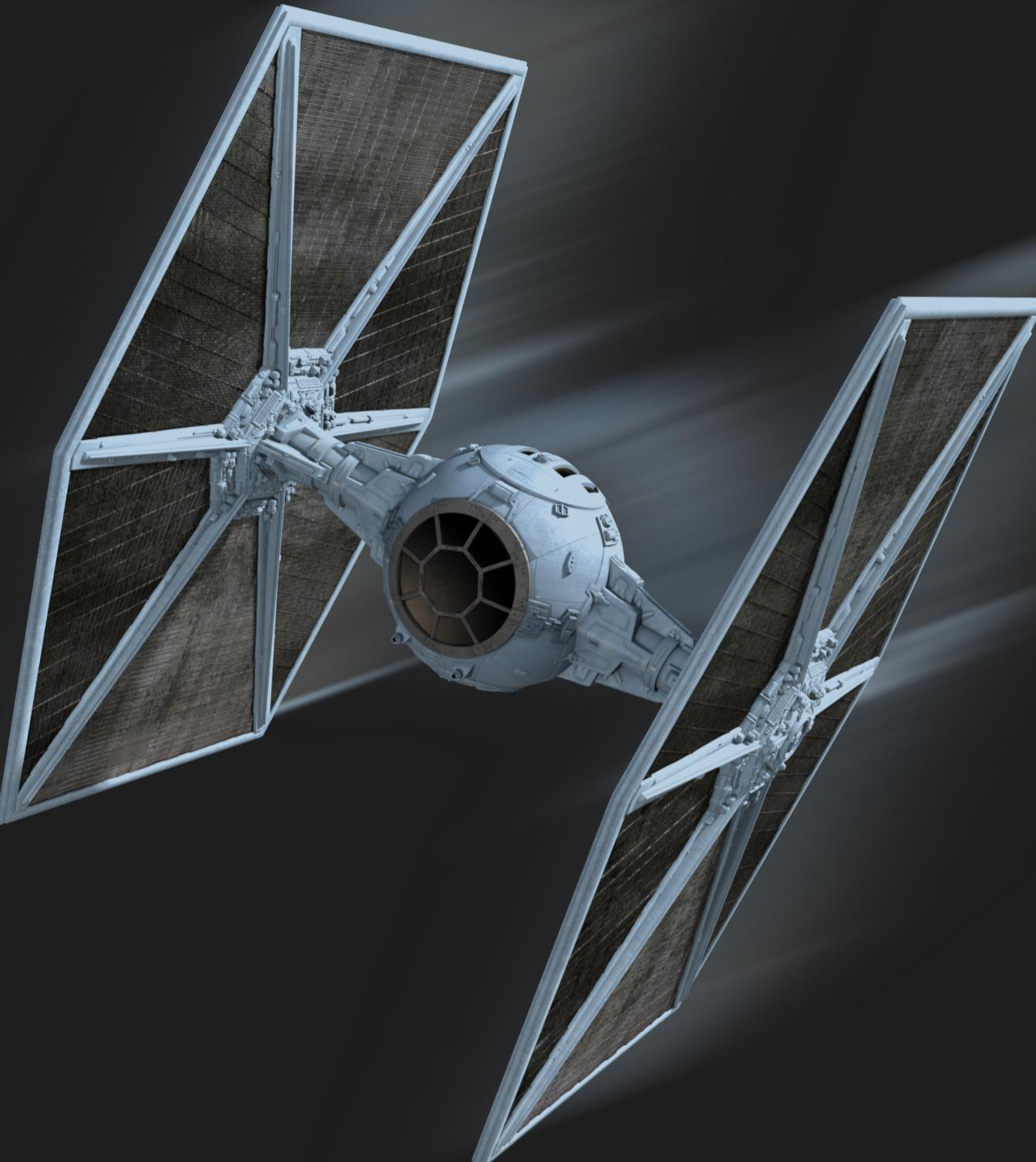
 # Give it it's claim from the get go
 def initialize(args={})
 @rent = args[:rental] || Rental.new
 # ...
 end

 def rentable?(start_date, end_date)
 !rented?(start_date - maintenance_days, end_date)
 end

 def rented?(start_date, end_date)
 rental.rented?(self, start_date, end_date)
 end

 def maintenance_days
 1
 end

 # ...
end
```



```
class TieFighter < Vehicle
 include Rentable

 def maintenance_days
 3
 end
end
```

```
class AtAtWalker < Vehicle
 include Rentable

 def maintenance_days
 10
 end
end
```

```
module Rentable
 attr_writer :rental

 def rental
 @rental ||= ::Rental.new
 end

 def rentable?(start_date, end_date)
 !rented?(start_date - maintenance_days)
 end

 def rented?(start_date, end_date)
 rental.rented?(self, start_date, end_date)
 end

 # default
 def maintenance_days
 0
 end
end
```

```
class MouseDroid
 include Rentable

 # ...
end
```