# Classes and Objects

Ruby Fundamentals

# Overview

- Create classes and instantiate objects
- Add instance variables and methods to your classes
- Control the visibility of these variables and methods
- Set initial state of the objects
- Create class variables and methods
- Leverage inheritance to re-use functionality between classes
- *self*, current context, executable class bodies and object equality

# Creating Classes, Instantiating Objects

```
class Spaceship
end
```

- Class names start with a capital letter and use CamelCase
- Capitalize abbreviations: XMLParser, JSONRequest

```
ship = Spaceship.new
```

# Objects vs. Variables

```
a = "abc"
```

```
b = a
```


```
def method(arg)
  arg
end

method(b)
```

```
b = a.clone
```


# Instance Variables and Methods

```
class Spaceship
  def launch(destination)
    # go towards destination
  end
end
```

# Instance Variables and Methods

```
class Spaceship
  def launch(destination)
    @destination = destination
    # go towards destination
  end
end
```

- *inspect* and *p* methods allow you to take a look inside objects
- Instance variables are private while methods are public by default

# Accessors

```
class Spaceship
  attr_accessor :destination
end

ship = Spaceship.new
ship.destination = "Earth"
puts ship.destination
```

# Accessors

```
class Spaceship
  attr_accessor :destination
  attr_reader :name
  attr_writer :name
end

ship = Spaceship.new
ship.name = "Dreadnought"
puts ship.name
```

# Accessors

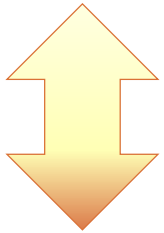
```
class Spaceship
  attr_accessor :destination, :name
end
```

# Accessors

```
class Spaceship
  attr_accessor :destination, :name

  def cancel_launch
    destination = ""       # creates local variable
    self.destination = ""
  end
end
```

# Virtual Attributes

```ruby
class Spaceship
  attr_accessor :destination
end
```



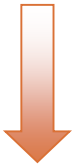
```ruby
class Spaceship
  def destination
    @destination
  end

  def destination=(new_destination)
    @destination = new_destination
  end
end
```

# Virtual Attributes

```
class Spaceship
  def destination
    @autopilot.destination
  end

  def destination=(new_destination)
    @autopilot.destination = new_destination
  end
end

ship = Spaceship.new
ship.destination = "Earth"
puts ship.destination # outputs Earth
```

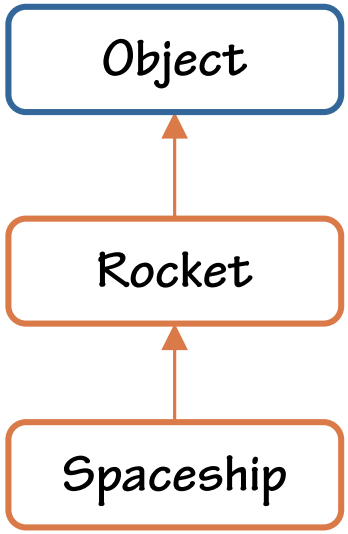
# Initialization

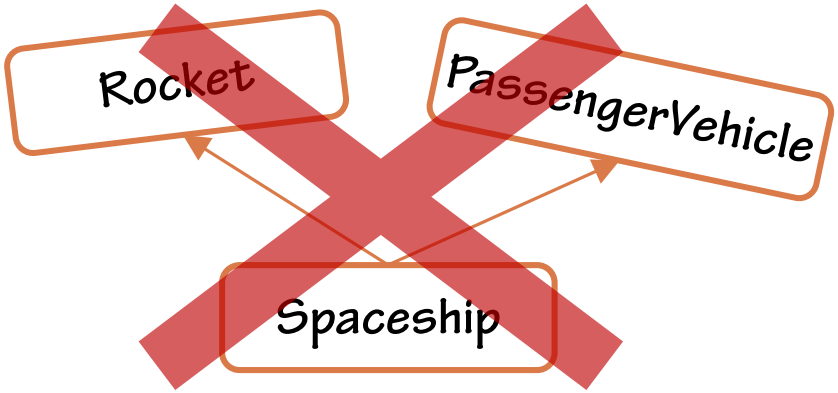
```
class Spaceship
  def initialize(name, cargo_module_count)
    @name = name
    @cargo_hold = CargoHold.new(cargo_module_count)
    @power_level = 100
  end
end

ship = Spaceship.new("Dreadnought", 4)
```

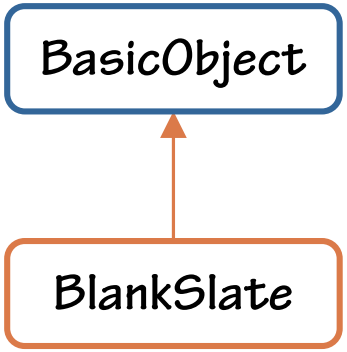


```
initialize("Dreadnought", 4)
```

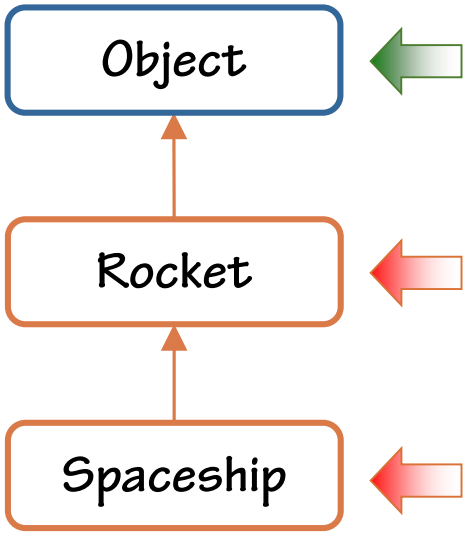
# Inheritance



# Inheritance




```
ship = Spaceship.new
ship.clone
```

# Inheritance

```ruby
class Probe
  def deploy
    # deploy the probe
  end
  def take_sample
    # do generic sampling
  end
end
```

```ruby
class MineralProbe < Probe
  def take_sample
    # take a mineral sample
  end
end

class AtmosphericProbe < Probe
  def take_sample
    # take a sample of the
    # atmosphere
  end
end
```

# Inheritance

- Inheritance is for reusing functionality, not enforcing interfaces

```
class Dockable
  def dock
    # implemented by subclasses
  end
end
```

```
class Probe  < Dockable
  def dock
    # probe specific
    # docking actions
  end
end
```

```
class Lander  < Dockable
  def dock
    # lander specific
    # docking actions
  end
end
```

# Inheritance

```
class Spaceship
  def capture(unit)
    unit.dock    # works on anything with dock method
    transport_to_storage(unit)
  end
end

ship.capture(probe)
ship.capture(lander)
```

# Class Methods and Class Variables

```
class Spaceship
  def self.thruster_count
    2
  end
end

Spaceship.thruster_count

ship = Spaceship.new
ship.thruster_count     # this doesn't work
```

# Class Variables

```
class Spaceship
  @@thruster_count = 2

  def self.thruster_count
    @@thruster_count
  end
end
```

# Class Instance Variables

```
class Spaceship
  @thruster_count = 2

  def self.thruster_count
    @thruster_count
  end
end
```

# Method Visibility

```
class Spaceship
  def launch
    batten_hatches
    # do other fun launch activities
  end

  def batten_hatches
    puts "Batten the hatches!"
  end
  private :batten_hatches
end
```

# Method Visibility

```
class Spaceship
  def launch
    batten_hatches
    light_seatbelt_sign
    # do other fun launch activities
  end

  private

  def batten_hatches
    puts "Batten the hatches!"
  end

  def light_seatbelt_sign
    puts "The seatbelt sign is now on."
  end
end
```

# Method Visibility

```
class Spaceship
  def launch
    batten_hatches
    light_seatbelt_sign
    # do other fun launch activities
  end

  def batten_hatches
    puts "Batten the hatches!"
  end

  def light_seatbelt_sign
    puts "The seatbelt sign is now on."
  end

  private :batten_hatches, :light_seatbelt_sign
end
```

# Method Visibility

```ruby
class Spaceship
  def self.disable_engine_containmemnt
    # dangerous – should be private!
  end

  # no error but does nothing
  private :disable_engine_containment

  # this is the correct way
  private_class_method :disable_engine_containment
end
```

# Method Visibility

- *public* is the default
- *private* means "can't be called with an explicit receiver"
- *private_class_method* is *private* for class methods
- *protected* means "allow access for other objects of the same class"
- *private* and *protected* not used a whole lot

# Executable Class Bodies

# *self*

```
class Spaceship

  def self.thruster_count
    2
  end

  def cancel_launch

    self.destination = ""

    seatbelt_sign(:off)

  end

end
```

*self == Spaceship*, hence we're adding this method to the class

*self == ship* inside method

```
ship.cancel_launch
```

No explicit object reference, so *seatbelt_sign* also called on *ship*

# Open Classes

```
class Spaceship
  def batten_hatches
    puts "Batten the hatches!"
  end
end

ship = Spaceship.new

class Spaceship
  def launch
    batten_hatches
    # do other fun launch activities
    puts "Launched!"
  end
end

ship.launch
```

# Monkey Patching

- **Adding or modifying behavior at runtime – particularly 3rd party code**

# Equality

# Summary

- Objects and classes
- Variables and methods in classes
- Inheritance and method visibility
- *self*, open classes, monkey patching and object equality