In this lecture, we will discuss…

- The advantages and disadvantages of Ruby being a dynamic language
- Dynamic Dispatch

Dynamic

- In static languages, like Java, the compiler requires you to define all the methods upfront
- In dynamic languages, such as Python and Ruby, methods don’t have to be predefined - they need to only be “found” when invoked
- Disadvantage?
  - Compiler can find bugs easier

Reporting System Example

- Say you have a `Store` class
  - Description and price of store products
- You are tasked with building a reporting system that can generate reports for different items in the store

Reporting System Example

```ruby
class Store
  def get_piano_desc
    "Excellent piano"
  end
  def get_piano_price
    120.00
  end
  def get_violin_desc
    "Fantastic violin"
  end
  def get_violin_price
    110.00
  end

  # ...many other similar methods...
end
```

Reporting System Example

```ruby
require_relative 'store'
class ReportingSystem
  def initialize
    @store = Store.new
  end
  def get_piano_desc
    @store.get_piano_desc
  end
  def get_piano_price
    @store.get_piano_price
  end

  # ...many more simimlar methods...
end

rs = ReportingSystem.new
puts "#{rs.get_piano_desc} costs #{rs.get_piano_price.to_s.ljust(6, '0')}"
# => Excellent piano costs 120.00
```

Calling Methods Dynamically

- So far, we have seen how to call methods using the dot notation `obj.method`
- It turns out, there is another way to call a method in Ruby - using the `send` method
- First parameter is the method name/symbol; the rest (if any) are method arguments
- Send?
  - Think of it as sending a message to an object

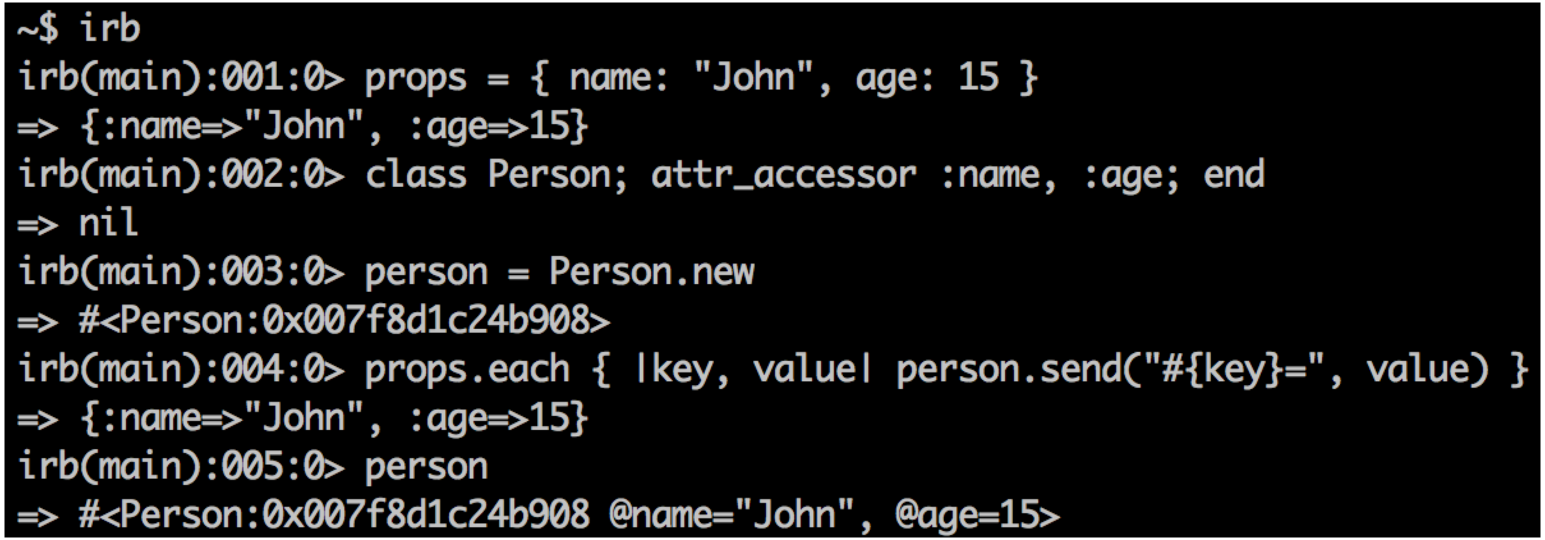
Calling Methods Dynamically

```ruby
class Dog
  def bark
    puts "Woof, woof!"
  end
  def greet(greeting)
    puts greeting
  end
end

dog = Dog.new
dog.bark # => Woof, woof!
dog.send("bark")  # => Woof, woof!
dog.send(:bark)  # => Woof, woof!
method_name = :bark
dog.send method_name # => Woof, woof!

dog.send(:greet, "hello")  # => hello
```

Dynamic Dispatch: Advantages

- ✧ Advantages to dynamic method calling, a.k.a. "*Dynamic Dispatch*"
  - Can decide at runtime which methods to call
- ✧ The code doesn't have to find out until runtime which method it needs to call

Dynamic Dispatch Example



```
~$ irb
irb(main):001:0> props = { name: "John", age: 15 }
=> {:name=>"John", :age=>15}
irb(main):002:0> class Person; attr_accessor :name, :age; end
=> nil
irb(main):003:0> person = Person.new
=> #<Person:0x007f8d1c24b908>
irb(main):004:0> props.each { |key, value| person.send("#{key}=", value) }
=> {:name=>"John", :age=>15}
irb(main):005:0> person
=> #<Person:0x007f8d1c24b908 @name="John", @age=15>
```

Summary

- ✧ Don’t need to call a method using the dot notation
- ✧ Can call methods dynamically using a string or symbol

What’s Next?

- ✧ Dynamic Methods