CLab 3
coen164
2020/04/20

yuan wang

1. make up example to check if class method is inherited

2. .Module1 has methods: method11, method12, method13
    Module2 has methods: method21, method22, method23

define these 2 modules, and define a class, so that object of this class can call these methods.

(check out the slide about module as mixin)

3. following are two class definitions (Push class) in two libraries. each of them is shown in
   # section.

```
#gym.rb
class Push
  def up
    40
  end
end


#dojo.rb
class Push
  def up
    30
  end
end


require "./gym"

gym_push = Push.new
p gym_push.up          #40

require "./dojo"

dojo_push = Push.new
p dojo_push.up         #30
```

As the dojo library is loaded after gym, it has overridden gym's class definition of Push and therefore creates an instance of Push defined in dojo.

use modules to solve this problem.

4. When you subclass, the instance method will be inherited, however, the initialize method in the parent class, will not be called automatically, you need to call it explicitly

check out the following code:

```
class Parent
  def initialize(name="nobody")
    @name = name
  end
end

class Child < Parent
  attr_accessor :name, :grade

  def initialize(name, grade)
    @grade = grade
  end
end


y = Child.new("yuan", 100)

print "name is: ", y.name
puts
puts y.grade
y.grade = 90

puts y.grade
```

add a statement to make the initialize in the parent get called. and check out the result again.

5. constant lookup

```
module Dojo
  A = 4
  module Kata
          B = 8
    module Roulette
```

```
      class ScopeIn
        def push
          15
        end
      end
    end
  end
end

A = 16
B = 23
C = 42
```

Print all constants. and create object to call “push()” method

6. The following is a module definition

```
module Greetings
  def english
    puts "Hello!"
  end

  def french
    puts "Bonjour!"
  end

  def spanish
    puts "Hola!"
  end
end
```

a. define a class ‘Hello”. the object of this class should be able to call #english, #french, #spanish, to output different languages hello

for example, if hello is an object of Hello, then

```
hello.spanish
=> Hola!
```

b. modify the class ‘Hello’ so that you can these methods directly, for example:

```
Hello.spanish
=> Hola
```

7. Module can be used as name spaces.

take a look at the following example:

```
#rendering.rb
module Rendering
  class Font
    attr_accessor :name, :weight, :size
    def initialize(name, weight=:normal, size=10)
      @name = name
      @weight = weight
      @size = size
    end
  end

  class PaperSize
    attr_accessor :name, :width, :height
    def initialize(name="Us Letter", width=8.5, height=11.0)
      @name = name
      @width = width
      @height=height
    end
  end
end
```

this Rendering module collect some classes and make them in one name space.

save this file in to one rendering.rb file
and include it in a new file.
then add some definition to this module (do not change the rendering.rb code, rather, add definition in your new file)

a. add constants DEFAULT_FONT and DEFAULT_PAPER_SIZE in the rendering module, the value for these two constants are a font object and a PaperSize object

b. add a instance method #check_default() in the two classes, the two methods will be printing default font and default paper_size

c. add a instance method #check_self() in the two classes, the two methods will be printing the object itself.

d. write code to print the result.

e. separate the module definition into two separate files while keep the definition unchanged.

f. assign the module into a variable, use the variable as the module.

8. There are several builtin MIXINs defined by ruby. One of them is Enumerable. If you include this MIXIN, you are able to do traversing, searching and sorting of collections. But you need to define “each” method which yields successive members of the collection, then you can use method like “map”, “inject”

define a class “VowelFinder”,
this class will define object with a “string” attribute to store a string, then a “each” method to pass each vowel in the string to the block.
(hint, use “scan” method and regular expression to find vowel in string)

then create an object from your class, then try map and inject
(for example, you can call object.inject(:+) to concatenate all vowels in the object)