CS 683 Emerging Technologies
Fall Semester, 2005
Doc 18 Ruby Classes, Containers & Methods
Contents

References

Programming Ruby, The Pragmatic Programmers' Guide, Dave Thomas, ISBN 0-9745140-5-5

Classes
Definition

```
class NoState
   def hello
      return 'hello'
   end

  def increase(x)
    x + 1
  end
end
```

```
require 'test/unit'

class TestExample < Test::Unit::TestCase
  def test_hello
    object = NoState.new
    assert_equal('hello', object.hello )
    assert( 'hello' == object.hello() )
    assert( 2 == object.increase(1)  )
  end
end
```

Instance Variables

```
class Book
   def initialize(title, author, date)
      @title = title
      @author = author
      @date = date
   end

  def to_s
    "Book: #@title by #{@author} on #{@date}"
  end
end

require 'test/unit'

class TestExample < Test::Unit::TestCase
  def test_new
    cat = Book.new("Cat", "Me", "1990")
    assert_instance_of(Book, cat)
    assert( "Book: Cat by Me on 1990" == cat.to_s)
    assert_match(/^Book:(\s)*Cat(.)*Me(.)*1990/, cat.to_s)
  end
end
```

Instance Variable Access

```
class Book
   def initialize(title, author, date)
      @title = title
      @author = author
      @date = date
   end

  def author
    @author
  end

  def author=(new_author)
    @author = new_author
  end
end

require 'test/unit'

class TestExample < Test::Unit::TestCase
  def test_author
    cat = Book.new("Cat", "Me", "1990")
    assert_not_nil( cat)
    assert( 'Me' == cat.author)
    assert( 'Me' == cat.author() )
    cat.author = 'You'
    assert_equal( 'You',  cat.author )
  end
end
```

Attributes

```
class Book
  attr_reader :title, :author, :date
  attr_writer :title
    def initialize(title, author, date)
        @title = title
        @author = author
        @date = date
    end
end
```

```
require 'test/unit'
require 'book'

class TestExample < Test::Unit::TestCase
  def test_author
    cat = Book.new("Cat", "Me", "1990")
    assert( 'Me' == cat.author)
    assert( 'Cat' == cat.title)
    cat.title = 'Dog'
    assert_equal( 'Dog', cat.title)
    assert_raise(NoMethodError) {cat.author = 'Me'}
  end
end
```

Class Methods and Variables

```
class Book
  @@count = 0

  def Book.count
    @@count
  end

    def initialize(title, author, date)
        @title = title
        @author = author
        @date = date
        @@count += 1
    end
end

require 'test/unit'
require 'book'

class TestExample < Test::Unit::TestCase
  def test_count

    assert( Book.count == 0, 'First count')
    cat = Book.new("Cat", "Me", "1990")
    assert( Book.count == 1, 'Second count')
    assert_raise(NoMethodError) {cat.count}
  end
end
```

Inheritance

```
class Book
   def initialize(title, author, date)
      @title = title
      @author = author
      @date = date
   end

  def to_s
    "Book: #@title by #{@author} on #{@date}"
  end
end

class ElectronicBook < Book
  def initialize(title, author, date, format)
    super(title, author, date)
    @format = format
  end
  def to_s
    super + " in #{@format}"
  end
end
```

Inheritance tests

```
require 'test/unit'
require 'book'

class TestExample < Test::Unit::TestCase
  def test_count

    cat = ElectronicBook.new("Cat", "Me", "1990", 'pdf')
    assert_kind_of(Book, cat )
    assert_kind_of(ElectronicBook, cat)

    assert( cat.class == ElectronicBook)
    assert_match( mary thom/(.)+1990(.)+pdf$/,  cat.to_s)

  end
end
```

Access Control

- Public methods
  Accessible by anyone
- Protected methods
  Accessible by defining class and subclasses
- Private Methods
  Accessible by defining class

```
class Foo
      def method1     #default is public
      end

   protected
      methods defined here are protected
      def bar
      end

   private
      define a bunch of private methods here
   public
      You can have different groups of same access
end
```

Alternative Access Declaration

```
class Foo
   def method1
   end

   def method2
   end
   ....
   public :method1, method3
   protected   :method2
   private     :method4, method5
end
```

Singleton

```
class Singleton
  private_class_method :new
  @@instance = new
  def Singleton.instance
    @@instance
  end
end
```

```
require 'test/unit'
require 'Singleton'

class TestSingleton < Test::Unit::TestCase
  def test_unique
    assert_raise(NoMethodError) {Singleton.new}
    assert_kind_of(Singleton, Singleton.instance)

    assert_same( Singleton.instance, Singleton.instance)
    assert( Singleton.instance.equal?( Singleton.instance))
    assert( Singleton.instance.object_id ==
              Singleton.instance.object_id)

  end
end
```

Operators

```
class Book
  attr_reader :title
    def initialize(title, author, date)
        @title = title
        @author = author
        @date = date
    end

  def <(book)
    return false if book.kind_of(Book)
    title < book.title
  end
end

require 'test/unit'
require 'book'

class TestExample < Test::Unit::TestCase
  def test_operator
    cat = Book.new("Cat", "Me", "1990")
    dog = Book.new("Dog", "Me", "1990")
    assert( cat < dog)
  end
end
```

Adding Methods to Existing Classes

```
class Fixnum
  def foo
    self + 1
  end
end

require 'test/unit'

class TestAddingMethods < Test::Unit::TestCase
  def test_integer_foo

    assert( 1.foo == 2 )
    assert( -45.foo == -44)

  end
end
```

Adding Methods to an Object

```
class Book
  attr_reader :title
    def initialize(title, author, date)
        @title = title
        @author = author
        @date = date
    end
 end



require 'test/unit'
require 'book'
class TestExample < Test::Unit::TestCase
  def test_singelton_method
    cat = Book.new("Cat", "Me", "1990")
    dog = Book.new("Dog", "Me", "1990")

    def cat.foo
      5
    end

    assert( cat.foo ==5 )
    assert_raise(NoMethodError) {dog.foo}
  end
end
```

Containers, Blocks & Iterators
Arrays

```
a = [1, 2, 3]
b = 1, 2, 3
c = Array.new
d = Array.new(5)
e = Array.new(5) { |k| k*k }
f = Array.new(5) { |k| k*2 + 1 }
g = Array,new( 5, 'cat')
```

d	[nil, nil, nil, nil,nil]
e	[0, 1, 4, 9, 16]
f	[1, 3, 5, 7, 9]
g	['cat', 'cat', 'cat', 'cat', 'cat']

f[0]	1
f[-1]	9
f[-2]	7
f[2, 3]	[5, 7, 9]
f[3, 1]	[7]
f[2..3]	[5, 7]
f[2...3]	[7]
f[2.9]	5

Array Methods

Array is a class

```
stack = Array.new
stack.push(5)
stack.push(4)
stack.push(3)
sorted = stack.sort
```

Listing methods

```
puts Array.public_instance_methods
```

Adding methods

```
class Array
  def foo
    'bar'
  end
end

puts [2, 1].foo
```

Iterators

Code	Output
`[1, 2, 3].each {|x| puts x}`	1 2 3

`x = 4`
`4.times do`
`  puts x`
`  x *= 2`
`end`
	4
	8
	16
	32

`2.upto(5) {|x| puts x}`	2 3 4 5
`3.step(10, 3) {|x| puts x}`	3 6 9
`('g'..'j').each {|char| puts char}`	h i j

Iterators

Code	Output
a = [1, 2, 3].collect {|x| x + 3} puts a	4 5 6
a = [1, 2.345, 3].select {|x| x.integer?} puts a	1 3
a = [1, 2.345, 3].find_all {|x| x.integer?} puts a	1 3

Blocks are Closures

```
x = 3
def test
  x = 1
  puts x
end

test
puts x
```

Output

```
1
3
```

```
x = 3
def test
  x = 1
  yield
end

test {puts x}
puts x
```

Output

```
3
3
```

Proc

```
class ProcSample
  def set_action(&action)
    @action = action
  end

  def do_it
    @action.call(self)
  end
end

require 'test/unit'

class TestExample < Test::Unit::TestCase
  def test_do_it
    a = ProcSample.new

    a.set_action {1}
    assert_equal( a.do_it,  1)
    a.set_action {'cat'}
    assert_equal(a.do_it, 'cat')

    x = lambda {'dog'}
    a.set_action(&x)
    assert_equal(a.do_it, 'dog')
  end
end
```

More Proc

```
x = lambda {puts 1}
3.times(&x)
3.times {puts 2}
```

More about Methods

Default Parameter Values

```
def default_values(a, b='cat', c='dog')
  "#{a}, #{b}, #{c}"
end

puts default_values(1)
```

Variable-Length Argument Lists

```
def variable_number(a, *b)
  (0..b.length - 1).each {|k| puts b[k]}
end

variable_number(1,2, 3,4, 5)
```

Expanding Arrays

```
def four(a, b, c, d)
  "#{a}, #{b}, #{c}, #{d}"
end

x = [1, 2, 3]
puts four(0, *x)
```

Expressions

Assignment

```
a = b = 1 + 2
a = (b = 1 + 2) + 3
a, b = b, a

x = 0
a, b, c = x, (x += 1), (x += 1)

a, b = [1, 2, 3, 4]
#a is 1
#b is 2

a, *b = [1, 2, 3, 4]
puts b
```

if

```
x = gets.to_i

if x > 12
  y = 6
else
  y = 1
end
puts y

y = if x > 12
      6
    else
      1
    end
puts y
```

More if

```
if x > 12 then
  y = 6
else
  y = 1
end

if x > 12 then y = 6
else y = 1
end

if x > 12:
  y = 6
else
  y = 1
end

if x > 12: y = 6
else y = 1
end
```

unless

negation of if

```
unless x <= 12
  y = 1
else
  y = 6
end
```

for in

syntax sugar for each

```
for x in (3..10)
  puts x
end
```

```
(3..10).each {|x| puts x}
```