CMSC 330: Organization of Programming Languages

Ruby is OO:
Methods, Classes
In Ruby, everything is an Object

- Ruby is object-oriented
- All values are (references to) objects
  - Java/C/C++ distinguish primitives from objects
- Objects communicate via method calls
- Each object has its own (private) state
- Every object is an instance of a class
  - An object’s class determines its behavior:
  - The class contains method and field definitions
    - Both instance fields and per-class ("static") fields
Everything is an Object

> 1.class
Integer

> 1.methods
[:to_s, :to_i, :abs, ...]

Object is the superclass of every class

> 1.class.ancestors
[Integern Numeric, Comparable, Object, Kernel, BasicObject]
Objects Communicate via Method Calls

+ is a method of the Integer class

\[
1 + 2 \quad \Rightarrow \quad 2 \\
1.+(2) \quad \Rightarrow \quad 3
\]

1 + 2 is syntactic sugar for 1.+(2)

\[
1.add(2) \quad \Rightarrow \quad 1.+(2) \Rightarrow \quad 1 + 2
\]

1.to_s = "1"

1.to_s() = "1"        no parens needed if no args
The nil Object

- Ruby uses nil (not null)
  - All uninitialized fields set to nil
  - `irb(main):004:0> @x`  
    `=> nil`

- `nil` is an object of class `NilClass`
  - Unlike null in Java, which is a non-object
  - nil is a singleton object – there is only one instance of it
    - `NilClass` does not have a `new` method
  - `nil` has methods like `to_s`, but not other methods
    - `irb(main):006:0> nil + 2`  
      `NoMethodError: undefined method `+` for nil:NilClass`
Classes are Objects too

> nil.class
NilClass

> 2.5.class
Float

> true.class
TrueClass

> Float.class
Class
First-class Classes

- Since classes are objects, you can manipulate them however you like
  - Here, the type of y depends on p
    - Either a String or a Time object

```plaintext
if p then
  x = String
else
  x = Time
End
y = x.new
```
Quiz 1

What is the type of variable \( x \) at the end of the following program?

\[
p = \text{nil} \\
x = 3 \\
\text{if } p \text{ then} \\
x = \text{"hello"} \\
\text{else} \\
x = \text{nil} \\
\text{end}
\]

A. Integer
B. NilClass
C. String
D. Nothing – there’s a type error
What is the type of variable \( x \) at the end of the following program?

\[
p = \text{nil} \\
x = 3 \\
if p \text{ then} \\
\quad x = \text{“hello”} \\
else \\
\quad x = \text{nil} \\
end
\]

A. Integer
B. NilClass
C. String
D. Nothing – there’s a type error
Standard Library: String class

- Strings in Ruby have class `String`
  - "hello".class == String

- The String class has many useful methods
  - s.length # length of string
  - s1 == s2 # structural equality (string contents)
  - s = "A line\n"; s.chomp # returns "A line"
    - Return new string with s's contents minus any trailing newline
  - s = "A line\n"; s.chomp!
    - Destructively removes newline from s
    - Convention: methods ending in ! modify the object
    - Another convention: methods ending in ? observe the object
Creating Strings in Ruby

- Substitution in double-quoted strings with `#{ }`
  - `course = "330"; msg = "Welcome to #{course}"`
  - "It is now #{Time.new}"`
  - The contents of `#{ }` may be an arbitrary expression
  - Can also use single-quote as delimiter
    - No expression substitution, fewer escaping characters

- Here-documents
  
s = <<END
    This is a text message on multiple lines
    and typing \n is annoying
  END

Creating Strings in Ruby (cont.)

- `sprintf`
  
  ```ruby
  count = 100
  s = sprintf("%d: %s", count, Time.now)
  => "100: 2021-01-27 19:56:06 -0500"
  ```

- `to_s` returns a **String** representation of an object
  - Like Java's `toString()`

- `inspect` converts any object to a string
  ```ruby
  irb(main):033:0> p.inspect
  => "#<Point:0x54574 @y=4, @x=7>"
  ```
Symbols

- Ruby *symbols* begin with a colon
  - :foo, :baz_42, :"Any string at all"
- Symbols are “interned” Strings,
- Symbols are more efficient than strings.
  - The same symbol is at the same physical address

```
"foo" == "foo"    # true
"foo".equal? "foo"  # false
:foo == :foo        # true
:foo.equal :foo     # true
```
Arrays and Hashes

- Ruby data structures are typically constructed from Arrays and Hashes
  - Built-in syntax for both
  - Each has a rich set of standard library methods
  - They are integrated/used by methods of other classes
Array

- Create an empty Array
  
  \[ t = \text{Array.new} \]
  
  \[ x = [ ] \]
  
  \[ b = \text{Array.new}(3) \]  # b = [nil, nil, nil]
  
  \[ b = \text{Array.new}(5, "a") \]  # b = ["a", "a", "a", "a", "a"]

- Arrays may be heterogeneous
  
  \[ a = [1, "foo", 2.14] \]
Array Index

> s = ["a","b","c", 1, 1.5, true]

<table>
<thead>
<tr>
<th></th>
<th>“a”</th>
<th>“b”</th>
<th>“c”</th>
<th>1</th>
<th>1.5</th>
<th>true</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>-6</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>

> s[0]
"a"

> s[-6]
"a"
Arrays Grow and Shrink

Arrays are **growable**

```ruby
b = []; b[0] = 0; b[5] = 0; b
=> [0, nil, nil, nil, nil, 0]
```

Arrays can also **shrink**

- Contents shift left when you delete elements
  ```ruby
  a = [1, 2, 3, 4, 5]
  a.delete_at(3)  # delete at position 3; a = [1, 2, 3, 5]
  a.delete(2)    # delete element = 2; a = [1, 3, 5]
  ```
Two-Dimensional Array

```ruby
> a = Array.new(3) { Array.new(3) } 
> a[1][1]=100

> a

```
```ruby
[
  [nil, nil, nil],
  [nil, 100, nil],
  [nil, nil, nil]
]
```
Some Array Operations

\[
\begin{align*}
a &= [1, 2, 3, 4] \\
b &= [3, 4, 5, 6]
\end{align*}
\]

Adding two arrays

\[
a + b \Rightarrow [1, 2, 3, 4, 3, 4, 5, 6]
\]

Union

\[
a \mid b \Rightarrow [1, 2, 3, 4, 5, 6]
\]

Intersection

\[
a \& b \Rightarrow [3, 4]
\]

Subtract

\[
a - b \Rightarrow [1, 2]
\]
Arrays as Stacks and Queues

- Arrays can model stacks and queues

```javascript
let a = [1, 2, 3] // Stack
a.push("a") # a = [1, 2, 3, "a"]
x = a.pop # x = "a"

// Queue: push and shift
// Deque: push, pop, shift, unshift
a.unshift("b") # a = ["b", 1, 2, 3]
y = a.shift # y = "b"
```

Note that `push`, `pop`, `shift`, and `unshift` all permanently modify the array.
Quiz 2: What is the output?

```python
a = [1,2,3]
a[1] = 0
a.shift
print a[1]
```

A. Error
B. 2
C. 3
D. 0
Quiz 2: What is the output?

```plaintext
a = [1,2,3]
a[1] = 0
a.shift
print a[1]
```

A. Error  
B. 2  
C. 3  
D. 0
A hash acts like an array, whose elements can be indexed by any kind of value

- Every Ruby object can be used as a hash key, because the Object class has a hash method

Elements are referred to like array elements

```ruby
italy = Hash.new       # or italy={}
italy["population"] = 58103033
italy[1861] = "independence"

p = italy["population"]   # pop is 58103033
planet = italy["planet"] # planet is nil
```
Hash methods

- **new(v)** returns hash whose default value is `v`
  - `h = Hash.new("fish")`;
  - `h["go"]` # returns "fish"

- **values**: returns array of a hash’s values
- **keys**: returns an array of a hash’s keys
- **delete(k)**: deletes mapping with key `k`
- **has_key?(k)**: is true if mapping with key `k` present
  - **has_value?(v)** is similar
Hash creation

Convenient syntax for creating literal hashes

- Use `{ key => value, ... }` to create hash table

```ruby
credits = {
  "cmsc131" => 4,
  "cmsc330" => 3,
}

x = credits["cmsc330"]  # x now 3
credits["cmsc311"] = 3
```

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmsc131</td>
<td>4</td>
</tr>
<tr>
<td>cmsc330</td>
<td>3</td>
</tr>
</tbody>
</table>
Hashes of Hashes

\[
\begin{align*}
    h &= \text{Hash.new}(\emptyset) \\
h[1] &= \text{Hash.new}(\emptyset) \\
h[1][2] &= 5 \\
h[2] &= \text{Hash.new}(\emptyset) \\
h[2][1] &= 1 \\
h[3] &= \text{Hash.new}(\emptyset) \\
h[3][3] &= 3
\end{align*}
\]

\[
\begin{align*}
h \text{ is} & \quad \{ \\
    1 \rightarrow \{2 \rightarrow 5\}, \\
    2 \rightarrow \{1 \rightarrow 1\}, \\
    3 \rightarrow \{3 \rightarrow 3\} \\
\} \\
\end{align*}
\]

\[
\begin{pmatrix}
0 & 5 & 0 \\
1 & 0 & 0 \\
0 & 0 & 3
\end{pmatrix}
\]
Quiz 3: What is the output?

```python
a = {"foo" => "bar"}
a["bar"] = "baz"
print a[1]
print a["foo"]
```

A. Error
B. bar
C. bazbar
D. baznilbar
Quiz 3: What is the output?

```python
a = {“foo” => “bar”}
a[“bar”] = “baz”
print a[1]
print a[“foo”]
```

A. Error
B. bar
C. bazbar
D. baznilbar
 Quiz 4: What is the output?

```

a = { "Yellow" => [] }

a["Yellow"] = {}

a["Yellow"]["Red"] = ["Green", "Blue"]

print a["Yellow"]["Red"][1]
```

A. Green  
B. (nothing)  
C. Blue  
D. Error
Quiz 4: What is the output?

```python
a = { "Yellow" => [] }
a["Yellow"] = {}
a["Yellow"]["Red"] = ["Green", "Blue"]
print a["Yellow"]["Red"][1]
```

A. Green
B. (nothing)
C. Blue
D. Error
Methods in Ruby

Methods are declared with `def...end`

List parameters at definition

```
def sayN(message, n)
i = 0
  while i < n
    puts message
    i = i + 1
  end
  return i
end
```

May omit parens on call

Invoke method

Like print, but adds newline

```
x = sayN("hello", 3)
puts(x)
```

Methods should begin with lowercase letter and be defined before they are called. Variable names that begin with uppercase letter are *constants* (only assigned once)

Note: Methods need not be part of a class
Methods: Terminology

- **Formal parameters**
  - Variable parameters used in the method
  - `def sayN(message, n)` in our example

- **Actual arguments**
  - Values passed in to the method at a call
  - `x = sayN("hello", 3)` in our example

- **Top-level methods are “global”**
  - Not part of a class. `sayN` is a top-level method.
Method Return Values

- Value of the `return` is the value of the last executed statement in the method
  - These are the same:

    ```ruby
    def add_three(x)
        return x+3
    end
    ```

    ```ruby
    def add_three(x)
        x+3
    end
    ```

- Methods can return multiple results (as an Array)

    ```ruby
    def dup(x)
        return x, x
    end
    ```
class Point
  def initialize(x, y)
    @x = x
    @y = y
  end
  def add_x(x)
    @x += x
  end
  def to_s
    return "(@x.to_s, @y.to_s)"
  end
end

p = Point.new(3, 4)
p.add_x(4)
puts(p.to_s)
Defining Your Own Classes

class Point
  def initialize(x)
    @x = x
  end
  def x=(x)
    @x = x
  end
  def x
    @x
  end
  private
  def prt
    "#{@x}"
  end
  # Make the below methods public
  public
    def to_s
      prt
    end
end

> p = Point.new(10)
#<Point:0x00007f8 @x=10>

> p.x= 100
100

> p.prt
NoMethodError
(private method `prt' called)
class Point
    def initialize(x)
        @x = x
    end
    def x=(x)
        @x = x
    end
    def x
        @x
    end
    private
    def prt
        "#{@x}"
    end
end

# Make the below methods public
public
    def to_s
        prt
    end
end

class Point
    attr_accessor :x
    attr_reader :y
    attr_writer :z
    private
    def prt
        "#{@x}, #{@y}"
    end
end

# Make the below methods public
public
    def to_s
        prt
    end
end
class Dog
  def smell(thing)
    "I smelled #{thing}"
  end
  def smell(thing, dur)
    "#{smell(thing)} for #{dur} seconds"
  end
end
fido = Dog.new
puts fido.smell("Alice", 3)

A. I smelled Alice for nil seconds
B. I smelled #{thing}
C. I smelled Alice
D. Error
Quiz 5: What is the output?

```ruby
class Dog
  def smell(thing)
    "I smelled #{thing}"
  end
  def smell(thing,dur)
    "#{smell(thing)} for #{dur} seconds"
  end
end

fido = Dog.new
puts fido.smell("Alice",3)
```

A. I smelled Alice for nil seconds
B. I smelled #{thing}
C. I smelled Alice
D. Error – call from Dog expected two args
Quiz 6: What is the output?

class Dog
  def smell(thing)
    "I smelled #{thing}"
  end
  def smelltime(thing,dur)
    "#{smell(thing)} for #{dur} seconds"
  end
end
fido = Dog.new
puts fido.smelltime("Alice",3)

A. I smelled Alice for 3 seconds
B. I smelled #{thing} for #{dur} seconds
C. I smelled Alice for 3 seconds
D. Error
Quiz 6: What is the output?

class Dog
  def smell(thing)
    "I smelled #{thing}"
  end
  
  def smelltime(thing,dur)
    "#{smell(thing)} for #{dur} seconds"
  end
end

fido = Dog.new
puts fido.smelltime("Alice",3)

A. I smelled Alice for seconds
B. I smelled #{thing} for #{dur} seconds
C. I smelled Alice for 3 seconds
D. Error
Update Existing Classes (Including Builtins!)

10.double => NoMethodError
(undefined method `double' for 10:Integer)

Add a method to the Integer class

class Integer
  def double
    self + self
  end
end

10.double => 20
Method naming style

- Names of methods that return *true* or *false* should end in 

- Names of methods that modify an object’s state should end in 

- Example: suppose \( x = [3,1,2] \) (this is an array)
  - \( x.member? \) 3 returns true since 3 is in the array \( x \)
  - \( x.sort \) returns a new array that is sorted
  - \( x.sort! \) modifies \( x \) in place
No Method Overloading in Ruby

- Thus there can only be one `initialize` method
  - A typical Java class might have two or more constructors

- No overloading of methods in general
  - You can code up your own overloading by using a variable number of arguments, and checking at run-time the number/types of arguments

- Ruby does issue an exception or warning if a class defines more than one `initialize` method
  - But last `initialize` method defined is the valid one
Recall that every class inherits from `Object`

```ruby
class A
  ## < Object
  def add(x)
    return x + 1
  end
end

class B < A
  def add(y)
    return (super(y) + 1)
  end
end

b = B.new
puts(b.add(3))
```

```ruby
b.is_a? A
true
b.instance_of? A
false
```
Quiz 7: What is the output?

class Gunslinger
  def initialize(name)
    @name = name
  end
  def full_name
    "#{@name}"
  end
end

class Outlaw < Gunslinger
  def full_name
    "Dirty, no good #{super}"
  end
end

d = Outlaw.new("Billy the Kid")
puts d.full_name

A. Dirty, no good Billy the kid
B. Dirty, no good
C. Billy the Kid
D. Error
class Gunslinger
  def initialize(name)
    @name = name
  end
  def full_name
    "#{@name}"
  end
end

class Outlaw < Gunslinger
  def full_name
    "Dirty, no good #{super}"
  end
end

d = Outlaw.new("Billy the Kid")
puts d.full_name

A. Dirty, no good Billy the kid
B. Dirty, no good
C. Billy the Kid
D. Error
Global Variables in Ruby

Ruby has two kinds of global variables

- Class variables beginning with `@@` (*static in Java*)
- Global variables across classes beginning with `$`

```ruby
class Global
  @@x = 0

  def Global.inc
    @@x = @@x + 1; $x = $x + 1
  end

  def Global.get
    return @@x
  end
end
```

```ruby
$x = 0
Global.inc
$x = $x + 1
Global.inc
puts(Global.get)
puts($x)
```

define a class ("singleton") method
What is a Program?

- In C/C++, a program is...
  - A collection of declarations and definitions
  - With a distinguished function definition
    - `int main(int argc, char *argv[]) { ... }
  - When you run a C/C++ program, it’s like the OS calls `main(...)`

- In Java, a program is...
  - A collection of class definitions
  - With some class (say, `MyClass`) containing a method
    - `public static void main(String[] args)
  - When you run `java MyClass`, the main method of class `MyClass` is invoked
A Ruby Program is...

- The class **Object**
  - When the class is loaded, any expressions not in method bodies are executed

```ruby
def sayN(message, n)
  i = 0
  while i < n
    puts message
    i = i + 1
  end
  return i
end

x = sayN("hello", 3)
puts(x)
```

defines a method of Object (i.e., top-level methods belong to Object)

- invokes `self.sayN`
- invokes `self.puts` (part of `Object`)