CS 360 Internet Programming
Ruby
Ruby Essentials

Daniel Zappala
Computer Science
Brigham Young University
1. **Classes, Objects, and Variables**
   - Methods
   - Inheritance
   - Attributes
   - Class Variables and Class Methods

2. **Containers, Blocks, and Iterators**
   - Containers
   - Blocks and Iterators

3. **Strings**
   - Format
   - Parsing
   - String Methods

4. **Methods**
   - Defining a Method
   - Argument Lists
   - Return Values
**Initialize Method**

```ruby
1 class Song
2   def initialize(name, artist, duration)
3     @name = name
4     @artist = artist
5     @duration = duration
6   end
7 end
8 song = Song.new("New York, New York", "Frank Sinatra", 260)
9 song2 = song
```

- initialize method called when a new object is created
- variables *reference* the object, so `song` and `song2` point to the same object in memory
Access Control

- **public** methods: can be called by any object
- **private** methods: can only be called by the instance
- **protected** methods: can be called by any object of the same class or a subclass

```
1 class MyClass
2   def method1  # default is 'public'
3     end
4   protected  # subsequent methods will be 'protected'
5     def method2  # will be 'protected'
6       end
7   private  # subsequent methods will be 'private'
8     def method3  # will be 'private'
9       end
10  public  # subsequent methods will be 'public'
11  end
```
```ruby
puts song.to_s

→ #<Song:0xb7cb9c1c>

class Song
  def to_s
    "Song: #{@name}--#{@artist} (#@duration)"
  end
end

song = Song.new("New York, New York","Frank Sinatra",260)
puts song.to_s

→ "Song: New York, New York—Frank Sinatra (260)"
```

- all objects inherit from class Object
- can override the methods of Object
Defining SubClasses

1 class KaraokeSong < Song
2   def initialize(name, artist, duration, lyrics)
3       super(name, artist, duration)
4       @lyrics = lyrics
5   end
6   def to_s
7       super + " [@lyrics]"
8   end
9 end

10 song = KaraokeSong.new("New York, New York","Frank Sinatra",260,"Start spreading the news . . . ")
11 puts song.to_s
12 -> "Song: New York, New York—Frank Sinatra (260) [Start spreading the news . . . ]"

- KaraokeSong is a subclass of Song, Song is the superclass
- super calls the same method of the superclass
### Attributes

- instance variables are private unless you expose them via methods

```ruby
class Song
  def name
    @name
  end
end
```

- attr_reader shortcut: the same as creating a method that returns the value of an instance variable
- automatically creates instance variables

```ruby
class Song
  attr_reader :name, :artist, :duration
end
```
Writable Attributes

```ruby
1 class Song
2   def duration=(new_duration)
3     @duration = new_duration
4   end
5 end
6 song.duration = 257
```

- If method name ends with an `=` symbol, Ruby lets you use `=` as an assignment operator for the attribute.

```ruby
1 class Song
2   attr_writer :duration
3 end
```

- `attr_writer` shortcut
- `attr_accessor` shortcut declares attribute as both readable and writeable
Class Variables

```ruby
class Song
  @@plays = 0
  def initialize(name, artist, duration)
    @name    = name
    @artist  = artist
    @duration= duration
    @plays   = 0
  end
  def play
    @plays  += 1
    @@plays += 1
    "This song: #@plays plays. Total #@@plays plays."
  end
end
```

- shared among all instances
Class Methods

- needed when a class method must work without being tied to a particular instance
- examples: `Song.new`, `File.delete`
- prefaced by the class name and a period

```ruby
1 class Song
2   def Song.version
3     "1.0–r1"
4   end
5 end

7 puts Song.version
8 -> 1.0–r1
```
```ruby
class SongList  # wrapper around an array
  def initialize
    @songs = Array.new
  end
  def append(song)
    @songs.push(song)
    self
  end
  def delete_first
    @songs.shift
  end
  def delete_last
    @songs.pop
  end
  def [](index)  # defines [] method
    @songs[index]
  end
end
```
Using Iterators

- for loop version

```ruby
1  def with_title(title)
2    for i in 0...@songs.length
3      return @songs[i] if title == @songs[i].name
4    end
5    return nil
6  end
```

- iterator version

```ruby
1  def with_title(title)
2    @songs.find{|song| title == song.name }
3  end
```

requires less knowledge about array implementation
Common Iterators

1. \[1, 3, 5, 7, 9\].find \{|v| v*v > 30 \} \rightarrow 7
2. \[1, 3, 5, 7, 9\].each \{|i| print i \} \rightarrow 13579
3. \[1, 3, 5, 7, 9\].collect \{|x| x.succ \} \rightarrow [2, 4, 6, 8, 10]
4. \[1, 3, 5, 7, 9\].inject \{|sum, element| sum+element\} \rightarrow 16
Creating an Iterator

```ruby
1   class File
2     def File.my_open(*args)
3         result = file = File.new(*args)
4         if block_given?
5             result = yield file
6                 file.close
7         end
8         return result
9     end
10    end
```

- use the iterator to define a block that must be run as a transaction
- example: a file open method that ensures the file closes itself when done
- returns an open file if no block given
Strings

- single-quoted
  - `\\` makes \`
  - `\'` makes ’
- double-quoted
  - many more escape sequences, e.g. `\n` and `\t`
  - value substitution with `#{expr}`
String Parsing

- file format:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/jazz/j00132.mp3</td>
<td>3:45</td>
<td>Fats Waller</td>
</tr>
<tr>
<td>2</td>
<td>/jazz/j00319.mp3</td>
<td>2:58</td>
<td>Louis Armstrong</td>
</tr>
<tr>
<td>3</td>
<td>/bgrass/bg0732.mp3</td>
<td>4:09</td>
<td>Strength in Numbers</td>
</tr>
</tbody>
</table>

- parser:

```ruby
1 File.open("songdata") do |song_file|
2   songs = SongList.new
3   song_file.each do |line|
4     file, length, name, title = line.chomp.split(/\s*\|\s*\)/)
5     songs.append(Song.new(title, name, length))
6   end
7   puts songs[1]
8 end
9 -> Song: Wonderful World—Louis Armstrong (2:58)
```
**Additional String Methods**

- extra spaces in the artist name
  - use `name.squeeze!(" ")`
  - the ! modifies the name in place
- convert 2:58 into seconds
  - use `mins,secs = length.split(/:/)` and then convert
  - or use `mins,secs = length.scan(/\d+/)`, with regular expression
- see the library reference for more
Defining a Method

- special method names
  - trailing `?:` acts as a query
  - trailing `!:` dangerous, or modifies the calling object
  - trailing `=:` may be used for assignment

- default arguments

```
1  def cool_dude(arg1="Miles", arg2="Coltrane", arg3="Roach")
2     "#{arg1}, #{arg2}, #{arg3}".
3  end
4  cool_dude -> Miles, Coltrane, Roach
5  cool_dude("Bart") -> Bart, Coltrane, Roach
6  cool_dude("Bart", "Elwood") -> Bart, Elwood, Roach
7  cool_dude("Bart", "Elwood", "Linus") -> Bart, Elwood, Linus
```
Variable-Length Argument Lists

```python
1  def varargs(arg1, *rest)
2      "Got #{arg1} and #{rest.join(', ')}"
3  end
4  varargs("one") -> "Got one and 
5  varargs("one", "two") -> "Got one and two"
6  varargs "one", "two", "three" -> "Got one and two, three"
```
Return Values

- returns value of last statement executed
- can return more than one value in an array

```ruby
1 def meth_three
2 100.times do |num|
3   square = num*num
4   return num, square if square > 1000
5 end
6 end
7 num, square = meth_three
```