CSE M226: Programming Languages

Lecture 9:Reflections

Rokan Uddin Faruqui

Dept. of Computer Science and Engineering
University of Chittagong, Bangladesh
Email: rufaruqui@cu.ac.bd

Reflection/Introspect

examining aspects of a program from within the program itself

A program can discover the following information about itself:

- What objects it contains

Reflection/Introspect

examining aspects of a program from within the program itself

A program can discover the following information about itself:

- What objects it contains
- Its class hierarchy

Reflection/Introspect

examining aspects of a program from within the program itself

A program can discover the following information about itself:

- What objects it contains
- Its class hierarchy
- The attributes and methods of objects

Reflection/Introspect

examining aspects of a program from within the program itself

A program can discover the following information about itself:

- What objects it contains
- Its class hierarchy
- The attributes and methods of objects
- Information on methods

Reflection/Introspect

examining aspects of a program from within the program itself

A program can discover the following information about itself:

- What objects it contains
- Its class hierarchy
- The attributes and methods of objects
- Information on methods

Looking at objects

```
1
  a = Complex(1, 2)
3 b = Complex(99, -100)
  ObjectSpace.each_object(Complex) {|x| puts x }
5
produces:
7   0+1i
    99-100i
9   1+2i

11
```

Looking at objects

We can get a list of all the methods to which an object will respond

```
r = 1..10 # Create a Range object
list = r.methods
list.length # => 111
list[0..3] # => [:==, :===, :eql?, :hash]

```

Looking at objects

We can check to see whether an object responds to a particular method:

```
r = 1..10
r.respond_to?("frozen?") # => true
r.respond_to?(:has_key?) # => false
"me".respond_to?("==") # => true
```

Looking at objects

```
class Demo
# class body
end

Demo.class_variables # => [:@@var]
Demo.constants(false) # => [:CONST]
demo = Demo.new
demo.instance_variables # => []
# Get 'public_method' to return its local variables
# and set an instance variable
demo.public_method # => [:i, :j]
demo.instance_variables # => [:@inst]


```

Dynamically calling methods

```
"John Coltrane".send(:length) # => 13
"Miles Davis".send("sub", /iles/, '.') # => "M.
    Davis"

trane = "John Coltrane".method(:length)
miles = "Miles Davis".method("sub")

trane.call # => 13
miles.call(/iles/, '.') # => "M. Davis"

```

Compile Time? Runtime? Anytime!

- In Ruby - there isn't a big difference between **compile time** and **runtime**. It's all the same.
- You can add code to a running process.
- You can redefine methods on the fly, change their scope from public to private, and so on. You can even alter basic types, such as Class and Object.
- Once you get used to this flexibility, it is hard to go back to a static language such as $C++$ or **Java**.