BCC
Engenharia de Software
2018.2

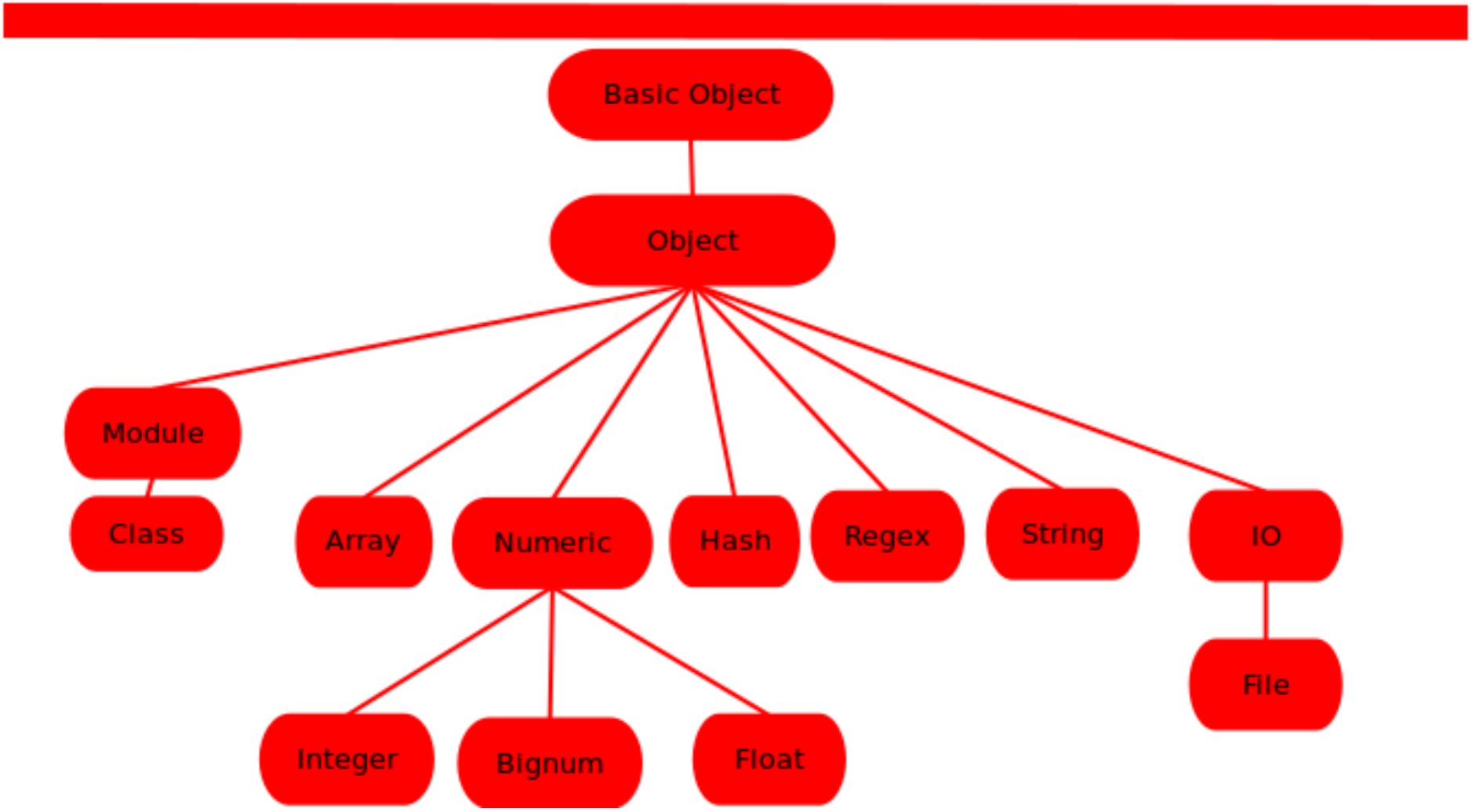
Professor Rodrigo Andrade

Ruby

Orientada a objetos

- Tudo é objeto
  - Tipos primitivos, nil…
- Todo objeto é instância de alguma classe
- Baseado em Smalltalk

Class Heirarchy



Chamada de método

- Toda operação representa uma chamada de método
- Toda operação devolve um valor

```
5+3
x == y
```

```
5.+(3)
x.==(y)
```

```
a.b
```

Hello World

```
puts ‘Hello World’
puts(‘Hello World’)
```

Dinamicamente tipada

- Tipos inferidos em tempo de execução
- Não declaramos os tipos de variáveis ou retornos

Dinamicamente tipada

```
x = 7              # inteiro
x = "house"  # string
x = 7.5            # real

# tudo eh objeto
'I love Ruby'.length
```

Variáveis

```
s = 'Hello’
s << ‘World!’
puts s

x = 10
puts x + 1
```

```
#variavel local
inteiro = 22
#variavel de instância
@inteiro = 22
#variavel global
$inteiro = 22
```

Métodos

```
def hello1(name)
  'Hello ' + name
end
puts(hello1('Alice'))
```

```
def hello2 name2
  'Hello ' + name2
end
puts(hello2 'Bob')
```

Métodos com ?

```
2.equal?(2)

x = ‘World’
“Hello, #{x}”.nil?
```

Comparativo com Java

Java	Ruby
`class MyString extends String`	`class MyString < String`
`class MyCollection extends Array`	`class MyCollection < Array`
`  implements Enumerable`	`  include Enumerable`
Variável estática: `static int anInt = 3`	Variável de classe: `@@an_int = 3`
Variável de instância: `this.foo = 1`	Variável de instância: `@foo = 1`
Método estático:	Método de classe:
`public static int foo(...)`	`def self.foo ... end`
Método de instância: `public int foo(...)`	Método de instância: `def foo ... end`

Estruturas básicas - IF

```
var = 5
if var > 4
  puts ‘maior que quatro’
elsif var == 4
  puts ‘igual a quatro’
else
 puts ‘menor que quatro
end
```

Estruturas básicas - UNLESS

```
unless login.nil?
  puts "Valid login"
end
```

```
var = 0
while var < 10
  puts var
  var += 1
end
```

Estruturas básicas - FOREACH

```
locations = ['Pune', 'Mumbai', 'Bangalore']
locations.each do |loc|
  puts 'I love ' + loc + '!'
end
```

Estruturas básicas - UNTIL

```
i = 0
num = 5
until i >= num  do
   puts('i ainda eh menor')
   i +=1
end
```

Símbolos Ruby

```
know_ruby = :yes
if know_ruby == :yes
  puts 'You are a Rubyist'
else
  puts 'Start learning Ruby'
end
```

- Símbolo é o objeto mais básico de Ruby
  - Nome e ID interno

Leitura de arquivos em JAVA

```
Scanner in = new Scanner(new
FileReader("filename.txt"));
while(in.hasNext()) {
    System.out.println(in.next());
}
in.close();
```

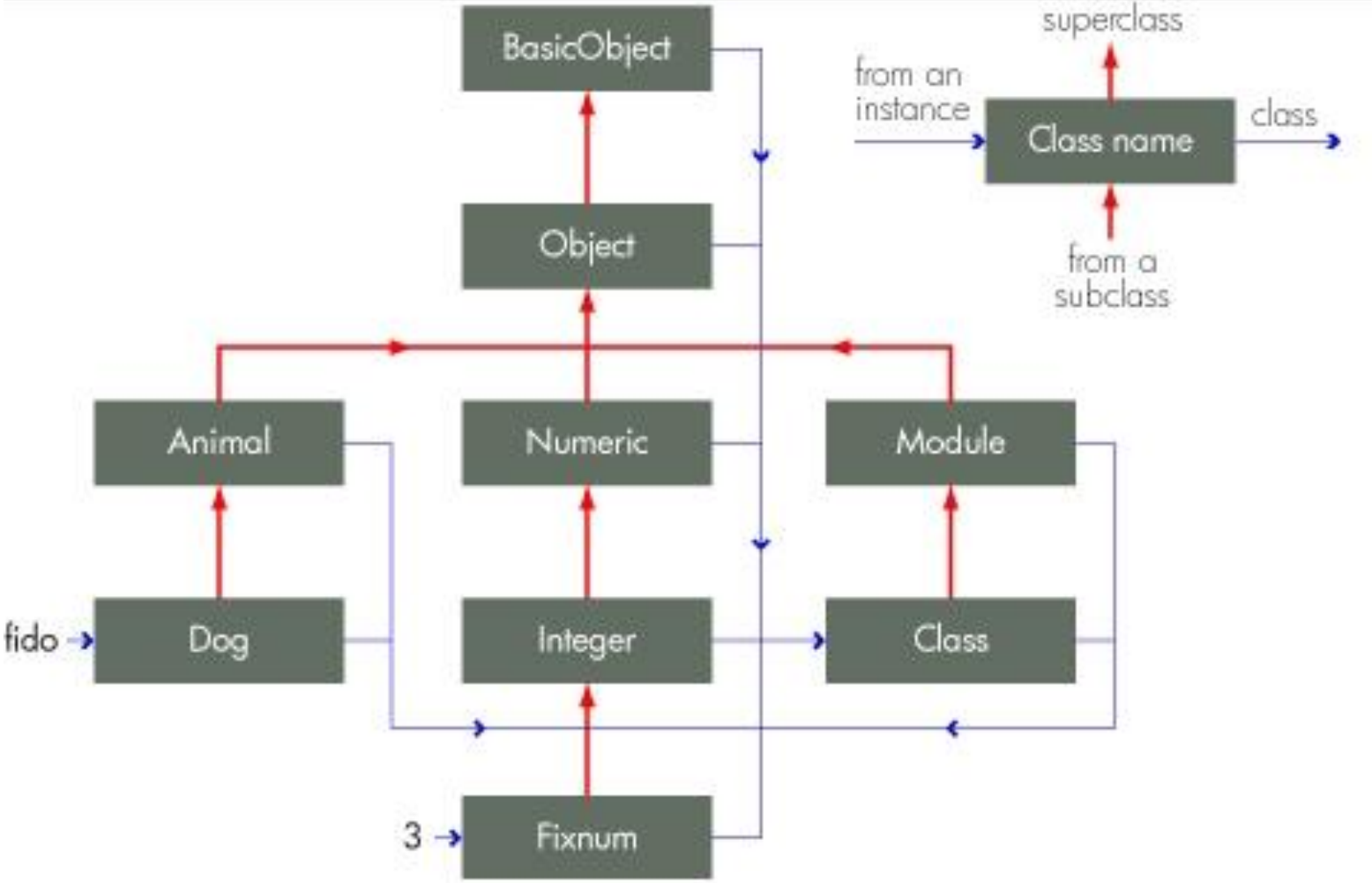
Leitura de arquivos em Ruby

```
File.open('arquivo.txt', 'r') do |f1|
  while line = f1.gets
    puts line
  end
end
```

Escrita de arquivos

```
File.open(arquivo.txt', 'w') do |f2|
  f2.puts ‘Linha1\nLinha2’
end
```

Classes em Ruby



Exemplo de uma classe

```
class Dog
  attr_accessor :breed,:name
  def initialize(breed, name)
    @breed = breed
    @name = name
  end
end
```

```
d = Dog.new('Bulldog', 'Dushi')
puts d.class.to_s
```

Toda programação é metaprogramação

```
class Dog
  attr_accessor :breed,:name
  …
end
```

Definindo get e set dinamicamente

Definindo só get dinamicamente

```
class Dog
  attr_reader :breed,:name
  …
end
```

Metaprogramação

- Criação de Código em tempo de execução
- Possível modificar classe em tempo de execução
- Método padrão method_missing é chamado caso Ruby não consiga resolver

Importar arquivos

require ‘dog’

load ‘dog.rb’

require_relative ‘dog’

Herança

```
class Mamifero
  def respirar
    puts ‘inspirar e expirar’
  end
end

class Gato < Mamifero
  def miar
    puts ‘miau’
  end
end
```

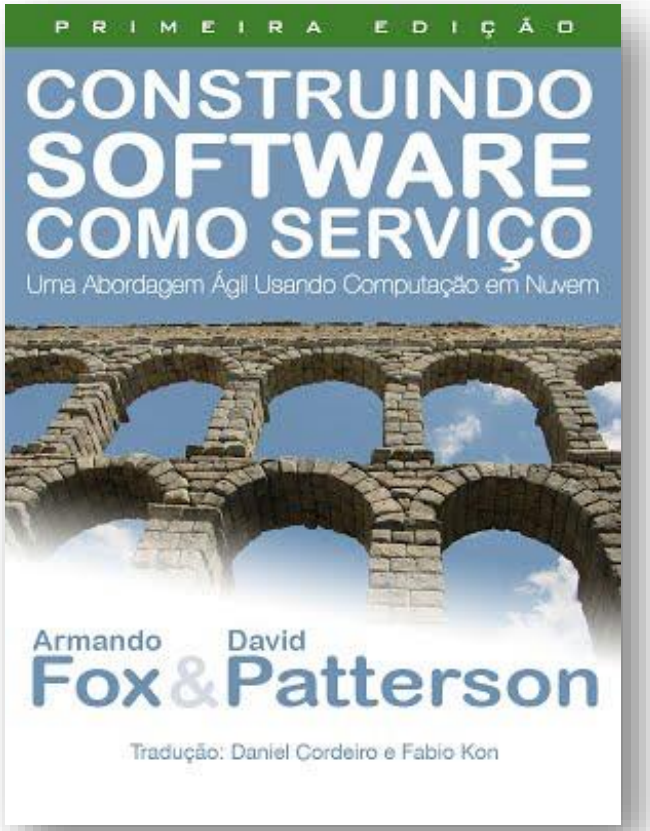
Levantando exceções

```
def inverse(x)
  raise ArgumentError,
   'Argument is not numeric’
    unless x.is_a? Numeric
  1.0 / x
end
```

Tratando exceções

```
def raise_and_rescue
  begin
    raise ‘Um erro ocorreu’
  rescue
    puts ‘Erro tratado’
  end
end
```

Referências



rubylearning.com

Helping Ruby Programmers become Awesome!

http://rubylearning.com/satishtalim/tutorial.html

Capítulo 3