Sesi 2: OOP with Ruby

Overview

Seperti yang telah disebutkan sebelumnya, Ruby merupakan bahasa pemrograman yang berorientasi objek. Semua yang bisa kita manipulasi di Ruby adalah sebuah objek.

Di dalam pemrograman berorientasi objek, sebuah kelas memuat berbagai properti, seperti method dan variabel. Kelas bisa mewarisi properti dari kelas pendahulu (parent class) atau superclass nya, sehingga membentuk hierarki kelas dengan base class di tempat paling atas. Kelas paling atas (base class) di Ruby adalah `Object`.

Sebuah kelas didefinisikan dengan keyword `class`, dan diakhiri dengan sebuah `end`.

```
class Hello
  def initialize( name )
    @name = name
  end

  def hello_txt
    puts "Hello, " + @name + "!"
  end
end

hi = Hello.new("World")
hi.hello_txt # => Hello, World!
```

Method `initialize` pada Ruby merupakan constructor, yang otomatis akan dieksekusi pertama kali setelah objek kelas tersebut diciptakan. Pada contoh di atas, method `initialize` akan mendefinisikan sebuah instance variable `@name` dan meng-assign-nya dengan argumen yang dimasukkan.

Instance Variables

Instance variabel adalah variabel yang ada di dalam instance sebuah kelas, dan cakupannya terbatas. Sebuah instance variable diawali dengan tanda `@`, seperti instance variable pada contoh sebelumnya `@name`. Kita bisa mendefinisikan sebuah instance variable di dalam atau di luar sebuah method. Namun, kita hanya bisa mengaksesnya dari luar objek dengan menggunakan method accessor: getter dan setter.

```
class Hello
  def initialize( name )
    @name = name
  end

  def hello_txt
    puts "Hello, " + @name + "!"
  end

  def name
    @name
```

```
  end

  def name=(value)
    @name = value
  end
end

hi = Hello.new("World")
hi.name # => World
hi.name = "CCBI"
hi.name # => CCBI
```

Accessors

Ruby menyederhanakan pembuatan getter dan setter secara meta-programming dari kelas `Module`, yakni menggunakan method `attr`, `attr_reader`, `attr_writer` dan `attr_accessor`.

Method `attr` akan membuat sebuah getter. Jika argumen keduanya `true`, ia juga akan membuat sebuah setter.

```
class Bird
  attr :chirp, true
end

Bird.instance_methods - Object.instance_methods # => ["chirp","chirp="]

bird = Bird.new
bird.chirp = "Tweet!"
bird.chirp # =>  Tweet!
```

Method `attr_reader` akan menciptakan getter, sedangkan method `attr_writer` akan menciptakan setter.

```
class Bird
  attr_reader :chirp # getter
  attr_writer :chirp # setter
end

Bird.instance_methods - Object.instance_methods # => ["chirp","chirp="]
```

Method `attr_accessor` akan memberikan efek yang sama seperti jika memanggil `attr_reader` dan `attr_writer`.

```
class Bird
  attr_accessor :chirp, :fly, :spawn
end

Bird.instance_methods - Object.instance_methods # => [:chirp, :chirp=, :fly,
:fly=, :spawn, :spawn=]
```

Class Variables

Class variable merupakan variabel yang digunakan bersama oleh semua instance suatu kelas, mirip seperti konsep static variable di Java atau C++. Class variable di Ruby harus diawali dengan @@.

```
class Bird
  @@count = 0

  def initialize
    @@count += 1
  end

  def count
    @@count
  end
end

b1 = Bird.new
b2 = Bird.new
b3 = Bird.new
b1.count # => 3
```

Class Methods

Class method (static method) merupakan method yang berasosiasi dengan kelas atau module di Ruby, bukan instance dari suatu kelas. Kita bisa mengeksekusi sebuah class method tanpa sebuah instance objek. Class method di Ruby selalu diawali dengan `self.`

```
class Bird
  def self.tweet
    puts "Tweeeeet!!"
  end
end

Bird.tweet # => Tweeeeet!!
```

Inheritance

Pewarisan (inheritance) pada Ruby dilakukan dengan menggunakan operator `<`. Tidak seperti C++ yang mendukung multiple inheritance, Ruby menggunakan single inheritance, yaitu sebuah kelas hanya bisa mewarisi properti dari 1 parent class.

```
class Bird
  attr_accessor :chirp, :egg
end

class Duck < Bird
  attr_accessor :quack
end

d = Duck.new
put d.respond_to?(:egg) # => true
```

Jika kelas `Bird` di atas berada di file yang berbeda dengan kelas `Duck`, maka terlebih dulu kita perlu `require` file tersebut. Ruby akan mencari file tersebut berdasarkan `$LOAD_PATH` yang telah didefinisikan.

Access Control

Akses ke metode dan konstan pada suatu kelas di Ruby, bisa diatur dengan kata kunci berikut ini:

- `public` Default, bisa diakses oleh siapa saja dari mana saja.
- `private` Membatasi cakupan suatu metode hanya bisa diakses oleh objek tersebut.
- `protected` Bisa diakses oleh objek dari kelas tersebut dan dari kelas-kelas turunannya.

```
class Dog
  def bark
    puts "Woof!"
  end

  private

  def walk
    puts "Walk!"
  end

  protected

  def sit
    puts "Sit!"
  end

  def eat
    puts "Eat!"
  end
end

class Chihuahua < Dog
  def sitdown
    self.sit
  end

  def run
    self.walk
  end
end

d = Dog.new
d.bark # => 'Woof!'
d.walk # => error
d.sit  # => error
d.eat  # => error

c = Chihuahua.new
c.bark    # => 'Woof!'
c.sitdown # => 'Sit!'
c.run     # => error
```

Modules

Pada Ruby, modul itu mirip seperti kelas, tetapi tidak bisa di-instantiate. Sebuah kelas dapat mengandung > 1 modul. sehingga ketika kelas tersebut di-instantiate, maka semua properti dari modul-modul tersebut akan tercakup pada objek si kelas. Hal ini memungkinkan konsep yang mirip

dengan multiple inheritance. Identifier pada tiap-tiap modul akan di-override oleh definisi yang terakhir. Dengan demikian, penamaan yang bertabrakan bisa dihindari.

Pada Ruby, modul juga merupakan sebuah namespace. Sebuah namespace adalah sekumpulan nama (seperti nama-nama method), yang memiliki cakupan/konteks. Sebuah kelas Ruby juga bisa dianggap sebagai namespace.

Penamaan modul harus diawali dengan huruf besar. Sebuah modul bisa mengandung metode, konstan, modul lain dan bahkan kelas. Sebuah modul bisa diwarisi oleh modul lain, tetapi tidak bisa diwarisi oleh sebuah kelas.

```
module FlyingAnimal
  def fly
    puts "Flying high.."
  end
end

module LayingAnimal
  def egg
    puts "Spawning.."
  end
end

class Bird
  include FlyingAnimal, LayingAnimal
end

b = Bird.new
b.fly # => Flying high..
b.egg # => Spawning..
```

Variable

Di Ruby, variabel tidak perlu dideklarasikan. Ruby menggunakan "duck typing", semacam dynamic typing. Jika sebuah nilai bertingkah laku seperti tipe tertentu, misalnya integer, maka Ruby akan memberinya suatu konteks dan memberlakukannya pada kontes tersebut. If it walks like a duck, quacks like a duck, flies like a duck, and swims like a duck (or integer or float, etc.), then it is probably a duck.

Sekilas ulasan kembali beberapa jenis variable di Ruby.

Local Variable

Variabel lokal memiliki cakupan yang lokal juga sesuai konteksnya, misalnya di dalam suatu metode atau perulangan. Variabel lokal pada Ruby hanya boleh diawali dengan huruf kecil atau underscore (_).

```
class Foo
  def self.bar
    Foobar = 1 # error
    _foobar = 2
    puts _foobar
  end
end
```

Instance Variable

Instance variable hanya bisa diakses dari luar suatu objek melalui accessor method. Instance variable selalu diawali dengan tanda @.

Class Variable

Class variable merupakan variabel yang digunakan bersama oleh semua instance suatu kelas, mirip seperti konsep static variable di Java atau C++. Class variable di Ruby selalu diawali dengan @@.

Global Variables

Global variable bisa diakses dari mana saja. Di Ruby, global variable harus diawali dengan tanda $.

Constants

Variabel konstan di Ruby harus diawali dengan huruf besar, dan biasanya (secara konvensi) nama sebuah konstan ditulis dengan huruf besar semua.