1. Tên một số mô hình lập trình:

* Lập trình Logic (Logical programming)
* Lập trình thủ tục (Procedural programming)
* Lập trình hàm (Functional programming)
* Lập trình hướng đối tượng (Object – oriented programming).

1. Lập trình hướng đối tượng là gì:

* OOP (viết tắt của Object Oriented Programming) – lập trình hướng đối tượng là một phương pháp lập trình dựa trên khái niệm về lớp và đối tượng. OOP tập trung vào các đối tượng thao tác hơn là logic để thao tác chúng, giúp code dễ quản lý, tái sử dụng được và dễ bảo trì.

1. Ưu và nhược điểm của lập trình hướng đối tượng:

* Ưu điểm:
* Dữ liệu không bị thay đổi tự do trong chương trình như lập trình cấu trúc.
* Khi thay đổi cấu trúc dữ liệu của một đối tượng, không cần thay đổi các mã nguồn của các đối tượng khác mà chỉ cần thanh đổi một một số hàm thành phần
* Có thể sử dụng lại mã nguồn qua kế thừa
* Phù hợp các phần mềm phức tạp, lớn.
* Được đánh giá là dễ học, năng suất, đơn giản, dễ bảo trì, dễ mở rộng…
* Nhược điểm:

+ Có thể khiến dữ liệu được xử lý tách rời, khi cấu trúc dữ liệu thay đổi sẽ dẫn đến việc thuật toán bị thay đổi theo. Bên cạnh đó OOP không tự động khởi tạo và giải phóng dữ liệu động, không mô tả được đầy đủ và trung thực hệ thống trong thực tế.

1. Các tính chất của OOP và cho ví dụ:

* Tính kế thừa: Lớp cha có thể chia sẻ dữ liệu và phương thức cho các lớp con, các lớp con không cần định nghĩa lại những logic chung, giúp chương trình ngắn gọn.

Ví dụ:

Text

Description automatically generated

Kết quả:



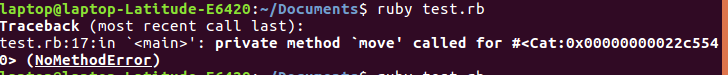
* Tính đóng gói (Encapsulation): Gói gọn các phương thức, các biến trong 1 class, không cho class con truy cập đến các phương thức hay biến một cách tùy chọn, giúp bảo mật dữ liệu trong quá trình kế thừa dữ liệu của lớp cha và con, nó thể hiện qua các phương thức public, private, protected.

Ví dụ:

Text

Description automatically generated

Kết quả:



* Tính đa hình (Polymorphism): Theo cách hiểu của mình thì đa tức là nhiều, hình được hiểu là hình thức từ nó thì hiểu khái quát đa hình là biểu diễn các method dưới nhiều hình thức khác nhau, đối với trong cùng 1 class thì nó có thể biểu diễn nhiều method giống nhau nhưng khác tham số thì được gọi là overload, còn với class kế thừa nhau các method cùng tên thì đó là override.

Ví dụ về overload:

Text

Description automatically generated

Kết quả:



**Tương tự ví dụ về override như sau**: thì mình xin chia sẽ 1 chút đó là overide tức là cùng 1 method giống nhau nhưng trả về giá trị khác với giá trị cha thôi, ví dụ thằng cha trả về animal nhưng thằng con không thích nó muốn trả về cat thì lúc này mình có thể dùng overide nhé.

Text

Description automatically generated

Kết quả:



* Tính trừu tượng (Abstraction): Tính trừu tượng giúp loại bỏ những thứ phức tạp, không cần thiết của đối tượng và chỉ tập trung vào những gì cốt lõi, quan trọng.

Ví dụ: Quản lý nhân viên thì chỉ cần quan tâm đến những thông tin như:

* Họ tên
* Ngày sinh
* Giới tính
* …

Chứ không cần phải quản lý thêm thông tin về:

* Chiều cao
* Cân nặng
* Sở thích
* Màu da
* …

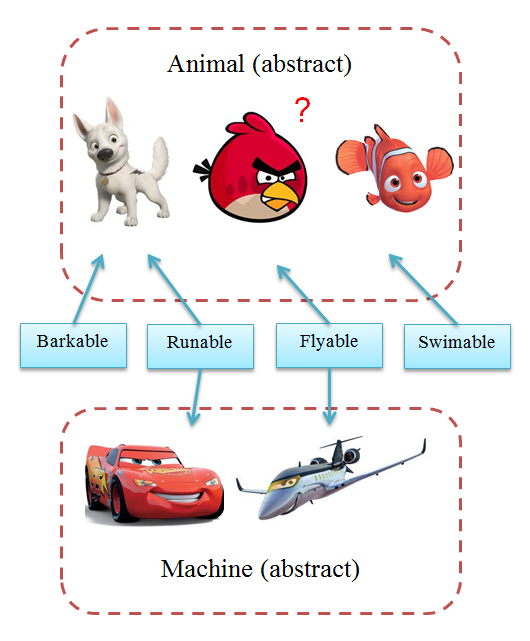
1. Overload và override method

* Overload: Nạp chồng phương thức đơn giản là có vài phương thức trùng tên nhưng khác nhau về đối số trong cùng 1 class. Cài chồng phương thức cho phép ta tạo nhiều phiên bản của một phương thức, mỗi phiên bản chấp nhận một danh sách đối số khác nhau, nhằm tạo thuận lợi cho việc gọi phương thức.
* Override: Override là một tính năng cho phép một lớp con cung cấp một triển khai cụ thể của phương thức đã được cung cấp bởi một trong các lớp cha của nó. Nói dễ hiểu hơn, nếu lớp con có một hoặc nhiều phương thức giống với một trong các lớp cha của nó, thì đó là ghi đè phương thức.

1. Phân biệt Abstract và Interface, trong Ruby chúng được thể hiện như thế nào?

– **Abstract class**: là một class cha cho tất cả các class có cùng bản chất. Bản chất ở đây được hiểu là kiểu, loại, nhiệm vụ của class. Hai class cùng hiện thực một interface có thể hoàn toàn khác nhau về bản chất. Hiểu đơn giản như một thằng con (child class) chỉ có thể là con của một thằng cha, có tính cách giống cha (abstract class) nó.

–**Interface**: là một chức năng mà bạn có thể thêm và bất kì class nào. Từ chức năng ở đây không đồng nghĩa với phương thức (hoặc hàm). Interface có thể bao gồm nhiều hàm/phương thức và tất cả chúng cùng phục vụ cho một chức năng.



1. Khai báo class, sử dụng constructor, getter/ setter

class Person

#constructor:

def initialize(name, age, address)

@name = name

@age = age

@address = address

end

#Setter:

attr\_writer :name, :age, :address

#or

def setName(name)

@name = name

end

def setAge(age)

@age = age

end

def setAddress(address)

@address = address

end

#Getter:

attr\_reader :name, :age, :address

#or

def getName(name)

return @name

end

def getAge(age)

return @age

end

def getAddress(address)

return @address

end

#or use for bose getter / setter by:

attr\_accessor :name, :age, :address

end

1. So sánh class và module:

- Module nó giống như 1 cái thư viện, có thể sử dụng nơi nào cần nó, còn class nó chỉ được sử dụng thông qua đối tượng thể hiện của nó.

- Class có tính kế thừa còn module thì không. Ngược lại, module có thể include vào bất cứ nơi nào trong khi class thì chỉ có thể sử dụng thông qua object, ...

1. Ruby có đa kế thừa hay không ? Nêu sự thể hiện của đa kế thừa trong Ruby

- Ruby không hỗ trợ đa kế thừa, nhưng nó vẫn có thể làm được điều đó qua cơ chế mix-in các module lại.

module Dog

    def bark

        puts "Bowl bowl"

    end

    def run

        puts "#{self.class.name} can run"

    end

    def priority

        puts "Im first"

    end

end

module Cat

    def scratch

        puts "Crake crake"

    end

    def run

        puts "#{self.class.name} can run"

    end

    def priority

        puts "Im second"

    end

end

class Doat

    include Cat

    include Dog

    def thinking(string)

        puts "#{self.class.name} is thinking about: #{string}!"

    end

end

doat = Doat.new

doat.scratch()

doat.bark()

doat.run()

doat.thinking("who am i, dog or cat ?")

doat.priority()

10) Kể tên các loại biến, cách khai báo và phạm vi sử dụng:

❖ Global variable (biến toàn cục) - ký hiệu $  
➔ Có sẵn ở mọi nơi trong tập lệnh Ruby của bạn  
❖ Local variable (biến cục bộ) cách tạo biến là chữ cái thường  
➔ Nó phụ thuộc vào phạm vi (tạo phạm vi mới với lớp mới, mô-đun mới, phương thức)  
❖ Instance variable (biến phiên bản)- ký hiệu @  
➔ Chỉ khả dụng trong một đối tượng cụ thể, trên tất cả các phương thức trong một lớp  
Không có sẵn trực tiếp từ các định nghĩa lớp.  
❖ Class variable (biến lớp) - ký hiệu @@  
➔ Có sẵn từ định nghĩa lớp và bất kỳ lớp con nào. Không có sẵn từ bất kỳ đâu bên ngoài.  
❖ Constants (hằng số) - Bắt đầu bằng một chữ cái viết hoa chữ cái đầu hoặc viết hoa toàn bộ tên biến . Biến hằng trong ruby có thể thay đổi giá trị.