ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH

BÁO CÁO BÀI TẬP LỚN

LẬP TRÌNH NÂNG CAO (MR) - CO203E

GVHD: Trương Tuấn Anh
SV: Nguyễn Tuấn Minh - 2110359

TP. HỒ CHÍ MINH, THÁNG 5/2023

Mục lục

1 So sánh lập trình hướng đối tượng (Object Orientated Programming - OOP) giữa Java và Ruby

Lập trình hướng đối tượng (OOP) nhắc đến những ngôn ngữ lập trình sử dụng đối tượng trong việc lập trình , sử dụng đối tượng là thành phần chính để biểu diễn những gì diễn ra trong chương trình. Mỗi đối tượng sẽ có những thuộc tính (attribute) chứa dữ liệu về đối tượng và những phương thức (method) giúp đối tượng có thể được sử dụng cho các mục đích chức năng khác nhau. Thông thường, khi nhắc về lập trình hướng đối tượng, người ta sẽ đề cập đến 4 tính chất quan trọng sau:

- **Tính đóng gói (Encapsulation):** Cho phép khả năng che dấu thông tin, tính chất này cho phép việc thay đổi các thuộc tính và phương thức quan trọng của đối tượng chỉ có thể được thực hiện thông qua các phương thức của chính nó, do người viết mã có khả năng quyết định khả năng truy cập từ bên ngoài đối tượng vào những phương thức và thuộc tính quan trọng. Trong các ngôn ngữ lập trình, tính chất này thường được thể hiện bằng các chỉ định truy cập (access modifier). Tính đóng gói đảm bảo sự toàn vẹn thông tin của đối tượng.
- **Tính trừu tượng (Abstraction):** là khả năng của chương trình bỏ qua hay không chú ý đến một số khía cạnh của thông tin mà nó đang trực tiếp làm việc lên, nghĩa là nó có khả năng tập trung vào những cốt lõi cần thiết. Tính trừu tượng còn thể hiện qua việc một đối tượng ban đầu có thể có một số đặc điểm chung cho nhiều đối tượng khác như là sự mở rộng của nó nhưng bản thân đối tượng ban đầu này có thể không có các biện pháp thi hành. Tính trừu tượng này thường được xác định trong khái niệm gọi là lớp trừu tượng hay lớp cơ sở trừu tượng.
- **Tính kế thừa (Inheritance):** do việc sử dụng lại các thuộc tính của đối tượng là thường xuyên đối với lập trình hướng đối tượng, tính kế thừa cho phép một đối tượng có thể có sẵn các đặc tính mà đối tượng khác có. Tính kế thừa giúp các đối tượng có thể chia sẻ hay mở rộng các đặc tính có sẵn mà không phải tiến hành định nghĩa lại.
- **Tính đa hình (Polymorphism):** tính đa hình cho phép các đối tượng khác nhau có thể hiện thực cùng một phương thức theo nhiều cách khác nhau. Tính đa hình được thể hiện trong hai giai đoạn: giai đoạn biên dịch (compile time) và giai đoạn thực thi (run time)
  - Giai đoạn biên dịch: được thể hiện qua việc nạp chồng (overloading) cho phép sử dụng cùng một tên gọi cho các hàm "giống nhau" (có cùng mục đích) nhưng khác nhau về kiểu dữ liệu tham số hoặc số lượng tham số (nạp chồng hàm) hoặc định nghĩa toán tử cho có sẵn phục vụ cho dữ liệu riêng do bạn tạo ra (nạp chồng toán tử)
  - Giai đoạn thực thi: được thể hiện qua việc ghi đè (overriding) cho phép một phương thức ở lớp cha có thể được hiện thực theo nhiều cách khác nhau ở các lớp con. Thao tác ghi đè thường liên quan đến từ khóa virtual (ảo)

1.1 Tổng quan

Java là ngôn ngữ lập trình biên dịch (compile) và là nền tảng điện toán để phát triển ứng dụng do Sun Microsystems phát triển, sau này được Tập đoàn Oracle mua lại vào năm 2009. Ngày nay, nền tảng Java thường được sử dụng làm nền tảng để phát triển và phân phối nội dung trên web. Java rất nhanh, mạnh mẽ, đáng tin cậy và an toàn.

Ruby là một ngôn ngữ thông dịch (interpreted language) được thiết kế và phát triển bởi

Yukihiro Matsumoto vào giữa năm 1990. Ruby là mã nguồn mở và nó cũng có sẵn miễn phí trên Web, phải tuân theo giấy phép.

Cả Java và Ruby đều có một hệ sinh thái các thư viện, công cụ, frameword... rất đa dạng giúp hỗ trợ rất nhiều cho lập trình viên

Ruby có thể được chạy trực tiếp mà không cần biên dịch để tạo bytecode:

```
>ruby helloruby.rb
```

Trong khi các chương trình viết bằng Java đều phải được biên dịch thông qua trình biên dịch javac trước khi khởi chạy:

```
>javac hellojava.java #compile
>java hellojava       #run compiled file
```

Cả Java và Ruby đều là các ngôn ngữ lập trình hướng đối tượng, tuy nhiên Ruby là ngôn ngữ lập trình thuần hướng đối tượng (pure object-orientated programming language) còn Java là ngôn ngữ lập trình kết hợp giữa hướng lập trình đối tượng và lập trình hàm (hybrid object-orientated programming language).

Về mặt tốc độ và độ phức tạp, Java có độ phức tạp nhỉnh hơn Ruby còn Ruby lại có tốc độ nhỉnh hơn Java

Trong Java câu lệnh **import** được sử dụng để thêm một gói thư viện, đối với Ruby là **require** hoặc **load**

Về kiểu biến (typed variables), cả Ruby và Java đều có kiểu biến là **strongly typed** nghĩa là kiểu của một biến hoặc một đối tượng sẽ không thế tự thay đổi một cách đột ngột mà phải thông qua lời gọi hàm, ép kiểu,... Tuy nhiên sự khác biệt nằm ở chỗ Ruby có biến dạng động (dynamically typed) còn Java thì là biến dạng cố định (statically typed). Biến dạng cố định được trình biên dịch (compiler) yêu cầu xác định rõ kiểu dữ liệu của biến trước khi sử dụng và mỗi biến chỉ có một kiểu có định liên kết với nó, trong khi biến dạng động không cần phải khai báo rõ kiểu dữ liệu khi sử dụng, kiểu dữ liệu của biến sẽ tùy thuộc vào dữ liệu mà biến đang chứa. Ví dụ:

- Ruby:

```
s = "This is string" #string type
s = 1                #integer type

```

- Java:

```
String s = "This is string"; #string type
String s = 1                 #error

```

Cũng chính vì vậy, trong Java thì có ép kiểu (casting) còn Ruby thì không có kiểu dữ liệu có thể ngầm thay đổi trong quá trình thực thi.

1.2 Lớp (Class) và Đối tượng (Object)

Trong lập trình hướng đối tượng, một Lớp được hiểu là một nguyên mẫu (prototype) nhằm xác định những thuộc tính và phương thức chung cho tất cả các đối tượng cùng loại.

Một Đối tượng (Object) là một hiện thực cụ thể của Lớp, một Lớp có thể tạo ra nhiều Đối tượng khác nhau với những giá trị thuộc tính khác nhau.

Trong Java, chỉ có các Lớp mới có thể tạo ra các đối tượng, còn đối với Ruby tất cả đều được coi như đối tượng, gồm các biến, các giá trị và các phương thức.

1.2.1 Definition

Đối với Java, một phương thức/lớp được định nghĩa bằng việc sử dụng cặp ngoặc nhọn , tên một Lớp có thể được viết thường tuy nhiên lập trình viên được khuyến khích viết in hoa để tăng tính đọc hiểu cho code, một phương thức cần có kiểu trả về, các chỉ định truy cập (access modifier) cũng cần được thêm vào. Ví dụ:

```
public class Hello{
    public String hello(String name){
        return "Hello " + name;
    }
}
```

Còn đối với Ruby, một lớp được bắt đầu bằng từ khóa class và một phương thức được bắt đầu bằng từ khóa def, kết thúc của một lớp hay phương thức là từ khóa end. Tên của Lớp cũng bắt buộc phải viết in hoa, do Ruby là dynamically typed nên không cần phải có kiểu trả về của phương thức. Ví dụ:

```
class Hello
def hello(name)
    return "Hello " + name
    end
end
```

Việc khởi tạo đối tượng ở Java và Ruby cũng có sự khác nhau:

- Ruby: đối tượng được khởi tạo không có kiểu dữ liệu, khởi tạo bằng từ khóa new, tham số truyền vào phương thức có thể có hoặc không có ngoặc.

```
    helloObj = Hello.new
    helloObj.hello("Minh")
    helloObj.hello "Minh"

```

- Java: đối tượng khởi tạo phải có kiểu dữ liệu của Lớp, dùng từ khóa new để khởi tạo, tham số truyền vào phương thức phải có ngoặc

```
    Hello helloObj=new Hello();
    helloObj.hello("Minh");

```

1.2.2 Constructor

Hàm khởi tạo (Constructor) của Java trùng tên với tên của lớp, giúp khởi tạo giá trị cho những thuộc tính của lớp:

```
public class Hello{
String msg;
public Hello(String helloMsg){
    msg=hellomsg;
    }
}
```

Các lớp của Ruby được cung cấp một hàm khởi tạo có cùng tên là initialize cho việc khởi tạo đối tượng:

```
class Hello
@msg
def initialize(helloMsg)
    @dmsg=helloMsg
    end
end
```

1.2.3 Destructor

Cả Java và Ruby đều có garbage collection giúp dọn dẹp và tránh rò rỉ bộ nhớ sau khi chương trình kết thúc, chính vì vậy, khác với C++, cả Java và Ruby đều không có Destructor như C++

1.3 Tính đóng gói (Encapsulation)

Về chỉ định truy cập (access modifier) cả Java và Ruby đều có những chỉ định truy cập là **public**, **protected** và **private**. Public cho phép các thuộc tính và phương thức của đối tượng có thể được thay đổi và sử dụng tại bất kỳ đâu; Protected chỉ cho phép các phương thước thuộc Lớp dẫn xuất (subclass) thay đổi và sử dụngp; Private chỉ cho pháp các phương thức bên trong Lớp thay đổi và sử dụng. Tuy nhiên bên cạnh đó Java còn có thêm một chỉ định truy cập nữa là **default** (không chỉ định public, private hay protected) cho pháp các thuộc tính và phương thức có thể được thay đổi và sử dụng ở trong cùng một package. Còn đối với Ruby, do các biến cũng là các đối tượng, vậy nên chỉ định truy cập của các biến đối tượng đó mặc định sẽ là private. Ví dụ:

- Ruby:

```
class Test
    #public method without using public keyword
     def t1
         puts "t1 is called"
     end
    #using public keyword
    public
     def t2
         puts "t2 is called"
     end
     def t3
         puts "t3 is called"
         t1      # calling t1 method
         self.t1 # calling t1 method with seft keyword
     end
```

```
    #using private keyword
    private
    def t4
        puts "t4 is called"
    end
    #using protected keyword
    protected
    def t5
        puts "t5 is called"
    end
end

class Subtest < Test
    # public method of Sudo class
    def st1
        puts "subclass"
        t5   #call protected method
    end
end

obj = Test.new
# calling method t1
obj.t1   ->  "t1 is called"
# calling method t2
obj.t2   ->  "t2 is called"
# calling method t3
obj.t3   ->  "t3 is called" , "t1 is called", "t1 is called"
# calling method t4
obj.t4   ->  Error

sub_obj = Subtest.new
#calling subclass method st1 -> calling t5
sub_obj.st1 -> "subclass", "t5 is called"

```

- Tương tự đối với Java

1.4 Tính trừu tượng (Abstraction)

1.4.1 Lớp trừu tượng (Abstract Class)

Lớp trừu tượng là những lớp chưa ít nhất một phương thức trừu tượng, phương thức trừu tượng là những phương thức không được hiện thực ngay tại lớp trừu tượng mà sẽ được hiện thực ở các lớp con kế thừa theo những cách khác nhau. Trong Java có từ khóa **abstract** cho các lớp trừu tượng.

```
abstract class Shape{  #abstract class
    abstract void draw(); #do not implement here
}
class Rectangle extends Shape{
void draw(){ #implememted by subclass
  System.out.println("Ve hinh chu nhat");
```

```
  }
}

class Circle1 extends Shape{
void draw(){ #implememted by subclass
   System.out.println("Ve hinh tron");
}
}
```

1.4.2 Interface

Interface là một giải pháp khác cho tính trừu tượng của lập trình hướng đối tượng. Interface là một lớp chỉ định nghĩa các phương thức mà không hiện thực bất kì phương thức nào, interface chỉ có nhiệm vụ quy định các phương thức mà các lớp con kế thừa interface phải hiện thực. Điều này giúp cho một lớp con có thể kế thừa nhiều interface mà không gặp phải vấn đề kim cương (diamond problem). Trong Java có từ khóa **interface** để định nghĩa các interface, các phương thức trong interface không được phép hiện thực bên trong interface. Đối với Ruby có **module mixin** có tác dụng gần như interface, các module này có thể hiện thực thân các phương thức, tuy nhiên Ruby có cơ chế lựa chọn gọi phương thức để tránh vấn đề kim cương. Chi tiết hơn về interface và module mixin sẽ được đề cập ở phần sau

1.5 Tính kế thừa (Inheritance)

Cả Java và Ruby đều chỉ hỗ trợ đơn kế thừa (single inheritance) đối với Lớp. Java dùng từ khóa extends trong khi Ruby dùng kí hiệu $<$ cho kế thừa:

- Ruby:

```
        class Animal
        //class constructor
            def initialize
                puts "This is Superclass"
            end
        //method for class
            def method
                puts "This is class"
            end
        end
        class Dog < Animal
        //subclass constructor
            def initialize
                puts "This is Subclass"
            end
        end

```

- Java:

```
        class Employee {
            int salary = 60000;
            }
```

```
4
5         // Inherited or Sub Class
6         class Engineer extends Employee {
7             int benefits = 10000;
8             }
9
```

Vấn đề của lập trình hướng đối tượng trong kế thừa có thể nói đến vấn đề kim cương (diamond problem) với khả năng đa kế thừa: một lớp có thể được kế thừa từ nhiều lớp cha, khi đó nếu những lớp cha có những phương thức có cùng nguyên mẫu (prototype) những lại có cách hiện thực khác nhau, lớp con sẽ không biết phải thực hiện phương thức của lớp cha nào. Chính vì vậy một số ngôn ngữ lập trình như Java, Ruby,... chỉ cho phép đơn kế thừa.

Tuy nhiên trong nhiều trường hợp, khi muốn thuận tiện trong việc kết hợp thuộc tính của nhiều đối tượng khác nhau, việc đơn kế thừa sẽ tạo ra sự phức tạp trong mối quan hệ giữa các lớp. Chính vì vậy, Java có thêm **interface** còn Ruby có thêm **module mixins**. Về cơ bản, interface cho phép quy định những phương thức có thể có của một lớp kế thừa interface đó, các interface sẽ chỉ quy định những phương thức nào có mà không hiện thực bất kỳ phương thức nào, do đó sẽ tránh được vấn đề kim cương ở trên. Trong Java, một interface được định nghĩa bằng từ khóa **interface** và được kế thừa bằng từ khóa **implements**, một lớp có thể kế thừa nhiều interface. Trong Ruby, module mixin chứa các phương thức, khi một lớp muốn sử dụng phương thức này thì sử dụng từ khóa **include** hoặc **extends** để thêm vào trong lớp và sử dụng, một lớp có thể có nhiều module mixin, nếu trường hợp có nhiều phương thức được hiện thực giống nhau trong các module, phương thức của module được include sau cùng sẽ được ưu tiên. Ví dụ:

- Ruby:

```
1         module Car
2             def a1
3                 puts 'Car purchased'
4             end
5         end
6         module Bike
7             def a2
8                 puts 'Bike purchased'
9             end
10        end
11
12        class Garage
13            include Car
14            include Bike
15            def display
16                puts 'Two vehicles have been purchased.'
17            end
18        end
19
20# Creating object
21object = Garage.new
22# Calling methods
23object.display
24object.a1
25object.a2
```

```

Output:
Two vehicles have been purchased.
Car purchased
Bike purchased

```

- Java:

```
interface AnimalEat {
   void eat();
}
interface AnimalTravel {
   void travel();
}
class Animal implements AnimalEat, AnimalTravel {
   public void eat() {
      System.out.println("Animal is eating");
   }
   public void travel() {
      System.out.println("Animal is travelling");
   }
}
public class Demo {
   public static void main(String args[]) {
      Animal a = new Animal();
      a.eat();
      a.travel();
   }
}

Output:
Animal is eating
Animal is travelling

```

1.6 Tính đa hình (Polymorphism)

1.6.1 Nạp chồng (Overloading)

Java chỉ cho phép nạp chồng các phương thức mà không nạp chồng được các toán tử. Ngược lại Ruby chỉ cho phép nạp chồng các toán tử mà không cho phép nạp chồng các phương thức, nếu một phương thức bị nạp chồng, phương thức đó sẽ không thể gọi được nữa. Ví dụ:

- Operator overloading in Ruby:

```
class Box
   def initialize(w,h)     # Initialize the width and height
      @width,@height = w, h
   end

   def +(other)       # Define + to do vector addition
```

```
7       Box.new(@width + other.width, @height + other.height)
8     end
9
10    def -@           # Define unary minus to negate width and height
11       Box.new(-@width, -@height)
12    end
13
14    def *(scalar)           # To perform scalar multiplication
15       Box.new(@width*scalar, @height*scalar)
16    end
17end
18
```

- Function overloading in Ruby:

```
1class OverloadingTest
2    def self.product(a,b)
3        puts(a*b)
4    end
5    def self.product(a,b,c) #Overloading
6        puts(a*b + c)
7    end
8
9end
10Test.sum(1,2) -> Error #Because first method with 2 arguments cannot be accessed
11Test.sum(1,2,3) -> 5 #Result of overloaded method
12
```

- Function overloading in Java:

```
1class OverloadingTest {
2    void sum(int a, int b) {
3        System.out.println(a + b);
4    }
5
6    void sum(int a, int b, int c) { #Overloading
7        System.out.println(a + b + c);
8    }
9
10   public static void main(String args[]) {
11       OverloadingTest obj = new OverloadingTest();
12       obj.sum(20, 20); -> 40
13       obj.sum(20, 20, 20); -> 60
14       #Both method with 2 and 3 arguments can return correct result
15   }
16}
17
```

1.6.2 Ghi đè (Overriding)

Cả Java và Ruby đều cho phép ghi đè phương thức với điều kiện hai lớp phải có mối quan hệ cha con với nhau và prototype phải giống nhau giữa phương thức ghi đè và bị ghi đè. Ví dụ:

- Ruby:

```
class Box
   def initialize(w,h)
      @width, @height = w, h
   end
   # instance method
   def getArea
      @width * @height
   end
end

class BigBox < Box
   # Overriding getArea method
   def getArea
      @area = @width * @height
      puts "Big box area is : #@area"
   end
end

```

- Java:

```
class Vehicle {
    void run() {
        System.out.println("Vehicle is running");
    }
}

class Bike extends Vehicle {
    void run() { #Overriding run method
        System.out.println("Bike is running safely");
    }

    public static void main(String args[]) {
        Bike obj = new Bike();
        obj.run(); #call run method of Bike class
    }
}

```

2 So sánh lập trình hàm (Funcitonal Programming) giữa Haskell và SML

Lập trình hàm nhắc đến những ngôn ngữ lập trình sử dụng hàm là công cụ chính trong việc lập trình, xem các chương trình là sự tính toán của các hàm, nhằm hạn chế sự thay đổi trạng

thái và các dữ liệu biến đổi. Lập trình hàm thiên về việc sử dụng các hàm số, thay vì kiểu lập trình mệnh lệnh tập trung vào sự thay đổi trạng thái. Việc sử dụng các hàm sẽ chỉ cho ra một giá trị đầu ra ứng với một giá trị đầu vào trong bất kể trường hợp nào, nhấn mạnh sự bất biến của các giá trị. Nhắc đến lập trình hàm ta đề cập đến một số khái niệm quan trọng:

- Pure Function: tính thuần hàm đảm bảo rằng hàm không hề tác động đến những tham số đầu vào, mà chỉ sử dụng giá trị của chúng để trả về kết quả, tính thuần hàm đảm bảo sẽ không có tác dụng phụ làm thay đổi giá trị của các biến nằm ngoài các hàm đó. Đây cũng là một tính chất quan trọng của hàm
- Immutability : sự bất biến, khi một giá trị hoặc một đối tượng được khởi tạo, giá trị của nó sẽ không thể bị thay đổi, việc sử dụng các giá trị và đối tượng là vẫn thực hiện được những việc thay đổi là không thể thực hiện.
- Higher-Order Functions: hàm bậc cao là những hàm có thể nhận hàm khác làm tham số, hoặc trả về hàm, hoặc cả 2.
- Function Composition: hàm hợp là xây dựng hàm dựa trên việc kết hợp giữa 2 hay nhiều hàm bằng việc truyền đầu ra của hàm này vào đầu vào của hàm khác theo thứ tự.
- Recursion: do biến và vòng lặp là các khái niệm gần như không tồn tại trong lập trình hàm, vậy nên để hiện thực vòng lặp trong lập trình hàm đòi hỏi phải sử dụng đệ quy.
- Tacit Programming: là kiểu lập trình hàm trong đó các định nghĩa hàm không xác định các đối số, thay vào đó chỉ định nghĩa các chức năng khác trong số đó là các tổ hợp thao tác các đối số.
- Currying function: là cách viết hàm chỉ nhận vào 1 đối số và trả về một hàm con, hàm này sẽ tiếp tục nhận các đối số tiếp theo.
- Closure:bao đóng của hàm là một scope của một hàm mà sẽ tồn tại chừng nào còn có tham chiếu đến hàm đó.
- Referential Transparency: tham chiếu minh bạch

2.1 Pure Function

Haskell là ngôn ngữ lập trình hàm thuần khi tất cả các hàm đều không gặp phải trường hợp side-effects khi các giá trị bị thay đổi sau khi gọi hàm. Hàm của Haskell gồm hai phần: phần khai báo và phần định nghĩa. Ví dụ:

```
1  add :: Int->Int->Int  #declaration
2  add x y = x + y       #definition
```

SML không được coi là ngôn ngữ lập trình hàm thuần vì có một số hàm có khả năng thay đổi trạng thái của các giá trị nằm ngoài hàm đó. Hàm của SML được định nghĩa với từ khóa **fun**. Ví dụ:

```
1  fun factorial n =
2      if n = 0 then 1 else n * factorial (n - 1)
```

2.2 Immutability

Đối với Haskell, một khi biến đã được gán giá trị thì giá trị của nó sẽ không thể bị thay đổi. Ví dụ:

```
>> let a = [1,2,3]
>> reverse a
[3,2,1]
>> a
[1,2,3]
```

2.3 Higher-Order Functions

Hàm bậc cao được hỗ trợ bởi cả Haskell và SML.

- Haskell:

```
        map f (x,y) = (f x, f y)

```

- SML:

```
        fun map f (x, y) = (f x, f y)

```

2.4 Currying Function

Cả 2 ngôn ngữ lập trình hàm Haskell và SML đều hỗ trợ Currying Function, như ví dụ dưới đây, hàm pow sẽ nhận vào một tham số là x, sau đó sẽ trả về một hàm con, hàm con này sẽ nhận tiếp tham số thứ 2 là y sau đó thực hiện các thao tác phù hợp. Currying Function được sử dụng rất nhiều trong các ngôn ngữ lập trình hàm do giúp xử lý dễ dàng hơn đối với các hàm nhiều tham số.

```
    #Haskell
    pow :: Int->Int->Int
    pow _ 0 = 1
    pow x y = x * pow x (y-1)
    #SML
    fun pow x 0 = 1
    | pow x y = x * pow x (y - 1)
```

2.5 Function Composition

Trong Haskell, ký hiệu cho hàm hợp là dấu "." trong khi kí hiệu cho hàm hợp trong SML là "o". Ví dụ:

```
    #Haskell
    twice f = f.f
    #SML
```

```
4    fun twice f = f o f
```

2.6 Lambda Expression

Biểu thức Lambda hay hàm Lambda là khái niệm hàm chỉ những hàm dùng 1 lần nên có thể định nghĩa ngắn gọn. Đối với Haskell, biểu thức Lambda được định nghĩa bởi ký hiệu \, trong khi SML sử dụng từ khóa **fn** cùng với => để thể hiện biểu thức Lambda. Ví dụ:

- Haskell:

```
1        addOne = \x -> x + 1
2
```

- SML:

```
1        addOne = fn x => x + 1
2
```

2.7 Pattern Matching

Trong cả SML và Haskell, Pattern Matching là một tính năng quan trọng giúp hàm kiểm tra các giá trị của đối số và áp dụng các hành động khác nhau tùy thuộc vào giá trị vừa kiểm tra. Pattern Matching được sử dụng nhiều trong những hàm đệ quy vì giúp đơn giản hóa cú pháp câu cũng như tăng tính đọc hiểu. Ví dụ:

- Haskell:

```
1        factorial :: Int->Int
2        factorial 0 = 1
3        factorial n = n * factorial (n-1)
4
```

- SML:

```
1        fun factorial 0 = 1
2        |   factorial n = n * factorial (n-1)
3
```

Trong ví dụ ở cả 2 ngôn ngữ, ta đều thấy hàm factorial sẽ kiểm tra giá trị đầu vào, nếu giá trị đầu vào là 0 thì hàm sẽ thực hiện trả về kết quả là 1, nếu giá trị đầu vào là một số khác thì hàm sẽ thực hiện các bước đệ quy tiếp theo để trả về kết quả phép tính giai thừa

3 New Programming Approaches: Data Wrangling & Smart contract

3.1 Data Wrangling

Data wrangling là quá trình chuyển đổi dữ liệu từ dạng "thô" sang dạng sẵn sàng để phân tích. Trạng thái sẵn sàng để phân tích tùy thuộc vào mục đích cũng như các thức phân tích dữ liệu.

Thông thường, khi một tập dữ liệu được thu thập, rất hiếm khi nào dữ liệu thô như vậy là đủ để dễ dàng truy cập và phân tích. Đây là một bước quan trọng trong tiền xử lý dữ liệu (data preprocessing) giúp cho dữ liệu trở nên dễ dàng tiếp cận hơn. Data wrangling bao gồm những công việc như nhập dữ liệu, làm sạch dữ liệu, cấu trúc dữ liệu, xử lý chuỗi, phân tích cú pháp HTML, xử lý ngày và giờ, xử lý dữ liệu bị thiếu và khai thác văn bản.

Nhìn chung, Data Wrangling gồm các bước như sau:

1. Discovery: tìm hiểu về dữ liệu giúp tạo ra cái nhìn tổng quan về dữ liệu, tạo ra những ý tưởng ban đầu về cách tiếp cận với dữ liệu. Việc này giống như việc trước khi nấu ăn ta phải nhìn một vòng vào trong tủ lạnh và quanh bếp xem có những nguyên liệu gì, những dụng cụ bếp nào có thể được sử dụng. Quan sát và tìm hiểu về dữ liệu ban đầu sẽ giúp xác định được xu hướng của dữ liệu, những đặc trưng cơ bản của dữ liệu, những vấn đề trong dữ liệu cần giải quyết,...

2. Structuring: cấu trúc dữ liệu, dữ liệu ở dạng thô thường có cấu trúc chưa hoàn chỉnh hoặc khó xử lý, việc cấu trúc lại dữ liệu cho phù hợp với việc phân tích là cần thiết.

3. Cleaning: làm sạch dữ liệu là loại bỏ những giá trị bị lỗi bởi nhiều lí do (cách thức thu thập, sai sót trong quá trình nhập dữ liệu,...) có thể tác động xấu đến việc phân tích dữ liệu, những giá trị này thường không có nhiều ý nghĩa và cần phải loại bỏ để có được kết quả phân tích khách quan nhất. Bên cạnh đó việc làm sạch dữ liệu cũng có thể kể đến chuẩn hóa dữ liệu, lọc bỏ các dữ liệu ngoại lai, với mục đích loại bỏ nhiều nhất có thể những yếu tố làm cho kết quả phân tích bị ảnh hưởng. *Data Cleaning và Data Wrangling thường bị nhầm lẫn do sự tương đồng nhất định, tuy nhiên Data Cleaning chỉ là loại bỏ những giá trị không chính xác hoặc bị lỗi, trong khi Data Wrangling là quá trình các bước để chuyển từ dữ liệu thô sang dạng có thể sử dụng để phân tích được, trong đó có bước Data Cleaning

4. Enriching: làm giàu dữ liệu là việc xem xét dữ liệu hiện đang có có đủ để phân tích hay không hay ta phải thu thập và xử lý thêm những bộ dữ liệu khác kết hợp để đủ điều kiện phân tích. Để xác định được khả năng phân tích của bộ dữ liệu hiện có đòi hỏi phải hiểu rõ khả năng của bộ dữ liệu hiện có cũng như những yếu tố dữ liệu còn thiếu.

5. Validating: xác thực dữ liệu bao gồm việc xác minh về độ nhất quán của tập dữ liệu cũng như chất lượng và độ tin cập của dữ liệu, công việc xác thực dữ liệu sẽ giúp xác định tập dữ liệu đã sẵn sàng cho việc phân tích chưa hay cần phải xử lý thêm những vấn đề phát sinh. Xác thực dữ liệu là bước cuối cùng cũng như bước tương đối quan trọng trước khi tập dữ liệu được đưa vào phân tích. Chính vì độ phức tạp và quan trọng như vậy, bước này thường được hỗ trợ bởi các phần mềm và chương trình tự động.

6. Publishing: công bố tập dữ liệu là bước chia sẻ tập dữ liệu đã sẵn sàng để phân tích cho tổ chức hoặc đội nhóm cho việc phân tích. Bước này chỉ được thực hiện khi tập dữ liệu đã trải qua đủ các khâu trong Data Wrangling và sẵn sàng để thực hiện phân tích.

Data Wrangling là việc làm quan trọng của các chuyên gia phân tích dữ liệu khi có thống kê cho rằng 80% thời gian của họ dành cho Data Wrangling trong khi thời gian xây dựng mô hình và phân tích chỉ là 20%

Người sắp xếp dữ liệu (Data Wranglers) sử dụng nhiều công cụ và ngôn ngữ khác nhau để thực hiện các quy trình Data Wrangling. R và Python là các ngôn ngữ thống kê phổ biến hỗ trợ tốt cho Data Wrangling. Các ngôn ngữ phổ biến khác để thực hiện bao gồm SQL, PHP và Scala. Các thư viện hữu ích của Python bao gồm Numpy, với Pandas, Matplotlib, Plotly và Theano. Các package trợ giúp R bao gồm Dplyr, Purrr, Splitstackshape, JSOnline và Magrittr. Các công cụ khác bao gồm bảng tính excel, OpenRefine, Tabula và CSVKit.

Với sự gia tăng trí tuệ nhân tạo trong khoa học dữ liệu, điều quan trọng tạo nên sự khác biết của các nhà khoa học dữ liệu là biết cách sắp xếp dữ liệu và có khả năng áp đặt và giám sát thủ công các biện pháp kiểm tra và cân bằng nghiêm ngặt đối với quy trình sắp xếp dữ liệu tự động.

3.2 Smart Contract

Smart Contract (Hợp đồng thông minh) là một giao thức giao dịch dựa trên công nghệ Blockchain, được chạy khi các điều kiện xác định được đáp ứng. Mục đích của hợp đồng này là thực hiện các điều khoản của hợp đồng mà không cần thông qua bên thứ ba. Nó tự động thực hiện, ghi nhớ lại hành động pháp lý của các bên giúp việc truy dấu dễ dàng hơn. Smart Contract được dùng để tự động hóa việc thực hiện một thỏa thuận. Đồng thời nó cũng được sử dụng để kích hoạt các hành động tiếp theo khi điều kiện được đáp ứng. Các điều khoản trên hợp đồng thông minh tương đương với một hợp đồng pháp lý sử dụng ngôn ngữ máy tính. Hợp đồng thông minh có thể được viết bằng nhiều ngôn ngữ như Golang, Javascript, C++, Java,...

Có 4 yếu tố quan trọng để hình thành một hợp đồng thông minh:

- **Chủ thể hợp đồng:** Các bên tham gia thực hiện giao kết hợp đồng, trong đó có những bên được cấp quyền truy cập, theo dõi tình hình xử lý và nội dung hợp đồng.
- **Điều khoản hợp đồng:** Các điều khoản quy định ở dạng chuỗi, được lập trình đặc biệt mà các bên tham gia phải đồng ý với các điều này.
- **Chữ ký số:** Các bên tham gia hợp đồng thông minh đồng thuận triển khai thỏa thuận về chữ ký số và phải thực hiện thao tác thông qua chữ ký số.
- **Nền tảng phân quyền:** Bước vào giai đoạn hoàn tất, hợp đồng thông minh cần được tải lên Blockchain. Chuỗi Blockchain tiếp tục phân phối dữ liệu về các node và lưu lại, không thể điều chỉnh.

Hiểu một cách đơn giản, Smart Contract hoạt động tương tự như máy bán nước tự động. Khi bạn bỏ tiền vào và lựa chọn đồ uống, nó sẽ tự động đưa cho bạn món bạn cần. Hay nếu món đó đã hết, máy có thể báo bạn thay đổi lựa chọn hoặc nhận lại tiền. Có thể thấy việc mua hàng từ máy bán nước này hoàn toàn tự động mà không cần đến sự can thiệp của bên thứ ba. Hoạt động dựa trên những câu lệnh điều kiện được mã hóa trên Blockchain, hợp đồng thông minh sẽ thực hiện các hành động khi có đủ các điều kiện cần thiết. Sau khi thực hiện xong giao dịch vừa được hoàn tất sẽ được tự động cập nhật.

Ưu điểm của hợp đồng thông minh:

- Các thủ tục được diễn ra vô cùng nhanh chóng và hiệu quả do được thực hiện một cách tự động ngay khi đủ điều kiện, không cần tốn chi phí và thời gian cho giấy tờ thủ công một cách rườm rà.

- Hợp đồng thông minh được thực hiện mà không cần bên thứ ba tham gia, đồng thời tất cả người tham gia đều có thể giám sát được các giao dịch và thủ tục. Chính vị vậy hợp đồng thông minh có độ tin cậy và minh bạch cao. Bên cạnh đó hợp đồng thông minh sử dụng chữ ký số nên mang lại tính bảo mật và tin cậy cao cho người dùng.
- Giao dịch được mã hóa trên Blockchain nên việc thay đổi thông tin là gần như không thể nên đảm bảo được tính bảo mật.
- Chi phí của các bên trung gian cũng như những chi phí giấy tờ, đi lại, thời gian cũng được tiết kiệm rất nhiều so với kiểu hợp đồng truyền thống.

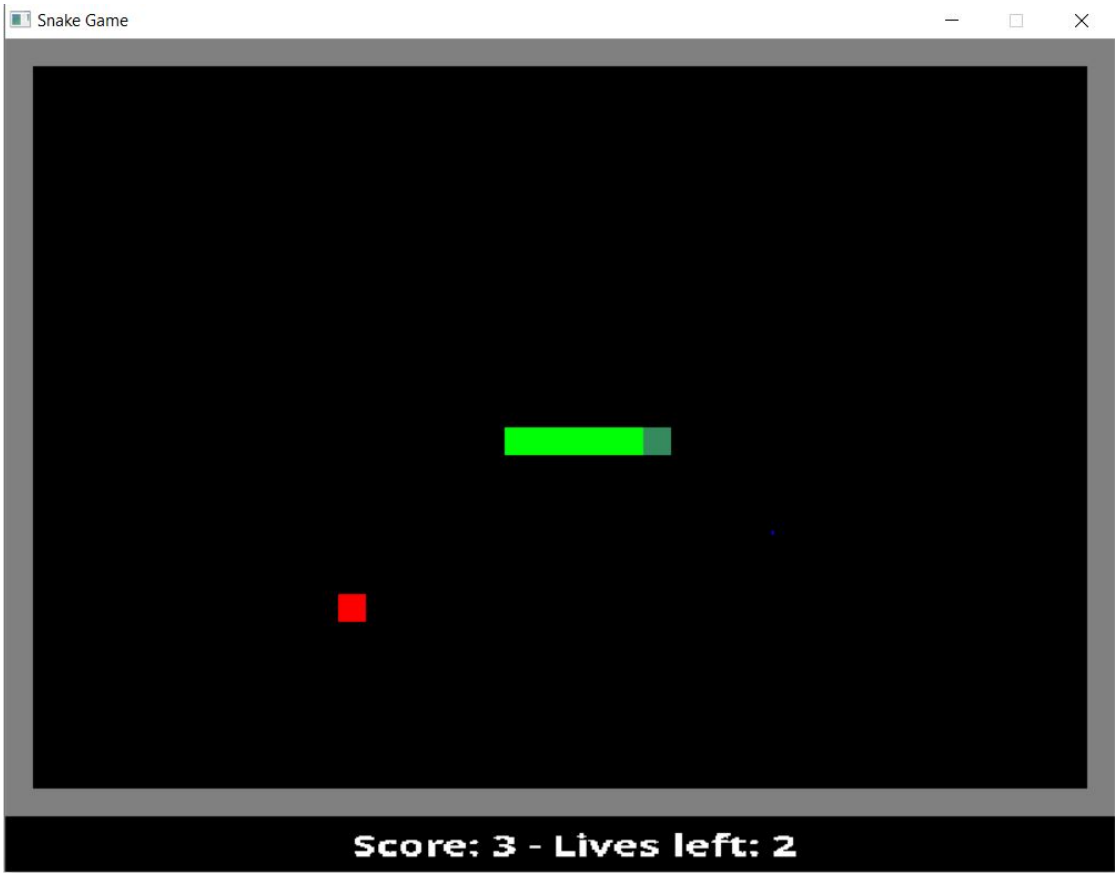
Một nhược điểm duy nhất của hợp đồng thông minh là trở ngại khi các điều khoản của hợp đồng, các giao dịch có sự thay đổi hoặc thỏa thuận lại, việc chỉnh sửa nội dung thông tin hợp đồng của gặp khó khăn do tính bảo mật cao và hạn chế tác động của bên thứ ba.

Ứng dụng của hợp đồng thông minh hiện tại là rất lớn trong thời đại phát triển số:

- Thương mại quốc tế
- Kết nối nhà bán lẻ và nhà cung cấp
- Giao dịch tiền kỹ thuật số
- ...

4 Trò chơi OOP đơn giản sử dụng C++: Rắn săn mồi (Snake Game)

Trò chơi là phiên bản đơn giản của trò chơi tuổi thơ Rắn săn mồi (Snake Game).



4.1 Setup

Trò chơi sử dụng thư viện SDL2 và ngôn ngữ C++, thư viện và source code đều có trong file SnakeGame.zip đính kèm.

Để khởi động trò chơi, giải nén file SnakeGame.zip sau đó chạy file SnakeGame.exe

Để compile lại trò chơi, phải đảm bảo MinGW g++ compiler đã được cài đặt, sử dụng lệnh **g++ *.cpp -I include -L lib -lmingw32 -lSDL2main -lSDL2 -lSDL2_ttf -o SnakeGame** để compile. Sau đó chạy file **SnakeGame.exe**

4.2 Cách chơi

Người chơi có tối đa 3 mạng, mạng sẽ bị giảm đi 1 khi người chơi đâm vào tường (màu xám) hoặc đâm vào chính thân con rắn (màu xanh lá nhạt). Người chơi sử dụng các phím mũi tên lên xuống trái phải để điều khiển con rắn ăn quả màu đỏ, sau 5 lần liên tiếp ăn quả mà không bị mất mạng, quả tiếp theo sinh ra sẽ có màu vàng và sau đó tăng tốc độ di chuyển. Mỗi quả đỏ cho 1 điểm, quả vàng cho 5 điểm. Người chơi tiếp tục chơi cho đến khi nào số mạng bằng 0 hoặc tắt trò chơi.