

IT002 – Object-Oriented Programming

Lập trình hướng đối tượng – ÔN TẬP

Object-Oriented Programming: Revision



Phan The Duy
Information Security Laboratory
University of Information Technology, VNU-HCM, Vietnam

1. Lập trình hướng đối tượng là gì?
2. Các đặc điểm quan trọng của lập trình hướng đối tượng?
3. Lớp đối tượng là gì? Đối tượng là gì?
4. Phân biệt các phạm vi truy xuất private, protected, public?
5. Constructor là gì? Constructor mặc định?
6. Destructor là gì?
7. Kế thừa là gì?
8. Phân biệt các kiểu kế thừa private, protected, public?
9. Đa hình là gì?
10. Lớp trừu tượng là gì? Phương thức thuần ảo là gì?
12. Đa năng hóa toán tử là gì?

- Đặc điểm của OOP:
 - Tính trừu tượng
 - Tính đóng gói
 - Tính kế thừa
 - Tính đa hình
- Tham số mặc nhiên, quá tải hàm trong C++
- Thiết kế lớp, phạm vi truy xuất thuộc tính và khai báo đối tượng
- Hàm khởi tạo (constructor) và hàm hủy (destructor)
- Điều kiện khởi tạo đối tượng tự động

- Tính kế thừa – phạm vi kế thừa
 - Khả năng truy xuất theo chiều ngang
 - Khả năng truy xuất theo chiều dọc
- Tính đa hình
 - Hàm ảo/hàm thuần ảo (virtual function)
 - Đa năng hóa toán tử (operator overloading)

- Là một thực thể phần mềm bao bọc các thuộc tính và phương thức liên quan
- Một đối tượng cụ thể được gọi là một thể hiện (instance) của lớp đó.

- Lớp đối tượng là một thiết kế (blueprint) hay một mẫu ban đầu (prototype) định nghĩa các thuộc tính và các phương thức chung cho tất cả các đối tượng của cùng một loại nào đó.
- Một đối tượng là một thể hiện cụ thể của một lớp.

- Trình bày trạng thái của đối tượng
- Các thuộc tính nắm giữ các giá trị dữ liệu trong một đối tượng, chúng định nghĩa một đối tượng đặc thù.

→ *Tìm danh từ khi thiết kế lớp*

Phương thức (method)



- Thực thi các hoạt động của đối tượng
- Là tác nhân làm thay đổi giá trị các thuộc tính của đối tượng.

→ *Tìm động từ khi thiết kế lớp*



Thông điệp (message)



- Là một lời yêu cầu một hoạt động/hành động (lời gọi tới phương thức)
- Được truyền khi một đối tượng gọi một hay nhiều phương thức của đối tượng khác để yêu cầu thông tin
- Một thông điệp gồm có:
 - Đối tượng nhận thông điệp
 - Tên của phương thức thực hiện
 - Các tham số mà phương thức cần để thực hiện



- **Hàm bạn** là loại hàm **không phải** là hàm thành viên của một lớp nhưng **có thể truy cập vào các thành phần, kể cả private** của lớp đó.
- Một hàm có thể là bạn của nhiều lớp. Lúc đó, nó có quyền truy cập tới tất cả các thuộc tính của các đối tượng trong các lớp này.
- **Lớp bạn (friend class):** Khi một lớp có lớp bạn thì tất cả hàm thành viên của lớp bạn sẽ trở thành hàm bạn của lớp đó.

Thành phần tĩnh (static member)



- **Thành phần dữ liệu tĩnh (static data member):**

- Các thuộc tính trong lớp được khai báo bằng từ khoá **static** được gọi là thành phần dữ liệu tĩnh.
- Các thuộc tính này được cấp phát một vùng nhớ cố định, tồn tại ngay cả khi lớp chưa có một đối tượng nào cả. Dữ liệu tĩnh là thành phần chung cho cả lớp, không của riêng từng đối tượng.

```
class AClass{  
    private:  
        static int x;//Thành phần dữ liệu tĩnh  
        int y;  
};  
void main() {  
    AClass a, b; //Khai báo 2 đối tượng a, b  
    int AClass::x = 10;//Khởi tạo cho x giá trị 10  
}
```



- **Hàm thành phần tĩnh (static member function):**
 - Các hàm trong lớp được khai báo bằng từ khoá static được gọi là hàm thành phần tĩnh.
 - Hàm thành phần tĩnh là chung cho toàn bộ lớp và không lệ thuộc vào một đối tượng cụ thể. Nó tồn tại ngay khi lớp chưa có đối tượng nào được tạo ra.
- Cú pháp định nghĩa hàm thành phần tĩnh:

```
static Kiểu_dữ_liệu Tên_hàm(Các_tham_số){  
    //các lệnh  
}
```

- Lời gọi hàm thành phần tĩnh như sau:

```
Tên_lớp::Tên_hàm_thành_phần_tĩnh(Các_tham_số);
```

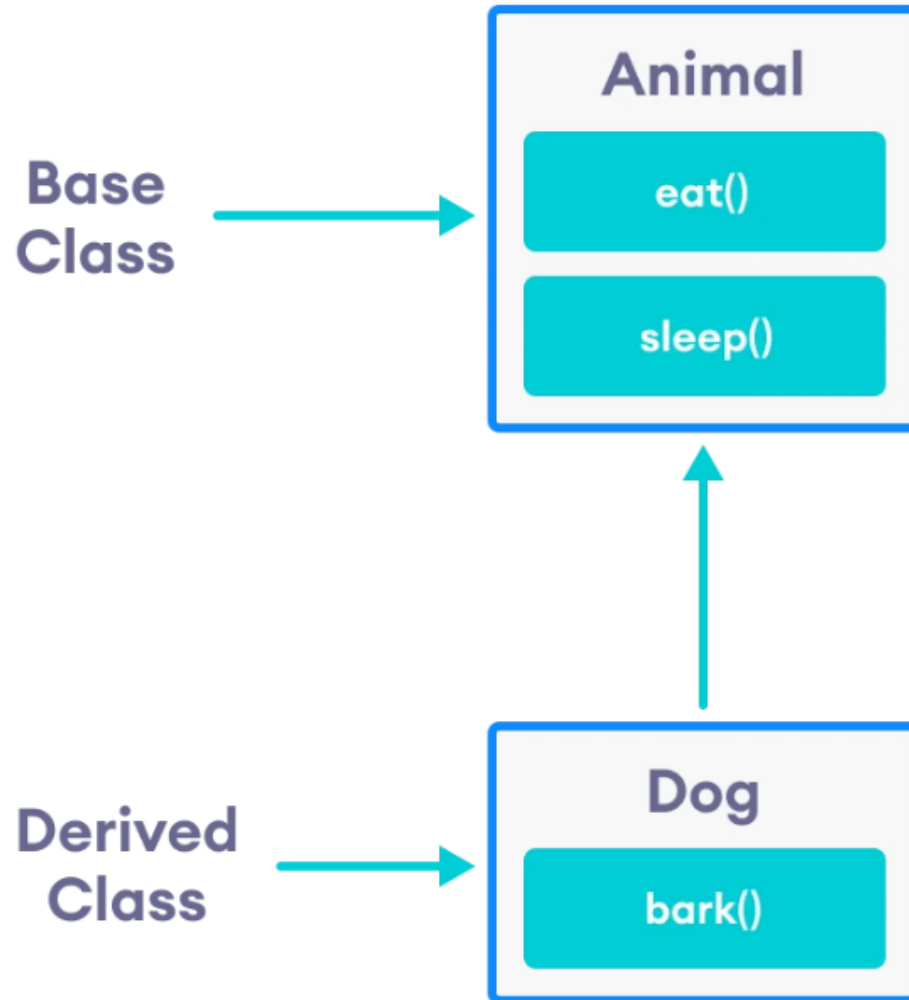
Tính đóng gói (Encapsulation)



- Che giấu chi tiết của đối tượng
- Đảm bảo sự toàn vẹn của đối tượng
- Chỉ có các phương thức nội tại (thành viên) của đối tượng cho phép thay đổi trạng thái của nó.



Tính kế thừa (Inheritance)



Tính kế thừa (Inheritance)



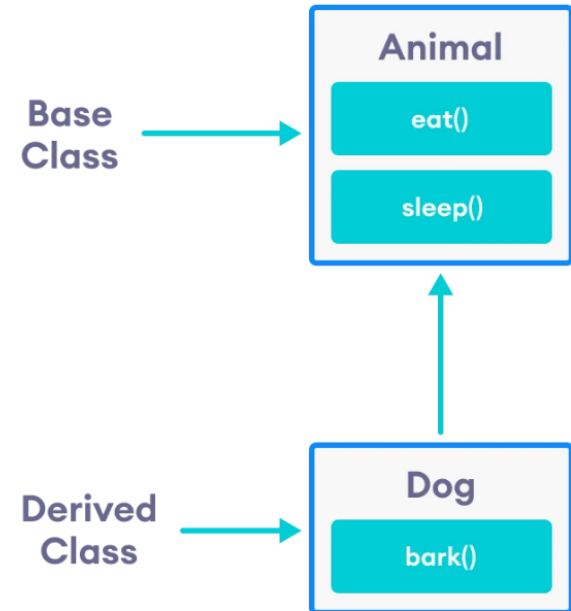
- Các phương thức và thuộc tính được định nghĩa trong một lớp có thể được sử dụng lại bởi lớp khác.
- **Lớp cha/ lớp cơ sở** (superclass/base class): có phương thức, thuộc tính được thừa hưởng bởi lớp khác
- **Lớp con/lớp dẫn xuất** (subclass/derived class): là lớp thừa hưởng một vài đặc tính chung của lớp cha và thêm vào những đặc tính riêng khác.



- Trong lớp dẫn xuất (derived class) có thể định nghĩa lại hàm thành phần của lớp cơ sở mà nó thừa kế được.
 - Như vậy, sẽ có hai phiên bản khác nhau của hàm thành phần trong lớp dẫn xuất.
 - Trong phạm vi lớp dẫn xuất, hàm định nghĩa lại trong lớp dẫn xuất sẽ ghi đè (overriding) lên hàm được định nghĩa trong lớp cơ sở.

- **Các hàm khởi tạo (constructor) của lớp cơ sở không được kế thừa.** Tuy nhiên, một đối tượng của lớp dẫn xuất về thực chất có thể xem là một đối tượng của lớp cơ sở.
 - Vì vậy, việc gọi hàm khởi tạo lớp dẫn xuất để tạo đối tượng của lớp dẫn xuất sẽ kéo theo việc gọi đến một hàm khởi tạo của lớp cơ sở. Hàm khởi tạo của lớp cơ sở được gọi trước rồi đến hàm khởi tạo của lớp dẫn xuất.
- **Hàm hủy (destructor) của lớp cơ sở cũng không được kế thừa.** Hàm hủy của lớp dẫn xuất thực thi trước hàm hủy của lớp cơ sở.

- Đơn kế thừa trong C++ là một lớp dẫn xuất được kế thừa từ một và chỉ một lớp cơ sở.



```
class <Tên_lớp_dẫn_xuất> : <Từ_khóa_dẫn_xuất> <Tên_lớp_cơ_sở>{
private:
    // Khai báo các thuộc tính của lớp_dẫn_xuất
public:
    // Định nghĩa các hàm thành phần của lớp_dẫn_xuất
};
```

Tính đa hình (polymorphism)



- Phương thức cùng tên có thể được thực hiện khác nhau đối với các đối tượng/lớp khác nhau.
- Hai khía cạnh của tính đa hình:
 - Đa hình thời gian chạy (Runtime)
 - Đa hình thời gian biên dịch (Compile-time)



Hàm ảo (virtual function)



```
1  #include <iostream>
2  using namespace std;
3
4  class Base{
5  public:
6      virtual void print(){//virtual function
7          cout<<"Base class";
8      }
9  };
10
11 class Derived : public Base{
12 public:
13     void print(){
14         cout<<"Derived class";
15     }
16 };
17
18
19 void main(){
20     Derived derived1;
21     Base* base1 = &derived1;
22
23     //calls function of Derived class
24     base1->print();
25     system("pause");
26 }
```

```
class Base {
    public:
        virtual void print() {
            // code
        }
};

class Derived : public Base {
    public:
        void print() {
            // code
        }
};

int main() {
    Derived derived1;
    Base* base1 = &derived1;

    base1->print();

    return 0;
}
```

print() of Derived
class is called
because print()
of Base class is
virtual



Hàm ảo (virtual function)



- Hàm ảo (virtual function) là một hàm thành viên trong lớp cơ sở mà lớp dẫn xuất khi kế thừa cần phải định nghĩa lại.
- Hàm ảo được sử dụng trong lớp cơ sở khi cần đảm bảo hàm ảo đó sẽ được định nghĩa lại trong lớp dẫn xuất. Việc này rất cần thiết trong trường hợp con trở có kiểu là lớp cơ sở trở đến đối tượng của lớp dẫn xuất.
- Con trở của lớp cơ sở có thể chứa địa chỉ của đối tượng thuộc lớp dẫn xuất, nhưng ngược lại thì không được.
- Hàm ảo chỉ khác hàm thành phần thông thường khi được gọi từ một con trở. Sử dụng hàm ảo khi muốn con trở đang trở tới đối tượng của lớp nào thì hàm thành phần của lớp đó sẽ được gọi mà không xem xét đến kiểu của con trở.



Hàm thuần ảo (pure virtual function)



- Hàm thuần ảo (pure virtual function) được sử dụng khi:
 - Không cần sử dụng hàm này trong lớp cơ sở, chỉ phục vụ cho lớp dẫn xuất
 - Lớp dẫn xuất bắt buộc phải định nghĩa lại hàm thuần ảo
- Hàm thuần ảo **không có thân hàm** và bắt buộc phải kết thúc với “= 0”.
- Lớp trừu tượng: Một lớp bao gồm hàm thuần ảo được gọi là lớp trừu tượng (abstract class).
- Chúng ta không thể tạo ra các đối tượng của một lớp trừu tượng. Mục đích chính của lớp trừu tượng là để các lớp khác kế thừa lại.



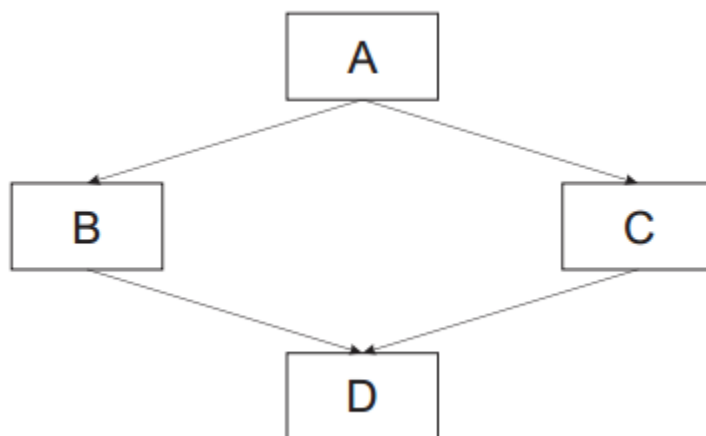
Lớp trừu tượng (abstract class)



- Là một lớp cha cho tất cả các lớp con có cùng bản chất
- Bản chất ở đây được hiểu là kiểu, loại, nhiệm vụ của lớp đó.
- Quan hệ với lớp đối tượng: “A là-một B” (A is a B)
 - A là lớp thừa kế (extend) của B.
 - B là một lớp trừu tượng
- Lớp trừu tượng là một lớp không có thông tin về nội dung thực hiện và bắt buộc các subclass (lớp con) phải định nghĩa các phương thức ở lớp trừu tượng.



Lớp cơ sở ảo (virtual base class)



- Hai lớp B và C kế thừa từ lớp A. Lớp D kế thừa từ cả hai lớp B và C. Như vậy, lớp A được kế thừa hai lần bởi lớp D. Lần thứ nhất được kế thừa thông qua lớp B, lần thứ hai được kế thừa thông qua lớp C.
- Lúc này, nếu đối tượng của lớp D gọi đến một hàm được kế thừa từ lớp A thì sẽ gây ra một sự mơ hồ. Không biết hàm đó được kế thừa gián tiếp từ lớp B hay lớp C.



Lớp cơ sở ảo (virtual base class)



```
1  #include <iostream>
2  using namespace std;
3
4  class A{
5  public:
6      void show(){
7          cout << "Hello from A \n";
8      }
9  };
10
11 class B : public A{
12 };
13
14 class C : public A{
15 };
16
17 class D : public B, public C{
18 };
19
20 void main(){
21     D object;
22     object.show();//error: ambiguous access of 'show'
23     system("pause");
24 }
```



```
class A{  
    //Định nghĩa lớp  
};  
class B : virtual public A{  
    //Định nghĩa lớp  
};  
class C : virtual public A{  
    //Định nghĩa lớp  
};  
class D : public B, public C{  
    //Định nghĩa lớp  
};
```

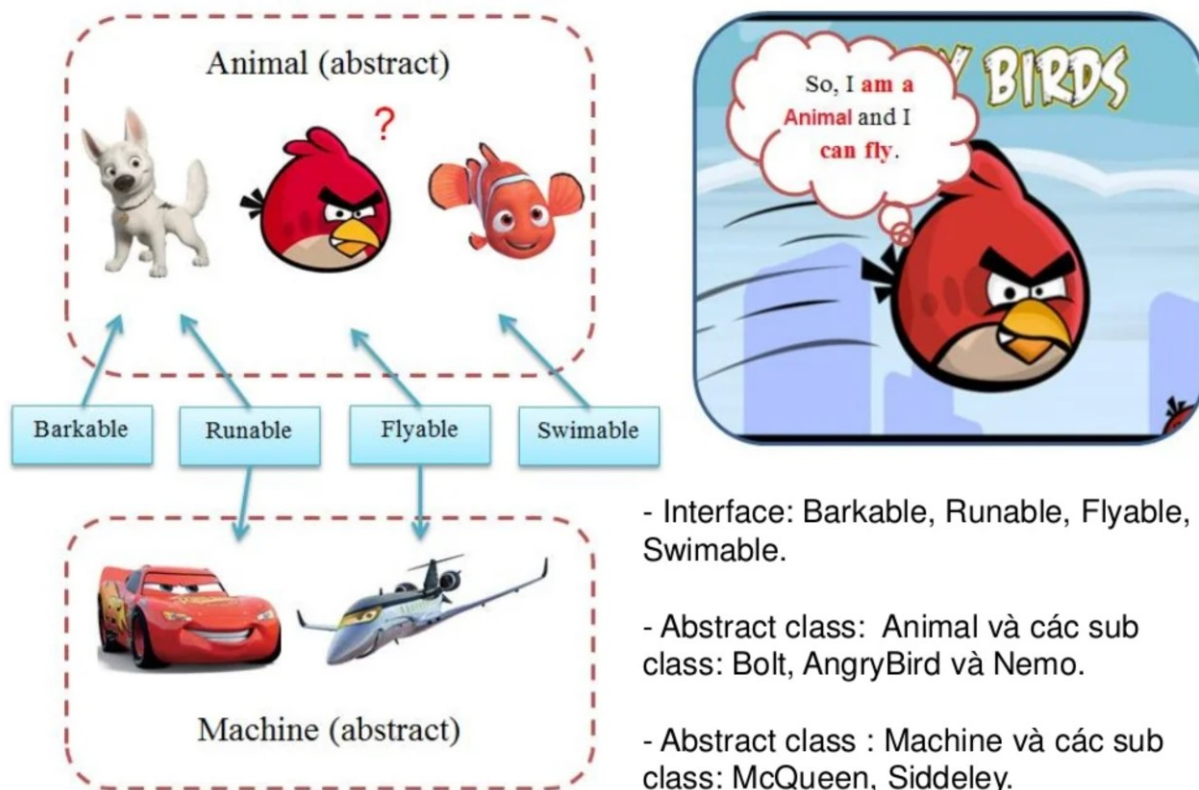
- Để giải quyết tính không rõ ràng này, C++ có một cơ chế mà nhờ đó chỉ có một bản sao của lớp A ở trong lớp D. Đó là sử dụng **lớp cơ sở ảo (virtual base class)**.
- Dùng từ khóa “virtual” để khai báo lớp A là lớp cơ sở ảo trong các lớp B, C.
- Lưu ý: Từ khóa virtual có thể đặt trước hoặc sau từ khóa public, private, protected.

- Là một chức năng mà người lập trình có thể thêm vào bất kỳ lớp nào
- Một chức năng bao gồm 01 hoặc nhiều phương thức
- Quan hệ với lớp: “**A** can do **b**”.
 - A là lớp hiện thực (implement) b.
 - b là một chức năng của A.
 - Ví dụ: Xe tải có thể chở hàng, Xe Buýt có thể chở khách,...
- Trong C++, việc khai báo một interface có nghĩa là chúng ta khai báo một lớp với hàm thuần ảo và một phương thức hủy ảo (virtual destructor).

Giao diện (interface)



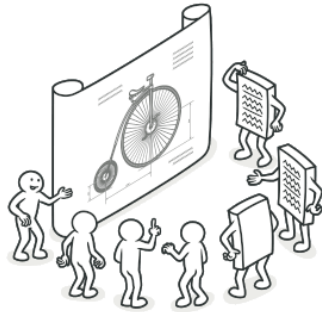
- Có thể hiểu Interface là một bản mô tả hành vi hoặc khả năng của một class mà không đưa ra cách thực hiện cụ thể của class đó như thế nào.



- Nạp chồng toán tử trong C++ là các hàm có tên đặc biệt. Tên hàm gồm từ khóa operator và theo sau là ký hiệu của toán tử đang được định nghĩa.
- Nạp chồng toán tử được dùng để định nghĩa lại các toán tử có sẵn như ++, --, +, -, *, /,... cho kiểu dữ liệu (class) do người lập trình tự định nghĩa.
 - Nhằm tạo ra toán tử cùng tên nhưng thực hiện trên các lớp khác nhau chứ không phải trên các kiểu dữ liệu nguyên thủy.
- Số lượng đối số của hàm đa năng hóa toán tử phụ thuộc vào:
 - Số ngôi của toán tử
 - Vị trí đặt của hàm (hàm thành viên hay hàm toàn cục)

- Mẫu thiết kế - Design Pattern

Mẫu thiết kế: Design Pattern



DESIGN PATTERNS

Design patterns are typical solutions to common problems in software design. Each pattern is like a blueprint that you can customize to solve a particular design problem in your code.

What's a design pattern?



Benefits of patterns

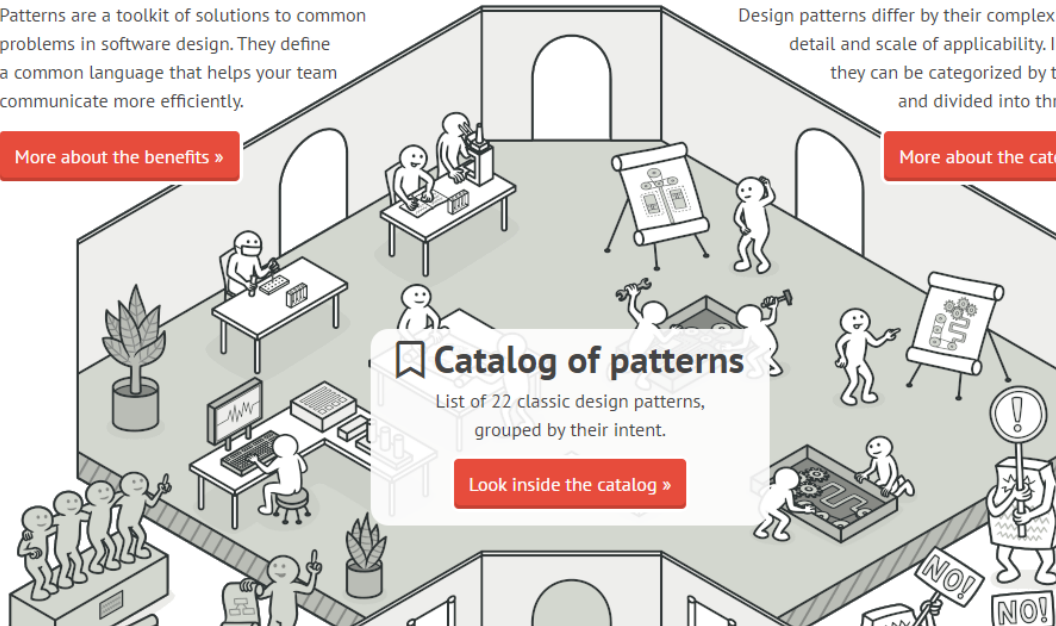
Patterns are a toolkit of solutions to common problems in software design. They define a common language that helps your team communicate more efficiently.

More about the benefits »

Classification

Design patterns differ by their complexity, level of detail and scale of applicability. In addition, they can be categorized by their intent and divided into three groups.

More about the categories »



Catalog of patterns

List of 22 classic design patterns, grouped by their intent.

Look inside the catalog »

<https://refactoring.guru/design-patterns>




Mô tả ngữ cảnh: **Quản lý danh sách tài khoản và lịch sử đăng nhập**

Bạn được yêu cầu xây dựng một ứng dụng đơn giản để quản lý danh sách các tài khoản và lịch sử đăng nhập của chúng. Ứng dụng này sẽ giúp bạn áp dụng kiến thức về OOP trong ngữ cảnh an toàn thông tin.

- Mỗi tài khoản được đặc trưng bởi một tên người dùng (username) và mã băm của mật khẩu (passwordHashed) và quyền hạn (permission) tương ứng để chỉ định quyền truy cập của người dùng (vd: user, guest,...)
- Hệ thống quản lý 2 loại tài khoản chính: RegularAccount và AdminAccount. Trong đó AdminAccount có thể thiết lập lại mật khẩu (resetPassword) cho các tài khoản khác.

- Nhập vào các tài khoản tạo ra trên hệ thống
- Quản lý (in/thống kê) các tài khoản đã đăng nhập trong thời hạn 01 tháng gần nhất. Mỗi tài khoản sẽ được liên kết với thông tin về thời điểm đăng nhập cuối cùng.
- Thiết kế và cài đặt các hàm thực hiện các hoạt động như thay đổi mật khẩu và đặt lại mật khẩu cho các tài khoản trong danh sách dưới quyền AdminAccount.
- Hiển thị thông tin của mỗi tài khoản, bao gồm cả lịch sử đăng nhập.



**Chúc các bạn có kỳ thi
cuối kỳ với kết quả như ý.**

A low-angle, upward-looking photograph of a modern, multi-story building with a light blue facade and numerous windows. The building is partially obscured by the dark green, silhouetted branches of a tree in the foreground. The sky is a pale, overcast blue. A semi-transparent dark blue rectangular box is overlaid on the right side of the image, containing white text.

UIT InSecLab

Phòng Thí nghiệm

An toàn thông tin

Email: inseclab@uit.edu.vn

Website: <https://inseclab.uit.edu.vn/>

Fanpage: <https://www.facebook.com/inseclab>