**OOP concepts**

● **Object**: Là một đối tượng thực thể rời rạc tại một thời điểm cụ thể. Có danh tính đóng gói thuộc tính và hành vi(behavior) riêng. Là thể hiện của một lớp, trong một chương trình có thể có 0 hoặc nhiều object. Lớp là bản thiết kế và là cơ sở để xây dựng một đối tượng. Được tạo từ các lướp

● **Link**: Một đối tượng liên kết với các đối tượng khác qua link. Các đối tượng và 1 liên kết giữa chúng là một sự thật.

● **Class**: đại diện cho một thực thể ở thế giới thực, , là tập hợp tất cả các đối tượng(Object) của cùng một loại nào đó có chung thuộc tính và các phương thức, cung cấp giao diện chung và triển khai riêng, ẩn dữ liệu và thực toán đối với người dùng, có ranh giới rõ ràng, có ý nghĩa, liên quan chặt chẽ. Một lớp có thể được coi là kiểu dữ liệu cấp cao. Lớp xác định thuộc tính và hành vi của tất cả các object thuộc lớp sẽ sở hữu.

● **Attributes**: là dữ liệu trình bày về đặc điểm nhận dạng của một lớp. Một attribute là sự mô tả của một thẻ tên cho một loại cụ thể. Chỉ định các giá trị mà một đối tượng nắm giữ.

● **Operation**: Là hành động mà một đối tượng có thể làm (chưa được định nghĩa sẽ làm như thế nào).

● **Method**: là định nghĩa thực hiện các Operation tác động lên các Attribute.

● **Association**: Một Class liên kết với các Class khác qua Association. Đại diện cho các link có thể có giữa các Object trong Class.

● **Software reuse**: sử dụng lại các sản phẩm công việc trong một hệ thống hoặc nhiều hệ thống. Src code, thành phần thiết kế, ý tưởng,..

● **Benefits**: giảm thời gian giao hàng, giảm chi phí để phát triển, chất lượng cao hơn, tăng cơ hội thành công, giảm chi phí bảo trì-phát triển, cải thiện hiệu suất, khả năng mở rộng, bảo mật, tính di động, khả năng bảo trì, sử dụng nguồn nhân lực hquả.

● **Tái sử dụng các phương pháp tiếp cận:** Function,procedure tái sử dụng, Cách tiếp cận hướng đối tượng(Lớp, đối tượng, giao diện, kế thưa,.., thư viện lớp, mẫu thiết kế), Dựa trên thành phần, Định hướng dịch vụ, Dòng sản phẩm phần mềm, gói hệ thống kế thừa.

|  |  |  |
| --- | --- | --- |
| **Khía cạnh** | **Lập trình thủ tục** | **Lập trình hướng đối tượng** |
| **Định nghĩa** | Tập trung vào việc chia nhỏ các tác vụ thành hàm | Tập trung vào các đối tượng và phương thức |
| **Mục đích** | Đơn giản hóa việc xử lý dữ liệu | Tái sử dụng mã nguồn, quản lý dự án lớn hơn |
| **Cấu trúc** | Chia thành các hàm riêng lẻ, các dữ liệu tách biệt khỏi hàm, dữ liệu được đặt vào cấu trúc riêng biệt và được xử lý bới chức năng thủ tục này | Sử dụng lớp và đối tượng để tổ chức mã nguồn. Các dữ liệu và hành vi được đóng gòi trong cùng một Class. |
| **Đóng gói và che giấu thông tin** | Không hỗ trợ tốt, sử dụng biến toàn cục, khó kiểm soát-> khó gỡ lỗi. | Hỗ trợ tốt, giúp giảm thiểu sự phụ thuộc. Các chức năng sủ dụng trực tiếp các data trong cùng class. |
| **Kế thừa và đa hình** | Không hỗ trợ | Hỗ trợ kế thừa và đa hình để tái sử dụng và linh hoạt |
| **Quản lý tài nguyên** | Phức tạp hơn do sự phụ thuộc lẫn nhau giữa các hàm | Dễ dàng hơn vì mỗi đối tượng quản lý tài nguyên của mình |
| **Mở rộng và bảo trì** | Khó khăn hơn trong việc mở rộng và bảo trì mã nguồn | Dễ dàng hơn vì mã nguồn được tổ chức thành các đối tượng |

● **Tính đóng gói (Encapsulation):** Class cung cấp cơ chế đóng gói. Là việc kết hợp một bộ các dữ liệu/thuộc tính (data/attribute) liên quan đến nhau cùng với một bộ các hàm/phương thức (functions/methods) hoạt động trên các dữ liệu đó, “ đóng gói” tất cả vào trong một thực thể duy nhất và trong class. Các Object khi được khởi tạo từ Class sẽ có sẵn các attribute và method của Class đó.

● **Che dấu dữ liệu (Data Hiding):** Class cung cấp cơ chế chỉ định quyền truy cập. là việc một số dữ liệu/thuộc tính (data/attribute) và hàm/phương thức (functions/methods) được class che giấu đi (ở dạng private,protected) để đảm bảo rằng các dữ liệu đó sẽ được truy cập và sử dụng đúng mục đích, đúng cách thông qua các hàm/phương thức (functions/methods) ở dạng public mà class cung cấp. Bạn không thể truy cập đến các private data hoặc gọi đến private methods của class từ bên ngoài hoặc class con.

● **Constructor**: thao tác tạo ra một thể hiện mới trên Object mới tạo của một lớp.

● **Mutator**: thao tác thay đổi trạng thái của các dữ liệu của một đối tượng.

● **Observer**: một thao tác báo cáo trạng thái của các dữ liệu (còn gọi là Accessors, Getters)

● **Iterator**: một thao tác cho phép xử lý tuần tự tất cả các thành phần của cấu trúc dữ liệu.

● **Phạm vi truy vập**: Variable(Khối biến được khai báo), Function(Toàn bộ chương trình), Struct member(Khối biến Struct khai báo). Private(Chỉ trong class), Protected(Trong class và class con), Public(Trong và ngoài class).

**Constructor and Destructor**

● **Constructor(Khởi tạo):** Cùng tên class, Không có kiểu trả về, Được gọi tự động khi object được tạo, Được gọi khi tạo một đối tượng, Một lớp có nhiều hàm tạo, Có thể over loading.

● **Ví dụ:**

class Product {

private: String m\_Name; String m\_Description; float m\_Weight; float m\_Price;

public: Product(); //default constructor

Product(string Name); };

● **Lưu ý: Nếu không có constructor nào đc tạo,mang tính chất constructor, compiler sẽ gọi Default Constructor**: không có biến đầu vào, Gọi các hàm tạo mặc định khác của thành viên dữ liệu nếu chúng là Object. Khi đã tạo ít nhất một Constructor thì default constructor không được gọi.

● **Copy Constructor**: Một cách khởi Object từ các Object khác. Default Copy Constructor tương tự như Default Constructor nhưng có đầu vào là một đối tượng.

● **Ví dụ:**

Test::Test(const Test& src) {

iSize = src.iSize;

ptr = new int [iSize];

for (int i=0; i<iSize; ++i) ptr[i] = src.ptr[i];

}

● **Assignment and Initialization(Phần tử hằng, Phần tử tham chiếu, Các đối tượng con yêu cầu đối số trong hàm tạo). Ví dụ Initialization:** ● **Date(int iNDay, int iNMonth, int iNYear) : iDay(iNDay), iMonth(iNMonth), iYear(iNYear){}**

● **Assignment operator**: Cần xóa bộ nhớ có sẵn và xét xem có tự gán cho bản thân không.

Test& Test::operator=(const Test& src) {

if (this != &src) {

delete [] ptr;

iSize = src.iSize;

ptr = new int [iSize];

(int i=0; i<iSize; ++i) ptr[i] = src.ptr[i];

}

return \*this; }

● **Default Assignment operator:** Tương tự Default Copy Constructor.

● **So sánh Copy Constructor và Assignment operator**: Copy Constructor: khởi tạo từ đầu, không cần dọn bộ nhớ trước khi gán, không cần kiểm tra tự gán chính mình.

● **Lưu ý:** Default Copy Constructor và Default Assignment operator khi trong Attribute có con trỏ sẽ gán địa chỉ từ nguồn vào đích. Nếu thay đổi một bên, bên kia cũng sẽ thay đổi. Vì vậy nếu trong Attribute có con trỏ cần định nghĩa Copy Constructor và Assignment operator.

|  |  |  |
| --- | --- | --- |
| **Tiêu chí** | **Copy Constructor** | **Operator Assignment** |
| **Mục đích** | Tạo một đối tượng mới từ một đối tượng có sẵn | Gán giá trị từ một đối tượng có sẵn cho một đối tượng đã tồn tại |
| **Cú pháp khai báo** | ClassName(const ClassName &other); | ClassName& operator=(const ClassName &other); |
| **Thời điểm được gọi** | Khi đối tượng mới được khởi tạo từ một đối tượng hiện có (ví dụ: truyền đối tượng vào hàm bằng giá trị, trả về đối tượng từ hàm, khởi tạo đối tượng mới từ đối tượng khác) | Khi một đối tượng đã tồn tại được gán giá trị từ một đối tượng khác |
| **Mặc định do compiler tạo** | Có (nếu không tự định nghĩa) | Có (nếu không tự định nghĩa) |
| **Hoạt động trên** | Đối tượng chưa được khởi tạo | Đối tượng đã được khởi tạo |
| **Cách hoạt động** | Sao chép tất cả các thành viên dữ liệu từ đối tượng nguồn sang đối tượng mới | Sao chép tất cả các thành viên dữ liệu từ đối tượng nguồn sang đối tượng đích, sau đó trả về đối tượng đích |
| **Kiểu trả về** | Không có kiểu trả về | Trả về một tham chiếu tới đối tượng đích (\*this) |
| **Xử lý tài nguyên** | Phải đảm bảo xử lý đúng các tài nguyên động (nếu có) để tránh lỗi sao chép sâu/shallow | Phải đảm bảo xử lý đúng các tài nguyên động (nếu có) để tránh lỗi sao chép sâu/shallow và giải phóng tài nguyên hiện có đúng cách |
| **Tình huống sử dụng** | Khi khởi tạo đối tượng mới từ một đối tượng có sẵn | Khi gán giá trị từ một đối tượng có sẵn cho một đối tượng đã tồn tại |
| **Tiêu chí** | **Member-wise Copy** | **Deep Copy** |
| **Mục đích** | Sao chép các thành viên của đối tượng trực tiếp | Tạo một bản sao hoàn chỉnh và độc lập của đối tượng |
| **Cách hoạt động** | Sao chép từng thành viên của đối tượng này sang đối tượng kia, chỉ sao chép giá trị của các thành viên | Sao chép từng thành viên của đối tượng, nhưng cũng tạo ra các bản sao riêng biệt cho bất kỳ tài nguyên động nào được quản lý bởi đối tượng |
| **Tình huống sử dụng** | Khi không có tài nguyên động (con trỏ, bộ nhớ động) hoặc tài nguyên không cần sao chép sâu | Khi đối tượng quản lý tài nguyên động hoặc các tài nguyên cần sao chép sâu để đảm bảo độc lập giữa các bản sao |
| **Rủi ro** | Gây ra lỗi khi đối tượng quản lý tài nguyên động, dẫn đến sao chép nông (shallow copy) và vấn đề giải phóng tài nguyên | Phức tạp hơn và tốn kém tài nguyên hơn, nhưng đảm bảo không có sự chia sẻ tài nguyên ngoài ý muốn giữa các đối tượng |
| **Ví dụ minh họa** | Sao chép giá trị số nguyên, số thực, chuỗi tĩnh | Sao chép các đối tượng chứa con trỏ, bộ nhớ động, cấu trúc phức tạp |

● **Destructor (Hàm hủy):** Được dùng để giải phóng các tài nguyên mà đối tượng sử dụng ( ví dụ: bộ nhớ được phân bổ cho một con trỏ, bộ mô tả tệp, ngữ nghĩa). Được gọi mặc định khi Object không sử dụng nữa/Khi bị xóa khỏi bộ nhớ (ví dụ: vượt qua phạm vi truy cập), Mỗi lớp 1 hàm hủy, không trả về kiểu dữ liệu, không tham số đầu vào, cùng tên class sau ~. Nếu ko có des, có thể tạo ra rò rỉ bộ nhớ.

● **Statics member**: Các thuộc tính và phương thức cấp độ lớp, Được chia sẻ giữa các đối tượng cùng lớp, Khởi tạo bên ngoài class, sử dụng :: để truy cập.

class Fraction {

private: static int m\_maxValue;

public: static int getMaxValue();

};

int Fraction::m\_maxValue = 10000;

void main() { int x = Fraction::getMaxValue(); }

**Operator, overloading**

● **Overloading(nạp chồng phương thức):** Trong một class có thể có nhiều Method cùng tên nhưng khác tham số đầu vào(số lượng hoặc kiểu). **Kiểu dữ liệu trả về** **không được xét** là điều kiện của over loading.

● **Operator overloading**: Để định nghĩa việc triển khai operator cho các kiểu dữ liệu mới do người dùng tạo. Các operator bình thường đã được định nghĩa cho các dữ liệu bình thường. Khi chúng ta có kiểu dữ liệu mới như Fraction chúng ta cần định nghĩa lại (overloading) lại các operator này. <returned-type> operator<op(vd: =,+,-...)>(arguments);

● **Một số Operator overloading**

|  |  |  |  |
| --- | --- | --- | --- |
| PhanSo& PhanSo::operator+=(const PhanSo& src) {  Tu=tu\*src.mau+src.tu\*mau;  Mau=mau\*src.mau;  return \*this;  } | bool PhanSo::operator==(const PhanSo& src) const {  return ((tu==src.tu) && (mau==src.mau));  } | | const PhanSo PhanSo::operator+(const PhanSo& src) const {  PhanSo temp;  temp.tu = tu\*src.mau + src.tu\*mau;  temp.mau=mau\*src.mau;  return temp;} |
| PhanSo PhanSo::operator++(int) // toán tử ++ dạng hậu tố{  PhanSo temp = \*this;  ++(\*this);  return temp;} | | PhanSo& PhanSo::operator++() // toán từ ++ dạng tiên tổ {  tu=tu + mau;  return \*this;} | |

● **Friend function**: không phải là hàm thành viên nên không phụ thuộc vào lớp nhưng được sử dụng trực tiếp các attribute,method trong private và protected thông qua tham số đầu vào có kiểu Class đó. Cấp quyền truy cập function/ class khác. Đem lại sự linh hoạt, không vi phạm tính đóng gói. Hàm bth x.f() hàm friend f(x). << và >> phải friend vì tham số đầu tiên là đối tượng thuộc lớp đó.

● **Đặc điểm của hàm friend:**

Hàm friend không phải là thành viên của lớp, nhưng có quyền truy cập vào các thành viên private và protected của lớp đó.

Hàm friend có thể được khai báo trong hoặc ngoài lớp.

Hàm friend có thể truy cập vào tất cả các đối tượng của lớp, bao gồm cả các đối tượng được tạo từ các đối tượng khác.

class Fraction {

friend ostream& operator <<(ostream &os, const Fraction &p);

};

ostream& operator <<(ostream &os, const Fraction &p) {

os << p.m\_num << “/” << p.m\_denom << endl; return os; }

● **Friend class**

Lớp friend trong C++ là một lớp được phép truy cập vào các thành viên private và protected của lớp khác mà nó được khai báo là bạn bên trong. Điều này có nghĩa là tất cả các thành viên của lớp bạn được khai báo là bạn với một lớp khác có thể truy cập vào tất cả các thành viên private và protected của lớp đó.

● **Đặc điểm của lớp friend:**

Lớp friend có thể truy cập vào tất cả các thành viên private và protected của lớp được khai báo là bạn với nó.

Lớp friend được khai báo bên trong lớp hoặc ngoài lớp.

Class A -------(friend)-🡪 Class B

|  |  |
| --- | --- |
| Class A{  Public:  Friend class B;  Private: int a;  }; | Class B{  Public: voidTest(){  test.a=2;}//truy xuất private A  Private:  A test; |

● **Lớp phụ thuộc**

Lớp chính thay đổi sẽ dẫn theo lớp phụ thuộc thay đổi

**Tính Kế Thừa (Inheritance)**

● **Kế Thừa (inheritance)** nghĩa là các hành động (phương thức) và các thuộc tính được định nghĩa trong một lớp có thể được thừa kế hoặc được sử dụng lại bởi lớp khác. Mục đích: Tái sử dụng thuộc tính và hành vi.

● **Superclass = base class = parent**: chứa các cấu trúc và hành vi chung của lớp con kế thừa từ nó.

● **Subclass = derived class = child class**: cụ thể hơn lớp cha, có thể sử dụng mọi thứ của tổ tiên của nó nếu tổ tiên nó cho phép, kết hợp với những đặc trưng riêng của lớp con.

● **Kiểu dữ liệu**: Inheritance-Type(Public,Protected-> Inheritance-Type)(trừ public: pro->pro).

● **Constructor:** Khi một Object của lớp con được tạo, sẽ gọi đến Constructor của cha trước xong mới đến con. Trong hàm tạo của con có thể chọn hàm tạo nào của cha sẽ được gọi nếu không sẽ gọi hàm tạo default. Không thể chọn hàm tạo của lớp ông.

class D : public B {

public: D(int t) : B(t) { ... }

};

Employee(string name, int age, float salary) : Person(name, age) { //… }

● **Destructor:** Khi một Object của lớp con hết vòng đời, sẽ gọi đến Destructor của con trước xong mới đến cha.

● **Để gọi một hàm ở lớp cha**: <Tên lớp cha>::f(x)

● **Overiding(Ghi đè phương thức):** định nghĩa lại một phương thức của lớp cha. Tên, tham số đầu vào hoàn toàn giống với phương thức ở lớp cha (**samesignature**). Overiding sẽ làm mất toàn bộ các overloading ở lớp cha.

● **So sánh Overiding và Overloading**: giống và khác tham số đầu vào.

|  |  |  |
| --- | --- | --- |
| **Khía cạnh** | **Overloading** | **Overriding** |
| **Định nghĩa** | Cho phép định nghĩa nhiều hàm cùng tên trong cùng một phạm vi với các đối số khác nhau. | Cho phép lớp con định nghĩa lại (override) phương thức của lớp cha với cùng tên, cùng đầu vào và đầu ra. |
| **Mục đích** | Giúp tăng tính linh hoạt của mã nguồn bằng cách cung cấp nhiều cách thức để xử lý cùng một hành động với các đối số khác nhau. | Được sử dụng để cung cấp tính đa hình (polymorphism) trong lập trình hướng đối tượng, cho phép lớp con có thể thay đổi hành vi của phương thức được kế thừa từ lớp cha. |
| **Phạm vi** | Được áp dụng trong cùng một lớp. | Được áp dụng trong quan hệ kế thừa giữa lớp cha và lớp con. |
| **Tham số** | Phương thức overloading phải có các tham số khác nhau (số lượng, kiểu dữ liệu hoặc cả hai). | Phương thức overriding phải có chữ ký giống nhau (cùng tên, cùng tham số và cùng kiểu trả về). |
| **Tính đa hình** | Không liên quan đến tính đa hình. | Liên quan đến tính đa hình trong lập trình hướng đối tượng. |
| **Quyết định** | Quyết định được thực hiện tại thời điểm biên dịch (compile time). | Quyết định được thực hiện tại thời điểm chạy (runtime), dựa trên loại đối tượng được tạo. |

● **Sử dụng using để giữ lại các overloading ở lớp cha** ví dụ: using B::test;

● **Assignment operator**: gọi toán tử gán lớp cha trước, để gán các attribute của phần lớp cha của 2 đối tượng. Sao đó, gán các attribute ở lớp con.

D& D::operator=(const D& src) {

if (this == &src)

return \*this;

B::operator=(src); //call the BASE

delete [] ptr;

iSize = src.iSize; ptr = new int [iSize];

for (int i=0; i<iSize; ++i) ptr[i] = src.ptr[i];

return \*this;

}

● **Tổ tiên có thể sử dụng thể hiện của lớp con, ngược lại ko thể,** ví dụ:

Person p2 = e;

Person p3 = Employee(”Nhan”, 25);

Employee e2 = Person(“Hung”, 30); //wrong

Employee e3 = p1; //wrong

● **Generalization(Khái quát hóa):** mối quan hệ is-a (Kế thừa)

● **Aggregation(Tập hợp):** Mối quan hệ has-a (chưa trong class, pri/pub/pro).

**Virtual functions and polymorphism**

●**Implicit type conversion in inheritance**:Một con trỏ kiểu class cha có thể được gán với một địa chỉ của đối tượng lớp con. Có thể truyền một đối tượng lớp con cho một hàm có đối số là kiểu dữ liệu của lớp cha.

Animal\* pAni;

Cat c;

pAni = &c; //OK

Animal::process(const Animal &a);

int main(){

Cat c;

Animal::process(c); //OK

}

● **Static binding**: Khi dùng overiding, nếu dùng con trỏ loại cha gán địa chỉ của lớp con, khi gọi hàm overiding chương trình sẽ gọi hàm của lớp cha không gọi hàm lớp con.

● **VirtualFunction:** Các hàm trong lớp cha sẽ thành hàm ảo khi khai báo virtual trước tên hàm. Lớp con có thể overiding lại hàm ảo.

● **Dynamic binding:** Khi sử dụng hàm ảo, chương trình sẽ đảm bào hàm hàm nào trong overiding sẽ được gọi dựa vào đối tượng gọi hàm.

int main() {

A varA; B varB; C varC;

A\* var1, \*var2;

var1 = &varC; var2 = &varB;

var1->print(); //C.print()

var2->print();//B.print() }

|  |  |  |
| --- | --- | --- |
| **Khía cạnh** | **Static Binding** | **Dynamic Binding** |
| **Định nghĩa** | Quyết định phương thức/hàm được gọi tại thời điểm biên dịch, dựa trên kiểu của biến con trỏ hoặc tham chiếu. | Quyết định phương thức/hàm được gọi tại thời điểm chạy, dựa trên kiểu thực tế của đối tượng được tạo hoặc con trỏ. |
| **Còn được gọi là** | Early binding | Late binding, runtime polymorphism |
| **Thực hiện** | Tại thời điểm biên dịch | Tại thời điểm chạy (runtime) |
| **Sử dụng** | Cho các hàm không ảo (non-virtual functions), hàm tĩnh (static member functions). | Cho các phương thức ảo (virtual functions). |
| **Kết quả ví dụ** | Chọn phương thức của lớp cơ sở tại thời điểm biên dịch. | Chọn phương thức của lớp dẫn xuất dựa trên đối tượng thực tế tại thời điểm chạy. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Loại quan hệ** | **Định nghĩa** | **Ví dụ** | **Ký hiệu** |
| **Association** | Mối quan hệ giữa hai lớp mà mỗi lớp có thể gửi hoặc nhận các thông tin từ lớp kia. Quan hệ này có thể là một-đến-một, một-đến-nhiều, hoặc nhiều-đến-nhiều. Không bắt buộc phụ thuộc chặt chẽ vào nhau. | Lớp Employee và lớp Department có quan hệ một-nhiều (một Department có nhiều Employee). | A black text on a white background  Description automatically generated |
| **Directed Association** | Là một loại Association mà chỉ ra hướng đi của thông tin giữa các đối tượng. | Lớp Order có thể có quan hệ directed association với lớp Customer, với thông tin chỉ được truyền từ Customer đến Order. | A black text on a white background  Description automatically generated |
| **Aggregation** | Mối quan hệ làm rõ mối quan hệ giữa một lớp "tạo thành" (container) và một lớp "được tạo thành" (content). Lớp "tạo thành" có thể tồn tại mà không cần đến lớp "được tạo thành". | Lớp Company có mối quan hệ aggregation với lớp Employee, vì Nhân Viên có thể tồn tại mà không cần có Công Ty. | A black line with black text  Description automatically generated |
| **Composition** | Mối quan hệ mạnh hơn so với Aggregation, trong đó các thành phần (lớp "được tạo thành") là một phần của lớp "tạo thành" và không thể tồn tại mà không có lớp "tạo thành". | Lớp House có mối quan hệ composition với lớp Room, vì phòng không thể tồn tại nếu không có nhà. |  |
| **Generalization** | Mối quan hệ giữa một lớp cơ sở (base class) và một lớp dẫn xuất (derived class), trong đó lớp dẫn xuất kế thừa các thuộc tính và phương thức của lớp cơ sở. | Lớp Animal là lớp cơ sở và lớp Dog là lớp dẫn xuất (kế thừa các thuộc tính và phương thức của Animal). |  |
| **Realization** | Mối quan hệ giữa một lớp (interface) và một lớp (class) mà lớp (class) triển khai các phương thức của lớp (interface). | Interface Shape có thể được triển khai bởi các lớp Circle, Rectangle,... | A black dotted line with black text  Description automatically generated |
| **Dependency** | Mối quan hệ mà một lớp sử dụng một lớp khác. Nó biểu thị mối quan hệ sự phụ thuộc giữa các lớp, khi mà thay đổi trong một lớp có thể ảnh hưởng đến các lớp khác. | Lớp Car có thể có mối quan hệ dependency với lớp Engine, vì Car cần Engine để hoạt động.  Lớp Person phụ thuộc vào lớp Book vì cần truy cập vào book để đọc nội dung của nó. Nhưng lớp Book không phụ thuộc vào Person | A black and white text  Description automatically generated with medium confidence |
| **Usage (Dependency)** | Mối quan hệ giữa một lớp (client) và một lớp khác (supplier) mà client sử dụng các thành phần của supplier. | Lớp Calculator có thể có mối quan hệ usage (dependency) với lớp MathLibrary, khi sử dụng các hàm toán học từ MathLibrary.  Lớp Car use Lớp FuelTank vì lớp Car cần truy cập vào các phương thức cũng như thuộc tính của FuelTank để kiểm tra fuel level, refill fuel,... | A black text and a line  Description automatically generated with medium confidence |

● **Đa hình (polymorphism)** là hiện tượng mà các đối tượng thuộc các class khác nhau có thểbiểu diễn cùng một thông thiệp theo các cách khác nhau. Một phương thức có nhiều hình thức khác nhau.

Compile time Polymorphism: Overloading, Overiding

Runtime Polymorphism: Phương thức ảo

● **Đa hình Virtual**: bao gồm Virtual Function và Dynamic binding.

* Trình biên dịch sẽ chỉ sử dụng Dynamic binding khi đối tượng được gọi thao tác thông qua con trỏ hoặc hàm chiếu.
* Gọi hàm bằng :: sẽ không sử dụng cơ chế ảo.
* Nếu trong lớp con không định nghĩa hàm ảo sẽ gọi hàm định nghĩa mới nhất của hàm này trong cấu trúc phân cấp kế thừa.
* Để triển khai tính đa hình, trình biên dịch phải lưu trữ thông tin kiểu của các đối tượng lớp A sử dụng để gọi đúng phiên bản hàm ảo. Thông thường chỉ đủ chỗ chứa một con trỏ.
* Cách triển khai điển hình của các hàm ảo là thêm vào mỗi đối tượng của một lớp có ít nhất 1 hàm ảo 1 con trỏ vào bảng hảm ảo. Bảng này chưa các con trỏ trỏ tới tất cả hàm ảo của các lớp mà đối tượng thuộc về.

● **Không có hàm tạo ảo** vì hàm tạo được sử dụng để khởi tạo lớp đó và hàm tạo được chạy theo quy tắc tạo class cha trước xong mới tạo class con.

● **Có hàm hủy ảo** để giải phóng bộ nhớ hoặc tài nguyên của đúng đối tượng.

● **Pure virtual function**: Các hàm trong lớp cha sẽ thành hàm thuần ảo khi khai báo virtual trước tên hàm và cho hàm = 0. Lớp con có thể overiding lại hàm thuần ảo.

● **Abstract class**: một lớp có 1 hoặc nhiều hàm thuần ảo thì lớp đó trở thành lớp trừu tượng. Không thể tạo các đối tượng thuộc lớp trừu tượng. Công dụng quan trọng của hàm thuần ảo là cung cấp dao diện mà không để lộ chi tiết triển khai nào.

● **Lưu ý**: Nếu lớp con không overiding hàm thuần ảo thì lớp con cũng là lớp trừu tượng. Với lớp có hàm ảo nên có hàm destructor ảo.

● **Abstraction(Tính trừu tượng):** Cho phép bạn tạo ra các lớp trừu tượng, trong đó lớp trừu tượng chứa các hàm thuần ảo. Tính trừu tượng muốn cho người dùng biết hành vi mà một đối tượng có thể làm mà không quan tâm đến chi tiết thực hiện như thế nào.

**Multiple inheritance**

● **Multiple inheritance**: Khi một class kế thừa từ 2 đến nhiều class thì gọi là đa kế thừa. Mọi thứ đều giống với kế thừa bth..

● **Function name class**: ambiguity(Xung đột tên hàm: mơ hồ): Overload trên các lớp khác nhau không được xem xét. Các lớp cha có tên hàm giống nhau(mặc dù đối số khác nhau), khi lớp con gọi sẽ gây ra mơ hồ vì không biết dùng cái nào. Để giải quyết sử dụng using như ở kế thừa using int main() {

A a;

a.doSth(); //error:ambiguous

a.B::doSth(); // OK

a.C::doSth(); // OK }

class B { void doSth(int); };

class C { void doSth(double); };

class A: public B, public C {...};

void test() {

A a;

a.doSth(10); //Error: ambiguous!

};

class B { void doSth(int); };

class C { void doSth(double); };

class A: public B, public C {

using C::doSth;

using B::doSth;

void doSth(char); };

void test(A& a) {

a.doSth(10); // B::doSth(int)

a.doSth(‘a’); // A::doSth(char)

a.doSth(5.2); // C::doSth(double) };

● **Replicated based class**: Một lớp có khả năng kế thừa từ nhiều lớp khác, có thể sẽ có một lớp cơ sở nhiều lần. Ví dụ: B kế thừa A, C kế thừa A, D kế thừa C và B, D kế thừa lớp A nhiều lần. Điều đó sẽ gây ra Diamond problem.

void test(D\* p) {

p->doSth(); // error: ambiguous

p->A::doSth(); // error: ambiguous

p->B::doSth(); // ok

p->C::doSth(); // ok }

● **GIải quyết:** Sử dụng Virtual base class.B public virtual A,C public virtual A. D chỉ kế thừa 1 A

class A {...};

class B: public virtual A {...};

class C: public virtual A {...};

class D: public B, public C {...};

**Exception handling**

● **Try**: chứa đoạn code có thể bị lỗi.

● **Throw**: nằm trong try, nếu đoạn code có lỗi thì gọi throw <tên lỗi>

● **Catch**: nằm sau try, được gọi đến khi hàm throw bắt được lỗi.

● **Lưu ý**: throw “Invalid age!”; catch(const char\* error){};

Syntas

try { // code may have errors/exceptions// throw ExceptionName(); }

catch( ExceptionName exceptionType1 ) { // catch block }

catch( ExceptionName exceptionType2 ) { // catch block }

● **Nếu trong cacth có xảy ra exception(file close). Cần tạo destructor**

class FileHandler { priviate: FILE\* m\_fileHandle;

-public: FileHandler(const char\* fileName, const char\* mode)

{ m\_fileHandle = fopen(fileName, mode); if (m\_fileHandle == NULL) throw “Unable to open file!” }

**-~FileHandler() { fclose(m\_fileHandle); // this is a very important part }**

-writeFile(char\* text) { if (fputs(text, m\_fileHandle) == EOF )

{ throw “Error when writing to file” ; } } };

● **Superclass: std::exception:**

logic\_error: invalid\_argument, domain\_error, length\_error, out\_of\_range

runtime\_error: range\_error, overflow\_error, underflow\_error,

bad\_typeid, bad\_cast, bad\_alloc, bad\_exception,

Member functions: constructor, destructor, opertator =, what.

**Design Patterns**

● **Design Patterns**: là một giải pháp chung và có thể tái sử dụng cho một vấn đề thường xảy ra. Một mẫu thiết kế có thể giải quyết nhiều vấn đề bằng cách cung cấp một khuôn khổ để xây dựng một ứng dụng. Với các mẫu thiết kế, quá trình thiết kế sẽ sạch hơn và hiệu quả hơn.

**Những phần thường được trình bày trong 1 pattern**

Intent (mục tiêu): mô tả ngắn gọn vấn đề và giải pháp.

Motivation (động lục): giải thích cách mà pattern làm cho solution khả thi với problem

Structure of classes: show từng phần của pattern và cách chúng liên kết

Code example: bằng 1 trong các ngôn ngữ lập trình nổi tiếng để dể hình dung về pattern

**Nguyên tắc SOLID** là năm nguyên tắc thiết kế phần mềm nhằm mục đích tạo ra phần mềm dễ bảo trì, mở rộng và sửa lỗi. Dưới đây là tóm tắt về năm nguyên tắc này:

**- Single Responsibility Principle (SRP) - Nguyên tắc trách nhiệm đơn lẻ:**

Mỗi lớp chỉ nên đảm nhận một trách nhiệm cụ thể và chỉ có một lý do duy nhất để thay đổi.

Ví dụ: Một lớp xử lý việc đọc dữ liệu từ cơ sở dữ liệu không nên đồng thời xử lý việc hiển thị dữ liệu đó.

**- Open/Closed Principle (OCP) - Nguyên tắc mở/đóng:**

Một lớp nên được mở để mở rộng nhưng đóng để sửa đổi.

Điều này có nghĩa là chúng ta có thể thêm chức năng mới bằng cách mở rộng lớp hiện tại, nhưng không nên thay đổi mã nguồn của lớp đó.

**- Liskov Substitution Principle (LSP) - Nguyên tắc thay thế Liskov:**

Các đối tượng của một lớp con phải có thể thay thế cho các đối tượng của lớp cha mà không làm thay đổi tính đúng đắn của chương trình.

Ví dụ: Nếu lớp Bird có phương thức fly(), thì lớp con Penguin (không bay được) không nên kế thừa từ lớp Bird.

**- Interface Segregation Principle (ISP) - Nguyên tắc phân chia giao diện:**

Các khách hàng không nên bị buộc phải phụ thuộc vào các giao diện mà họ không sử dụng.

Thay vì có một giao diện lớn, nên chia nhỏ thành nhiều giao diện cụ thể hơn để các lớp chỉ cần triển khai những gì họ thực sự cần.

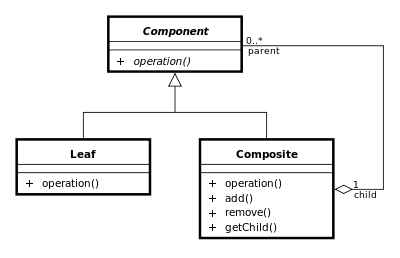
**Dependency Inversion Principle (DIP) - Nguyên tắc đảo ngược phụ thuộc**:

Các module cấp cao không nên phụ thuộc vào các module cấp thấp. Cả hai nên phụ thuộc vào các abstractions.

Các abstractions không nên phụ thuộc vào các chi tiết. Các chi tiết nên phụ thuộc vào các abstractions.

Việc áp dụng SOLID sẽ giúp phần mềm trở nên dễ bảo trì, mở rộng, giảm thiểu rủi ro khi thực hiện các thay đổi.

● **Composite pattern**: Mẫu thiết kế lồng vào nhau. Có các đối tượng không chứa ai.



● **Strategy pattern** (được gọi là chính sách) là mẫu thiết kế phần mềm hành vi cho phép chọn một hành vi (thuật toán) trong thời gian chạy.

Đặt vấn đề khi có một chức năng nào đó được tất cả các lớp con overiding lại nhưng trong đó có nhiều lớp con sẽ có hành vi giống nhau và quá mã cài đặt lại class quá dài, điều này gây trùng lặp nhiều. Vì vậy, strategy pattern xuất hiện. Sẽ có một lớp cha chứa chức năng có thể overiding và các lớp con overiding lại chức năng đó. Có thêm một đối tượng làm việc với người dùng sẽ chứa lớp cha.

**● Sử dụng Strategy Pattern khi:**

Nhiều lớp liên quan chỉ khách nhan về hành vi cụ thể.

Để tránh code phức tạp.

Có nhiều thuật toán cho hành vi cụ thể. Muốn người dùng chọn một hành vi khi ctrinh chạy.

● Những lợi ích Đóng gói Stratery pattern.

Giữ cho các thay đổi của lớp không bắt buộc các thay đổi của lớp khác.

Triển khai các thuật toán độc lập với Context class.

Đơn giản hóa việc chuyển đổi, hiểu và mở rộng các thuật toán.

Tránh trùng lặp mã. Loại bỏ các câu lệnh có điều kiện.

Linh hoạt. Khách hàng có thể lựa chọn giữa các chiến lược khác nhau.

● Hạn chế

Chi phí liên lạc giữa Chiến lược và Bối cảnh.

Số lượng đối tượng tăng lên.

Khách hàng phải nhận thức được các chiến lược và sự khác biệt của chúng

A diagram of a computer program

Description automatically generated

**State pattern**: lớp có nhiều dạng mỗi dạng thực hiện một chức năng. Tạo ra một lớp cha chứa các chức năng thực hiện tên là State, lớp con là các dạng của lớp (State1, State2,..). Sẽ có thêm một đối tượng làm việc với người dùng chứa state để coi trạng thái hiện tại của đối tượng để biết một hành vi sẽ thực hiện thế nào.

Ưu điểm

|  |  |
| --- | --- |
| Ưu điểm | Nhược điểm |
| Đảm bảo nguyên tắc Single Responsibility (SRP): Tách biệt mỗi State tương ứng với 1 class riêng biệt.  Đảm bảo nguyên tắc Open/Closed Principle (OCP): chúng ta có thể thêm một State mới mà không ảnh hưởng đến State khác hay Context hiện có. Giữ hành vi cụ thể tương ứng với mỗi State (trạng thái).  Giúp chuyển State một cách rõ ràng.  Loại bỏ các câu lệnh xét trường hợp (If, Switch case) giúp đơn giản code của context | Việc sử dụng state pattern có thể quá mức cần thiết nếu state machine chỉ có một vài trạng thái hoặc hiếm khi thay đổi có thể dẫn đến việc tăng độ phức tạp của code |

A screenshot of a computer

Description automatically generated

● **Single Ton**

Mẫu singleton là một mẫu thiết kế được sử dụng để hạn chế việc khởi tạo một lớp chỉ với một đối tượng. Nó rất hữu ích khi hệ thống cần chính xác một đối tượng.

**class Singleton {**

**public:**

**static Singleton\* Instance() {**

**if (!singleton) singleton = new Singleton;**

**return singleton;**

**}**

**private:**

**static Singleton\* singleton;**

**Singleton() {};   
 Singleton(const Singleton&);// prevent copy-construction**

**Singleton& operator=(const Singleton&); //prevent =**

**};**

**Singleton\* singleton = nullptr;**

**int main() {**

**Singleton\* pS = new Singleton;**

**//...**

**delete pS;**

**//...**

**return 0;**

**}**

**class Singleton {**

**public:**

**static Singleton& Instance() {**

**static Singleton singleton;**

**return singleton;**

**}**

**private:**

**Singleton() {};   
 Singleton(const Singleton&);// prevent copy-construction**

**Singleton& operator=(const Singleton&); //prevent =**

**};**

**INT array.**

**MyIntArray::MyIntArray(const MyIntArray& temp) {**

**size = temp.size;**

**pArr = new int[size];**

**for (int i = 0; i < size; i++) {**

**pArr[i] = temp.pArr[i];**

**}**

**}**

**int& MyIntArray::operator[](int a) {**

**return pArr[a];**

**}**

**MyIntArray& MyIntArray::operator=(const MyIntArray& temp) {**

**if(pArr!=nullptr) delete[] pArr;**

**if (this != &temp) {**

**size = temp.size;**

**pArr = new int[size];**

**for (int i = 0; i < size; i++) {**

**pArr[i] = temp.pArr[i];**

**}**

**}**

**return \*this;**

**}**

**Fraction**

**ostream& operator<<(ostream& out, PhanSo& temp) {**

**out << temp.Tu << "/" << temp.Mau << endl;**

**return out;**

**}**

**istream& operator>>(istream& in, PhanSo& temp) {**

**temp.inputFraction(temp);**

**return in;**

**}**

**PhanSo& PhanSo::operator+(const PhanSo& temp) {**

**PhanSo fraction;**

**fraction.Tu = Tu \* temp.Mau + temp.Tu \* Mau;**

**fraction.Mau = Mau \* temp.Mau;**

**return fraction;**

**}**

**int finducln(int a, int b) {**

**if (a == 0 || b == 0) {**

**return a + b;**

**}**

**int min;**

**if (a < b) min = a;**

**else min = b;**

**for (int i = min; i > 1; i--) {**

**if (a % i == 0 && b % i == 0) {**

**return i;**

**}**

**}**

**}**

**void PhanSo::simpify() {**

**int temp = finducln(abs(Tu), abs(Mau));**

**Tu /= temp;**

**Mau /= temp;**

**if ((Tu < 0 && Mau < 0)|| (Tu > 0 && Mau < 0)) {**

**Tu \*= -1;**

**Mau \*= -1;**

**}**

**}**

**void MangPS::writeFile(MangPS& temp) {**

**string namefile;**

**fstream file;**

**cout << "Nhap ten file ghi mang phan so: "; cin >> namefile;**

**file.open(namefile, ios\_base::out);**

**for (int i = 0; i < temp.size-1; i++) {**

**file << temp.fraction[i].getTu() << "/" << temp.fraction[i].getMau() << endl;**

**}file << temp.fraction[temp.size-1].getTu() << "/" << temp.fraction[temp.size-1].getMau();**

**file.close();**

**}**

**void MangPS::readFile() {**

**string namefile;**

**char\* a = new char[5];**

**int b,c ;**

**int sz=0;**

**fstream file;**

**cout << "Nhap ten file doc mang phan so: "; cin >> namefile;**

**file.open(namefile, ios\_base::in);**

**while (!file.eof()) {**

**file.getline(a,5,'\n');**

**sz++;**

**}**

**file.close();**

**size = sz;**

**fraction = new PhanSo[sz];**

**file.open(namefile, ios\_base::in);**

**for(int i=0;i<sz;i++) {**

**file >> b;**

**file.getline(a, 2, '/');**

**file >> c;**

**PhanSo fr(b, c);**

**fraction[i] = fr;**

**}**

**file.close();**

**}**

**Complex**

**Complex operator\* (Complex const& C1, Complex const& C2) {**

**Complex temp;**

**temp.Re = C1.Re \* C2.Re + C1.Im \* C2.Im \* -1;**

**temp.Im = C1.Re \* C2.Im + C1.Im \* C2.Re;**

**return temp;**

**}**

**Complex operator/ (Complex const& C1, Complex const& C2) {**

**Complex temp;**

**temp.Re = (C1.Re \* C2.Re + C1.Im \* C2.Im) / pow(C2.Re, 2) + pow(C2.Im, 2);**

**temp.Im = (C1.Im \* C2.Re - C1.Re \* C2.Im) / pow(C2.Re, 2) + pow(C2.Im, 2);**

**return temp;**

**}**

**Đơn thức**

**DonThuc::DonThuc() {**

**heso = 0;**

**bac = 0;**

**}**

**DonThuc::DonThuc(float a, int b) {**

**heso = a;**

**bac = b;**

**}**

**float DonThuc::getHeso() {**

**return heso;**

**}**

**int DonThuc::getBac() {**

**return bac;**

**}**

**void DonThuc::inputFraction(DonThuc& temp) {**

**cout << "Nhap he so: "; cin >> temp.heso;**

**cout << "Nhap bac: "; cin >> temp.bac;**

**}**

**void DonThuc::inputhesoDT(int n) {**

**bac = n;**

**cout << "Nhap he so bac " << n << " : "; cin >> heso;**

**}**

**void DonThuc::khoitaobacDT(int n) {**

**bac = n;**

**heso = 0;**

**}**

**ostream& operator<<(ostream& out, DonThuc& temp) {**

**if (temp.bac == 0) out << temp.heso;**

**else if (temp.heso == 0) return out;**

**else out << temp.heso << "x^" << temp.bac;**

**return out;**

**}**

**istream& operator>>(istream& in, DonThuc& temp) {**

**cout << "Nhap don thuc" << endl;**

**temp.inputFraction(temp);**

**return in;**

**}**

**long long DonThuc::giatriDT(int x) {**

**long long temp = heso \* pow(x, bac);**

**return temp;**

**}**

**DonThuc& DonThuc::operator=(const DonThuc& a) {**

**heso = a.heso;**

**bac = a.bac;**

**return \*this;**

**}**

**DonThuc& DonThuc::operator+(const DonThuc& a) {**

**if (bac != a.bac) {**

**DonThuc temp;**

**return temp;**

**}**

**DonThuc temp1;**

**temp1.bac = bac;**

**temp1.heso = heso + a.heso;**

**return temp1;**

**}**

**DonThuc& DonThuc::operator-(const DonThuc& a) {**

**if (bac != a.bac) {**

**DonThuc temp;**

**return temp;**

**}**

**DonThuc temp1;**

**temp1.bac = bac;**

**temp1.heso = heso - a.heso;**

**return temp1;**

**}**

**DonThuc& DonThuc::operator\*(const DonThuc& a) {**

**DonThuc temp1;**

**temp1.bac = bac + a.bac;**

**temp1.heso = heso \* a.heso;**

**return temp1;**

**}**

**DonThuc& DonThuc::operator/(const DonThuc& a) {**

**DonThuc temp1;**

**temp1.bac = bac - a.bac;**

**temp1.heso = heso / a.heso;**

**return temp1;**

**}**

**Đa thức**

**ArrDaThuc::ArrDaThuc() {**

**donthuc = nullptr;**

**size = 0;**

**}**

**ArrDaThuc& ArrDaThuc::operator=(const ArrDaThuc& a) {**

**if (donthuc != nullptr) delete[]donthuc;**

**size = a.size;**

**donthuc = new DonThuc[size + 1];**

**for (int i = 0; i < size + 1; i++) {**

**donthuc[i] = a.donthuc[i];**

**}**

**return \*this;**

**}**

**void ArrDaThuc::setArrDaThuc() {**

**ArrDaThuc temp = \*this;**

**for (int i = temp.size; i >= 0; i--) {**

**if (temp.donthuc[i].getHeso() != 0) {**

**size = i;**

**break;**

**}**

**else size = -1;**

**}**

**if (size == -1) {**

**size++;**

**donthuc = new DonThuc[size + 1];**

**donthuc[0].khoitaobacDT(0);**

**return;**

**}**

**donthuc = new DonThuc[size + 1];**

**for (int i = 0; i < size + 1; i++) {**

**donthuc[i] = temp.donthuc[i];**

**}**

**}**

**ArrDaThuc::ArrDaThuc(int sz) {**

**donthuc = new DonThuc[sz + 1];**

**for (int i = 0; i < sz + 1; i++) {**

**donthuc[i].khoitaobacDT(i);**

**}**

**size = sz;**

**}**

**ostream& operator<<(ostream& out, ArrDaThuc& a) {**

**for (int i = 0; i < a.size; i++) {**

**if (a.donthuc[i].getHeso() == 0) continue;**

**out << a.donthuc[i] << " + ";**

**}out << a.donthuc[a.size] << endl;**

**return out;**

**}**

**istream& operator>>(istream& in, ArrDaThuc& a) {**

**if (a.donthuc != nullptr) delete[]a.donthuc;**

**cout << "Nhap bac cao nhat cua da thuc: ";**

**cin >> a.size;**

**a.donthuc = new DonThuc[a.size + 1];**

**for (int i = 0; i < a.size + 1; i++) {**

**a.donthuc[i].inputhesoDT(i);**

**}**

**return in;**

**}**

**long long ArrDaThuc::giatriDaT(int x) {**

**long long S = 0;**

**for (int i = 0; i <= size; i++) {**

**S += donthuc[i].giatriDT(x);**

**}**

**return S;**

**}**

**ArrDaThuc& ArrDaThuc::operator+(const ArrDaThuc& a) {**

**ArrDaThuc temp;**

**ArrDaThuc temp1;**

**if (size < a.size) {**

**temp = a;**

**temp1 = \*this;**

**}**

**else {**

**temp1 = a;**

**temp = \*this;**

**}**

**for (int i = 0; i <= temp1.size; i++) {**

**temp.donthuc[i] = temp.donthuc[i] + temp1.donthuc[i];**

**}**

**temp.setArrDaThuc();**

**return temp;**

**}**

**ArrDaThuc& ArrDaThuc::operator-(const ArrDaThuc& a) {**

**ArrDaThuc temp;**

**ArrDaThuc temp1;**

**if (size < a.size) {**

**temp = a;**

**temp1 = \*this;**

**}**

**else {**

**temp1 = a;**

**temp = \*this;**

**}**

**for (int i = 0; i <= temp1.size; i++) {**

**temp.donthuc[i] = donthuc[i] - a.donthuc[i];**

**}**

**temp.setArrDaThuc();**

**return temp;**

**}**

**ArrDaThuc& ArrDaThuc::operator\*(const ArrDaThuc& a) {**

**ArrDaThuc temp(size + a.size);**

**for (int i = 0; i <= a.size; i++) {**

**for (int j = 0; j <= size; j++) {**

**DonThuc b = donthuc[j] \* a.donthuc[i];**

**temp.donthuc[b.getBac()] = temp.donthuc[b.getBac()] + b;**

**}**

**}**

**temp.setArrDaThuc();**

**return temp;**

**}**

**ArrDaThuc& ArrDaThuc::operator/(const ArrDaThuc& a) {**

**if (this->size < a.size) {**

**ArrDaThuc Thuong(0);**

**return Thuong;**

**}**

**ArrDaThuc Thuong(size - a.size);**

**ArrDaThuc Sodu = \*this;**

**DonThuc temp, nhan;**

**while (Sodu.size >= a.size) {**

**temp = Sodu.donthuc[Sodu.size] / a.donthuc[a.size];**

**Thuong.donthuc[temp.getBac()] = Thuong.donthuc[temp.getBac()] + temp;**

**for (int i = 0; i <= a.size; i++) {**

**nhan = temp \* a.donthuc[i];**

**Sodu.donthuc[nhan.getBac()] = Sodu.donthuc[nhan.getBac()] - nhan;**

**}**

**Sodu.setArrDaThuc();**

**if (Sodu.size == 0 && Sodu.donthuc[0].getHeso() == 0) break;**

**}**

**return Thuong;**

**}**

**Student::Student(long id, double gp)**

**:Person(id), gpa(gp)**

**{**

**assert(gpa >= 0 && gpa <= 10);**

**}**

**string Point2D::toString() {**

**string a;**

**a = "x: " + to\_string(this->x) + " y: " + to\_string(this->y);**

**return a;**

**}**

**Ghi file**

**void Registrar::readStdFile() {**

**fstream f;**

**string a, b;**

**f.open("Student.txt", ios::in);**

**getline(f, b, '\n');**

**szStd = stoi(b);**

**this->std = new Student[szStd];**

**for (int i = 0; i < szStd; i++) {**

**getline(f,a,',');**

**getline(f, b, '\n');**

**Student st(b, a);**

**std[i] = st;**

**}**

**f.close();**

**}**

**Đa hình**

**bool Circle::IsSelected(Point p) {**

**return sqrt(pow((C.X - p.X), 2) + pow((C.Y - p.Y), 2)) <= R;**

**}**

**string Circle::getTypeClass() {**

**return "Circle";**

**}**

**Shape\* Circle::Clone() {**

**Circle\* c = new Circle;**

**if (c == nullptr) return nullptr;**

**c->C = C;**

**c->R = R;**

**return c;**

**}**

**bool Rectangle::IsSelected(Point p) {**

**return ((p.X >= X.X) && (p.X <= X.X + W) && (p.Y >= X.Y) && (p.Y <= X.Y + H));**

**}**

**Shape\* Rectangle::Clone() {**

**Rectangle\* r = new Rectangle;**

**if (r == nullptr) return nullptr;**

**r->H = H;**

**r->W = W;**

**r->X = X;**

**return r;**

**}**

**string Rectangle::getTypeClass() {**

**return "Rectangle";**

**}**

**Exception handling**

**if (second == 0) {**

**throw invalid\_argument("Error! Devide by zero!");**

**}**

**try**

**{**

**cout << "Result of division: "**

**<< stdQuotient(num1, num2);**

**cout << endl;**

**}**

**catch (invalid\_argument ex) {**

**cout << ex.what() << endl;**

**}**

**class myException {**

**private:**

**string message;**

**public:**

**myException() {**

**message = "";**

**}**

**myException(string msg) {**

**message = msg;**

**}**

**string getMessage() {**

**return message;**

**}**

**};**

**double division() {**

**if (num2 == 0) {**

**throw myException("Error! Devide by zero!");**

**}**

**return (double)num1 / (double)num2;**

**}**

**try {**

**d.input();**

**cout<<d.division()<<endl;**

**}**

**catch (myException e) {**

**cout << e.getMessage() << endl;**

**}**

**Nhân Viên**

**NhanVien::NhanVien() {**

**this->MSNV = new char[100];**

**this->HoTen = new char[100];**

**this->DiaChi = new char[100];**

**this->GioiTinh = new char[100];**

**this->NgaySinh.d = this->NgaySinh.m = this->NgaySinh.y = 0;**

**}**

**NhanVien::NhanVien(char\* ms, char\* ht, NgSinh ns, char\* DiaChi, char\* gt) {**

**this->MSNV = ms;**

**this->HoTen = ht;**

**this->GioiTinh = gt;**

**this->NgaySinh.d = ns.d;**

**this->NgaySinh.m = ns.m;**

**this->NgaySinh.y = ns.y;**

**this->DiaChi = DiaChi;**

**}**

**NhanVien::NhanVien(const NhanVien& nv) {**

**this->MSNV = nv.MSNV;**

**this->HoTen = nv.HoTen;**

**this->GioiTinh = nv.GioiTinh;**

**this->NgaySinh.d = nv.NgaySinh.d;**

**this->NgaySinh.m = nv.NgaySinh.m;**

**this->NgaySinh.y = nv.NgaySinh.y;**

**this->DiaChi = nv.DiaChi;**

**}**

**void NhanVien::print() {**

**cout << "MSNV: " << this->MSNV << endl;**

**cout << "Ho Ten: " << this->HoTen << endl;**

**cout << "Gioi Tinh: " << this->GioiTinh << endl;**

**cout << "Ngay Sinh: " << this->NgaySinh.d << "/" << this->NgaySinh.m << "/" << this->NgaySinh.y << endl;**

**cout << "Dia Chi: " << this->DiaChi << endl;**

**}**

**char\* NhanVien::getMS() {**

**return this->MSNV;**

**}**

**char\* NhanVien::getT() {**

**return this->HoTen;**

**}**

**int NhanVien::getTS() {**

**return this->NgaySinh.m;**

**}**

**void NhanVien::inputNhanVien() {**

**this->MSNV = new char[100];**

**this->HoTen = new char[100];**

**this->DiaChi = new char[100];**

**this->GioiTinh = new char[100];**

**cin.ignore();**

**do {**

**try {**

**cout << "Nhap MSNV: "; cin.getline(this->MSNV, 100);**

**if (strlen(this->MSNV) != 5) throw "Loi! Nhap Id";**

**for (int i = 0; i < strlen(this->MSNV); i++) {**

**if (MSNV[i] < 48 || MSNV[i]>57) throw "Loi! Nhap Id";**

**}break;**

**}**

**catch (const char\* a) {**

**cout << a << endl;**

**}**

**} while (1);**

**do {**

**try {**

**cout << "Nhap ho ten: "; cin.getline(this->HoTen, 100);**

**for (int i = 0; i < strlen(this->HoTen) - 1; i++) {**

**if (HoTen[i] == ' ' && (HoTen[i + 1] < 65 || HoTen[i + 1] > 90)) throw "Loi! Nhap Ho Ten";**

**}break;**

**}**

**catch (const char\* a) {**

**cout << a << endl;**

**}**

**} while (1);**

**cout << "Nhap gioi tinh: "; cin.getline(this->GioiTinh, 100);**

**do {**

**try {**

**cout << "Nhap ngay sinh" << endl;**

**cout << "Ngay: "; cin >> this->NgaySinh.d;**

**cout << "Thang: "; cin >> this->NgaySinh.m;**

**cout << "Nam: "; cin >> this->NgaySinh.y;**

**if (strcmp(this->GioiTinh, "Nam") == 0 && (2024 - NgaySinh.y < 18 || 2024 - NgaySinh.y > 60)) throw "Loi! Nhap Ngay Sinh";**

**if (strcmp(this->GioiTinh, "Nu") == 0 && (2024 - NgaySinh.y < 18 || 2024 - NgaySinh.y > 55)) throw "Loi! Nhap Ngay Sinh";**

**break;**

**}**

**catch (const char\* a) {**

**cout << a << endl;**

**}**

**} while (1);**

**cin.ignore();**

**cout << "Nhap dia chi: "; cin.getline(this->DiaChi, 100);**

**}**

**char\* NhanVien::getDC() {**

**return DiaChi;**

**}**

**Nhân Viên Công Nhật**

**NVCongNhat::NVCongNhat() :NhanVien() {**

**SoNC = 0;**

**}**

**NVCongNhat::NVCongNhat(int snc, char\* ms, char\* ht, NgSinh ns, char\* DiaChi, char\* gt) :NhanVien(ms, ht, ns, DiaChi, gt) {**

**SoNC = snc;**

**}**

**NVCongNhat::NVCongNhat(const NVCongNhat& temp) :NhanVien(temp) {**

**this->SoNC = temp.SoNC;**

**}**

**void NVCongNhat::writeFile(fstream& file) {**

**char a[30] = "NVCongNhat";**

**file.write((char\*)a, sizeof(a));**

**file.write(reinterpret\_cast<char\*>(MSNV), 100);**

**file.write(reinterpret\_cast<char\*>(HoTen), 100);**

**file.write(reinterpret\_cast<char\*>(DiaChi), 100);**

**file.write(reinterpret\_cast<char\*>(GioiTinh), 100);**

**file.write(reinterpret\_cast<char\*>(&NgaySinh), sizeof(NgSinh));**

**file.write(reinterpret\_cast<char\*>(&SoNC), sizeof(SoNC));**

**}**

**void NVCongNhat::readFile(fstream& file) {**

**file.read(reinterpret\_cast<char\*>(MSNV), 100);**

**file.read(reinterpret\_cast<char\*>(HoTen), 100);**

**file.read(reinterpret\_cast<char\*>(DiaChi), 100);**

**file.read(reinterpret\_cast<char\*>(GioiTinh), 100);**

**file.read(reinterpret\_cast<char\*>(&NgaySinh), sizeof(NgSinh));**

**file.read(reinterpret\_cast<char\*>(&SoNC), sizeof(SoNC));**

**}**

**NVCongNhat& NVCongNhat::operator=(const NVCongNhat& temp) {**

**this->HoTen = temp.HoTen;**

**this->DiaChi = temp.DiaChi;**

**this->GioiTinh = temp.GioiTinh;**

**this->MSNV = temp.MSNV;**

**this->SoNC = temp.SoNC;**

**this->NgaySinh = temp.NgaySinh;**

**return \*this;**

**}**

**void NVCongNhat::print() {**

**NhanVien::print();**

**cout << "So ngay cong: " << SoNC << endl;**

**}**

**void NVCongNhat::inputNhanVien() {**

**NhanVien::inputNhanVien();**

**cout << "Nhap so ngay cong: "; cin >> this->SoNC;**

**cout << endl;**

**}**

**long long NVCongNhat::getLuong() {**

**return this->SoNC \* 300000;**

**}**

**string NVCongNhat::getTypeNV() {**

**return "NVCongNhat";**

**}**

**Nhân Viên Sản Xuất**

**NVSanXuat::NVSanXuat() :NhanVien() {**

**SoSP = 0;**

**}**

**NVSanXuat::NVSanXuat(int ssp, char\* ms, char\* ht, NgSinh ns, char\* DiaChi, char\* gt) :NhanVien(ms, ht, ns, DiaChi, gt) {**

**SoSP = ssp;**

**}**

**NVSanXuat::NVSanXuat(const NVSanXuat& temp) :NhanVien(temp) {**

**this->SoSP = temp.SoSP;**

**}**

**void NVSanXuat::writeFile(fstream& file) {**

**char a[30] = "NVSanXuat";**

**file.write((char\*)a, sizeof(a));**

**file.write(reinterpret\_cast<char\*>(MSNV), 100);**

**file.write(reinterpret\_cast<char\*>(HoTen), 100);**

**file.write(reinterpret\_cast<char\*>(DiaChi), 100);**

**file.write(reinterpret\_cast<char\*>(GioiTinh), 100);**

**file.write(reinterpret\_cast<char\*>(&NgaySinh), sizeof(NgSinh));**

**file.write(reinterpret\_cast<char\*>(&SoSP), sizeof(SoSP));**

**}**

**void NVSanXuat::readFile(fstream& file) {**

**file.read(reinterpret\_cast<char\*>(MSNV), 100);**

**file.read(reinterpret\_cast<char\*>(HoTen), 100);**

**file.read(reinterpret\_cast<char\*>(DiaChi), 100);**

**file.read(reinterpret\_cast<char\*>(GioiTinh), 100);**

**file.read(reinterpret\_cast<char\*>(&NgaySinh), sizeof(NgSinh));**

**file.read(reinterpret\_cast<char\*>(&SoSP), sizeof(SoSP));**

**}**

**NVSanXuat& NVSanXuat::operator=(const NVSanXuat& temp) {**

**this->HoTen = temp.HoTen;**

**this->DiaChi = temp.DiaChi;**

**this->GioiTinh = temp.GioiTinh;**

**this->MSNV = temp.MSNV;**

**this->SoSP = temp.SoSP;**

**this->NgaySinh = temp.NgaySinh;**

**return \*this;**

**}**

**void NVSanXuat::print() {**

**NhanVien::print();**

**cout << "So san pham: " << SoSP << endl;**

**}**

**void NVSanXuat::inputNhanVien() {**

**NhanVien::inputNhanVien();**

**cout << "Nhap so san pham: "; cin >> this->SoSP;**

**cout << endl;**

**}**

**long long NVSanXuat::getLuong() {**

**return this->SoSP \* 20000;**

**}**

**string NVSanXuat::getTypeNV() {**

**return "NVSanXuat";**

**}**

**QLCongTy**

**QLCongTy::QLCongTy() {**

**listNV = nullptr;**

**SLNV = 0;**

**}**

**QLCongTy::QLCongTy(NhanVien\*\* listnv, int slnv) {**

**if (listNV != nullptr) delete[]listNV;**

**listNV = new NhanVien \* [slnv];**

**this->SLNV = slnv;**

**for (int i = 0; i < SLNV; i++)**

**listNV[i] = listnv[i];**

**}**

**void QLCongTy::ghiFileBi() {**

**string a = "NhanVien.dat";**

**fstream file;**

**file.open(a, ios::out | ios::binary);**

**file.write(reinterpret\_cast<char\*>(&SLNV), sizeof(SLNV));**

**for (int i = 0; i < SLNV; i++) {**

**listNV[i]->writeFile(file);**

**}**

**file.close();**

**}**

**void QLCongTy::docFileBi() {**

**string a = "NhanVien.dat";**

**fstream file;**

**char b[30];**

**file.open(a, ios::in | ios::binary);**

**file.read((char\*)&SLNV, sizeof(SLNV));**

**if (listNV != nullptr)**

**delete[]listNV;**

**listNV = new NhanVien \* [SLNV];**

**for (int i = 0; i < SLNV; i++) {**

**file.read((char\*)b, sizeof(b));**

**if (strcmp(b, "NVCongNhat") == 0) {**

**listNV[i] = new NVCongNhat;**

**listNV[i]->readFile(file);**

**}**

**else {**

**listNV[i] = new NVSanXuat;**

**listNV[i]->readFile(file);**

**}**

**}**

**file.close();**

**this->printListNV();**

**}**

**void QLCongTy::inputListNV() {**

**cout << "Nhap so luong nhan vien: "; cin >> SLNV;**

**listNV = new NhanVien \* [SLNV];**

**int Chon;**

**for (int i = 0; i < SLNV; i++) {**

**cout << "Loai nhan vien: 1. Nhan vien cong nhat 2. Nhan vien san xuat" << endl;**

**cout << "Chon: "; cin >> Chon;**

**if (Chon == 1) {**

**NVCongNhat\* nv = new NVCongNhat;**

**nv->inputNhanVien();**

**listNV[i] = nv;**

**}**

**else {**

**NVSanXuat\* nv = new NVSanXuat;**

**nv->inputNhanVien();**

**listNV[i] = nv;**

**}**

**}**

**}**

**void QLCongTy::addNhanVien() {**

**if (SLNV == 0) {**

**SLNV++;**

**listNV = new NhanVien \* [SLNV];**

**}**

**else {**

**NhanVien\*\* nv = new NhanVien \* [SLNV];**

**for (int i = 0; i < SLNV; i++) {**

**nv[i] = listNV[i];**

**}**

**SLNV++;**

**listNV = new NhanVien \* [SLNV];**

**for (int i = 0; i < SLNV - 1; i++) {**

**listNV[i] = nv[i];**

**}**

**}**

**int Chon;**

**cout << "Chon loai nhan vien: 1. Nhan vien cong nhat 2. Nhan vien san xuat" << endl;**

**cout << "Chon: "; cin >> Chon;**

**if (Chon == 1) {**

**NVCongNhat\* nv1 = new NVCongNhat;**

**nv1->inputNhanVien();**

**listNV[SLNV - 1] = nv1;**

**}**

**else {**

**NVSanXuat\* nv1 = new NVSanXuat;**

**nv1->inputNhanVien();**

**listNV[SLNV - 1] = nv1;**

**}**

**this->ghiFileBi();**

**}**

**void QLCongTy::deleteNhanVien(char\* id) {**

**int pos = -1;**

**if (SLNV == 1 && strcmp(id, listNV[0]->getMS()) == 0) {**

**cout << "Danh sach khong con nhan vien" << endl;**

**listNV = nullptr;**

**SLNV = 0;**

**return;**

**}**

**NhanVien\*\* nv = new NhanVien \* [SLNV];**

**for (int i = 0; i < SLNV; i++) {**

**nv[i] = listNV[i];**

**if (strcmp(id, listNV[i]->getMS()) == 0) pos = i;**

**}**

**if (pos == -1) {**

**cout << "Khong ton tai nhan vien nay" << endl;**

**return;**

**}**

**SLNV--;**

**int k = 0;**

**cout << SLNV << endl;**

**listNV = new NhanVien \* [SLNV];**

**for (int i = 0; i < SLNV + 1; i++) {**

**if (strcmp(id, nv[i]->getMS()) == 0) continue;**

**listNV[k] = nv[i];**

**k++;**

**}**

**}**

**void QLCongTy::printListNV() {**

**cout << "Danh sach nhan vien: " << endl;**

**if (listNV != nullptr) {**

**for (int i = 0; i < this->SLNV; i++) {**

**listNV[i]->print();**

**cout << endl;**

**}**

**}**

**else {**

**cout << "Khong co sinh vien" << endl;**

**}**

**}**

**void QLCongTy::sumSalary() {**

**long long Sum, SumCN, SumSX;**

**Sum = SumCN = SumSX = 0;**

**for (int i = 0; i < this->SLNV; i++) {**

**if (listNV[i]->getTypeNV() == "NVCongNhat") SumCN += listNV[i]->getLuong();**

**else SumSX += listNV[i]->getLuong();**

**}**

**Sum = SumSX + SumCN;**

**cout << "Tong luong NV Cong Nhat: " << SumCN << endl;**

**cout << "Tong luong NV San Xuat: " << SumSX << endl;**

**cout << "Tong luong Nhan Vien: " << Sum << endl;**

**}**

**NhanVien\*\* QLCongTy::maxSalary(int& dem) {**

**long long Salary = 0;**

**for (int i = 0; i < SLNV; i++) {**

**if (Salary < listNV[i]->getLuong()) {**

**Salary = listNV[i]->getLuong();**

**}**

**}**

**dem = 0;**

**for (int i = 0; i < SLNV; i++) {**

**if (Salary == listNV[i]->getLuong()) {**

**dem++;**

**}**

**}**

**if (dem == 0) {**

**NhanVien\*\* temp = nullptr;**

**return temp;**

**}**

**NhanVien\*\* temp = new NhanVien \* [dem];**

**dem = 0;**

**for (int i = 0; i < SLNV; i++) {**

**if (Salary == listNV[i]->getLuong()) {**

**temp[dem] = listNV[i];**

**dem++;**

**}**

**}**

**return temp;**

**}**

**long long QLCongTy::AvgSalary() {**

**long long Avg;**

**long long Sum = 0;**

**for (int i = 0; i < this->SLNV; i++) {**

**Sum += listNV[i]->getLuong();**

**}**

**Avg = Sum / SLNV;**

**return Avg;**

**}**

**NhanVien\* QLCongTy::findMSNhanVien(char\* ms) {**

**NhanVien\* temp = nullptr;**

**for (int i = 0; i < this->SLNV; i++) {**

**if (strcmp(ms, listNV[i]->getMS()) == 0) {**

**temp = listNV[i];**

**}**

**}**

**return temp;**

**}**

**NhanVien\*\* QLCongTy::findTenNhanVien(char\* ten, int& dem) {**

**dem = 0;**

**for (int i = 0; i < this->SLNV; i++) {**

**if (strcmp(ten, listNV[i]->getT()) == 0) {**

**dem++;**

**}**

**}**

**if (dem == 0) {**

**NhanVien\*\* temp = nullptr;**

**return temp;**

**}**

**NhanVien\*\* temp = new NhanVien \* [dem];**

**dem = 0;**

**for (int i = 0; i < this->SLNV; i++) {**

**if (strcmp(ten, listNV[i]->getT()) == 0) {**

**temp[dem] = listNV[i];**

**dem++;**

**}**

**}**

**return temp;**

**}**

**int QLCongTy::birthday5() {**

**int dem = 0;**

**for (int i = 0; i < this->SLNV; i++) {**

**if (listNV[i]->getTS() == 5) {**

**dem++;**

**}**

**}**

**return dem;**

**}**

**void QLCongTy::getUpperNV() {**

**int dem = 0;**

**long long avg = this->AvgSalary();**

**for (int i = 0; i < this->SLNV; i++) {**

**if (listNV[i]->getLuong() > avg) {**

**dem++;**

**}**

**}**

**NhanVien\*\* temp = new NhanVien \* [dem];**

**dem = 0;**

**for (int i = 0; i < this->SLNV; i++) {**

**if (listNV[i]->getLuong() > avg) {**

**temp[dem] = listNV[i];**

**dem++;**

**}**

**}**

**fstream file;**

**file.open("UpperNV.dat", ios::out | ios::binary);**

**file.write(reinterpret\_cast<char\*>(&dem), sizeof(dem));**

**file.write(reinterpret\_cast<char\*>(temp), sizeof(\*temp) \* dem);**

**file.close();**

**file.open("UpperNV.dat", ios::in | ios::binary);**

**file.read(reinterpret\_cast<char\*>(&dem), sizeof(dem));**

**if (temp != nullptr) delete[]temp;**

**temp = new NhanVien \* [dem];**

**file.read(reinterpret\_cast<char\*>(temp), sizeof(temp) \* dem);**

**file.close();**

**cout << "Danh sach nhan vien: " << endl;**

**if (temp != nullptr) {**

**for (int i = 0; i < dem; i++) {**

**temp[i]->print();**

**cout << endl;**

**}**

**}**

**else {**

**cout << "Khong co sinh vien" << endl;**

**}**

**}**

**void QLCongTy::sortDSNhanVien() {**

**NhanVien\* temp;**

**for (int i = 0; i < this->SLNV - 1; i++) {**

**for (int j = i; j < this->SLNV; j++) {**

**if (strcmp(listNV[i]->getMS(), listNV[j]->getMS()) == 1) {**

**temp = listNV[i];**

**listNV[i] = listNV[j];**

**listNV[j] = temp;**

**}**

**}**

**}**

**}**

**Student::Student(int id, const string& name, double gpa) : stdId(id), stdGpa(gpa)**

**{**

**strcpy\_s(stdName,name.length(), name.c\_str());**

**if (stdId < 1 || stdId > 99)**

**{**

**cout << "Identity is out of range. Program aborted.";**

**assert(false);**

**}**

**if (stdGpa < 0.0 || stdGpa > 4.0)**

**{**

**cout << "The gpa value is out of range. Program aborted.";**

**assert(false);**

**}**

**}**

**Composite**

class Object {

protected:

int weight;

int cost;

public:

virtual int getWeight() = 0;

virtual int getCost() = 0;

virtual float wrapFee() = 0;

};

class Box : public Object {

private:

list<Object\*> objects;

public:

Box() {

this->weight = 20;

this->cost = 2;

}

float wrapFee() override {

return this->getWeight() \* 0.05;

}

void addObject(Object\* object) {

objects.push\_back(object);

}

void removeObject(Object\* object) {

objects.remove(object);

}

int getWeight() override {

int totalWeight = this->weight;

for (Object\* object : objects) {

totalWeight += object->getWeight();

}

return totalWeight;

}

int getCost() override {

int totalCost = this->cost;

for (Object\* object : objects) {

totalCost += object->getCost() + object->wrapFee();

}

return totalCost;

}

};

class Item : public Object {

public:

Item(int weight, int cost) {

this->weight = weight;

this->cost = cost;

}

int getWeight() override {

return weight;

}

int getCost() override {

return cost;

}

float wrapFee() override {

return 0;

}

};

**Stratery pattern**

class IFlyComponent

{

public:

virtual void fly() = 0;

};

class FlyWithWings : public IFlyComponent

{

public:

void fly()

{

cout << "I'm flying with wings." << endl;

}

};

class FlyNoWay : public IFlyComponent

{

public:

void fly()

{

cout << "I can't fly." << endl;

}

};

class FlyRocketPowered : public IFlyComponent

{

public:

void fly()

{

cout << "I'm flying with a rocket." << endl;

}

};

class IQuackComponent

{

public:

virtual void quack() = 0;

};

class BaseQuack : public IQuackComponent

{

public:

void quack() override

{

cout << "Base Quack" << endl;

}

};

class AnotherQuack : public IQuackComponent

{

public:

void quack() override

{

cout << "Another Quack" << endl;

}

};

class Duck {

private:

string name;

protected:

IFlyComponent\* flyComponent;

IQuackComponent\* quackComponent;

public:

Duck(string name)

{

this->name = name;

this->flyComponent= nullptr;

this->quackComponent = nullptr;

}

virtual void display() = 0;

virtual void fly() = 0;

virtual void quack() = 0;

};

class BaseDuck : public Duck

{

public:

BaseDuck(string name) : Duck(name)

{

this->flyComponent = new FlyWithWings();

this->quackComponent = new BaseQuack();

}

void display() override

{

cout << "I'm a Base"<<endl;

}

void fly() override

{

this->flyComponent->fly();

}

void quack() override

{

this->quackComponent->quack();

}

};

class AnotherDuck : public Duck

{

public:

AnotherDuck(string name) : Duck(name)

{

this->flyComponent = new FlyNoWay();

this->quackComponent = new AnotherQuack();

}

void display() override

{

cout << "I'm Another" << endl;

}

void fly() override

{

this->flyComponent->fly();

}

void quack() override

{

this->quackComponent->quack();

}

};

#include <iostream>

#include <memory>

// Forward declaration

class Context;

// State Interface

class State {

public:

    virtual void handle(Context& context) = 0;

    virtual ~State() = default;

};

// Context Class

class Context {

private:

    std::shared\_ptr<State> state;

public:

    Context(std::shared\_ptr<State> state) : state(state) {}

    void setState(std::shared\_ptr<State> newState) {

        state = newState;

    }

    void request() {

        state->handle(\*this);

    }

};

// Concrete State A

class ConcreteStateA : public State {

public:

    void handle(Context& context) override;

};

// Concrete State B

class ConcreteStateB : public State {

public:

    void handle(Context& context) override;

};

// Method definitions to avoid circular dependency issues

void ConcreteStateA::handle(Context& context) {

    std::cout << "State A handling request and changing to State B" << std::endl;

    context.setState(std::make\_shared<ConcreteStateB>());

}

void ConcreteStateB::handle(Context& context) {

    std::cout << "State B handling request and changing to State A" << std::endl;

    context.setState(std::make\_shared<ConcreteStateA>());

}

int main() {

    // Initial state is State A

    std::shared\_ptr<State> stateA = std::make\_shared<ConcreteStateA>();

    Context context(stateA);

    // Request transitions

    context.request();  // State A handling request and changing to State B

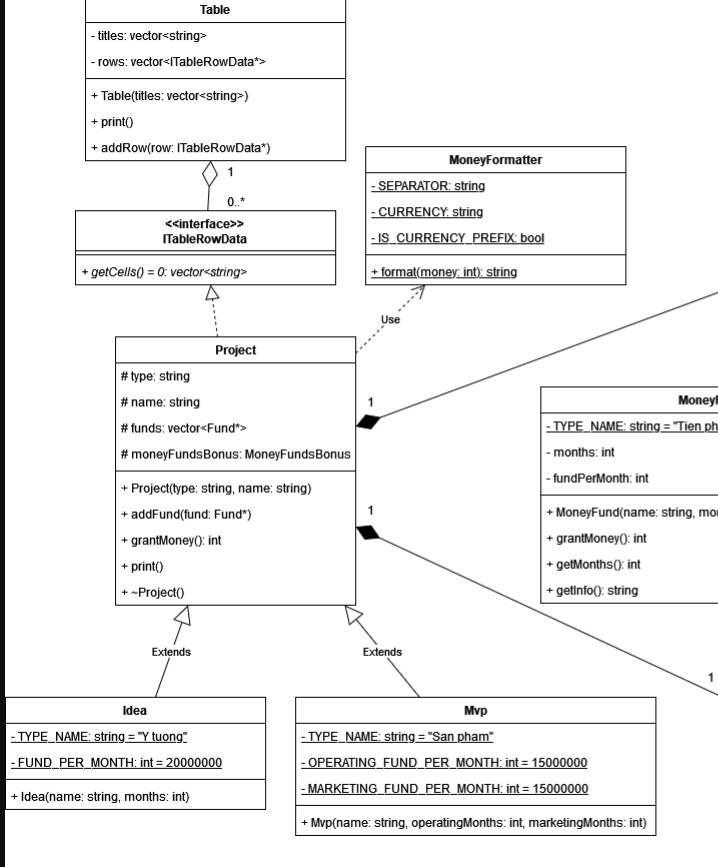
    context.request();  // State B handling request and changing to State A

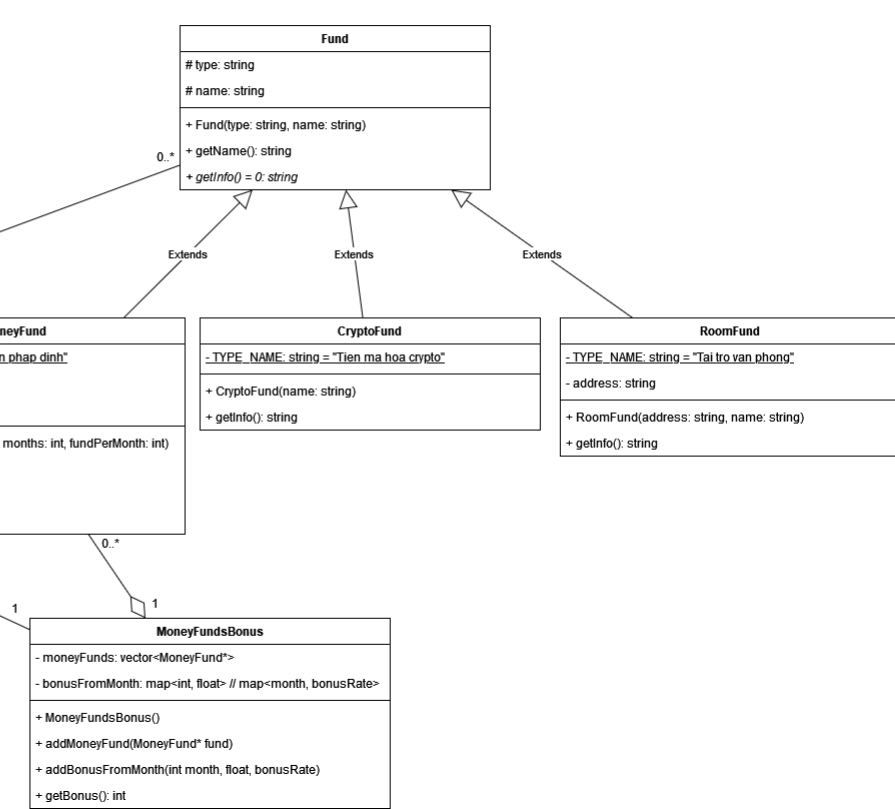
    context.request();  // State A handling request and changing to State B

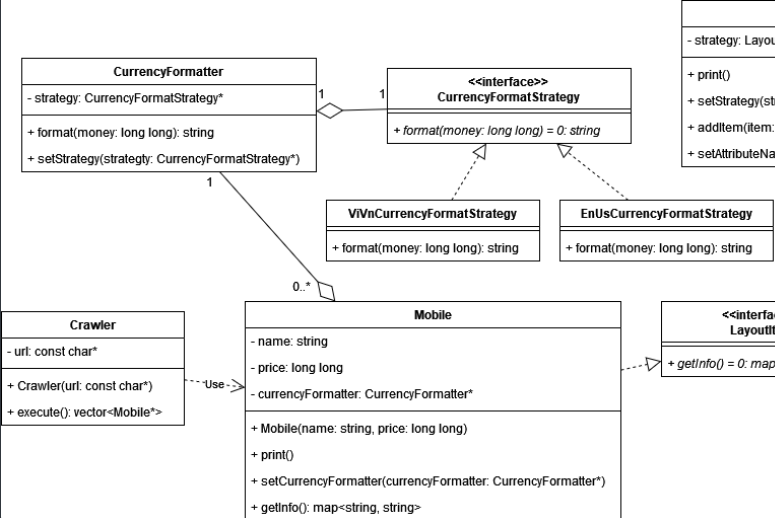
    context.request();  // State B handling request and changing to State A

    return 0;

}

****

****

****

**A diagram of a computer

Description automatically generated with medium confidence**

|  |  |
| --- | --- |
| **Is-A** | **Has-A** |
| Là 1 khái niệm của kế thừa (Inheritance) | Là 1 khái niệm của bao hàm phụ thuộc (Composite) |
| 1 lớp không thể mở rộng hơn 1 lớp | 1 lớp có thể có mối quan hệ Has-A với nhiều lớp khác |
| Lớp cuối không thể được mở rộng trong kế thừa | Có thể dùng lại lớp cuối trong bao hàm phụ thuộc |
| Sự kế thừa là liên kết tĩnh (static binding) và ko thể bị thay đổi tại thời điểm chạy ctrinh | Composite là liên kết động (dynamic binding) và có thể thay đổi linh hoạt |

**Giống:** Đều là khái niệm của đa hình tại thời điểm chạy chương trình**.** Prototype của 2 hàm giữ nguyên trong suốt chương trình**.** Đều không thể là toàn cục hoặc tĩnh

|  |  |
| --- | --- |
| **Virtual function** | **Pure virtual function** |
| **Là 1 member function của lớp cha mà có thể được định nghĩa lại bởi lớp con. Định nghĩa hàm được viết trong lớp cha** | **Là 1 member function của lớp cha mà chỉ khai báo hàm được viết trong lớp cha và nên được định nghĩa lại bởi lớp con nếu không lớp con cũng trở thành lớp trừu tượng. Không có định nghĩa hàm trong lớp cha** |
| **Sự tồn tại của nó không biến lớp thành lớp trừu tượng** | **Lớp cha chứa nó trở thành lớp trừu tượng** |
| **Lớp cha chứa nó có thể được khởi tạo** | **Lớp cha chứa nó sẽ trở thành abstract và không thể được khởi tạo** |
| **Nếu lớp con không định nghĩa lại hàm ảo của lớp con thì cũng không ảnh hưởng sự biên dịch** | **Nếu lớp con không định nghĩa lại hàm thuần ảo thì cũng không có lỗi biên dịch, nhưng lớp con cũng sẽ trở thành lớp trừu tượng giống lớp cha** |