Chương 10

Lập trình hướng đối tượng

--Thừa kế--

Lê Thành Sách

Nội dung

- Tại sao cần đến thừa kế
- Các khái niệm
- Thừa kế là gì?
- Các kiểu thừa kế
- Thiết kế các lớp (I).
- Khởi tạo lớp cha từ lớp con
- Thiết kế các lớp (II).
- Tổng kết

Tại sao cần đến thừa kế

- Giả sử một hệ thống phần mềm cho một trường đại học (Bách Khoa). Nhiều nhóm người dùng có thể dùng hệ thống này, họ có thể là:
  a) Giảng viên (lecturer)
  b) Sinh viên (student)
  c) Nhân viên văn phòng (clerk)
  d) Bảo vệ (guardian)
  e) Người dọn dẹp (cleaner)
  f) v.v
- Mỗi nhóm người dùng có những tính năng khác nhau, hệ thống xử lý dữ liệu với từng nhóm cũng khác nhau.
  - **Giải pháp là gì để phầm mềm xử lý dữ liệu với từng nhóm người theo cách khác nhau?**

Tại sao cần đến thừa kế

- (1) Tạo chung một cấu trúc "User", cấu trúc này có trường thông tin "type". Giải thuật xử lý có dạng:

```
switch (type){
    case STUDENT:{
        //Xử lý, nếu là sinh viên
    }
    case LECTURER:{
        //Xử lý, nếu là giảng viên
    }
    ...
};
```

Tại sao cần đến thừa kế

- (1) Tạo chung một cấu trúc "User", cấu trúc này có trường thông tin "type". Giải thuật xử lý có dạng:

  - Nhược điểm:
    - Code dài dòng
    - Khó thay đổi
    - Khó mở rộng
    - …

Tại sao cần đến thừa kế

- (2) Chia thành các nhóm nhỏ (lớp) nhỏ như: Student, Lecturer, … Các phương thức xử lý gắn kèm với từng loại.

```
class Student{
public:
      //Phương thức cho sinh viên
};
```

```
class Lecturer{
public:
      //Phương thức cho giảng viên
};
```

...

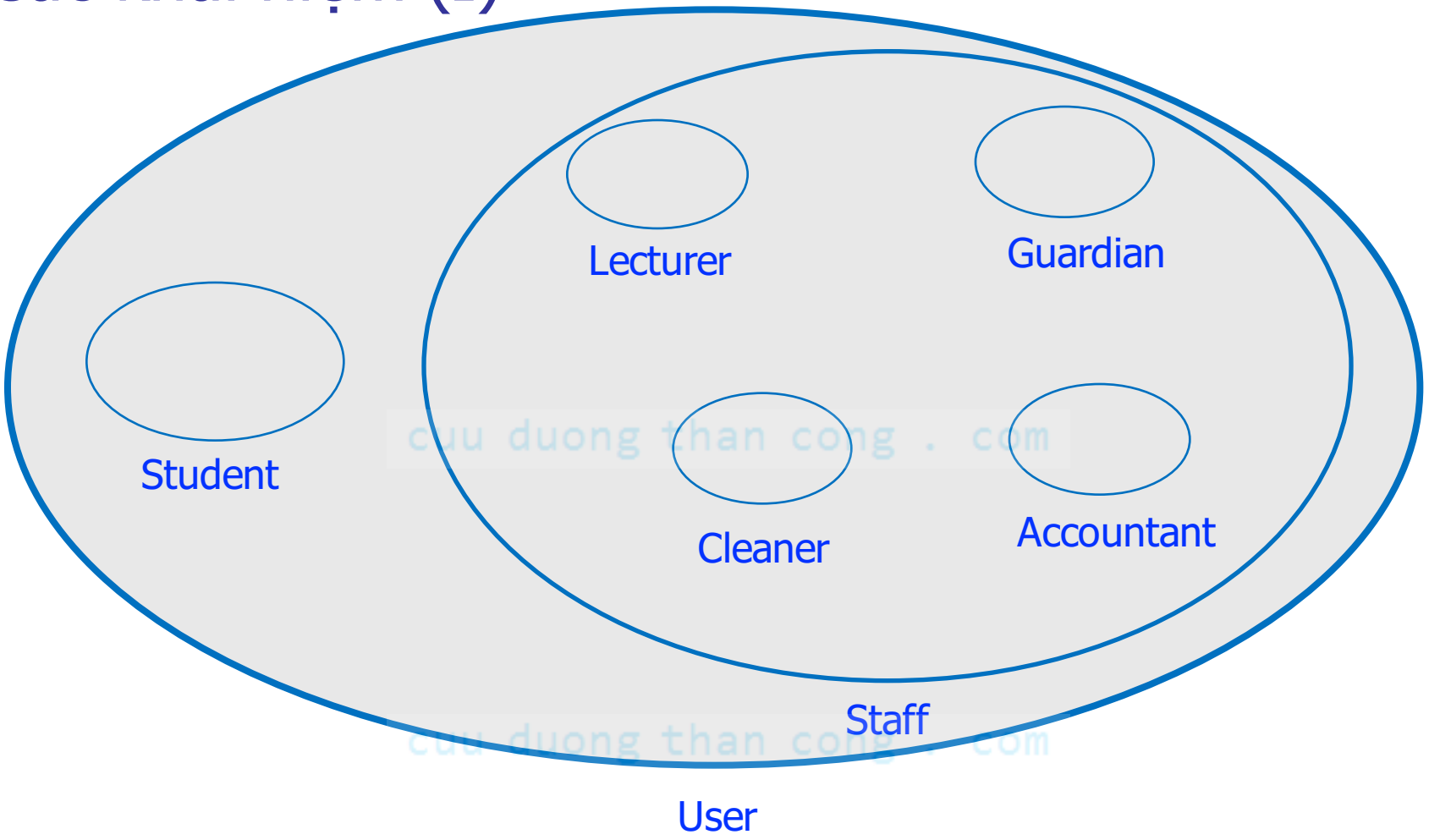
Tại sao cần đến thừa kế

- (2) Chia thành các nhóm (lớp) nhỏ như: Student, Lecturer, … Các phương thức xử lý gắn kèm với từng loại.
  - Nhược điểm:
    - Lặp lại code (**code duplication**)
      - Ví dụ:
        - Phương thức "getName"/"setName" (lấy/gán tên) đều phải hiện thực lại cho tất cả các lớp.
    - Khó bảo trì
    - Khó thay đổi hay nâng cấp
    - …

Tại sao cần đến thừa kế

- (3) Sử dụng tính năng thừa kế (inheritance)
  - Chia tập lớn thành các lớp nhỏ (lớp nhỏ, như giải pháp số 2)
  - Với các lớp có quan hệ "**is-a**", hãy khai báo thừa kế cho chúng
  - Tính năng thừa kế của ngôn ngữ lập trình (C++):
    - **Các lớp con có thể thừa kế các thành viên từ lớp cha.**
    - ➔ Tránh được sự lặp lại code nói trên.
    - **Các lớp cha có thể đại diện cho lớp con để xử lý một thông điệp** (tính *polymorphism*)
    - ➔ Dễ thiết kế + dễ thay đổi.

Các khái niệm (I)

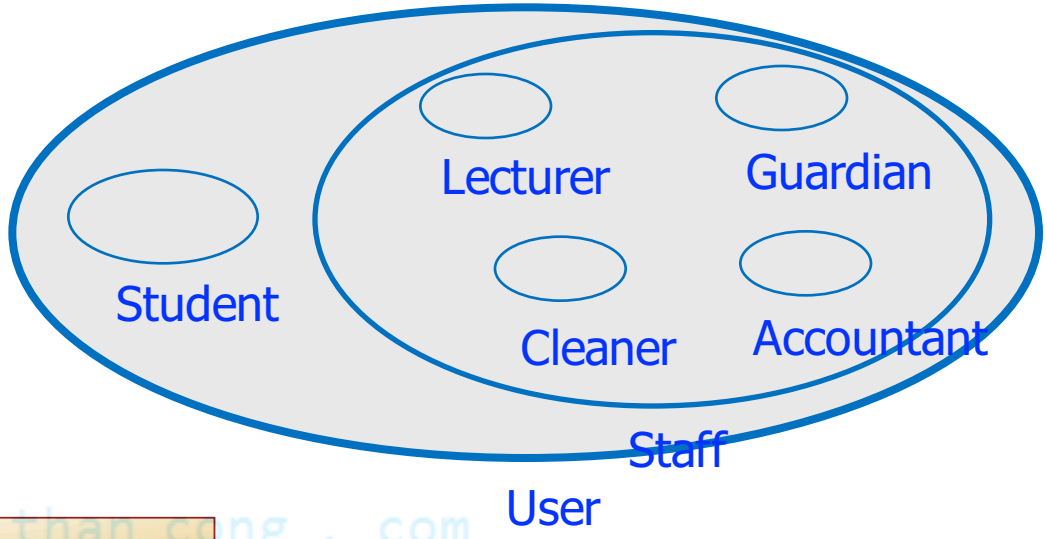


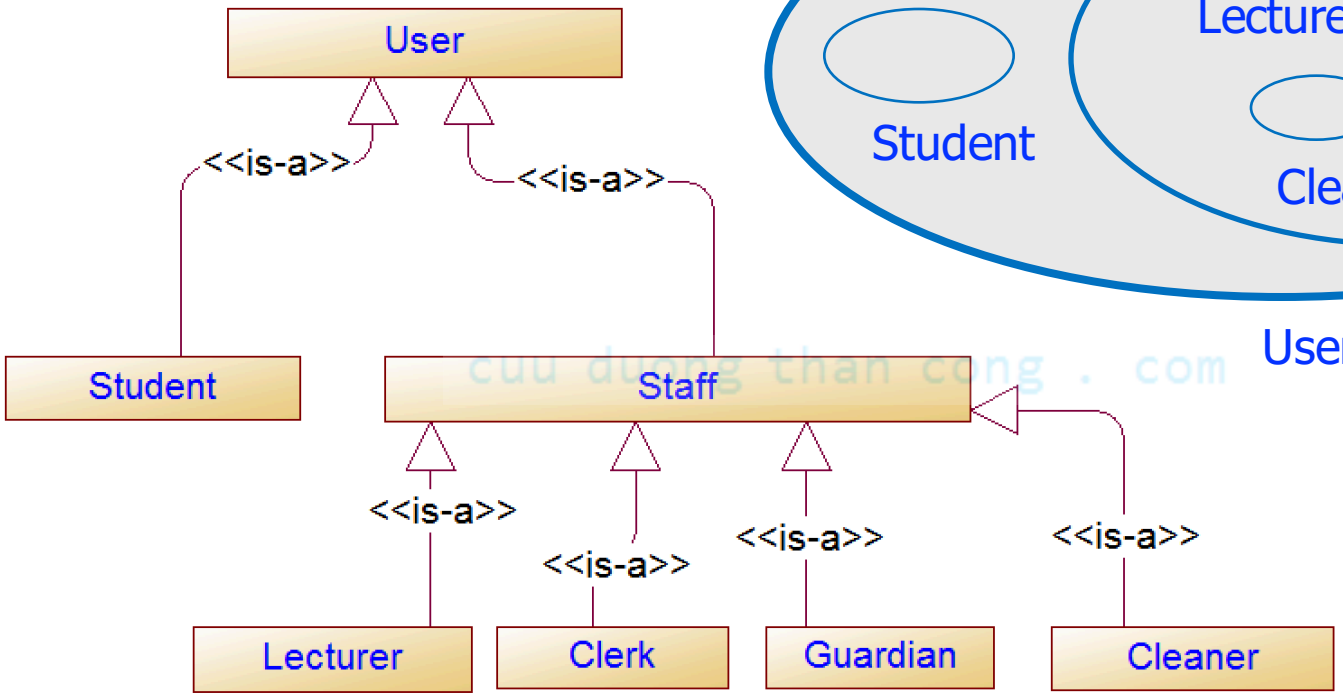
Chia nhỏ tập hợp "User" thành các tập hợp con

Các khái niệm (I)

Lý thuyết tập hợp	Hướng đối tượng	Thuật ngữ
Tập cha	Lớp cha	**Base class** Parent class Super-class
Tập con	Lớp con	**Derived class** Child-class Sub-class

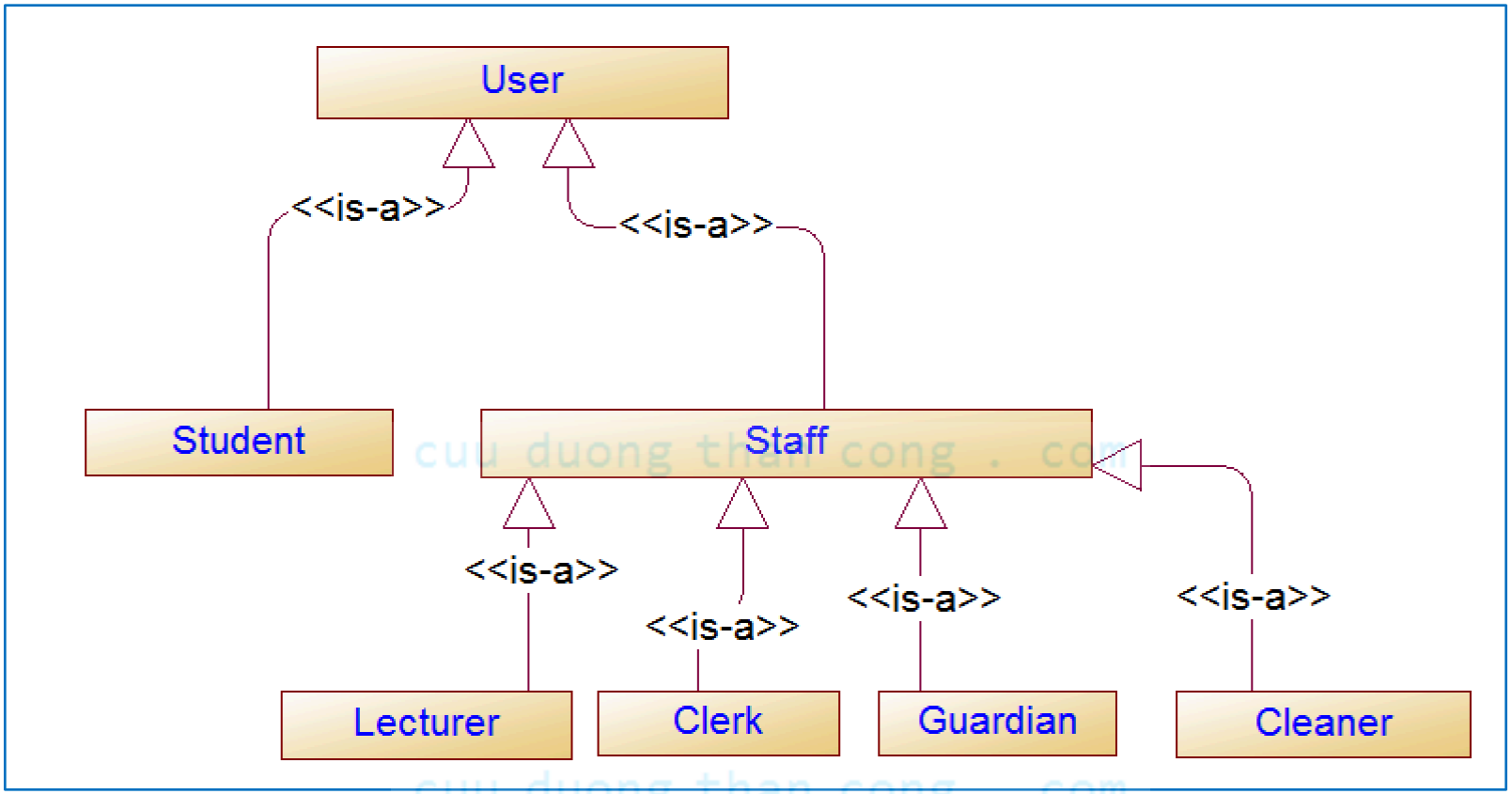
Các khái niệm (I)





Mô hình cây tương ứng

Các khái niệm (I)



- Các lớp: **Hình chữ nhật**, trong đó có các thuộc tính và phương thức (nếu cần)
- Quan hệ thừa kế: **mũi tên** từ lớp con đến lớp cha

Thừa kế là gì?

- Là một tính chất cho biết:
  - Một lớp con thể thừa hưởng các thành viên (thuộc tính + phương thức) có tính public và protected trong lớp cha.
    - **Không thừa hưởng thành viên có tính** private.
  - Cũng có nghĩa,
    - Lớp con không khai báo nhưng vẫn có các thành viên public và protected của lớp cha.
    - Với phương thức: có thể thừa kế và thay đổi nội dung của phương thức (xem: Phần định nghĩa phương thức - Overriding)

Thừa kế là gì?: Minh hoạ (I)

```
class X{
public:
        string sayHello(){
                return "Hello";
        }
};

class Y: public X{
}


int main(){
        Y obj;
        obj.sayHello();
        return 0;
}
```

Khai báo:
Lớp Y thừa kế lớp X

Chú ý: dấu hai chấm **":"** và từ khoá **public** trước tên lớp cha

Thừa kế là gì?: Minh hoạ (I)

```
class X{
public:
        string sayHello(){
                return "Hello";
        }
};
class Y: public X{
}


int main(){
        Y obj;
        obj.sayHello();
        return 0;
}
```

sayHello: được gọi với đối tượng obj, kiểu Y.

Trong phần mô tả của lớp Y, không có sayHello.

Nhưng, lớp Y thừa kế sayHello (public) của lớp X.

Do đó, biên dịch thành công và chạy được.
Xuất ra màn hình:

Hello

Thừa kế là gì?: Minh hoạ (II)

```
201 class ClassA{
202 private:
203     string name;
204 public:
205     string getName(){
206         return this->name;
207     }
208     void setName(string name){
209         this->name = name;
210     }
211 };
212 class ClassB: public ClassA{
213 public:
214     void print(){
215         cout << "My name is " << this->name << endl;
216     }
217 };
218
219
220 int main(){
221     ClassB obj;
222     obj.setName("Nguyen Van An");
223     cout << obj.name << endl;
224
```

(1) "name" là thuộc tính có tính private
➔ ClassB không thừa kế được từ ClassA

(2) Truy xuất đến "name" trong ClassB hay trong main (bên ngoài ClassA) → có lỗi biên dịch

std::string ClassA::name

Error: member "ClassA::name" (declared at line 203) is inaccessible

Thừa kế là gì?: Minh hoạ (III)

```
class ClassA{
private:
        string name;
public:
        string getName(){
                return this->name;
        }
        void setName(string name){
                this->name = name;
        }
};
class ClassB: public ClassA{
public:
        void print(){
                cout << "My name is " << this->getName() << endl;
        }
};
```

(1) getName(): có tính public trong ClassA =>ClassB thừa kế được phương thức này.

(2) getName(): có thể dùng được trong các phương thức của ClassB

Thừa kế là gì?: Minh hoạ (III)

(1) getName(): có tính public trong ClassA =>ClassB thừa kế được phương thức này.

```
int main(){
        ClassB obj;
        obj.setName("Nguyen Van An");
        cout << obj.getName() << endl;

        return 0;
}
```


(2) getName(): trong ClassB là được thừa kế từ ClassA.

Do từ khoá public trong dòng:


```
class classB: public ClassA{ …};
```

Nên getName() trong ClassB cũng có tính public → **dùng được ở trong hàm "main" hay bất kỳ đâu.**

Thừa kế là gì?: Minh hoạ (III)

(1) getName(): có tính public trong ClassA =>ClassB thừa kế được phương thức này.

```
int main(){
        ClassB obj;
        obj.setName("Nguyen Van An");
        cout << obj.getName() << endl;

        return 0;
}
```

(3) Dòng này sẽ bị báo lỗi nếu thay từ public trong dòng sau thành: protected hay private - Xem "các kiểu thừa kế".

```
class classB: public ClassA{ …};
```

Thừa kế là gì?: Minh hoạ (IV)

```
201 class ClassA{
202 protected:
203     string name;
204 public:
205     string getName(){
206         return this->name;
207     }
208     void setName(string name){
209         this->name = name;
210     }
211 };
212 class ClassB: public ClassA{
213 public:
214     void print(){
215         cout << "My name is " << this->name << endl;
216     }
217 };
218 int main(){
219     ClassB obj;
220     obj.setName("Nguyen Van An");
221     cout << obj.name << endl;
222
223     return 0;
224 }
```

(1): "name" có tính protected trong ClassA
➔ ClassB thừa kế được nó.

(2): Do đó, truy cập "name" trong ClassB là không bị báo lỗi

std::string ClassA::name

Error: member "ClassA::name" (declared at line 203) is inaccessible

Thừa kế là gì?: Minh hoạ (IV)

```
201 class ClassA{
202 protected:
203     string name;
204 public:
205     string getName(){
206         return this->name;
207     }
208     void setName(string name){
209         this->name = name;
210     }
211 };
212 class ClassB: public ClassA{
213 public:
214     void print(){
215         cout << "My name is " << this->name << endl;
216     }
217 };
218 int main(){
219     ClassB obj;
220     obj.setName("Nguyen Van An");
221     cout << obj.name << endl;
222
223     return 0;
224 }
```

(3): Trên dòng 212, khai kháo thừa kế có tính public ➔ "name" trong ClassB vẫn có tính protected như trong ClassA (Xem: các kiểu thừa kế).

➔ Truy cập "name" bên ngoài classB là có lỗi.

std::string ClassA::name

Error: member "ClassA::name" (declared at line 203) is inaccessible

Các kiểu thừa kế

Nếu ClassY thừa kế từ ClassX, thì có 3 dạng sau:

```
class ClassY: public ClassX{
};
```

← (phổ biến)

```
class ClassY: protected ClassX{
};
```

```
class ClassY: private ClassX{
};
```



```
class ClassY: virtual public ClassX{
};
```

Và các dạng tương tự: trình bày trong phần "Bài toán kim cương"

Các kiểu thừa kế

- Thừa kế public:

```
class ClassY: public ClassX{
};
```

  - Các thành viên (thuộc tính + phương thức) có tính public và protected có trong ClassX **vẫn giữa nguyên tính khả kiến** của nó trong ClassY.
    - **Đây là dạng phổ biến nhất trong 3 dạng**

  - Lưu ý: ClassY không thể thừa kế các thành viên (thuộc tính + phương thức) có tính private từ ClassX

Các kiểu thừa kế

- Thừa kế protected:

```
class ClassY: protected ClassX{
};
```

  - Các thành viên (thuộc tính + phương thức) có tính public và protected có trong ClassX **đều có tính protected** trong ClassY.
    - ➔ Thành viên có tính public trong ClassX: sẽ không thể truy cập được từ đối tượng kiểu ClassY.

  - Lưu ý: ClassY không thể thừa kế các thành viên (thuộc tính + phương thức) có tính private từ ClassX

Các kiểu thừa kế: Minh hoạ (I)

```
class ClassA{
protected:
        string name;
public:
        string getName(){
                return this->name;
        }
        void setName(string name){
                this->name = name;
        }
};
class ClassB: protected ClassA{
public:
        void print(){
                cout << "My name is " << this->getName() << endl;
        }
};
```

(1) Thừa kế kiểu protected:
➔ Cả setName() và getName():
sẽ có tính protected trong ClassB.

Các kiểu thừa kế: Minh hoạ (I)

```
218  int main(){
219      ClassA ob1;
220      ob1.setName("Phan Thi Nhan");
221      cout << ob1.getName() << endl;
222
223      ClassB ob2;
224      ob2.setName("Nguyen Van An");
225      cout << ob2.getName() << endl;
226
227      return 0;
228  }
```

➔ Cả setName() và getName(): Vẫn được nhìn thấy thông qua ClassA

```
std::string ClassA::getName()

Error: function "ClassA::getName" (declared at line 205) is inaccessible
```

➔ Cả setName() và getName():
Không được nhìn thấy thông qua ClassB
(vì ClassB thừa kế kiểu protected từ ClassA)

Các kiểu thừa kế

- Thừa kế private:

```
class ClassY: private ClassX{
};
```

  - Các thành viên (thuộc tính + phương thức) có tính public và protected có trong ClassX **đều có tính private** trong ClassY.
    - ➔ Thành viên có tính public trong ClassX: sẽ không thể truy cập được từ đối tượng kiểu ClassY.

  - Lưu ý: ClassY không thể thừa kế các thành viên (thuộc tính + phương thức) có tính private từ ClassX

Các kiểu thừa kế: Minh hoạ (II)

```
class ClassA{
protected:
        string name;
public:
        string getName(){
                return this->name;
        }
        void setName(string name){
                this->name = name;
        }
};
class ClassB: private ClassA{
public:
        void print(){
                cout << "My name is " << this->getName() << endl;
        }
};
```

(1) Thừa kế kiểu private:
➔ Cả setName() và getName():
sẽ có tính private trong ClassB.

`this->getName()`: thừa kế từ ClassA.
➔ Gọi được!
ClassB dùng được getName. Nhưng các lớp con của ClassB không dùng được getName được nữa – vì nó đã có tính private trong ClassB

Các kiểu thừa kế: Minh hoạ (II)

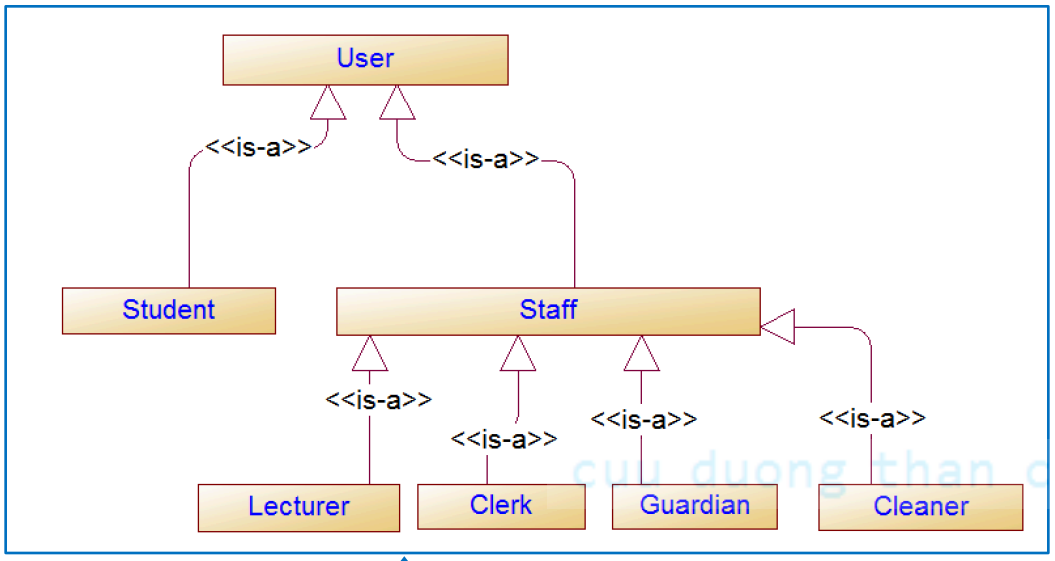
```
218  int main(){
219      ClassA ob1;
220      ob1.setName("Phan Thi Nhan");
221      cout << ob1.getName() << endl;
222
223      ClassB ob2;
224      ob2.setName("Nguyen Van An");
225      cout << ob2.getName() << endl;
226
227      return 0;
228  }
```

➔ Cả setName() và getName(): Vẫn được nhìn thấy thông qua ClassA

```
std::string ClassA::getName()

Error: function "ClassA::getName" (declared at line 205) is inaccessible
```

➔ Cả setName() và getName():
Không được nhìn thấy thông qua ClassB
(vì ClassB thừa kế kiểu private từ ClassA)

Thiết kế các lớp (I)




```
class User{
};
class Student: public User{
};
class Staff: public User{
};
class Lecturer: public Staff{
};
class Clerk: public Staff{
};
class Guardian: public Staff{
};
class Cleaner: public Staff{
};
```

Tương đương giữa code và sơ đồ

Thiết kế các lớp (I)

Nếu tách riêng phần khai báo và phần định nghĩa vào hai tập tin: *.h và *.cpp

Xem kết quả sau.

Thiết kế các lớp (I)

User.h

```
#ifndef USER_H
#define USER_H
class User{
};
#endif
```

Giúp tránh lỗi "tái định nghĩa" một khái niệm (redefinition); khái niệm ở đây là lớp có tên "User"

Student.h

```
#ifndef STUDENT_H
#define STUDENT_H

#include "User.h"
class Student : public User{
};
#endif
```

Từ "public" cho biết:
Lớp "Student" **không thay đổi tính khả kiến** của các biến/hàm thành viên trong lớp cha (User)

Dấu hai chấm ":" cho biết:
Lớp "Student" **thừa kế** lớp "User"

Thiết kế các lớp (I)

User.h

```
#ifndef USER_H
#define USER_H
class User{
};
#endif
```

Giúp tránh lỗi "tái định nghĩa" một khái niệm (redefinition); khái niệm ở đây là lớp có tên "User"

Student.h

```
#ifndef STUDENT_H
#define STUDENT_H

#include "User.h"
class Student : public User{
};
#endif
```

Phải chèn "header" file của lớp cha tại đây để bộ biên dịch biết được danh hiệu "User" là gì.

Nếu không, có lỗi "undefined data-type"

Thiết kế các lớp (I)

User.h

```
#ifndef USER_H
#define USER_H
class User{
};
#endif
```

Giúp tránh lỗi "tái định nghĩa" một khái niệm (redefinition); khái niệm ở đây là lớp có tên "User"

Staff.h

```
#ifndef STAFF_H
#define STAFF_H

#include "User.h"
class Staff: public User{
};
#endif
```

Thiết kế các lớp (I)

Tương tự:

Lecturer.h

```
#ifndef LECTURER_H
#define LECTURER_H
#include "Staff.h"

class Lecturer : public Staff{
};
#endif
```

Guardian.h

```
#ifndef GUARDIAN_H
#define GUARDIAN _H
#include "Staff.h"

class Guardian : public Staff{
};
#endif
```

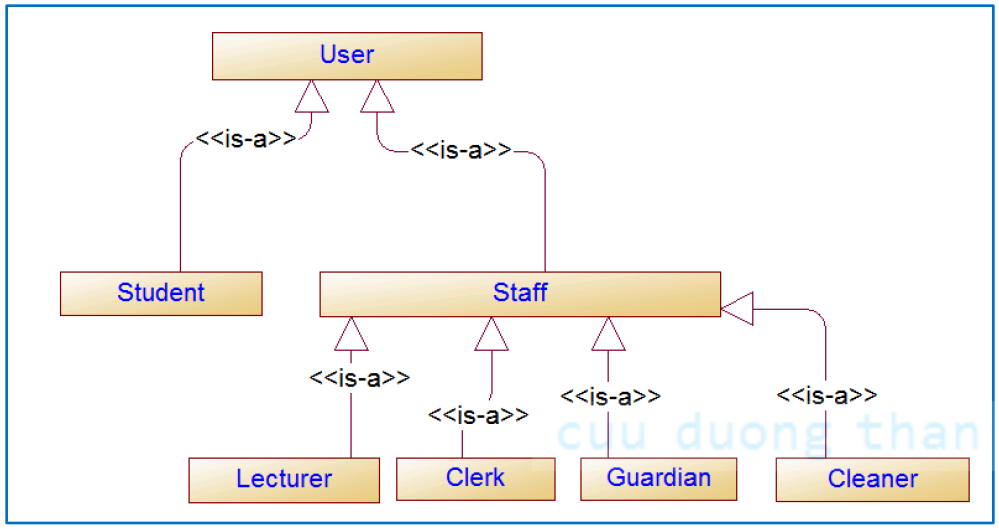
Clerk.h

```
#ifndef CLERK_H
#define CLERK_H
#include "Staff.h"

class Clerk : public Staff{
};
#endif
```

Cleaner.h

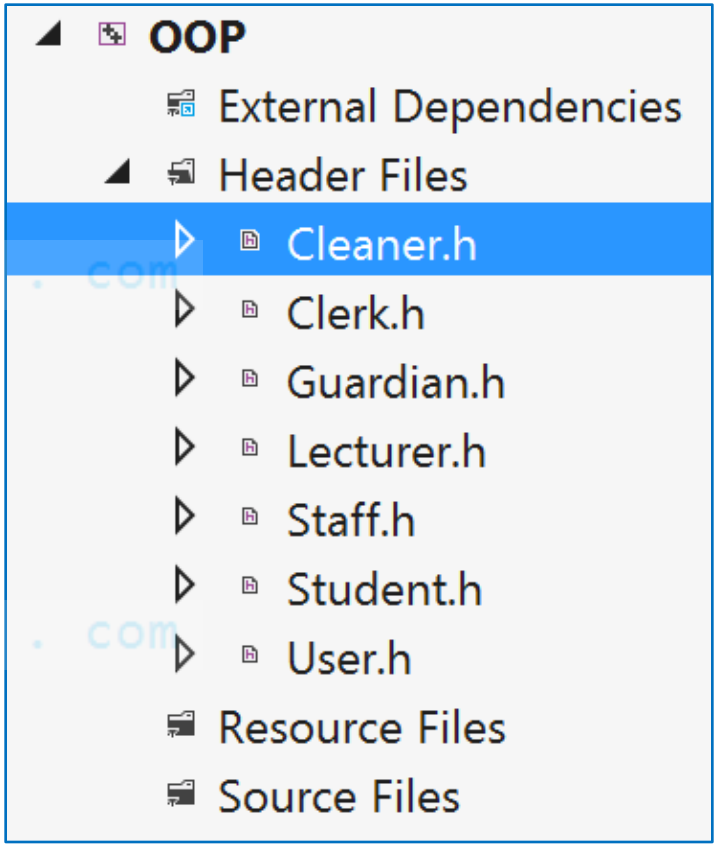
```
#ifndef CLEANER_H
#define CLEANER_H
#include "Staff.h"

class Cleaner : public Staff{
};
#endif
```

Thiết kế các lớp (I)

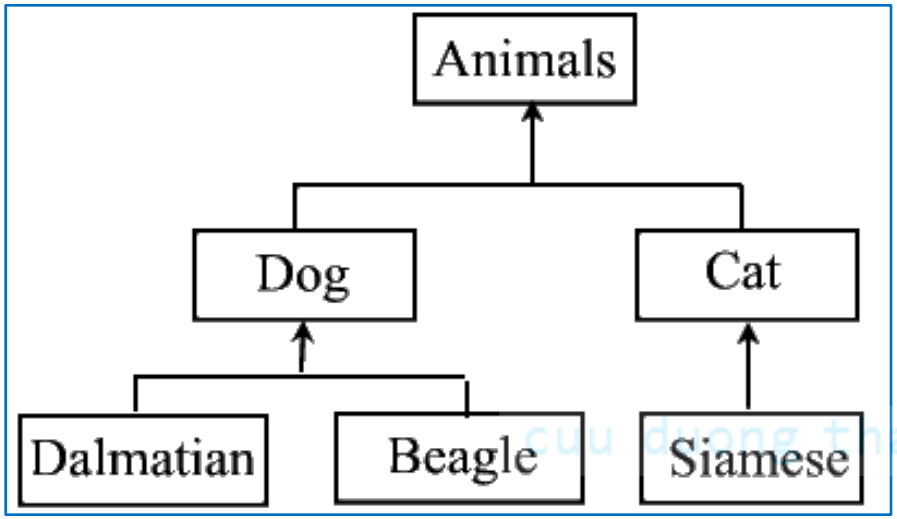


Sau khi sinh code, dự án có dạng:

(chưa có *.cpp, các lớp đều rỗng)



Thiết kế các lớp (II)

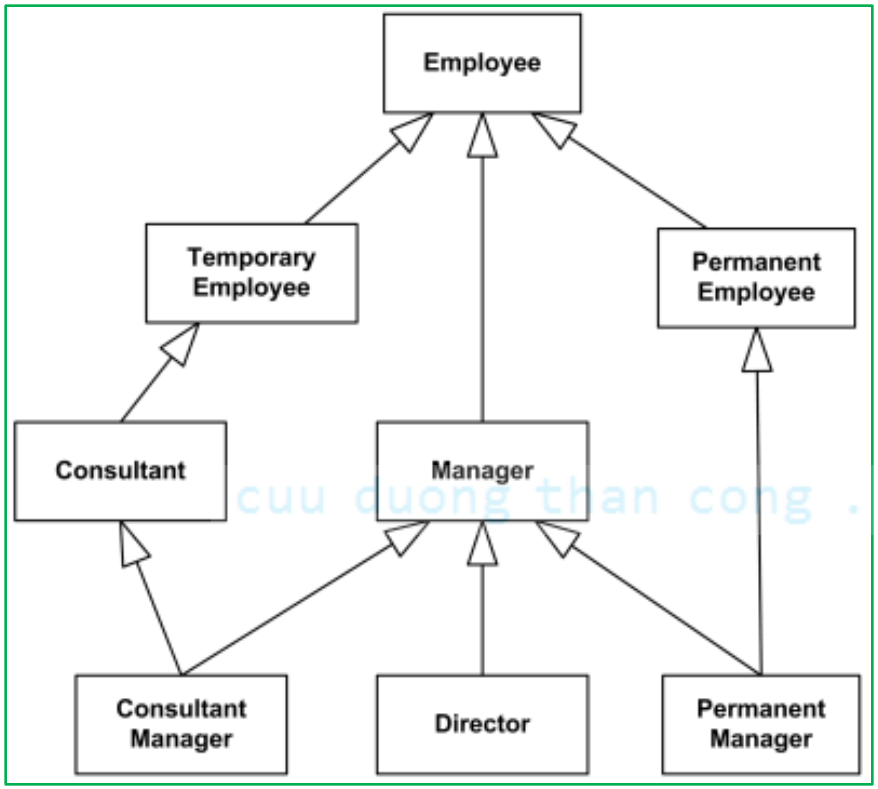



```
class Animals{
};
class Dog: public Animals{
};
class Cat: public Animals{
};
class Dalmatian: public Dog{
};
class Beagle: public Dog{
};
class Siamese: public Cat{
};
```

Tương đương giữa code và sơ đồ

Nguồn hình: https://www.usna.edu/Users/cs/schulze/ic211/classes/L11/Class.html

Thiết kế các lớp (III)



http://www.uml-diagrams.org/generalization.html

Đa thừa kế: lớp **ConsultantManager** và **PermanentManager**, có đến 2 lớp cha.

Trường hợp tổng quát: có thể có nhiều cha.

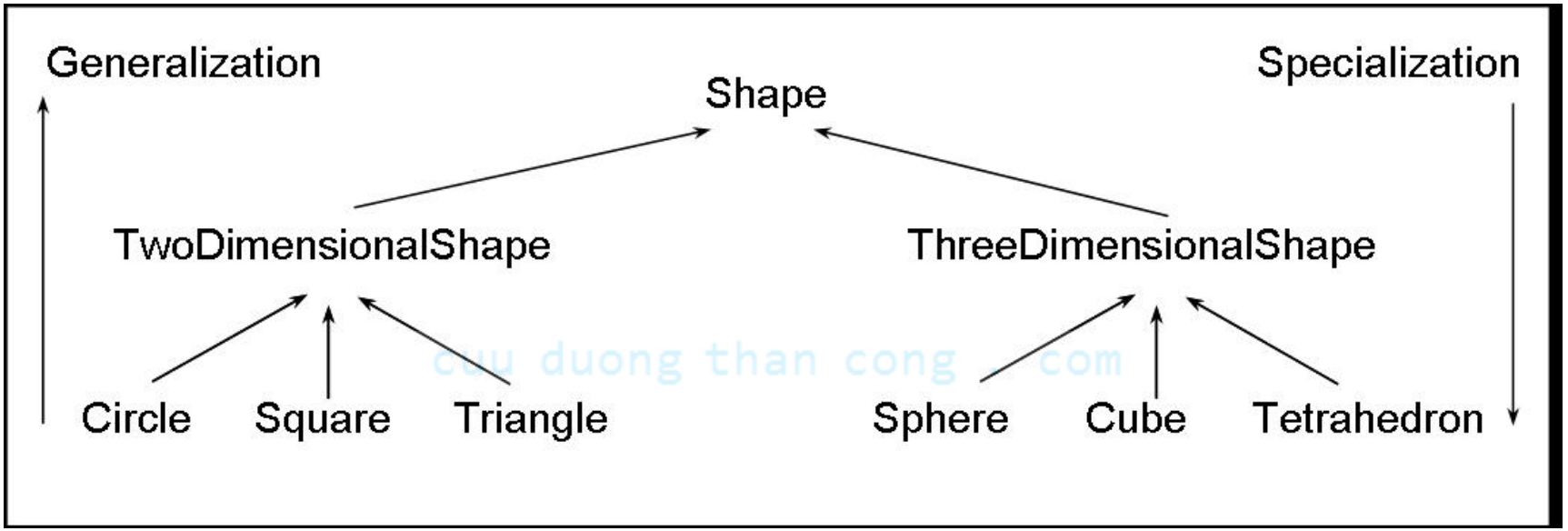
Thiết kế các lớp (III)

```
class Employee{
};
class TemporaryEmployee: public virtual Employee{
};
class PermanentEmployee: public virtual Employee{
};
class Consultant: public TemporaryEmployee{
};
class Manager: public virtual Employee{
};
class ConsultantManager: public Consultant, public Manager{
};
class Director: public Manager{
};
class PermanentManager: public Manager, public PermanentEmployee{
};
```

Dùng dấu phẩy ",” để liệt kê các lớp cha

Tương đương giữa code và sơ đồ, từ khoá virtual được giải thích sau

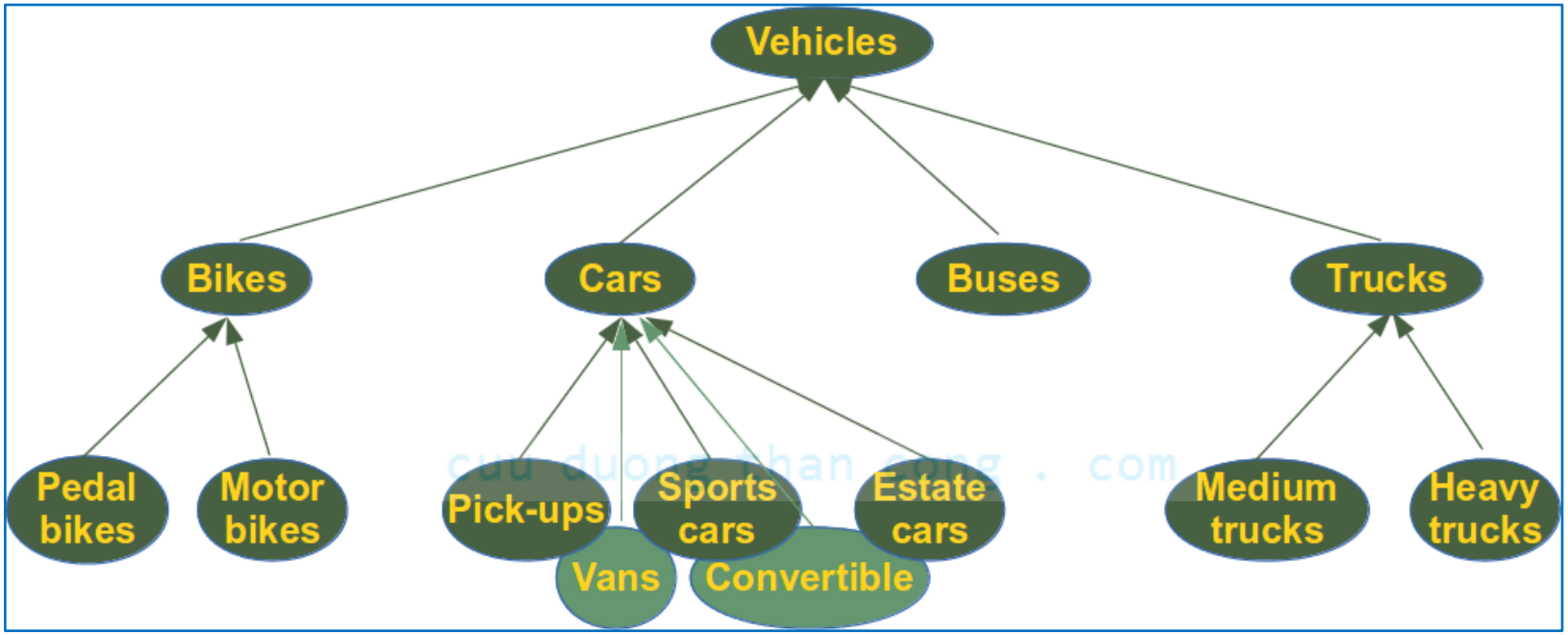
Thiết kế các lớp: Bài tập 1



https://www.usna.edu/Users/cs/schulze/ic211/classes/L11/Class.html

- Chuyển sang code C++ từ sơ đồ

Thiết kế các lớp: Bài tập 2



http://www.python-course.eu/python3_inheritance.php

- Chuyển sang code C++ từ sơ đồ

Thiết kế các lớp (IV)

- Bài toán:
  - Xét bài toán về người dùng ở trường đại học - Xem: "**Thiết kế các lớp (I)**".
  - Thực tế yêu cầu:
    - Mỗi người dùng điều có tên.

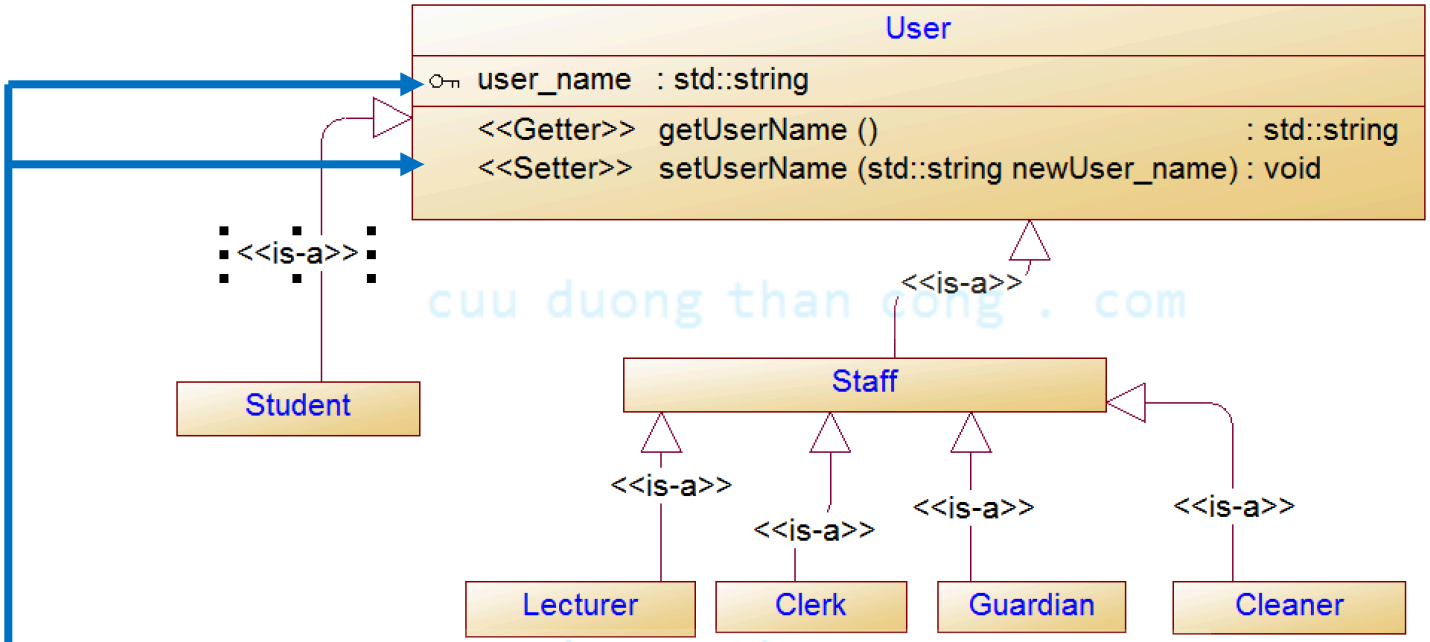
  - Lời giải:
    - Bổ sung một thuộc tính "name" vào đối tượng.

Thiết kế các lớp (IV)

- Bài toán:
  - Xét bài toán về người dùng ở trường đại học - Xem: "**Thiết kế các lớp (I)**".
  - Các câu hỏi khác:
    - Bổ sung vào lớp nào? tính khả kiến là gì?
    - Lời giải:
      - Bổ sung vào lớp "User", và sẽ dùng chung cho các lớp khác là lớp con của "User"
      - ➔ Tính khả kiến được chọn là: protected thay cho private hay public.
      - ➔ Bổ sung một getter và setter cho thuộc tính này để cho phép các lớp khác không phải con của User chạm được đến thuộc tính "name"

Thiết kế các lớp (IV)

Sơ đồ sau khi bổ sung thuộc tính + getter/setter



- Biểu tượng ổ khoá đã khoá (-): private
- Để trống (+): public
- Biểu tượng chìa (#): protected

Thiết kế các lớp (IV): code C++

User.h

```
#ifndef USER_H
#define USER_H

#include <string>
class User
{
public:
    std::string getUserName(void);
    void setUserName(std::string newUserName);

protected:
    std::string userName;
};
#endif
```

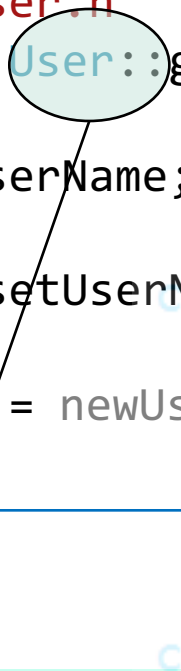

Getter/Setter, tính public

thuộc tính userName, Tính protected

Thiết kế các lớp (IV): code C++

User.cpp

```
#include "User.h"
std::string User::getUserName(void)
{
    return userName;
}
void User::setUserName(std::string newUserName)
{
    userName = newUserName;
}
```



"User::" dùng khi phương thức được định nghĩa bên ngoài phạm vi class

(:: là toán tử phân giải tầm vực)

Thiết kế các lớp (IV): code C++

main.cpp

```
#include <iostream>
#include "Student.h"
#include "Guardian.h"
#include "Lecturer.h"
using namespace std;
int main(){
        Student a;
        a.setUserName("Nguyen Ngoc Anh");
        Guardian b;
        b.setUserName("Le Van Bao");
        Lecturer c;
        c.setUserName("Nguyen Minh Phuong");

        cout << "Student: " << a.getUserName() << endl;
        cout << "Guardian: " << b.getUserName() << endl;
        cout << "Lecturer: " << c.getUserName() << endl;
        return 0;
}
```

Thiết kế các lớp (IV): code C++

Chương trình in ra kết quả sau:

```
Student: Nguyen Ngoc Anh
Guardian: Le Van Bao
Lecturer: Nguyen Minh Phuong
```


Lưu ý: Gọi setName() và getName() được, từ các đối tượng a,b, và c – vì: các lớp Student, Guardian, và Lecturer thừa kế chúng từ lớp User.

Khởi tạo lớp cha từ lớp con

Xét câu lệnh:

```
Lecturer x;
```

Câu hỏi: Bộ thực thi sẽ làm gì khi gặp câu lệnh này?

Trả lời: Bộ thực thi sẽ tạo ra một đối tượng, đặt tên là x. Cũng có nghĩa, nó tạo ra một vùng nhớ để chứa đối tượng "**Lecturer**" và đặt tên cho vùng nhớ đó là x.

Khởi tạo lớp cha từ lớp con

Xét câu lệnh:

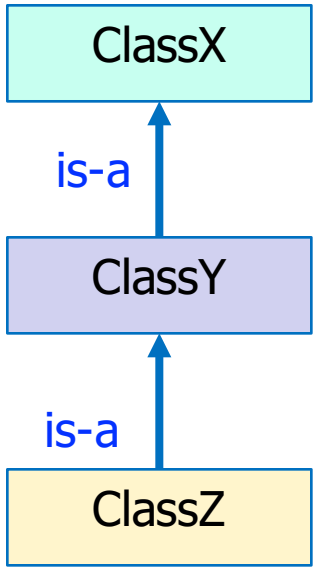
```
Lecturer x;
```

Câu hỏi: Vì vùng nhớ của x có gói luôn cả các đối tượng thuộc lớp cha của "**Lecturer**" là "**Staff**" và "**User**", thứ tự tạo vùng nhớ này là như thế nào?

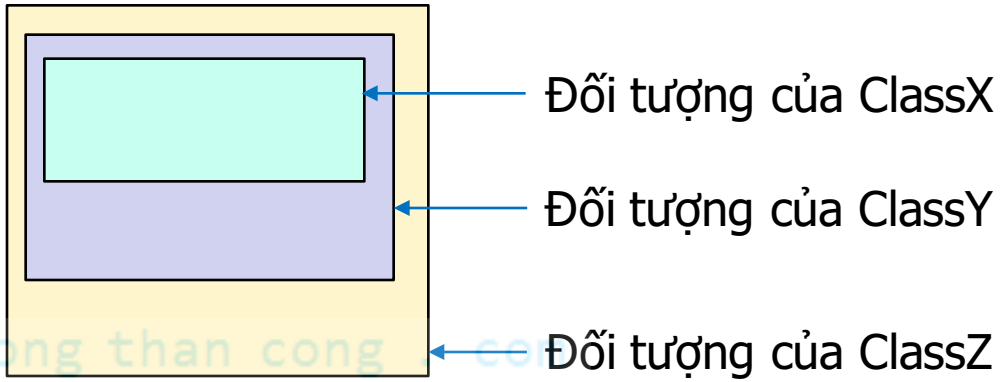
Trả lời: Thự tự đó là

a) Tạo đối tượng kiểu "User", gọi hàm khởi tạo của User
b) Tạo đối tượng kiểu "Staff", gói đối tượng ở bước a) vào trong, và gọi hàm khởi tạo của "Staff".
c) Tạo đối tượng kiểu "Lecturer", gói đối tượng ở bước b) vào trong, và gọi hàm khởi tạo của Lecturer.

Khởi tạo lớp cha từ lớp con: Minh hoạ (I)



Cây thừa kế



Chỉ 01 dòng lệnh:

```
ClassZ v;
```

Đã tạo ra một đối tượng kiểu ClassZ, **gói trong đó** cả đối tượng của ClassY và ClassX

Khởi tạo lớp cha từ lớp con: Minh họa (I)

```
#include <iostream>
using namespace std;
class ClassX{
public:
	ClassX(){ cout << "Constructor of ClassX" << endl; };
};
class ClassY: public ClassX{
public:
	ClassY(){ cout << "Constructor of ClassY" << endl; };
};
class ClassZ: public ClassY{
public:
	ClassZ(){ cout << "Constructor of ClassZ" << endl; };
};

int main(){
	ClassZ v;
	return 0;
}
```

Dòng này in ra gì?

Khởi tạo lớp cha từ lớp con: Minh hoạ (I)

```
Constructor of ClassX
Constructor of ClassY
Constructor of ClassZ
```


Chương trình xuất ra **theo thứ tự gọi hàm khởi tạo** cho ClassX, ClassY, cuối cùng là cho ClassZ

Khởi tạo lớp cha từ lớp con: Minh hoạ (II)

```
#include <iostream>

using namespace std;
class ClassX{
private:
        string name;
public:
        ClassX(string name){
                this->name = name;
                cout << "Constructor of ClassX" << endl;
        };
};
```


Khi ClassX có hàm khởi tạo cần đối số

Khởi tạo lớp cha từ lớp con: Minh hoạ (II)

```
class ClassY: public ClassX{
public:
        ClassY(string name): ClassX(name){
                cout << "Constructor of ClassY" << endl;
        };
};
class ClassZ: public ClassY{
public:
        ClassZ(string name): ClassY(name){
                cout << "Constructor of ClassZ" << endl;
        };
};

int main(){
        ClassZ v("Nguyen Ngoc An");
        return 0;
}
```

Cách gọi hàm khởi tạo của lớp cha, từ hàm khởi tạo của lớp con.

Chú ý:

- Dấu hai chấm ":"
- Và, vị trí là đứng liền trước thân hàm khởi tạo

Khởi tạo lớp cha từ lớp con: Minh hoạ (III)

```
class ClassT: public ClassY{
private:
        const int const_value;
        int ID;
        string str_value;
public:
        ClassT(string name, string value):
                ClassY(name), const_value(100), str_value(value)

        {
                cout << "Constructor of ClassT" << endl;
        }
};
```

Gọi hàm khởi tạo của lớp cha

Khởi tạo hằng (bắt buộc)

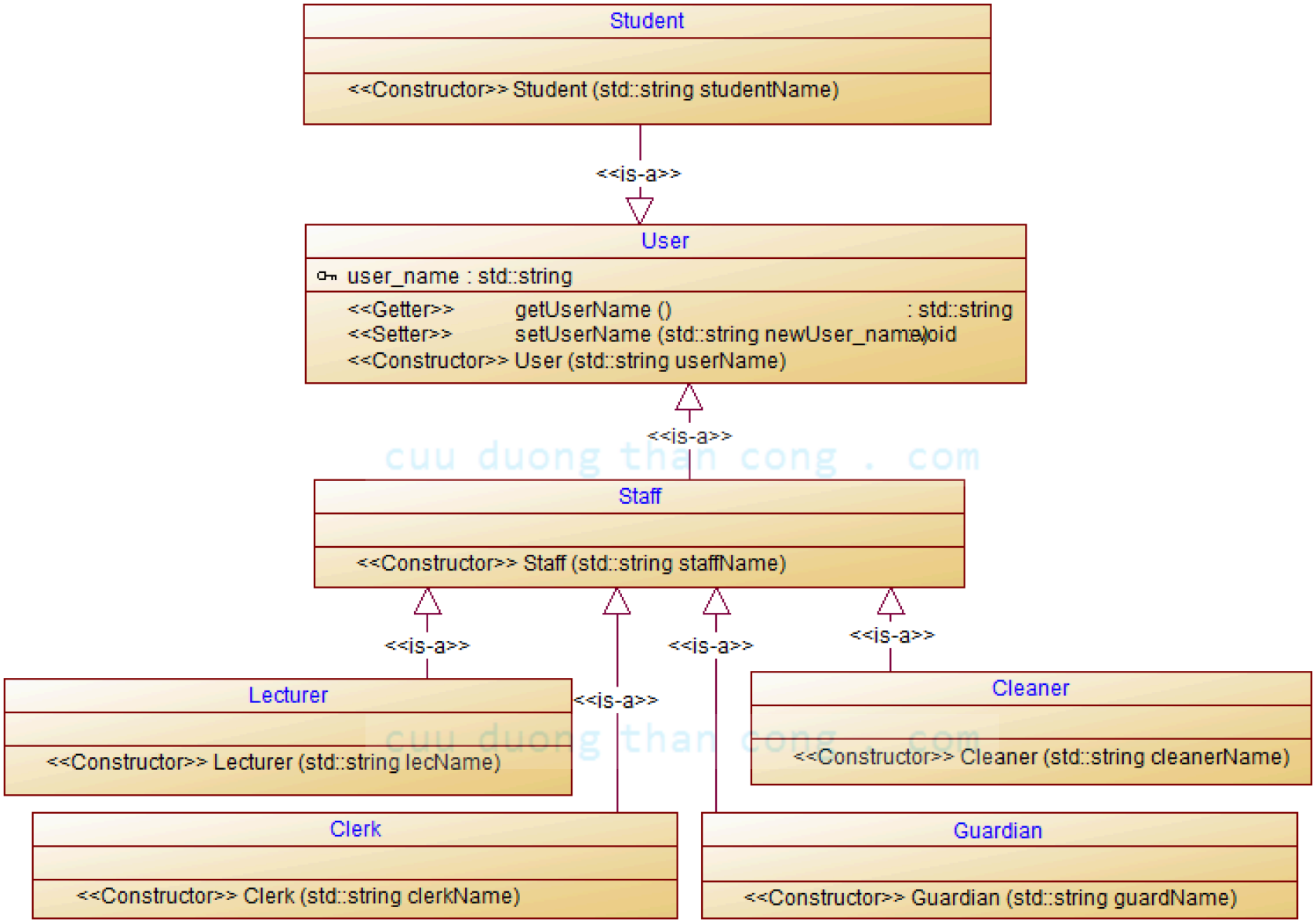
Khởi tạo biến thành viên

Khởi tạo lớp cha từ lớp con: Bài tập

- Bổ sung mã nguồn để cho phép người lập trình có thể truyền tên của người dùng vào hàm khởi tạo của tất cả các nhóm người dùng, như ví dụ:

```
Student a("Nguyen Ngoc Anh");
Guardian b("Le Van Bao");
Lecturer c("Nguyen Minh Phuong");
```

- Gợi ý:
  - Xem hình vẽ sau
  - Sử dụng cách khởi động như trong trước



Định nghĩa lại phương thức (Overriding)

- Overriding là gì?
  - Là khả năng mà một lớp con có thể **định nghĩa lại** (override) những phương thức mà nó thừa kế được từ lớp cha
  - Định nghĩa lại để làm gì?
    - Để phản ánh dữ liệu mà nó đang giữ
    - Để phù hợp hơn với kiểu hiện tại

Định nghĩa lại phương thức: Minh hoạ

```
class ClassA{
private:
        string name;
public:
        ClassA(string name){
                this->name = name;
        }
        string getName(){
                return this->name;
        }
        void setName(string name){
                this->name = name;
        }
};
```


ClassA: có phương thức getName(): trả về dữ liệu "name" nó đang giữ

Định nghĩa lại phương thức: Minh hoạ

```
//Need: #include <algorithm>
class ClassB: public ClassA{
public:
        ClassB(string name): ClassA(name){
        }

        string getName(){
                std::string str_temp = ClassA::getName();
                std::transform(str_temp.begin(),
                        str_temp.end(),
                        str_temp.begin(),
                        ::toupper);
                return str_temp;
        }
};
```

ClassB thừa kế ClassA:

- Đã có sẵn getName():
- Nhưng **ClassB muốn** tên được trả về phải được chuẩn hoá (đổi sang chữ HOA)

Định nghĩa lại phương thức: Minh hoạ

```
//Need: #include <algorithm>
class ClassB: public ClassA{
        ...
        string getName(){
                std::string str_temp = ClassA::getName();
                std::transform(str_temp.begin(),
                        str_temp.end(),
                        str_temp.begin(),
                        ::toupper);
                return str_temp;
        }
};
```

ClassA::getName():
gọi lại getName() trong lớp cha.

Chuyển sang chữ hoa, dùng hàm transform.

Định nghĩa lại phương thức: Minh hoạ

```
int main(){
        ClassB obj("nguyen van an");

        cout << obj.ClassA::getName() << endl;

        cout << obj.getName() << endl;

        return 0;
}
```

Trong đối tượng obj: **có hai phương thức getName()**: một cho đối tượng từ lớp cha (ClassA), và một cho classB.

Cách này dùng để gọi getName() cho lớp ClassA → in ra chữ thường

Gọi getName() của ClassB → in ra chữ HOA.
Xem kết quả sau:

Kết quả in ra màn hình:

```
nguyen van an
NGUYEN VAN AN
```

Định nghĩa lại phương thức: Minh hoạ (II)

```
class ClassX{
public:
        void display(){
                cout << "ClassX" << endl;
        }
};

class ClassY: public ClassX{
public:
        void display(){
                cout << "ClassY" << endl;
        }
};
```


Định nghĩa lại phương thức "display" của lớp cha

Định nghĩa lại phương thức: Minh hoạ (II)

```
class ClassZ: public ClassY{
public:
        void display(){
                cout << "ClassZ" << endl;
        }
};

class ClassT: public ClassZ{
public:
        void display(){
                cout << "ClassT" << endl;
        }
};
```


Định nghĩa lại phương thức "display" của lớp cha

Định nghĩa lại phương thức: Minh hoạ (II)

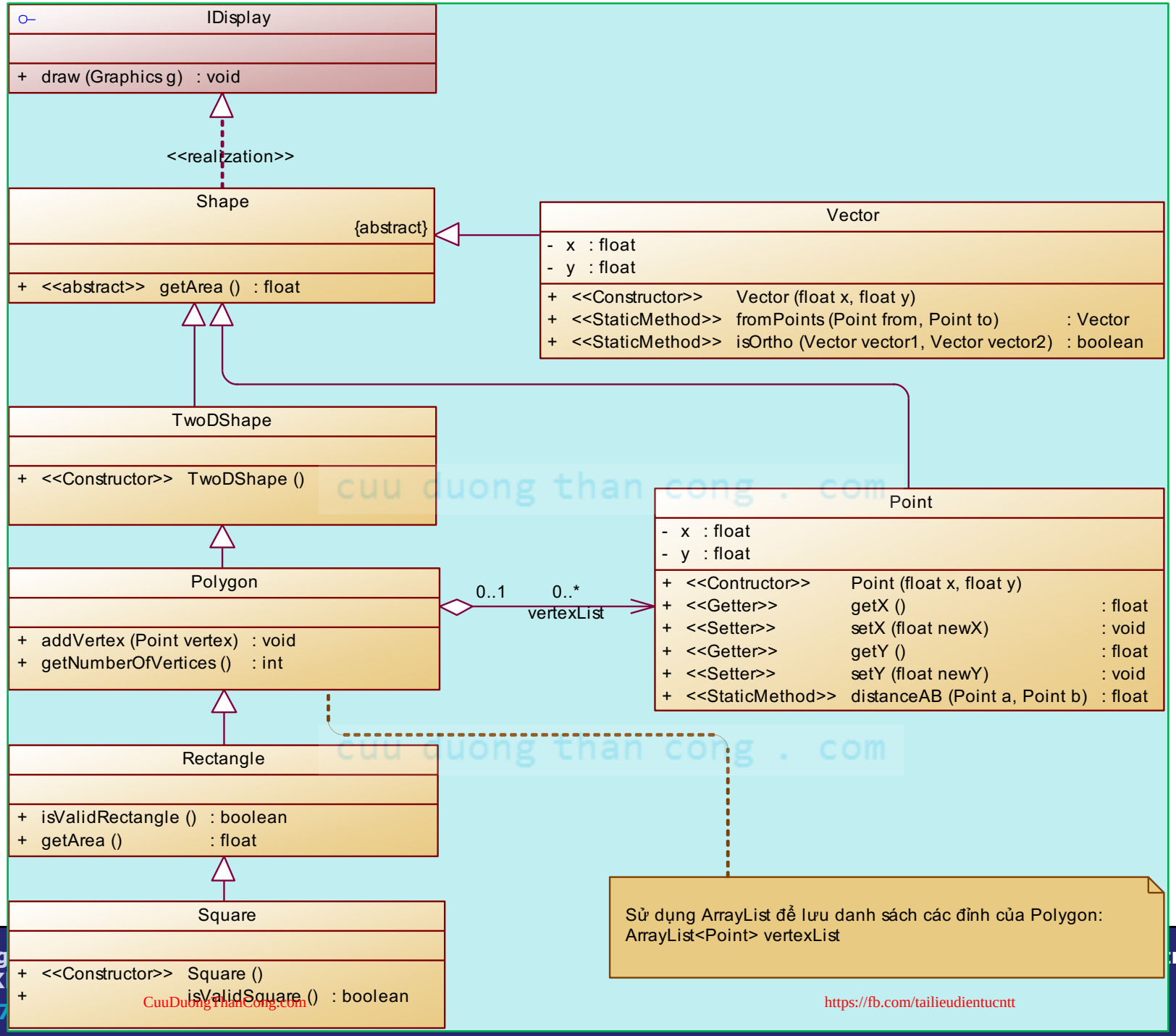
```
int main(){
        ClassT obj;

        obj.ClassX::display();
        obj.ClassY::display();
        obj.ClassZ::display();
        obj.ClassT::display();

        return 0;
}
```


Đối tượng "obj" chứa bên trong các đối tượng của kiểu: ClassX, ClassY, ClassZ, và ClassT. Do đó, **có đến 4 phương thức display.**

Trên đây là cách gọi từng phương thức trong đó.
Kết quả in ra là gì?

Thiết kế các lớp (II)

- Bài toán:
  - Viết chương trình cho phép vẽ các đối tượng hình học 2D đơn giản:
    - a) Hình Chữ nhật
    - b) Hình vuông
    - c) Hình đa giác
    - d) Điểm
    - c) Véc-tơ
- Câu hỏi:
  - Nên có những lớp gì? Quan hệ ra sao trong bài toàn này?
  - Xem bản thiết kế sau.
    - Trong đó có hai khái niệm chưa giải thích, đến thời điểm này:
      - Interface
      - Virtual Method + Abstract method



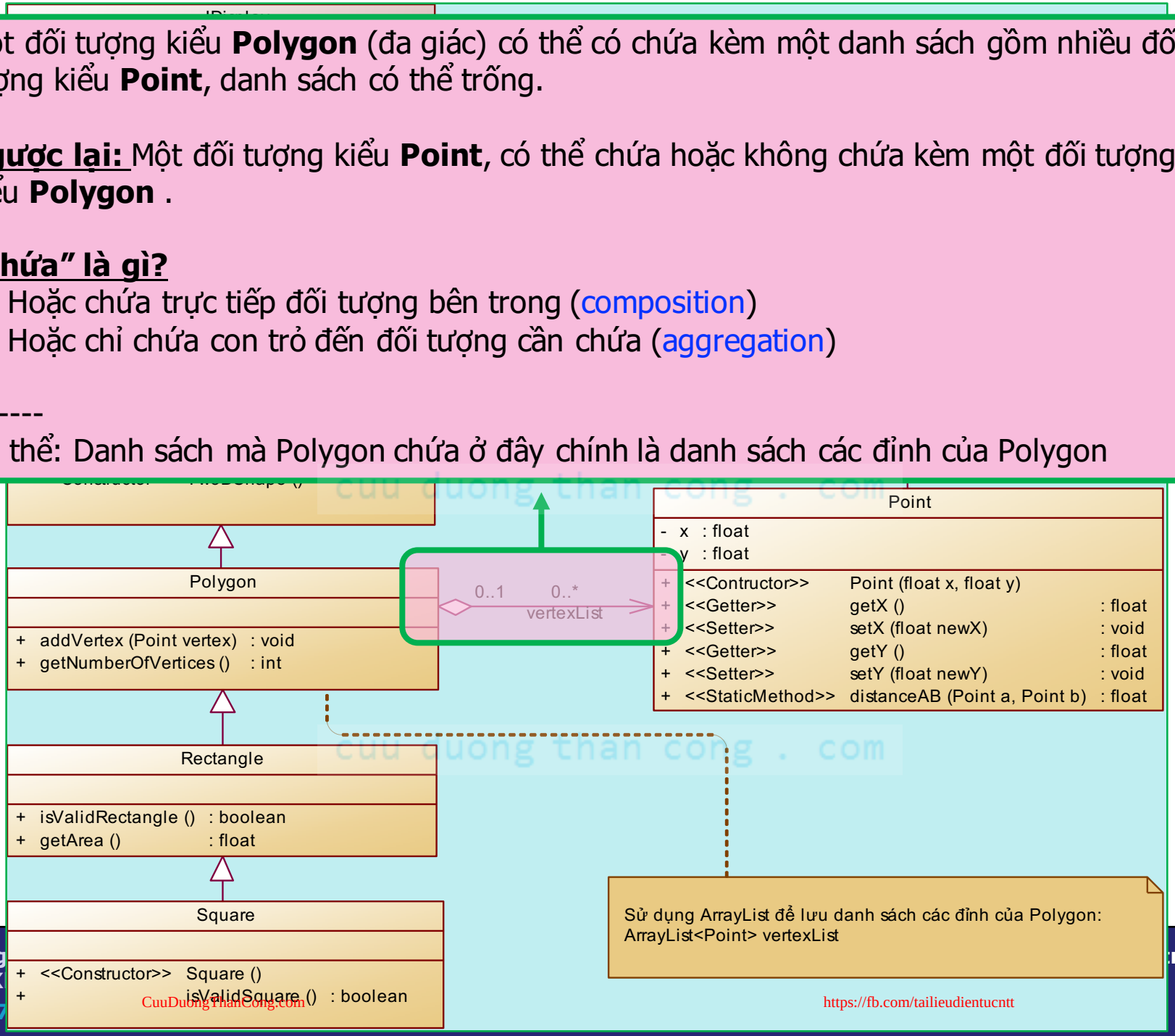
Một đối tượng kiểu **Polygon** (đa giác) có thể có chứa kèm một danh sách gồm nhiều đối tượng kiểu **Point**, danh sách có thể trống.

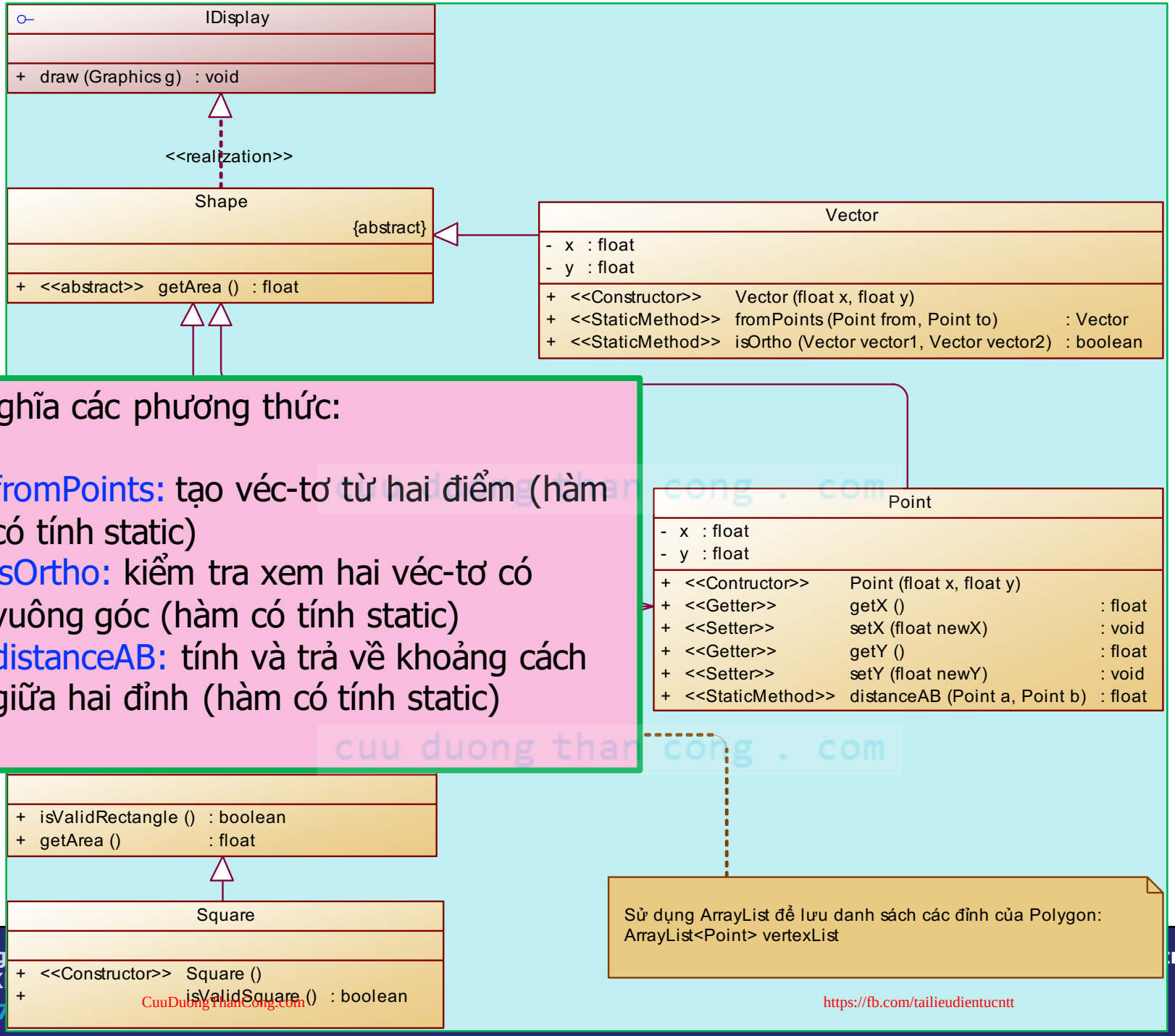
Ngược lại: Một đối tượng kiểu **Point**, có thể chứa hoặc không chứa kèm một đối tượng kiểu **Polygon** .

"Chứa" là gì?

- Hoặc chứa trực tiếp đối tượng bên trong (composition)
- Hoặc chỉ chứa con trỏ đến đối tượng cần chứa (aggregation)

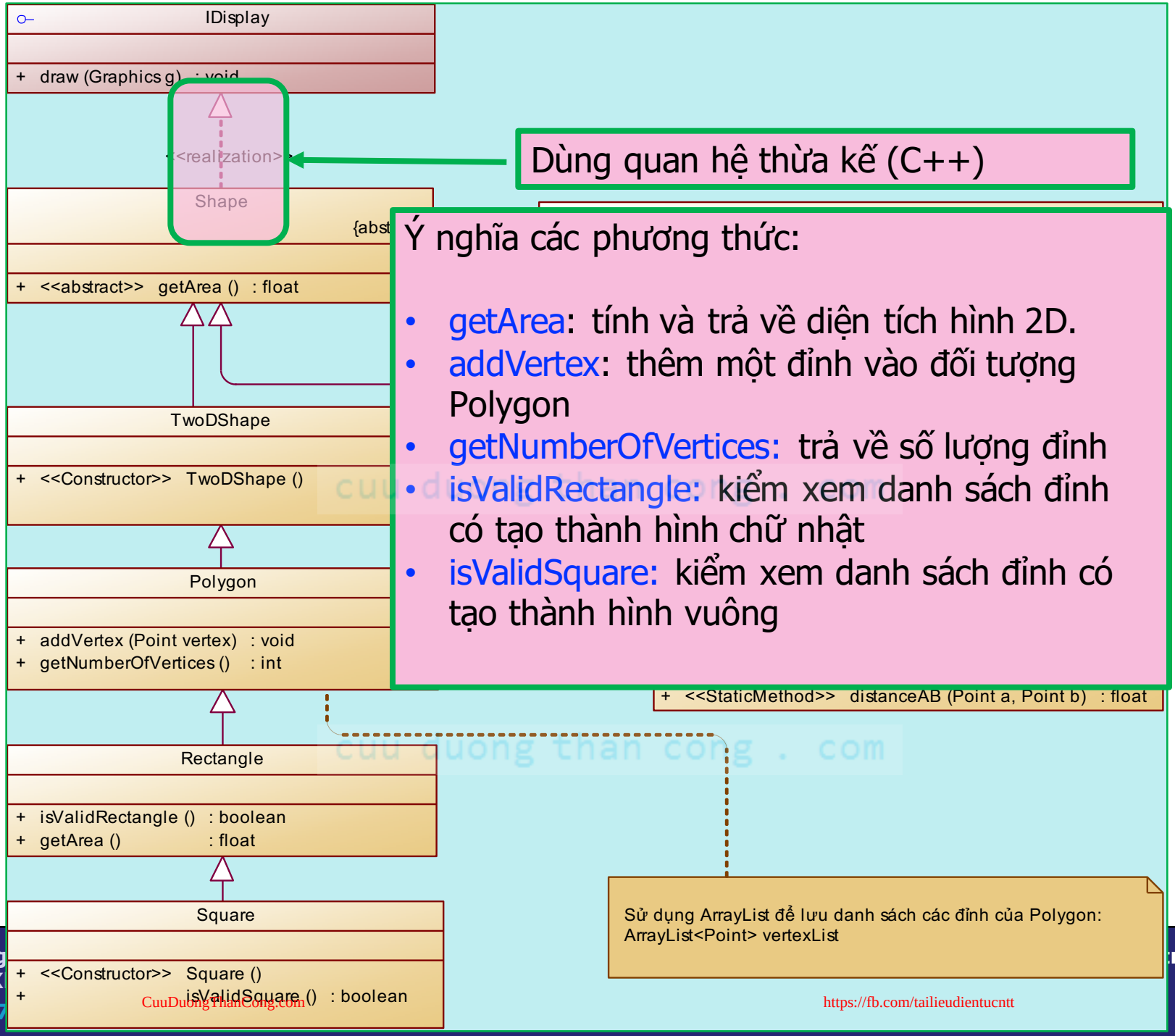
Cụ thể: Danh sách mà Polygon chứa ở đây chính là danh sách các đỉnh của Polygon





Ý nghĩa các phương thức:

- fromPoints: tạo véc-tơ từ hai điểm (hàm có tính static)
- isOrtho: kiểm tra xem hai véc-tơ có vuông góc (hàm có tính static)
- distanceAB: tính và trả về khoảng cách giữa hai đỉnh (hàm có tính static)



Thiết kế các lớp (II): bài tập

- Hãy thực hiện chuyển sang code C++ cho sơ đồ.
- Bổ sung các hàm khởi tạo để giúp tạo đối tượng dễ dàng cho các kiểu, kể cả bổ sung hàm khởi tạo mặc nhiên và copy.
- Bổ sung các toán tử cho các đối tượng
- Viết chương trình dùng các đối tượng vừa tạo

Tổng kết

- Các điểm quan trọng vừa học
  - Tại sao cần thừa kế
  - Hiểu rõ ý nghĩa thực sự của thừa kế
  - Tính khả kiến (public, protected, và private) tác động gì đến sự thừa kế
    - Có thừa kế được thành viên có tính private?
  - Thừa kế kiểu: public, protected, và private là gì
  - Khởi tạo lớp cha từ lớp con.

Tổng kết

- Các điểm quan trọng vừa học
  - Định nghĩa lại phương thức của lớp cha + cách truy cập phương thức của lớp cha từ lớp con.
  - Biểu diễn bằng sơ đồ cho:
    - Các lớp
    - Quan hệ thừa kế (mũi tên)
  - Vận dụng thừa kế trong thiết kế phần mềm