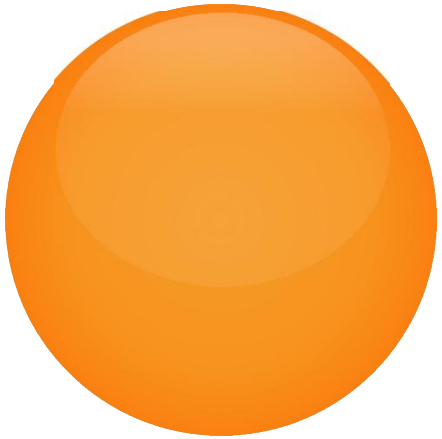
Phương pháp lập trình hướng đối tượng

Tuần 09:

Đa kế thừa

Phạm Tú San
ptsan@fit.hcmus.edu.vn

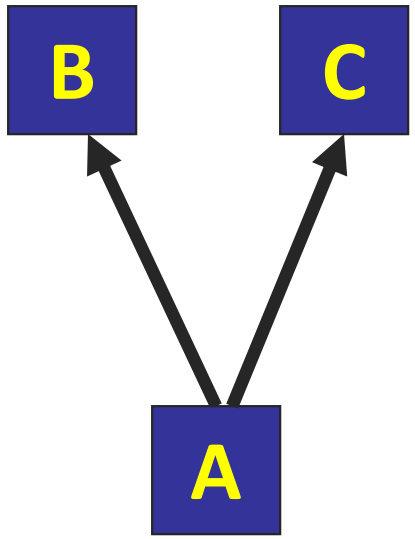


ĐA KẾ THỪA

Đa kế thừa

- *Đa kế thừa :* Một lớp có thể kế thừa từ nhiều lớp khác. Khi đó
- Ví dụ:

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



Đa kế thừa

- Các thành phần dữ liệu và phương thức của B và C sẽ được kế thừa trong lớp dẫn xuất A tương tự như kế thừa đơn
- Tính chất đa xạ (thông qua phương thức ảo) vẫn hoạt động tương tự như trong kế thừa đơn.

Ví dụ

```
class B {
    void draw();
};
class C {
    void cal();
};
class A: public B,
         public C
{
    void process();
};
```

```
void doSth(A& a)
{
    // B::draw()
    a.draw();
    // C::cal()
    a.cal();
    // A::process()
    a.process();
}
```

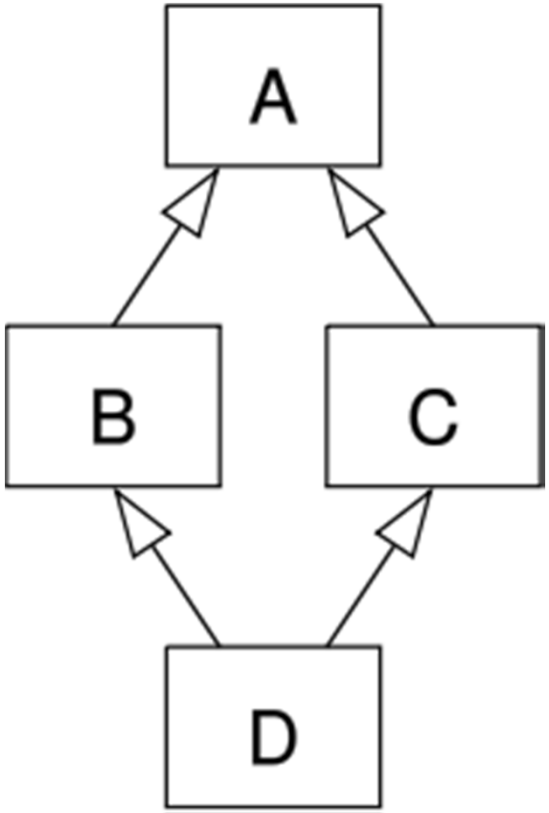
Đa kế thừa: nhập nhằng hàm trùng tên

- Nếu trong lớp B và C có các biến hay phương thức trùng tên nhau, thì tại lớp A để xác định là biến/phương thức đó từ B hay từ C, chúng ta phải sử dụng toán tử phạm vi ::

```
int main()
{
    A a;
    a.tinh(); //error: ambiguous
    a.B::tinh(); // OK
    a.C::tinh(); // OK
}
```

Vấn đề hình thoi

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



- Theo như khai báo ở trên, các biến/phương thức trong A sẽ được kế thừa lặp lại trong D

Vấn đề hình thoi – giải pháp

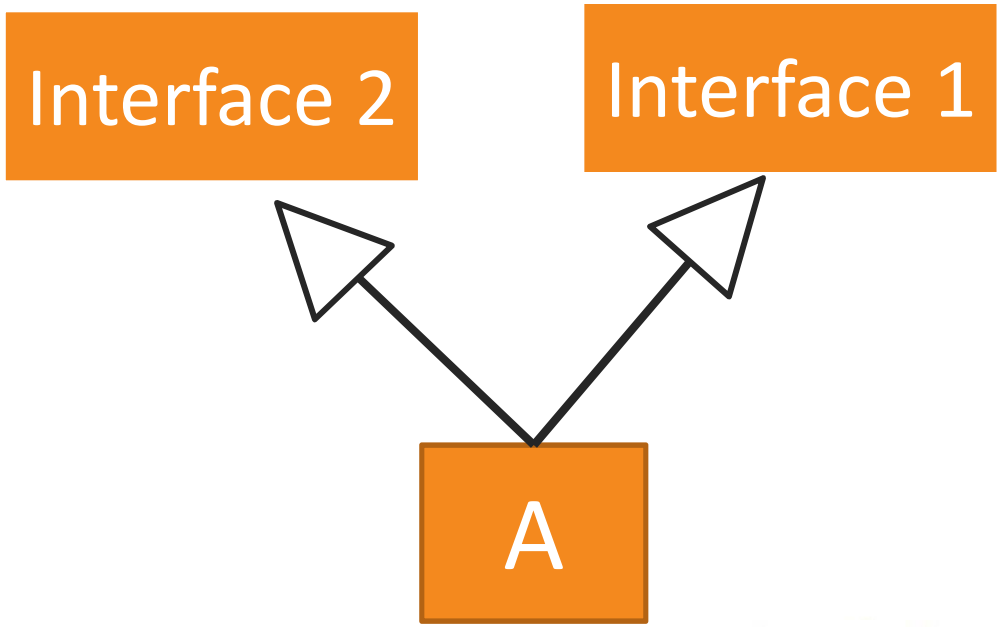
- Giải pháp: sử dụng lớp cơ sở ảo (`virtual base class`)

```
class A {...};
class B: public virtual A
{...};
class C: public virtual A
{...};
class D: public B, public C
{...};
```

- Khi đó trong lớp D chỉ có duy nhất 1 A

Đa kế thừa - interface

- Không được hỗ trợ trong C# và java
- Thay vào đó là khái niệm **interface**



Khái niệm Interface

- Interface là một phần của lớp
  - Lớp = Interface + cài đặt
- Interface quy định tính năng cung cấp ra bên ngoài
  - Tivi: interface là hệ thống nút bấm

Interface - Java

```
interface Bicycle
{
    // wheel revolutions per minute
    void changeGear(int newValue);
    void speedUp(int increment);
    void applyBrakes(int decrement);
}
```

```
class ACMEBicycle implements Bicycle
 {
     // remainder of this class
     // implemented as before
}
```

```
Bicycle b = new ACMEBicycle();
```

Interface – C++

- C++ giả lập interface bằng *abstract class*
- *abstract class*
  - Tất cả các phương thức đều thuần ảo

Dynamic binding - Interface

```
class A: public Interface1, public Interface2
{public:
        void    ham1(){}
        void    ham2(){}
};
```

```
class Interface1
{public:
        virtual void ham1() = 0;
};
```

```
class Interface2
{public:
        virtual void ham2() = 0;
};
```

```
void main()
{
        Interface1 *i1 = new A();
        i1->ham1();
        Interface2 *i2 = new A();
        i2 -> ham2();
}
```

Interface

```
Public class A: Interface1, Interface2
{public void     ham1(){}
public  void     ham2(){}
};
```

```
public Interface1
{public    void ham1() = 0;
};
```

```
public Interface2
{public    void ham2() = 0;
};
```

Tài liệu tham khảo

- Slide PPLTHĐT của
  - Thầy Đinh Bá Tiến
  - Thầy Nguyễn Minh Huy

Phương pháp lập trình hướng đối tượng

Tuần 09:

Xử lý lỗi và ngoại lệ

Phạm Tú San
ptsan@fit.hcmus.edu.vn

Các loại lỗi

- Các loại lỗi
  - Lỗi biên dịch
  - Lỗi thực thi
  - Lỗi ngữ nghĩa
- Xét lớp PhanSo – các lỗi thực thi có thể xảy ra

```
PhanSo( int tu, int mau);
void NghichDao();
void Nhap();
PhanSo ChiaPhanSo(const PhanSo &p);
```

Traditional Error Handling

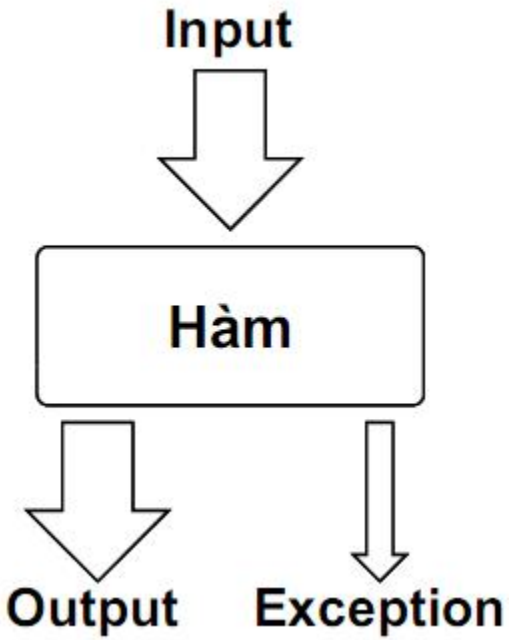
- Xử lý lỗi truyền thống
  - Xử lý lỗi cho từng hàm
  - Dùng kết quả để báo lỗi
    - Chỉ trả về 1 loại lỗi
    - Ảnh hưởng tới kết quả
  - Dùng tham số để báo lỗi
    - Tăng số lượng tham số
  - Dùng biến toàn cục
    - Có thể quên check

```
int timUCLN(int a, int b)
{
        if (b == 0)
            return 0;
        // Tìm UCLN...
}

int timUCLN(int a, int b, bool &err)
{
        err = (b == 0);
        // Tìm UCLN...
}
```

Exception Handling

- Xử lý lỗi qua kênh dành riêng
  - Kênh Exception
- Tách rời phần xử lý lỗi
  - Rõ ràng, trong sáng
  - Dễ quản lý, bảo trì
- Có khả năng thông báo lỗi ra bên ngoài cho các hàm ở cấp cao hơn
- Xây dựng sẵn trong ngôn ngữ



Exception Handling - Cách sử dụng

- Từ khóa “throw”
  - Đưa lỗi vào kênh Exception
- Từ khóa “try”
  - Dò lỗi trong kênh Exception
- Từ khóa “catch”
  - Bắt lỗi từ kênh Exception

```
int timUCLN(int a, int b)
{
        if (b == 0)
            throw 112;
        // Tìm UCLN...
}
void main()
{
        try
        {
            timUCLN(8, 0);
        }
}       catch (int ex)
        {
            cout << “Lỗi chia 0”;
        }
}
```

Exception Handling - Lớp lỗi

```
#define ERR_001   112
#define ERR_002   113
#define ERR_003   114
void f(int a, int b, int c)
{
        if (a < 0) throw ERR_001;
        if (b < 0) throw ERR_002;
        if (c < 0) throw ERR_003;
        //...
}
```

```
void main()
{
        try
        {
                f(0, 1, 2);
        }
        catch (int ex)
        {
                switch (ex)
                {
                        case ERR_001:
                        case ERR_002:
                        case ERR_003:
                }
        }
}
```

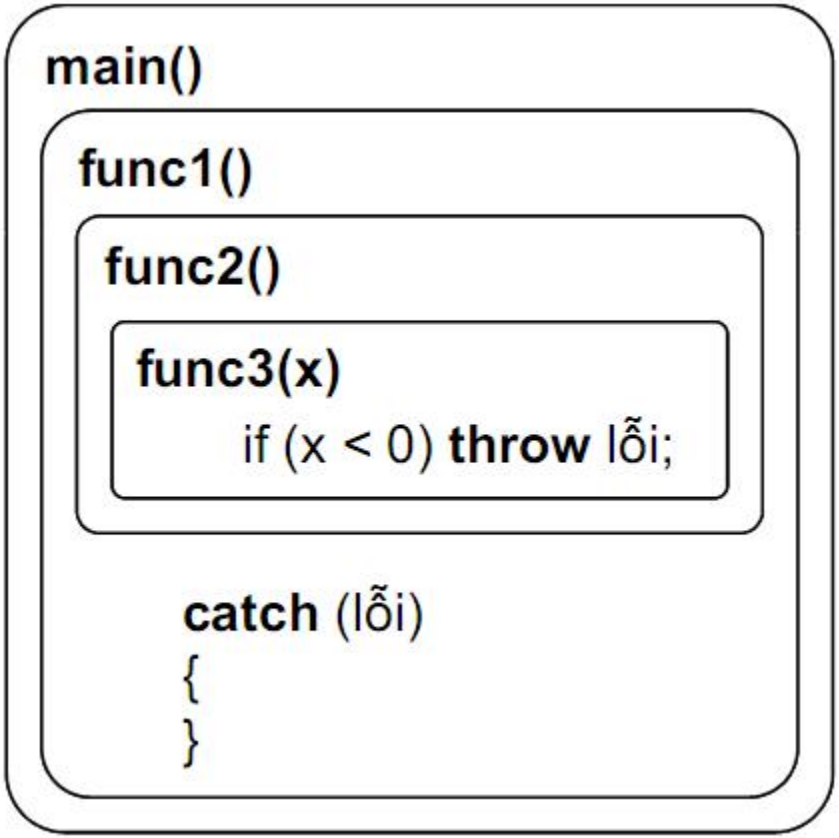
Exception Handling – lớp lỗi

```
class ERR_001
{
        char* getErr()
        {
                return “Description of Error”;
        }
};
class ERR_002 { };
class ERR_003 { };
void f(int a, int b, int c)
{
        if (a < 0) throw new ERR_001;
        if (b < 0) throw new ERR_002;
        if (c < 0) throw new ERR_003;
        //...
}
```

```
void main()
{
        try
        {
                f(0, 1, 2);
        }
        catch (ERR_001 ex)
        {
                cout << ex.getErr();
        }
        catch (ERR_002 ex)
        {
        }
        catch (ERR_003 ex)
        {
        }
}
```

Qui tắc xử lý lỗi

- **Throw early, catch late!**



Vấn đề rò rĩ bộ nhớ khi throw

- Khi throw, các biến con trỏ phía trước chưa được dọn dẹp
  - Rò rĩ bộ nhớ
- Giải pháp:
  - Dọn dẹp trước khi throw
  - Đưa các biến có sử dụng tài nguyên vào bên trong các lớp – phương pháp RAII (Resource Acquisition Is Initialization)
    - Phương thức hủy sẽ dọn dẹp

Bài tập

- Xây dựng lại lớp PhanSo và bắt tất cả các lỗi có thể có

Tài liệu tham khảo

- Slide PPLTHĐT của
  - Thầy Nguyễn Minh Huy

Phương pháp lập trình hướng đối tượng

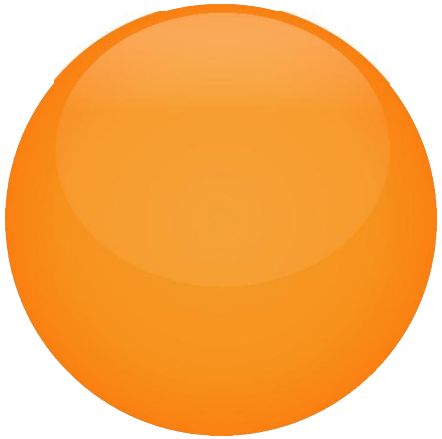
Tuần 09:

Các mẫu thiết kế hướng đối tượng

Phạm Tú San
ptsan@fit.hcmus.edu.vn

Bài toán gói hàng

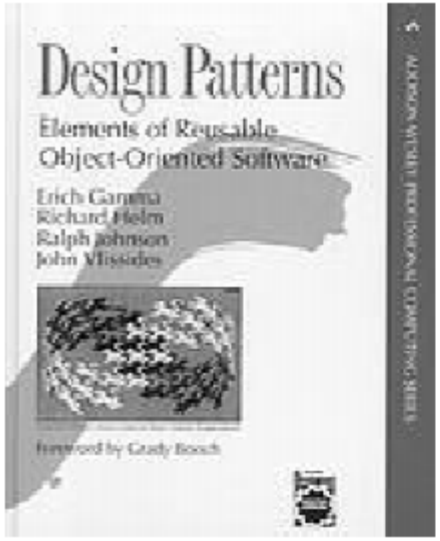
- Một doanh nghiệp chuyên vận chuyển hàng hóa thường xuyên phải đóng gói hàng vào các thùng. Một thùng hàng có thể chứa nhiều đơn vị hàng hoặc nhiều thùng hàng con. Mỗi đơn vị hàng có khối lượng xác định.
- Hãy xây dựng chương trình cho phép:
  - Nhập một thùng hàng (cho nhập vào các thùng hàng hoặc đơn vị hàng)
  - Cho phép thêm hoặc bớt 1 thùng hàng con hoặc 1 đơn vị hàng
  - Tính tổng khối lượng của thùng hàng



MẪU THIẾT KẾ

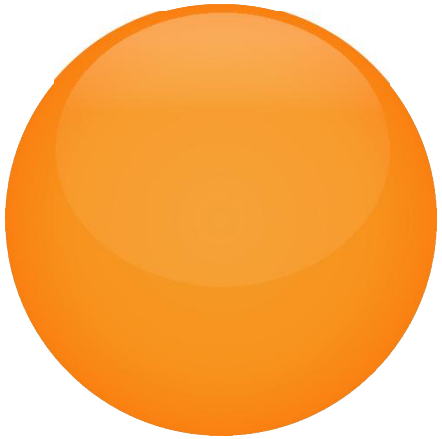
Mẫu thiết kế

- Mẫu thiết kế là gì
  - Nguyên lý HĐT còn mơ hồ, khó hiểu -> Cần có những bài giải mẫu
  - Mẫu thiết kế Gang of Four
    - Ra đời tại OOPSLA 1994
    - 23 bài giải mẫu cho những vấn đề thường gặp



Giới thiệu mẫu thiết kế HĐT

- Phân loại mẫu thiết kế Gang of four
  - Mẫu tạo lập đối tượng (creational patterns)
  - Mẫu cấu trúc đối tượng (structural patterns)
  - Mẫu hành xử đối tượng (behavioral patterns)
- Tư tưởng chính
  - Giao tiếp thông qua interface thay vì lớp cụ thể
  - Ưu tiên composition, hạn chế kế thừa lớp

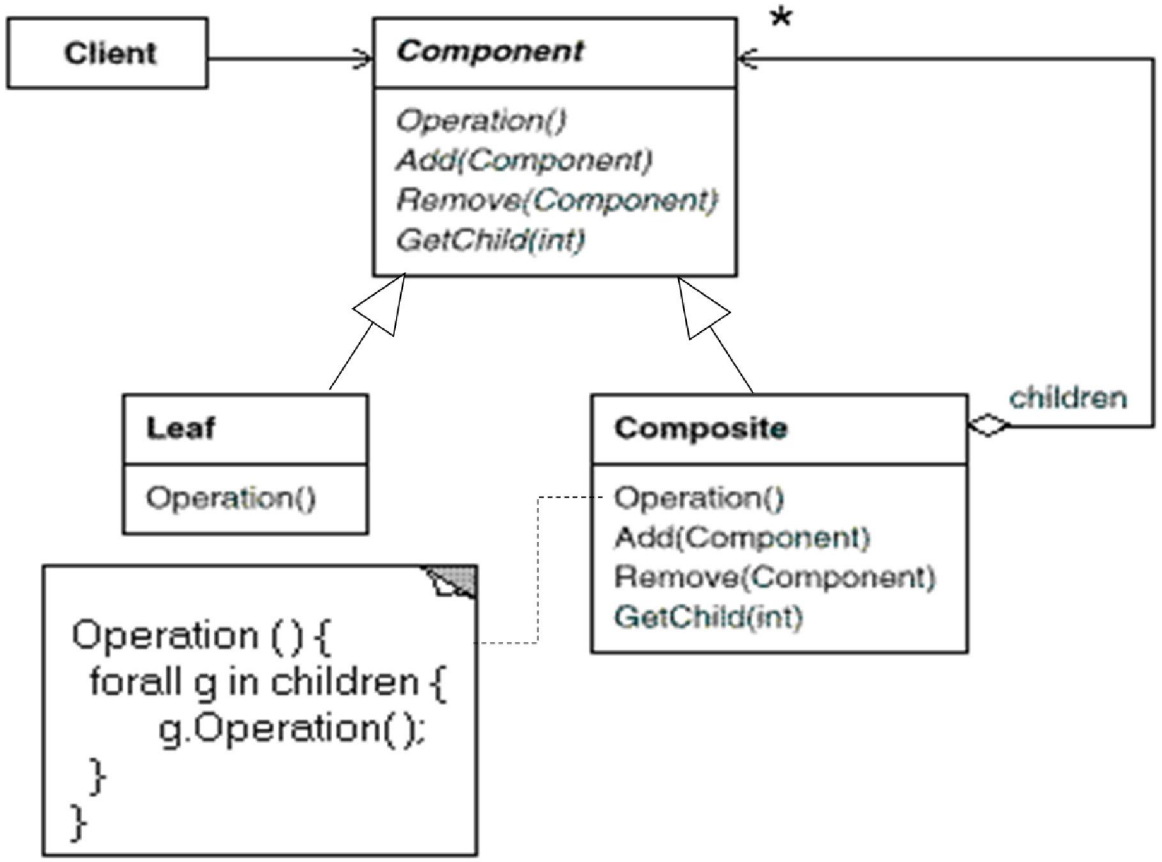


COMPOSITE

Composite

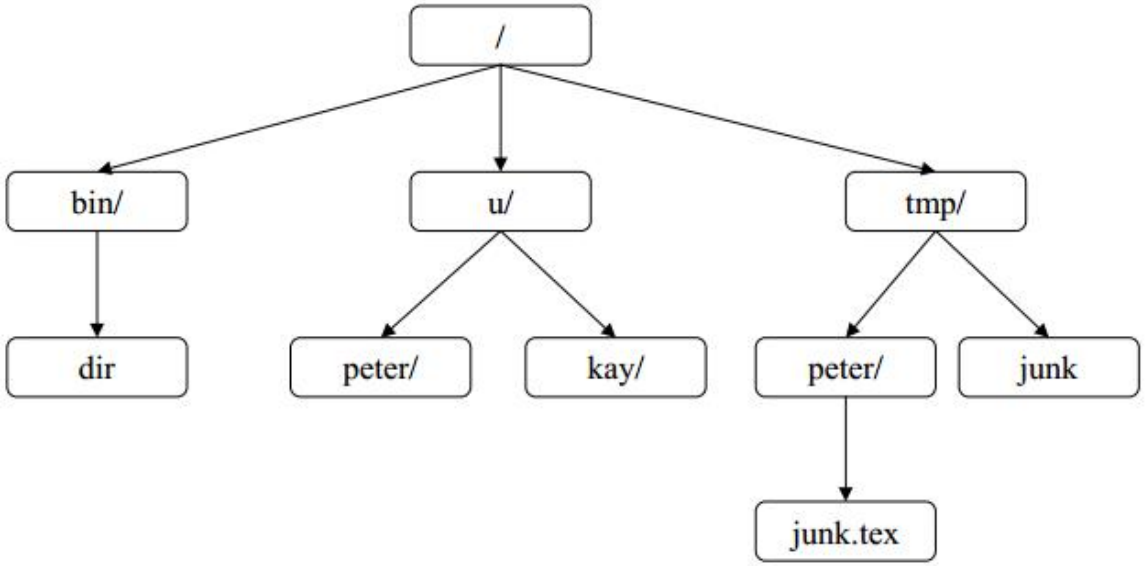
- Structural pattern
  - “Structural patterns are concerned with how classes and objects are composed to form larger structures. Structural *class* patterns use inheritance to compose interfaces or implementations”
- Các lớp đối tượng phân cấp đệ qui.
- Ví dụ:
  - Một hình vẽ phức tạp chứa các hình vẽ đơn giản.
  - Một mạch điện chứa các mạch đơn, mạch nối tiếp và mạch song song.
  - Một thư mục chứa các thư mục con và tập tin

Composite - structure

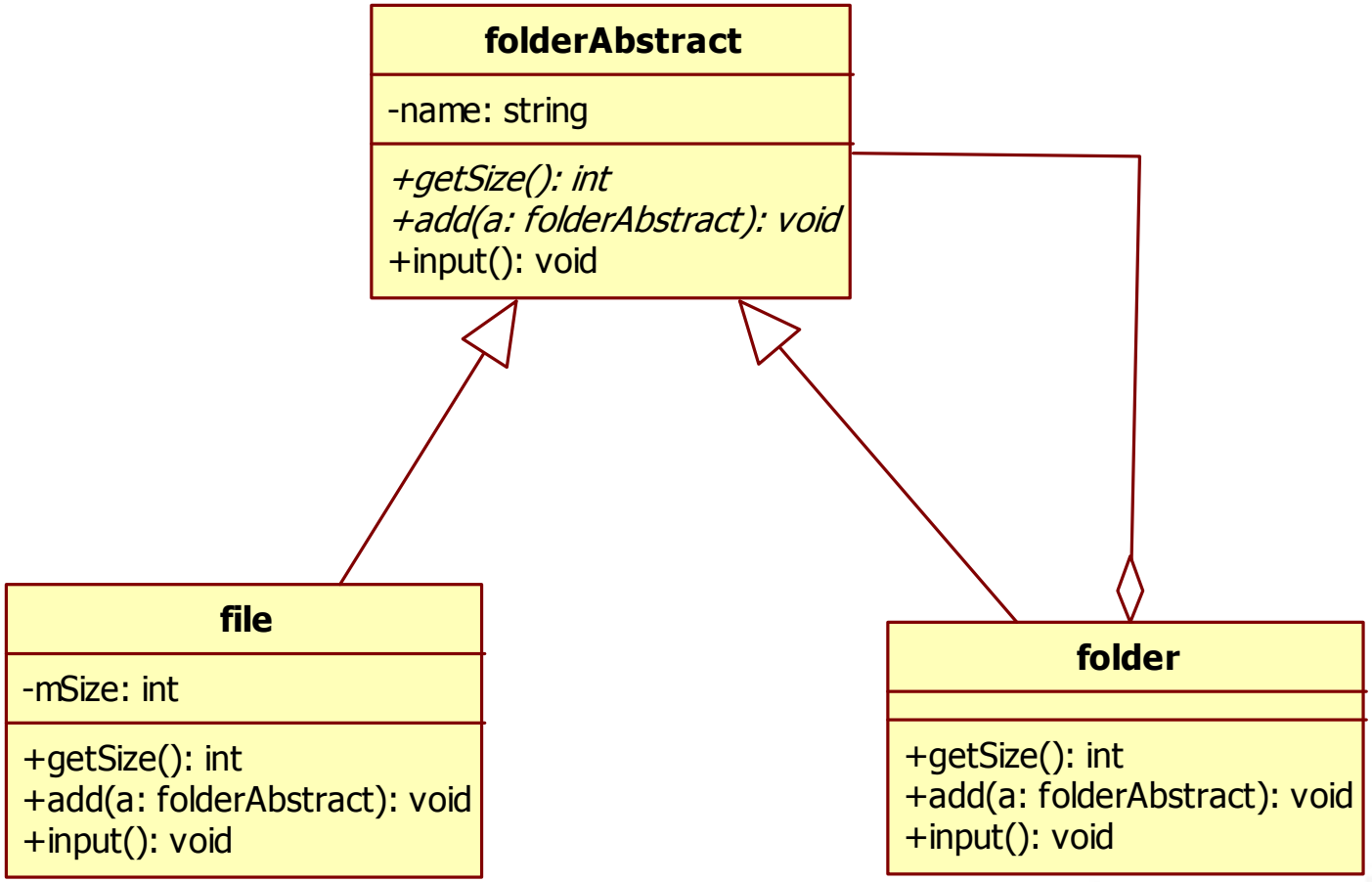


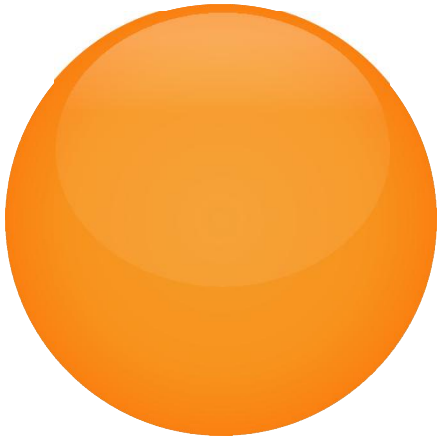
Ví dụ - bài toán thư mục

- Mỗi thư mục có chứa các thư mục con hoặc file
- Viết chương trình cho phép nhập, xuất hoặc tính kích thước 1 thư mục



Ví dụ - bài toán thư mục



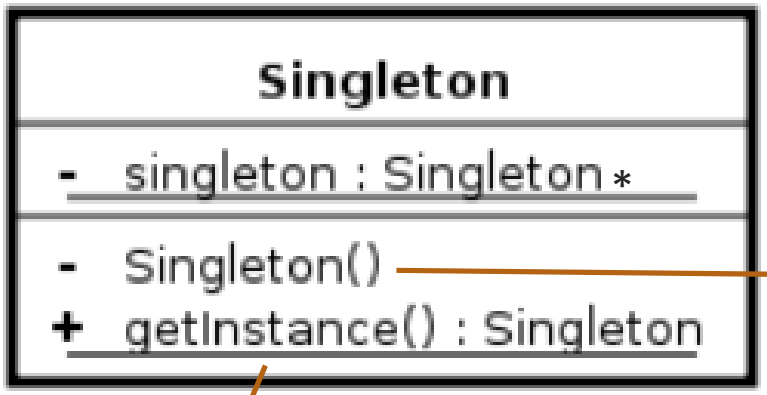


SINGLETON

Singleton

- Creational Pattern
  - “Creational design patterns abstract the instantiation process. They help make a system independent of how its objects are created, composed, and represented”
- Bảo đảm mỗi lớp chỉ có thể tạo ra duy nhất một đối tượng.
- Chỉ cho phép sử dụng đối tượng duy nhất này.
- Ví dụ:
  - Cửa sổ Open/Save tập tin.
  - Chương trình Task Manager.
  - Lớp FigureManager.
  - …

Singleton - structure



Phương thức thiết lập:
- Phải được cài đặt
- Ở mức private hoặc protected

```
Singleton getInstance()
{
    if (_singleton==NULL)
        _singleton = new Singleton();
    return _singleton;
}
```

Singleton– sử dụng

s1 và s2 cùng trỏ về một đối tượng

```
1. void main()
2. {
3.     Singleton* s1 = Singleton.getInstance();
4.     Singleton* s2 = Singleton.getInstance();
5.     Singleton s3;
6. }
```

Singleton - sample code

```
class MazeFactory {
public:
    static MazeFactory* Instance();

    // existing interface goes here
protected:
    MazeFactory();
private:
    static MazeFactory* _instance;
};
```

```
MazeFactory* MazeFactory::_instance = 0;

MazeFactory* MazeFactory::Instance () {
    if (_instance == 0) {
        _instance = new MazeFactory;
    }
    return _instance;
}
```

Thao khảo

- Design Patterns: Elements of Reusable Object-Oriented Software
- Slide PPLTHDT của
  - Cô Đặng Thị Thanh Nguyên
  - Thầy Nguyễn Minh Huy