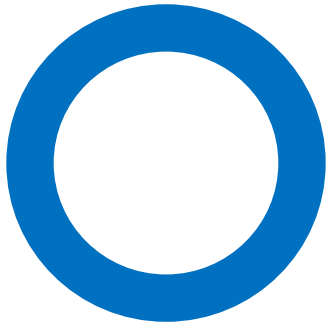
Lớp vs Đối tượng

Lập trình hướng đối tượng

Hồ Tuấn Thanh

htthanh@fit.hcmus.edu.vn

Sự phát triển trong lập trình



1. Một hàm main duy nhất

```
#include <iostream>
using namespace std;

int main()
{
    int tu1, mau1;
    int tu2, mau2;
    int tu3, mau3;
```

1. Một hàm main duy nhất

```
cout << "Nhap phan so 1" << endl;
cout << "Nhap tu: ";
cin >> tu1;
do
{
    cout << "Nhap mau: ";
    cin >> mau1;
}while(mau1 == 0);
if(mau1 < 0)
{
    tu1 = -tu1;
    mau1 = -mau1;
}
```

1. Một hàm main duy nhất

```
cout << "Nhap phan so 2" << endl;
cout << "Nhap tu: ";
cin >> tu2;
do
{
    cout << "Nhap mau: ";
    cin >> mau2;
}while(mau2 == 0);
if(mau2 < 0)
{
    tu2 = -tu2;
    mau2 = -mau2;
}
```

1. Một hàm main duy nhất

```
    tu3 = tu1 * mau2 + tu2 * mau1;
    mau3 = mau1 * mau2;
```

1. Một hàm main duy nhất

```
    cout << tu1 << "/" << mau1;
    cout << " + ";
    cout << tu2 << "/" << mau2;
    cout << " = ";
    cout << tu3 << "/" << mau3;
    cout << endl;
    return 0;
}
```

Vấn đề

- Code quá dài, khó đọc.
- Quá nhiều đoạn code giống nhau.
  - Nhập ps1 vs nhập ps2.
  - In ps1 vs in ps2 vs in ps3.
- Copy – paste – **EDIT.**
- Khi modify 1 chỗ, phải nhớ tìm và update những chỗ còn lại.

Ví dụ

```
cout << "Nhap phan so 2" << endl;
cout << "Nhap tu: ";
cin >> tu2;
do
{
    cout << "Nhap mau: ";
    cin >> mau1;
}while(mau2 == 0);
if(mau2 < 0)
{
    tu2 = - tu2;
    mau2 = -mau2;
}
```

2. Viết và sử dụng hàm

```
#include <iostream>
using namespace std;

void nhap(int &tu, int &mau);
void cong(int tu1, int mau1, int tu2, int mau2, int &tu3, int &mau3);
void xuat(int tu, int mau);
```

2. Viết và sử dụng hàm

```
int main()
{
    int tu1, mau1;
    int tu2, mau2;
    int tu3, mau3;

    cout << "Nhap phan so 1" << endl;
    nhap(tu1, mau1);

    cout << "Nhap phan so 2" << endl;
    nhap(tu2, mau2);

    cong(tu1, mau1, tu2, mau2, tu3, mau3);

    xuat(tu1, mau1);
    cout << " + ";
    xuat(tu2, mau2);
    cout << " = ";
    xuat(tu3, mau3);
    cout << endl;
    return 0;
}
```

2. Viết và sử dụng hàm

```
void nhap(int &tu, int &mau)
{
    cout << "Nhap tu: ";
    cin >> tu;
    do
    {
        cout << "Nhap mau: ";
        cin >> mau;
    }while(mau == 0);
    if(mau < 0)
    {
        tu = -tu;
        mau = -mau;
    }
}
```

2. Viết và sử dụng hàm

```
void cong(int tu1, int mau1, int tu2, int mau2, int &tu3, int &mau3)
{
    tu3 = tu1 * mau2 + tu2 * mau1;
    mau3 = mau1 * mau2;
}
```

2. Viết và sử dụng hàm

```
void xuat(int tu, int mau)
{
    cout << tu << "/" << mau;
}
```

Vẫn còn vấn đề...

- Sử dụng nhiều biến quá.
- Râu ông nọ, cắm càm bà kia.

Ví dụ

```
int main()
{
    int tu1, mau1;
    int tu2, mau2;
    int tu3, mau3;

    cout << "Nhap phan so 1" << endl;
    nhap(tu1, mau1);

    cout << "Nhap phan so 2" << endl;
    nhap(tu2, mau1);

    cong(tu1, mau1, tu1, mau2, tu3, mau3);

    xuat(tu1, mau1);
    cout << " + ";
    xuat(tu2, mau2);
    cout << " = ";
    xuat(tu3, mau1);
    cout << endl;
    return 0;
}
```

3. Sử dụng struct

```
#include <iostream>
using namespace std;

struct PhanSo
{
    int tu;
    int mau;
};

void nhap(PhanSo &p);
PhanSo cong(PhanSo p1, PhanSo p2);
void xuat(PhanSo p);
```

3. Sử dụng struct

```
int main()
{
    PhanSo p1, p2, p3;

    cout << "Nhap phan so 1" << endl;
    nhap(p1);

    cout << "Nhap phan so 2" << endl;
    nhap(p2);

    p3 = cong(p1, p2);

    xuat(p1);
    cout << " + ";
    xuat(p2);
    cout << " = ";
    xuat(p3);
    cout << endl;
    return 0;
}
```

3. Sử dụng struct

```
void nhap(PhanSo &p)
{
    cout << "Nhap tu: ";
    cin >> p.tu;
    do
    {
        cout << "Nhap mau: ";
        cin >> p.mau;
    }while(p.mau == 0);
    if(p.mau < 0)
    {
        p.tu = -p.tu;
        p.mau = -p.mau;
    }
}
```

3. Sử dụng struct

```
PhanSo cong(PhanSo p1, PhanSo p2)
{
    PhanSo p3;
    p3.tu = p1.tu * p2.mau + p2.tu * p1.mau;
    p3.mau = p1.mau * p2.mau;
    return p3;
}
```

3. Sử dụng struct

```
void xuat(PhanSo p)
{
    cout << p.tu << "/" << p.mau;
}
```

4. Sử dụng class

```
#include <iostream>
using namespace std;

class PhanSo
{
private:
    int tu;
    int mau;
public:
    void nhap();
    PhanSo cong(PhanSo p2);
    void xuat();
};
```

4. Sử dụng class

```
int main()
{
    PhanSo p1, p2, p3;

    cout << "Nhap phan so 1" << endl;
    p1.nhap();

    cout << "Nhap phan so 2" << endl;
    p2.nhap();

    p3 = p1.cong(p2);

    p1.xuat();
    cout << " + ";
    p2.xuat();
    cout << " = ";
    p3.xuat();
    cout << endl;
    return 0;
}
```

4. Sử dụng class

```
void PhanSo::nhap()
{
    cout << "Nhap tu: ";
    cin >> this->tu;
    do
    {
        cout << "Nhap mau: ";
        cin >> this->mau;
    }while(this->mau == 0);
    if(this->mau < 0)
    {
        this->tu = -this->tu;
        this->mau = -this->mau;
    }
}
```

4. Sử dụng class

```
PhanSo PhanSo::cong(PhanSo p2)
{
    PhanSo p3;
    p3.tu = this->tu * p2.mau + p2.tu * this->mau;
    p3.mau = this->mau * p2.mau;
    return p3;
}
```

4. Sử dụng class

```
void PhanSo::xuat()
{
    cout << this->tu << "/" << this->mau;
}
```

Lưu ý

- Gom thuộc tính (attribute, property, data member) và hàm (method, operation) vào trong class.

```
class PhanSo
{
private:
    int tu;
    int mau;
public:
    void nhap();
    PhanSo cong(PhanSo p2);
    void xuat();
};
```

Thay đổi cách gọi hàm

- Lấy biến làm trung tâm vs lấy hàm làm trung tâm.

```
p1.nhap();
p2.nhap();
p3 = p1.cong(p2);
p3.xuat();
```

```
nhap(p1);
nhap(p2);
p3 = cong(p1, p2);
xuat(p3);
```

Thay đổi cách viết hàm

- Mỗi hàm giảm 1 tham số.

```
void nhap();
PhanSo cong(PhanSo p2);
void xuat();
```

```
void nhap(PhanSo &p);
PhanSo cong(PhanSo p1, PhanSo p2);
void xuat(PhanSo p);
```

Toán tử 4 chấm ::

- Scope resolution operator.
- Khai báo hàm: khai báo trong lớp.
- Cài đặt hàm: thường cài đặt ngoài lớp, để dễ đọc code ➔ toán tử :: cho biết hàm-đang-cài-đặt thuộc class nào.

```
class PhanSo
{
private:
    int tu, mau;
public:
    void xuat();
};
```

```
void PhanSo::xuat()
{
    cout << this->tu << "/" << this->mau;
}
```

Con trỏ this

- Con trỏ this đại diện cho đối tượng gọi hàm.
- Thông thường, ko cần viết tường minh con trỏ this.
- Chỉ viết this, khi muốn phân biệt giữa tên thuộc tính vs tên biến cục bộ cùng tên.

Ví dụ

```
void PhanSo::xuat()
{
    cout << this->tu << "/" << this->mau;
}
```

```
p1.xuat();
```

```
p2.xuat();
```

```
p3.xuat();
```

Không cần viết this

```
void PhanSo::nhap()
{
    cout << "Nhap tu: ";
    cin >> tu;
    do
    {
        cout << "Nhap mau: ";
        cin >> mau;
    }while(mau == 0);
    if(mau < 0)
    {
        tu = -tu;
        mau = -mau;
    }
}
```

Không cần viết this

```
PhanSo PhanSo::cong(PhanSo p2)
{
    PhanSo p3;
    p3.tu = tu * p2.mau + p2.tu * mau;
    p3.mau = mau * p2.mau;
    return p3;
}
```

Không cần viết this

```
void PhanSo::xuat()
{
    cout << tu << "/" << mau;
}
```

Tầm vực (scope, visibility)

- Dùng để giới hạn khả năng truy cập vào 1 thuộc tính, 1 hàm.
- private:
  - trong lớp truy xuất được
  - ngoài lớp **KHÔNG** truy xuất được
- public:
  - trong lớp truy xuất được
  - ngoài lớp truy xuất được

Ví d

```
class PhanSo
{
private:
    int tu;
    int mau;
public:
    void nhap();
    PhanSo cong(PhanSo p2);
    void xuat();
};
```

```
void PhanSo::xuat()
{
    cout << tu << "/" << mau;
}
```

```
int main()
{
    PhanSo p1;

    cout << "Nhap phan so 1" << endl;
    p1.nhap();

    cout << p1.tu << endl; // ERROR
    return 0;
}
```

Ví d

```
class PhanSo
{
public:
    int tu;
    int mau;
public:
    void nhap();
    PhanSo cong(PhanSo p2);
    void xuat();
};
```

```
void PhanSo::xuat()
{
    cout << tu << "/" << mau;
}
```

```
int main()
{
    PhanSo p1;

    cout << "Nhap phan so 1" << endl;
    p1.nhap();

    cout << p1.tu << endl; // OK
    return 0;
}
```

Ví dụ

```
class PhanSo
{
private:
    int tu;
    int mau;
    void nhap();
public:
    PhanSo cong(PhanSo p2);
    void xuat();
};
```

```
int main()
{
    PhanSo p1;

    cout << "Nhap phan so 1" << endl;
    p1.nhap(); // ERROR

    return 0;
}
```

Ví dụ

```
class PhanSo
{
private:
    int tu;
    int mau;
public:
    void nhap();
    PhanSo cong(PhanSo p2);
    void xuat();
};
```

```
int main()
{
    PhanSo p1;

    cout << "Nhap phan so 1" << endl;
    p1.nhap();

    return 0;
}
```

Lập trình hướng đối tượng là gì?

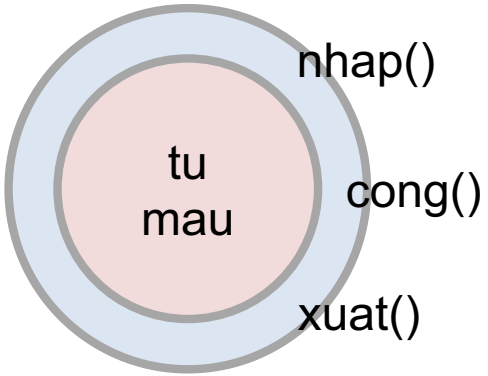
- LT HDT (object oriented programming, OOP) là một phương pháp lập trình (programming paradigm) dựa trên 2 khái niệm lớp (class) và đối tượng (object, instance).
- Mỗi đối tượng bao gồm:
  - Thành phần dữ liệu (data member, attribute, property) mô tả thông tin của đối tượng.
  - Các xử lí (method, operation, function) liên quan đến đối tượng đó.

Lớp vs đối tượng

- Đối tượng: object, instance, variable
  - Sự vật, hiện tượng cụ thể
  - VD: p, p1, p2, p3
- Lớp: class, data type
  - Loại đối tượng
  - VD: class PhanSo

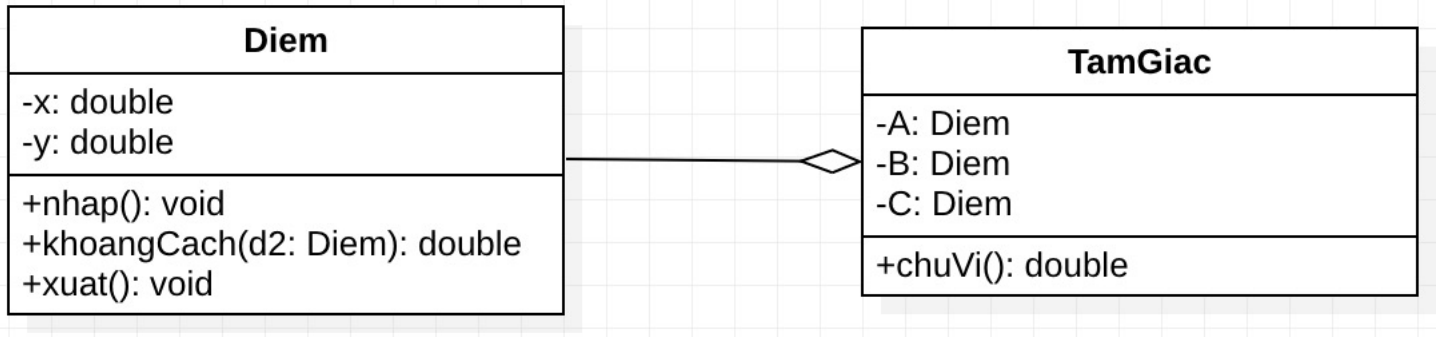
3 tính chất của LT HDT

- Tính đóng gói bao bọc (encapsulation) và ẩn dữ liệu (data hiding).
  - Gom các thuộc tính và hàm vào trong 1 lớp.
  - Sử dụng 2 từ khóa private, public để qui định khả năng truy xuất được vào các thuộc tính và hàm đó.
- Tính kế thừa (inheritance).
- Tính đa hình (polymorphism).



Sơ đồ lớp (class diagram)

- Khi giải 1 bài bằng PP LT HDT, ta quan tâm bài đó cần những class nào, thuộc tính nào, hàm nào, mối liên hệ giữa các class ra sao ➔ sử dụng class diagram



StarUML™
The Open Source UML/MDA Platform


Q
&
A