Chương 9
THIẾT KẾ LỚP

1. ThS. Nguyễn Hữu Lợi
2. ThS. Nguyễn Văn Toàn
3. TS. Nguyễn Duy Khánh
4. TS. Nguyễn Tấn Trần Minh Khang

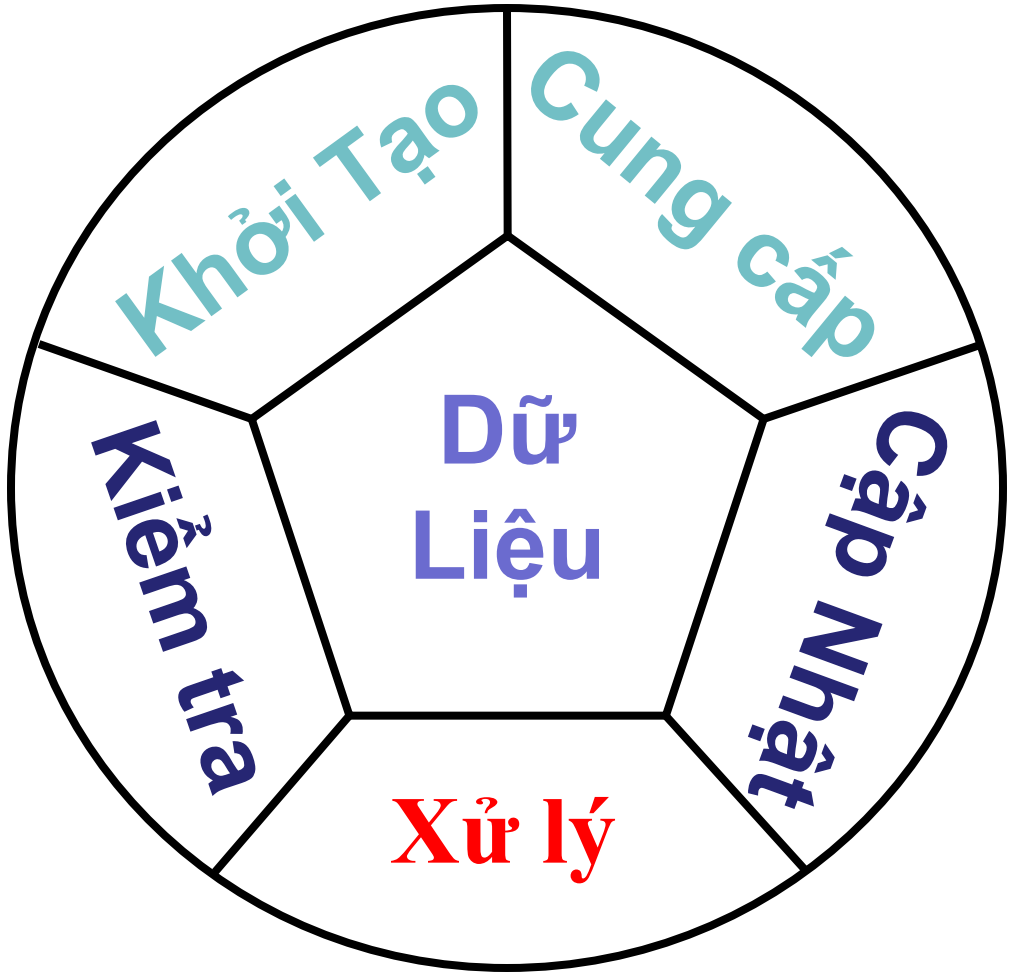
1. ĐẶT VẤN ĐỀ

1. Đặt vấn đề

—Hãy thiết kế và xây dựng lớp CSoPhuc trong toán học.

2. PHÂN LOẠI PHƯƠNG THỨC

2. Phân loại phương thức



2. Phân loại phương thức

– Các phương thức (method) của một lớp đối tượng (class) được chia thành 5 loại như sau:

+ Nhóm các phương thức khởi tạo.
+ Nhóm các phương thức cung cấp thông tin.
+ Nhóm các phương thức cập nhật thông tin.
+ Nhóm các phương thức kiểm tra.
+ Nhóm các phương thức xử lý.

2. Phân loại phương thức

- Nhóm các phương thức khởi tạo
  + Khởi tạo (init) thông tin cho đối tượng.
  + Các phương thức thiết lập (constructors) thuộc nhóm các phương thức khởi tạo.
  + Toán phần thực nhập (operator >>) thuộc nhóm các phương thức khởi tạo.
  + Các loại phương thức khởi tạo.
    - Phương thức khởi tạo mặc định.
    - Phương thức khởi tạo dựa vào đối tượng khác cùng thuộc về lớp.
    - Phương thức khởi tạo khi biết đầy đủ thông tin.

2. Phân loại phương thức

- Nhóm các phương thức cung cấp thông tin
  - + Các phương thức cung cấp thông tin của một lớp đối tượng (class) giữ nhiệm vụ cung cấp các thông tin (attributes, properties, information, data) của những đối tượng (objects) thuộc về lớp cho thế giới bên ngoài.
  - + Toán phần thực xuất (operator <<) thuộc nhóm các phương thức cung cấp thông tin.
  - + Trong một lớp có bao nhiêu thuộc tính (attributes, properties) thì sẽ có ít nhất bấy nhiêu phương thức cung cấp thông tin.
  - + Thông thường các phương thức cung cấp thông tin bắt đầu bằng cụm từ `get`.

2. Phân loại phương thức

- Nhóm các phương thức cập nhật thông tin
  - + Các phương thức cập nhật thông tin của một lớp đối tượng (class) giữ nhiệm vụ cập nhật các thông tin (attributes, properties, information) của những đối tượng (objects) thuộc về lớp khi có sự thay đổi (sự hiệu chỉnh, sự biến động).
  - + Trong một lớp có bao nhiêu thuộc tính (attributes, properties) thì sẽ có ít nhất bấy nhiêu phương thức cập nhật thông tin.
  - + Thông thường các phương thức cập nhật thông tin bắt đầu bằng cụm từ `set`.

2. Phân loại phương thức

- Nhóm các phương thức kiểm tra
  + Các phương thức kiểm tra của một lớp đối tượng (class) giữ nhiệm vụ kiểm tra (test, check) tính hợp lệ của những thông tin (attributes, properties, information) thuộc về các đối tượng, kiểm tra một tính chất nào đó của đối tượng.
  + Các phương thức toán phần thực so sánh (relational operators, comparison operators) thuộc nhóm các phương thức kiểm tra.
  + Thông thường các phương thức kiểm tra bắt đầu bằng cụm từ `is`.

2. Phân loại phương thức

- Nhóm các phương thức xử lý.
  + Các phương thức xử lý của một lớp đối tượng (class) giữ nhiệm vụ thực hiện các xử lý, tính toán cho lớp đối tượng.
  + Các toán phần thực số học (arithmetic operators), toán phần thực gán (operator =) thuộc nhóm các phương thức xử lý.
  + Phương thức phá hủy (destructor) thuộc nhóm các phương thức xử lý.

3. THIẾT KẾ LỚP SỐ PHỨC

3. Thiết kế lớp số phức

- Thiết kế các thuộc tính của lớp số phức.
- Lớp `CSoPhuc` có hai thuộc tính là phần thực (`Thuc`) và phần ảo (`Ao`) với kiểu dữ liệu là số thực (`float`).

```
11.class CSoPhuc
12.{
13.    private:
14.        float Thuc;
15.        float Ao;
16.    …
17.};
```

3. Thiết kế lớp số phức

– Thiết kế các phương thức khởi tạo.

```
11.class CSoPhuc
12.{
13.        // Nhóm các phương thức khởi tạo
14.        void KhoiTao();
15.        void KhoiTao(float);
16.        void KhoiTao(float,float);
17.        void KhoiTao(CSoPhuc&);
18.        CSoPhuc();
19.        CSoPhuc(float);
20.        CSoPhuc(float,float);
21.        CSoPhuc(CSoPhuc&);
22.};
```

3. Thiết kế lớp số phức

– Thiết kế các phương thức khởi tạo.

```
11.class CSoPhuc
12.{
13.        // Nhóm các phương thức khởi tạo
14.        void KhoiTao();
15.        void KhoiTao(float,float);
16.        void KhoiTao(CSoPhuc&);
17.        CSoPhuc();
18.        CSoPhuc(float,float);
19.        CSoPhuc(CSoPhuc&);
20.};
```

3. Thiết kế lớp số phức

- Định nghĩa các phương thức khởi tạo.

```
11.void CSoPhuc::KhoiTao()
12.{
13.    Thuc = 0;
14.    Ao = 0;
15.}
```

- Phương thức khởi tạo mặc định, không nhận tham số đầu vào, các thông tin ban đầu của đối tượng được thiết lập mặc định như sau: phần thực lấy giá trị 0, phần ảo lấy giá trị 0.

3. Thiết kế lớp số phức

- Định nghĩa các phương thức khởi tạo.

```
11.void CSoPhuc::KhoiTao(float thth, float aoao)
12.{
13.    Thuc = thth;
14.    Ao = aoao;
15.}
```

- Phương thức khởi tạo khi biết đầy đủ thông tin, nhận hai tham số đầu vào là `thth`, `aoao` các thông tin ban đầu của đối tượng được thiết lập như sau: phần thực (`Thuc`) lấy giá trị `thth`, phần ảo (`Ao`) lấy giá trị `aoao`.

3. Thiết kế lớp số phức

- Định nghĩa các phương thức khởi tạo.

```
11.void CSoPhuc::KhoiTao(CSoPhuc &x)
12.{
13.    Thuc = x.Thuc;
14.    Ao = x.Ao;
15.}
```

- Phương thức khởi tạo dựa vào đối tượng khác cùng thuộc về lớp, nhận một tham số đầu vào là `x` là đối tượng thuộc lớp `CSoPhuc`, các thông tin ban đầu của đối tượng được thiết lập như sau: phần thực (`Thuc`) lấy giá trị `x.Thuc`, phần ảo (`Thuc`) lấy giá trị `x.Ao`.

3. Thiết kế lớp số phức

- Định nghĩa các phương thức khởi tạo.

```
11.CSoPhuc::CSoPhuc()
12.{
13.    Thuc = 0;
14.    Ao = 0;
15.}
```

- Phương thức thiết lập mặc định, không nhận tham số đầu vào, các thông tin ban đầu của đối tượng được thiết lập mặc định như sau: phần thực (`Thuc`) lấy giá trị 0, phần ảo (`Ao`) lấy giá trị 1.

3. Thiết kế lớp số phức

- Định nghĩa các phương thức khởi tạo.

```
11.CSoPhuc::CSoPhuc(float thth,float aoao)
12.{
13.    Thuc = thth;
14.    Ao = aoao;
15.}
```

- Phương thức thiết lập khi biết đầy đủ thông tin, nhận hai tham số đầu vào là `thth`, `aoao` các thông tin ban đầu của đối tượng được thiết lập như sau: phần thực (`Thuc`) lấy giá trị `thth`, phần ảo (`Ao`) lấy giá trị `aoao`.

3. Thiết kế lớp số phức

- Định nghĩa các phương thức khởi tạo.

```
11.CSoPhuc::CSoPhuc(CSoPhuc &x)
12.{
13.    Thuc = x.Thuc;
14.    Ao = x.Ao;
15.}
```

- Phương thức thiết lập sao chép, nhận một tham số đầu vào là `x` là đối tượng thuộc lớp `CSoPhuc`, các thông tin ban đầu của đối tượng được thiết lập như sau: phần thực (`Thuc`) lấy giá trị `x.Thuc`, phần ảo (`Ao`) lấy giá trị `x.Ao`.

3. Thiết kế lớp số phức

– Thiết kế các phương thức khởi tạo.

```
11.class CSoPhuc
12.{
13.         …
14.         // Nhóm các phương thức khởi tạo
15.         void Nhap();
16.         friend istream& operator>>
17.                         (istream&, CSoPhuc&);
18.         …
19.};
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức khởi tạo.

```
11.void CSoPhuc::Nhap()
12.{
13.    cout<<"Nhap thuc: ";
14.    cin>>Thuc;
15.    cout<<"Nhap ao: ";
16.    cin>>Ao;
17.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức khởi tạo.

```
11.istream& operator>>(istream&is,CSoPhuc&x)
12.{
13.    cout<<"Nhap Thuc: ";
14.    is>>Thuc;
15.    cout<<"Nhap Ao: ";
16.    is>>Ao;
17.    return is;
18.}
```

3. Thiết kế lớp số phức

– Thiết kế các phương thức cung cấp thông tin.

```
11.class CSoPhuc
12.{
13.    …
14.    public:
15.        // Nhóm các phương thức
16.        // cung cấp thông tin
17.        float getThuc();
18.        float getAo();
19.
20.    …
21.};
```

3. Thiết kế lớp số phức

- Định nghĩa các phương thức cung cấp thông tin.
- Cách 01.

```
11.float CSoPhuc::getThuc()
12.{
13.    return Thuc;
14.}
```

- Cách 02.

```
11.float CSoPhuc::getThuc()
12.{
13.    return this->Thuc;
14.}
```

Bên trong thân phương thức của một lớp đối tượng, this là một con trỏ đối tượng thuộc về lớp mà phương thức đó thuộc về, con trỏ đối tượng this giữ địa chỉ của đối tượng đang gọi thực hiện phương thức. Hơn nữa, *this chính là đối tượng đang gọi thực hiện phương thức.

3. Thiết kế lớp số phức

- Định nghĩa các phương thức cung cấp thông tin.
- Cách 01.

```
11.float CSoPhuc::getAo()
12.{
13.    return Ao;
14.}
```

- Cách 02.

```
11.float CSoPhuc::getAo()
12.{
13.    return this->Ao;
14.}
```

Bên trong thân phương thức của một lớp đối tượng, this là một con trỏ đối tượng thuộc về lớp mà phương thức đó thuộc về, con trỏ đối tượng this giữ địa chỉ của đối tượng đang gọi thực hiện phương thức. Hơn nữa, *this chính là đối tượng đang gọi thực hiện phương thức.

3. Thiết kế lớp số phức

– Thiết kế các phương thức cung cấp thông tin.

```
11.class CSoPhuc
12.{
13.        …
14.        // Nhóm các phương thức cung cấp thông tin
15.        void Xuat();
16.        friend ostream& operator<<
17.                        (ostream&, CSoPhuc&);
18.        void getThuc();
19.        void getAo();
20.        …
21.};
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức cung cấp thông tin.

```
11.void CSoPhuc::Xuat()
12.{
13.     cout << Thuc;
14.     if (Ao == 0)
15.         return;
16.     if (Ao > 0)
17.        cout << " + " << Ao;
18.     if (Ao < 0)
19.        cout << " - " << abs(Ao);
20.     cout << "i";
21.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức cung cấp thông tin.

```
11.ostream& operator<<(ostream&os,CSoPhuc&x)
12.{
13.    os << x.Thuc;
14.    if (x.Ao == 0)
15.       return os;
16.    if (x.Ao > 0)
17.       os << " + " << x.Ao;
18.    if (x.Ao < 0)
19.       os << " - " << abs(x.Ao);
20.    os << "i";
21.    return os;
22.}
```

3. Thiết kế lớp số phức

– Thiết kế các phương thức cập nhật thông tin.

```
11.class CSoPhuc
12.{
13.        …
14.        // Nhóm các phương thức cập nhật thông tin
15.        void setThuc(float);
16.        void setAo(float);
17.        CSoPhuc& operator=(const CSoPhuc&);
18.        …
19.};
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức cập nhật thông tin.

```
11.void CSoPhuc::setThuc(float thth)
12.{
13.    Thuc = thth;
14.}

15.void CSoPhuc::setAo(float aoao)
16.{
17.    Ao = aoao;
18.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức cập nhật thông tin.

```
11.CSoPhuc& CSoPhuc::operator=(const CSoPhuc& x)
12.{
13.    Thuc = x.Thuc;
14.    Ao = x.Ao;
15.    return *this;
16.}
```

3. Thiết kế lớp số phức

– Thiết kế các phương thức xử lý.

```
11.class CSoPhuc
12.{
13.        // Nhóm các phương thức xử lý
14.        ~CSoPhuc();
15.        CSoPhuc Tong(const CSoPhuc&);
16.        CSoPhuc Hieu(const CSoPhuc&);
17.        CSoPhuc Tich(const CSoPhuc&);
18.        CSoPhuc Thuong(const CSoPhuc&);
19.        CSoPhuc LuyThua(int);
20.        …
21.};
```

3. Thiết kế lớp số phức

– Thiết kế các phương thức xử lý.

```
11.class CSoPhuc
12.{
13.        …
14.        // Nhóm các phương thức xử lý
15.        CSoPhuc operator+(const CSoPhuc&);
16.        CSoPhuc operator-(const CSoPhuc&);
17.        CSoPhuc operator*(const CSoPhuc&);
18.        CSoPhuc operator/(const CSoPhuc&);
19.        CSoPhuc operator^(const CSoPhuc&);
20.        …
21.};
```

3. Thiết kế lớp số phức

– Thiết kế các phương thức xử lý.

```
11.class CSoPhuc
12.{
13.        ...
14.        // Nhóm các phương thức xử lý
15.        CSoPhuc& operator+=(CSoPhuc);
16.        CSoPhuc& operator-=(CSoPhuc);
17.        CSoPhuc& operator*=(CSoPhuc);
18.        CSoPhuc& operator/=(CSoPhuc);
19.        …
20.};
```

3. Thiết kế lớp số phức

– Thiết kế các phương thức xử lý.

```
11.class CSoPhuc
12.{
13.        // Nhóm các phương thức xử lý
14.        float Module();
15.        CSoPhuc LienHop();
16.};
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức xử lý.

```
11.CSoPhuc CSoPhuc::Tong(const CSoPhuc& x)
12.{
13.     CSoPhuc temp;
14.     temp.Thuc = Thuc + x.Thuc;
15.     temp.Ao = Ao + x.Ao;
16.     return temp;
17.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức xử lý.

```
11.CSoPhuc CSoPhuc::Hieu(const CSoPhuc& x)
12.{
13.     CSoPhuc temp;
14.     temp.Thuc = Thuc - x.Thuc;
15.     temp.Ao = Ao - x.Ao;
16.     return temp;

17.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức xử lý.

```
11.CSoPhuc CSoPhuc::Tich(const CSoPhuc& x)
12.{
13.     CSoPhuc temp;
14.     temp.Thuc = (Thuc * x.Thuc - Ao * x.Ao);
15.     temp.Ao = (Thuc * x.Ao - Ao * x.Thuc);

16.     return temp;
17.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức xử lý.

```
11.CSoPhuc CSoPhuc::Thuong(const CSoPhuc& x)
12.{
13.     CSoPhuc temp;
14.     temp.Thuc = (Thuc * x.Thuc + Ao * x.Ao) /
15.                         (x.Thuc * x.Thuc + x.Ao * x.Ao);
16.     temp.Ao = (Ao * x.Thuc - x.Ao * Thuc) /
17.                         (x.Thuc * x.Thuc + x.Ao * x.Ao);
18.     return temp;
19.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức xử lý.

```
11.CSoPhuc CSoPhuc::LuyThua(int n)
12.{
13.     CSoPhuc temp(1,0);
14.     for (int i = 1; i <= n; i++)
15.         temp = temp * *this;
16.     return temp;
17.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức xử lý.

```
11.CSoPhuc CSoPhuc::operator+(const CSoPhuc& x)
12.{
13.     CSoPhuc temp;
14.     temp.Thuc = Thuc + x.Thuc;
15.     temp.Ao = Ao + x.Ao;
16.     return temp;
17.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức xử lý.

```
11.CSoPhuc CSoPhuc::operator-(const CSoPhuc& x)
12.{
13.     CSoPhuc temp;
14.     temp.Thuc = Thuc - x.Thuc;
15.     temp.Ao = Ao - x.Ao;
16.     return temp;
17.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức xử lý.

```
11.CSoPhuc CSoPhuc::operator*(const CSoPhuc& x)
12.{
13.     CSoPhuc temp;
14.     temp.Thuc = (Thuc * x.Thuc - Ao * x.Ao);
15.     temp.Ao = (Thuc * x.Ao - Ao * x.Thuc);
16.     return temp;
17.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức xử lý.

```
11.CSoPhuc CSoPhuc::operator/(const CSoPhuc& x)
12.{
13.     CSoPhuc temp;
14.     temp.Thuc=(Thuc*x.Thuc+Ao*x.Ao)/
15.                              (x.Thuc*x.Thuc+x.Ao*x.Ao);
16.     temp.Ao=(Ao*x.Thuc-x.Ao*Thuc)/
17.                              (x.Thuc*x.Thuc+x.Ao*x.Ao);
18.     return temp;
19.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức xử lý.

```
11.CSoPhuc CSoPhuc::operator^(int n)
12.{
13.     CSoPhuc temp(1,0);
14.     for (int i = 1; i <= n; i++)
15.         temp = temp * *this;
16.     return temp;
17.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức xử lý.

```
11.CSoPhuc& CSoPhuc::operator+=(const CSoPhuc& x)
12.{
13.     Thuc = Thuc + x.Thuc;
14.     Ao = Ao + x.Ao;
15.     return *this;
16.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức xử lý.

```
11.CSoPhuc& CSoPhuc::operator-=(const CSoPhuc& x)
12.{
13.     Thuc = Thuc - x.Thuc;
14.     Ao = Ao - x.Ao;
15.     return *this;
16.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức xử lý.

```
11.CSoPhuc& CSoPhuc::operator*=(const CSoPhuc& x)
12.{
13.     Thuc = Thuc * x.Thuc - Ao * x.Ao;
14.     Ao = Ao * x.Thuc + x.Ao * Thuc;
15.     return *this;
16.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức xử lý.

```
11.CSoPhuc& CSoPhuc::operator/=(const CSoPhuc& x)
12.{
13.    Thuc=(Thuc*x.Thuc+Ao*x.Ao)/(x.Thuc*x.Thuc+x.Ao*x.Ao);
14.    Ao = (Ao*x.Thuc-x.Ao*Thuc)/(x.Thuc*x.Thuc+x.Ao*x.Ao);
15.    return *this;
16.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức xử lý.

```
11.float CSoPhuc::Module()
12.{
13.    return sqrt(Thuc * Thuc + Ao * Ao);
14.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức xử lý.

```
11.CSoPhuc CSoPhuc::LienHop()
12.{
13.     CSoPhuc temp;
14.     temp.Thuc = Thuc;
15.     temp.Ao = - Ao;
16.     return temp;
17.}
```

3. Thiết kế lớp số phức

– Thiết kế các phương thức kiểm tra.

```
11.class CSoPhuc
12.{
13.        // Nhóm các phương thức kiểm tra
14.        int operator>(CSoPhuc&);
15.        int operator<(CSoPhuc&);
16.        int operator>=(CSoPhuc&);
17.        int operator<=(CSoPhuc&);
18.        int operator==(CSoPhuc&);
19.        int operator!=(CSoPhuc&);
20.        …
21.};
```

3. Thiết kế lớp số phức

– Thiết kế các phương thức kiểm tra.

```
11.class CSoPhuc
12.{
13.          …
14.          // Nhóm các phương thức kiểm tra
15.          int ktThuc();
16.          int ktThuanAo();
17.          int ktThuoc1();
18.          int ktThuoc2();
19.          int ktThuoc3();
20.          int ktThuoc4();
21.          …
22.};
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức kiểm tra.

```
11.int CSoPhuc::operator>(const CSoPhuc& x)
12.{
13.    if (Thuc * Thuc + Ao * Ao > x.Thuc * x.Thuc + x.Ao
14.                                          * x.Ao)
15.        return 1;
16.    return 0;
17.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức kiểm tra.

```
11.Int CSoPhuc::operator>=(const CSoPhuc& x)
12.{
13.    CSoPhuc temp = *this;
14.    float a = temp.Module();
15.    CSoPhuc temp2 = x;
16.    float b = temp2.Module();
17.    if (a >= b)
18.        return 1;
19.    return 0;
20.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức kiểm tra.

```
11.Int CSoPhuc::operator<(const CSoPhuc& x)
12.{
13.    CSoPhuc temp = *this;
14.    float a = temp.Module();
15.    CSoPhuc temp2 = x;
16.    float b = temp2.Module();
17.    if (a < b)
18.        return 1;
19.    return 0;
20.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức kiểm tra.

```
11.Int CSoPhuc::operator<=(const CSoPhuc& x)
12.{
13.    CSoPhuc temp = *this;
14.    float a = temp.Module();
15.    CSoPhuc temp2 = x;
16.    float b = temp2.Module();
17.    if (a <= b)
18.        return 1;
19.    return 0;
20.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức kiểm tra.

```
11.Int CSoPhuc::operator!=(const CSoPhuc& x)
12.{
13.    CSoPhuc temp = *this;
14.    float a = temp.Module();
15.    CSoPhuc temp2 = x;
16.    float b = temp2.Module();
17.    if (a != b)
18.        return 1;
19.    return 0;
20.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức kiểm tra.

```
11.float CSoPhuc::ktThuc()
12.{
13.    if(Ao==0)
14.        return 1;
15.    return 0;
16.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức kiểm tra.

```
11.float CSoPhuc::ktThuanAo()
12.{
13.    if(Thuc==0 && Ao!=0)
14.        return 1;
15.    return 0;
16.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức kiểm tra.

```
11.float CSoPhuc::ktThuoc1()
12.{
13.    if(Thuc>0 && Ao>0)
14.        return 1;
15.    return 0;
16.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức kiểm tra.

```
11.float CSoPhuc::ktThuoc2()
12.{
13.    if(Thuc<0 && Ao>0)
14.        return 1;
15.    return 0;
16.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức kiểm tra.

```
11.float CSoPhuc::ktThuoc3()
12.{
13.    if(Thuc<0 && Ao<0)
14.        return 1;
15.    return 0;
16.}
```

3. Thiết kế lớp số phức

– Định nghĩa các phương thức kiểm tra.

```
11.float CSoPhuc::ktThuoc4()
12.{
13.    if(Thuc>0 && Ao<0)
14.        return 1;
15.    return 0;
16.}
```

4. REVIEW

Class Quality

5. CLASS QUALITY

Class Quality

- Booch proposes five metrics to measure the quality of classes:
  - + Coupling
  - + Cohesion
  - + Sufficiency
  - + Completeness
  - + Primitiveness

Coupling

- How closely do classes rely on each other?
- Inheritance makes for strong coupling (generally a bad thing) but takes advantage of the re-use of an abstraction (generally a good thing).

Cohesion

- How tied together are the state and behavior of a class? Does the abstraction model a cohesive, related, integrated thing in the problem space?

Sufficiency

- Does the class capture enough of the details of the thing being modeled to be useful?

Completeness

- Does the class capture all of the useful behavior of the thing being modeled to be re-usable? What about future users (reusers) of the class?
- How much more time does it take to provide completeness?
- Is it worth it?

Primitiveness

- Do all the behaviors of the class satisfy the condition that they can only be implemented by accessing the state of the class directly? If a method can be written strictly in terms of other methods in the class, it isn't primitive.
- Primitive classes are smaller, easier to understand, with less coupling between methods, and are more likely to be reused. If you try to do too much in the class, then you're likely to make it difficult to use for other designers.

Primitiveness

- Sometimes issues of efficiency or interface ease-of-use will suggest you violate the general recommendation of making a class primitive. For example, you might provide a general method with many arguments to cover all possible uses, and a simplified method without arguments for the common case.
- The truism "Make the common case fast" holds, but in this case 'fast 'means "easy". Easy may violate primitive.

Cảm ơn quí vị đã lắng nghe

Nhóm tác giả
TS. Nguyễn Tấn Trần Minh Khang