Thuộc tính và phương thức lớp

```
Stack *Stack::create(int sz)
{
    Stack *ps = new Stack(sz);
    if (!ps->st)
    {
        delete ps;
        return NULL;
    }
    return ps;
}
void main()
{
    Stack *ps = new Stack(50000); // ko biet tao duoc stack khong
    Stack *pr = Stack::create(50000);
    if (!pr)
        { cout << "Khong the tao stack"; return; }
    else
        pr->push(5); // ...
}
```

Phương thức lớp Stack::create được dùng thay cho phương thức thiết lập

```
#import <Foundation/Foundation.h>

@interface CDate : NSObject
+(BOOL)leapYear : (int)y;
+(int)dayOfMonth: (int)m andYear : (int)y;
+(BOOL)validDay: (int)d andMonth : (int)m andYear : (int)y;
-(void)input;
-(void)print;

@end
```

Phương thức lớp

Thuộc tính và phương thức lớp

Thuộc tính lớp

```
#import "CDate.h"

@implementation Cdate
{
    int day, month, year;
}
static const int dayTab[][13] = {
    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
    {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
};

BOOL betw(int x, int a, int b) {
    return a <= x && x <= b;
}

+(BOOL)leapYear:(int)y {
    return (y % 400 == 0) || (y % 4 == 0 && y % 100 != 0);
}
```

```
+(int)dayOfMonth : (int)m andYear : (int)y {
   BOOL b = [CDate leapYear : (y)];
   return dayTab[b][m];
}

+(BOOL)validDay : (int)d andMonth : (int)m
   andYear : (int)y {
   return betw(d, 1, [CDate dayOfMonth : m
   andYear : y]);
}

-(void)print {
   NSLog(@"%02d / % 02d / %d: ", day, month,
   year);
}
```

```
-(void)input {
   int d, m, y;
   scanf("%d%d%d", &d, &m, &y);
   while (![CDate validDay : d andMonth : m
   andYear : y]) {
      NSLog(@"Enter valid date(d m y : ");
      scanf("%d%d%d", &d, &m, &y);
   }
   day = d;
   month = m;
   year = y;
}

@end
```

```objc
#import <Foundation/Foundation.h>
#import "cdate.h"

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        CDate *d = [[CDate alloc] init];
        NSLog(@"Enter valid date : ");
        [d input];
        [d print];
    }
    return 0;
}
```

Thiết lập và huỷ bỏ đối tượng

- Ta cần kiểm soát khi nào phương thức thiết lập được gọi, khi nào phương thức huỷ bỏ được gọi.
- Khi nào đối tượng thiết lập được gọi? Khi đối tượng được tạo ra.
- Khi nào phương thức huỷ bỏ được gọi? Khi đối tượng bị huỷ đi.
- Thời gian từ khi đối tượng được tạo ra đến khi nó bị huỷ đi được gọi là thời gian sống.

Thời gian sống

- Thời gian sống của đối tượng khác nhau tuỳ thuộc cách đối tượng được lưu trữ.

1. Đối tượng tự động.
2. Đối tượng tĩnh
3. Đối tượng trong vùng nhớ tự do
4. Đối tượng tạm thời
5. Đối tượng thành phần

Đối tượng tự động

- Đối tượng tự động (automatic objects) là đối tượng được tự động sinh ra và tự động bị huỷ đi. Đối tượng tự động có thể là đối tượng địa phương, là tham số truyền bằng giá trị hoặc là đối tượng giá trị trả về.
- Đối tượng địa phương
  - Được khai báo, định nghĩa bên trong một khối.
  - Được tự động sinh ra khi chương trình thực hiện ngang dòng lệnh chứa định nghĩa và bị huỷ đi sau khi chương trình hoàn tất khối chứa định nghĩa đó.

Đối tượng tự động

- Khi khởi động một đối tượng bằng một đối tượng cùng kiểu, cơ chế tạo đối tượng mặc nhiên là sao chép từng thành phần, do đó đối tượng được khởi động có khả năng sẽ chia sẻ tài nguyên với đối tượng nguồn.
- Khi 2 đối tượng này bị huỷ đi, phần *tài nguyên chung sẽ bị giải phóng 2 lần!*.

Đối tượng địa phương

```
inline char *__strdup(const char *s) {
    return strcpy(new char[strlen(s) + 1], s);
}

class String {
    char *p;
public:
    String(const char *s) { p = __strdup(s); }
    ~String() { if (p) delete p;}
    int getLength() const { return strlen(p); }
    void print();
    void set(const char *s);
    void input();
    void concat(const String &b);
};
```

Đối tượng địa phương

```
#include <iostream>
using namespace std;
#include "string.h"
int main()
{
   String a("This is a string");
   a.print();
   cout << "\n";
   String b = a; // String b(a);
   b.print();
   cout << "\n";
   return 0;
}
```

Phương thức huỷ bỏ được gọi cho cả a và b, phần tài nguyên chung bị giải phóng 2 lần

output

```
new 00007FF6D68EBE28
a: Dai Hoc Tu Nhien
b: Dai Hoc Tu Nhien
Dai Hoc Tu Nhien
delete 000001CAB9DFF860
delete 000001CAB9DFF860
```

Đối tượng là tham trị

```
class String {
   char *p;
public:
   String(const char *s) { p = __strdup(s); }
   ~String() { if (p) delete p;}
   int getLength() const { return strlen(p); }
   void print();
   void set(const char *s);
   void input();
   void concat(const String &b);
};
```

Đối tượng là tham trị

- Đối tượng là tham số hàm, truyền bằng giá trị thì tham số hình thức là bản sao của tham số thực sự, nên có nội dung vật lý giống tham số thực sự do cơ chế sao chép từng thành phần.
- Khi kết thúc hàm, đối tượng tham trị bị huỷ, phương thức huỷ bỏ được gọi, phần *tài nguyên chung sẽ bị giải phóng* trong khi tham số thực sự vẫn tồn tại*!*.

Đối tượng là tham trị

```
class String
{
   char *p;
public:
   String(const char *s) { p = __strdup(s);
   }
   ~String() { if (p) { delete[] p;}
   int getLength() const { return strlen(p);
   }
   void print() const;
   void input();
   int compare(String s) const;
};
```

Đối tượng là tham trị

```
int String::compare(String s) const
{
   return strcmp(p, s.p);
}
void main()
{
   String a("Dai Hoc Tu Nhien");
   String b("Dai Hoc Bach Khoa");
    cout << "a = "; a.print(); cout << "\n";
    cout << "b = "; b.print(); cout << "\n";
    int c = a.compare(b);
   cout << (c > 0 ? "a > b" : c == 0 ? "a = b" :
   "a < b") << "\n";
}
```

Phương thức huỷ bỏ được gọi bao nhiêu lần?

Output

```
new 00007FF6AB1EBE28
new 00007FF6AB1EBC70
a = Dai Hoc Tu Nhien
b = Dai Hoc Bach Khoa
delete 0000027934D42D80
a > b
delete 0000027934D42D80
```

Đối tượng giá trị trả về

```
class String
{
   char *p;
public:
   String(const char *s = "Alibaba") { p =
   __strdup(s);}
   ~String() { if (p) { delete[] p;}
   int getLength() const { return strlen(p); }
   int compare(String s) const;
   String upCase() const;
};
```

Đối tượng giá trị trả về

```
String String::upCase() const
{
   String r = *this;
   strupr(r.p);
   return r;
}

void main()
{
   String a("Dai Hoc Tu Nhien");
   cout << "a = "; a.print(); cout << "\n";
   String A;
   A = a.upCase();
   cout << "a = "; a.print(); cout << "\n";
   cout << "A = "; A.print(); cout << "\n";
}
```

Output

```
new 00007FF658D9BE28
new 00007FF658D9BC70
a: Dai Hoc Tu Nhien
b: Dai Hoc Bach Khoa
aa: Dai Hoc Tu Nhien
delete 0000014732F41050
b: ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
a > b
a = Dai Hoc Tu Nhien
delete 0000014732F416E0
a = ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
A = ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
delete 0000014732F416E0
```

Đối tượng giá trị trả về

- Đối tượng giá trị trở về là bản sao của biểu thức trả về, do đó có sự chia sẻ tài nguyên (*sai*).
- Trong ví dụ trên, tài nguyên còn bị chia sẻ do phép gán.
- Có thể thay phép gán bằng khởi động

```
String A a.upCase();
```

Nhưng vẫn còn lỗi chia sẻ tài nguyên do đối tượng giá trị trả về.

Phương thức thiết lập bản sao

- Các lỗi sai nêu trên gây ra do sao chép đối tượng, ta có thể khắc phục bằng cách "dạy" trình biên dịch sao chép đối tượng một cách luận lý thay vì sao chép từng thành phần theo nghĩa vật lý. Điều đó được thực hiện nhờ phương thức thiết lập bản sao.
- Phương thức thiết lập bản sao là phương thức thiết lập có tham số là đối tượng cùng kiểu, dùng để sao chép đối tượng.

Phương thức thiết lập bản sao

- Phương thức thiết lập bản sao thực hiện sao chép theo nghĩa logic, thông thường là tạo nên tài nguyên mới (sao chép sâu).
- Phương thức thiết lập bản sao cũng có thể chia sẻ tài nguyên cho các đối tượng (sao chép nông). Trong trường hợp này, cần có cơ chế để kiểm soát việc huỷ bỏ đối tượng.

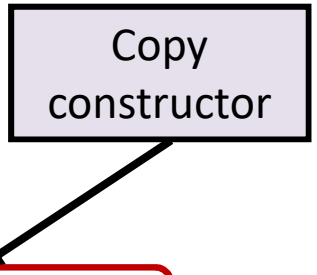
Phương thức thiết lập bản sao

- Tham số của phương thức thiết lập bản sao bắt buộc là tham chiếu.
- Phương thức thiết lập bản sao có thể được dùng để sao chép nông, tài nguyên vẫn được chia sẻ nhưng có một biến đếm để kiểm soát.

Lưu ý:

- Nếu đối tượng không có tài nguyên riêng thì không cần phương thức thiết lập bản sao.
- Khi truyền tham số là đối tượng thuộc lớp có phương thức thiết lập bản sao, ta nên truyền bằng tham chiếu và có thêm từ khoá const.

```
class String
{
    char *p;
public:
    String(const char *s = "Alibaba") { p =
    __strdup(s); }
    String(const String &s) { p = __strdup(s.p); }
    ~String() { if (p) { cout << "delete " << (void
    *)p << "\n";  delete[] p; } }
    int getLength() const { return strlen(p); }
    void print() const;
    void input();
    void set(const char *s);
    void concat(const String &b);
    int compare(String s) const;
    String upCase() const;
};
```



```
inline int String::compare(String s)
const
{
   return strcmp(p, s.p);
}

inline String String::upCase() const
{
   String r = *this;
   strupr(r.p);
   return r;
}
```

```
void main()
{
    String a("Dai Hoc Tu Nhien");
    String b("Dai Hoc Bach Khoa");
    String aa = a;
    cout << "a: "; a.print(); cout << "\n";
    cout << "b: "; b.print(); cout << "\n";
    cout << "aa: "; aa.print(); cout << "\n";
    int c = a.compare(b);
    cout << "b: "; b.print(); cout << "\n";
    cout << (c > 0 ? "a > b" : c == 0 ? "a = b" : "a
    < b") << "\n";
    cout << "a = "; a.print(); cout << "\n";
    String A = a.upCase();
    cout << "a = "; a.print(); cout << "\n";
    cout << "A = "; A.print(); cout << "\n";
}
```

Output

```
a: Dai Hoc Tu Nhien
b: Dai Hoc Bach Khoa
aa: Dai Hoc Tu Nhien
delete 000001D7BFE46BC0
b: Dai Hoc Bach Khoa
a > b
a = Dai Hoc Tu Nhien
delete 000001D7BFE47250
a = Dai Hoc Tu Nhien
A = DAI HOC TU NHIEN
delete 000001D7BFE46A80
delete 000001D7BFE46B20
delete 000001D7BFE475C0
delete 000001D7BFE469E0
```

Sao chép nông và sao chép sâu

- Dùng phương thức thiết lập bản sao như trên, trong đó đối tượng mới có tài nguyên riêng *là sao chép sâu.*
- Ta có thể *sao chép nông* bằng cách chia sẻ tài nguyên và dùng một *biến đếm* để kiểm soát số thể hiện các đối tượng có chia sẻ cùng tài nguyên.
- Khi một đối tượng thay đổi nội dung, nó phải được tách ra khỏi các đối tượng dùng chung tài nguyên, nói cách khác, nó phải có tài nguyên riêng (như sao chép sâu).

```
class StringRep
{
    friend class String;
    char *p;
    int n;
    StringRep(const char *s) { p = strdup(s); n = 1; }
    ~StringRep() { cout << "delete " << (void *)p << "\n";
    delete [] p;}
};

class String {
    StringRep *rep;
public:
   String(const char *s = "Alibaba"){ rep = new StringRep(s); }
   String(const String &s) { rep = s.rep; rep->n++; }
   ~String();
   void print() const { cout << rep->p; }
   int compare(String s) const;
   String upCase() const;
   void toUpper();
};
```

```
String::~String()
{
   if (--rep->n <= 0)
      delete rep;
}

int String::compare(String s) const
{
   return strcmp(rep->p, s.rep->p);
}

String String::upCase() const
{
   String r(rep->p);
   strupr(r.rep->p);
   return r;
}
```

```
int main()
{
    String a("Dai Hoc Tu Nhien");
    String b("Dai Hoc Bach Khoa");
    String aa = a;
    cout << "a: "; a.print(); cout << "\n";
    cout << "b: "; b.print(); cout << "\n";
    cout << "aa: "; aa.print(); cout << "\n";
    int c = a.compare(b);
    cout << "b: "; b.print(); cout << "\n";
    cout << (c > 0 ? "a > b" : c == 0 ? "a = b" : "a < b")
    << "\n";
    cout << "a = "; a.print(); cout << "\n";
    String A = a.upCase();
    cout << "a = "; a.print(); cout << "\n";
    cout << "A = "; A.print(); cout << "\n";
    return 0;
}
```

```
a: Dai Hoc Tu Nhien
b: Dai Hoc Bach Khoa
aa: Dai Hoc Tu Nhien
b: Dai Hoc Bach Khoa
a > b
a = Dai Hoc Tu Nhien
a = Dai Hoc Tu Nhien
A = DAI HOC TU NHIEN
delete 00000146B7A90750
delete 00000146B7A901B0
delete 00000146B7A90390
Press any key to continue . . .
```

Đối tượng tĩnh

Trong C++

- Đối tượng *tĩnh* có thể là đối tượng được định nghĩa toàn cục, được định nghĩa có thêm từ khoá static (toàn cục hoặc địa phương).
- Đối tượng toàn cục hoặc tĩnh toàn cục được tạo ra khi *bắt đầu chương trình* và bị huỷ đi khi *kết thúc chương trình.*
- Đối tượng tĩnh địa phương được tạo ra khi chương trình thực hiện đoạn mã chứa định nghĩa đối tượng lần đầu tiên và bị huỷ đi khi *kết thúc chương trình.*

Đối tượng tĩnh

Trình tự thực hiện chương trình gồm:

1. Gọi phương thức thiết lập cho các đối tượng toàn cục
2. Thực hiện hàm main()
3. Gọi phương thức huỷ bỏ cho các đối tượng toàn cục

Đối tượng tĩnh

Hãy sửa đoạn chương trình sau:

```
void main()
{
   cout << "Hello, world.\n";
}
```

Để có xuất liệu:

Entering a C++ program saying...

Hello, world.

And then exitting...

Yêu cầu không thay đổi hàm main() dưới bất kỳ hình thức nào.

```
void main()
{
   cout << "Hello, world.\n";
}

class Dummy
{
public:
   Dummy() { cout << "Entering a C++ program
   saying...\n"; }
   ~Dummy() { cout << "And then
   exitting..."; }
}d;
```

Đối tượng là thành phần của lớp

- Đối tượng có thể là thành phần của đối tượng khác, khi một đối tượng thuộc lớp "lớn" được tạo ra, các thành phần của nó cũng được tạo ra. Phương thức thiết lập (nếu có) sẽ được tự động gọi cho các đối tượng thành phần.
- Nếu đối tượng thành phần phải được cung cấp tham số khi thiết lập thì đối tượng kết hợp (đối tượng lớn) phải có phương thức thiết lập để cung cấp tham số thiết lập cho các đối tượng thành phần.

Đối tượng là thành phần của lớp

- Cú pháp để khởi động đối tượng thành phần trong C++ là dùng dấu hai chấm (:) theo sau bởi tên thành phần và tham số khởi động.
- Khi đối tượng kết hợp bị huỷ đi thì các đối tượng thành phần của nó cũng bị huỷ đi, nghĩa là phương thức huỷ bỏ sẽ được gọi cho các đối tượng thành phần, sau khi phương thức huỷ bỏ của đối tượng kết hợp được gọi.
- Trong Objective C, đối tượng thành phần không phải kiểu cơ bản bắt buộc là *con trỏ*.

```
class Point
{
   double x, y;
public:
   Point(double xx, double yy) { x = xx; y = yy; }
   // ...
};

class Triangle
{
   Point A, B, C;
public:
   void draw() const;
   // ...
};

Triangle t;// Error
```

```
class String
{
    char *p;
public:
    String(const char *s) { p = __strdup(s); }
    String(const String &s) { p = __strdup(s.p); }
    ~String() { if (p) delete[] p; }
    //...
};
class Student
{
    String MaSo;
    String HoTen;
    int NamSinh;
public:
};
Student s; // Error
```

Khởi động đối tượng thành phần

```
class Point
{
    double x, y;
public:
    Point(double xx, double yy) { x = xx; y = yy; }
    // ...
};
class Triangle
{
    Point A, B, C;
public:
    Triangle(double xA, double yA, double xB, double yB,
    double xC, double yC): A(xA, yA), B(xB, yB), C(xC, yC)
    {}
    void draw() const;
    // ...
};

Triangle t(100, 100, 200, 400, 300, 300); // Ok
```

```
class String
{
    char *p;
    public:
    String(const char *s) { p = __strdup(s); }
    String(const String &s) { p = __strdup(s.p); }
    ~String() { if (p) delete[] p; }
    //...
};
class Student
{
    String id;
    String name;
    int birthYear;
public:
    Student(const char *n, const char *i, int y): name(n),
    id(i) { birthYear = y; }
};
Student s("To Van Ve", "20182145", 2000); // Ok
```

Đối tượng là thành phần của lớp

- Cú pháp để khởi động đối tượng thành phần trong C++ dùng dấu hai chấm (:) cũng được dùng cho đối tượng thành phần thuộc kiểu cơ bản.

```
class Point
{
   double x, y;
public:
   Point(double xx, double yy):x(xx), y(yy) { }
   // ...
};
```

Đối tượng là thành phần của lớp

```
class Student
{
   String id;
   String name;
   int birthYear;
public:
   Student(const char *n, const char *i, int
   y): name(n), id(i), birthYear(y) { }
};

Student s("Vo Van Ve", "20182145", 2000);// Ok
```

Đối tượng thành phần – Huỷ bỏ

- Khi đối tượng kết hợp bị huỷ bỏ, các đối tượng thành phần của nó cũng bị huỷ bỏ. Phương thức huỷ bỏ cho các đối tượng thành phần sẽ được gọi một cách tự động.

Đối tượng hành phần – huỷ bỏ

```
class Student
{
   String id;
   String name;
   int birthYear;
   int nSubjects;
   double *aPoints;
public:
   Student(const char *n, const char *i, int y,
   int nS, double *d) : name(n), id(i),
   birthYear(y), nSubjects(nS) {
   memcpy(aPoints = new double[nSubjects], d,
   nSubjects * sizeof(double));
   }
   ~Student() { delete[] aPoints; }
};
```

Đối tượng là thành phần của mảng

- Khi một mảng được tạo ra, các phần tử của nó cũng được tạo ra, do đó phương thức thiết lập sẽ được gọi cho từng phần tử một.
- Vì không thể cung cấp tham số khởi động cho tất cả các phần tử của mảng nên khi khai báo mảng, mỗi đối tượng trong mảng phải có khả năng tự khởi động, nghĩa là có thể được thiết lập không cần tham số.

Đối tượng là thành phần của mảng

- Đối tượng có khả năng tự khởi động trong các trường hợp sau:
  1. Lớp không có phương thức thiết lập.
  2. Lớp có phương thức thiết lập không tham số.
  3. Lớp có phương thức thiết lập mà mọi tham số đều có giá trị mặc nhiên.

Đối tượng là thành phần của mảng

```
class Point {
   double x, y;
public:
   Point(double xx, double yy) { x = xx; y = yy; }
   // ...
};
Point ap[5]; // Error

class String {
   char *p;
public:
   String(const char *s) { p = __strdup(s); }
   String(const String &s) { p = __strdup(s.p); }
   ~String() { if (p) delete[] p; }
   //...
};
String as[3]; // Error
```

Đối tượng là thành phần của mảng

```
class Student
{
   String id;
   String name;
   int birthYear;
public:
   Student(const char *n, const char *i, int
   y): name(n), id(i), birthYear(y) { }
   // ...
};

Student aS[4]; // Error
```

Đối tượng là thành phần của mảng

```
class Point {
   double x, y;
public:
    Point(double xx, double yy) { x = xx; y = yy; }
    Point() : x(0), y(0) {}
    // ...
};
Point ap[5]; // Ok

class String {
   char *p;
public:
   String(const char *s = "Alibaba") { p = __strdup(s); }
   String(const String &s) { p = __strdup(s.p); }
   ~String() { if (p) delete[] p; }
   //...
};
String as[3]; // Ok
```

Đối tượng là thành phần của mảng

```
class Student
{
   String id;
   String name;
   int birthYear;
public:
   Student(const char *n = , const char *i,
   int y): name(n), id(i), birthYear(y) { }
   // ...
};

Student aS[4]; // Error
```

PTTL với tham số mặc nhiên

```
class Student
{
   String id;
   String name;
   int birthYear;
public:
   Student(const char *n = "Aladin", const
   char *i = "20182132", int y = 2000):
   name(n), id(i), birthYear(y) { }
   // ...
};

Student aS[4]; // Ok
```

PTTL với tham số mặc nhiên

```
class Student
{
   String id;
   String name;
   int birthYear;
public:
   Student(const char *n = "Aladin", const
   char *i = "20182132", int y = 2000):
   name(n), id(i), birthYear(y) { }
   // ...
};

Student aS[4]; // Ok
```

PTTL không tham số

```
class Student
{
   String id;
   String name;
   int birthYear;
public:
   Student(const char *n, const char *i, int y):
   name(n), id(i), birthYear(y) { }
   Student() : name("Aladin"), id("20182132"),
   birthYear(2000) { }

   // ...
};

Student aS[4]; // Ok
```

Đối tượng được cấp phát động

- Đối tượng được cấp phát động là đối tượng được tạo ra bằng phép toán new và bị huỷ đi bằng phép toán delete
- Phép toán new cấp đối tượng trong vùng heap (hay vùng free store) và gọi phương thức thiết lập cho đối tượng được cấp.
- Dùng new có thể cấp một đối tượng và dùng delete để huỷ một đối tượng.
- Dùng new và delete cũng có thể cấp nhiều đối tượng và huỷ nhiều đối tượng.

Đối tượng được cấp phát động

```
class Point {
   double x, y;
public:
   Point(double xx, double yy) { x = xx; y = yy; }
   // ...
};

class String {
   char *p;
public:
   String(const char *s) { p = __strdup(s); }
   String(const String &s) { p = __strdup(s.p); }
   ~String() { if (p) delete[] p; }
   //...
};
```

Cấp và huỷ 1 đối tượng

```
int *pi = new int;
int *pj = new int(15);
Point *pd = new Point(20, 40);
String *pa = new String("Nguyen Van A");
//...
//...



delete pa;
delete pd;
delete pj;
delete pi;
```

Đối tượng được cấp phát động

```
class Point {
   double x, y;
public:
   Point(double xx, double yy) { x = xx; y = yy; }
   Point() : x(0), y(0) {}
   // ...
};

class String {
   char *p;
public:
   String(const char *s = "Alibaba") { p =
   __strdup(s); }
   String(const String &s) { p = __strdup(s.p); }
   ~String() { if (p) delete[] p; }
   //...
};
```

Cấp và huỷ nhiều đối tượng

```
// Cac doi tuong phai co
// kha nang tu khoi dong

int *pai = new int[10];
Point *pad = new Point[5];// Ok
String *pas = new String[5]; // Ok
// ...
// ...


delete[] pas;
delete[] pad;
delete[] pai;
```

Giao diện và chi tiết cài đặt

- Lớp có hai phần tách rời, một là phần giao diện khai báo trong phần public để người sử dụng "thấy" và sử dụng, và hai là chi tiết cài đặt bao gồm dữ liệu khai báo trong phần private của lớp và chi tiết mã hoá các hàm thành phần, vô hình đối với người dùng.
- Ta có thể thay đổi uyển chuyển chi tiết cài đặt, nghĩa là có thể thay đổi tổ chức dữ liệu của lớp, cũng như có thể thay đổi chi tiết thực hiện các phương thức (do sự thay đổi tổ chức dữ liệu hoặc để cải tiến giải thuật). Nhưng phải bảo đảm *không thay đổi phần giao diện* để không ảnh hưởng đến người sử dụng, và do đó không làm đổ vỡ kiến trúc của hệ thống.

Giao diện và chi tiết cài đặt

Ví dụ:

- Lớp Time biểu diễn khái niệm thời điểm, có thể được cài đặt bằng ba thuộc tính dữ liệu biểu diễn giờ phút giây hoặc bằng một thuộc tính dữ liệu là tổng số giây tính từ 0g.
- Lớp Stack biểu diễn cấu trúc dữ liệu hoạt động theo nguyên tắc LIFO có để được cài đặt bằng mảng hoặc danh sách liên kết

Time: hour, minute, second

```
class Time
{
   int hour, minute, second;
   static bool valid(int h, int m, int s);
public:
   Time(int h = 0, int m = 0, int s = 0) { set(h, m,
   s); }
   void set(int h, int m, int s);
   int getHour() const { return hour; }
   int getMinute() const { return minute; }
   int getSecond() const { return second; }
   void input();
   void print() const;
   void increase();
   void decrease();
};
```

Time: totalSeconds

```
const long SECONDS_OF_DAY = 3600L * 24;
class Time
{
  int totalSeconds;
  static bool valid(int h, int m, int s);
public:
  Time(int h = 0, int m = 0, int s = 0) { set(h, m, s); }
  void set(int h, int m, int s);
  int getHour() const { return totalSeconds / 3600; }
  int getMinute() const { return (totalSeconds % 3600) /
  60; }
  int getSecond() const { return totalSeconds % 60; }
  void input();
  void print() const;
  void increase();
  void decrease();
};
```

Các nguyên tắc xây dựng lớp

- Khi điều ta nghĩ đến là một khái niệm riêng rẽ, xây dựng lớp biểu diễn khái niệm đó. Ví dụ lớp Sinh Viên, lớp Tam Giác, lớp Người, lớp Phân Số…
- Khi điều ta nghĩ đến là một thực thể cụ thể riêng biệt, tạo đối tượng thuộc lớp. Ví dụ đối tượng Sinh viên “Nguyen Van A” (và các thuộc tính khác như mã số, năm sinh…).

Các nguyên tắc xây dựng lớp

- Lớp là biểu diễn cụ thể của một khái niệm, vì vậy lớp luôn luôn là *danh từ*.
- Các thuộc tính của lớp là các thành phần dữ liệu, nên chúng luôn luôn là *danh từ*.
- Các hàm thành phần là các thao tác chỉ rõ hoạt động của lớp nên các hàm này là *động từ*.
- Các thuộc tính dữ liệu phải vừa đủ để mô tả khái niệm, *không dư*, không thiếu.

Các nguyên tắc xây dựng lớp

- Các thuộc tính có thể được suy diễn từ những thuộc tính khác thì dùng phương thức để thực hiện tính toán. Chu vi, diện tích tam giác, hình ellipse là thuộc tính suy diễn.

```
class CEllipse // Sai
{
   Point T;
   double rx, ry;
   double area, perimeter;
public:
};
```

```
class CEllipse // Đúng
{
   Point T;
   double rx, ry;
public:
   double getArea() const;
   double getPerimeter() const;
};
```

Các nguyên tắc xây dựng lớp

- Cá biệt có thể có một số thuộc tính suy diễn đòi hỏi nhiều tài nguyên hoặc thời gian để thực hiện tính toán, ta có thể khai báo là dữ liệu thành phần. Ví dụ tuổi thọ trung bình của một quốc gia.

```
class Country
{
    long population;
    double area;
    double avgLogevity;
    //...
public:
    double getAvgLongevity() const;
    //...
};
```

Các nguyên tắc xây dựng lớp

- Chi tiết cài đặt, bao gồm dữ liệu và phần mã hoá các hàm thành phần có thể thay đổi uyển chuyển nhưng phần giao diện, nghĩa là phần khai báo các hàm thành phần cần phải cố định để không ảnh hưởng đến người sử dụng. Tuy nhiên nên cố gắng cài đặt dữ liệu một cách tự nhiên theo đúng khái niệm.

Các nguyên tắc xây dựng lớp

- Dữ liệu thành phần nên được kết hợp thay vì phân rã.

```
// Nên
class CEllipse
{
   Point T;
   double rx, ry;
public:
// ...
};
```

```
// Không Nên
class CEllipse
{
   double xT, yT;
   double rx, ry;
public:
// ...
};
```

Các nguyên tắc xây dựng lớp

- Trong mọi lớp, định nghĩa một hoặc nhiều phương thức để khởi động đối tượng.
- Nên có phương thức thiết lập có khả năng tự khởi động không cần tham số.
- Nếu đối tượng tài nguyên luận lý thì phải có phương thức thiết lập, phương thức thiết lập *bản sao* để khởi động đối tượng bằng đối tượng cùng kiểu và có phương thức huỷ bỏ để dọn dẹp tài nguyên. (Ngoài ra còn phải có phép gán, chương tiếp theo, C++).

Các nguyên tắc xây dựng lớp

- Ngược lại, đối tượng đơn giản không cần tài nguyên riêng thì không cần phương thức thiết lập bản sao và cũng không cần phương thức huỷ bỏ.