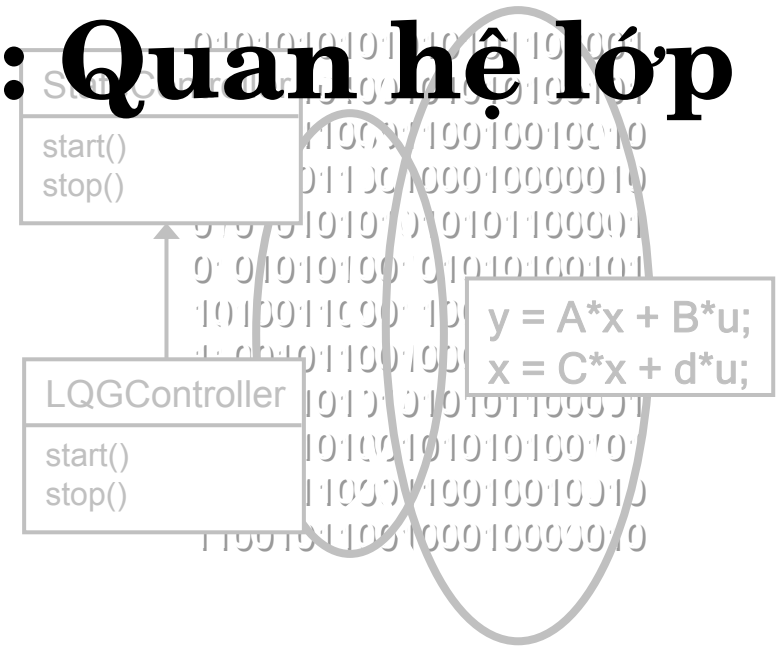
Kỹ thuật lập trình

Chương 7: Quan hệ lớp



12/3/2007

Nội dung chương 7

7.1 Quan hệ lớp

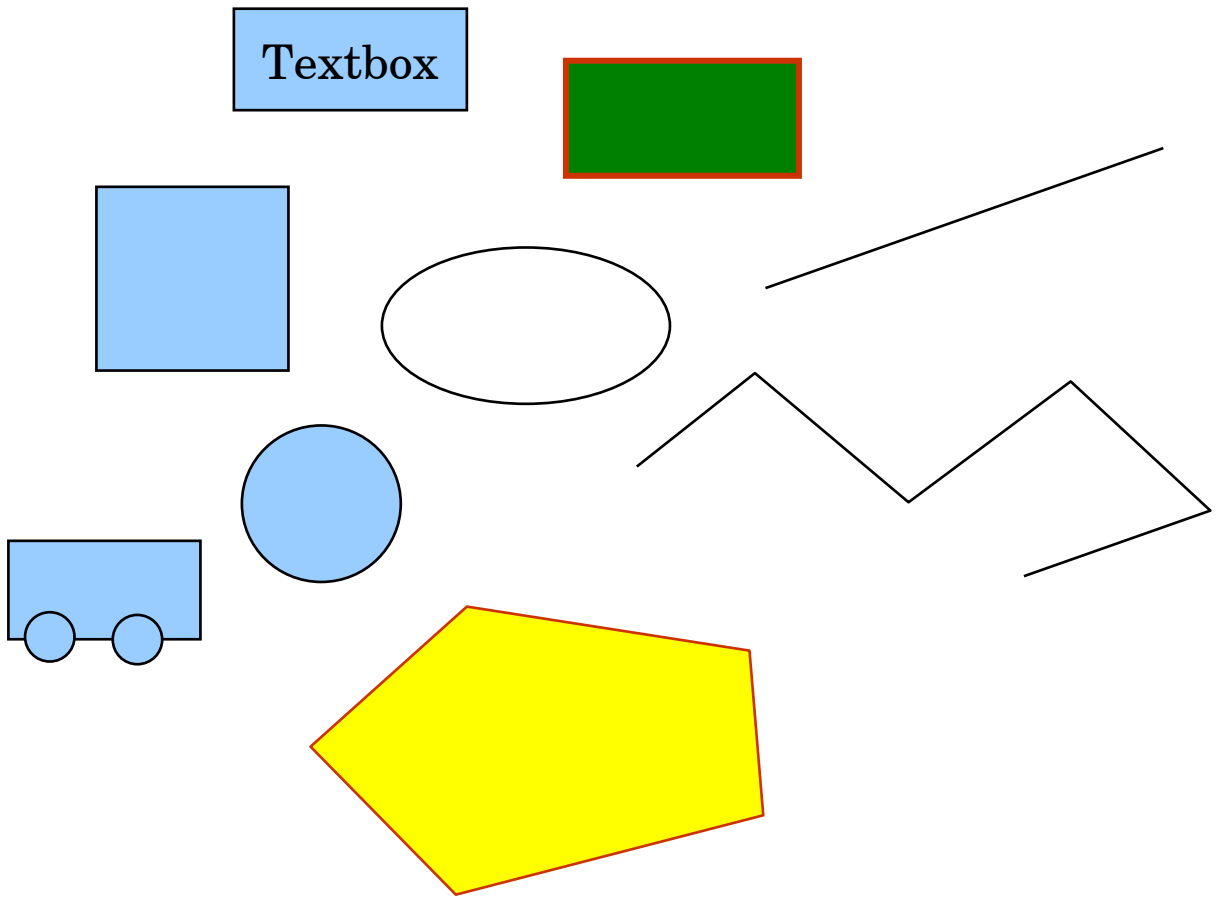
7.2 Dẫn xuất và thừa kế

7.3 Hàm ảo và nguyên lý đa hình/đa xạ

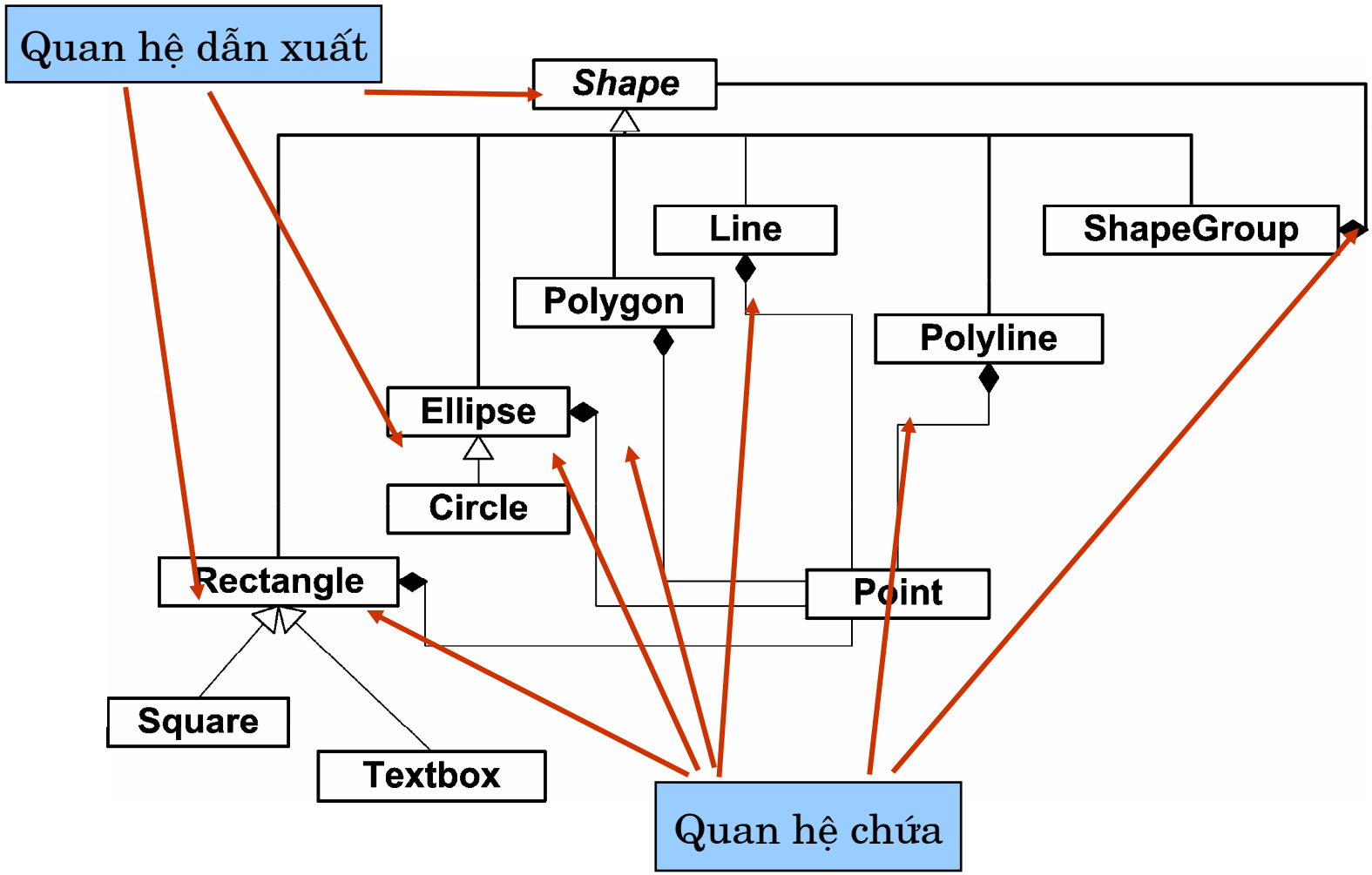
7.4 Ví dụ thư viện khối chức năng

7.1 Phân loại quan hệ lớp

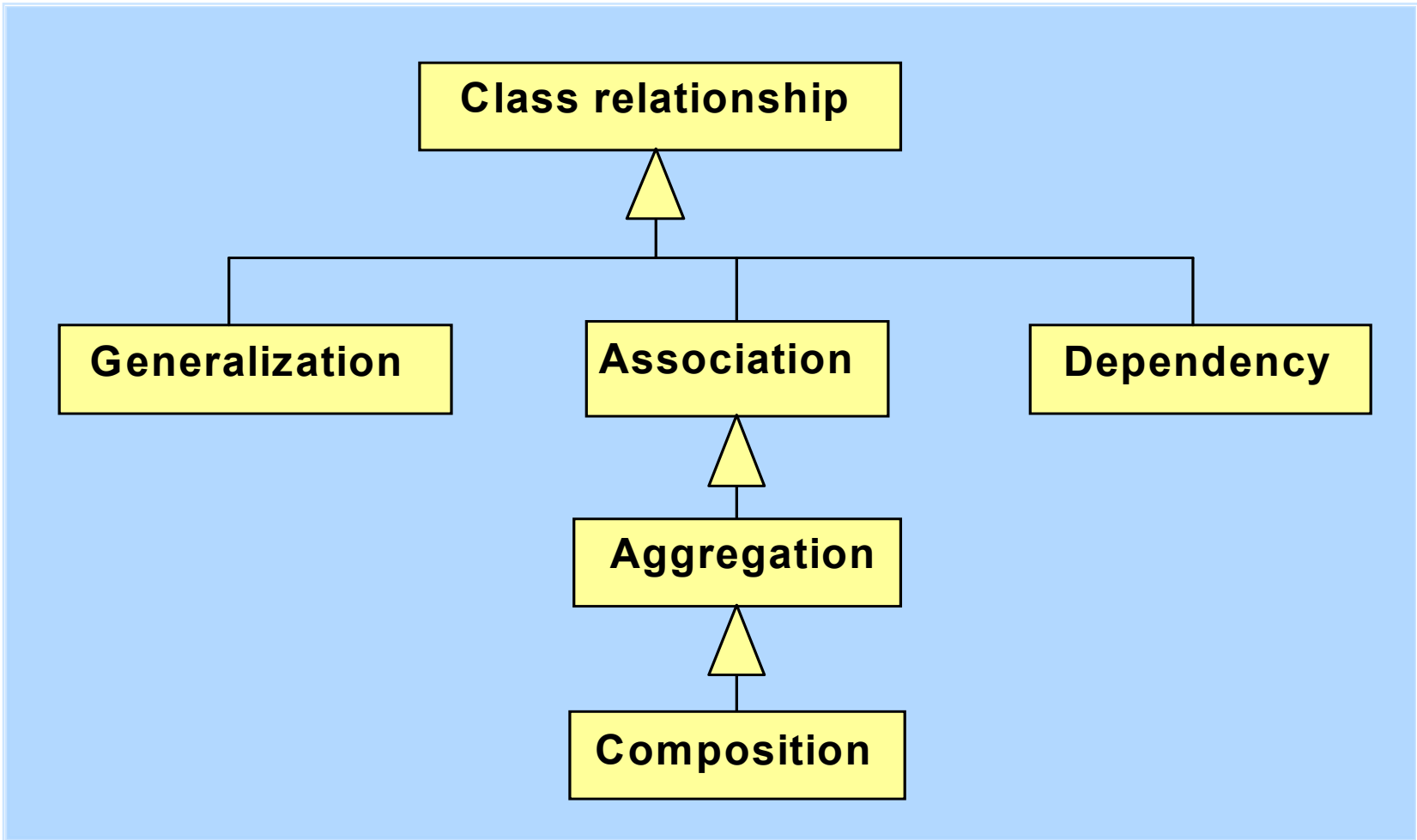
- Ví dụ minh họa: Các lớp biểu diễn các hình vẽ trong một chương trình đồ họa
  - Rectangle
  - Square
  - Ellipse
  - Circle
  - Line
  - Polygon
  - Polyline
  - Textbox
  - Group



Biểu đồ lớp (Unified Modeling Language)

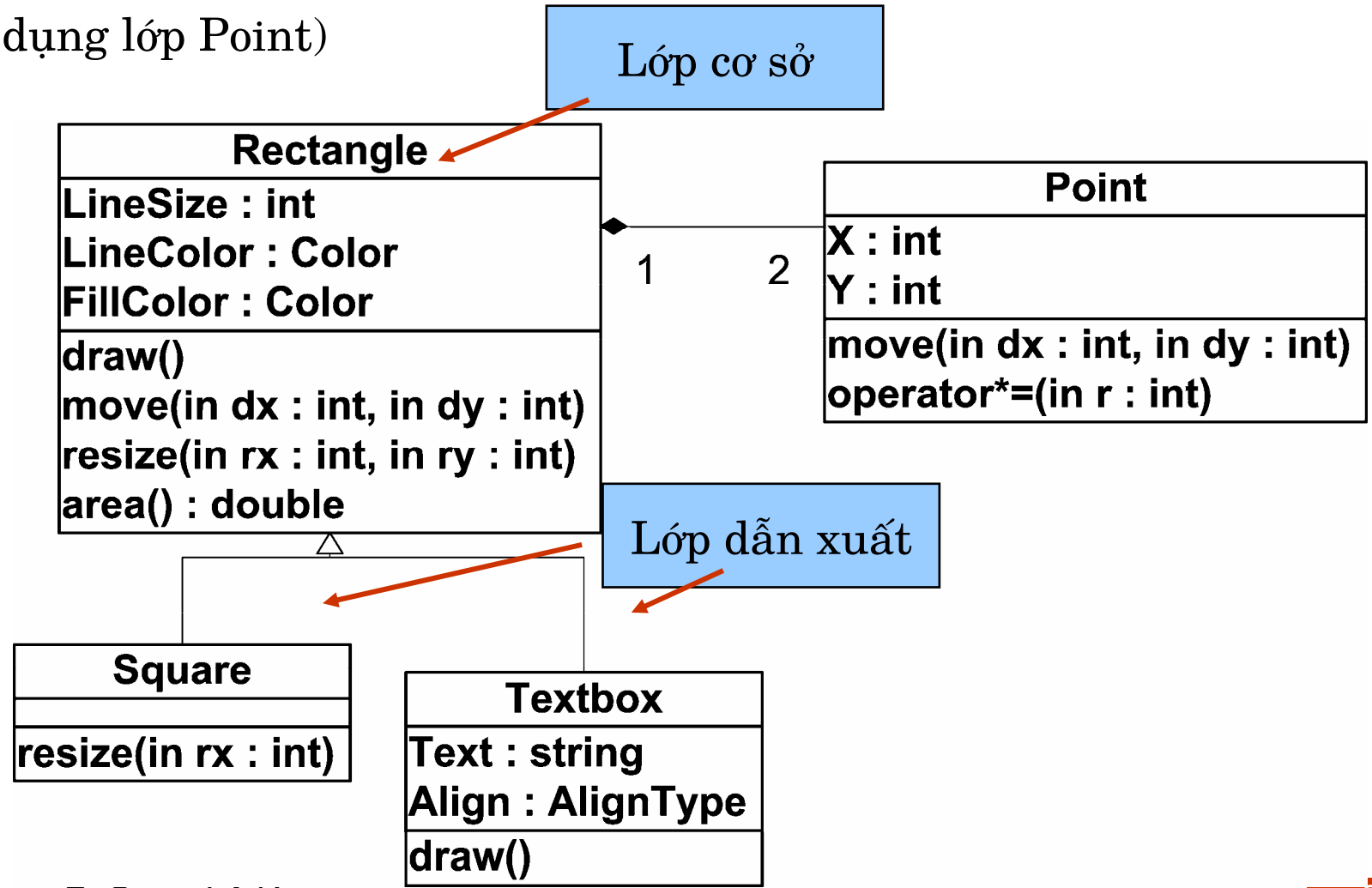


Các dạng quan hệ lớp (meta model)



7.2 Dẫn xuất và thừa kế

- Ví dụ xây dựng các lớp: Rectangle, Square và Textbox (sử dụng lớp Point)



Thực hiện trong C++: Lớp Point

```
class Point
{
   int X,Y;
public:
   Point() : X(0), Y(0) {}
   Point(int x, int y): X(x), Y(y) {}
   int x() const { return X; }
   int y() const { return Y; }
   void move(int dx, int dy) {
        X += dx;
        Y += dy;
   }
   void operator*=(int r) {
        X *= r;
        Y *= r;
   }
};
Point operator-(const Point& P1, const Point& P2) {
   return Point(P2.x()-P1.x(),P2.y()-P1.y());
}
```

Thực hiện trong C++: Lớp Rectangle

```
#include <iostream>
#include <string>
#include "Point.h"
typedef int Color;
class Rectangle
{
   Point TL, BR;
   Color LineColor, FillColor;
   int   LineSize;
public:
   Point getTL() const { return TL; }
   Point getBR() const { return BR; }
   void  setTL(const Point& tl) { TL = tl; }
   void  setBR(const Point& br) { BR = br; }
   Color getLineColor() const  { return LineColor; }
   void  setLineColor(Color c) { LineColor = c; }
   int   getLineSize() const   { return LineSize; }
   void  setLineSize(int s)    { LineSize = s; }
```

```
    Rectangle(int x1=0, int y1=0, int x2=10, int y2=10)
        : TL(x1,y1), BR(x2,y2), LineColor(256),FillColor(0) {}

    Rectangle(const Point& tl, const Point& br, Color lc, Color fc)
            : TL(tl), BR(br), LineColor(lc), FillColor(fc) {}
    void draw() {
       std::cout << "\nRectangle:\t[" << TL << BR << ']';
    }
    void move(int dx, int dy) {
       TL.move(dx,dy);
       BR.move(dx,dy);
       draw();
    }
    void resize(int rx, int ry) {
       TL *= rx;
       BR *= ry;
       draw();
    }
    double area() const {
       Point d = BR - TL;
       int a = d.x()*d.y();
       return a > 0 ? a : - a;
    }
};
```

Thực hiện trong C++: Lớp Square

```
#include "Rectangle.h"
class Square : public Rectangle
{
public:
   Square(int x1=1, int y1=0, int a=10)
        : Rectangle(x1,y1,x1+a,y1+a) {}

   void resize(int r) {
        Rectangle::resize(r,r);
   }
};
```

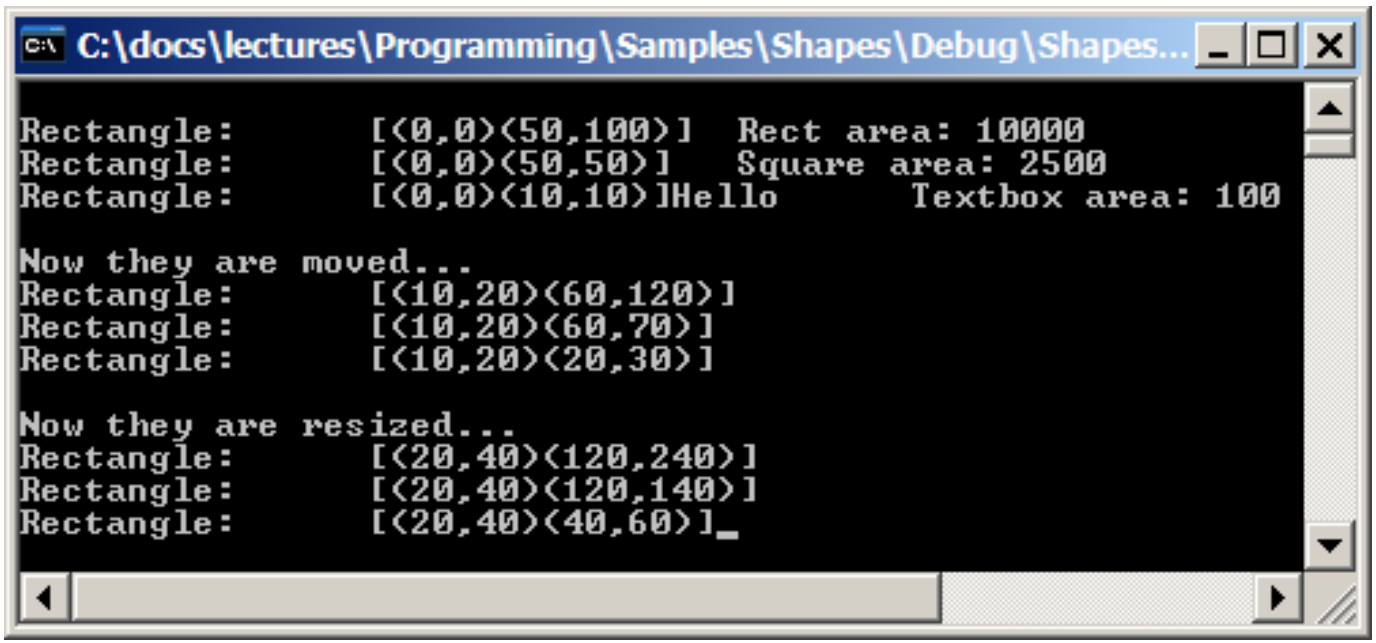
Thực hiện trong C++: Lớp Textbox

```
#include "Rectangle.h"
enum AlignType { Left, Right, Center};
class TextBox : public Rectangle
{
   std::string Text;
   AlignType   Align;
public:
   TextBox(const string& text = "Text")
        : Text(text), Align (Left) {}
   TextBox(const Point& tl, const Point& br, Color lc, Color fc,
           const string& text):
        Rectangle(tl,br,lc,fc), Text(text), Align(Left) {}

   void draw() {
        Rectangle::draw();
        std::cout << Text << '\n';
   }
};
```

Chương trình minh họa

```
#include "Rectangle.h"
#include  "Square.h"
#include "TextBox.h"
#include <conio.h>
void main()
{
   Rectangle rect(0,50,0,100);
   Square        square(0,0,50);
   TextBox       text("Hello");

   rect.draw();
   std::cout << "\t Rect area: " << rect.area();
   square.draw();
   std::cout << "\t Square area: " << square.area();
   text.draw();
   std::cout << "\t Textbox area: " << text.area();
```

```
    getch();
    std::cout << "\n\nNow they are moved...";
    rect.move(10,20);
    square.move(10,20);
    text.move(10,20);
    getch();
    std::cout << "\n\nNow they are resized...";
    rect.resize(2,2);
    square.resize(2);
    text.resize(2,2);
    getch();
}
```



Truy nhập thành viên

- Các hàm thành viên của lớp dẫn xuất có thể truy nhập thành viên "protected" định nghĩa ở lớp cơ sở, nhưng cũng không thể truy nhập các thành viên "private" định nghĩa ở lớp cơ sở

  Phản ví dụ:

```
Rectangle rect(0,0,50,100);
Square    square(0,0,50);
square.TL = 10;
```

- Lớp dẫn xuất được "thừa kế" cấu trúc dữ liệu và các phép toán đã được định nghĩa trong lớp cơ sở, nhưng không nhất thiết có quyền sử dụng trực tiếp, mà phải qua các phép toán (các hàm công cộng hoặc hàm public)
- Quyền truy nhập của các thành viên "public" và "protected" ở lớp dẫn xuất được giữ nguyên trong lớp cơ sở

Thuộc tính truy nhập kế thừa

Thuộc tính truy nhập của các thành viên lớp cơ sở X	Thuộc tính kế thừa của lớp dẫn xuất Y	
	class Y: private X	class Y: public X
private	Được kế thừa nhưng các thành viên của X không thể truy nhập trong Y	
protected	Các thành viên của X sẽ trở thành các thành viên **private** của Y và có thể được truy nhập trong Y	Các thành viên của X sẽ trở thành các thành viên **protected** của Y và có thể truy nhập trong Y
public	Thành viên của X sẽ trở thành thành viên **private** của Y và có thể truy nhập trong Y	Thành viên của X sẽ trở thành thành viên **public** của Y và có thể truy nhập trong Y

Ví dụ

```
void func2(int a, int b) {...}
int xy;
class X {
  private:
        int x1;
  protected:
        int x2;
  public:
        int x3;
        int xy;
        X(int a, int b, int c)
        {
                x1 = a;
                x2 = b;
                x3 = xy = c;
        }
        void func1(int, int);
        void func2(int, int);
};
void X::func1(int i, int j) {...}
void X::func2(int k, int l) {...}
```

```
class Y:public X {
  private:
        int y1;
  public:
        int y2;
        int xy;
        Y(int d, int e, int f, int g, int h):X(d, e, f)
        {
                y1 = g;
                y2 = xy = h;
        }
        void func2(int, int);
};
void Y::func2(int m, int n)
{
   int a, b;
   x1 = m;      //Error, x1 is private in the basic class X
   x2 = m;
   x3 = m;
   xy = m;
   X::xy = m;
   ::xy = m;
   y1 = n;
   y2 = n;
```

```
func1(a,b); OK, X::func1(...)
X::func2(a,b); OK, X::func2(...)
::func2(a,b)
}

void f()
{
   const int a = 12;
   Y objY(3, 4, 5, 6, 7);
   objY.x1 = a; //Error, x1 is private
   objY.x2 = a; //Error, x2 is protected
   objY.x3 = a;
   objY.xy = a;
   objY.y1 = a; //Error, y1 is private
   objY.y2 = a;
   objY.X::xy = a;
   objY.func1(a, a);
   objY.func2(a, a);
}
```

Chuyển đổi kiểu đối tượng

- Một đối tượng hay con trỏ, hoặc tham chiếu đối tượng kiểu lớp dẫn xuất sẽ có thể được chuyển đổi kiểu tự động về kiểu lớp cơ sở (nếu được kế thừa public) nhưng không đảm bảo theo chiều ngược.

Ví dụ:

```
class X { ... X(...){...} ... };
class Y:public X { ... Y(...):X(...){...} ... };
X objX(...);
Y objY(...);
X* xp = &objX;          //OK
X* xp = &objY;          //OK
Y* yp = &objX;          //Error
Y* yp = (Y*)&objX;      //OK, but not guaranteed!
```

- Chuyển đổi kiểu tự động cho đối tượng có kiểu lớp cơ sở sang kiểu lớp dẫn xuất sẽ không thể thực hiện vì không đảm bảo được quyền truy nhập của các thành viên của lớp cơ sở, chắc chắn không được nếu kế thừa private.

Chuyển đổi kiểu đối tượng

Ví dụ:

```
class X {
 public:
     int x;
};
class Y:private X {
};
void f()
{
     Y objY;
     X *xp;
     xp = &objY;              //Error
     xp = (X*) &objY;
     xp->x = 5;
}
```

7.3 Hàm ảo và cơ chế đa hình/đa xạ

- Trong quá trình liên kết, lời gọi các hàm và hàm thành viên thông thường được chuyển thành các lệnh nhảy tới địa chỉ cụ thể của mã thực hiện hàm => **"liên kết tĩnh"**
- Vấn đề thực tế:
  - Các đối tượng đa dạng, mặc dù giao diện giống nhau (phép toán giống nhau), nhưng cách thực hiện khác nhau => thực thi như thế nào?
  - Một chương trình ứng dụng chứa nhiều kiểu đối tượng (đối tượng thuộc các lớp khác nhau, có thể có cùng kiểu cơ sở) => quản lý các đối tượng như thế nào, trong một danh sách hay nhiều danh sách khác nhau?

Vấn đề của cơ chế "liên kết tĩnh"

- Xem lại chương trình trước, hàm Rectangle::draw đều in ra tên "Rectangle" => chưa hợp lý nên cần được định nghĩa lại ở các lớp dẫn xuất

```
void Square::draw() {
    std::cout << "\nSquare:\t[" << getTL() << getBR() << ']';
 }
void TextBox::draw() {
   std::cout << "\nTextbox:\t[" << getTL() << getBR() << ' '
             << Text << ']';
}
```

Chương trình minh họa 1

```
void main()
{
        Rectangle rect(0,0,50,100);
        Square    square(0,0,50);
        TextBox   text("Hello");

        rect.draw();
        square.draw();
        text.draw();

        getch(); std::cout << "\n\nNow they are moved...";
        rect.move(10,20);
        square.move(10,20);
        text.move(10,20);

        getch(); std::cout << "\n\nNow they are resized...";
        rect.resize(2,2);
        square.resize(2);
        text.resize(2,2);
        getch();
}
```

Kết quả: Như ý muốn?

```
Rectangle:      [(0,0)(50,100)]
Square:         [(0,0)(50,50)]
Textbox:        [(0,0)(10,10) Hello]

Now they are moved...
Rectangle:      [(10,20)(60,120)]
Rectangle:      [(10,20)(60,70)]
Rectangle:      [(10,20)(20,30)]

Now they are resized...
Rectangle:      [(20,40)(120,240)]
Rectangle:      [(20,40)(120,140)]
Rectangle:      [(20,40)(40,60)]
```

Gọi hàm draw() của Rectangle!

Chương trình minh họa 2

```
void main()
{
        const N =3;
        Rectangle rect(0,0,50,100);
        Square    square(0,0,50);
        TextBox   text("Hello");
        Rectangle* shapes[N] = {&rect, &square, &text};

        for (int i = 0; i < N; ++i)
                shapes[i]->draw();
        getch();
}
```

Quản lý các đối tượng chung trong một danh sách nhờ cơ chế dẫn xuất!

Kết quả: các hàm thành viên của lớp dẫn xuất cũng không được gọi

```
Rectangle:      [(0,0)(50,100)]
Rectangle:      [(0,0)(50,50)]
Rectangle:      [(0,0)(10,10)]
```

Giải pháp: Hàm ảo

```
class Rectangle {
...
public:
   ...
   virtual void draw();
}
```

Kết quả: Như mong muốn!

Chương trình 1

```
Rectangle:      [(0,0)(50,100)]
Square:         [(0,0)(50,50)]
Textbox:        [(0,0)(10,10) Hello]

Now they are moved...
Rectangle:      [(10,20)(60,120)]
Square:         [(10,20)(60,70)]
Textbox:        [(10,20)(20,30) Hello]

Now they are resized...
Rectangle:      [(20,40)(120,240)]
Square:         [(20,40)(120,140)]
Textbox:        [(20,40)(40,60) Hello]
```

Chương trình 2

```
Rectangle:      [(0,0)(50,100)]
Square:         [(0,0)(50,50)]
Textbox:        [(0,0)(10,10) Hello]
```

Hàm ảo

```
class X {
...
public:
   virtual void f1() {...}
   virtual void f2() {...}
   virtual void f3() {...}
   void f4() {...}
};
```

```
class Y:public X {
...
public:
   void f1() {...}
   void f2(int a) {...}
   char f3() {...}
   void f4() {...}
};
```

```
void function()
{
   Y y;
   X* px = &y; //Typ-Convert Y* to X*
   px->f1();   //virtual function Y::f1()
   px->f2();   //virtual function X::f2()
   px->f3();   //virtual function X::f3()
   px->f4();   //function X::f4()
}
```

Ví dụ hàm ảo

```
class X {
  protected:
        int x;
  public:
        X(int x_init) { x = x_init;}
        virtual void print();
};
class Y:public X {
  protected:
        int y;
  public:
        Y(int x_init, int y_init):X(x_init) {y = y_init;}
        void print();
};
class Z:public Y {
  protected:
        int z;
  public:
        Z(int x_init, int y_init, int z_init):Y(x_init, y_init)
        {z = z_init;}
  void print();
};
```

```
class U:public Y {
  protected:
        int u;
  public:
        Z(int x_init, int y_init, int u_init):Y(x_init, y_init)
        {u = u_init;}
  void print();
};
void X::print()
{
   cout << "Data of Class X: " << x << endl;
}
void Y::print()
{
   cout << "Data of Class X+Y: " << x + y << endl;
}
void Z::print()
{
   cout << "Data of Class X+Y+Z: " << x + y + z << endl;
}
void U::print()
{
   cout << "Data of Class X+Y+U: " << x + y + u << endl;
}
```

```
void print_data(X* px)
{
   px->print();
}
main()
{
   X* pobjX = new X(1);
   Y* pobjY = new Y(10, 20);
   Z* pobjZ = new Z(100, 200, 300);
   U* pobjU = new U(1000, 2000, 3000);
   print_data(pobjX);
   print_data(pobjY);
   print_data(pobjZ);
   print_data(pobjU);
   delete pobjX;
   delete pobjY;
   delete pobjZ;
   delete pobjU;
}
```

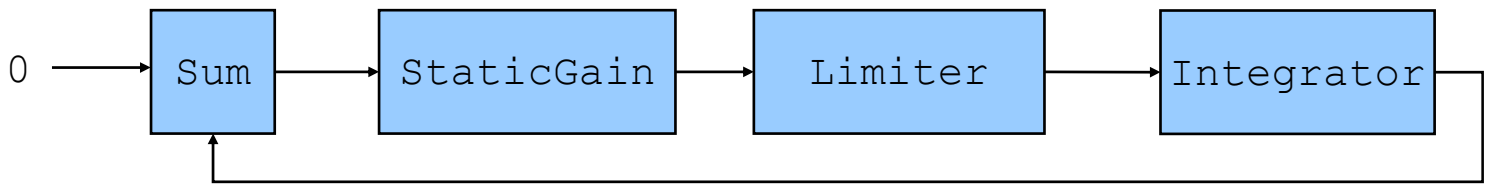
```
main()
{
int x;
X *pobj[4];
pobj[0] = new X(1);
pobj[1] = new Y(10, 20);
pobj[2] = new Z(100, 200, 300);
pobj[3] = new U(1000, 2000,
   3000);
for(x = 0; x < 4; x++)
print_data(pobj[x]);
delete[4] pobj;
}
```

Kết quả:

```
Data of Class X: 1
Data of Class X+Y: 30
Data of Class X+Y+Z: 600
Data of Class X+Y+U: 6000
```

7.4 Ví dụ thư viện khối chức năng

- Bài toán:
  - Xây dựng một thư viện các khối chức năng phục vụ tính toán và mô phỏng tương tự trong SIMULINK
  - Viết chương trình minh họa sử dụng đơn giản
- Ví dụ một sơ đồ khối



Biểu đồ lớp