KẾ THỪA & ĐA HÌNH

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

GVHD: Trương Toàn Thịnh

NỘI DUNG

- Giới thiệu
- Kế thừa
- Đa hình
- Cơ chế truy xuất
- Hàm & lớp bạn
- Phương thức hủy ảo
- Một số kỹ thuật
  - Xác định tên lớp
  - Tạo đối tượng dùng bảng mẫu
  - Tạo đối tượng dùng chuỗi chứa tên lớp
  - Tổng quát hóa qui trình & thuật toán

GIỚI THIỆU

- Quan hệ kế thừa (tổng quát hóa và đặc biệt hóa) rất tự nhiên
- Ví dụ: Rectangle là đặc biệt hóa của parallelogram, và là tổng quát hóa của Square
- Có thể phát biểu:
  - Square kế thừa (là) Rectangle
  - Square là lớp kế thừa, Rectangle là lớp cơ sở
- Một số lợi ích:
  - Lớp kế thừa dùng lại đoạn mã trong lớp cơ sở
  - Lớp kế thừa thuộc tính & phương thức riêng
  - Đoạn mã thực thi trên lớp cơ sở cũng thực thi trên lớp kế thừa

NỘI DUNG

- Giới thiệu
- Kế thừa
- Đa hình
- Cơ chế truy xuất
- Hàm & lớp bạn
- Phương thức hủy ảo
- Một số kỹ thuật
  - Xác định tên lớp
  - Tạo đối tượng dùng bảng mẫu
  - Tạo đối tượng dùng chuỗi chứa tên lớp
  - Tổng quát hóa qui trình & thuật toán

KẾ THỪA

- Khai báo

```
#ifndef _Rectangle_h_
#define _Rectangle_h_
#include <iostream>
using namespace std;
class Rectangle{
  protected:
    float width, height;
  public:
    Rectangle(float, float);
    float Area();
    void Input(istream&);
};
#endif
```

```
#ifndef _Square_h_
#define _Square_h_
#include "Rectangle.h"

class Square : public Rectangle{
  public:
    Square(float a = 0);
    void Input(istream&);
};
#endif
```

  - Square kế thừa width & height của Rectangle
  - Square nạp chồng lại Input (override)

KẾ THỪA

- Định nghĩa

```
#include "Rectangle.h"
Rectangle::Rectangle(float w, float h){
  width = w; height = h;
}
void Rectangle::Input(istream& is){
    float is>>width>>height;
}
float Rectangle::Area(){
  return width*height;
}
```

```
#include "Square.h"
Square::Square(float a ){
  width = height = a;
}
void Square::Input(istream& is){
  is>>width; height = width;
}
```

```
#include "Rectangle.h"
#include "Square.h"
void main(){
  Rectangle Rec;
  Square Sq;
  Rec.Input(cin);
  cout<<Rec.Area()<<endl;
  Sq.Input(cin);
  cout<<Sq.Area()<<endl;
}
```

KẾ THỪA

- Biến con trỏ đối tượng của lớp cha có thể giữ địa chỉ của biến đối tượng thuộc lớp con
- Ví dụ:

```
Rectangle* pRec;
Square Sq(10);
pRec = &Sq;
pRec = new Square(5);
```

- C++ vẫn có thể tắt cơ chế này.

KẾ THỪA

- Không kế thừa chỉ để tận dụng lại thuộc tính.
  - Ví dụ: xét thấy DUONGTRON có tâm thuộc DIEM nên cho DUONGTRON kế thừa DIEM
  - Ví dụ: xét thấy RECTANGLE có hai thuộc tính width & height, trong khi SQUARE có một thuộc tính nên cho RECTANGLE kế thừa SQUARE và bổ sung thêm một thuộc tính.

NỘI DUNG

- Giới thiệu
- Kế thừa
- Đa hình
- Cơ chế truy xuất
- Hàm & lớp bạn
- Phương thức hủy ảo
- Một số kỹ thuật
  - Xác định tên lớp
  - Tạo đối tượng dùng bảng mẫu
  - Tạo đối tượng dùng chuỗi chứa tên lớp
  - Tổng quát hóa qui trình & thuật toán

ĐA HÌNH

- Gia tăng tính tái sử dụng cho đoạn mã
- Các đoạn mã được viết tổng quát & tùy chỉnh lại cho một vài trường hợp đặc biệt
- Ví dụ

```
void main(){
  Rectangle* pRec;
  Rectangle Rec;
  Square Sq;
  pRec = &Rec;
  pRec->Input(cin);
  cout<<Rec.Area()<<endl;
  pRec = &Sq;
  pRec->Input(cin);
  cout<<Sq.Area()<<endl;
}
```

Input của Rectangle được gọi → không cơ chế đa hình

Input của Square được gọi → có cơ chế đa hình

ĐA HÌNH

- Cần hiệu chỉnh lại lớp Rectangle để có cơ chế đa hình (Thêm virtual vào Input)

```
#ifndef _Rectangle_h_
#define _Rectangle_h_
#include <iostream>
using namespace std;
class Rectangle{
   //…
   virtual void Input(istream&);
};
#endif
```

→ **Phương thức đa hình**

- Phương thức Input trong Square tự trở thành đa hình (không cần thêm virtual trong Square)

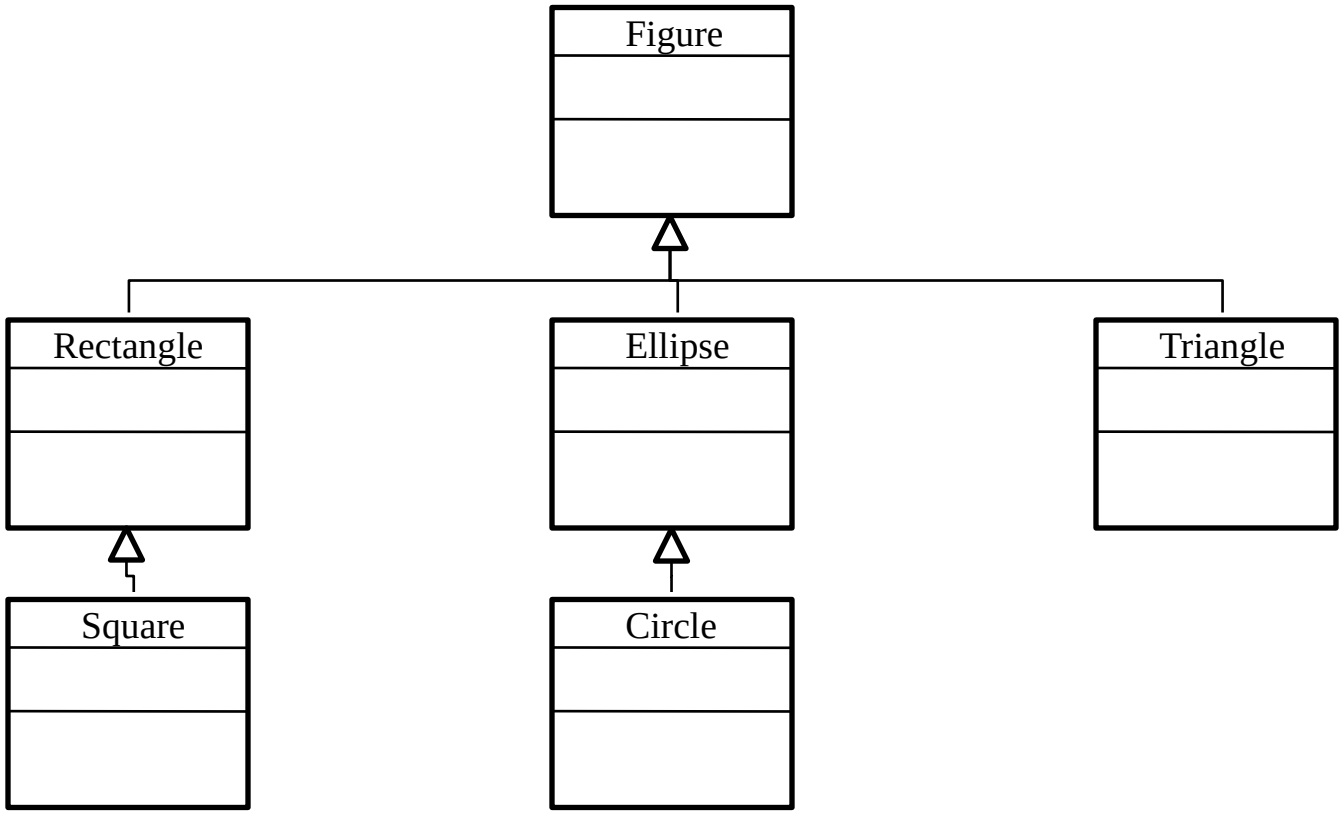
ĐA HÌNH

- Một số lớp không có ý nghĩa khi tạo đối tượng → Lớp trừu tượng
- Lớp có chứa ít nhất một phương thức thuần ảo thì đó gọi là lớp trừu tượng
- Phương thức thuần ảo là phương thức chỉ có khai báo mà không có cài đặt.

```
#ifndef _Figure_h_
#define _Figure_h_
#include <iostream>
using namespace std;
class Figure{
  public:
    virtual void Input(istream&) = 0;
    virtual float Area() = 0;
};
#endif
```

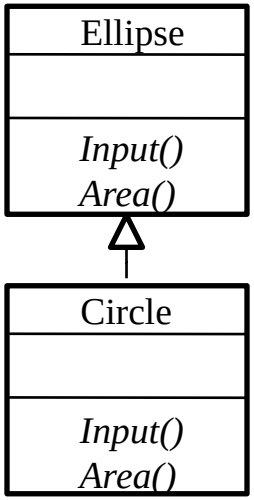
ĐA HÌNH

- Ví dụ: xây dựng hệ thống hình theo cây kế thừa sau



ĐA HÌNH

- Ví dụ: Lưu ý các lớp hình đều có hai phương thức ảo là Input và Area



ĐA HÌNH

- Ví dụ: ta đã có hai lớp Rectangle, Square và lớp trừu tượng Figure.
- Bổ sung khai báo lớp Rectangle

```
#ifndef _Rectangle_h_
#define _Rectangle_h_
#include "Figure.h"

class Rectangle : Figure{
  protected:
    float width, height;
  public:
    Rectangle(float, float);
    virtual float Area();
    virtual void Input(istream&);
};
#endif
```

ĐA HÌNH

- Ví dụ: cài đặt lớp Ellipse.

```
#ifndef _Ellipse_h_
#define _Ellipse_h_
#define PI 3.14159F
#include “Figure.h”
class Ellipse : public Figure{
  protected:
    float Ra, Rb;
  public:
    Ellipse(float a = 0, float b = 0);
    virtual float Area();
    virtual void Input(istream&);
};
#endif
```

```
#include “Ellipse.h”
Ellipse::Ellipse(float a, float b){
  Ra = a; Rb = b;
}
float Ellipse::Area(){
  return PI * Ra * Rb;
}
void Ellipse::Input(istream& is){
  is>>Ra>>Rb;
}
```

ĐA HÌNH

- Ví dụ: cài đặt lớp Circle.

```
#ifndef _Circle_h_
#define _Circle_h_
#include “Ellipse.h”
class Circle : public Ellipse{
  public:
    Circle(float r = 0);
    virtual void Input(istream&);
};
#endif
```

```
#include “Circle.h”
Circle::Circle(float r){
  Ra = Rb = r;
}
void Circle::Input(istream& is){
  is>>Ra;
  Rb = Ra;
}
```

ĐA HÌNH

- Ví dụ: cài đặt lớp Triangle.

```
#ifndef _Triangle_h_
#define _Triangle_h_
#include “Figure.h”
class Triangle : public Figure{
  protected:
    float baseSide, height;
  public:
    Triangle(float a = 0, float h = 0);
    virtual float Area();
    virtual void Input(istream&);
};
#endif
```

```
#include “Triangle.h”
Triangle::Triangle (float a, float h){
  baseSide = a; height = h;
}
float Triangle::Area(){
  return (baseSide * height) / 2;
}
void Triangle::Input(istream& is){
  is>>baseSide>>height;
}
```

ĐA HÌNH

- Ví dụ: Giả sử cần tìm hình có diện tích lớn nhất trong mảng các hình có sẵn

```
#include “Figure.h”
#include “Rectangle.h”
#include “Square.h”
#include “Ellipse.h”
#include “Circle.h”
#include “Triangle.h”
void main(){
  Figure* f[] = {new Rectangle(9.3, 9.7), new Circle(4.5),
                 new Ellipse(4.2, 4.7), new Square(9.5),
                 new Triangle(10, 6.4), new Ellipse(3.7, 7.8)};
  int n = sizeof(f)/sizeof(Figure*);
  Figure* fig = findMaxArea(f, n);
  if(!fig) cout << fig->Area() << endl;
};
```

ĐA HÌNH

- Ví dụ: Cài đặt hàm findMaxArea()

```
Figure* findMaxArea(Figure* f[], int n){
  Figure* a = NULL;
  if(n > 0){
    a = f[0];
    for(int i = 0; i < n; i++){
      if(f[i]->Area() > a->Area()){
        a = f[i];
      }
    }
  }
  return a;
}
```

NỘI DUNG

- Giới thiệu
- Kế thừa
- Đa hình
- Cơ chế truy xuất
- Hàm & lớp bạn
- Phương thức hủy ảo
- Một số kỹ thuật
  - Xác định tên lớp
  - Tạo đối tượng dùng bảng mẫu
  - Tạo đối tượng dùng chuỗi chứa tên lớp
  - Tổng quát hóa qui trình & thuật toán

CƠ CHẾ TRUY XUẤT

- Mỗi thuộc tính hay phương thức có một trong ba loại phạm vi (tầm vực):
  - public: Có thể truy xuất tùy ý và được truyền xuống cho các lớp con
  - protected: Chỉ truy xuất từ bên trong lớp đó và cũng được truyền xuống cho lớp con
  - private: Chỉ truy xuất từ bên trong lớp đó và KHÔNG được truyền xuống cho lớp con

CƠ CHẾ TRUY XUẤT

- Ví dụ:

```
class A{
  private:
    float a1, a2;
  protected:
    float b;
  public:
    float c;
    A(float);
};
A::A(float v){
  a1 = c = v;
}
```

```
class B : public A{
  public:
    int getb();
    void copyb(A& obj);
    int geta2();
};
int B::getb(){return b;}
void B::copyb(A& obj){
  c = obj.c;
  b = obj.b;//Sai
}
int B::geta2(){
  return a2;//Sai
}
```

```
void main(){
  A obj1(8); B obj2(11);
  obj1.a1 = 2;//Sai
  obj1.c = 7;
}
```

CƠ CHẾ TRUY XUẤT

- Bảng truy xuất
  - Kế thừa public → tầm vực theo lớp cha
  - Kế thừa private hay protected → tầm vực theo lớp con

Lớp cha \ Lớp con	**kế thừa private**	**kế thừa protected**	**kế thừa public**
Khai báo protected	Dùng trong phạm vi lớp cha và trở thành **private** trong lớp con	Dùng trong phạm vi lớp cha và trở thành **protected** trong lớp con	Dùng trong phạm vi lớp cha và trở thành **protected** trong lớp con
Khai báo public	Có thể truy xuất từ bên ngoài và trở thành **private** trong lớp con	Có thể truy xuất từ bên ngoài và trở thành **protected** trong lớp con	Có thể truy xuất từ bên ngoài và trở thành **public** trong lớp con
Khai báo private	Dùng trong phạm vi lớp cha và không truyền xuống lớp con		

HÀM BẠN & LỚP BẠN

- C++ hỗ trợ từ khóa friend cho phép truy xuất vào các trường private hay protected
- Có hai trường hợp
  - Hàm bạn: là bạn của một lớp nào đó và có thể truy xuất tất cả các trường của lớp đó bất chấp tầm vực
  - Lớp bạn: là bạn của một lớp nào đó và có thể truy xuất tất cả các trường của lớp đó bất chấp tầm vực

HÀM BẠN & LỚP BẠN

- Ví dụ:

```
class Point{
  int x, y;
  public:
    Point(int x0 = 0, int y0 = 0){
      x = x0; y = y0;
    }
    friend class RECT;
    friend void PointMove(Point&, int, int);
};
```

```
class RECT{
  Point P, Q;
  public:
    Point Center(){
      Point M;
      M.x = (P.x + Q.x)/2;
      M.y = (P.y + Q.y)/2;
      return M;
    }};
```

```
void PointMove(Point& P, int dx, int dy){
  P.x+=dx; P.y+=dy;//Đúng
}
void ABC(Point& P, int dx, int dy){
  P.x+=dx; P.y+=dy;//Sai
}
```

RECT là lớp bạn của lớp **Point**
PointMove là hàm bạn của lớp **Point**

PHƯƠNG THỨC HỦY ẢO

- Phương thức hủy vẫn có thể khai báo đa hình
- Phương thức hủy của lớp con gọi trước sau đó là phương thức hủy của lớp cha

```
Rectangle* r = new Square();
r->Input(cin);
//…
delete r;
```

- Câu lệnh ‘delete r’ sẽ:
  - Kích hoạt hàm hủy của Rectangle nếu hàm hủy của Rectangle **không có** từ virtual
  - Kích hoạt hàm hủy của Square nếu hàm hủy của Rectangle **có** từ virtual

PHƯƠNG THỨC HỦY ẢO

- Ví dụ thứ tự hàm tạo & hủy trong kế thừa

```
class Point{
  int x, y;
  public:
    Point(int x0 = 0, int y0 = 0){x = x0; y = y0;}
    friend class RECT;
};
class Polygon {
protected:
  int n; Point* arr;
  public:
    Polygon(int size = 0) {
      n = 0; arr = NULL;
      if (size > 0) {
        n = size; arr = new Point[n];
      }
    }
    virtual ~Polygon() {
      if (arr != NULL) {
        delete[]arr; arr = NULL; n = 0;}}};
```

```
class RECT : public Polygon {
char* buffer;
public:
  RECT(Point A, Point B) :Polygon(4) {
    arr[0] = A; arr[1] = { A.y, B.x };
    arr[2] = B;arr[3] = { A.x, B.y };
    int s = labs((A.x - B.x)*(A.y - B.y));
    buffer = new char[s];
  }
  virtual ~RECT() {
    if (buffer != NULL) {
      delete[]buffer; buffer = NULL;
    }}};
```

```
void main() {
  Polygon* p = new RECT({4,5},{1,1});
  delete p;
};
```

Poly → RECT → ~RECT → ~Poly

NỘI DUNG

- Giới thiệu
- Kế thừa
- Đa hình
- Cơ chế truy xuất
- Hàm & lớp bạn
- Phương thức hủy ảo
- Một số kỹ thuật
  - Xác định tên lớp
  - Tạo đối tượng dùng bảng mẫu
  - Tạo đối tượng dùng chuỗi chứa tên lớp
  - Tổng quát hóa qui trình & thuật toán

XÁC ĐỊNH TÊN LỚP

- Xét ví dụ về các lớp hình học:
  - Nhận thấy biến con trỏ kiểu Figure có thể giữ địa chỉ các biến kiểu Rectangle, Square…
  - Có hai cách cơ bản để nhận biết đang giữ địa chỉ của lớp nào
    - Dùng đa hình
    - Dùng kế thừa
  - Đa hình: bổ sung phương thức virtual trả ra tên lớp
  - Kế thừa: bổ sung thuộc tính tên lớp vào Figure

XÁC ĐỊNH TÊN LỚP

- Dùng đa hình

```
#ifndef  _Figure_h_
#define _Figure_h_
#include <iostream>
using namespace std;
class Figure{
  public:
    //…
    virtual char* className() = 0;
};
#endif
```

```
#ifndef  _Ellipse_h_
#define _Ellipse_h_
#define PI 3.14159
#include “Figure.h”
class Ellipse : public Figure{
   //…
  public:
    //…
    virtual char* className(){
      return “Ellipse”;
    }
};
#endif
```

```
#ifndef  _Circle_h_
#define _Cirle_h_
#include “Ellipse.h”
class Circle : public Ellipse{
   //…
  public:
    //…
    virtual char* className(){
      return “Cirle”;
    }
};
#endif
```

```
void main(){
  Figure* a[] = {new Rectangle(9, 8), new Circle(4.5F), new Ellipse(4.2F, 4.7F)};
  int n = sizeof(a)/sizeof(Figure*);
  for(int i = 0; i < n; i++)cout << a[i]->className() << endl;
}
```

XÁC ĐỊNH TÊN LỚP

- Dùng kế thừa

```
#ifndef  _Figure_h_
#define _Figure_h_
#include <iostream>
using namespace std;
class Figure{
  protected:
    char* name;
  public:
    //…
    char* getClass(){return name;}
};
#endif
```

```
#include “Ellipse.h”
Ellipse::Ellipse(float a, float b){
  Ra = a; Rb = b;
  name = “Ellipse”;
}
#endif
```

```
#include “Circle.h”
Cirle::Cirle(float r){
  Ra = Rb = r;
  name = “Cirle”;
}
#endif
```

```
void main(){
  Figure* a[] = {new Rectangle(9, 8), new Circle(4.5F), new Ellipse(4.2F, 4.7F)};
  int n = sizeof(a)/sizeof(Figure*);
  for(int i = 0; i < n; i++)cout << a[i]->getClass() << endl;
}
```

TẠO ĐỐI TƯỢNG NHỜ BẢNG MẪU

- Tình huống: cần tạo ra một bản sao của đối tượng do con trỏ kiểu Figure đang giữ địa chỉ.
- Vấn đề: Ta không biết con trỏ kiểu Figure đang giữ địa chỉ thuộc kiểu nào (Rectangle, Square hay Ellipse…)
- Giải pháp: xây dựng hàm thuần ảo Clone trong lớp Figure

TẠO ĐỐI TƯỢNG NHỜ BẢNG MẪU

- Ví dụ:

```
#ifndef  _Figure_h_
#define _Figure_h_
#include <iostream>
using namespace std;
class Figure{
  public:
    //…
    virtual Figure* Clone() = 0;
};
#endif
```

```
#ifndef  _Ellipse_h_
#define _Ellipse_h_
#define PI 3.14159
#include “Figure.h”
class Ellipse : public Figure{
   //…
  public:
    //…
    virtual Figure* Clone(){
      return new Ellipse(*this);
    }
};
#endif
```

```
#ifndef  _Circle_h_
#define _Cirle_h_
#include “Ellipse.h”
class Circle : public Ellipse{
   //…
  public:
    //…
    virtual Figure* Clone(){
      return new Circle(*this);
    }
};
#endif
```

```
void main(){
  Figure* a[] = {new Rectangle(9, 8), new Circle(4.5F), new Ellipse(4.2F, 4.7F)};
  int n = sizeof(a)/sizeof(Figure*);
  for(int i = 0; i < n; i++)cout << a[i]->Clone()->getClass() << endl;
}
```

TẠO ĐỐI TƯỢNG DÙNG CHUỖI CHỨA TÊN LỚP

- Xét ví dụ Figure: Trong tình huống cần tạo một hàm

  Figure* createObject(char* clsName),
- Trong đó tham số clsName là chuỗi tên lớp cần tạo.
- Giải pháp:
  - Xây dựng sẵn một ‘toolbar’ chứa tất cả các hình có thể chứa
  - Gọi hàm createObject với tên clsName cần tạo.

TẠO ĐỐI TƯỢNG DÙNG CHUỖI CHỨA TÊN LỚP

- Ví dụ

```
#ifndef  _Figure_h_
#define _Figure_h_
#include <iostream>
#include <vector>
using namespace std;
class Figure{
  private:
    static vector<Figure*> arr;
  protected:
    static void add(Figure*);
  public:
    static Figure* createObject(char*);
    virtual char* getClass() = 0;
    virtual Figure* Clone() = 0;
};
#endif
```

Mảng chứa các hình mẫu

Hàm thêm một hình mẫu vào mảng

```
#include "Figure.h"
vector<Figure*> Figure::arr;
void Figure::add(Figure* f){
  if(f==NULL) return;
  int i = arr.size();
  while(--i>=0){
    Figure* o = arr[i];
    if(strcmp(f->getClass(), o->getClass()) == 0)
      break;
  }
  if(i < 0) arr.push_back(f);
}
```

TẠO ĐỐI TƯỢNG DÙNG CHUỖI CHỨA TÊN LỚP

- Ví dụ

```
#ifndef  _Figure_h_
#define _Figure_h_
#include <iostream>
#include <vector>
using namespace std;
class Figure{
  private:
    static vector<Figure*> arr;
  protected:
    static void add(Figure*);
  public:
    static Figure* createObject(char*);
    virtual char* getClass() = 0;
    virtual Figure* Clone() = 0;
};
#endif
```

Hàm tạo một bản sao của hình mẫu

```
#include "Figure.h"
Figure* Figure::createObject(char* clsName){
  if(clsName == NULL) return NULL;
  int i = arr.size();
  while(--i>=0){
    Figure* o = arr[i];
    if(strcmp(clsName, o->getClass()) == 0)
      break;
  }
  if(i >= 0) arr[i]->Clone();
  else return NULL;
}
```

TẠO ĐỐI TƯỢNG DÙNG CHUỖI CHỨA TÊN LỚP

- Ví dụ

```
#include "Ellipse.h"
Ellipse::Ellipse(float a, float b){
  Ra = a; Rb = b;
  add(this);
}
```

```
#include "Circle.h"
Circle::Circle(float r){
  Ra = Rb = r;
  add(this);
}
```

```
#include "Rectangle.h"
Rectangle::Rectangle(float w,
float h){
  width = w; height = h;
  add(this);
}
```

```
#include "Square.h"
Square::Square(float
a){
  width = height = a;
  add(this);
}
```

```
void main(){
  Figure* a[] = {new Rectangle(9, 8), new Circle(4.5F), new Ellipse(4.2F, 4.7F)};
  Figure* rec = Figure::createObject("Rectangle");
  cout << rec->getClass() << endl;
}
```

TỔNG QUÁT HÓA QUY TRÌNH & THUẬT TOÁN

- Nhìn chung, để thực hiện một chương trình nào đó cần qua các bước sau:
  - *Thông báo mở đầu* & *nhập liệu*
  - *Kiểm tra dữ liệu* nhập
  - Nếu dữ liệu không hợp lệ thì
    - *Thông báo lỗi* & yêu cầu nhập lại
  - *Xử lý yêu cầu*
  - *Xuất kết quả*
  - *Hỏi tiếp tục hay không*
  - Nếu tiếp tục quay lại từ đầu
  - Dừng chương trình nếu không muốn tiếp tục

TỔNG QUÁT HÓA QUY TRÌNH & THUẬT TOÁN

- Cài đặt lớp ProgramFrame

```
#include <iostream>
using namespace std;
class ProgramFrame{
  protected:
    virtual void startMessage(ostream&);
    virtual void input(istream&) = 0;
    virtual void check() = 0;
    virtual void errorMessage(ostream&);
    virtual void output(ostream&) = 0;
    virtual void process() = 0;
    virtual bool continue(istream&, ostream&);
  public:
    void run(istream&, ostream&);
};
#endif
```

```
#include "ProgramFrame.h"
void ProgramFrame::startMessage(ostream
&os){
  os<<"Welcome, entering data: ";
}
void ProgramFrame::errorMessage(ostream
&os){
  os<<"Input data error!"<<endl;
}
bool askContinue(istream& is, ostream& os){
  os<<"Press y to continue, others to stop: ";
  char ch;  is>>ch;
  return (ch=='Y' || ch == 'y');
}
```

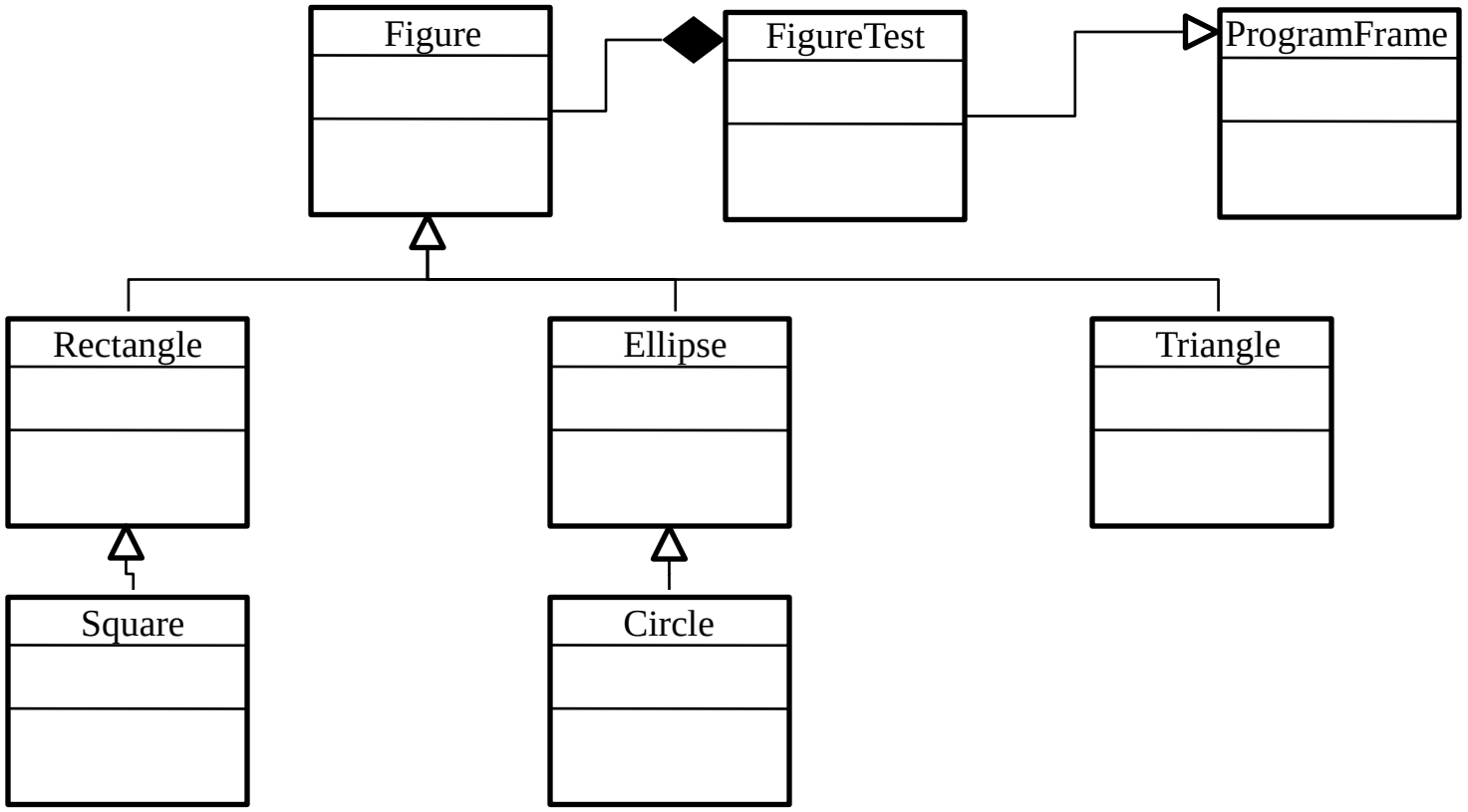
TỔNG QUÁT HÓA QUY TRÌNH & THUẬT TOÁN

- Cài đặt phương thức khuôn ‘run’

```
void ProgramFrame::run(istream &is, ostream &os){
  bool b;
  do{
    startMessage(os);
    input(is);
    if(!check()){
      errorMessage(os);
      continue;
    }
    process();
    output(os);
    b = continue(is, os);
  }while(b);
}
```

TỔNG QUÁT HÓA QUY TRÌNH & THUẬT TOÁN

- Ứng dụng kiểm thử lớp Figure



TỔNG QUÁT HÓA QUY TRÌNH & THUẬT TOÁN

- Kiểm lớp nào → xây dựng lớp Test tương ứng (Kế thừa & override)

```
#include "Figure.h"
class FigureTest: public ProgramFrame{
  Figure* f;
  public:
    FigureTest(Figure* p){f = p;}
    virtual void input(istream &is){
      if(!f) f->Input(is);
    }
    virtual void check(){
      if(!f) return false;
      return f->isValid();
    }
    virtual void output(ostream &os){
      if(!f) return;
      os<<f->getClass() << ", Area: "<<f->Area() << endl;
    }
    virtual void process() {}
};
```

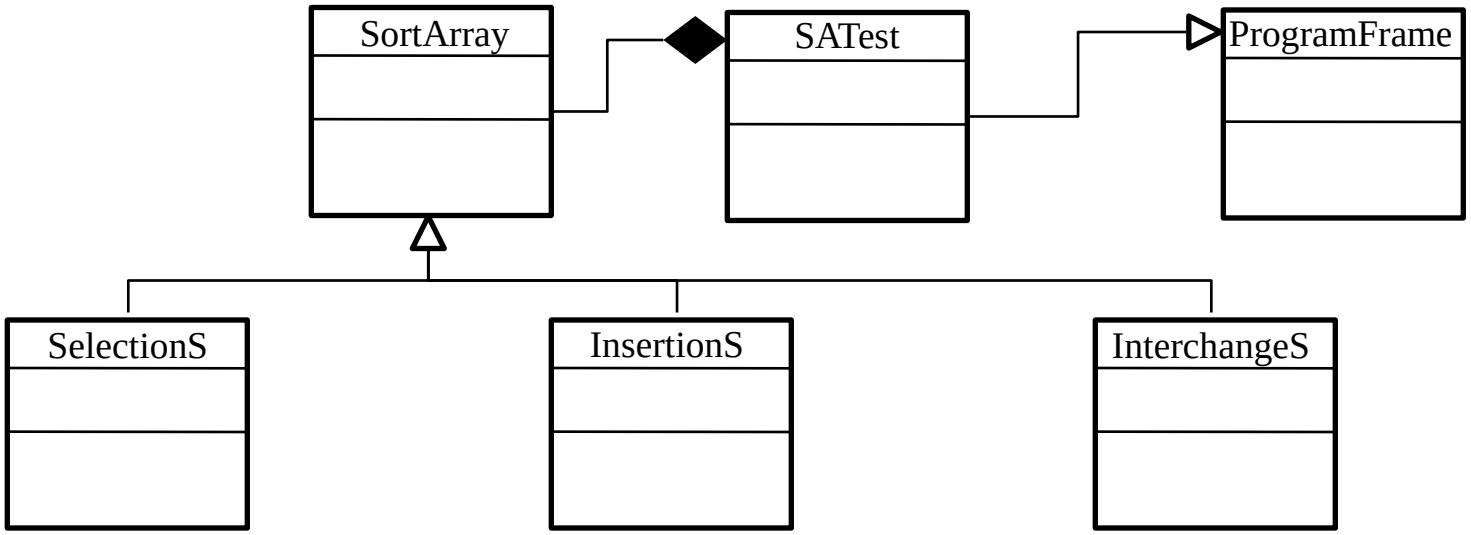
TỔNG QUÁT HÓA QUY TRÌNH & THUẬT TOÁN

- Sử dụng FigureTest trong hàm main

```
#include <iostream>
using namespace std;
void main(){
  FigureTest ft;
  ft.run(cin, cout);
}
```

BÀI TẬP

- Bài tập 1: áp dụng mẫu phương thức khuôn, xây dựng chương trình kiểm tra các thuật toán sắp xếp trên mảng số thực



BÀI TẬP

- Bài tập 2: Cho biết những gì sẽ xuất hiện trên màn hình trong các trường hợp sau
  - [yyy] & [zzz] trống
  - [yyy] trống & [zzz] virtual
  - [yyy] virtual & [zzz] trống
  - [yyy] virtual & [zzz] virtual

BÀI TẬP

- Bài tập 2:

```
class A{
  public:
    [yyy] void f1() { cout << “Good morning.\n”; f2(); }
    [zzz] void f2() { cout << “Good afternoon.\n”; }
};
class B: public A{
  public:
    void f1() { cout << “Good evening.\n”; f2(); }
    void f2() { cout << “Good night.\n”; }
};
void main(){
  A* pObj = new B;
  pObj->f1();
}
```