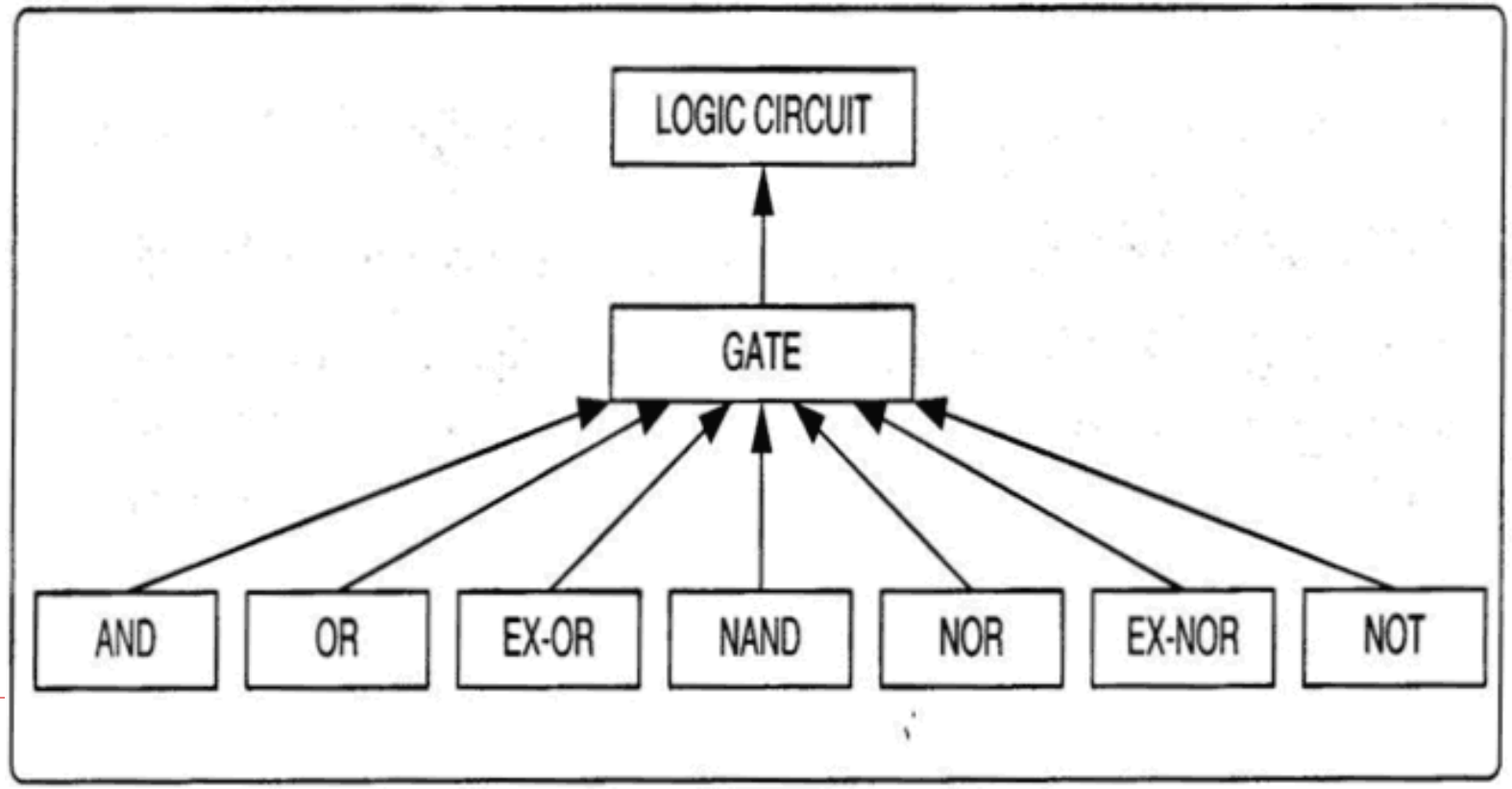
Composition

Jiun-Long Huang
National Chiao Tung University

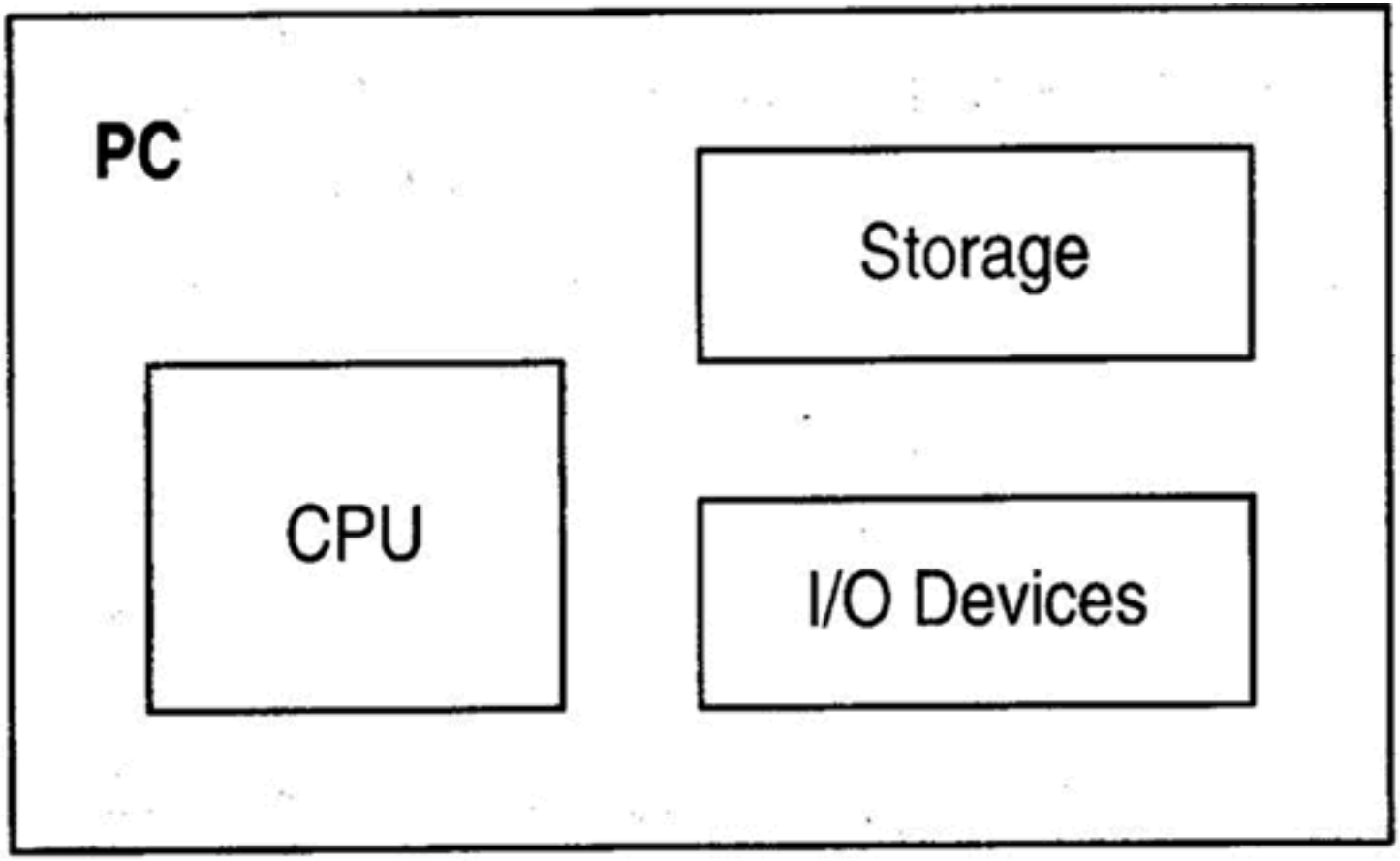
Inheritance

- Inheritance
  - is-a relationship



- Composition
  - has-a relationship.
  - A CPU is not a type of PC
    - Inheritance X
  - A PC has a CPU as well as many other components.
    - Composition O
- A new class can be composed from many existing classes

Has-A Relationship



Class Declaration

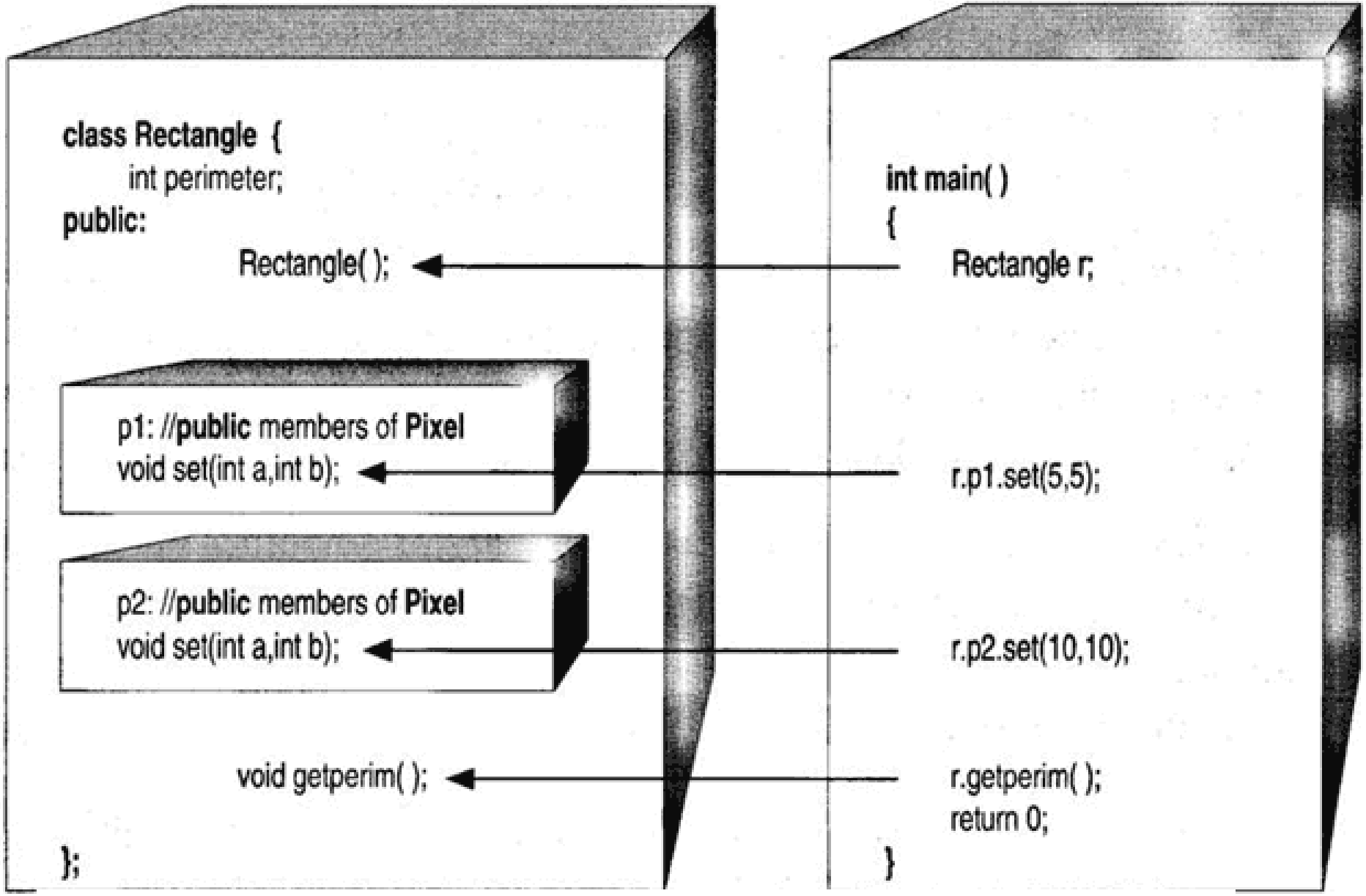
- To compose a class from existing classes, an object of each class should be declared as a member of the new class.
- Storage class, which is composed of the three classes Hard_Disk, RAM, and Floppy

```
Class Storage      //composed class
{
  Hard_Disk hd;    //an object of the Hard_Disk class
  RAM ram;         //an object of the RAM class
  Floppy fp;       //an object of the Floppy class
  ...
}
```

```
#include <iostream>
using namespace std;
class Pixel {
  int x, y; //x and y coordinates of a pixel
public:
  Pixel(){ x=0; y=0; }
  void set(int a, int b) { x=a; y=b; }
  int getx() const { return x; }
  int gety() const { return y; }
};
class Rectangle{
  int perimeter;
public:
  Pixel p1, p2; //embedded objects as public members
  Rectangle(){ perimeter=0; }
  void getperim();
};
```

```
void Rectangle::getperim() {
  int x1 = p1.getx(); //Calls the embedded objects‘
  int x2 = p2.getx(); //member functions
  int y1 = p1.gety();
  int y2 = p2.gety();
  cout<<"Top-left corner coordinates : [";
  cout<<x1<<','<<y1<<']'<<endl;
  cout<<"Bottom-right corner coordinates : [";
  cout<<x2<<','<<y2<<']'<<endl;
  cout<<"Perimeter = "<<(2*(x2-x1)+2*(y2-y1));
}
int main()
{
  Rectangle r;      //composed object
  r.p1.set(5,5);    //Using composed object to access
  r.p2.set(10,10); //members of the embedded objects.
  r.getperim();
  return 0;
}
```

```
Top-left corner coordinates : [5,5]
Bottom-right corner coordinates : [10,10]
Perimeter = 20
```



It is not a good design since users know that Rectangle has two pixels

```
#include <iostream>
using namespace std;
class Pixel {
  int x, y;
public:
  Pixel(){ x = 0; y = 0; }
  void set(int a, int b) { x = a; y = b; }
  int getx() const { return x; }
  int gety() const { return y; }
};
class Rectangle {
  int perimeter;
  Pixel p1, p2; //embedded objects as private members
public:
  Rectangle(){ perimeter = 0; }
  void getperim();
  void set(int, int, int, int);
};
void Rectangle::set(int x1, int y1, int x2, int y2) {
  p1.set(x1,y1);
  p2.set(x2,y2);
}
```

```
void Rectangle::getperim()
{
  int x1 = p1.getx();
  int x2 = p2.getx();
  int y1 = p1.gety();
  int y2 = p2.gety();
  cout<<"Top-left corner coordinates : [";
  cout<<x1<<','<<y1<<']'<<endl;
  cout<<"Bottom-right corner coordinates : [";
  cout<<x2<<','<<y2<<']'<<endl;
  cout<<"Perimeter = "<<(2*(x2-x1)+2*(y2-y1));
}
int main()
{
  Rectangle r;  //Instantiates a composed object and
  r.set(5,5,10,10);
  r.getperim(); //calls its interface function
  return 0;
}
```

```
class Rectangle {
  public:
      Rectangle( );
      void getperim( );
  private:
      p1: //public members of Pixel
            int getx( );
            int gety( );
            void set(int a,int b);

      p2: //public members of Pixel
            int getx( );
            int gety( );
            void set(int a,int b);

      int perimeter;
};
```

```
int main( )
{
      Rectangle r;
      r.getperim( );
      return 0;
}
```

Facade design pattern

Constructing and Destroying Composed Classes

- If a class is composed from one or more classes, the embedded objects will be constructed first when the composed object is instantiated.
- If an embedded class uses a non-default constructor function with arguments, the programmer must define a composed class constructor with a constructor initialization list.

- A constructor initialization list should be designed if the subclasses have constructors with arguments.
- The names of the embedded objects are used instead of the names of the base classes to pass values between constructors.

```
composed-constructor( parameter list ):
subobj1(values1),subobj2(values2),...,subobjn(valuesn)
{
//body of the composed constructor
}
```

- The same syntax can also be used if a class is composed of built-in types, such as int, float…
- The constructors of sub-objects are called before the execution of the body of the composed class constructor.

```
Class Pixel
{
  int x, y;
public:
  Pixel(int, int);
}
class Rectangle
{
  Pixel p1, p2;
public:
  Rectangle(int a, int b, int c, int d) : p1(a, b),
    p2(c, d)
  {
  }
  ...
}
```

```
#include <iostream>
using namespace std;
class Speaker {
  float impedance;
public:
  Speaker(float imp): impedance(imp) {
      cout<<"Constructing speaker."<<endl;
  }
  float getimp()const { return impedance; }
  ~Speaker() { cout<<"Destroying speaker."<<endl; }
};
class Amplifier {
  float impedance;
public:
  Amplifier(float imp ): impedance(imp) {
    cout<<"Constructing amplifier."<<endl;
  }
  float getimp()const { return impedance;}
  ~Amplifier() { cout<<"Destroying amplifier."<<endl; }
};
```

```
class Stereo {
  Speaker sp;
  Amplifier amp;
public:
  Stereo(float x, float y):sp(x), amp(y) {
    cout<<"Constructing stereo."<<endl;
  }
  void matching() {
    if(sp.getimp()==amp.getimp())
      cout<<"Impedances are matched."<<endl;
    else
      cout<<"Impedances are not matched."<<endl;
  }
  ~Stereo()
  {
    cout<<"Destroying stereo."<<endl;
  }
};
```

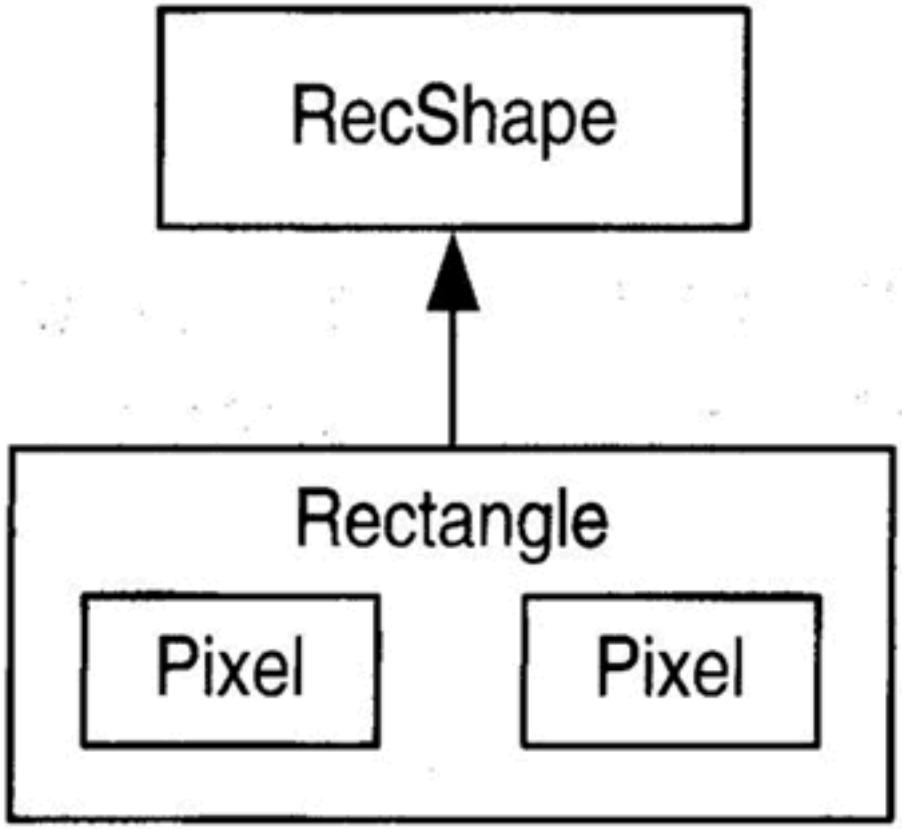
```
int main()
{
  Stereo st(8, 8);
  st.matching();
  return 0;
}
```

```
Constructing speaker.
Constructing amplifier.
Constructing stereo.
Impedances are matched.
Destroying stereo.
Destroying amplifier.
Destroying speaker.
```

- A composed class can contain pointers to sub-objects as members.
- This class may use its constructor function to dynamically allocate sub-objects and store their addresses in the member pointers.
- The composed class destructor must free memory dynamically allocated by the constructor.

```
class Stereo {
  Speaker * sp;
  Amplifier * amp;
public:
  Stereo(float x, float y) {
    sp=new Speaker(x);
    amp=new Amplifier(y);
    cout<<"Constructing stereo."<<endl;
  }
  void matching() {
    if(sp->getimp()==amp->getimp())
      cout<<"Impedances are matched."<<endl;
    else
      cout<<"Impedances are not matcheed."<<endl;
  }
  ~Stereo() {
    delete sp;
    delete amp;
    cout<<"Destroying stereo."<<endl;
  }
};
```

Combining Inheritance



```
#include <iostream>
using namespace std;
class Pixel {
  int x, y;
public:
  Pixel(int a, int b) {
    x = a;
    y = b;
    cout<<"SubConstructor"<<" x="<<x<<" y="<<y<<endl;
  }
  ~Pixel(){cout<<"SubDestructor"<<endl;}
};
class RecShape {
protected:
  int lg, wd;
public:
  RecShape(int l, int w) {
    lg = l;
    wd = w;
    cout<<"BaseConstructor"<<" lg="<<lg<<" wd="<<wd<<endl;
  }
  ~RecShape(){cout<<"BaseDestructor"<<endl;}};
```

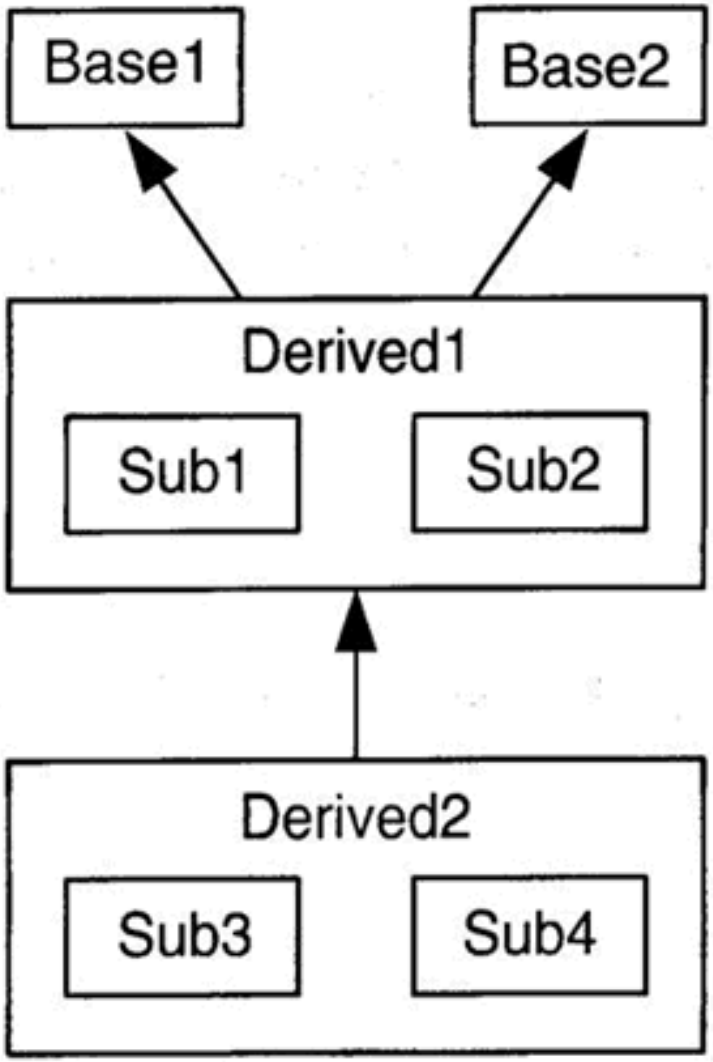
```
class Rectangle:public RecShape {
  int perimeter;
  Pixel p1, p2;
public:
  Rectangle(int x1,int y1,int x2,int y2):
  RecShape(x2-x1,y2-y1),p1(x1,y1),p2(x2,y2) {
    perimeter = 0;
    cout<<"CombConstructor"<<" x1="<<x1<<" y1="<<y1;
    cout<<" x2="<<x2<<" y2="<<y2<<endl;
  }
  void getperim() {
    cout<<"Perimeter = "<<(2*lg + 2*wd)<<endl;
  }
  ~Rectangle(){cout<<"CombDestructor"<<endl;}
};
int main() {
   Rectangle r(5,5,10,10);
   r.getperim();
   return 0;
}
```

Derived & Composed class Base class Embedded objects

```
Rectangle(int x1,int y1,int x2,int y2): RecShape(x2-x1,y2-y1),p1(x1,y1),p2(x2,y2)
{

    //body of constructor
}
```

```
BaseConstructor lg=5 wd=5
SubConstructor x=5 y=5
SubConstructor x=10 y=10
CombConstructor x1=5 y1=5 x2=10 y2=10
Perimeter = 20
CombDestructor
SubDestructor
SubDestructor
BaseDestructor
```



Constructor calls:	Destructor calls:
Base1	Derived2
Base2	Sub4
Sub1	Sub3
Sub2	Derived1
Derived1	Sub2
Sub3	Sub1
Sub4	Base2
Derived2	Base1