

Building Up Classes



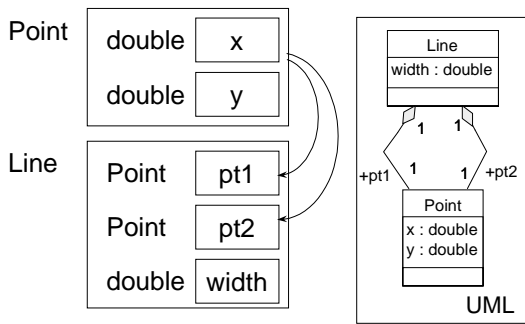
• Composition

- Incorporate existing class objects
- Objects become members of new class

• Inheritance

- Extend an existing class (“is a”)
- New class based on existing class

Composition: Object Structure



Component Class Definition

```

class Point
{
public:
    Point (double xval = 0.0,
           double yval = 0.0);
    double GetX() const;
    double GetY() const;
private:
    double x;
    double y;
    ...};
  
```

Class to describe a point, as specified in Cartesian coordinates.

Composite Class Definition

```

class Line  New class provides interface
{          to embedded objects
public:
  Line ();
  const Point& GetP1() const;
  void SetWidth (double w);
  ...
private:
  Point pt1;
  Point pt2;
};

```

Point objects members of class Line

Member Initialization Review

- **Member objects private**
 - Almost always
- **Member constructors called**
 - Before enclosing class constructor
 - Default constructor (by default)
 - Or as specified in initializer list of enclosing class constructor

Member Initialization Example

```

Line constructor arguments
Line::Line (const Point& p1,
           const Point& p2,
           double wid)
{
  : pt1(p1), pt2(p2), width(wid)
  // No other init (?) Initialize Point
}

```

members

Another Option

• **Composition**

- Member object part of composite
 - But distinct from composite object
 - Line object “has a” point member

• **Inheritance**

- New class extends existing class
 - Fancy line object “is a” line object

Inheritance

• **New class derived from base**

- One base class (multiple in special cases)

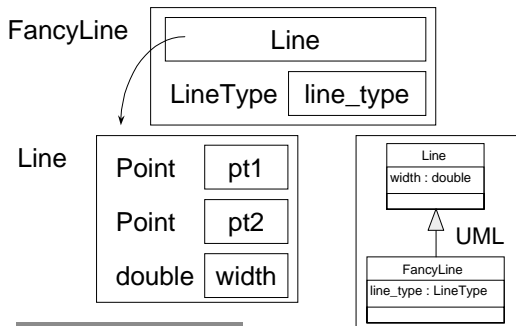
• **Base is subobject of derived**

- All base class members are included

• **Derived “is a” base**

- Base behavior inherited by derived

Derived Object Structure



Derived Class Declaration

Derived (new) class Base (existing) class

```

class FancyLine: public Line
{
public:
    enum LineType { SOLID, DASHED };
    void SetWidth (double w); ...
private:
    LineType line_type; ← New data member
... }

```

All base class members still present

What About This?

New function with same name as a member of the base class

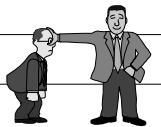
```

class FancyLine: public Line
{
public:
    enum LineType { SOLID, DASHED };
    void SetWidth (double w); ...
private:
    LineType line_type;
... }

```

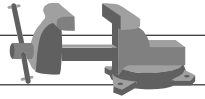
Which one would be called?

Overriding Functions



- **Derived member function**
 - Same name as base member function
- **Overrides (hides) base**
 - `lf: FancyLine line1;`
 - `line1.SetWidth(2)`
 - Derived version
 - Hides all overloads of same name

Base Construction



Base "constructor" in derived initializer list

```

FancyLine::FancyLine
(const Point& p1,
const Point& p2,
LineType lt,
double wid)
: Line(p1, p2, wid),
  line_type (lt)
{
    ...
};

```

Note: base class name; base subobject has no name (unlike member subobject)

Horizontal lines for notes.

Constructors & Destructors

• Constructors

- Derived class does not override
 - Called in sequence: base to derived
- Members: member declaration order

• Destructors

- Called in reverse order of constructors

Horizontal lines for notes.

Non-Inherited Functions

• Normal functions inherited

- Including most operators

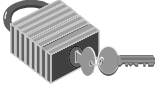
• Exceptions

- Constructors
 - Including copy constructor
- Destructor
- Assignment - operator=
 - Compiler-generated default

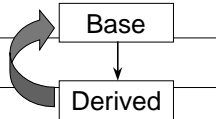
Horizontal lines for notes.

Access Keywords Revisited

- **Public**
 - Members accessible globally
- **Private (default)**
 - Members accessible within class
- **Protected**
 - Access within class & derived classes
 - Member functions
 - Data members (bad idea?)



Upcasting



- **Derived “is a” base**
- **Can be used as a base**
 - E.g., calculate length of `FancyLine`
 - Automatically handled by compiler
 - Be careful about passing by value
 - Object “sliced”
 - Keeps base part, discards derived part
 - Base copy constructor is used

Inheritance Summary

- **Derived object includes base**
- **Base member functions**
 - Still accessible on derived object
 - Unless overridden in derived class
- **Derived “is a” base**
 - Base class “contract” must be fulfilled
- **Purpose: reuse/extend**
 - Careful design required for success
