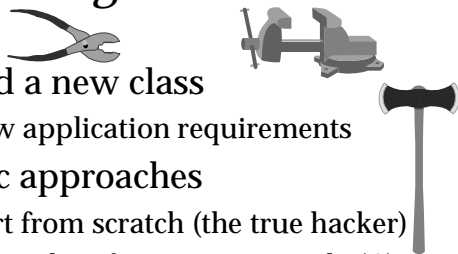


CS481 - Object-Oriented Programming


Building Abstractions



- ❖ Need a new class
 - New application requirements
- ❖ Basic approaches
 - Start from scratch (the true hacker)
 - Copy ideas from existing code (C)
 - Incorporate existing classes (C++)

1

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

2

Subobjects

- ❖ Composition
 - yields "object members"
 - ◆ embedded inside the new object
- ❖ Inheritance
 - yields 'base class objects'
 - ◆ embedded inside the new object

3

Composition

- ❖ New class has members
 - Built-in types
 - User-defined types (classes)
- ❖ Object members
 - Any number of objects
 - ◆ Same or different types (even array)

4

Composition

Point	double	x
	double	y

Line	Point	pt1
	Point	pt2
	double	width

5

Composition

```

class Point
{
public:
    Point(double xval = 0.0,
          double yval = 0.0);
    double getX() const { return x; }
    double getY() const { return y; }
private:
    double x;
    double y;
...};

```

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

6

CS481 - Object-Oriented Programming

Composition

```

class Line
{
public:
    Line();
    const Point& getP1() const
        { return pt1; }
    setWidth(double w); ...
private:
    Point pt1, pt2;
    double width;
};
    
```

Class to describe a line, as determined by its endpoints.

7

Composition

```

class Line
{
public:
    Line();
    const Point& getP1() const
        { return pt1; }
    setWidth(double w); ...
private:
    Point pt1, pt2;
    double width;
};
    
```

Point objects embedded as private members in class Line

8

Composition

```

class Line
{
public:
    Line();
    const Point& getP1() const
        { return pt1; }
    setWidth(double w); ...
private:
    Point pt1, pt2;
    double width;
};
    
```

New class controls interface to embedded objects

9

Composition

```

class Line
{
public:
    Line();
    const Point& getP1() const
        { return pt1; }
    setWidth(double w); ...
private:
    Point pt1, pt2;
    double width;
};
    
```

Note: this makes Point type visible to user of Line

10

Composition

```

class Line
{
public:
    Line();
    const Point& getP1() const
        { return pt1; }
    setWidth(double w); ...
private:
    Point pt1, pt2;
    double width;
};
    
```

This requires the user to know about implementation!

11

Composition

```

class Line
{
public:
    Line ();
    const Point& getP1() const
        { return pt1; }
    set
private:
    double getPlX () const
        { return p1.getX(); }
};
    
```

Alternative (??):

12

CS481 - Object-Oriented Programming

Member Initialization

- ❖ Member objects usually private
 - But, not always! (car example)
- ❖ Member constructors called
 - Before enclosing class constructor
 - Default constructor (by default!)
 - ◆ Can be non-default! (special syntax)

13

Member Construction

```
Line::Line(const Point& p1,
           const Point& p2,
           double wid)
: pt1(p1), pt2(p2), width(wid)
{ /* other Line set-up */
...
};
```

14

C'tor Initializer List

Note colon and special syntax

```
Line::Line(const Point& p1,
           const Point& p2,
           double wid)
: pt1(p1), pt2(p2), width(wid)
{ /* other Line set-up */
...
};
```

15

C'tor Initializer List

Line constructor takes endpoints and width

```
Line::Line(const Point& p1,
           const Point& p2,
           double wid)
: pt1(p1), pt2(p2), width(wid)
{ /* other Line set-up */
...
};
```

16

C'tor Initializer List

Initializer list: member constructors

```
Line::Line(const Point& p1,
           const Point& p2,
           double wid)
: pt1(p1), pt2(p2), width(wid)
{ /* other Line set-up */
...
};
```

17

C'tor Initializer List

Initializer list: member constructors

Note: member name, not class name!

```
Line::Line(const Point& p1,
           const Point& p2,
           double wid)
: pt1(p1), pt2(p2), width(wid)
{ /* other Line set-up */
...
};
```

18

CS481 - Object-Oriented Programming

Member Construction

- ❖ Memberwise initialization
 - Constructors called in declaration order (not initializer list order)
- ❖ Constructor used
 - As specified, if in initializer list
 - Otherwise, default - (built-in types???)

19

Another Option

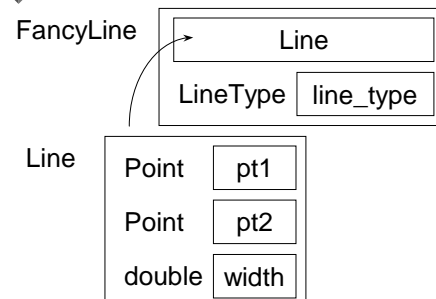
- ❖ Composition
 - One or more member objects
 - ◆ Part of composite object (subobject?)
 - ◆ But distinct from composite object
- ❖ Inheritance
 - New class “built on” existing class

20

Inheritance

- ❖ New class specifies base class
 - One base (more for multiple inheritance)
- ❖ Base is a subobject of derived
 - All base class members are included
- ❖ Derived “is a” base
 - Base behavior inherited by derived

21

Composition & Inheritance

22

Inheritance

```

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

23

Inheritance

Base declaration: public means
base class public members remain so

```

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

24

CS481 - Object-Oriented Programming

Inheritance

New members declared;
all base class members still present

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

25

Inheritance

Overriding member declared;
hides base class member (bad idea?)

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

26

Overriding Functions

- ❖ Derived member function
 - Same name as base member function
- ❖ Overrides (hides) base
 - If: FancyLine line1;
 - line1.setWidth() - derived version
 - Hides all overloads of name

27

Base Construction

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

28

Base Construction

Base "constructor" in initializer list

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

29

Members (Again)

Data member name in initializer list

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

30

CS481 - Object-Oriented Programming

Members (Again)

Data member name in initializer list

```
FancyLine::FancyLine
```

Note: pseudo-constructor for built-in types; one argument of appropriate type.

```
    double wid)
: Line(p1, p2, wid),
  (line_type(lt))
{ ...
};
```

31

Ctors & Dtors

❖ Constructors

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

❖ Destructors

- Called in reverse order of constructors

32

Non-Inherited Fns.

- ❖ Normal functions inherited
 - Including most operators
- ❖ Exceptions
 - Constructors & destructors
 - Assignment - operator=
 - ◆ Compiler-generated default

33

Member Access (Again)

- ❖ Public
 - Members accessible globally
- ❖ Private (default)
 - Members accessible within class
- ❖ Protected
 - Access within class & derived classes

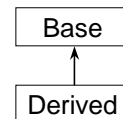
34

Inheritance Access

- ❖ Public (most common)
 - As in example
- ❖ Private (default)
 - Seldom used; private object instead
- ❖ Protected
 - Limits knowledge of base class

35

Upcasting



- ❖ Derived "is a" base
- ❖ So derived can be used when a base is called for
 - Automatically by compiler
 - If base class is visible
 - ◆ Public inheritance (or protected?)

36

CS481 - Object-Oriented Programming

