

Chapter 3

Using Objects of Different Classes

Section 1.5:

Composition of Objects

Composition:

An organized collection of components interacting to achieve a coherent, common behavior.

Why Compose Classes?

- Permits “lego block” approach
 - Each object captures one reusable concept
 - Composition conveys programming intent clearly
- Creates more readable code
- Helps reuse
- Simplifies propagation of change

Whole-Part Relationship: “Has A”

Example:

Web browser = HTML parser + viewer

- Web browser
 - “has a” HTML parser
 - “has a” viewer

Two Forms of Composition

- Association (acquaintance)
 - Example:
 - linked list of head, nodes connected by pointers
 - Used in Chapter 3 - more later
- Aggregation (containment)
 - Example
 - Web browser object contains private parser, viewer objects
 - Used later in Chapter 4

Section 3.1:

Communicating Objects

- Think of the "Borg" on Star Trek.
 - 1 Borg crew member = 1 object
 - "Collective" of borgs = composition of objects
- Each Borg must continually communicate
- A Borg can be
 - a "sender" or
 - a "receiver"
- Similarly, objects can be senders or receivers

3 Ways Objects Communicate

- by name

```
Stack s;  
main() {  
    class MyClass { int x; C() { x=s.pop(); }  
    ... }  
    MyClass m;  
    ...  
}
```

- by parameter passing

```
{ Stack s; f(s); }
```

- by return value

```
Stack f()  
{ Stack* s; s = new Stack(); ...; return s; }
```

3 Ways Objects Communicate

- **by name:**

one object is in a scope where name is visible to other object

- **by parameter passing:**

one class's method takes an object as one or more of its parameters.

- **by return value:**

method's return type is object

What's the sending/receiving object in each case?

Different Ways to Communicate

- Is object communicated by
 - copying
 - reference
 - pointer
- Can receiver modify object?
- If receiver modifies, does sender see changes?
- What syntax is used in receiver to access (. or ->)

Characteristics of Communicated Objects

Table 3-1

Technique	Copied	Changeable	Visible	Syntax
by copy	yes	yes	no	.
by reference	no	yes	yes	.
by pointer	no	yes	yes	->
by const reference	no	no	no	.
by const pointer	no	no	no	->

Comparison

■ By Copy:

- + Sender is "isolated" from changes by receiver
- No good if sender/receiver want to share object
- Bad if object is large (why?)

■ By Identity (pointer or ref):

- No isolation
- + Permit sharing
- + Good for large objects

Section 3.2:

Remember the Frame Class?

Let's refine Frame to use more OO concepts...

```
class Frame {                                // Version 1
public:
    Frame (char *name, int initXCoord, int initYCoord,
           int initWidth, int initHeight);
    Frame (char *name, int initXCoord, int initYCoord);
    Frame (char *name);
    Frame ();
    void MoveTo (int newXCoord, int newYCoord );
    void Resize (int newHeight, int newWidth );
};
```

The Location Class

```
class Location {                                // Version 1
private:
    // encapsulated implementation goes here
public:
    Location(int x, int y);                    // specific location
    Location();                                // default location
    int Xcoord();                              // return x-axis coordinate
    int Ycoord();                              // return y-axis coordinate
};
```

The Shape Class

```
class Shape {                                  // Version 1
private:
    // encapsulated implementation goes here
public:
    Shape(int width, int height);             // specific shape
    Shape();                                  // default shape
    int Height();                             // return height
    int Width();                              // return width
};
```

Create Location and Shape Objects

```
Location nearTop(20, 20),
           nearCenter(500, 500);
Shape     smallSquare(50, 50);
Shape     largeSquare(500, 500);
```

Revised Frame Class (Version 3)

Redefine the Frame class to take advantage of Location and Shape classes:

```
class Frame {                                // Version 3
private:
    // encapsulated implementation goes here
public:
    Frame(char* name, Location p, Shape s); // exact description
    Frame(char* name, Shape s, Location p); // exact description
    Frame(char* name, Location p);         // default shape
    Frame(char* name, Shape s);           // default location
    Frame(char* name );                   // name only
    Frame();                               // all defaults;
    void MoveTo(Location newLocation);    // move the window
    void Resize(Shape newShape);         // change shape
    void Resize(float factor);           // grow/shrink by factor
    ...                                  // other methods
};
```

Revised Frame Class (Version 3 Continued)

```
class Frame {                                     // Version 3
(continued)
private:
    ...
public:
    ...                                     // other methods
    void DrawText(char *text, Location loc);
    void DrawLine(Location end1, Location end2);
    void DrawCircle(Location center, int radius);
    void Clear();
    void Clear(Location corner, Shape rectangle);
    ...
};
```

Create Frame Object (Version 3)

Frame objects can be created in the following way:

```
Frame smallTop("Square Near Top", nearTop, smallSquare);
Frame largeCenter("Big at Middle", nearCenter,
                 largeSquare);
Frame someWhere("Big Somewhere", largeSquare);
Frame someSize("At Middle", nearCenter);
Frame anyKind("Name Only - Rest Defaults");
```

Summary of Advantages

- Location/Shape convey more than x,y
- Code is self-documenting
 - `"void Clear(Location corner, Shape rectangle);"`
 - `"nearTop"`
- (See textbook for more...)

Section 3.2, page 87:

Returning Objects by Copy

A method can return an object to its caller...

Returning an Object

Old Way:

```
class Frame { //Version 1
public:
    void TextSize (char *msg, int& width, int& height);
    ...
};
```

New Way:

```
class Frame { // Version 3
public:
    Shape TextSize(char *msg);
    ...
};
```

Old Versus New Code

Old:

```
display.TextSize(msg, width, height);
Shape msgShape(width, height);
display.Clear(msgLocation, msgShape);
```

Inconsistent args

New:

```
Shape msgShape = display.TextSize(msg);
display.Clear(msgLocation, msgShape);
```

The File Class Example: Returning Objects by Copy

```
// A class that represents a file in the file system
class File {
private:
    // encapsulated implementation goes here
public:
    File(char* fileName); // represents file with given
                        // name
    File();              // unknown, as yet, file
    char* Name();       // reply name of file
    int  Exists();      // does file Exist?
    void View();        // scrollable view window
    void Edit(char* editor); // edit file using
                        // "editor"
    void Delete();      // delete from file system
                        // (gone!)
    ~File();            // free name
};
```

The FileQuery Class Example: Returning Objects by Copy

```
class FileQuery {
private:
    // encapsulated implementation goes here
public:
    FileQuery( char* path, char* filter );
        // prompt with path and filter
    FileQuery( char* path ); // prompt with path default
                        // filter
    FileQuery( );           // use all defaults
    File AskUser();        // get file from user via
                        // dialog
    ~FileQuery();
};
```

Using the File and FileQuery class

```
FileQuery query("/home/kafura", "*.ps");  
File file = query.AskUser();  
file.View();
```

The FileChooser Example: Returning Objects by Copy

```
class FileChooser {  
private:  
    // encapsulated implementation goes here  
public:  
    FileChooser(char* path, char* filter);  
        // search at path with filter  
    FileChooser(char* path); // search at path, no filter  
    FileChooser();           // search at CWD, no filter  
    File AskUser();         // get file via dialogue  
    ~FileChooser();         // clean up  
};
```

The FileNavigator Example: Returning Objects by Copy

```
class FileNavigator {
private:
    // encapsulated implementation goes here
public:
    FileNavigator(char* path, char* filter);
        // start at path using filter
    FileNavigator(char* path); // start at path, no filter
    FileNavigator();          // start at CWD, no filter
    File AskUser();           // get file via dialogue
    ~FileNavigator();         // clean up
};
```

Section 3.3:

Anonymous Objects

- An object that has no name associated with it
- Eliminates having to name objects that are needed temporarily
- Useful in providing the default value for a parameter that is an object of a class rather than a built-in type.

Example of the Need for Anonymous Objects

```
Location initialLocation(100, 100),
        displayLocation(200,200);
Shape    initialShape(150, 200),
        displayShape(300, 200);

Frame window (initialLocation, initialShape);
Frame display (displayLocation, displayShape);
...

Location newLocation(300,300);
Location newShape (150,150);
window.MoveTo(newLocation);
display.Resize(newShape);
```

Previous Example using Anonymous Objects

```
Frame window ( Location(100,100), Shape(150, 200) );
Frame display ( Location(200,200), Shape(300, 200) );

...

window.MoveTo( Location(300,300) );
display.Resize( Shape(150,150) );
```

Example of Anonymous Objects Providing Defaults

```
class Frame {
  private:
  ...
  public:
  ...
  void MoveTo (Location loc = Location(10,10));
  ...
};
```

Communicating Objects by Reference and Pointers

■ Uses of reference and pointer communication:

■ result parameters

- | sender can see modifications
- | `void TextSize(int& width, int& height);`

■ managers

- | one object manages others
- | `void MinimizeAll();`

■ associations

- | allow ongoing interaction
- | `void NotifyOnChange(Counter* count);`

The Query Class

```
class Query {
private:    // encapsulated implementation
public:

    Query (char* searchText);
    Query();
    void  SetSearch(char* searchText);
    char* GetSearch();
    void  AskUser();
    void  SetResult(char* resultText);
    char* GetResult();
    ~Query();
};
```

File Class Extension: Example of Passing Objects

```
class File {
private:

public:
    ...
    void SearchFor (Query& q);           // by reference
    void SearchFor (Query* q);         // by pointer
    ...
};
```

Usage of the Query Class: Example of Passing Objects

```
Query query1("object"), query2("oriented");
Query *query3;
Query *query4;

query3 = new Query("programming");
query4 = new Query("C++");

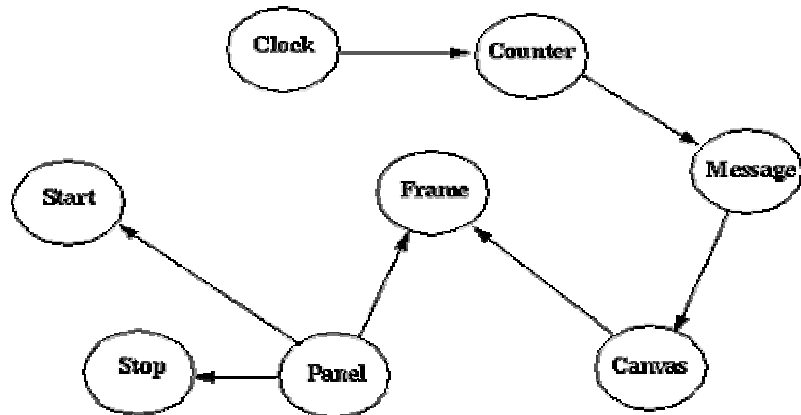
File bookList("booklist");
bookList.SearchFor( query1);           // by reference
bookList.SearchFor(&query2);          // by pointer
bookList.SearchFor( query3);          // by pointer
bookList.SearchFor(*query4);          // by reference

char* result1 = query1.GetResult();
char* result2 = query2.GetResult();
char* result3 = query3->GetResult();
char* result4 = query4->GetResult();
```

Building Systems by Assembling Parts

- **Composition**
 - an organized collection of components interacting to achieve a coherent, common behavior
- **Association**
 - a composition of independently constructed and externally visible parts.
- **Key Language Ideas**
 - Passing objects by copy
 - Passing objects by reference, pointer

An Association of Objects



The Message Class: A Simple Association

```
class Message {
private:    //encapsulated implementation
public:
    Message(char *textString, Location whereAt);
    Message(Location whereAt);
    void DisplayIn(Frame& whichFrame);
    void MoveTo(Location newLocation);
    void setText(char* newText);
    char* getText();
    void Clear();
    void Draw();
    ~Message();
};
```

The Data for the Message Class

```
class Message {
private:    // encapsulated implementation
    char *msgText;    // display this text string
    Frame *msgFrame;    // in this Frame
    Location msgLocation; // at this Location in the Frame

public:

    ...

};
```

Establishing the Association

By Reference:

```
DisplayIn (Frame& whichFrame) {
    msgFrame = &whichFrame;
}
```

By Pointer:

(not defined in Message class, but could be)

```
DisplayIn (Frame* whichFrame) {
    msgFrame = whichFrame;
}
```

Using the Association

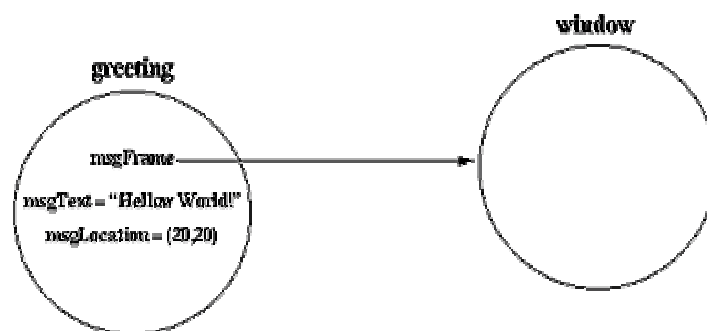
// declaration

```
Frame window("Message Test", Location(100,100),  
             Shape(200,200));  
Message greeting("Hello World!", Location(20,20));
```

// code

```
greeting.DisplayIn(window);
```

A Simple Association



The PrimitiveMessage Class

```
class PrimitiveMessage {
private:
    . . .
public:
    PrimitiveMessage(char *text);
    void SetText(char* newText);
    char* GetText();
    ~PrimitiveMessage();
};
```

Blinking Text Example Using PrimitiveMessage Class: A Simple Association

```
Frame window("Blinking Text",
             Location(100,100),Shape(200,200));
PrimitiveMessage greeting("Hello World!");
Location greetingLocation(20, 50);
int onoff;          // is text visible: yes=1, no=0

void OnStart() {
    window.Clear();
    window.DrawText(greeting.GetText(), greetingLocation);
    onoff = 1;
}

. . . .
```

Blinking Text Example Using PrimitiveMessage Class: A Simple Association (Continued)

```
. . . . .
void OnTimerEvent() {
    if (onoff == 1) {           // text is visible
        Shape greetingShape =
            window.TextShape(greeting.GetText());
        window.Clear(greetingLocation, greetingShape);
        onoff = 0;
    }
    else {                     // text is not visible
        window.DrawText(greeting.GetText(),greetingLocation);
        onoff = 1;
    }
}
void OnPaint() {
    if (onoff == 1) // text is visible
        window.DrawText(greeting.GetText(), greetingLocation);
}
}
```

Message Class

```
class Message {
private: // hidden data
public:
    Message(char *textString, Location whereAt);
    Message(Location whereAt);
    void DisplayIn(Frame & whichFrame);
    void MoveTo(Location newLocation);
    void SetText(char *newText);
    char * GetText();
    void Clear();
    void Draw();
    ~Message();
};
```

Using Message Class

```
Frame window("Message Test", Location(10, 10),
             Shape(200, 200));
Message greeting("Hello World", Location(20, 50));
int onoff = 1;
void OnStart() {
    window.Clear();
    greeting.DisplayIn(window);
    greeting.Draw();
}
void OnTimerEvent() {
    if (onoff) {greeting.Clear(); onoff = 0;}
    else      {greeting.Draw(); onoff = 1;}
}
void OnPaint() {
    if (onoff) greeting.Draw();
}
```

BlinkMessage Class

```
class BlinkMessage {
private: // hidden data
public:
    BlinkMessage(char *textString, Location whereAt);
    BlinkMessage(Location whereAt);
    void DisplayIn(Frame& aFrame);
    void MoveTo(Location newLocation);
    void SetText(char *newText);
    char * GetText();
    void Blink();
    void Redraw();
    ~BlinkMessage();
};
```

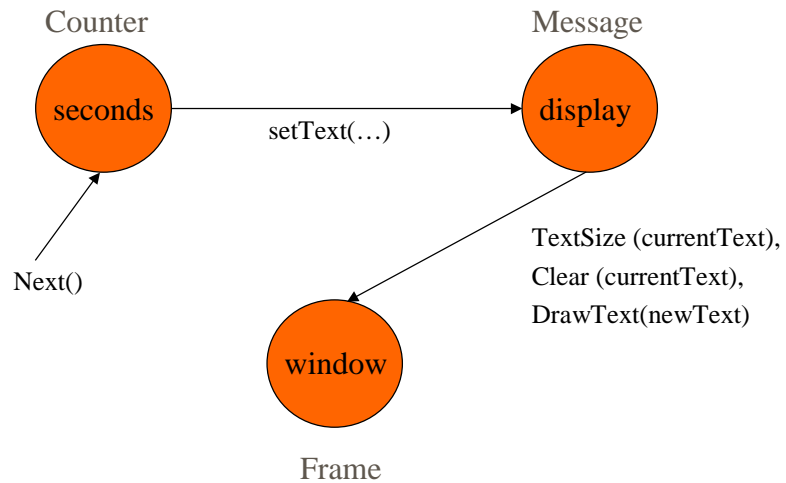
Using BlinkMessage

```
Frame window("Message Test", Location(10,10),
             Shape(200,200));
BlinkingMessage greeting("Hello World", Location(20, 50));
void OnStart() {
    window.Clear();
    greeting.DisplayIn(window);
    greeting.Blink();
}
void OnTimerEvent() {
    greeting.Blink();
}
void OnPaint() {
    greeting.Redraw();
}
```

Comparing the Alternatives

- **BlinkMessage**
 - + solves blinking text problem
 - unused functionality in non-blink problems
- **Message**
 - + more general interface
 - limited functionality
- **PrimitiveMessage**
 - + low overhead
 - no functionality (but suitable in some cases)

A Simple Counter Association



The Counter Class: Another Association

```
class Counter {
private:
    // encapsulated implementation goes here
public:
    Counter (int start, int end); // count up/down from
                                // start to end
    Counter(); // count upwards from zero
    void Next(); // increment/decrement by 1
    void Reset(); // reset to original state
    void Reset(int nowThis); // reset to specified value
    void ConnectTo(Message& msg); // show current
                                // value here
    ~Counter(); // destructor
};
```

Using Counter

```
Frame window("Counter", Location(100,100), Shape (200,200));
TextBox countDisplay("", Location(10,10));
Counter clickCount;

void OnStart() {
    countDisplay.DisplayIn(window);
    clickCount.ConnectTo(countDisplay);
}

void OnPaint() {
    countDisplay.Draw();
}

void OnTimerEvent() {}

void OnMouseEvent() {char *frameName, int x, int y, int
    buttonState) {
    if (buttonState & leftButtonDown) {
        clickCount.Next();
    }
}
```

The Clock Class: Another Association

```
class Clock {
private:
    // encapsulated implementation goes here
public:
    Clock (int interval);    // milliseconds between "ticks"

    void ConnectTo(Counter & count); // change count on each
    // "tick"

    void Start();           // (re)start Clock

    void Stop();           // halt Clock
};
```

Using the Counter and Clock Class: Another Association

```
Frame window ("Timer", Location(100,100),
              Shape(200,200));
Message label("Seconds:", Location(10,10));
Message display("", Location(100,10));
Counter seconds;
Clock timer(1000);

void OnStart() {
    timer.ConnectTo(seconds);
    second.ConnectTo(display);
    display.DisplayIn(window);
    timer.Start();
}
. . . .
```

Using the Counter and Clock Class: Another Association (Continued)

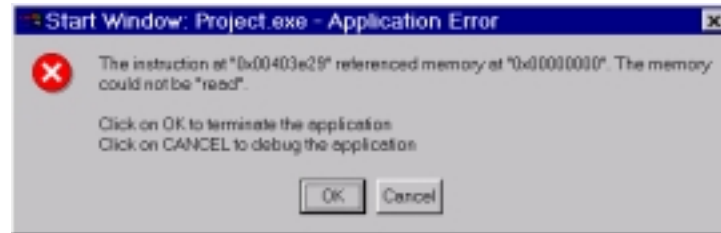
```
. . . .
void OnPaint() {
    display.Draw();
}

void OnTimerEvent() {}

void OnMouseEvent() {char *frameName, int x, int y, int
                    buttonState) {}}
```

A mystery...

- Closing timer window gives this...



- Why?
- How do I fix the problem?

Solution

- Program may end abnormally if timer goes off while program is in process of terminating.
- Solution:

```
void OnMouseEvent(char *frameName, int x, int y,
                  int buttonState){
    if(buttonState & leftButtonDown)
        timer.Stop();
}
```

Try this now in small groups

- Write program to count mouse drags
- (There's a similar example in text - don't look at that.)

Solution

```
Frame window("Counter", Location(1,1), Shape(200,200));
Message countDisplay("", Location(10,10));
Counter clickCount;
void OnStart() {
    countDisplay.DisplayIn(window);
    clickCount.ConnectTo(countDisplay);
}
void OnPaint() {
    countDisplay.Draw();
}
void OnTimerEvent() {}
void OnMouseEvent(char* n, int x,int y, int bState) {
    if (bState & isDragging) clickCount.Next(); }
```

Section 3.5, page 120:

A New Version of Class Frame

Frame Version 3 can

- Draw circles, lines, text

Does it “scale up” to a commercial version

- New shapes (ovals, polygons, ...)?
- Fills and patterns?
- New methods to add buttons, sliders, ...?

No! So let's take break it into multiple classes...

New Organization

- Frame: Resize, MoveTo
- Canvas: DrawText/Line/Circle, Clear
- Panel: Add Button, TextBox

- TextBox: GetText, SetText
- Button
- Globals: void OnPush(char* buttonName)

Section 3.6:

Frame Class (Version 4)

```
class Frame { // Version 4
private:
    ...
public:
    Frame(char* name, Location p, Shape s); //exact description
    Frame(char* name, Shape s, Location p); //exact description
    Frame(char* name, Location p); //default shape
    Frame(char* name, Shape s); //default location
    Frame(char* name ); //name only
    Frame(); //all defaults;
    int IsNamed(char* aName); //is it your name?
    void MoveTo(Location newLocation); //move the window
    void Resize(Shape newShape); //change shape
    void Resize(float factor); //grow/shrink factor
    ~Frame();
};
```

The Canvas Class

```
class Canvas {
private:    ...
public:
    Canvas(Frame& fr, char* nm, Location loc, Shape sh);
    int IsNamed(char* aName);
    void DrawText(char *text, Location loc);
    Shape TextSize(char *msg);
    void DrawLine(Location p1, Location p2);
    void DrawCircle(Location center, int radius);
    void Clear();
    void Clear(Location corner, Shape rectangle);
    ~Canvas();
};
```

Canvas goes in Frame

Drawing
methods

The Panel Class

```
class Panel {
  private: ...
  public:
    Panel( Frame& fr,
          char *nm,
          Location loc,
          Shape sh);
    char* getName();
    void Add(Button& button);
    void Add(TextBox& tbox);
    ~Panel();
};
```

Panel goes in Frame

Associates interactive stuff with the panel

The Button Class

```
class Button {
  private: // ...
  public:
    Button(char* name, Location loc, Shape sh);
    int IsNamed(char* name);
    ~Button();
};
```

Used with global "OnPush(...)" method.

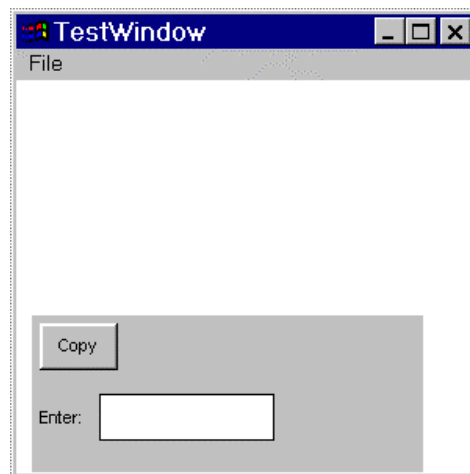
The TextBox Class

```
class TextBox {  
    private: // ...  
    public:  
        TextBox(Location p, Shape s, char* label);  
        TextBox(Location p, Shape s);  
        TextBox(char* label);  
        TextBox();  
        ~TextBox();  
        char* GetText();  
        void SetText(char* val);  
};
```

Get what user types

Set initial value

Example Using Buttons and TextBoxes



Example Using Buttons and TextBoxes

```
Frame    window ("TestWindow", Location(100,100),
              Shape(500, 300));

Canvas   canvas (window, "DrawAreas", Location(1, 1),
              Shape(100, 100));

Panel    panel  (window, "Controls", Location(150, 10),
              Shape(300, 100));

Button   button ("Copy", Location(5, 5), Shape(50,30));

TextBox  tbox   (Location(5,50), Shape(150,30),
              "Enter:");

char     *textS;
```

Example Using Buttons and TextBoxes (Continued)

```
void OnStart() {           // called once on button push
    canvas.Clear();
    panel.Add(button);
    panel.Add(tbox);
    textS = (char*)0;
}

void OnPush(char *buttonLabel) {
    if (button.IsNamed(buttonLabel)) {
        canvas.Clear();
        textS = copystring(tbox.GetText());
        canvas.Drawtext(textS, Location(20, 20));
    }
}

void OnPaint() {
    canvas.DrawText(textS, Location(20,20));
}
```

Section 3.7:

Self Referencing

- A class definition may refer to itself

- Function parameters

```
class Location {
public:
    int SameAs(Location other);
}
```

- Return type

```
class Shape {
public:
    Shape Resize(float scaleFactor);
}
```

The Extended File Class: Self-Referencing Example

```
class File {
private:
public:
    File(char* fileName); // file w/ known name
    File(); // file w/ unknown name
    char* Name(); // reply name of file
    int Exists(); // does file Exist?
    void View(); // scrollable view window
    void Edit(char* editor); // edit file using "editor"
    void Delete(); // delete file

    void CopyTo(File& other); // copy me to other
    void CopyFrom(File& other); // copy other to me

    ~File(); // free name
};
```

Usage of the Extended File Class: Self-Referencing Example



```
FileNavigator nav;  
File sourceFile = nav.AskUser();  
File targetFile = nav.AskUser();  
  
sourceFile.CopyTo(targetFile);  
sourceFile.View();  
targetFile.View();
```