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

Communicating Objects

- Objects that communicate with each other often use other objects to carry information

- Example: Passing a note to your friend in class
  - Consider you and your friend to be objects
  - The paper note you pass is an object that conveys information
  - The note can be passed one way (unidirectional communication) or back and forth (bidirectional communication)
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 does modify, does sender see changes?
- What access syntax is used in receiver (. ->)
Comparison of answers

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3-1: Characteristics of Communicated Objects

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 Object?**

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

```cpp
class Frame { // Version 1
public:
    Frame (char *name, int initXCoord, int initYCoord, 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)
Redefined the Frame class to take advantage of the Location and Shape class.

```java
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 ResizerShape(newShape);            // change shape
     void Resizer(float factor);             // grow/shrink by factor
     // other methods
};
```

Create Frame Object (Version 3)

Frame objects can be created in the following way:

```java
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");
```

Revised Frame Class (V.3 Cont’d)

```java
class Frame { // Version 3 (continued)
private:
     ...                            // 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);
     ...;
};
```
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...)

Returning Objects by Copy

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

Returning an Object

Old Way:

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

New Way:

```java
class Frame { // Version 3
    ...
    public:
    Shape TextSize(char *msg);
    ...
};
```
Old Versus New Code

**Old:**
```java
display.TextSize(msg, width, height);
Shape msgShape(width, height);
display.Clear(msgLocation, msgShape);
```

**New:**
```java
Shape msgShape = display.TextSize(msg);
display.Clear(msgLocation, msgShape);
```

---

Section 3.2, page 90:

Example of Returning Objects by *Copy*

**File Dialog Example**

---

**Class `File`**

```java
class File {   // Represents file in file system
private:
    public:
        File(char* fileName); // file w/ known name
        File(); // file name unknown, as yet
        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
    }
```
class FileQuery {  // Prompts user to type a file name
private:
    ...
public:
    FileQuery( char* path, char* filter );  // prompt with path and filter
    FileQuery( char* path );  // prompt w/ path default filter
    FileQuery();  // use all defaults
FILE AskUser();  // get file from user via dialog

Return value is copy of object

Using Classes File and FileQuery

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

1st Alternative to FileQuery: Class FileChooser

class FileChooser {
private:...
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();  // pop up box
    ~FileChooser();  // clean up
2nd Alternative:  
Class FileNavigator  

```java
class FileNavigator {
    private:
        ...
    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
        FileAskUser(); // get file via dialogue
        ~FileNavigator(); // clean up
};
```

Section 3.3:  

**Anonymous Objects**

- **Anonymous** = object has no name  
  (However, objects class *do* have a name.)

- Avoids naming “temporary” objects

- Can provide default value for parameter that is an object of a class (not a built-in type.)

Something to Note in Previous Code

- Multiple classes often implement a common *interface*
- Example:  
  - `AskUser` method is in all 3 classes:  
    FileQuery, FileChooser, FileNavigator  
  - So you could substitute on class for the others  
  - (More on this in Chapter 4...)

5/25/99
CS2704 - Parks - Chapter3
Why Use Anonymous Objects?

Anonymous Objects as Default Value

class Frame {
private:

public:

    void MoveTo (Location loc = Location(10,10));

};

Section 3.4:

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);
**Example: Class Query**

class Query {
    private: ...
    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
    ...
};

Note programming trick:
Overloaded methods provide convenience for class user.
How would you define one method in terms of the other?

---

**Example of Query & File**

```
Query query;
File file("./Code/Frame.h");
TextFrame result("Query Results");
void OnStart(void) {
    query.SetSearch("Draw Text");
    result << "The query is: " << query.GetSearch() << "\n";
    file.SearchFor(query);
    result << "The search yields: " << query.GetResult() << "\n";
}
```
**Using Query Class: Passing Objects**

```cpp
Query query1("object");
Query query2("oriented");
Query *query3 = new Query("programming");
Query *query4 = new Query("C++");
```

Which identifiers are

- Objects?
- Pointers?
- References?

---

**Fill in the 8 blanks...**

```cpp
Query query1(...);
Query query2(...);
Query *query3 ...;
Query *query4 ...

File bookList("booklist");

bookList.SearchFor( ___query1 );
bookList.SearchFor( ___query2 );
bookList.SearchFor( ___query3 );
bookList.SearchFor( ___query4 );

char* result1 = query1__GetResult();
char* result2 = query2__GetResult();
char* result3 = query3__GetResult();
char* result4 = query4__GetResult();
```

---

**Section 3.4, page 105:**

**Building Systems by Assembling Parts**

- **Composition** (organized collection of components interacting to achieve a coherent, common behavior)

- **Association** (composition of
  - independently constructed &
  - externally visible parts)

(Aggregation -- Chapter 4)
Recall Hello World Program...

Frame win("Hello World Program", 200, 200, 400, 400); int lastx, lasty;
void OnStart(void) {
    win.Clear(); win.DrawText("Hello", 20,20); lastx = lasty = 20; }
void OnMouseEvent(char *frameName, int x, int y, int buttonState)
    if (buttonState & leftButtonDown) {
        win.Clear();
        win.DrawText("Hello",x,y); lastx = x; lasty = y;}
void OnPaint(void)
    win.Clear(); win.DrawText("Hello World!", lastx, lasty); }

That code is messy!

- lastx, lasty & DrawText
  used all over the place

- lastx, lasty keeps track of where we last drew
  "Hello" after mouse click

- A cleaner way: class Message ...

A Simple Association: The Message Class

Message keeps track of
  • text (e.g., "Hello")
  • where to draw itself
  • how to move itself
Improved Code

```cpp
Frame win("Hello World Program", Location(100,100), Shape(400, 400));
Message greeting("Hello", Location(20,20));
void OnStart(void) { win.Clear(); greeting.DisplayIn(win); greeting.Draw(); }
void OnMouseVent(char* frameName, int x, int y, int buttonState) {
    if (buttonState & leftButtonDown) {
        greeting.MoveTo(Location(x,y));
    }
}
void OnPaint(void) { greeting.Draw(); }
```

A Simple Association: The Message Class

```cpp
class Message {
    private: //...
    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 ();
};
```

Data for Message Class

```cpp
class Message {
    private: // encapsulated implementation
        char *msgText; // display this text string
        Frame *msgFrame; // in this Frame
        Location msgLocation; // at this Location in Frame
    public:
        ~
};
```
Section 3.5:

More On Associations

- Who’s responsible for what in Hello World?
  - Chapter 2 version:
    - Class Frame: draw/erase text
    - Globals: what to draw & where
  - With Message:
    - Class Frame: erase window
    - Globals: when to invoke Message's methods
    - Class Message: what to draw & where

So altering responsibility affects object structure.

The PrimitiveMessage Class

Here’s a pretty lazy class. It doesn’t want to take responsibility for much! It let’s the globals do all the hard work! We’re just asking for problems with this code...

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

Blinking Text, V.1: Uses PrimitiveMessage Class

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

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

Look at how much the globals must remember!
Blinking Text Version 1, continued

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

void OnPaint()
{ if (onoff == 1) // text is visible
  window.DrawText(greeting.GetText(), greeting.Location);
}

V.2: Let's use class Message from earlier...

class Message {
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 () ;
};

Message keeps track of
• text (e.g., "Hello")
• where to draw itself
• how to move itself

frame window("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();
}
Version 3

Class BlinkMessage
private: // hidden data
public:
    BlinkMessage (char *text, Location where);
    BlinkMessage (Location whereAt);
    void DisplayIn (Frame & aFrame);
    void MoveTo (Location newLocation);
    void SetText (char* newText);
    char* GetText () ;
    void Blink () ;
    void Redraw () ;
~BlinkMessage();

Blinking Text, V.3: Uses BlinkingMessage Class

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(); }
### A Simple Counter Association

![Diagram of a counter association between a counter and a message with a window and frame]

- **Counter**
  - seconds
  - Next()

- **Message**
  - display
  - TextSize (currentText), Clear (…), DrawText(newText, loc)

- **Window**
- **Frame**

---

### Review: The Message Class

```cpp
class Message {
  private: //...
  public:
    Message (char *textString, Location WhereAt);  // Message (Location whereAt);
    void DisplayIn (Frame & whichFrame);  // DisplayIn (Frame & whichFrame);
    void MoveTo (Location newLocation);  // MoveTo (Location newLocation);
    char* getText ();  // getText ()
    void Clear ();  // Clear ()
    void Draw ();  // Draw ()
    ~Message ();  // Destructor
};
```

---

### The Counter Class: Another Association

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

---
The Clock Class: Another Association

class Clock {
    private:
        ... 
    public:
        Clock (int interval);  // milliseconds 
        void ConnectTo(Counter& count);  // between "ticks"
        void Start();  // (re)start Clock
        void Stop();  // halt Clock
    };

Using the Counter & 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);  // Tick every 1 second
void OnStart() {
    timer.ConnectTo(seconds);
    seconds.ConnectTo(display);
    label.DisplayIn(window);
    display.DisplayIn(window);
    timer.Start();
}

Using the Counter & Clock Class: Another Association (Cont.)

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

void OnTimerEvent ()
{
    void OnMouseEvnt (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:

```c
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;

```c
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:
        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* name); // is it your name?
        void MoveTo(Location newLocation); // move the window
        void Resize(Shape newShape); // change shape
        void Resize(float factor); // grow/shrink factor
    public:
        ~Frame();
};

---

**The Canvas Class**

class Canvas { // Canvas goes in Frame
    private:
        Canvas(Frame &fr, char* nm, Location loc, Shape sh);
        int IsNamed(char* name);
        void DrawText(char *text, Location loc);
        void DrawLine(Location pl, Location p2);
        void DrawCircle(Location center, int radius);
        void Clear();
        void Clear(Location corner, Shape rectangle);
    public:
        ~Canvas();
};

---

**The Panel Class**

class Panel { // Panel goes in Frame
    private:
        Panel(Frame &fr, char* nm, Location loc, Shape sh);
        char* getName();
    public:
        // Associates interactive stuff with the panel
        void Add(Button& button);
        void Add(TextBox& textbox);
        ~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);
    }

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

char *textS;

Example Using Buttons and TextBoxes (Cont’d)

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

void OnPush(char *buttonLabel)
{ if (button.IsNamed(buttonLabel))
  { canvas.Clear();
    textS = copyString(textbox.GetText());
    canvas.DrawText(textS, Location(20, 20));
  }
}

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

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);
    }

Section 3.7:
class File { // Version 2
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();