Chapter 8

Typecasting and Inheritance

Type Casting

- Allows flexible structures to be built

- **Idea:**
  Treat a derived class object as if it were a base class object

- **Syntax:**
  BaseClassName (derivedClassObject)
  or
  (BaseClassName) derivedClassObject

  This syntax cannot be used with pointers.
Type Casting

We may only call methods from DisplayableNumber on dnp, even though it's really an object of type Number!
Type Casting Using References

TextBox display(Location(100,100), Shape(75, 50));
Number number(100);
number.Next();

DisplayableNumber&
displayable = (DisplayableNumber&) number;

displayable.ShowIn(display);
displayable.Show();

Type Casting Errors

DisplayableNumber* numPtr;
Number *count1 = new Number(100);
Number count2(200);
umPtr = (DisplayableNumber*) count1;
DisplayableNumber& numRef = (DisplayableNumber)count2

numPtr->Next();
umRef.Next();

Wrong! DisplayableNumber doesn't have a Next() method!
Why Type Casting?

- Allows us to treat a collection of objects uniformly by viewing them as their base class.
- Example 1: Shapes in a Graphical Editor
- Example 2: NumberPanel class example

Shapes

- Derive Circle and Rectangle from “Shape”
  - avoid code duplication
  - ShapeManager handles instances of Shape
  - E.g.:

```
Shape
Location loc
Rectangle bounds
SetLocation
GetBounds
...

GameCircle
SetCenter
SetRadius
Draw
...

GameRectangle
SetSize
Draw
...
```
Type Casting to Create a Polymorphic Structure

class NumberPanel {
    private:
        DisplayableNumber *number[3]; // array for simplicity
        int last;
        Panel *panel;
    public:
        NumberPanel();
        void Add( DisplayableNumber* num ); // add num
        void ShowIn(Panel & panel); // put all objects here
        void Show(); // Show all objects in panel
};

Implementation of the NumberPanel Class

NumberPanel::NumberPanel()
{   last = -1;
    panel = (Panel *)0;
}

void NumberPanel::Add( DisplayableNumber* num)
{   if ( last < 2 ) {
        number[++last] = num;
        TextBox* tbox = new TextBox(Location(last*60,20)
            Shape(50,20));
        num->ShowIn(*tbox);
        if (panel) panel->Add(*tbox);
    }
}
Implementation of NumberPanel Class (Cont.)

    void NumberPanel::ShowIn( Panel& panel)
    { this->panel = &panel; }

    void NumberPanel::Show()
    {   if (last > -1 )
        for(int i = 0; i <= last; i++)
            number[i]->Show();
    }

    NumberPanel::~NumberPanel() {}
Example of NumberPanel Flexibility (Continued)

// manipulate individual counter objects
number.Next();
octal.Next();
length.Next(50);

// display all of the new values
numberPanel.Show();

Implicit vs. Explicit Type Casting

Explicit
Number *number;
DisplayableNumber *displayable =(DisplayableNumber*)number

Implicit
Number *number;
DisplayableNumber *displayable = number;

Avoid implicit typecasting.
Say what you mean in your code!
Implicit vs. Explicit Type Casting In Parameter Passing

Given the following method:
NumberPanel::Add(DisplayableNumber * dn);

Call with Explicit Type Cast
NumberPanel panel;
Number *n = new Number(100);
panel.Add((DisplayableNumber*)n);

Call with Implicit Type Cast
NumberPanel panel;
Number *c = new Number(100);
panel.Add(c);

Other developers may not realize that Add takes a DisplayableNumber.

Widening vs. Narrowing

* Widening
  * Type casting from a derived class to a base class.
  * **Always Safe!**
    * Can be checked by the compiler.

* Narrowing
  * Type casting from a base class to a derived class.
  * Safety depends on the programmer.
Widening/Narrowing Example

DisplayableNumber *DNptr;
Number *number = new Number(100);
Cycler *cycler;

DNptr = (DisplayableNumber*)number;  // safe; it widens

cycler = (Cycler*)DNptr;  // oops!

cycler->Next();  // who knows what this will do!