

## Chapter 6: Part 2

---

Generalization:  
Inheritance &  
Polymorphism

1

## Steps to Unlocking the Power of Inheritance

---

**Pure Virtual  
Methods**

**Virtual  
Methods**

**Type  
Casting**

2

# Type Casting

- Allows flexible structures to be built

- **Idea:**

Treat a derived class object as if it were a base class object

- **Syntax:**

This syntax cannot be used with pointers.

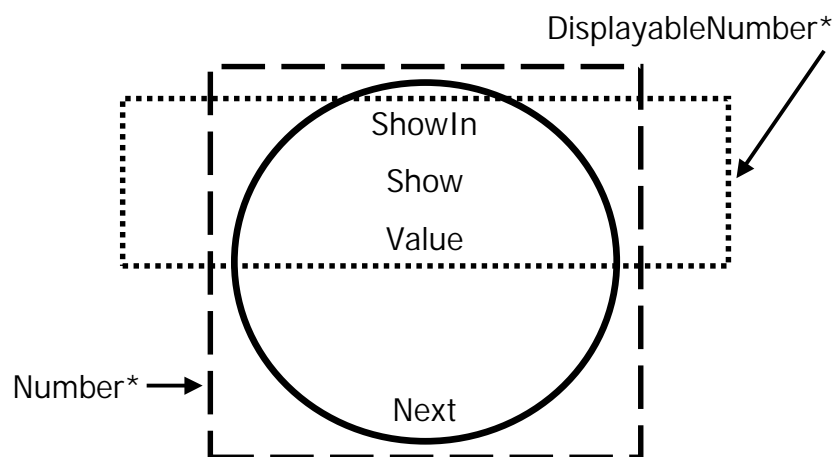
BaseClassName (derivedClassObject)

or

(BaseClassName) derivedClassObject

3

# Type Casting



4

## Type Casting Using Pointers to an Object

```
TextBox display(Location(100,100), Shape(75, 50));  
Number* number = new Number(100);  
DisplayableNumber* dnp;  
Number->Next();
```

Type Cast

```
dnp = (DisplayableNumber*) number;
```

```
dnp->ShowIn(display);  
dnp->Show();
```

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

5

## 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();
```

6

## Type Casting Errors

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

```
numPtr->Next();  
numRef.Next();
```

**Wrong!**

DisplayableNumber doesn't  
have a Next() method!

7

## Why Type Casting?

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

8

# 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!

9

# 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

10

## 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.

11

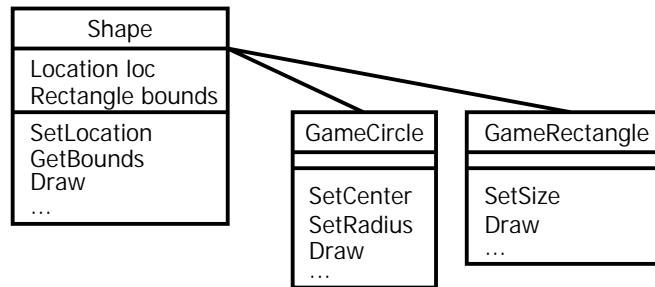
## Widening/Narrowing Example

```
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!
```

12

## Interface Sharing

### ■ Suppose



```

GameCircle * circ = new GameCircle(Location(100,100));
Shape * shape = (Shape*)circ;
shape.Draw(); // which Draw() is called?
  
```

13

## Revised Clock Class: Interface Sharing Example

```

class Clock {
private:
    Number* number; // Connect(ed)To a Number
    Cycler* cycler; // Connect(ed)To a Cycler
    ...
public:
    ...
    ConnectTo(Number& cnt);
    ConnectTo(Cyclor& cnt);
    ...
};
  
```

14

## Revised Clock Class: (Continued)

```
void Clock::ConnectTo(Number& cnt) { number = &cnt;};  
void Clock::ConnectTo(Cycler& cnt) { cycler = &cnt;};  
void Clock::Notify()  
{ if (number) number->Next();  
  else if (cyclier) cyclier->Next();  
}
```

15

## Repetitive Code in the Clock Class

```
class Clock {  
private:  
    Number*      number;      //Connect(ed)To a Counter  
    Cycler*      cyclier;     //Connect(ed)To a Cycler  
    BiCycler*    biCyclier;   //Connect(ed)To a BiCycler  
    UpCounter*   upCounter;   //Connect(ed)To a UpCounter  
    DownCounter* downCounter; //Connect(ed)To a DownCounter  
    BatchCounter* batchCounter; //Connect(ed)To a BatchCounter  
    JumpCounter* jumpCounter; //Connect(ed)To a JumpCounter  
    SwitchCounter* switchCounter; //Connect(ed)To a SwitchCounter  
    ...  
};
```

**YUCK!**

We need a better way!

16

## Declaring a Virtual Method

```
class DisplayableNumber {
    private:
        ...
    public:
        ...
        virtual void Next();    // virtual Next method
        ...
};

// in implementation file

void DisplayableNumber::Next() {} //default definition
```

17

## Redefining the Clock Class

```
class Clock
{
    private:
        DisplayableNumber *number; // only refer to base class

    public:
        ...
        void ConnectTo(DisplayableNumber* dn);
        void ConnectTo(DisplayableNumber& dn);
        void Notify();
};
```

18

## Redefining the Clock Class (Continued)

```
// in implementation file
void Clock::ConnectTo (DisplayableNumber* dn)
{ number = dn; }

void Clock::ConnectTo (DisplayableNumber& dn)
{ number = &dn; }

void Clock::Notify()
{ number->Next(); // invokes derived class method
  number->Show(); // invokes base class method
}
```

19

## Using the Revised Clock Class

```
Clock oneMinute(60*1000),
      oneSecond(1000);
Number minutes(0);
Cycler seconds(60);

oneMinute.ConnectTo( (DisplayableNumber&) minutes);
oneSecond.ConnectTo( (DisplayableNumber&) seconds);

oneMinute.Start();
oneSecond.Start();
```

20

## Example of Run-Time Binding

```
class Clock {
private:
    DisplayableNumber *number; // only refer to base class
public:
    void Notify();
};
void Clock::Notify() {
    number->Next();    // invokes derived class method
    ...
}
```

*Binding of Next() in Notify() is done at compile-time by an invisible pointer since actual type isn't known until execution!*

21

## Binding

- Selection of code for a function invocation
- Static, compile time
  - efficient
- Dynamic, run time
  - flexible
  - default in many o-o languages
    - Java, Smalltalk
  - Must be explicitly programmed in C++

22

## Pure Virtual Methods

### Definition of “Abstract” Base Class

```
class AbstractDisplayableNumber {  
    ...  
    public:  
        virtual void Next() = 0; // pure virtual method  
    ...  
};
```

### Definition of a “Concrete” Derived Class

```
class ConcreteCounter : public AbstractDisplayableNumber {  
    ...  
};  
void ConcreteCounter::Next() {...some implementation ... }
```

23

## Examples of Using Abstract Classes

```
AbstractDisplayableNumber *adn; // ok - pointer to abstract  
                                // class
```

```
ConcreteCounter*cc; // ok - pointer to concrete class
```

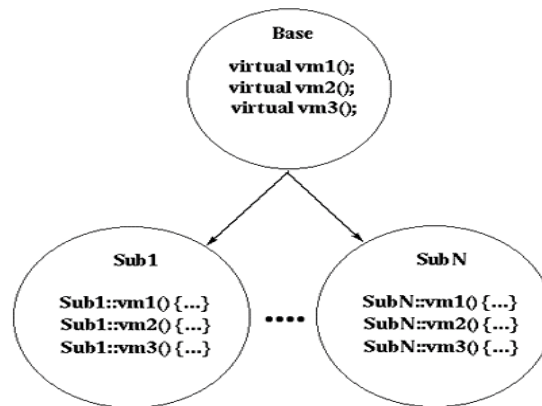
```
// NO! - class is abstract  
AbstractDisplayableNumber n(100);
```

```
ConcreteCounter c(100); // ok - instance of concrete class
```

```
cc = &c; // ok - types are the same  
adn = &c; // ok  
adn->Next(); // ok
```

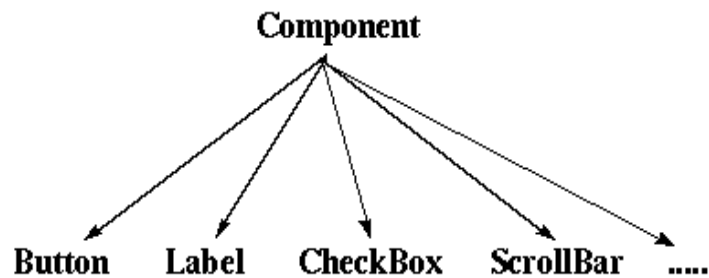
24

## Class Structure for Polymorphism



25

## Typical Component Class Hierarchy



26

## Example from Project 1

- No default Draw() for Shape class
- Want ShapeManager to call Draw() on its Shape objects.
- What do we do?

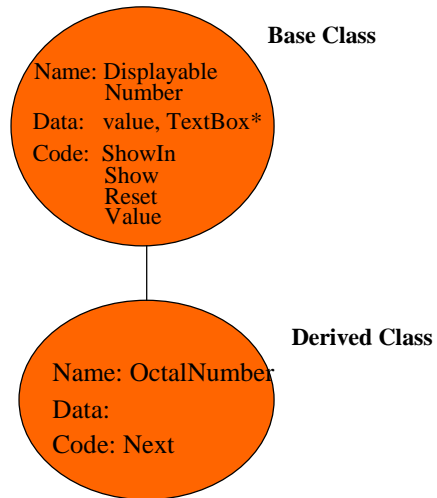
27

## Use of Polymorphism: Component Example

```
class ComponentManager {
private:
    Component *member[10];
    int    number;
    ...
public:
    ComponentManager() : number(0) {}
    void Add(Component *cmp) { member[number++] = cmp; }
    Component* Inside(int x, int y)
        {for(int next = 0; next<number; next++)
          if (member[next]->inside(x,y) )
            return member[next];
          return NULL;
        }
    ...
};
```

28

## What happens if I don't get the base class right the first time?



29

## Problem

**Problem:** The representation of the octal number in the TextBox should appear in an octal number representation: preceded by a zero and without the digits 8 or 9.

**Solution:** Override the base class method **Show**

**Implication:** The TextBox variable must be moved to the protected section

### **Design decisions revealed:**

- The DisplayableNumber's value is displayed in a TextBox,
- The TextBox is accessed via a pointer,
- The name of the pointer is textBox, and
- The TextBox has a SetText method.

30

## Solution

### Solution:

Factor the base class to separate:

Formatting the character string to be displayed, and

Displaying the string in the TextBox.

31

## Refactoring the DisplayableNumber Class

```
class DisplayableNumber {
private:
    TextBox* textBox;
protected:
    int value;
    virtual char* Format(); // produce string to display
    virtual int Parse(char* input); // convert user
                                   // input to value
public:
    DisplayableNumber(int initValue = 0);
    void ShowIn(TextBox& p);
    void Show();
    void Reset();
    ~DisplayableNumber();
};
```

32

## Refactoring the DisplayableNumber Class

```
char* DisplayableNumber::Format()
{ char* asString = new char[10];
  ostream format(asString);
  format << value;          // use decimal formatting
  return asString;
}

void DisplayableNumber::Show()
{ if (textBox) textBox->SetText(Format());
}
```

33

## Refactoring the DisplayableNumber Class

```
int DisplayableNumber::Parse(char* input)
{ int decimalValue;
  istream format(input);
  format >> decimalValue;    // use decimal formatting
  return decimalValue;
}

void DisplayableNumber::Reset()
{
  if (textBox) value = Parse(textBox->GetText());
}
```

34

## Defining the Octal Number Class

```
class OctalNumber : public Number {
protected:
    char* Format();           // how to print an octal
                             // number
    int Parse(char* input) // how to read an octal
                             // number

public:
    OctalNumber(int init);
    ~OctalNumber();
};
```

35

## Defining the Octal Number Class

```
OctalNumber::OctalNumber (int init) : Number(init) {}

char* OctalNumber::Format()
{ char* asString = new char[10];
  ostream format(asString);
  format << oct << value; // format value as octal
  return asString;
}

int OctalNumber::Parse(char* input)
{ int octalValue;
  istream is(input);
  is.flags(ios::oct); // set stream to read as octal
  is >> octalValue;
  return octalValue;
}

OctalNumber::~~OctalNumber() {}
```

36