

## Lafore: Object-Oriented Programming in C++ (4<sup>th</sup>) Chapter 6: Classes

```
// 1-SmallObject.cpp -- demonstrates a small, simple object
#include <iostream>
using namespace std;

////////////////////////////////////
class SmallObject { // Specify a class (no memory allocated)
private:
    int nSomeData; // Class data
public:
    void SetData(int n) // Member function to set data
    {
        nSomeData = n;
    }
    void ShowData() // Member function to display data
    {
        cout << "Data is " << nSomeData << endl;
    }
};

////////////////////////////////////
void main()
{
    SmallObject s1, s2; // Define two objects of class SmallObject
                        // (actual memory now allocated)
    s1.SetData(1066); // Call member function to set data of object s1
    s2.SetData(1776); // Call member function to set data of object s2
    s1.ShowData(); // Call member function to display data of object s1
    s2.ShowData(); // Call member function to display data of object s2
}
```

```
// 2-objpart.cpp -- widget Part as a class
#include <iostream>
using namespace std;

////////////////////////////////////
class Part { // Specify new data type -- to create objects later
private:
    int nModelNumber; // ID number of widget
    int nPartNumber; // ID number of widget Part
    float fCost; // Cost of Part
public:
    void SetPart(int nM, int nP, float fC) // Set data
    {
        nModelNumber = nM;
        if (nP < 1000 || nP > 100000)
            cout << "Error";
        else
            nPartNumber = nP;
        fCost = fC;
    }
    void ShowPart() // Display data
    {
        cout << "Model " << nModelNumber;
        cout << ", Part " << nPartNumber;
        cout << ", costs $" << fCost << endl;
    }
};

////////////////////////////////////
void main()
{
    Part part1; // Define object of class Part
    part1.SetPart(6244, 373, 217.55F); // Call member function
    part1.ShowPart(); // Call member function
}
```

```

// 3-circles.cpp -- Create Circle objects
#include "msoftcon.h" // for graphics functions
////////////////////
class Circle {
private: // Data members hold current state of each object
    int nCentreX, nCentreY; // Coordinates of center
    int nRadius; // Radius of Circle
    color nFillColor; // Color
    fstyle nFillStyle; // Fill pattern
public: // Member functions implement the behaviour of objects
    void Set(int nX, int nY, int nR, color nFC, fstyle nFS)
    { // Sets Circle attributes
        nCentreX = nX; nCentreY = nY;
        nRadius = nR;
        nFillColor = nFC;
        nFillStyle = nFS;
    }
    void Draw() // Draws the circle, using the object's data members
    {
        SetColor(nFillColor); // Set color
        SetFillStyle(nFillStyle); // Set fill
        DrawCircle(nCentreX, nCentreY, nRadius); // Draw solid circle
    }
};
////////////////////
void main()
{
    InitializeGraphics(); // Initialize graphics system
    Circle c1, c2, c3; // Declare, define
    c1.Set(15, 7, 5, cBLUE, X_FILL); // Initialize
    c2.Set(41, 12, 7, cRED, O_FILL); // Initialize
    c3.Set(65, 18, 4, cGREEN, MEDIUM_FILL); // Initialize
    c1.Draw(); c2.Draw(); c3.Draw();
    SetCursorPosition(1, 25); // Lower left corner on a standard console screen
}

```

Latore: Object-Oriented Programming In C++ (4<sup>th</sup>)

Chapter 6: Classes

```

// 4-englobj.cpp -- objects using English measurements
#include <iostream>
using namespace std;
////////////////////
class Distance { // English Distance class
private:
    int nFeet;
    float finches;
public:
    void Set(int nFt, float fin) // Overloaded Set -- argument values
    {
        nFeet = nFt;
        finches = fin;
    }
    void Set() // Overloaded Set -- uses keyboard to set values
    {
        cout << "\nEnter nFeet: "; cin >> nFeet;
        cout << "Enter finches: "; cin >> finches;
    }
    void Display()
    {
        cout << nFeet << "ft" << finches << "in";
    }
};
////////////////////
void main()
{
    Distance dist1, dist2; // Define two length objects
    dist1.Set(11, 6.25); // Calls 2-argument version of Set()
    dist2.Set(); // Calls keyboard version of Set()
    cout << "\ndist1 = "; dist1.Display();
    cout << "\ndist2 = "; dist2.Display();
    cout << endl;
}

```

Latore: Object-Oriented Programming In C++ (4<sup>th</sup>)

Chapter 6: Classes

```

// 5-counter.cpp -- object represents a counter variable
#include <iostream>
using namespace std;
////////////////////////////////////
class Counter {
private:
    unsigned int unCount;
public:
    Counter() : unCount(0)           // Constructor
    { /*empty body*/ }
    void inc_Counter()              // Increment unCount
    { unCount++; }
    int Get()                        // Return copy of value in unCount
    { return unCount; }
};
////////////////////////////////////
void main()
{
    Counter c1, c2;                // Declare, define (and initialize in constructor)
    cout << "\nc1=" << c1.Get(); // Display before increment
    cout << "\nc2=" << c2.Get();
    c1.inc_Counter();              // Increment c1
    c2.inc_Counter();              // Increment c2
    c2.inc_Counter();
    cout << "\nc1=" << c1.Get(); // Display after increment
    cout << "\nc2=" << c2.Get();
    cout << endl;
}

```

```

// 6-circor.cpp -- Circle objects use constructor for initialization
#include "msoftcon.h" // for graphics functions
////////////////////////////////////
class Circle {
private: // Data members hold current object state (location, size, color, etc.)
    int nCentreX, nCentreY; // Coordinates of center
    int nRadius;           // Radius of Circle
    color nFillColor;     // Color
    fstyle nFillStyle;     // Fill pattern
public: // Member functions implement the behaviour of objects
    Circle(int nX, int nY, int nR, color nFC, fstyle nFS) // Sets Circle attributes
    {
        nCentreX = nX; nCentreY = nY;
        nRadius = nR;
        nFillColor = nFC;
        nFillStyle = nFS;
    }
    void Draw() // Draws the Circle, using object's data members
    {
        SetColor(nFillColor); // Set color
        SetFillStyle(nFillStyle); // Set fill
        DrawCircle(nCentreX, nCentreY, nRadius); // Draw solid Circle
    }
};
////////////////////////////////////
void main()
{
    InitializeGraphics(); // Initialize graphics system
    Circle c1(15, 7, 5, cBLUE, X_FILL); // Declare, define, initialize
    Circle c2(41, 12, 7, cRED, O_FILL); // Declare, define, initialize
    Circle c3(65, 18, 4, cGREEN, MEDIUM_FILL); // Declare, define, initialize
    c1.Draw(); // Draw circles
    c2.Draw();
    c3.Draw();
    SetCursorPosition(1, 25); // Lower left corner on a standard console screen
}
// 7-englcon.cpp -- constructors, adds objects using member function

```

```

#include <iostream>
using namespace std;
////////////////////////////////////
class Distance { // English Distance class
private:
    int nFeet;
    float finches;
public:
    Distance() : nFeet(0), finches(0.0f) {} // Constructor (no args)
    Distance(int nFt, float fin) : nFeet(nFt), finches(fin) {} // Constructor (two args)

    void Set() // set length based on user input
    {
        cout << "\nEnter nFeet: "; cin >> nFeet;
        cout << "Enter finches: "; cin >> finches;
    }

    void Display()
    { cout << nFeet << "\n" << finches << "\n"; }

    void AddDistance(Distance d1, Distance d2); // Function Declaration only
};

//-----
void Distance::AddDistance(Distance d1, Distance d2) // Function Definition
{
    finches = d1.finches + d2.finches;
    nFeet = 0;
    if (finches >= 12.0f) {
        finches -= 12.0f;
        nFeet++;
    }
    nFeet += d1.nFeet + d2.nFeet;
}

void main()
{
    Distance dist1;
    dist1.Set();
    Distance dist2(11, 6.25f);
    Distance dist3;
    dist3.AddDistance(dist1, dist2);
    cout << "\ndist1 = "; dist1.Display();
    cout << "\ndist2 = "; dist2.Display();
    cout << "\ndist3 = "; dist3.Display();
    cout << endl;
}

```

Latore: Object-Oriented Programming In C++ (4<sup>th</sup>)

```

// 8-ecopycon.cpp -- initialize objects using default copy constructor
#include <iostream>
using namespace std;
////////////////////////////////////
class Distance { // English Distance class
private:
    int nFeet;
    float finches;
public:
    Distance() : nFeet(0), finches(0.0f) {} // Constructor (no args)
    // Note: no one-arg constructor
    Distance(int ft, float in) : nFeet(ft), finches(in) {} // Constructor (two args)

    void Set()
    {
        cout << "\nEnter nFeet: "; cin >> nFeet;
        cout << "Enter finches: "; cin >> finches;
    }

    void Display()
    { cout << nFeet << "\n" << finches << "\n"; }
};

//-----
void main()
{
    Distance dist1(11, 6.25f); // two-arg constructor
    Distance dist2(dist1); // one-arg constructor
    Distance dist3 = dist1; // also one-arg constructor
    cout << "\ndist1 = "; dist1.Display();
    cout << "\ndist2 = "; dist2.Display();
    cout << "\ndist3 = "; dist3.Display();
    cout << endl;
}

```

Latore: Object-Oriented Programming In C++ (4<sup>th</sup>)

```

// 9-englret.cpp -- function returns value of type Distance
#include <iostream>
using namespace std;
////////////////////////////////////
class Distance { // English Distance class
private:
    int nFeet;
    float finches;
public:
    Distance() : nFeet(0), finches(0.0f) {} // Constructor (no args)
    Distance(int nFt, float fln) : nFeet(nFt), finches(fln) {} // (2 args)
    void Set()
    {
        cout << "\nEnter nFeet: "; cin >> nFeet;
        cout << "Enter finches: "; cin >> finches;
    }
    void Display()
    { cout << nFeet << "\'\'" << finches << "\'\'"; }
    Distance Add(Distance d2);
};
//-----
Distance Distance::Add(Distance d2)
{
    Distance dTemp; // Temporary variable--only exists in this function
    dTemp.fInches = finches + d2.fInches;
    if (dTemp.fInches >= 12.0f) {
        dTemp.fInches -= 12.0f;
        dTemp.nFeet = 1;
    }
    dTemp.nFeet += nFeet + d2.nFeet;
    return dTemp; // return values stored in dTemp
}
}

void main()
{
    Distance dist1, dist3;
    Distance dist2(11, 6.25f);
    dist1.Set();
    dist3 = dist1.Add(dist2);
    cout << "\ndist1 = "; dist1.Display();
    cout << "\ndist2 = "; dist2.Display();
    cout << "\ndist3 = "; dist3.Display();
    cout << endl;
}

```

Latore: Object-Oriented Programming In C++ (4<sup>th</sup>)

```

// 10-cardobj.cpp -- cards as objects
#include <iostream>
using namespace std;
const int jack = 11; // from 2 to 10 are
const int queen = 12; // integers without names
const int king = 13;
const int ace = 14;
enum Suit { clubs, diamonds, hearts, spades };
////////////////////////////////////
class Card {
private:
    int nNumber; // 2 to 10, jack, queen, king, ace
    Suit sSuit; // clubs, diamonds, hearts, spades
public:
    Card () {}
    Card (int n, Suit s) : nNumber(n), sSuit(s) {}
    void Display(); // Display Card
    bool isEqual(Card); // Is it the same as another Card object?
};
//-----
void Card::Display() // Display the Card
{
    if (nNumber >= 2 && nNumber <= 10)
        cout << nNumber << " of ";
    else {
        switch(nNumber) {
            case jack: cout << "jack of "; break;
            case queen: cout << "queen of "; break;
            case king: cout << "king of "; break;
            case ace: cout << "ace of "; break;
        }
    }
    switch(sSuit) {
        case clubs: cout << "clubs"; break;
        case diamonds: cout << "diamonds"; break;
        case hearts: cout << "hearts"; break;
        case spades: cout << "spades"; break;
    }
}

bool Card::isEqual(Card c2) // return true if cards equal
{ return (nNumber==c2.nNumber && sSuit==c2.sSuit); }
}
}

```

Latore: Object-Oriented Programming In C++ (4<sup>th</sup>)

```
void main()
{
    Card cTemp, cChosen, cPrize; // Define various cards
    Card cCard1 ( 7, clubs ); // Define & initialize cCard1
    cout << "\nCard 1 is the ";
    cCard1.Display(); // Display cCard1
    Card cCard2( jack, hearts ); // Define & initialize cCard2
    cout << "\nCard 2 is the ";
    cCard2.Display(); // Display cCard2
    Card cCard3( ace, spades ); // Define & initialize cCard3
    cout << "\nCard 3 is the ";
    cCard3.Display(); // Display cCard3
    cPrize = cCard3; // cPrize is the Card to guess
    cout << "\nI'm swapping Card 1 and Card 3";
    cTemp = cCard3; cCard3 = cCard1; cCard1 = cTemp;
    cout << "\nI'm swapping Card 2 and Card 3";
    cTemp = cCard3; cCard3 = cCard2; cCard2 = cTemp;
    cout << "\nI'm swapping Card 1 and Card 2";
    cTemp = cCard2; cCard2 = cCard1; cCard1 = cTemp;

    int nPosition;
    cout << "\nNow, where (1, 2, or 3) is the ";
    cPrize.Display(); // Display cPrize Card
    cout << "?";
    cin >> nPosition; // Input user's guess of nPosition
    switch (nPosition) { // Set cChosen to user's choice
        case 1: cChosen = cCard1; break;
        case 2: cChosen = cCard2; break;
        case 3: cChosen = cCard3; break;
    }
    if (cChosen.IsEqual(cPrize)) // is cChosen Card the cPrize?
        cout << "That's right! You win!";
    else
        cout << "Sorry. You lose.";
    cout << "\n You chose the "; cChosen.Display(); cout << endl;
}
}
```

```
// 11-Statdata.cpp -- static class data
#include <iostream>
using namespace std;
////////////////////////////////////
class foo {
private:
    static int nCount; // only one data item for all objects
public:
    //note: *declaration* only!
    foo() // increments nCount when object created
    { nCount++; }
    int getCount() // returns nCount
    { return nCount; }
};

//-----
int foo::nCount = 0; // *definition* of nCount
////////////////////////////////////
void main()
{
    foo f1, f2, f3; // Create three objects

    cout << "\nCount is " << f1.getCount() << endl; //each object
    cout << "\nCount is " << f2.getCount() << endl; //sees the
    cout << "\nCount is " << f3.getCount() << endl; //same value
}
}
```

```
// 12-constfu.cpp -- demonstrates const member functions
class aClass {
private:
    int nAlpha;

public:
    void nonFunc()           // non-const member function
    { nAlpha = 99; }        // OK
    void confunc() const    // const member function
    { nAlpha = 99; }        // error: can't modify a member
};
```

```
// 13-engConst.cpp -- using const
#include <iostream>
using namespace std;
////////////////////////////////////
class Distance {
private:
    int nFeet;
    float finches;

public:
    Distance() : nFeet(0), finches(0.0f) { }
    Distance(int nFt, float fln) : nFeet(nFt), finches(fln) { }
    void Set()
    {
        cout << "\nEnter nFeet: "; cin >> nFeet;
        cout << "Enter finches: "; cin >> finches;
    }
    void Display() const
    { cout << nFeet << "\n" << finches << "\n"; }
};

Distance add_dist(const Distance& rdOperand) const;
//-----
Distance Distance::add_dist(const Distance& rdOperand) const
{
    Distance dTmp;           // Temporary variable
    // nFeet = 0;           // ERROR: can't modify
    // rdOperand.nFeet = 0; // ERROR: can't modify rdOperand
    dTmp.finches = finches + rdOperand.finches;
    if (dTmp.finches >= 12.0f) {
        dTmp.finches -= 12.0f;
        dTmp.nFeet = 1;
    }
    dTmp.nFeet += nFeet + rdOperand.nFeet;
    return dTmp;
}
// 14-constObj.cpp -- constant Distance objects
```

```
void main()
{
    Distance dist1, dist3;
    Distance dist2(11, 6.25);
    dist1.Set();
    dist3 = dist1.add_dist(dist2);
    cout << "\ndist1 = ", dist1.Display();
    cout << "\ndist2 = ", dist2.Display();
    cout << "\ndist3 = ", dist3.Display();
}
```

```
#include <iostream>
using namespace std;

////////////////////////////////////

class Distance {
private:
    int nFeet;
    float finches;

public:
    // 2-arg constructor
    Distance(int ft, float in) : nFeet(ft), finches(in) { }
    void Set() // User input; non-const func
    {
        cout << "\nEnter nFeet: "; cin >> nFeet;
        cout << "Enter finches: "; cin >> finches;
    }
    void Display() const // Display distance; const func
    { cout << nFeet << "\n" << finches << "\n"; }
};

////////////////////////////////////

void main()
{
    const Distance football(300, 0.0f);

    // football.Set(); // Error: Set() not const
    cout << "football = ";
    football.Display(); // OK
    cout << endl;
}
```