

EffectiveCPP.cpp

```

1  ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2  // Demo: CPP-02.09D - Effective C++ (Maximum Call Version) //
3  ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
4
5  #include <iostream>
6  using namespace std;
7
8
9
10 class CPoint
11 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
12 {
13 private:
14     static int iCtorCount;
15     static int iDtorCount;
16     static int iCopyCount;
17     static int iAssignCount;
18
19 public:
20     int iX;
21     int iY;
22
23 public:
24     CPoint() : iX(0), iY(0) {iCtorCount ++;}
25     CPoint(int x, int y) : iX(x), iY(y) {iCtorCount++;}
26     CPoint(const CPoint& Source);
27     CPoint& operator=(const CPoint& Source);
28     ~CPoint() {iDtorCount++;}
29
30     static void listCount(void)
31     {
32         cout << "--> Call Count: "
33              << iCtorCount + iDtorCount + iCopyCount + iAssignCount
34              << " (Ctor: " << iCtorCount
35              << ", Dtor: " << iDtorCount
36              << ", Copy: " << iCopyCount
37              << ", Assign: " << iAssignCount << ")" << endl;
38     }
39 };
40 // class: CPoint ////////////////////////////////////////////////////////////////////
41
42
43
44
45 // define and init static data members ////////////////////////////////////////////////////////////////////
46 int CPoint::iCtorCount = 0;
47 int CPoint::iCopyCount = 0;
48 int CPoint::iDtorCount = 0;
49 int CPoint::iAssignCount = 0;
50
51
52
53 // point copy constructor ////////////////////////////////////////////////////////////////////
54 CPoint::CPoint(const CPoint& Source)
55 {
56     *this = Source;
57     iCopyCount++;
58 }
59 ////////////////////////////////////////////////////////////////////
60
61
62
63 // point assignment operator ////////////////////////////////////////////////////////////////////
64 CPoint& CPoint::operator=(const CPoint& Source)
65 {
66     iX = Source.iX;
67     iY = Source.iY;
68     iAssignCount++;
69     return *this;
70 }
71 ////////////////////////////////////////////////////////////////////
72

```

```

73
74
75 class CLine
76 ///////////////////////////////////////////////////
77 {
78 private:
79     CPoint P1;
80     CPoint P2;
81
82 public:
83     CLine(const CPoint p1, const CPoint p2);
84     void List(void);
85 };
86 // class: CLine //////////////////////////////////////
87
88
89
90 CLine::CLine(const CPoint p1, const CPoint p2)
91 ///////////////////////////////////////////////////
92 {
93     P1 = p1;
94     P2 = p2;
95 }
96 ///////////////////////////////////////////////////
97
98
99
100 void CLine::List(void)
101 ///////////////////////////////////////////////////
102 {
103     cout << "Line Object:" << endl;
104     cout << " P1: iX = " << P1.iX << "   iY = " << P1.iY << endl;
105     cout << " P2: iX = " << P2.iX << "   iY = " << P2.iY << endl << endl;
106 }
107 ///////////////////////////////////////////////////
108
109
110
111 int main(void)
112 ///////////////////////////////////////////////////
113 {
114     // show initial call count
115     CPoint::listCount();
116     cout << endl;
117
118     // definitions
119     CLine* pL1;
120     CPoint P1;
121     CPoint P2(30, 30);
122
123     // dynamic instantiation of a line object
124     pL1 = new CLine(P1, P2);
125
126     // list values of line
127     pL1->List();
128
129     // show final call count
130     CPoint::listCount();
131     cout << endl;
132
133     // delete dynamic line object
134     delete pL1;
135
136     return 0;
137 }
138

```