

C++

# Ders 6

---

## Çoklu Benzeşim

## C++

# Çoklubenzetim

- Farklı sınıflardaki nesnelerin aynı fonksiyon çağrılmasına farklı şekilde cevap verebilmesini sağlar. Diğer bir deyişle benzer durumlara genel bir arabirimin uygulanma işlemidir. Böylece, “bir arabirim çok metod” felsefesini tamamlar.
- Sanal fonksiyonlar aracılığıyla uygulanır.
  - Kök-sınıftan bir pointer (veya referans) bir sanal fonksiyon çağırır
  - C++ nesnedeki doğru tekrar tanımlanmış fonksiyonu seçer
- Eğer birden fazla sınıfta sanal olmayan üye fonksiyon tanımlanmış ve kök-sınıf pointer aracılığıyla çağrılmışsa, o zaman kök-sınıf fonksiyonu çağrılır.
  - Eğer türetilmiş sınıf pointer’ ı tarafından çağrılmışsa türetilmiş sınıftaki fonksiyon kullanılır
- print fonksiyonunun sanal fonksiyon olmadığını düşünün

```
Employee e, *ePtr = &e;
HourlyWorker h, *hPtr = &h;
ePtr->print(); // base-sınıf print fonksiyonu
hPtr->print(); // derived-sınıf print fonksiyonu
ePtr=&h; // dönüşüm mümkündür
ePtr->print(); // ama hala base-sınıf print fonksiyonunu çağırır
```

Employee      Kök sınıfını,  
HourlyWorker    Türetilmiş-sınıfı göstermektedir. Kök-sınıfındaki print() türetilmiş sınıflarada açıktır, fakat bunu açık bir şekilde yapmak gerekir, şöyleki;  
hPtr -> Employee :: print ();

C++

# Örnek (Çoklubenzetim)

```
1 // Fig. 10.1: employ2.h
2 // Abstract base class Employee
3 #ifndef EMPLOY2_H
4 #define EMPLOY2_H
5
6 class Employee {
7 public:
8     Employee( const char *, const char * )
9     ~Employee();    // destructor reclaims
10    const char *getFirstName() const;
11    const char *getLastName() const;
12
13    // Pure virtual function makes Employee
14    virtual double earnings() const = 0;    // pure virtual
```

**earnings** saf(pure) ve **virtual** belirtiliyor, çünkü uygulama **earnings** fonksiyonun hangi türetilmiş sınıfta kullanılacağına bağlı

**Employee** abstract base sınıf

C++

# Örnek (devamı)

---

```
15     virtual void print() const;           // virtual
16 private:
17     char *firstName;
18     char *lastName;
19 };
20
21 #endif
22 // Fig. 10.1: employ2.cpp
23 // Member function definitions for
24 // abstract base class Employee.
25 // Note: No definitions given for pure virtual functions.
26 #include <iostream>
27
28 using std::cout;
29
30 #include <cstring>
31 #include <cassert>
32 #include "employ2.h"
```

C++

# Örnek (devamı)

---

```
33
34 // Constructor dynamically allocates space for the
35 // first and last name and uses strcpy to copy
36 // the first and last names into the object.
37 Employee::Employee( const char *first, const char *last )
38 {
39     firstName = new char[ strlen( first ) + 1 ];
40     assert( firstName != 0 );    // test that new worked
41     strcpy( firstName, first );
42
43     lastName = new char[ strlen( last ) + 1 ];
44     assert( lastName != 0 );    // test that new worked
45     strcpy( lastName, last );
46 }
47
48 // Destructor deallocates dynamically allocated memory
49 Employee::~Employee()
50 {
51     delete [] firstName;
52     delete [] lastName;
53 }
```

C++

# Örnek (devamı)

---

```
54
55 // Return a pointer to the first name
56 // Const return type prevents caller from modifying private
57 // data. Caller should copy returned string before destructor
58 // deletes dynamic storage to prevent undefined pointer.
59 const char *Employee::getFirstName() const
60 {
61     return firstName;    // caller must delete memory
62 }
63
64 // Return a pointer to the last name
65 // Const return type prevents caller from modifying private
66 // data. Caller should copy returned string before destructor
67 // deletes dynamic storage to prevent undefined pointer.
```

C++

# Örnek (devamı)

---

```
68 const char *Employee::getLastName() const
69 {
70     return lastName;    // caller must delete memory
71 }
72
73 // Print the name of the Employee
74 void Employee::print() const
75     { cout << firstName << ' ' << lastName; }
76 // Fig. 10.1: boss1.h
77 // Boss class derived from Employee
78 #ifndef BOSS1_H
79 #define BOSS1_H
80 #include "employ2.h"
81
```

C++

# Örnek (devamı)

---

```
82 class Boss : public Employee {  
  
83 public:  
  
84     Boss( const char *, const char *, double = 0.0 );  
  
85     void setWeeklySalary( double );  
  
86     virtual double earnings() const;  
  
87     virtual void print() const;  
  
88 private:  
  
89     double weeklySalary;  
  
90 };  
  
91  
  
92 #endif
```



C++

# Örnek (devamı)

---

```
93 // Fig. 10.1: boss1.cpp
94 // Member function definitions for class Boss
95 #include <iostream>
96
97 using std::cout;
98
99 #include "boss1.h"
100
101 // Constructor function for class Boss
102 Boss::Boss( const char *first, const char *last, double s )
103     : Employee( first, last ) // call base-class constructor
104 { setWeeklySalary( s ); }
105
106 // Set the Boss's salary
107 void Boss::setWeeklySalary( double s )
108     { weeklySalary = s > 0 ? s : 0; }
109
```

C++

# Örnek (devamı)

```

110// Get the Boss's pay
111double Boss::earnings() const { return weeklySalary; }
112
113// Print the Boss's name
114void Boss::print() const
115{
116    cout << "\n                Boss: ";
117    Employee::print();
118}
119// Fig. 10.1: commis1.h
120// CommissionWorker class derived from Employee
121#ifndef COMMIS1_H
122#define COMMIS1_H
123#include "employ2.h"
124
125class CommissionWorker : public Employee {

```

Tekrar yazılmış  
(overriden) **earnings**  
ve **print** fonksiyonları  
kök sınıfta **virtual**  
olarak bildirildiler.

C++

# Örnek (devamı)

---

```
126public:
127    CommissionWorker( const char *, const char *,
128                      double = 0.0, double = 0.0,
129                      int = 0 );
130    void setSalary( double );
131    void setCommission( double );
132    void setQuantity( int );
133    virtual double earnings() const;
134    virtual void print() const;
135private:
136    double salary;           // base salary per week
137    double commission;       // amount per item sold
138    int quantity;           // total items sold for week
139};
140
141#endif
```

C++

# Örnek (devamı)

---

```
142// Fig. 10.1: commis1.cpp
143// Member function definitions for class CommissionWorker
144#include <iostream>
145
146using std::cout;
147
148#include "commis1.h"
149
150// Constructor for class CommissionWorker
151CommissionWorker::CommissionWorker( const char *first,
152    const char *last, double s, double c, int q )
153    : Employee( first, last ) // call base-class constructor
154{
155    setSalary( s );
```

C++

# Örnek (devamı)

---

```
156   setCommission( c );
157   setQuantity( q );
158}
159
160// Set CommissionWorker's weekly base salary
161void CommissionWorker::setSalary( double s )
162    { salary = s > 0 ? s : 0; }
163
164// Set CommissionWorker's commission
165void CommissionWorker::setCommission( double c )
166    { commission = c > 0 ? c : 0; }
167
168// Set CommissionWorker's quantity sold
169void CommissionWorker::setQuantity( int q )
```

C++

# Örnek (devamı)

```
170    { quantity = q > 0 ? q : 0; }
171
172// Determine CommissionWorker's earnings
173double CommissionWorker::earnings() const
174    { return salary + commission * quantity; }
175
176// Print the CommissionWorker's name
177void CommissionWorker::print() const
178{
179    cout << "\nCommission worker: ";
180    Employee::print();
181}
```

Tekrar yazılmış **earnings** ve **print** fonksiyonları kök sınıfta **virtual** olarak bildirildiler.

C++

# Örnek (devamı)

---

```
182// Fig. 10.1: piece1.h
183// PieceWorker class derived from Employee
184#ifndef PIECE1_H
185#define PIECE1_H
186#include "employ2.h"
187
188class PieceWorker : public Employee {
189public:
190    PieceWorker( const char *, const char *,
191                double = 0.0, int = 0 );
192    void setWage( double );
193    void setQuantity( int );
194    virtual double earnings() const;
```

C++

# Örnek (devamı)

---

```
195     virtual void print() const;
196 private:
197     double wagePerPiece; // wage for each piece output
198     int quantity;        // output for week
199 };
200
201 #endif
202 // Fig. 10.1: piece1.cpp
203 // Member function definitions for class PieceWorker
204 #include <iostream>
205
206 using std::cout;
207
208 #include "piece1.h"
209
210 // Constructor for class PieceWorker
211 PieceWorker::PieceWorker( const char *first, const char *last,
212                          double w, int q )
```



C++

# Örnek (devamı)

```

213 : Employee( first, last ) // call base-class constructor
214 {
215     setWage( w );
216     setQuantity( q );
217 }
218
219 // Set the wage
220 void PieceWorker::setWage( double w )
221 { wagePerPiece = w > 0 ? w : 0; }
222
223 // Set the number of items output
224 void PieceWorker::setQuantity( int q )
225 { quantity = q > 0 ? q : 0; }
226
227 // Determine the PieceWorker's earnings
228 double PieceWorker::earnings() const
229 { return quantity * wagePerPiece; }
230
231 // Print the PieceWorker's name
232 void PieceWorker::print() const
233 {
234     cout << "\n      Piece worker: ";
235     Employee::print();
236 }

```

Bir kez daha tekrar yazılmış **earnings** ve **print** fonksiyonları kök sınıfta **virtual** olarak bildirildiler.

C++

# Örnek (devamı)

---

```
237// Fig. 10.1: hourly1.h
238// Definition of class HourlyWorker
239#ifndef HOURLY1_H
240#define HOURLY1_H
241#include "employ2.h"
242
243class HourlyWorker : public Employee {
244public:
245    HourlyWorker( const char *, const char *,
246                double = 0.0, double = 0.0);
247    void setWage( double );
248    void setHours( double );
249    virtual double earnings() const;
```

C++

# Örnek (devamı)

---

```
250     virtual void print() const;
251 private:
252     double wage;    // wage per hour
253     double hours;   // hours worked for week
254 };
255
256 #endif
257 // Fig. 10.1: hourly1.cpp
258 // Member function definitions for class HourlyWorker
259 #include <iostream>
260
261 using std::cout;
262
263 #include "hourly1.h"
```

C++

# Örnek (devamı)

---

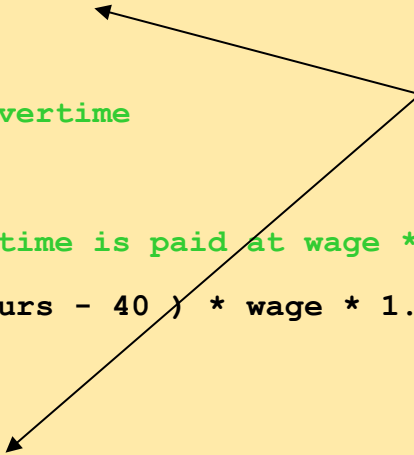
```
264
265// Constructor for class HourlyWorker
266HourlyWorker::HourlyWorker( const char *first,
267                             const char *last,
268                             double w, double h )
269    : Employee( first, last )    // call base-class constructor
270{
271    setWage( w );
272    setHours( h );
273}
274
275// Set the wage
276void HourlyWorker::setWage( double w )
277    { wage = w > 0 ? w : 0; }
```

C++

# Örnek (devamı)

```
278
279// Set the hours worked
280void HourlyWorker::setHours( double h )
281    { hours = h >= 0 && h < 168 ? h : 0; }
282
283// Get the HourlyWorker's pay
284double HourlyWorker::earnings() const
285{
286    if ( hours <= 40 ) // no overtime
287        return wage * hours;
288    else                // overtime is paid at wage * 1.5
289        return 40 * wage + ( hours - 40 ) * wage * 1.5;
290}
291
292// Print the HourlyWorker's name
```

Overridden fonksiyonlar



C++

# Örnek (devamı)

---

```
293 void HourlyWorker::print() const
294 {
295     cout << "\n    Hourly worker: ";
296     Employee::print();
297 }
298 // Fig. 10.1: fig10_01.cpp
299 // Driver for Employee hierarchy
300 #include <iostream>
301
302 using std::cout;
303 using std::endl;
304
305 #include <iomanip>
306
307 using std::ios;
```

C++

# Örnek (devamı)

---

```
308using std::setiosflags;
309using std::setprecision;
310
311#include "employ2.h"
312#include "boss1.h"
313#include "commis1.h"
314#include "piece1.h"
315#include "hourly1.h"
316
317void virtualViaPointer( const Employee * );
318void virtualViaReference( const Employee & );
319
320int main()
321{
322    // set output formatting
323    cout << setiosflags( ios::fixed | ios::showpoint )
324          << setprecision( 2 );
325
```

C++

# Örnek (devamı)

```

326 Boss b( "John", "Smith", 800.00 );
327 b.print();
328 cout << " earned $" << b.earnings();
329 virtualViaPointer( &b ); // use
330 virtualViaReference( b ); // use
331
332 CommissionWorker c( "Sue", "Jones", 200.00 );
333 c.print();
334 cout << " earned $" << c.earnings();
335 virtualViaPointer( &c ); // use
336 virtualViaReference( c ); // use
337
338 PieceWorker p( "Bob", "Lewis", 2.5, 200.00 );
339 p.print();
340 cout << " earned $" << p.earnings();
341 virtualViaPointer( &p ); // use
342 virtualViaReference( p ); // use
343
344 HourlyWorker h( "Karen", "Price", 13.75, 100.00 );

```

Boss: John Smith earned \$800.00

Boss: John Smith earned \$800.00

Boss: John Smith earned \$800.00

Kendi nesnesini kullanarak print fonksiyonu çağırmak.

Bir base-sınıf pointeri kullanarak print fonksiyonu çağırmak.

Bu virtual fonksiyonları kullanır ve dinamik ömürlüdür.

Bir base-sınıf referansı kullanarak print fonksiyonu çağırmak.

Bu virtual fonksiyonu kullanılır ve dinamik ömürlüdür.

Piece worker: Bob Lewis earned \$500.00

Piece worker: Bob Lewis earned \$500.00 earned \$" &lt;&lt; h.earnings();

Piece worker: Bob Lewis earned \$500.00

Commission worker: Sue Jones earned \$650.00

Commission worker: Sue Jones earned \$650.00

Commission worker: Sue Jones earned \$650.00



C++

# Örnek (devamı)

```

347 virtualViaPointer( &h );
348 virtualViaReference( h );
349 cout << endl;
350 return 0;
351 }
352
353 // Make virtual function calls off
354 // using dynamic binding.
355 void virtualViaPointer( const Employee *baseClassPtr )
356 {
357     baseClassPtr->print();
358     cout << " earned $" << baseClassPtr->earnings();
359 }
360
361 // Make virtual function calls off a base-class reference
362 // using dynamic binding.
363 void virtualViaReference( const Employee &baseClassRef )
364 {
365     baseClassRef.print();
366     cout << " earned $" << baseClassRef.earnings();
367 }

```

Hourly worker: Karen Price earned  
\$550.00

Hourly worker: Karen Price earned  
\$550.00

Hourly worker: Karen Price earned  
\$550.00

Bir base-sınıfı pointer ile  
print virtual fonksiyonu  
çağırarak.

Bir base-sınıfı reference ile  
print virtual fonksiyonu  
çağırarak.

C++

# Program çıktısı

---

```
Boss: John Smith earned $800.00
    Boss: John Smith earned $800.00
    Boss: John Smith earned $800.00
Commission worker: Sue Jones earned $650.00
Commission worker: Sue Jones earned $650.00
Commission worker: Sue Jones earned $650.00
    Piece worker: Bob Lewis earned $500.00
    Piece worker: Bob Lewis earned $500.00
    Piece worker: Bob Lewis earned $500.00
    Hourly worker: Karen Price earned $550.00
    Hourly worker: Karen Price earned $550.00
    Hourly worker: Karen Price earned $550.00
```

C++

# Yeni Sınıflar ve Dinamik Bağlantı

---

- Çoklubenzeşim ve sanal (virtual) fonksiyonlar
  - Bütün sınıflar önceden bilinmediğinde iyi çalışır.
  - Yeni sınıfları bir sisteme yerleştirirken dinamik bağlantı kullanır.
- Dinamik bağlantı (geç bağlantı)
  - Bir sanal fonksiyon için nesnenin türünün derleme zamanında bilinmesi gerekmez.
  - Sanal fonksiyon çağırma çalışma zamanında eşleştirilir.

C++

# Sanal destructor

---

- Problem:
  - Eğer türetilmiş bir nesneye işaret eden bir kök-sınıf işaretçisi silinirse kök-sınıf destructor' u nesne üzerinde etkili olacaktır.
- Çözüm:
  - Uygun destructor' un çağrılacağını garanti etmek için bir sanal kök-sınıf destructor tanımlanır.

C++

# Örnek: Miras Arayüzü

**Point, circle, cylinder hiyerarşisini genişletmek için:**

- Hiyerarşinin başı olarak Shape soyut (abstract) kök-sınıfı kullanıldı:
  - İki saf sanal fonksiyon **printShapeName** ve **print**
  - Diğer iki sanal fonksiyon **volume** ve **area**
- Point, Shape sınıfından türetildi ve miras olarak bu yapıyı aldı.

C++

# Örnek (Sanal destructor)

```
1 // Fig. 10.2: shape.h
2 // Definition of abstract base class Shape
3 #ifndef SHAPE H
4 #define SHAPE H
5
6 class Shape {
7 public:
8     virtual double area() const { return 0.0; }
9     virtual double volume() const { return 0.0; }
10
11     // pure virtual functions overridden in derived classes
12     virtual void printShapeName() const = 0;
13     virtual void print() const = 0;
14 };
15
16 #endif
17 // Fig. 10.2: point1.h
18 // Definition of class Point
19 #ifndef POINT1 H
20 #define POINT1 H
21
22 #include <iostream>
```

Not :Her bir sınıf tarafından tekrar yazılan virtual fonksiyonlar.

C++

# Örnek (devamı)

```
23
24 using std::cout;
25
26 #include "shape.h"
27
28 class Point : public Shape {
29 public:
30     Point( int = 0, int = 0 ); // default constructor
31     void setPoint( int, int );
32     int getX() const { return x; }
33     int getY() const { return y; }
34
35     virtual void printShapeName() const { cout << "Point: "; }
36     virtual void print() const;
37 private:
38     int x, y; // x and y coordinates of Point
39 };
40 #endif
```

**Point** abstract base sınıfından miras alıyor.

C++

# Örnek (devamı)

---

```
41 // Fig. 10.2: point1.cpp
42 // Member function definitions for class Point
43 #include "point1.h"
44
45 Point::Point( int a, int b ) { setPoint( a, b ); }
46
47 void Point::setPoint( int a, int b )
48 {
49     x = a;
50     y = b;
51 }
52
53 void Point::print() const
54     { cout << '[' << x << ", " << y << ']' ; }
```



C++

# Örnek (devamı)

```
55 // Fig. 10.2: circle1.h
56 // Definition of class Circle
57 #ifndef CIRCLE1_H
58 #define CIRCLE1_H
59 #include "point1.h"
60
61 class Circle : public Point {
62 public:
63     // default constructor
64     Circle( double r = 0.0, int x = 0, int y = 0 );
65
66     void setRadius( double );
67     double getRadius() const;
68     virtual double area() const;
69     virtual void printShapeName() const { cout << "Circle: "; }
70     virtual void print() const;
```

Circle, Point sınıfından miras alıyor.

C++

# Örnek (devamı)

---

```
71 private:
72     double radius;    // radius of Circle
73 };
74
75 #endif
76 // Fig. 10.2: circle1.cpp
77 // Member function definitions for class Circle
78 #include <iostream>
79
80 using std::cout;
81
82 #include "circle1.h"
83
84 Circle::Circle( double r, int a, int b )
85     : Point( a, b )    // call base-class constructor
86 { setRadius( r ); }
```

C++

# Örnek (devamı)

---

```
87
88 void Circle::setRadius( double r ) { radius = r > 0 ? r : 0; }
89
90 double Circle::getRadius() const { return radius; }
91
92 double Circle::area() const
93     { return 3.14159 * radius * radius; }
94
95 void Circle::print() const
96 {
97     Point::print();
98     cout << "; Radius = " << radius;
99 }
```

C++

# Örnek (devamı)

```
100// Fig. 10.2: cylindr1.h
101// Definition of class Cylinder
102#ifndef CYLINDR1_H
103#define CYLINDR1_H
104#include "circle1.h"
105
106class Cylinder : public Circle {
107public:
108    // default constructor
109    Cylinder( double h = 0.0, double r = 0.0,
110            int x = 0, int y = 0 );
111
112    void setHeight( double );
113    double getHeight();
114    virtual double area() const;
```

Cylinder  
Circle'den miras  
alıyor.

C++

# Örnek (devamı)

---

```
115     virtual double volume() const;
116     virtual void printShapeName() const { cout << "Cylinder: "; }
117     virtual void print() const;
118 private:
119     double height;    // height of Cylinder
120 };
121
122 #endif
123 // Fig. 10.2: cylindr1.cpp
124 // Member and friend function definitions for class Cylinder
125 #include <iostream>
126
127 using std::cout;
128
129 #include "cylindr1.h"
130
131 Cylinder::Cylinder( double h, double r, int x, int y )
132     : Circle( r, x, y )    // call base-class constructor
133 { setHeight( h ); }
```

C++

# Örnek (devamı)

---

```
134
135void Cylinder::setHeight( double h )
136    { height = h > 0 ? h : 0; }
137
138double Cylinder::getHeight() { return height; }
139
140double Cylinder::area() const
141{
142    // surface area of Cylinder
143    return 2 * Circle::area() +
144           2 * 3.14159 * getRadius() * height;
145}
146
147double Cylinder::volume() const
148    { return Circle::area() * height; }
149
150void Cylinder::print() const
151{
152    Circle::print();
153    cout << "; Height = " << height;
154}
```

C++

# Örnek (devamı)

---

```
155// Fig. 10.2: fig10_02.cpp
156// Driver for shape, point, circle, cylinder hierarchy
157#include <iostream>
158
159using std::cout;
160using std::endl;
161
162#include <iomanip>
163
164using std::ios;
165using std::setiosflags;
166using std::setprecision;
167
168#include "shape.h"
169#include "point1.h"
170#include "circle1.h"
171#include "cylindr1.h"
172
173void virtualViaPointer( const Shape * );
174void virtualViaReference( const Shape & );
175
```

C++

# Örnek (devamı)

```

176 int main()
177 {
178     cout << setiosflags( ios::fixed | ios::showpoint )
179         << setprecision( 2 );
180
181     Point point( 7, 11 );
182     Circle circle( 3.5, 22, 8 );
183     Cylinder cylinder( 10, 3.3, 10, 10 ); // create a Cylinder
184
185     point.printShapeName(); // static binding
186     point.print(); // static binding
187     cout << '\n';
188
189     circle.printShapeName(); // static binding
190     circle.print();
191     cout << '\n';
192
193     cvlinder.printShapeName(); // static binding
194     cvlinder.print(); // static binding
195     cout << "\n\n";
196

```

Point: [7, 11]

Circle: [22, 8]; Radius = 3.50

Cylinder: [10, 10]; Radius = 3.30; Height = 10.00

Nesnenin kendisini kullanarak Print çağırılıyor.



C++

# Örnek (devamı)

```

197 Shape *arravOfShapes[ 3 ]: // array of base
198
199 // aim arravOfShapes[0] at derived-class Point
200 arravOfShapes[ 0 ] = &point;
201
202 // aim arravOfShapes[1] at derived-class Circle
203 arravOfShapes[ 1 ] = &circle;
204
205 // aim arravOfShapes[2] at derived-class Cylinder
206 arravOfShapes[ 2 ] = &cylinder;
207
208 // Loop through arravOfShapes and call virtual functions
209 // to print the shape name, attributes, area, and volume
210 // of each object using dynamic binding.
211 cout << "Virtual function calls made off "
212      << "base-class pointers\n";
213
214 for ( int i = 0; i < 3; i++ )
215     virtualViaPointer( arravOfShapes[ i ] )
216
217 // Loop through arravOfShapes and call virtual functions
218 // to print the shape name, attributes, area, and volume
219 // of each object using dynamic binding.

```

Base sınıf işaretçileri dizisi oluşturuluyor. Bunlara nesneleri atayıp, base sınıf işaretçileriyle print fonksiyonunu tekrar çağırılınca uygun virtual fonksiyonlar çağrılacak.

Virtual function calls made off base-class pointers

Circle: [22, 8];  
Radius = 3.50  
Area = 38.48  
Volume = 0.00

Cylinder: [10, 10]; Radius = 3.30; Height = 10.00  
Area = 275.77  
Volume = 342.12

Point: [7, 11]  
Area = 0.00  
Volume = 0.00

C++

# Örnek (devamı)

```
220     cout << "Virtual function calls made off "
221           << "base-class references\n";
222
223     for ( int j = 0; j < 3; j++ )
224         virtualViaReference( *arrayOfShapes[ j ] );
225
226     return 0;
227 }
228
229 // Make virtual function calls off a base-class pointer
230 // using dynamic binding.
231 void virtualViaPointer( const Shape *baseClassPtr )
232 {
233     baseClassPtr->printShapeName();
234     baseClassPtr->print();
235     cout << "\nArea = " << baseClassPtr->area()
236          << "\nVolume = " << baseClassPtr->volume() << "\n\n";
237 }
238
```

base-sınıfı referansları kullanılarak işlemler tekrarlanıyor.

C++

# Örnek (devamı)

```
239// Make virtual function calls off a base-class reference
240// using dynamic binding.
241void virtualViaReference( const Shape &baseClassRef )
242{
243    baseClassRef.printShapeName();
244    baseClassRef.print();
245    cout << "\nArea = " << baseClassRef.area()
246          << "\nVolume = " << baseClassRef.volume() << "\n\n";
247}
```

Virtual function calls made off base-class references

Point: [7, 11]

Area = 0.00

Volume = 0.00

Circle: [22, 8]; Radius = 3.50

Area = 38.48

Volume = 0.00

Cylinder: [10, 10]; Radius = 3.30; Height = 10.00

Area = 275.77

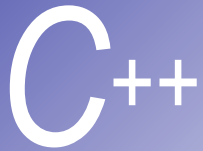
Volume = 342.12

C++

# Örnek (devamı)

---

```
Point: [7, 11]
Circle: [22, 8]; Radius = 3.50
Cylinder: [10, 10]; Radius = 3.30; Height = 10.00
Virtual function calls made off base-class pointers
Point: [7, 11]
Area = 0.00
Volume = 0.00
Circle: [22, 8]; Radius = 3.50
Area = 38.48
Volume = 0.00
Cylinder: [10, 10]; Radius = 3.30; Height = 10.00
Area = 275.77
Volume = 342.12
Virtual function calls made off base-class references
Point: [7, 11]
Area = 0.00
Volume = 0.00
Circle: [22, 8]; Radius = 3.50
Area = 38.48
Volume = 0.00
Cylinder: [10, 10]; Radius = 3.30; Height = 10.00
Area = 275.77
Volume = 342.12
```



# Çoklubenzeşim, sanal fonksiyonlar ve dinamik bağlama

---

- Ne zaman Çoklubenzeşim kullanılır?
  - Çoklubenzeşim bir çok ek yük gerektirir
  - Çoklubenzeşim performansı iyileştirmek için STL' de (Standard Template Library) kullanılmamıştır.
- sanal fonksiyon tablosu (vtable):
  - Bir sanal fonksiyona sahip her sınıf bir vtable sahibidir.
  - Her sanal fonksiyona ait vtable uygun fonksiyona işaret eden bir işaretçiye sahiptir
  - Eğer türetilmiş sınıf kök-sınıfı aynı fonksiyona sahipse fonksiyon işaretçisi kök-sınıf fonksiyonuna işaret eder.