

C++

## Ders 3

Nesne Erişim Kontrolü  
Friend fonksiyonları ve  
`this` pointer'ı

C++

# Bir sınıfın üyesi olan nesneler (Composition)

- Sınıflar, diğer sınıflardan nesne üyelere sahip olabilirler. Buna **Composition** denir
- Üye nesneler, tanımlandıkları sırada oluşturulurlar (initializer bölümünde yazıldıkları sırada değil).
- Üye nesneler, üyesi oldukları sınıftan önce oluşturulurlar.
- “Software reusability”nin en yaygın formu composition’ dur

```
*date1.h  
classDate
```

```
*date1.cpp  
#include"date1.h"
```

```
*empty1.h  
#include"date1.h"
```

```
*empty1.cpp  
#include"empty1.h"  
#include"date1.h"
```

```
*fig07_04.cpp  
#include"empty1.h"
```

C++

# Örnek (Sınıf Üyeleri)

---

```
1 // Fig. 7.4: date1.h
2 // Declaration of the Date class.
3 // Member functions defined in date1.cpp
4 #ifndef DATE1_H
5 #define DATE1_H
6
7 class Date {
8 public:
9     Date( int = 1, int = 1, int = 1900 ); // default constructor
10    void print() const; // print date in month/day/year format
11    ~Date(); // provided to confirm destruction order
12 private:
13    int month; // 1-12
14    int day; // 1-31 based on month
15    int year; // any year
```

C++

# Örnek (devamı)

---

```
16
17     // utility function to test proper day for month and year
18     int checkDay( int );
19 };
20
21 #endif
22 // Fig. 7.4: date1.cpp
23 // Member function definitions for Date class.
24 #include <iostream>
25
26 using std::cout;
27 using std::endl;
28
29 #include "date1.h"
30
31 // Constructor: Confirm proper value for month;
32 // call utility function checkDay to confirm proper
33 // value for day.
```

C++

# Örnek (devamı)

```
34 Date::Date( int mn, int dy, int yr )
35 {
36     if ( mn > 0 && mn <= 12 )           // validate the month
37         month = mn;
38     else {
39         month = 1;
40         cout << "Month " << mn << " invalid. Set to month 1.\n";
41     }
42
43     year = yr;                           // should validate the year
44     day = checkDay( dy );                // validate the day
45
46     cout << "Date object constructor for date ";
47     print();                             // interesting: a print with no arguments
48     cout << endl;
49 }
50
```

Constructor  
çağırıldığında bu yazıyı  
yazacak

C++

# Örnek (devamı)

```
51 // Print Date object in form month/day/year
52 void Date::print() const
53     { cout << month << '/' << day << '/' << year; }
54
55 // Destructor: provided to confirm destruction order
56 Date::~Date()
57 {
58     cout << "Date object destructor for date ";
59     print();
60     cout << endl;
61 }
62
63 // Utility function to confirm proper day value
64 // based on month and year.
65 // Is the year 2000 a leap year?
66 int Date::checkDay( int testDay )
67 {
68     static const int daysPerMonth[ 13 ] =
69         {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
70
71     if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
72         return testDay;
```

Destructor  
çağrıldığında bu  
yazıyı yazacak.

C++

# Örnek (devamı)

---

```

74     if ( month == 2 &&          // February: Check for leap year
75         testDay == 29 &&
76         ( year % 400 == 0 ||
77         ( year % 4 == 0 && year % 100 != 0 ) ) )
78         return testDay;
79
80     cout << "Day " << testDay << " invalid. Set to day 1.\n";
81
82     return 1;  // leave object in consistent state if bad value
83 }
84 // Fig. 7.4: employ1.h
85 // Declaration of the Employee class.
86 // Member functions defined in employ1.cpp
87 #ifndef EMPLOY1_H
88 #define EMPLOY1_H
89
90 #include "date1.h"
91
92 class Employee {

```

C++

# Örnek (devamı)

```
93 public:
94     Employee( char *, char *, int, int, int, int, int, int );
95     void print() const;
96     ~Employee(); // provided to confirm destruction order
97 private:
98     char firstName[ 25 ];
99     char lastName[ 25 ];
100     const Date birthDate;
101     const Date hireDate;
102 };
103
104 #endif
105 // Fig. 7.4: emply1.cpp
106 // Member function definitions for Employee class.
107 #include <iostream>
108
109 using std::cout;
110 using std::endl;
111
```

Başka sınıftan iki üye tanımlandı!



C++

# Örnek (devamı)

```

112#include <cstring>
113#include "employ1.h"
114#include "date1.h"
115
116Employee::Employee( char *fname, char *lname,
117                    int bmonth, int bday, int byear,
118                    int hmonth, int hday, int hyear )
119    : birthDate( bmonth, bday, byear ),
120      hireDate( hmonth, hday, hyear )
121{
122    // copy fname into firstName and be sure that
123    int length = strlen( fname );
124    length = ( length < 25 ? length : 24 );
125    strncpy( firstName, fname, length );
126    firstName[ length ] = '\0';
127
128    // copy lname into lastName and be sure that
129    length = strlen( lname );
130    length = ( length < 25 ? length : 24 );
131    strncpy( lastName, lname, length );
132    lastName[ length ] = '\0';
133

```

Const nesnelerdeki  
üyelere ilk değerler  
atanıyor

Bu constructor tanımına  
dikkat edin: aslında  
initializer aracılığıyla üye  
nesnenin constructor'una  
bilgi aktarılıyor.

C++

# Örnek (devamı)

```
134 cout << "Employee object constructor: "  
135     << firstName << ' ' << lastName << endl;  
136}  
137  
138void Employee::print() const  
139{  
  
140     cout << lastName << ", " << firstName << "\nHired: "  
  
141     hireDate.print();  
142     cout << "    Birth date: "  
143     birthDate.print();  
  
144     cout << endl;  
  
145 }
```

Constructor  
çağrıldığında bu yazıyı  
yazacak

Burada print de, date nesnesi de const  
olduğu için print fonksiyonu date nesnesini  
kullanarak ekrana yazı yazabiliyor.

Print fonksiyonunun hiç bir parametresi  
olmadığına dikkat edin: Çünkü print  
fonksiyonu onu çağıran nesneye bağlıdır.

C++

# Örnek (devamı)

---

```
146
147// Destructor: provided to confirm destruction order
148Employee::~Employee()
149{
150    cout << "Employee object destructor: "
151          << lastName << ", " << firstName << endl;
152}
153// Fig. 7.4: fig07_04.cpp
154// Demonstrating composition: an object with member objects.
155#include <iostream>
156
```

C++

# Örnek (davamı)

```
157using std::cout;
158using std::endl;
159
160#include "employ1.h"
161
162int main()
163{
164    Employee e( "Bob", "Jones", 7, 24, 1949, 3, 12, 1988 );
165
166    cout << '\n';
167    e.print();
168
169    cout << "\nTest Date constructor with invalid values:\n";
170    Date d( 14, 35, 1994 ); // invalid Date values
```

Sadece **employ.h** dosyası yüklenmelidir. Bu dosya **date.h** dosyasını kendi yükler.

C++

# Program çıktısı

```
171     cout << endl;  
172     return 0;  
173 }
```

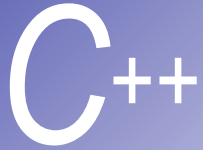
```
Date object constructor for date 7/24/1949  
Date object constructor for date 3/12/1988  
Employee object constructor: Bob Jones
```

```
Jones, Bob  
Hired: 3/12/1988   Birth date: 7/24/1949
```

```
Test Date constructor with invalid values:  
Month 14 invalid. Set to month 1.  
Day 35 invalid. Set to day 1.  
Date object constructor for date 1/1/1994
```

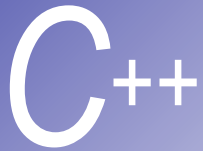
```
Date object destructor for date 1/1/1994  
Employee object destructor: Jones, Bob  
Date object destructor for date 3/12/1988  
Date object destructor for date 7/24/1949
```

Dikkat: Hangi nesne ilk olarak oluşturuluyor ve yok ediliyor!



## Friend Fonksiyon ve Sınıfları

- Friend fonksiyon bir nesnenin ve sınıflarının özel ve korunan üyelerine ulaşabilir.
- Ancak, friend fonksiyonlar sınıfın üyesi değildirler.
- A nesnesi B nesnesinde friend olarak tanımlı ise, bu B nesnesini A nesnesinde friend yapmaz.
- Bir C nesnesinde B nesnesi friend ise, bu A nesnesi için C nesnesini friend yapmaz.



## Friend bildirimleri

- Fonksiyon ismi prototipi ve türünden önce yazılırlar.

```
friend int myFunction( int x );
```

- Sınıf ismi ve türünden önce yazılırlar.

```
friend class ClassTwo;
```

C++

# Friend private üyelere ulaşabilir

```

1 // Fig. 7.5: fig07_05.cpp
2 // Friends can access private members of a class.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 // Modified Count class
9 class Count {
10     friend void setX( Count &, int ); // friend declaration
11 public:
12     Count() { x = 0; } // constructor
13     void print() const { cout << x << endl; } // output
14 private:
15     int x; // data member
16 };
17
18 // Can modify private data of Count because
19 // setX is declared as a friend function of Count
20 void setX( Count &c, int val )

```

**setX count türü bir friend' dir (private dataya ulaşabilir).**

**setX fonksiyonu Count' un bir üyesi olmadığından normal olarak tanımlanabilir.**



C++

# Program çıktısı

---

```
21 {
22     c.x = val;  // legal: setX is a friend of Count
23 }
24
25 int main()
26 {
27     Count counter;
28
29     cout << "counter.x after instantiation: ";
30     counter.print();
31     cout << "counter.x after call to setX friend function: ";
32
33     setX( counter, 8 );  // set x with a friend
34
35     counter.print();
36
37     return 0;
38 }
```

```
counter.x after instantiation: 0
counter.x after call to setX friend function: 8
```

C++

# Örnek

```

1 // Fig. 7.6: fig07_06.cpp
2 // Non-friend/non-member functions cannot access
3 // private data of a class.
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8
9 // Modified Count class
10 class Count {
11 public:
12     Count() { x = 0; } // constructor
13     void print() const { cout << x << endl; } // output
14 private:
15     int x; // data member
16 };
17
18 // Function tries to modify private data of Count.
19 // but cannot because it is not a friend of Count.
20 void cannotSetX( Count &c, int val )
21 {
22     c.x = val; // ERROR: 'Count::x' is not accessible
23 }

```

cannotSetX count için bir friend olmadığından private dataya ulaşamaz

C++

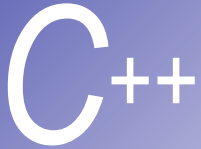
# Programın Çıktısı

```
24
25 int main()
26 {
27     Count counter;
28
29     cannotSetX( counter, 3 ); // cannotSetX is not a friend
30     return 0;
31 }
```

```
Compiling...
Fig07_06.cpp
D:\books\2000\cpphttp3\examples\Ch07\Fig07_06\Fig07_06.cpp(22) :
    error C2248: 'x' : cannot access private member declared in
        class 'Count'
        D:\books\2000\cpphttp3\examples\Ch07\Fig07_06\
        Fig07_06.cpp(15) : see declaration of 'x'
Error executing cl.exe.

test.exe - 1 error(s), 0 warning(s)
```

**Private** dataya ulaşamadığından oluşan bir derleyici hatası



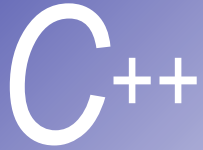
# This pointer' ının kullanımı

- **This** pointeri her nesnenin kendi adresine ulaşımı sağlar.
- Nesnenin bir parçası değildir, nesnenin üyelerine erişmek için kullanılabilecek bir pointer' dır.
- **Employee** türündeki bir sınıfın **non-constant** üye fonksiyonu için **const this** pointer' i gösterimi (**Employee** nesnesini gösteren **const** pointer)

**Employee \* const**

- **Employee** türündeki bir sınıfın **constant** üye fonksiyonu için **const this** pointer' in gösterimi (**const Employee** nesnesini gösteren **const** pointer)

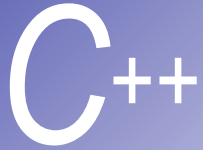
**const Employee \* const**



## **This pointer' inin kullanımı**

- **This** pointer örnekleri  
**this->x** veya  
**( \*this ).x**
- İç-içe fonksiyon üyesi çağrımını mümkün kılar.
- Fonksiyon kendi nesnesinin adresi ile geri döner ve bu nesnenin diğer fonksiyonlarda da kullanılmasına imkan sağlar:

```
{ return *this; }
```



## This pointer' inin kullanımı

- **setHour**, **setMinute**, ve **setSecond** üye fonksiyonlarının hepsinin geri dönüş değeri **\*this** pointer' ıdır (kendi nesnesinin adresi).
  - Bu komut çalışınca:  
**t.setHour(1).setMinute(2).setSecond(3);**
  - **t.setHour(1)** önce çalıştırılır, geriye **\*this** (t' nin adresi ile) döner. Yani ifade aşağıdaki gibi olur:  
**t.setMinute(2).setSecond(3);**
  - **t.setMinute(2)** kısmı çalışınca nesnenin adresini tekrar döndürerek **t.setSecond(3);** halini alır.
  - **t.setSecond(3)**, nesnenin adresini tekrar döndürerek **t** olur.

C++

# Örnek (this pointer' i)

```

1 // Fig. 7.7: fig07_07.cpp
2 // Using the this pointer to refer to object members.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 class Test {
9 public:
10     Test( int = 0 );           // default constructor
11     void print() const;
12 private:
13     int x;
14 };
15
16 Test::Test( int a ) { x = a; } // constructor
17
18 void Test::print() const       // ( ) around const
19 {
20     cout << "        x = " << x
21         << "\n  this->x = " << this->x
22         << "\n(*this).x = " << ( *this ).x << endl;
23 }
24

```

x 'i -> operatörü kullanarak ekrana yazdırmak

Direkt olarak x ekranda görünür.

Nokta operatörü(.) kullanılarak x görüntülenir fakat (.) operatörünün öncelik seviyesi \* operatöründen daha fazla olduğu için parantez kullanılmalıdır

# C++

## Program çıktısı

```
25 int main()  
26 {  
27     Test testObject( 12 );  
28  
29     testObject.print();  
30  
31     return 0;  
32 }
```

```
        x = 12  
this->x = 12  
(*this).x = 12
```

Her üç metot da aynı sonucu verir



C++

# Örnek II (this pointer' i)

```
1 // Fig. 7.8: time6.h
2 // Cascading member function calls.
3
4 // Declaration of class Time.
5 // Member functions defined in time6.cpp
6 #ifndef TIME6_H
7 #define TIME6_H
8
9 class Time {
10 public:
11     Time( int = 0, int = 0, int = 0 ); // default constructor
12
13     // set functions
14     Time &setTime( int, int, int ); // set hour, minute, second
15     Time &setHour( int ); // set hour
16     Time &setMinute( int ); // set minute
17     Time &setSecond( int ); // set second
18
```

Dikkat **Time &...** fonksiyonu **Time** nesnesinin adresini döndürür.

C++

# Örnek II (this pointer' i)

---

```
19 // get functions (normally declared const)
20 int getHour() const; // return hour
21 int getMinute() const; // return minute
22 int getSecond() const; // return second
23
24 // print functions (normally declared const)
25 void printMilitary() const; // print military time
26 void printStandard() const; // print standard time
27 private:
28 int hour; // 0 - 23
29 int minute; // 0 - 59
30 int second; // 0 - 59
31 };
32
33 #endif
```

C++

# Örnek II (this pointer' i)

---

```
34 // Fig. 7.8: time.cpp
35 // Member function definitions for Time class.
36 #include <iostream>
37
38 using std::cout;
39
40 #include "time6.h"
41
42 // Constructor function to initialize private data.
43 // Calls member function setTime to set variables.
44 // Default values are 0 (see class definition).
45 Time::Time( int hr, int min, int sec )
46     { setTime( hr, min, sec ); }
47
```

C++

# Örnek II (this pointer' i)

```
48 // Set the values of hour, minute, and second.
49 Time &Time::setTime( int h, int m, int s )
50 {
51     setHour( h );
52     setMinute( m );
53     setSecond( s );
54     return *this;    // enables cascading
55 }
56
57 // Set the hour value
58 Time &Time::setHour( int h )
59 {
60     hour = ( h >= 0 && h < 24 ) ? h : 0;
61
62     return *this;    // enables cascading
63 }
```

**\*this** değerinin döndürülmesi ile iç-içe fonksiyonlar çağırılabilir.

C++

# Örnek II (this pointer' i)

---

```
65 // Set the minute value
66 Time &Time::setMinute( int m )
67 {
68     minute = ( m >= 0 && m < 60 ) ? m : 0;
69
70     return *this;    // enables cascading
71 }
72
73 // Set the second value
74 Time &Time::setSecond( int s )
75 {
76     second = ( s >= 0 && s < 60 ) ? s : 0;
77
78     return *this;    // enables cascading
79 }
80
81 // Get the hour value
82 int Time::getHour() const { return hour; }
83
```

C++

# Örnek II (this pointer' i)

---

```
84 // Get the minute value
85 int Time::getMinute() const { return minute; }
86
87 // Get the second value
88 int Time::getSecond() const { return second; }
89
90 // Display military format time: HH:MM
91 void Time::printMilitary() const
92 {
93     cout << ( hour < 10 ? "0" : "" ) << hour << ":"
94         << ( minute < 10 ? "0" : "" ) << minute;
95 }
96
97 // Display standard format time: HH:MM:SS AM (or PM)
98 void Time::printStandard() const
99 {
100     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
101         << ":" << ( minute < 10 ? "0" : "" ) << minute
102         << ":" << ( second < 10 ? "0" : "" ) << second
103         << ( hour < 12 ? " AM" : " PM" );
104 }
```

C++

# Örnek II (this pointer' i)

```
105// Fig. 7.8: fig07_08.cpp
106// Cascading member function calls together
107// with the this pointer
108#include <iostream>
109
110using std::cout;
111using std::endl;
112
113#include "time6.h"
114
115int main()
116{
117    Time t;
118
119    t.setHour( 18 ).setMinute( 30 ).setSecond( 22 );
120    cout << "Military time: ";
121    t.printMilitary();
```

İç-içe fonksiyon çağırılmasına dikkat!

C++

# Örnek II (this pointer' i)

```
122 cout << "\nStandard time: ";
123 t.printStandard();
124
125 cout << "\n\nNew standard time: ";
126 t.setTime( 20, 20, 20 ).printStandard();
127 cout << endl;
128
129 return 0;
130 }
```

**printStandard** fonksiyonu nesnenin adresini döndürmediği için sadece en sondaki iç-içe fonksiyon olabilir.

Yani **t.printStandard().setTime()** ; yazılsa idi compiler hata verirdi.

Military time: 18:30  
Standard time: 6:30:22 PM

New standard time: 8:20:20 PM



C++

# new ve delete Operatörleri ile Dinamik Bellek Kullanımı

malloc allocate edilen hafıza bloğunu initialize etmez



fakat C++ da bu kolaylıkla sağlanır

Eğer new hafızada bir boşluk bulamazsa 0 pointer geri dönderir

- **malloc** ve **free** yerine bellek tahsisatı yapmak için daha üstün C++ fonksiyonlarıdır.
- **new** ile bir nesne oluşturulur, onun constructor' unu çağırır ve doğru türde bir pointer geri döndürür.
- **delete** nesneleri yok eder ve hafızayı serbest bırakır.
- **new** örnekleri: `c de → typeNamePtr=malloc(sizeof(TypeName));`  
**TypeName \*typeNamePtr;**
- **TypeName** nesnesi için bir pointer oluşturmak:  
**typeNamePtr = new TypeName;**
- **new TypeName** nesnesini oluşturur, geri dönüş değeri **typeNamePtr** pointer' ına atanır.

C++

# **new ve delete Operatörleri ile Dinamik Bellek Tahsisatı**

---

**delete örnekleri:**

```
delete typeNamePtr;
```

■ **typeName** nesnesinin destructor' u çağrılır ve kullandığı bellek boşaltılır.

**delete [ ] arrayPtr;** dizi dinamik olarak silinir.

**Nesnelere ilk değer vermek:**

```
double *thingPtr = new double(3.14159);
```

■ **double** türden nesneye ilk değer olarak 3.14159 atanıyor.

```
int *arrayPtr = new int[ 10 ];
```

■ **arrayPtr** pointer' i ile 10 elemanlı **int** türünden dizi oluşturmak.

C++

# static Sınıf Üyeleri

## Örnek:

martian sınıfından türemiş tüm nesneler eğer martianCount' u bilimleri ve güncel veri almaları gerekiyorsa sadece bir kopyası çoğaltılır buda **static** ile olur

- **static** üye, sınıfının bütün nesnelerince paylaşılır.
- Normal olarak her nesne kendi kopyasını kullanır.
- Tek bir veri tüm sınıf üyelerince kullanılacaksa yararlıdır. Bu tek veriyi her nesne değiştirebilir.
- Global bir değişkenden farkı, sadece o sınıf nesnelerinin ulaşabilmesidir.
- Dosya faaliyet alanı içinde ilk değer verilir.
- Hiçbir nesne oluşturulmasa bile **static** üye oluşturulur.
- Fonksiyon da, datalar da **static** olabilir.
- **Public**, **private** veya **protected** olabilir.

C++

# static sınıf üyeleri

- **static** değişkenlere kendi sınıfından herhangi bir nesne ulaşabilir.
- **public static** değerlere ‘scope resolution’ (::) operatorü ile ulaşılabilir:

**Employee::count**

- **private static** değerlere bir sınıf hiç nesnesi olmadığında sadece **public static** üye fonksiyonundan ulaşılabilir. ‘scope resolution’ (::) operatörü ve fonksiyon ismi kullanılarak:

**Employee::getCount()**

C++

# static fonksiyonlar

- **static** üye fonksiyonlar statik olmayan data ve fonksiyonlara ulaşamazlar.
- Bir **static** fonksiyon için **This** pointer yoktur. Nesnelerden bağımsızdırlar.

C++

# Örnek (static fonksiyonlar)

```
1 // Fig. 7.9: employ1.h
2 // An employee class
3 #ifndef EMPLOY1_H
4 #define EMPLOY1_H
5
6 class Employee {
7 public:
8     Employee( const char*, const char* ); // constructor
9     ~Employee(); // destructor
10    const char *getFirstName() const; // return first name
11    const char *getLastName() const; // return
12
13    // static member function
14    static int getCount(); // return # objects instantiated
15
```

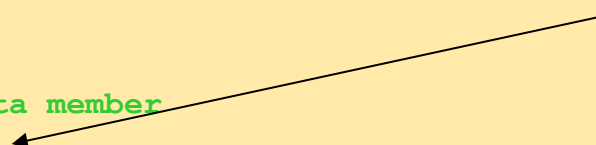
Static üye fonksiyon bildirimi

C++

# Örnek (devamı)

```
16 private:
17     char *firstName;
18     char *lastName;
19
20     // static data member
21     static int count; // number of objects instantiated
22 };
23
24 #endif
25 // Fig. 7.9: employ1.cpp
26 // Member function definitions for class Employee
27 #include <iostream>
28
29 using std::cout;
30 using std::endl;
31
```

Static üye değişken bildirimi



C++

# Örnek (devamı)

```
32 #include <cstring>
33 #include <cassert>
34 #include "employ1.h"
35
36 // Initialize the static data member
37 int Employee::count = 0;
38
39 // Define the static member function that
40 // returns the number of employee objects instantiated.
41 int Employee::getCount() { return count; }
42
43 // Constructor dynamically allocates space for
44 // first and last name and uses strcpy to copy
45 // the first and last names into the object
46 Employee::Employee( const char *first, const char *last )
47 {
48     firstName = new char[ strlen( first ) + 1 ];
49     assert( firstName != 0 ); // ensure memory allocated
50     strcpy( firstName, first );
51
52     lastName = new char[ strlen( last ) + 1 ];
53     assert( lastName != 0 ); // ensure memory allocated
54     strcpy( lastName, last );
```

Dinamik bellek tahsisatı  
**assert** ile test edilir.



C++

# Örnek (devamı)

```
55
56     ++count; // increment static count of employees
57     cout << "Employee constructor for " << firstName
58           << ' ' << lastName << " called." << endl;
59 }
60
61 // Destructor deallocates dynamically allocated memory
62 Employee::~Employee()
63 {
64     cout << "~Employee() called for " << firstName
65           << ' ' << lastName << endl;
66     delete [] firstName; // recapture memory
67     delete [] lastName;  // recapture memory
68     --count; // decrement static count of employees
69 }
70
```

Bir constructor - destructor çağrıldığında **static count** değişkeni değişir.

C++

# Örnek (devamı)

---

```
71 // Return first name of employee
72 const char *Employee::getFirstName() const
73 {
74     // Const before return type prevents client from modifying
75     // private data. Client should copy returned string before
76     // destructor deletes storage to prevent undefined pointer.
77     return firstName;
78 }
79
80 // Return last name of employee
81 const char *Employee::getLastName() const
82 {
83     // Const before return type prevents client from modifying
84     // private data. Client should copy returned string before
85     // destructor deletes storage to prevent undefined pointer.
86     return lastName;
87 }
```

C++

# Örnek (devamı)

```

88 // Fig. 7.9: fig07 09.cpp
89 // Driver to test the employee class
90 #include <iostream>
91
92 using std::cout;
93 using std::endl;
94
95 #include "employ1.h"
96
97 int main()
98 {
99     cout << "Number of employees before instantiation is "
100         << Employee::getCount() << endl;    // use class name
101
102     Employee *e1Ptr = new Employee( "Susan", "Baker" );
103     Employee *e2Ptr = new Employee( "Robert", "Jones" );
104
105     cout << "Number of employees after instantiation is "
106         << e1Ptr->getCount();
107
108     cout << "\n\nEmployee 1: "

```

Constructor **new** 'den  
çağırdığından dolayı  
**Count** artar.

Hiç **Employee** nesnesi yoksa  
**getCount** 'a sınıf ismi (::) ile  
erişilir.

Employee sayısı 0

Employee sayısı 2

C++

# Örnek (devamı)

```

109         << e1Ptr->getFirstName()
110         << " " << e1Ptr->getLastName()
111         << "\nEmployee 2: "
112         << e2Ptr->getFirstName()
113         << " " << e2Ptr->getLastName() << "\n\n";
114
115     delete e1Ptr;    // recapture memory
116     e1Ptr = 0;
117     delete e2Ptr;    // recapture memory
118     e2Ptr = 0;
119
120     cout << "Number of employees after deletion is "
121           << Employee::getCount() << endl;
122
123     return 0;
124 }

```

Employee 1: Susan Baker  
Employee 2: Robert Jones

~Employee() called for Susan Baker  
~Employee() called for Robert Jones

C++

# Program çıktısı

```
Number of employees before instantiation is 0  
Employee constructor for Susan Baker called.  
Employee constructor for Robert Jones called.  
Number of employees after instantiation is 2
```

```
Employee 1: Susan Baker  
Employee 2: Robert Jones
```

```
~Employee() called for Susan Baker  
~Employee() called for Robert Jones  
Number of employees after deletion is 0
```

Count sıfıra geri döner

