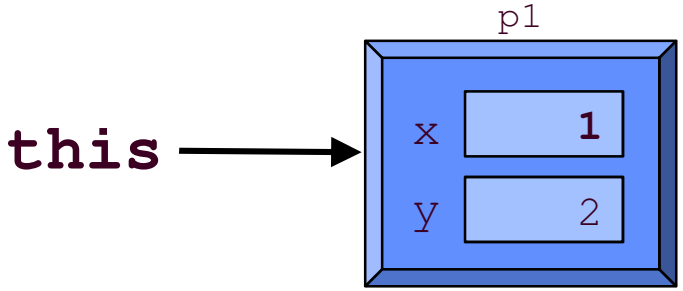
`this` Göstergesi (pointer)

- Belleği etkin kullanmak üzere, her sınıf için tanımlanan üye fonksiyonlar o sınıf türündeki nesnelerin içine tekrar tekrar kopyalanmaz. Üye fonksiyonun tek kopyası bütün nesneler tarafından paylaşılır.
- Fonksiyonun hangi nesne tarafında çağrıldığı this göstergesi (`this` pointer) tarafından belirlenir.
- Her nesne, kendi adresine `this` (a C++ keyword) olarak adlandırılan gösterge ile ulaşır.

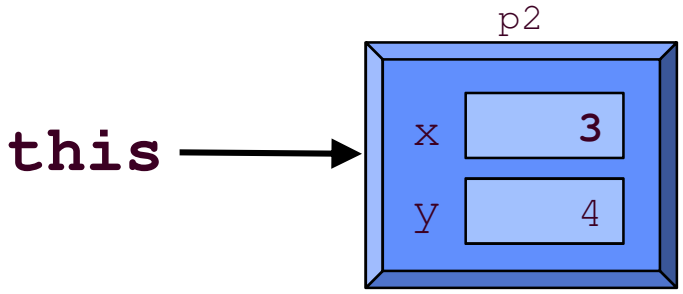
`this` Göstergesi (pointer)

- `p1.print();` fonksiyonu işletildiğinde, print fonksiyonu kodlarındaki **`this`**, p1 nesnesinin adresini tutar.



Fonksiyon çıktısı:
(1,2)

- `p2.print();` fonksiyonu işletildiğinde, print fonksiyonu kodlarındaki **`this`**, p2 nesnesinin adresini tutar.



Fonksiyon çıktısı:
(3,4)

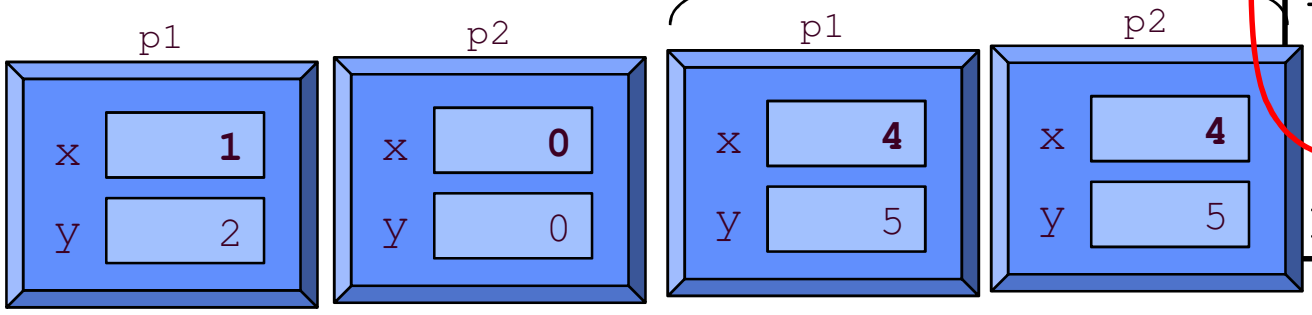
```
#include <iostream>
using namespace std;
class Point{
            int x;
            int y;
public:
            Point(int X=0,int Y=0)
            :x(X),y(Y){}
            void print() const{
                 cout <<"("<< this->x
                      <<","
                      <<(*this).y<<",";
            }
};

int main(){
            Point p1(1,2);
            Point p2(3,4);
            p1.print();
            p2.print();
            return 0;
}
```

`this` Göstergesi (pointer)

- Üye fonksiyon, ait olduğu nesnenin kopyasını döndürmesi gerektiğinde **`this`** göstergesinden faydalanılır.
  - `p2=p1.add(3);` *komutu ile, add fonksiyonu p1 nesnesinin x ve y değerlerini 3 arttırır. Daha sonra fonksiyon, p1 nesnesinin kopyasını döndürür. Bu nesne p2 nesnesine kopyalanır.*

```
#include <iostream>
using namespace std;
class Point{
            int x;
            int y;
public:
            Point(int X=0,int Y=0)
            :x(X),y(Y){}
            void print() const{
                 cout <<"("<< this->x
                      <<","
                      <<(*this).y<<",";
            }
            Point add(int i){
                      x+=i;
                      y+=i;
                      return (*this);
            }
};
int main(){
            Point p1(1,2);
            Point p2;
            p2=p1.add(3);
            return 0;
}
```

`p2=p1.add(3);` fonksiyonu işletildikten sonra



`this` Göstergesi (pointer) – Bir örnek

```
// test.cpp
// Using the this pointer to refer to object members.
#include <iostream>
using namespace std;

class Test
{
public:
        Test( int = 0 ); // default constructor
        void print() const;
private:
        int x;
}; // end class Test

// constructor
Test::Test( int value )
        : x( value ) // initialize x to value
{
        // empty body
} // end constructor Test
```

`this` Göstergesi (pointer) – Bir örnek

```
// print x using implicit and explicit this pointers;
// the parentheses around *this are required
void Test::print() const
{
        // implicitly use the this pointer to access the member x
        cout << " x = " << x;

        // explicitly use the this pointer and the arrow operator
        // to access the member x
        cout << "\n this->x = " << this->x;
        // explicitly use the dereferenced this pointer and
        / the dot operator to access the member x
        cout << "\n(*this).x = " << ( *this ).x << endl;
} // end function print

int main()
{
        Test testObject( 12 ); // instantiate and initialize testObject

        testObject.print();
} // end main
```

`this` Göstergesi (pointer) – Bir örnek

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

`this` Göstergesi (pointer) – Bir kullanımı

- `this` göstergesinin (pointer) kullanımlarından birisi de, kaskat üye fonksiyon çağrısıdır (cascaded member function calls).
  - *Birden çok fonksiyon çağrısı, tek bir komut içinde yapılabilmektedir.*

`this` Göstergesi (pointer) – Bir kullanımı

```
// Time.h
// Cascading member function calls.

// Time class definition.
// Member functions defined in Time.cpp.
#ifndef TIME_H
#define TIME_H

class Time
{
public:
        Time( int = 0, int = 0, int = 0 ); // default constructor

        // set functions (the Time & return types enable cascading)
        Time &setTime( int, int, int ); // set hour, minute, second
        Time &setHour( int ); // set hour
        Time &setMinute( int ); // set minute
        Time &setSecond( int ); // set second

        // get functions (normally declared const)
        int getHour() const; // return hour
        int getMinute() const; // return minute
        int getSecond() const; // return second
```

`this` Göstergesi (pointer) – Bir kullanımı

```
        // print functions (normally declared const)
        void printUniversal() const; // print universal time
        void printStandard() const; // print standard time
private:
        int hour; // 0 - 23 (24-hour clock format)
        int minute; // 0 - 59
        int second; // 0 - 59
}; // end class Time

#endif
```

`this` Göstergesi (pointer) – Bir kullanımı

```
// Time.cpp
// Time class member-function definitions.
#include <iostream>
#include <iomanip>
#include "Time.h" // Time class definition
using namespace std;

// constructor function to initialize private data;
// calls member function setTime to set variables;
// default values are 0 (see class definition)
Time::Time( int hr, int min, int sec )
{
        setTime( hr, min, sec );
} // end Time constructor

// set values of hour, minute, and second
Time &Time::setTime( int h, int m, int s ) // note Time & return
{
        setHour( h );
        setMinute( m );
        setSecond( s );
        return *this; // enables cascading
} // end function setTime
```

`this` Göstergesi (pointer) – Bir kullanımı

```
// set hour value
Time &Time::setHour( int h ) // note Time & return
{
        hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
        return *this; // enables cascading
} // end function setHour

// set minute value
Time &Time::setMinute( int m ) // note Time & return
{
        minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
        return *this; // enables cascading
} // end function setMinute

// set second value
Time &Time::setSecond( int s ) // note Time & return
{
        second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
        return *this; // enables cascading
} // end function setSecond
```

`this` Göstergesi (pointer) – Bir kullanımı

```
// get hour value
int Time::getHour() const
{
        return hour;
} // end function getHour

// get minute value
int Time::getMinute() const
{
        return minute;
} // end function getMinute

// get second value
int Time::getSecond() const
{
        return second;
} // end function getSecond
```

`this` Göstergesi (pointer) – Bir kullanımı

```
// print Time in universal-time format (HH:MM:SS)
void Time::printUniversal() const
{
        cout << setfill( '0' ) << setw( 2 ) << hour << ":"
                << setw( 2 ) << minute << ":" << setw( 2 ) << second;
} // end function printUniversal

// print Time in standard-time format (HH:MM:SS AM or PM)
void Time::printStandard() const
{
        cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
                << ":" << setfill( '0' ) << setw( 2 ) << minute
                <<":"<< setw( 2 ) << second << (hour<12 ? " AM":" PM" );
} // end function printStandard
```

`this` Göstergesi (pointer) – Bir kullanımı

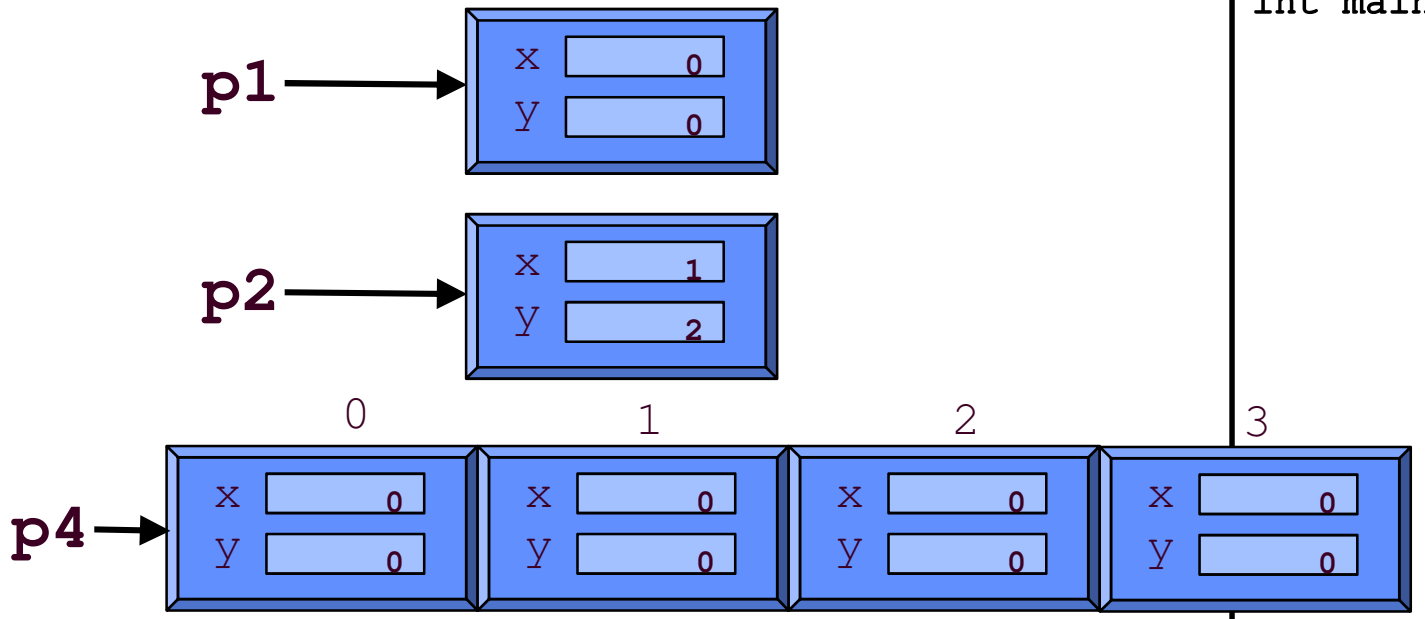
```
// main.cpp
// Cascading member-function calls with the this pointer.
#include <iostream>
#include "Time.h" // Time class definition
using namespace std;

int main()
{
         Time t; // create Time object

         // cascaded function calls
         t.setHour( 18 ).setMinute( 30 ).setSecond( 22 );

         // output time in universal and standard formats
         cout << "Universal time: ";
         t.printUniversal();

         cout << "\nStandard time: ";
         t.printStandard();

         cout << "\n\nNew standard time: ";

         // cascaded function calls
         t.setTime( 20, 20, 20 ).printStandard();
         cout << endl;
} // end main
```

`this` Göstergesi (pointer) – Bir kullanımı

```
Universal time: 18:30:22
Standard time: 6:30:22 PM

New standard time: 8:20:20 PM
```

Nesneler ve Diziler(Arrays) için Dinamik Bellek Kullanımı

- Nesneler için, new komutu ile nesne boyutunda bellekten yer alınabilir. Bellekte alınan bu isimsiz nesnenin adresi bir göstergeye (pointer) atanıp, nesnenin üyelerine −> operatörü ile erişilebilir.




```
#include <iostream>
using namespace std;
class Point{
          int x;
          int y;
public:
          Point(int X=0,int Y=0)
          :x(X),y(Y){}
          void print() const{
              cout <<"("<<x<<","<<y<<",";
          }
};
int main(){
          Point *p1=new Point;
          Point *p2=new Point(1,2);
          Point p3[4];
          Point *p4=new Point[3];

          p1->print();
          p2->print();
          p3[2].print();
          p4[1].print();

          delete p1;
          delete p2;
          delete [] p4;
          return 0;
}
```

Statik (static) Sınıf Üyeleri

- Bir sınıf türündeki tüm nesnelerin ortak kullandıkları üyelere statik (static) sınıf üyeleri denir.
- Bu üyeler tanımlanırken, static anahtar kelimesinin kullanılması gerekir.
- static üyeler public, private ya da protected olarak tanımlanabilir.
- static üyeler, bir kere istenilen değerle başlatılabilir.

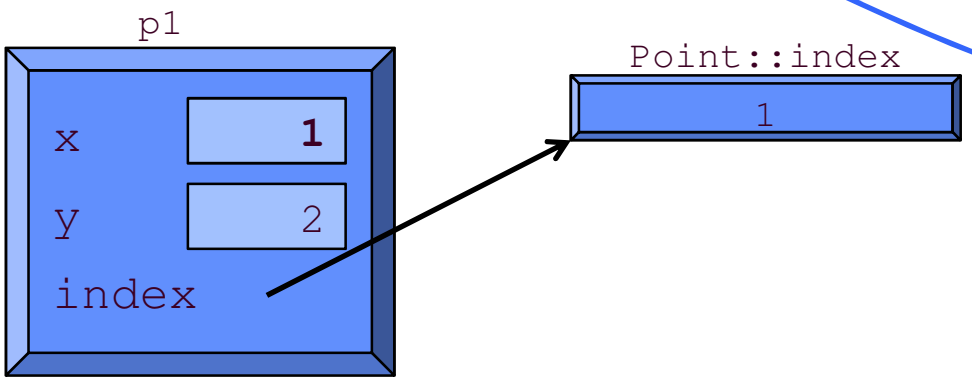
```
#include <iostream>
using namespace std;
class Point{
            int x;
            int y;
            static int index;
public:
            Point(int X=0,int Y=0,int i=0)
            :x(X),y(Y),index(i){}
            void print() const{
                cout <<"("<<x<<","<<y<<",";
            }
            static void setIndex(int i){
                index=i;
            }
};
int Point::index=0;
int main(){
            Point p1(1,2,1);
            Point p2(3,4,2);
            Point p3(5,6,3);
            return 0;
}
```

Statik(static) Sınıf Üyeleri

- Program çalıştığında, henüz hiçbir nesne yokken statik üyeler mevcuttur.

- `Point::setIndex(5);` komutu işletildiğinde, index değişkenin değeri değiştirilebilir.

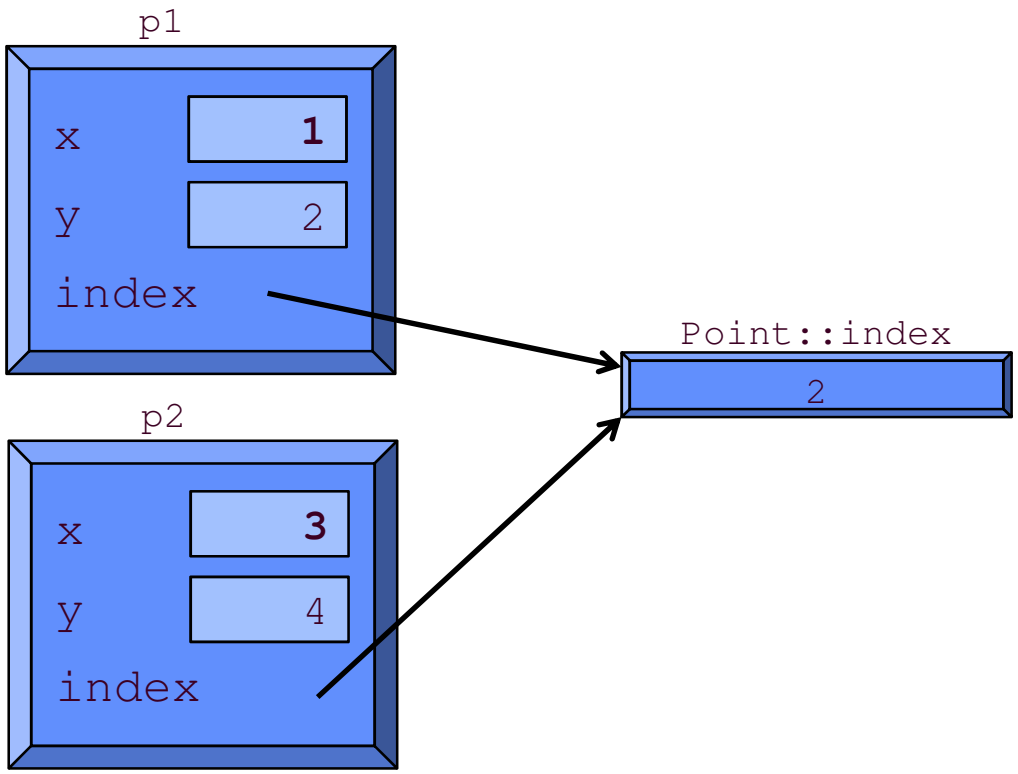
- `Point p1(1,2,1);` komutu işletildiğinde




```
#include <iostream>
using namespace std;
class Point{
            int x;
            int y;
            static int index;
public:
            Point(int X=0,int Y=0,int i=0)
            :x(X),y(Y),index(i){}
            void print() const{
                cout <<"("<<x<<","<<y<<",";
            }
            static void setIndex(int i){
                index=i;;
            }
};
int Point::index=0;
int main(){
            Point::setIndex(5);
            Point p1(1,2,1);
            Point p2(3,4,2);
            Point p3(5,6,3);
            return 0;
}
```

Statik (static) Sınıf Üyeleri

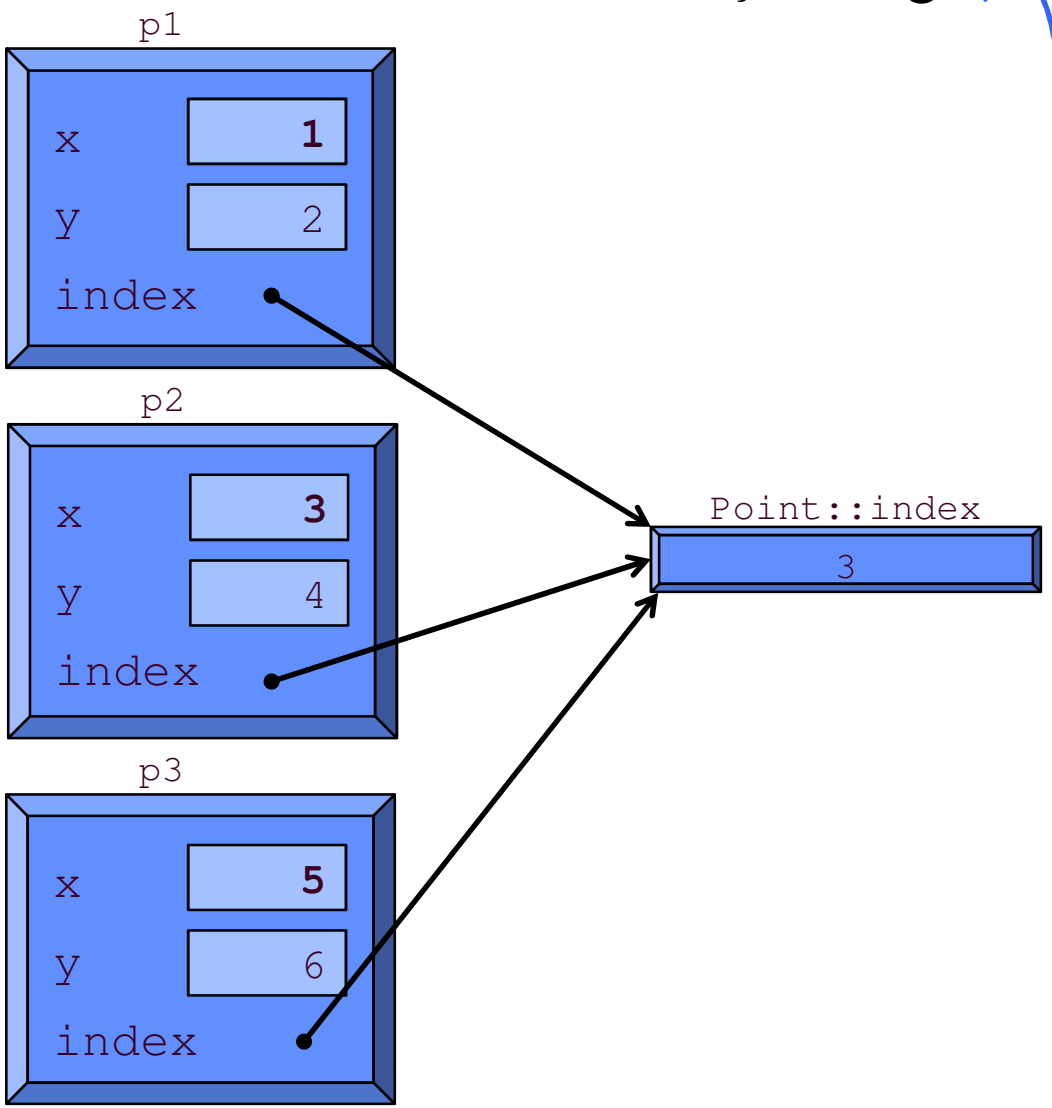
- `Point p2(3,4,2);` komutu işletildiğinde




```
#include <iostream>
using namespace std;
class Point{
            int x;
            int y;
            static int index;
public:
            Point(int X=0,int Y=0,int i=0)
            :x(X),y(Y),index(i){}
            void print() const{
                cout <<"("<<x<<","<<y<<",";
            }
            static void setIndex(int i){
                index=i;;
            }
};
int Point::index=0;
int main(){
            Point::setIndex(5);
            Point p1(1,2,1);
            Point p2(3,4,2);
            Point p3(5,6,3);
            return 0;
}
```

Statik (static) Sınıf Üyeleri

- `Point p3(5,6,3);` komutu işletildiğinde




```
#include <iostream>
using namespace std;
class Point{
            int x;
            int y;
            static int index;
public:
            Point(int X=0,int Y=0,int i=0)
            :x(X),y(Y),index(i){}
            void print() const{
                cout <<"("<<x<<","<<y<<",";
            }
            static void setIndex(int i){
                index=i;;
            }
};
int Point::index=0;
int main(){
            Point::setIndex(5);
            Point p1(1,2,1);
            Point p2(3,4,2);
            Point p3(5,6,3);
            return 0;
}
```

Statik (static) Sınıf Üyeleri – Employee Sınıfı

```
// Employee.h
// Employee class definition with a static data member to
// track the number of Employee objects in memory
#ifndef EMPLOYEE_H
#define EMPLOYEE_H

#include <string>
using namespace std;

class Employee
{
public:
     Employee( const string &, const string & ); // constructor
     ~Employee(); // destructor
     string getFirstName() const; // return first name
     string getLastName() const; // return last name

     // static member function
     static int getCount(); // return number of objects instantiated
private:
     string firstName;
     string lastName;

     // static data
     static int count; // number of objects instantiated
}; // end class Employee

#endif
```

Statik (static) Sınıf Üyeleri – Employee Sınıfı

```
// Employee.cpp
// Employee class member-function definitions.
#include <iostream>
#include "Employee.h" // Employee class definition
using namespace std;

// define and initialize static data member at global namespace scope
int Employee::count = 0; // cannot include keyword static

// define static member function that returns number of
// Employee objects instantiated (declared static in Employee.h)
int Employee::getCount()
{
    return count;
} // end static function getCount

// constructor initializes non-static data members and
// increments static data member count
Employee::Employee( const string &first, const string &last )
                          : firstName( first ), lastName( last )
{
        ++count; // increment static count of employees
```

Statik(static) Sınıf Üyeleri – Employee Sınıfı

```
    cout << "Employee constructor for " << firstName
    << ' ' << lastName << " called." << endl;
} // end Employee constructor

// destructor deallocates dynamically allocated memory
Employee::~Employee()
{
    cout << "~Employee() called for " << firstName
        << ' ' << lastName << endl;
    --count; // decrement static count of employees
} // end ~Employee destructor

// return first name of employee
string Employee::getFirstName() const
{
    return firstName; // return copy of first name
} // end function getFirstName

// return last name of employee
string Employee::getLastName() const
{
    return lastName; // return copy of last name
} // end function getLastName
```

Statik(static) Sınıf Üyeleri – Employee Sınıfı

```
// main.cpp
// static data member tracking the number of objects of a class.
#include <iostream>
#include "Employee.h" // Employee class definition
using namespace std;

int main()
{
    // no objects exist; use class name and binary scope resolution
    // operator to access static member function getCount
    cout << "Number of employees before instantiation of any objects is "
         << Employee::getCount() << endl; // use class name

    // the following scope creates and destroys
    // Employee objects before main terminates
    {
        Employee e1( "Susan", "Baker" );
        Employee e2( "Robert", "Jones" );
        // two objects exist; call static member function getCount again
        // using the class name and the binary scope resolution operator
        cout << "Number of employees after objects are instantiated is "
             << Employee::getCount();
```

Statik(static) Sınıf Üyeleri – Employee Sınıfı

```
        cout << "\n\nEmployee 1: "
             << e1.getFirstName() << " " << e1.getLastName()
             << "\nEmployee 2: "
             << e2.getFirstName() << " " << e2.getLastName() <<
        "\n\n";
    } // end nested scope in main

    // no objects exist, so call static member function getCount again
    // using the class name and the binary scope resolution operator
    cout << "\nNumber of employees after objects are deleted is "
        << Employee::getCount() << endl;
} // end main
```

Statik (static) Sınıf Üyeleri – Employee Sınıfı

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

Employee 1: Susan Baker
Employee 2: Robert Jones

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

Number of employees after objects are deleted is 0
```

Kaynaklar

- T.C. Lethbridge and R. Laganiere, Object-Oriented Software Engineering - Practical software development using UML and Java, McGraw Hill, Second Edition, 2005.
- H.M.Deitel and P.J.Deitel, C++ How To Program, 9E, Pearson Press, 2014.
- B. Stroustrup, The C++ Programming Language, 3rd Edition, Special Edition, Addison Wesley, 2000.
- Dr. Feza Buzluca, Ders Notları.
- Ç. Turhan ve F.C. Serçe, C++ Dersi: Nesne Tabanlı Programlama, 2nci Baskı, 2014.