

* Nested Namespace: → namespace içinde namespace

```
namespace A::B::C {  
}
```

```
namespace A::B::C {  
    int x;  
}  
  
namespace A::B {  
    int ival;  
}  
  
namespace A {  
    int* p;  
}  
  
int main()  
{  
    A::p = nullptr;  
    A::B::ival = 45;  
    A::B::C::x = 10;  
}
```

* Inline Namespace:

```
namespace nec {  
    inline namespace erg {  
        int x, y;  
    }  
    // using nec::x → inject alternate.  
    // using namespace erg → visibility  
}  
  
int main()  
{  
    nec::x = 10;  
}
```

Böylece, bu isimlerin
referansları isimler, onu
kapsayan namespace'de
gördür.

```

namespace A:::B:: inline C{
    int x = 10;
}

int main()
{
    A::B::C::x = 34;
    A::B::x = 34;
}

```

- inline funktion → using namespace kann nicht
a lternativer. Amac garantie . Fakat !!)
- Nested Namespace te, using namespace → ADL error
in line namespace → ADL works

* Nested Types

- using / typedef / enum

```

struct Word {
    short x, y;
};

class Myclass {
    Word m_a; → struct word (short)
    typedef int Word;
    Word m_b; → typedef word (int)
};

```

```

struct Word {
    short x, y;
};

class Myclass {
    void func()
    {
        Word wx; } →
    typedef int Word;
};

Fakturda wrt int.
Corre class scope random or. fine

```

```

struct Word {
    short x, y;
};

class Myclass {

    Word a; → short
    void func()
    {
        Word b; → int
    }

    typedef int Word;

    Word c; → int
};

```

```

class Myclass {
public:
    class Nested {
private:
    void func();
};

friend class Myclass;
};

void foo()
{
    n.func();
}

private:
Nested n;
};


```

Sanırsız olmasınca namer, nested type'ının private'ına erişmemeli.

nested torden bir private data member'i.

```

//Dikkat C++11 önce
class Myclass {
private:
    static int x;
public:
    class Nested {
        void foo()
        {
            x = 5;
        }
    };
};


```

Static birde içinde, private variable erişimde sorun olmaz.

+ İçinden dışarıya
private yoktur.

* Necə Burada Bunu giblər sey vaptı *

```

4 class Myclass {
5     private:
6         class Nested {
7             ;
8     };
9     public:
10        static Nested foo();
11    };
12
13 int main()
14 {
15     auto x = Myclass::foo()
16 }


```

→ Normalde bu erism legal olardı
→ auto type deduction ile Nested adını kullanmadığı için, access kontrol'e takıldı.

- Nested type'i return eden bir fonksiyon'u tanımlarken basına enclosing class name eklenmel!

- Bu fonksiyon, enclosing class içinde, tüm metodlar

* Composition: (Contourment):

→ Composition yorumlu türm → class tarafından daa member
→ handle pointer Tie konstruktor / null when destruktor

```

5 class Engine {
6     public:
7         void start();
8         void stop();
9     };
10
11
12 class Car {
13     public:
14         void start()
15         {
16             eng.start();
17         }
18     private:
19         Engine eng;
20     };
21
22 int main()
23 {
24     "Car Mycar
25     Mycar.start() → Syntax Notes"

```

Föret bu sınıfı, elementin ilişkisi indirip, kullanımını

*Föret bu sınıfın
dogrudan erişme istenirken
vermektedir.*

*her arabanın
bir motora var, "has a" relationship*

I

```
#include <vector>

class istack {
public:
    bool empty()const
    {
        return ivec.empty();
    }
    void push(int val)
    {
        ivec.push_back(val);
    }
private:
    std::vector<int> ivec;
};
```

Vector Interface intkuland!

```

class Engine {
private:
    void set_xyz();
    friend class Car; // eger friend tanimini verirmisseydi
};

class Car {
public:
    void carfunc()
    {
        eng.set_xyz(); ]> bu ogren legal dimdi.
    }
private:
    Engine eng;
};

```

Cunki set_xyz privatefonksiyonu.

* Hatalar:

* Class'a sahip olmak, sahip olunan class'ın
private bolümüne erişebiliriz onlarda
BİLMİZ!

```

int main()
{
    std::cout << "main basladi\n";
    {
        Car mycar;
    }

    std::cout << "main devam ediyor\n";
}

```

main basladi

Engine default ctor → scope'a giren elementler konumlanır.

Car default ctor

Car destructor → scope sonunda car destruct

Engine destructor → scope sonunda, elementler obiectet.

main devam ediyor

* Hatalar: • Eger constructor'ı compilera default ettirsem, övers elementlerin hepsi default init eder.

↳ Yani referans ve const memberler

de default init edilmeye calistir, syntax error olur.

→ Eger primitive type me in determinate value

→ Eger class ise o class'in default constructor'

```

class Student {
public:
    Student(const char *pname, const char *psurname) → Doğru kullanım: Default init mde
    {                                              List kullanılmış!
        m_name = pname;
        m_surname = psurname;                    ↗ Bu modülef bir kullanım
    }                                              ↗ Aşında 2. kez değer atadır. ↗ Bu modülef
                                                class'in önb. da atanmış.
private:
    std::string m_name, m_surname;
};                                                 ↗ Gerek bu private var elementleri, default constructor çağrılıdığında
                                                Değer atılarası nesne gelir

```

```

#include <iostream>

class Member {
public:
    Member(int);
};

class Owner {
public:
    Member mx;
};

```

* Owner'in default ctor Durumu: deleted

- Gerek Owner'in default constructor, yani her var elementi olan mx'de default construct edilir. ! Friend member'in default ctor'u temelli değil! → Direct sendox hatalı

- Bu yozden deleted

```

class Member {
public:
    Member(int);
};

class Owner {
public:
    Owner() I: mx{10} {}           ↗ Default Ctor
                                    ↗ Ctor init list
                                    ↗ ne
private:
    Member mx;
}; // Member mx {10} → In class
    // initialization

```

Doğru kullanım bu şekilde olmalı.

→ Alternatif olarak: In class initializ.

* Eğer Derleyici `copy constructor`ı default ederse, Copy İsteklerinde memberlerin da `copy ctor`ı çağrılır.

```
nestd_02.txt inline_namespace_01.txt nested_01.txt (Global Scope)

Member() = default;
Member(const Member&)
{
    std::cout << "Member copy ctor\n";
}

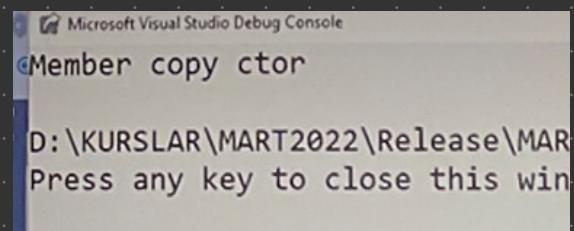
};

class Owner {
public:

private:
    Member mx;
};

int main()
{
    Owner x;
    Owner y = x;
}
```

- Member'a bir kırıtkırsızlık istenildiği için, member copy constructor ile çağrılır.



```
Microsoft Visual Studio Debug Console
Member copy ctor
D:\KURSLAR\MART2022\Release\MAR
Press any key to close this window
```

/ Fakat, Copy Ctor'ı Default ettiğimiz yerine, kırıtkırsızdır → Elemanın kopyalanmasından kırıtkırsızlaşır!

```
class Owner {
public:
    Owner() = default;
    Owner(const Owner& other) : mx(other.mx) / :mx(other.mx)
    {
        std::cout << "Owner copy ctor\n";
    }
private:
    Member mx;
};

int main()
{
    Owner x;
    Owner y = x;
}
```

↓
Copy init. list

member default ctor çağırılır
Once member default ctor
Sonra Owner copy ctor

değiş! +
değer almazısa da
kırıtkırsızlaşır!

* Derleyicinin yediği move ctor:

```
class Member {  
};  
  
class Owner {  
public:  
    //other special member function  
    Owner(const Owner& other) : mx(other.mx) {} → Derleyicinin copy Ctor  
    Owner(Owner&& other) : mx(std::move(other.mx)) {}  
private:  
    Member mx;  
};
```

- Eğer std::move (r value) dimesi, copy ctor
 çağırır.

* Copy memberlerin delete edildiği zaman:

```
class Member {  
public:  
    Member(const Member&) = delete;  
    Member& operator=(const Member&) = delete;  
    Member(Member&&);  
    Member& operator=(Member&&);  
};
```

→ Kopyalanmaz Anna teslimabilir.

* Sınıfın Vendi Térindən Vər elementi olmaz:

```
class Myclass {  
  
private:  
    static Myclass x;|  Folot static Vər element  
};
```