

## # Mini Tekrar:

- Hem composition, hem inheritance da nesne içinde başka bir nesne var
- Derived class hayata gaurun onun içindeki base class da hayata gaur. önce derived içindeki base

↓  
Sonra derived içindeki member  
↓  
Sonra derived ctor çağılır.

```
12 class Base
13 {
14 public:
15     Base()
16     {
17         std::cout<< "Base constructor\n";
18     }
19     ~Base()
20     {
21         std::cout<< "Base destructor\n";
22     }
23 };
24
25
26
27 class Der : public Base
28 {
29 public:
30     Der()
31     {
32         std::cout<< "Der constructor\n";
33     }
34     ~Der()
35     {
36         std::cout<< "Der destructor\n";
37     }
38 };
39
40
41
42 int main()
43 {
44     Der myDer;
45     return 0;
46 }
```

Base constructor  
Der constructor  
Der destructor  
Base destructor

Sırası önemli, sonra gelen ile olur.

- Sıra olarak 1. eğer varsa base member
- 2. base ctor
- 3. eğer varsa derived member
- 4. derived ctor
- 5. ve tam tersi şekilde destructor

```
class Base {
public:
    Base(int x) {
        std::cout << "Base(int x) x = " << x << "\n";
    }
};

class Der : public Base {
public:
    Der(int i) : Base(i) {
    }
};

int main()
{
    Der myder(12);
}
```

Burada Base default ctor değil

→ kome parentı ile olurdu Base()

→ Der ctor böyle olmalı. yoksa syntax hatası

→ Böyle olmazsa

Der'in default ctor'u girmez.

I

### • Multi level Inheritance:

```
class Base {
public:
    Base(int);
};

class Der : public Base {
public:
};

class SDer : public Der {
public:
    SDer(): Base(0) {}
};
```

SDer, direkt olarak Base'i böyle  
init eder.

### • Copy Constructor:

```
class Base {
public:
    Base()
    {
        std::cout << "Base default ctor\n";
    }
    Base(const Base&)
    {
        std::cout << "Base copy ctor\n";
    }
};
```

```
class Der : public Base {
    → Derived class'ın copy constructor'ını  
    compiler default etti.
};
```

```
int main()
{
    Der der1;
    Der der2 = der1;
}
```

Microsoft Visual Studio Debug Console

```
Base default ctor
Base copy ctor
D:\KURSLAR\MART2022\Release\MART2022.e
Press any key to close this window . .
```

• Fakat: Biz kendi elimizde  
Der için copy ctor yazmadık  
işler değişir.

↓  
Bu durumda der, copy  
construct edilse bile, Default  
ctor çağırır!!



## • Eger Copy Constructor, Derived Tem Bir uygulamak:

```

19 class Der : public Base {
20 public:
21     Der() = default;
22     Der(const Der&other) : Base(other) {
23     {
24     }
25 }
26
27 };
28
29
30 int main()
31 {
32     Der der1;
33
34     Der der2 = der1;
35 }

```

Bu şekilde elemanı tanımlı

## • Move Constructor: → Copy etekleri derunun aynı, move etekleri de geçerli

```

class Base {
public:
};

class Der : public Base {
public:
    Der() = default;
    Der(Der&&other) : Base(std::move(&_Arg:other)) {
    //
    }
};

int main()
{
    Der der1;
    Der der2 = std::move(&_Arg:der1);
}

```

bu şekilde elemanı tanımlı

## • Copy Assignment:

```

class Base {
public:
    Base& operator=(const Base&)
    {
        std::cout << "Base copy assignment\n";
        return *this;
    }
};

class Der : public Base {
public:
    Der& operator=(const Der&other)
    {
        Base::operator=(other);
        return *this;
    }
};

int main()
{
    Der x, y;
    x = y;
}

```

bu şekilde elemanı tanımlı

• Her bir şekilde base belirtilmemişse, base copy assignment böyle uygulanabilir.

→ x=y ikisi de Der türünde

• Alternatif olarak şöyle bir upcast uygulanır.

```

class Der : public Base {
public:
    Der& operator=(const Der&other)
    {
        *(Base*)this = other;
        return *this;
    }
};

```



→ Çok genel bir özellik için olan.

### \* Move Assignment:

```
public:
    Base& operator=(Base&&)
    {
        std::cout << "Base move assignment\n";
        return *this;
    };

class Der : public Base {
public:
    Der& operator=(Der&& other)
    {
        Base::operator=(std::move(other));
        return *this;
    }
};
```

### → Notlar:

eğer şu 3 işlevden biri bildirilmiş ise derleyici sınıfın move member'larını yazmaz

copy ctor  
move ctor  
destructor

eğer şu 2 işlevden biri bildirilmiş ise derleyici sınıfın copyt member'larını delete eder

move ctor  
move assignment

### \* Run-Time Polymorphism:

- concrete class → somut sınıf
- abstract class → soyut sınıf } → base class interface verir. } → Biz source yazmak zorundayız.
- polymorphic class } → base class interface verir. Biz istersek override ederiz.  
kod.

```
class Airplane {
public:
    void takeoff();
    virtual void land(); //virtual function
    virtual void fly() = 0; //pure virtual function
};
```

bu şekilde tanımlanmış olan  
1 tane base class  
ve polymorphic class

bu şekilde tanımlanmış olan  
1 tane base class  
ve abstract class



## • Function Overriding:

→ Eğer aynı fonksiyonun çağırıldığı run time da belirlenirse **late binding**. Function override temamen late binding

```
class Base {  
public:  
    virtual void vfunc();  
};
```

```
class Der : public Base {  
public:  
    void vfunc(); } → Aynı parametre ve return  
                        değeri olmalı
```

• const bildirimi de imzanın parçası **override** eden de const kullanmalı.

→ Deriv class'ın vfunc'ini virtual olarak tanımladığı da tanımlanmış olan override eder.

→ Belirtilmek zorunda değil.

## → Virtual Dispatch:

• Taban sınıf pointer, türetilmiş sınıf nesnesini gösterebilir referans. " " " bağlanabilir.

```
class Der : public Base {  
public:  
    void vfunc()  
    {  
        std::cout << "Der::vfunc()\n";  
    }  
};  
  
int main()  
{  
    Der myder;  
  
    Base* baseptr = &myder;  
  
    baseptr->vfunc(); } → Bu derived class'ın / compiler  
                        garanti ediyor.
```

```
int main()  
{  
    Der myder;  
  
    Base& baseref = myder; } → referans semantikle  
                                bu şekilde  
  
    baseref.vfunc();  
}
```

• Fakat fonksiyon, base nesnesinin kendisiyle çağırılırsa **virtual dispatch** denir.

→ virtual dispatch sadece pointer ve referansla yapılır.

→ bu olaya object string'e sahip veri.