

* Özet / Hatırlatma:

→ compiler special member fonksiyonları **public**, **static inline** olarak **default** eder.

→ Örnek:

```
class MyClass {
public:

private:
    MyClass() = default;
};
```

→ private kısmına da tanımlamak
private default ettirebiliriz.
→ Bu tüm s.m.f'ler için geçerli.

→ Geçici Nesne / Temporary Objects:

→ Value kategorisi olarak **pr value** fonksiyon **kodda adı olmayan** objekt

```
class MyClass {
};

int main()
{
    MyClass{}
}
```

→ pr value
→ pr value'nin içinde

→ T → pr
T & → L value
T && → x value

→ Geçici nesneler ile **member fonksiyonlar** çağrılabilir.

```
class MyClass {
public:
    MyClass(int, int);
    void foo();
    void func()const;
};

int main()
{
    MyClass{ 12, 45 }.foo();
    MyClass{ 12, 45 }.func();
}
```

→ Geçici nesneler **pr value expr** olduğuna göre parametreleri → const class f
→ L value (MyClass &&) → Kontrolün
→ class türünden → Call by value

→ Geçici nesne ile **isimiz bilince** **destructor** **tekrar** çağrılır.

```
class Date {
public:
    Date(int day, int mon, int year);
};
```

```
void func(const Date&);
```

```
int main()
```

```
{
```

```
    Date mydate{ 16, 5, 2022 }; }
    func(mydate);
```

```
}
```

→ Yanında func fonksiyonu için,
yapının bir ağı

→ Scope leakage'a sebebiyet
verir. Çünkü, func fonksiyonu
dışında da kullanılabilir.

→ Scope leakage zararları:

— myDate, kaynak tutulmuş, destructor:
Çağrılmadığı için kaynakları geri verilemiyor

→ Bir referans bind edilirse, o referans scope'un sonuna kadar life time extend edilir. Eğer bind edilmesse olmaz.

→ const L value &
R value ?

* Copy Elision:

→ Derleyici koda bakarak, gereksiz kopyalama işlemlerinden kaçınmak için, optimizasyon yapar / kopyalama dışı kalır.

→ C++ 17 standardı: • Derleyici optimizasyonculu → yapmaya zorunlu değildir.

→ Copy Elision 3 durumda gerçekleşir 1. Sınıf nesnesi temp object / R value ile kopula gelince

2. Fonksiyonun return değeri temp object ise

* Parametresi bir sınıf türünden olan fonksiyona bir R value Expr. ile çağrı yapılırsa, copy elision olamaz olur.

→ mandatory elision

* R value optimization'a bak.

Conversion Constructor: → Dönüştürme Kurucu İşlev

* Bu bir special member function DEĞİL

```
class MyClass {  
public:  
    MyClass() = default;  
    MyClass(int);  
};  
  
int main()  
{  
    MyClass m;  
  
    m = 10;  
}
```

Bu olur çağılır. int'ten değeri MyClass'te type conversion yapar.

Class nesnesi int

→ Sınıfın tek parametresi constructor, aynı zamanda tek dönüştürme tan da kullanılır.

→ Değerler tek parametresi olan parametresiz türden bir sınıfa dönüştürülmesi durumunda ilk önce olan sınıf nesnesi oluşturulur.

*Function Overloading Notasyonu: • Eğer bir conversion, normal olarak yok ise fonksiyonun kullanılmasıyla bu dönüşüm

yapılıyorsa bu → user-defined conversion.

⇒ a = b farklı tür.

, farklı olarak, veri kaybolmadan/alınarak yapılan dönüşüm = standard conversion

→ Compiler tek dönüşüm yaparken eğer sunarla yapılmıyorsa standard conversion + user defined conversion } Bu tek dönüşümün gerçekleştirilmesidir.

```
MyClass(int x)  
{  
    std::cout << "MyClass(int x) x = " << x << " this: " << this << "\n";  
}  
};  
  
int main()  
{  
    MyClass m;  
    (void) getchar();  
    m = 1.5;  
    std::cout << "main devam ediyor\n";  
}
```

double to int standard conversion
+
int to MyClass user defined conversion.

Bu yüzden m = 1.5 da geçerlidir.

* Float: user defined + user defined conversion UYUMLAMAZ.


```
void func(Myclass);
void foo(const Myclass&);
```

const L value ref

```
Myclass bar()
```

```
{
    return 10;
}
```

```
int main()
```

```
{
    Myclass m1 = 10;
    Myclass m2(10);
    Myclass m3{10};
```

```
func(10);
```

```
foo(20);
```

prvalue → const L value ref' e kind edit.

#Explicit Constructor:

→ Implicit conversion 'a engel den ve yalnizca explicit type conversion ise conversion yapar.

```
class Myclass {
public:
    explicit Myclass(int);
};
```

→ explicit conversion yapilmadan conversion mümkün degil!!

```
int main()
```

```
{
    //Myclass m1 = 10; → copy init hata
    Myclass m2(10); → direct init ✓
    Myclass m3{10}; → brace init ✓
}
```

```
class MyClass {  
public:  
    explicit MyClass(int);  
    MyClass(double);  
};
```

→ Function overload
resolution'a girer → 02.23 ..

```
int main()  
{  
    MyClass m = 10;  
}
```

Syntax hatası yok



* Static Data Members:

→ Her bir class için object sayısı kadar değil, yalnızca 1 adet.

→ Global var vs. Static

global → namespace scope

static → class scope