*Özet / Hatırlatma:

→ Compiler special member functionları public, static inline olarak default eder.

→Fakat:

```
class Myclass {
public:

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

→ private kısmına da tanımlarak private default ettirebiliriz.

→ Bu tüm s.m.f'ler için geçerli.

→ Geçici Nesne / Temporary Objects:

→ Value kategori olarak PR value fakat kodda adı olmayan objeler

```
class Myclass {

};

int main()
{
    Myclass{}
}
```

→ pvoot.h'ı incele

→ Pr value

→ T → pr
T & → L value
T && → x value

→ Geçici nesneler ile member fonksiyonlar çağırılabilir.

```
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 parametreler → const class &
→ & value (myClass &&)
→ class türünden
→ Const lvalue ref
Call by value

→ Geçici nesne ile isimsiz bildirim deyimleri kodu çağırılır.

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


void func(const Date&);

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


}
```

→ Yanızca func fonksiyonu için, yaratılmış bir objedir

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

→ Scope leakage'ın Zararları:

— myDate, kaynak tutsaydı, destructor çağırılmadığı için kaynakları geri veremeyecek

—

→ Bir referans bind edilirse, o referansın scope'unun sonuna kadar life time extend edilir. Eğer bind edilmezse yıkılır.

→ const L value &
R value &&

* Copy Elision:

→ Derleyici kodu tarayıp, gereksiz kopyalama işleminden kaçınmak için, optimizasyon yapar / kopyalama devre dışı kalır.

→ C++ 17 öncesinde: • Derleyici optimizasyonuydu → yapmaya zorunlu değildir.

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

2. Fonksiyonun return değeri temp object ise

* Parametresi bir sınıf türünden olan fonksiyona bir PR value Expr. ile çağrı yapılırsa, copy elision devreye girer.

→ mandatory elision

*Return value optimizationlara bak.

Conversion Constructor: → Dönüştüren Kurucu İşlev

* Bu bir special member function DEĞİL

```
class Myclass {
public:
    Myclass() = default;
    Myclass(int);
};

int main()
{
    Myclass m;

    m = 10;
}
```

Bu ctor çağrılır. int'ten değerin MyClass türüne type conversion yapar.

m → Class nesnesi

10 → int

→ Sınıfın tek parametreli constructor'ı aynı zamanda tür dönüşümü için de kullanılır.

→ Derleyici tek parametreli ctor parametre türünden bir sınıf türüne dönüştürülmesi durumunda ilgili value olan sınıf nesnesi oluşturur.

Function Overloading Hatırlatma: • Eğer bir conversion, normal olarak yok ise fakat yazdığımız bir fonksiyonun kullanılmasıyla bu dönüşüm yapılıyorsa bu → User-defined conversion.

=> a = b (farklı tür.)

, örtülü olarak, nesne kopyası olan/olmasın yapılan dönüşüm = standard conversion

→ Compiler tür dönüşümü yaparken eğer sınırlı yapılmışsa standard conversion + user defined conversion
user defined conversion + standard conversion → Bu tür dönüşümünü gerçekleştirir.

```
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 de legaldir.

* Fakat: user defined + user defined conversion YAPILAMAZ.

```
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 bind edilir.

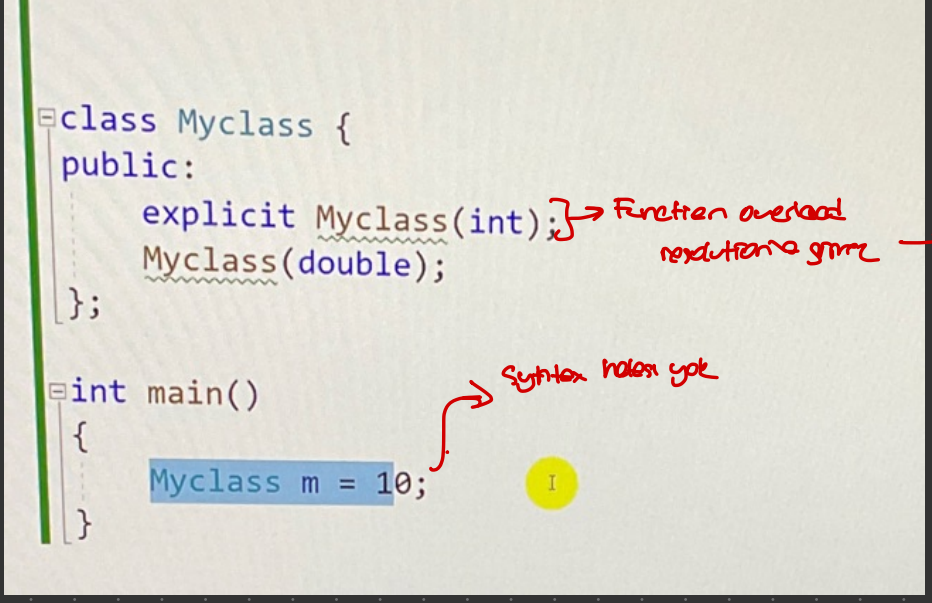
Explicit Constructor:

→ Implicit conversion'a engel olan ve yalnızca explicit type conversion'a izin veren constructor yapar

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

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

explicit conversion yapılmadan conversion mümkün değil!!



* Static Data Members:

→ Hayatta olan object sayısı kadar değil, yalnız 1 adet.

→ Global var vs. Static

global → namespace scope
static → class scope