- Tekrar: • Special member functionlar 3 durumdan 1'inde olabilir.
  - ↳ Yoktur
  - ↳ User Declared → to be defaulted
    → to be deleted
  - ↳ Implicitly Declared → to be defaulted
    → to be deleted incase of syntax err.
- Default cons. dışında, destructorın yanısı en iyisi

- How Did Move Semantics Get Started - Slayt !!

Special Members

compiler implicitly declares

	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted

user declares

- If the user declares no special members or constructors, all 6 special members will be defaulted.
- This part is no different from C++98/03

Special Members

compiler implicitly declares

	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted

user declares

- "defaulted" can mean "deleted" if the defaulted special member would have to do something illegal, such as call another deleted function.
- Defaulted move members defined as deleted, actually behave as not declared.

Special Members

compiler implicitly declares

	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted

user declares

- If the user declares any non-special constructor, this will inhibit the implicit declaration of the default constructor.

Special Members

compiler implicitly declares

	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
destructor	defaulted	user declared	defaulted	defaulted	not declared	not declared

user declares

- A user-declared destructor will inhibit the implicit declaration of the move members.
- The implicitly defaulted copy members are deprecated.
  - If you declare a destructor, declare your copy members too, even though not necessary.

Special Members

compiler implicitly declares

user declares	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
destructor	defaulted	user declared	defaulted	defaulted	not declared	not declared
copy constructor	not declared	defaulted	user declared	defaulted	not declared	not declared
copy assignment	defaulted	defaulted	defaulted	user declared	not declared	not declared
move constructor	not declared	defaulted	deleted	deleted	user declared	not declared
move assignment	defaulted	defaulted	deleted	deleted	not declared	user declared

- Destructor ya declared ya da defaulted her zaman
- Move member yazarsak, copy memberlar delete edilir. → Yani onlarını istiyorsak default etmesini isteyeceğiz.
- Bazı Classlar non-copyable but moveable dır
  copyable and moveable
  non-copyable and non-moveable

```
#include <string>


int main()
{
    using namespace std;

    string s1;
    //
    string s2 = s1; →copied
    string s3 = std::move(s1); → moved
}
```

⇒ copyable and moveable

R value

→ Elemanların int, double, array ise taşımada kopyalama arasında fark olmaz.

↳ Move'un faydası bir - handle
- pointer 'ta gibi.
- reference

* Dinamik Ömürlü Nesneler:

- Programın dilediği noktada hayatını başlatıp / bitireceği nesneler.
- Storageları heapte → run time'da verilir. Maliyeti arttırır.
- Kodun yazılıp okunması zor.

- new operatörlerinden biri ile hayata gelir.
  delete operatörlerinden biri ile son bulur.

- New yeni bir adres üretir. Üretilen adres hayata gelen nesnenin adresidir. → Bu pointerdır!!

> → Mini Anlatım: Smart Pointers:
>
> - O sınıf türünden değişkenler, pointer gibi davranırsa → pointer like classes
>
> ↓
>
> Dinamik ömürlü classların hayatını kontrol eden pointer like classes ⇒ Smart Pointers!!!

```
int main()
{
    auto p1 = new Myclass;
    Myclass *p2 = new Myclass;
    auto *p3 = new Myclass;

    delete p1;
    delete p2;
    delete p3;
}
```

→ Hepsi aynı anlama geliyor. (p1, p2, p3 tanımları)

→ böyle delete edilir. hayatı sonlandırılır. (delete satırları)

* Malloc'tan farklı olarak malloc başaramazsa → returns Nullptr
new başaramazsa → throws exception

→ new, operator new çağırır. →

```
void *operator new(std::size_t)
```

(std::size_t → sizeof(MyClass))

→ Eğer delete edilmezse → destructor çağrılmaz

↳ Raii bir nesne resource leak'e neden olabilir. → Raii: Resource acquisition is initialization.

↳ Ayrılan memory verilmez, memory leak oluşur.

```
}

class Myclass {
public:
    Myclass()
    {
        std::cout << "Myclass default ctor. this : " << this << "\n";
    }

    ~Myclass()
    {
        std::cout << "Myclass destructor. this : " << this << "\n";
    }

private:
    unsigned char m_buffer[1024]{};
};
```

yani construct edilecek adresin değeri

```
int main()
{
    std::cout << "sizeof(Myclass) = " << sizeof(Myclass) << "\n";

    auto p = new Myclass;
    std::cout << "p = " << p << "\n";

    delete p;
}
```

```
void* operator new(std::size_t n)
{
    std::cout << "operator new called! n = " << n << "\n";
    void* vp = std::malloc(n);
    if (vp == nullptr) {
        throw std::bad_alloc{};
    }
    std::cout << "the addres of allocated block is : " << vp << "\n";

    return vp;
}
```

Bu vp, bizim classımızın this'i oluyor.

* Array New Operatörü:

- Birden fazla dinamik ömürlü nesneyi oluşturur.

```
int main()
{
    std::cout << "kac elemanli bir dizi: ";
    int n;

    std::cin >> n;
    new Myclass[n];
}
```

n adet Myclass oluşur,
n adet Myclass constructor çağırılır.

```
        std::cout << "Myclass destructor. this : " << this << '\n';
    }
private:
    unsigned char m_buffer[256]{};
};


int main()
{
    std::cout << "kac elemanli bir dizi: ";
    int n;

    std::cin >> n;
    Myclass* p = new Myclass[n];
}
```

```
Microsoft Visual Studio Debug Console
kac elemanli bir dizi: 5
Myclass default ctor. this : 00DA206C
Myclass default ctor. this : 00DA216C
Myclass default ctor. this : 00DA226C
Myclass default ctor. this : 00DA236C
Myclass default ctor. this : 00DA246C

D:\KURSLAR\MART2022\Debug\MART2022.exe (process 26700) exited with code 0.
Press any key to close this window . . .
```

delete[]p → ile silinir.

Fakat constructor edildiği sıranın tersinde delete edilir.

* C'den Hatırlatma *

```
int main()
{
    int a[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

    for (int i = 0; i < 10; ++i) {
        a[i]  i[a]
    }
```

dereferans edecek hesaplıyor []

bu ikisi aynı anlama geliyor. Çünkü: *(i+a)