2023.09.18

Kalıtım Inheritance

```cpp
// Kalıtım

class Base
{
    int x {};
    int y {};
}
class Der : public Base // base object inheritance
{
    int a{};
    int b{};
    Base b // member object containment
}

int main()
{
    std::cout << "sizeof(Base)  = " << sizeof(Base) << "\n"; // 8
    std::cout << "sizeof(Der)  = " << sizeof(Der) << "\n"; // 16

    // Der classı Baseden inh edilidiği için onu size'nida kapsar
    // Her oluşan Der nesnesinin içinde Base object vardır
}
```

```cpp
class Base
{
    private:
        int mx;
        void foo();
    protected:
        int my;
        int func();
}
class Der : public Base
{
    void bar()
    {
        // syntax hatası var çünkü Base class'ın private bölümüne erişemeyiz
        foo()
        mx = 5

        // protected kısmına erişebiliriz
        my = 6;
        func();
    }
}
```

```
/*
    Dikkat
    Taban sınıfta ve türemiş sınıfta bildirilen aynı isimli fonksiyonlar
    overload değildir
*/
class Base
{
    public:
        void foo(int);
};

class Der : public Base
{
    public:
        void foo(double);
};
```

Kalıtım ve İsim Arama

```
class Base
{
    public:
        void foo();
};

class Der : public Base
{
    public:
        void foo(int);
};

int main()
{
    Der myder;
    myder.foo(); // hatalı
    /*
        İsim arama türemiş sınıf içinde bitti. Türemiş sınıfın içinde
        foo(int) fonksiyonunu derleyici bulur. Hatanın sebebi ise deallocate
        Gerekli int argumanı foo() fonksiyonuna göndermediğimizden kaynaklı

    */

    myder.foo(12); // Der sınıfının foo'su
    myder.Base::foo(); // Base class'ın foo'su
}
```

Türemiş sınıfın bir üye fonksiyonu içinde bir isim nitelenmeden kullanılırsa

- isim önce blokta
    - sonra kapsayan blokta
        - sonra türemiş sınıfın tanımında
            - sonra taban sınıfın tanımında
                - sonra namespace'de aranır

```
int x;
class Base
{
    public:
        int x;
};

class Der : public Base
{
    public:
        void foo(()
        {
            int x;
            auto y = x; // kapsamında içindeki x
            // Der deki x
            y = Der::x;
            y = this->x;

            // Base deki x
            y = Base::x
            //Global x
            y = ::x
        }

    int x;
};
```

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

class Der : public Base
{
    public:
        void foo(int)
        {
            foo(4); // recursive call
        }
};
```

Kalıtımda Using Bildirimi

```
class Base
{
    public:
        void foo(int);
        void foo(double);
};

class Der : public Base
{
    public:
        using Base::foo; // bu bildirim ile foo fonksiyonları aynı kapsamaa geldi
        void foo(long);
};

int main()
{
    Der myder;
    // using Base::foo bildirimiyle bütün foo() fonksiyonları overload olmuş oldu
    myder.foo(1.2); // Base::foo(double)
    myder.foo(1); // Base::foo(int)
    myder.foo(12L); //Der::foo(long)
}
```

```
// multi level inheritance
class A
{

};

class B : public A
{

};

class C : public B
{

};
// C açısından B 'ye baktığımızda direct Base  class (immediate Base class)
// C açısından A ise indirect Base class
```

```
// multi level inheritance
class A
{

};

class B : public A
{

};

class C : public B
{

};
// C açısından B 'ye baktığımızda direct Base  class (immediate Base class)
// C açısından A ise indirect Base class
```

Der türemiş bir sınıf olsun

Bir Der nesnesi hayata geldiğinde:

1) önce Der içindeki base class object hayata gelecek. (ctor çağırılacak)

2) Sonra bildirimdeki sırayla member object'ler hayata gelecek

3) sonra Der sınıfının ctor ana bloğuna girecek

Bir Der nesnesinin hayatı bittiğinde:

1) Der sınıfın dtor çağırılacak

2) En son bildirimden başlayarak member objectlerin dtor çağırılacak

3) Base sınıfın dtor çağırılacak

```
class Base
{
    public:
        Base(int); // durum 1
        Base() = delete; // durum 2
    private:
        Base(); // durum 3
};
class Der : public Base
{};
int main()
{
        Der myder; // hatalı
        /*
            Eğer sınıfın special member fonksiyonu derleyici implictyli
            declared edip default ederse ama yazacağı kodda syntax hatası olursa
            default etmesi gereken ctor delete eder.
            Yani Der class'nın ctor delete edilmiş durumda
        */
}
```

```
class Base
{
    Base(int x, int y);
    Base(double dval);
};

class Der : public Base
{
    Der() : Base{10 , 20}
    {

    };

    Der (double dval) : Base{dval}
    {

    };
};
```

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

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

class Der : public Base
{
    public:
        Der() : mx(20), Base(3.4) // Buradaki sıranın önemi yok
        {
            std::cout << "Der default ctor\n";
        }
    private:
        Member mx;
};
int main()
{
    Der myder;
}

/*
    1) "Base (double x) d = " yazısını görücez
    2) "Member(int x)  x = " yazısını görücez
    3) "Der default ctor\n" yazısını görücez

*/
```

```
class Base
{
    public:
        void foo();
};

class Der : public Base
{};

int main ()
{
    Der myder;
    Base *p = &myder;
    Base&r = myder;

    Base mybase = myder; // (object slicing) bu yapılmamalı

    myder.foo();
    /*
        Aslında burda foo()'Un içinde Base* argumanı alıyormuş gibi düşünüyor
        derleyici
    */
}
```

Kalıtımda special fonksiyonların durumları:

Copy and Move Ctor

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

        Base(Base &&)
        {
            std::cout << "Base move ctor\n";
        }
};

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

```
int main()
{
    // Base default ctor
    // Der default ctor
    // Base default ctor
    // Der copy ctor
    Der ader;
    Der bder(ader);
}
/*
    Eğer türemiş bir sınıf için  copy ctor yazarsak türemiş sınıfın copy
    ctor'unda Base sınıfın copy ctor çağrılmasından biz sorumluyuz.

        Der(const Der&) : Base() // derleyici Base'in default ctor çağrılrı
        {
            std::cout << "Der copy ctor \n";
        }

        Der(const Der&) : Base(other) // Base'in copy ctor çalışması için
        {
            std::cout << "Der copy ctor \n";
        }

        Move ctor içinde:
        Der(const Der&) : Base(std::move(other)) // Base'in move ctor çalışması
için
        {
            std::cout << "Der copy ctor \n";
        }

*/
```

Copy and move assigment

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

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

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

        Der& operator=(Der&& other)
        {
            Base::operator=(std::move(other));
            std::cout << "Der copy assignment \n";
        }
};

int main()
{
    Der d1, d2;

    d1 = d2;
    d1 = std::move(d2);
}
```