2023 08 18

Classes

```
class Myclass {
    public:
        Myclass(int); // conversion constructır
};
```

delete bildirimi

```
// delete bildirimi
void func(int) = delete; // bildirilmiş ama delete edilmiş
```

```
// sadece int parametreli func çağrılabilecek
template <typename T>
void func(T) = delete;

void func(int);

int main()
{
    func(2.3); // syntax hatası
}
```

Class Special Functions

- default ctor
- destructor
- copy ctor
- move ctor
- copy assignment
- move assignment

Classların özel fonksiyonlarının 3 durumu vardır:

1. not declared
2. user declared
    a. default
    b. delete
3. implicitly declared
    a. derleyici tarafından default
    b. derleyici tarafından delete

Derleyici default ctor default etmek zorunda ve derleyicinin yazdığı defult ctor class'ın data memberlarını default etmek zorunda. Aşağıdaki durumda derleyici int &r ve const int x default edemeyeceği için Myclass sınıfın default ctor'u delete edilmiş durumda

```
class Myclass
{
    public:
    private:
        int &r;
        const int x;
};

int main()
{
    // default ctor deleted
    Myclass m;
}
```

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

class Tamer {
    // Tamer sınıfın default ctor durumu: implicitly declared deleted
    private:
        Member mx;
};
int main()
{
    Tamer tx;
}
```

Aggregate Class

Bir aggregate class, yalnızca public veri üyelerini içeren ve özel üye fonksiyonları veya üye değişkenleri olmayan bir sınıftır. Bu, C++'ta bir struct veya bir class tanımı içinde yer alabilir.

```
class Myclass {
    public:
        int mx{};
        int my{};
};

static_assert(std::is_aggregate_v<Myclass>);
int main()
{
    Myclass m = {1, 2};
}
```

Copy Constructor

Eğer bir sınıf nesnesi hayata değerini aynı türden; bir başka sınıf nesnesinden alarak geliyor ise copy ctor kullanılır.

```
class Nec
{
    public:
        Nec()
        {
            std::cout << "Default ctor this : " << this << "\n";
        }

        Nec(const Nec&)
        {
            std::cout << "Copy ctor this = " << this << "\n";
        }
        ~Nec()
        {
            std::cout << "Destructor this : " << this << "\n";
        }
};

void foo(Nec)
{

}

int main()
{
    Nec mynec;
    std::cout << "&mynec = " << &mynec << "\n";

    foo(mynec);
    std::cout << "main devam ediyor\n";
}
```

```
class Nec {

};
int main()
{
    Nec x; // default ctor
    // copy ctor
    Nec n1 = x;
    Nec n2(x);
    Nec n3{x};
}
```

```
    class A{};
    class B{};
    class C{};

    class Myclass {
        public:
            Myclass() : ax(), bx(), cx()
            {

            }
            Myclass(const Myclass &other) : ax(other.ax), bx(other.bx),
cx(other.cx)
            {

            }
        private:
            A ax;
            B bx;
            C cx;
    };

    rule of zero
        sınıfı özel fonksiyonları derleyici tarafından yazılmasına denir.
```