18 Mayıs 2022

sınıfların static üye fonksiyonları

sınıfın static veri elemanları sınıfın elemanları ama instance'ın elemanları değil. Sınıf başına static eleman bir tane.
Assembly düzeyinden bakıldığında global değişkenlerden başka birşey değil.
Farkı :
1) class scope'a girer
2) isim arama kuralı değişik
3) access control'a dahil olur

Global değişkenlerin biraz islah edilmiş hali.

C++ 17 ile gelen değişiklik
sınıfın static elemanları inline sözcüğü ile sınıf içinde tanımlanabilir.

```
class Myclass{
public:
   inline static int x = 5; //  const. veya tam sayı olması zorunlu değil !!
};
```

// constructor init. list : sınıfın non-static elemanlarını init. eder.

```
class Myclass{
public:
    void func()
    {
        x = 5;
    }
private:
    static int x; //
};
```

zaten bildiğimiz üzere private static elemana client kodlar doğrudan erişemeyecek. Üye fonksiyon erişebilir.

There can not be multiple copies of same static variables for different objects. Also because of this reason static variables can not be initialized using constructors.
a static variable inside a class should be initialized explicitly by the user using the class name and scope resolution operator outside the class as shown below:

```
// myclass.h
class Myclass{
public:
    static int x;  // decleration
};

//myclass.cpp
int Myclass::x{10}

int main()
{
    std::cout << Myclass::x << "\n";
}
```

```
class Myclass{
public:
    Myclass() : x{10}   // yanlış !!!!
    {
        // non-static elemanları init eder
    }
private:
    static int x;
};
```

```
class Myclass{
public:
    inline static int x;
};

inline static Myclass::int x;  // yanlış
```

soru:

```
class Myclass{
public:
    void func()const  // const ne anlamda
    {
        // bu fonk.'nun this göstericisi const Myclass*
        x = 5; // static üye fonksiyonu değiştirmek sorun olur mu ?
        cevap : geçerli, const üye fonk. sınıfın static veri elemanını değiştirebilirler.
    }
private:
    static int x; //
};

// constructor init. list : sınıfın non-static elemanlarını init. eder.
```

```
class Myclass{
public:
    void func()
    {
        x = 10;
        Myclass::x = 10;
        this->x = 10;
        // hiçbirinde hata yok
    }

private:
    static int x;
};
```

```
class Myclass{
public:
    void func()
    {
        double x ;
        x = 10; // static veri ismi yerel değişken tarafından gizlenir. Name-masking (name-hiding)
    }
private:
    static int x;
};
```

```
    class Data{
        Data& r;
        Data *ptr;
        // kendi türünden elemanı olamaz ama pointer, referans olabilir ????
    };
```

```
   class Data{
       Data dx; // sentaks hatası ????
       static Data dx;  // ok
   };
```

sınıfların static üye fonksiyonları:

Tıpkı global fonksiyonlar gibi this pointer'ı olmayan çağırılmaları için sınıf instance'ı gerekmeyen fonksiyonlardır.
instance ile değil class ile iş yaparlar.

```
class Data{

public:
    void func();
    static void func()  // this pointer'ı yok
    {
        mx = 5; // name-lookup hatası yok !!  fakat ortada bir sınıf nesnesi yok çünkü this yok.
sentax hatası
    }
private:
    int mx, my;
};
```

```
// data.h
class Data{

public:
    static void foo();
private:
    int mx, my;
};

// data.cpp
void Data::foo()   // static anahtarı kullanılmamalı
{

}
```

```
//myclass.h
class Myclass{
public:
    void foo()
    {
        func(); // ok
    }
private:
    static void func()
    {
        // foo(); // isim arama açısından problem yok, this pointer'ı olmadığından sentax hatası
    }
};
```

```
//myclass.h
class Myclass{
public:
    void foo()
    {

    }
    static void func()
    {
        x = 5; // sınıfın static üye fonk. sınıfın static üyelerini kullanabilir
    }
    static int x;
};
```

```
static void func()const  // yanlış !!!
```

const gizli pointer parametresini
niteler. static fonk.'nun böyle bir parametresi olmadığından yanlıştır.

asla asla move constructor'ı delete etmeyin !!!!

```
class Myclass{

public:
    Myclass() = default;
    Myclass(const Myclass&);
     Myclass(Myclass&&)=delete;
    // move  const. delete edilirse
    move çağırmak sentax hatası !!!
};
```

```
class Myclass{

public:
    Myclass() = default;
    Myclass(const Myclass&);
    // move const. is not declared
};

int main()
{
    Myclass mx;

    Myclass my = std::move(mx);
   // taşıma işlemi yapılmaz copy yapılır,
    ama sentax hatası değil
}
```

taşıma gereken her yerde taşıma değil copy yapılacak !!! çünkü move const. yok.
Taşımadan bir avantaj elde edilmeyecekse taşımaya gerek yok. Gayet doğal bir durum.

öyle sınıflar var ki copy'e kapatmak ama move semantik ile kullanmak istiyorsun. Thread sınıfı böyle.