2023.09.22

Virtual DTOR

```
class Base
{
    public:
        virtual ~Base()
        {
            std::cout << "Base Desturctor\n";
        }
};

class Der : public Base
{
    public:
        ~Der()
        {
            std::cout << "Der destructor\n";
        }
};
int main()
{
    Base *p = new Der;
    delete p;

    /*
        Eğer Base sınfın dtor virtual olarak tanımlanmazsa p nesnesi delete
        edildiğin de Base sınıfın dtor çağırılır bu da tanımsız davranıştır.

        Ancak virtual olarak tanımlanırsa önce Der sınıfın dtor sonra
        Base sınıfın dtor'u çağrılır.
    */
}
```

Polimorfik classların dtorları ya public virtual ya da protected non-virtual olmalı

```
class Base
{
    protected:
        ~Base()
        {
            std::cout << "Base Desturctor\n";
        }
};
class Der : public Base
{
    public:
        ~Der()
        {
            std::cout << "Der destructor\n";
        }
};
int main()
{
    Base *p = new Der;
    delete p;  // hatalı olur. Çünkü base dtor protected ve Der classı bunu
çağramaz
}
```

Virtual Dispatch'e alternatifler:

1. type erasure
2. std::variant
3. CRTP (curiously recurring template pattern)

```cpp
class Base
{
    int x{}; // 4 byte
    int y{}; // 4 byte
    void f1();  // 0 byte

    // 4 byte
    virtual ~Base();
    virtual void f2();
    virtual void f3();
};

class Der : public Base
{

};

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

```
class Base
{
    int mx;
    virtual void foo();
}

base address
------------
mx
vptr (virtaul function table pointer)


Virtual Function Table Pointer:

Bir veri yapısıdır (data structure) ve her nesne için değil her class
için vardır.


virtual function table for class Audi
-------------------------------------
0
-------------------------------------
1       &Audi::start
2       &Audi::run
3       &Audi::stop

Example:

void car_game(Car *p)
{
    p->run(); ---> p->vptr[2]();

}

dereferencing : adresteki nesneye erişmek


Her sanal fonksiyon çağırısı için virtual dispatch uygulanmayabilir.
Compiler optimazition yapılabilir.

Devirtualization
```

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

class Der : public Base
{
    public:
        // Overloading olur böylece
        using Base::func;
        void func(double);
}

int main()
{
    Der myder;
    myder.func(12); // func(int) çağrılır
}
```

Inheritanced Ctor

```
class Base
{
    public:
        Base(int);
        Base(int, int);
        Base(const char*);
};

class Der : public Base
{
    public:
        using Base::Base; // 1 option --> inheritanced ctor
        Der(int x, int y) : Base (x, y) {} //2 option
        void bar();
}

int main()
{
    Der myder(12, 56)
}
```

Covariance or Variant Return Type

```
class B{};
class  D : public B{};

class Base
{
    public:
        virtaul B* foo();
        virtual B& bar();
        virtual B baz();
}

class Der : public Base
{
    public:
        D *foo()override;
        D &bar()override;
        D baz()override; // hatalı pointer veya referans olmak zorunda
        //foo ve bar fonkisyonları covariant oldu
}

int main()
{
    Der myder;
    D* dp = myder.foo();
    auto dp2 = myder.foo();
}
```

NVI (non-virtual interface)

```
class Base
{
    public:
        void foo(int x)
        {
            vfoo(x);
        }
    private:
        virtual void vfoo(int);
};

class Der : public Base
{
    public:
        void vfoo(int) override;
}
```

Final Contextual Keyword

- final class
- final override

Final Class

```
// Der sınıfından kalıtım yapmak syntax hatası olur final kullanırsak
class Base
{

};

class Der final : public Base
{

};

class NDer : public Der
{
    // syntax hatası -- final keyword kaynaklı
};
```

Final Override

```
class Base
{
    public:
        virtual void func();
};

class Der : public Base
{
        void func() override final;
};

class NDer : public Der
{
        void func()override; // syntax hatası
};
```