• Tekrar: Run time'da belli olan program late binding
dynamic binding
Taban sınıfın pointer / referansı ile virtual dispatch olmaya yarar.

• Modern C++'da contextual keyword olan override eklendi

```
class Bmw : public Car {
public:
    void start()override
    {
        std::cout << "Bmw has just started!\n";
    }
    void run()
    {
        std::cout << "Bmw is running now!\n";
    }
    void stop()
    {
        std::cout << "Bmw has stopped!\n";
    }
```

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

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

• Logic hata: virtual olmayan bir fonksiyon, override edilmeye çalışılmış. Taban sınıfın fonksiyonu çağrılır.

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

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

• Logic hata: İmzası farklı bir override yapılmış. Eğer override keyword'ü olmasaydı syntax hatası olmazdı.

→ override keyword eklendiği için: Artık syntax hatası override compiler'a kontrol ettiriyor. Bu override legal mi diye.

• RTTI (Run Time Type Information / Identification):

- Run time'da olusacak sınıfın hangi derived class olduğunu anlamaya yarayan araç

* Tür kavramına sahip dillerde static typing olmak üzere iki adet tür vardır. Static typing → verinin türünü çevirir sırasında
dynamic typing → // // run timeda

```
void car_game(Car& cr)
{
    cr.start();
    cr.run();
    cr.stop();
    std::cout << "----------
```

Compiler açısından cr, Car türünden

→ Ama run time'da nasıl bir araç olabilir

→ Dynamic type olması için polymorphic class olmalıdır.

- Dynamic: davranışsal
- Static: derleyicinin, translate edip, kontrolünü yaptığı

- Virtual Dispatch'in çalışmadığı durumlar:

* Kısıtlama:

```
class Base {
private:
    virtual void vfunc();
};

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

→ Private, override edilebilir.

• Aynı şekilde derived class, private'da da override eder.

→ Override'ın Access Control'e takılma durumu yoktur. Access Control compile time için

→ Varsayılan Argüman Compile Time'a ilişkin bir mekanizmadır. Bu yüzden

```
class Base {
public:
    virtual void vfunc(int x = 10)
    {
        std::cout << "Base::vfunc(int x) x = " << x << "\n";
    }
};

class Der : public Base {
public:
    void vfunc(int x = 77)override
    {
        std::cout << "Der::vfunc(int x) x = " << x << "\n"
    }
};
```

→ Der vfunc, 10 değeriyle çağırılır.

→ Compiler, default argümanı 10 olarak alır.

Biz override edilmiş hali istiyorsak:

```
int main()
{
    Der myder;

    myder.vfunc();
}
```

→ myder.vfunc() olarak çağırılmalı

bu statiktir.

* Name lookup → Compile time / static type
* Access control → Compile time / static type
* Default Argument → Compile time / static type

```cpp
class Base {
public:
    Base()
    {
        vfunc();
    }
private:

    virtual void vfunc()
    {
        std::cout << "Base::vfunc()\n";
    }
};


class Der : public Base {
public:
    void vfunc()override
    {
        std::cout << "Der::vfunc()\n";
    }
};


int main()
{
    Der myder;
}
```

→ Override edilemeyen fonksiyonlar, override edilebilen fonksiyonları çağırabilir.

→ Ancak constructor'da durum farklı

• Fakat: Virtual dispatch devreye girmez.

→ Çünkü hayata gelme sırasında önce base class constructor çağrılır. Henüz derived class hayata gelmedi.

→ Destructor'da da virtual dispatch devreye girmez.

- Neden başka func içinde virtual çağırılır

```cpp
class Base {
public:
    void foo(Base *p)
    {
        vfunc();
    }
```

- Çünkü argümanla dışarıdan fonksiyonun parametresi olmasa bile aslında Base* p var.

→ Özetle: * base object (pointer veya ref değil) direkt kendi adıyla fonksiyon çağırırsa

* base class'ın ismiyle nitelendirilmiş fonksiyonların çağrısında

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

önce base

→ sanal virtual

+ base class constructor da,
+ base class destructor da

= Virtual dispatch devreye girmez!!

• Sınıf nesnesinin içinde sanal olmayan fonksiyonlar yer kaplamaz!!

Sanal fonksiyonlar ise = Burada durum farklı. Sanal fonksiyonlar → classı polymorphic class yapar. Polymorphic classlar da (tüm veri elemanları + 1 adet pointer) sizeındadır.

- Bu pointer vptr / sanal fonksiyon tablo gösterici

• Virtual Function Table :

```
0
1   &Toyota::start
2   &Toyota::run
3   &Toyota::stop

virtual function table for class Dacia
0
1   &Dacia::start
2   &Dacia::run
3   &Dacia::stop
```

- Aynı hiyerarşideki sanal fonksiyonlar için, compile time da oluşan tablo

- Bu indexler, her dar class için aynı.

```
Car *carptr;
carptr->run();
carptr->vptr[1]();
```

derleyicinin ürettiği kod analiz eder.

- Override edilmemişse, tabloya base class'ınki eklenir.

- maaliyeti: vptr'ye erişmek ve vptr[2] gibi dereferencing yapması

```
=======================================

her polimorfik nesne için vptr'nin initialize edilmesi
her sanal işlev çağrısı için (derleyici optimizasyon yapmıyor ise)
    2 x dereferencing


her polimorfik sınıf için sanal fonksiyon tablosu veri yapısı oluşturuluyor

her polimorfik nesne için  bir pointer daha fazla olacak. ve, bellekte yer kaplayan fonksiyon tablosu
```

- Pure virtual Function Tanımı:

```
class Car {
public:
    virtual void start() = 0;

    virtual void run() = 0;

    virtual void stop() = 0;
};
```

- Car mx; gibi instantiate edilemez X

- Derived class concrete class olmak için hepsini implement etmeli. Yoksa, o da abstract olarak kalır.

* Constructorlar virtual olamaz X. Syntax hatası. Onun yerine virtual constructor idiom / clone constructor idiom var

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

→ Clone constructor idiom:

```
class Toyota : public Car {
public:
    Car* clone()override
    {
        return new Toyota(*this);
    }
```

```
void car_game(Car* p)
{
    Car* carptr = p->clone();
    carptr->start();
    p->start();
    p->run();
    p->stop();
}
```

→ Clone ile kendinden bir adet daha yarattı

* Destructor virtual olabilir ✓. Çoğu zaman olmak zorunda. Yoksa, taban sınıf nesnesinin pointer'ı ile, türemiş sınıf nesnesi delete edildiğinde → undefined behavior.

→ ya public virtual
ya da non-virtual protected tanımlanmış polimorfik sınıflar → nesnenin yaşamını bitir