**2023.09.20**

**dynamic binding or late binding**

Bir base sınıfın fonksiyonları:

1. Türemiş sınıflara hem bir interface (arayüz)

hemde bir implementasyon veren fonksiyonlar

1. Türemiş sınıflara hem bir interface (arayüz)

hemde bir default implementasyon veren fonksiyonlar

1. Türemiş sınıflar yalnızca bir interface veren implementasyon

vermeyen fonksiyonlar

* Bir sınıfın en az 2. maddeki gibi bir fonkisyonu varsa böyle sınıflara

polymorphic (çok biçimli) sınıf denir.

* Bir sınıfın en az 3. maddeki gibi bir fonkisyonu varsa böyle sınıflara

abstract (çok biçimli) sınıf denir.

* Eğer bir sınıf abstract sınıf ise o sınıftan bir nesne oluşturamayız

**class** Airplane

**{**

**public:**

**void** takeoff**();** ***// 1. maddeye örnek***

**virtual** **void** fly**();** ***// 2. maddeye örnek***

**virtual** **void** land**()** **=** 0**;** ***// 3. maddeye örnek (pure virtual function)***

**};**

**class** Airbus **:** **public** Airplane

**{**

**};**

**void** flygame**(**Airplane **);** ***// abstract class'ta geçerli değil***

**void** flygame**(**Airplane**\*** **);** ***// geçerli***

**void** flygame**(**Airplane**&** **);** ***// geçerli***

override ve final keywordleri contextual keyword (bağlamsal anahtar sözcük)

Eğer bir fonksiyon çağrısı taban sınıf türünden bir nesne ile değil taban sınıf türünden bir pointer değişken ya da taban sınıf türünden bir referans değişken ile yapılıyor ise "virtual dispatch" (sanal gönderim)

**class** Base

**{**

**public:**

**virtual** **int** func**(int);**

**};**

**class** Der **:** **public** Base

**{**

**public:**

**double** func**(int);**

***// overriding değil overloading değil redefination var***

**int** func**(int)** **override;** ***//overriding***

**}**

**EXAMPLE ( RUN TİME POLYMORPHSİM)**

*/// Example*

***// Run Time Polymorphism***

**class** Car

**{**

**public:**

**virtual** **void** start**()**

**{**

std**::**cout **<<** "Car::start()\n"**;**

**}**

**virtual** **void** stop**()**

**{**

std**::**cout **<<** "Car::stop()\n"**;**

**}**

**};**

**class** Audi **:** **public** Car

**{**

**public:**

**void** start**()**

**{**

std**::**cout **<<** "Audi::start()\n"**;**

**}**

**void** stop**()**

**{**

std**::**cout **<<** "Audi::stop()\n"**;**

**}**

**};**

**class** Tesla **:** **public** Car

**{**

**public:**

**void** start**()**

**{**

std**::**cout **<<** "Tesla::start()\n"**;**

**}**

**void** stop**()**

**{**

std**::**cout **<<** "Tesla::stop()\n"**;**

**}**

**};**

Car**\*** create\_random\_car**()**

**{**

**static** std**::**mt19937 eng**{**std**::** random\_device**{}()** **};**

**static** std**::**uniform\_int\_distribution dist**{** 0**,** 1**};**

**switch** **(**dist**(**eng**))**

**{**

**case** 0**:** std**::**cout **<<** "Tesla\n"**;** **return** **new** Tesla**;**

**case** 1**:** std**::**cout **<<** "Audi\n"**;** **return** **new** Tesla**;**

**}**

**return** **nullptr;**

**}**

**int** main**()**

**{**

***// programın hangi fonksiyonu çağıracağı run-time'da belli olur.***

***// buna run time polymorphism denir***

**for** **(int** i **=** 0**;** i**<** 1000**;** **++**i**)**

**{**

Car**\*** p **=** create\_random\_car**();**

p**->**start**();**

p**->**stop**();**

**delete** p**;**

**}**

**}**

**class** Base

**{**

**public:**

**virtual** **void** foo**();**

***// NVI (Non virtual Interface)***

**void** bar**()**

**{**

foo**();**

**}**

**};**

**class** Der **:** **public** Base

**{**

**public:**

**void** foo**();**

**};**

**void** gf1**(**Base**\*** p**)**

**{**

p**->**foo**();**

**}**

**void** gf2**(**Base**&** p**)**

**{**

p**.**foo**();**

**}**

**int** main**()**

**{**

Der myder**;**

myder**.**bar**();** ***// Der sınıfın foo() fonksiyonu çağrılır.***

**}**

**Override Keyword**

* Derleyici fonksiyonun virtual olup olmadığını kontrol etmek zorunda oluyor böylece virtual fonksiyonda değişiklik olursa (aldığı parametre değişirse gibi) derleyici hata döner.
* Okuyucuya fonksiyonun override edildiğini belirtiyor

**class** Base

**{**

**public:**

**virtual** **void** foo**(long);**

**virtual** **void** bar**(double);**

**};**

**class** Der **:** **public** Base

**{**

**public:**

**void** foo**(long)** **override;**

**void** bar**(int)** **override;** ***// hatalı***

**};**

Bir taban sınıfın sanal fonksiyonunu override eden türemiş sınıfın fonksiyonu virtual specifier ile nitelenmemiş olsa da yine sanal fonksiyondur.

**Virtual Functions -Multi Level Inheritance**

**class** Base

**{**

**public:**

**virtual** **void** foo**(long);**

**};**

**class** Der **:** **public** Base

**{**

**public:**

***// virtual ile nitelendirmesek bile virtual'dır***

**virtual** **void** foo**(long)** **override;**

**};**

**class** NDer **:** **public** Der

**{**

**public:**

***// NDer class'ı foo fonksiyonunu override edebilir.***

**void** foo**(long)** **override;**

**}**

***/\****

***Eğer NDer override etmeseydi foo(long) fonksiyonu Der sınıfın foo(long)***

***fonksiyonu çağrılırdı. Eğer Der override etmeseydi Base sınıfın foo(long)***

***fonksiyonu çağrılırdı.***

***\*/***

**class** Base

**{**

**private:**

**virtual** **void** foo**()**

**{**

std**::**cout **<<** "Base::foo()\n"**;**

**}**

**};**

**class** Der **:** **public** Base

**{**

**public:**

**virtual** **void** foo**()**

**{**

std**::**cout **<<** "Der::foo()\n"**;**

**}**

**};**

***// Base class'taki foo fonksiyonu private olmasına rağmen bu kod legal***

**int** main**()**

**{**

Der myder**;**

myder**.**foo**();** ***// "Der::foo()" yazısını ekrana yazar***

**}**

**ÖNEMLİ !**

**class** Base

**{**

**public:**

**virtual** **void** foo**()**

**{**

std**::**cout **<<** "Base::foo()\n"**;**

**}**

**private:**

**virtual** **void** bar**();**

**};**

**class** Der **:** **public** Base

**{**

**private:**

**virtual** **void** foo**()**

**{**

std**::**cout **<<** "Der::foo()\n"**;**

**}**

**public:**

**void** baz**();**

**void** bar**()override;**

**};**

**void** gf**(**Base **\***p**)**

**{**

p**->**foo**();** ***// erişim kontrolu devreye girmez***

***/\****

***Kontrol tamamen base göre yapılıyor. Eğer base class'ın foo()***

***fonksiyonu private olsaydı erişim kontrolu devreye girerdi.***

***Erişim kontrolu run-time ilişkin (virtual dispatch)***

***\*/***

***/\****

***Derleyici koda baktığında compiler time'da p'nın türü Base ancak***

***davranışı belirleyen tür Der sınıfı (run-time). İsim arama static***

***türe bağlı olarak yapıldığı için p nesnesi Base sınıfın public foo()***

***fonksiyonu olarak ilişkilendirilecek.***

***\*/***

p**->**baz**();**

***/\****

***hatalı olur çünkü isim arama static türe göre olur yani Base sınıfında***

***baz fonkisyonu olmadığı için isim arama hatası olur.***

***\*/***

p**->**bar**();** ***// erişim kontrolu hatası olur***

**}**

**int** main**()**

**{**

Der myder**;**

gf**(&**myder**);**

myder**.**foo**();** ***// erişim kontrolu olur***

***// erişim kontrolu compiler-time ilişkin***

**}**

***/\****

***C++ dilini yeni öğrenenler tipik olarak statik tür kavramı (static type)***

***dinamik tür kavramını (dynamic type) karıştırırlar.***

***Statik tür derleyicinin koda bakarak gördüğü tür demek iken***

***Dinamik tür ise run-time'daki davranışı belirleyen tür demek***

***\*/***

**Varsayılan Arguman – Virtual Functions**

**class** Base

**{**

**public:**

**virtual** **void** foo**(int** x **=** 9**)**

**{**

std**::**cout **<<** "Base::foo(int x) x = " **<<** x **<<** "\n"**;**

**}**

**};**

**class** Der **:** **public** Base

**{**

**public:**

**virtual** **void** foo**(int** x **=** 99**)**

**{**

std**::**cout **<<** "Der::foo(int x) x = " **<<** x **<<** "\n"**;**

**}**

**};**

**void** gf**(**base **&**r**)**

**{**

r**.**foo**();**

***/\****

***Varsayılan arguman static türle ilgili olduğu için Base sınıfının***

***varsayılan argumanı kullanılacak yani x = 9***

***Ama r nesnesi runtime da belirlendiği için Der'in foo() fonskiyonu***

***çağrılır***

***Çıktı:***

***Der::foo(int x) x = 9***

***\*/***

**}**

**int** main**()**

**{**

Der myder**;**

gf**(**myder**);**

myder**.**foo**();** ***// Der::foo(int x) x = 99***

**}**

**Overloading Functions – Inheritance**

**class** Base

**{**

**public:**

***//overloading fonksiyonlardır***

**virtual** **void** foo**(int);**

**virtual** **void** foo**(double);**

***// Sadece non-static member functions virtual olabilir***

**virtual** **static** **void** bar**();** ***// syntax hatası***

***// Ctor virtual olarak bildirilemez.***

**virtual** Base**(int);** ***// syntax hatası***

**}**

**class** Der **:** **public** Base

**{**

**public:**

***// overloading fonksiyonlar***

**void** foo**(int)** **override;**

**void** foo**(double)** **override;**

**}**

**Virtual Ctor C++ (Clone idiom)**

C++ dilinde sınıfların ctor'ları virtual anahtar sözcüğü ile bildirilemez. Ancak virtual ctor özellikle OOP paradigmasında bir ihtiyaç olabilir. Virtual ctor yerine C++ dilinde: virtual ctor idiom ya da clone idiom

**class** Car

**{**

**public:**

**virtual** Car**\*** clone**()** **=** 0**;**

**};**

**class** Volvo **:** **public** Car

**{**

Volvo**();**

Car**\*** clone**()override**

**{**

**new** Volvo**(\*this);**

**}**

**};**

**void** car\_game**(**Car **\***p**)**

**{**

***/\****

***Buraya (run time'da) hangi türden araba gelirse oyunun senaryosu***

***gereği run-time'da aynı türden bir araba oluşturalacak***

***\*/***

Car **\***p **=** p**->**clone**();**

**}**

**int** main**()**

**{**

Volvo v**;**

car\_game**(&**v**);**

**}**

**Dikkat!**

* Base sınıfın ctor'i içinde yapılan virtual sınıf fonksiyon çağrıları için; virtual dispatch uygulanmaz. Çünkü henüz hayata gelmemiş Der nesnesi için sanki o hayattaymış gibi üye fonksiyonu çağrırdık.
* Dtor içinde virtual dispatch uygulanmaz. Çünkü en son base sıfının dtor çağrılır. Der sınıf çoktan yokolmuş olacak. Bu yüzden virtual dispatch uygulanmaz

**class** Base

**{**

**public:**

Base**()**

**{**

vfunc**();**

**}**

**~**Base**()**

**{**

vfunc**();**

**}**

**virtual** **void** vfunc**()**

**{**

std**::**cout **<<** "Base::vfunc()\n"**;**

**}**

**void** foo**()**

**{**

Base**::**foo**();**

***//Nitelenmiş isimlerde virtual dispatch devreye girmez.***

**}**

**};**

**class** Der **:** **public** Base

**{**

**public:**

**virtual** **void** vfunc**()override**

**{**

std**::**cout **<<** "Der::vfunc()\n"**;**

**}**

**};**

**int** main**()**

**{**

***// Base::vfunc()\n yazar***

Der myder**;**

**}**

Virtual dispatch devreye girmediği yerler:

1. Virtual fonksiyon çağrısı Base sınıfın nesnesi ile yapılıyorsa
2. Virtual fonksiyon çağrısı Base sınfın ctor içinde yapılıyorsa
3. Virtual fonksiyon çağrısı Base sınıfın dtor içinde yapılıyorsa
4. Virtual fonksiyon çağrısı nitelendirmiş isim olarak yapılırsa