**2023.09.27**

**RTTI ( Run Time Type Information)**

* dynamic\_cast
* typeid
  + type\_info

upcasting : türemiş sınıftan taban sınıfa yapılmasına denir.

downcasting : taban sınıftan türemiş sınıfa yapılmasına denir.

**Dynamic\_cast**

dynamic\_cast<target tope>(variable)

Mercedes \*p = dynamic\_cast<Mercedes\*>(car\_ptr)

Mercedes &p = dynamic\_cast<Mercedes&>(car\_ref)

Eğer cast başarılı olmazsa p nullptr olur

**class** Base

**{**

**public:**

**virtual** **~**Base**()** **=** **default;**

**};**

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

**{**

**};**

**void** foo**(**Base**\*** baseptr**)**

**{**

***/\****

***Dynamic cast olabilmesi için base sınıfı polymorfik olması gerekir***

***yani en az bir tane virtual fonksiyonu olması gerekiyor.***

***\*/***

Der**\*** derptr **=** **dynamic\_cast<**Der**\*>(**baseptr**);**

**};**

Dynamic Cast Usage:

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

**{**

Tesla **\***tp **=** **dynamic\_cast<**Tesla**\*>(**carptr**)**

**if** **(**tp**)**

**{**

tp**->**autopilot**();**

**}**

***// böyle yapmak daha havalı***

**if** **(**Tesla **\***tp **=** **dynamic\_cast<**Tesla**\*>(**carptr**))**

**{**

tp**->**autopilot**();**

**}**

**}**

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

**{**

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

**{**

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

car\_game**(**p**);**

**delete** p**;**

**}**

**}**

***// Reference semantiği ile (Not Null Ref yok)***

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

**{**

**try**

**{**

Tesla **&**tp **=** **dynamic\_cast<**Tesla**&>(\***carptr**);** ***// down\_casting***

tr**.**autopilot**();**

**}**

**catch** **(const** std**::**exception**&**ex**)**

**{**

std**::**cout **<<** "exception caught: " **<<** ex**.**what**()** **<<** "\n"**;**

**}**

***/\****

***Dynamic\_cast kullanımında hedef tür referans türü ise***

***down-casting güvenli bir şekilde yapılamıyor ise standart***

***kütüphanemizin std::bad\_cast sınıfı türünden exception throw edilir.***

***\*/***

**}**

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

**{**

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

**{**

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

car\_game**(**p**);**

**delete** p**;**

**}**

**}**

**Typeid**

***/\****

***typeid bir operatordür***

***typeid(\*ptr);***

***typeid(x);***

***typeid(10);***

***type(Tesla);***

***type(int);***

***type info sınıf türünden const bir nesneye referans***

***her ayrı (distinct) tür için bir type\_info nesnesi var***

***\*/***

#include <typeinfo>

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

**{**

**int** x **=** 10**;**

**typeid(**x**);** ***// bu ifadenin türü typeinfo***

**auto** y **=** **typeid(**x**);** ***// syntax hatası çünkü typeid copy ctor delete edilmiş.***

***//legal***

**const** **auto** **&**r **=** **typeid(**x**);**

**auto** **&**r2 **=** **typeid(**x**);**

**}**

**Unevaluated Context**

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

**{**

**int** x **=** 10**;**

**auto** sz **=** **sizeof(**x**++);**

**decltype(**x**++)** y **=** x**;** ***// unevaluated context***

std**::**cout **<<** "x = " **<<** x **<<** "\n"**;**

***// x = 10 çünkü sizeof unevaluated context***

**int** a**[**10**]{};**

**auto** s **=** a**[**30**]** ***// tanımsız davranış***

**auto** sz **=** **sizeof(**a**[**30**]);** ***// tanımsız davranış değil***

**int** **\***ptr**{};**

**auto** x **=** **\***ptr**;** ***// tanımsız davranış***

**auto** szz **=** **sizeof(\***ptr**);** ***// tanımsız davranış değil***

**int** x **=** 12**;**

**const** **auto&**r **=** **typeid(++**x**);** ***// unevaluated context***

std**::**cout **<<** "x = " **<<** x **<<** "\n"**;** ***// x = 12***

**int** **\***ptr**{};**

**const** **auto&**p **=** **typeid(\***ptr**);** ***// tanımsız davranış değil***

**}**

**class** Nec

**{**

**};**

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

**{**

Nec mynec**;**

***// derleyiciden derleyiciye çıktı değişir o yüzden karşılaştırma yapmamak lazım***

std**::**cout **<<** **typeid(**Nec**).**name**()** **<<** "\n"**;**

**typeid(**mynec**).operator==(typeid(int))** ***// false***

**typeid(**mynec**).operator==(typeid(**Nec**))** ***// true***

**}**

**class** Base

**{**

**virtual** **~**Base**(){};**

**};**

**class** Der **:** **public** Base **{};**

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

**{**

Der myder**;**

Base **\***ptr **=** **&**myder**;**

***/\****

***Eğer Base polymorfik bir class ise 1 döner ama değilse 0 döner***

***Burada Base polymorfik o yüzden 1 döner***

***\*/***

std**::**cout **<<** **(typeid(\***ptr**)** **==** **typeid(**Der**))** **<<** "\n"**;**

**}**

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

**{**

**if** **(typeid(\***ptr**)** **==** **typeid(**Audi**))**

**{**

std**::**cout **<<** "This a Audi\n"**;**

**}**

**if** **(typeid(\***ptr**)** **==** **typeid(**Tesla**))** ***// run-time'da yapılır***

**{**

**static\_cast<**Tesla **\*>(**ptr**)->**autopilot**();**

***// dynamic\_cast'deki gibi Teslanın alt sınıfları bu if'e girmez***

**}**

**}**

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

**{**

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

**{**

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

car\_game**(**p**);**

**delete** p**;**

**}**

**}**

**static\_cast in class**

**class** Base**{};**

**class** Der **:** **public** Base **{};**

**class** Nec **{};**

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

**{**

Nec mynec**;**

***//legal değil***

Der **\***derptr **=** **static\_cast<**Der**\*>(&**mynec**);**

***// legal***

Base mybase**;**

Der **\***derptr **=** **static\_cast<**Der**\*>(&**mybase**);**

**}**

Note: Virtual Function Table'da typeid için yer ayrılıyor

1. **RTTI hiç kullanılmazsa ilave bir maliyet var mı?**

Var çünkü run-time'da her türlü typeid nesneleri oluşturuluyor ama

derleyiciye bağlı olarak biz run-time type nesnesini kullanmacağım dersek

RTTI disable edersek bize maliyet oluşturmadan kapatırız

1. **Büyük bir hiyerarşi var ve typeid ya da dynamic\_cast kullancaksın ikiside**

**işimizi görüyor. Hangisinin maliyeti düşüktür?**

Typeid'nin maliyeti daha düşük çünkü typeid classların türemiş sınıflarını kontrol

etmez ancak dynamic\_cast türemiş sınıfları kontrol eder.

typid(\*ptr)

polimorfik tür söz konusunda olduğunda eğer typeid operatörünün operandı olan ifade dereference edilmiş pointer ifadesi ise pointer değerinin nullptr olması durumunda:

std::bad\_typeid sınıfı türünden exception throw edilir. bad\_typeid sınıfı std::exception sınıfından kalıtım yoluyle elde edilir.

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

**{**

**try**

**{**

**if** **(typeid(\***ptr**)** **==** **typeid(**Tesla**))**

**{**

**static\_cast<**Tesla **\*>(**ptr**)->**autopilot**();**

**}**

**}**

**catch** **(const** std**::**bad\_typeid**&** ex**)**

**{**

std**::**cout **<<** "exception caught: " **<<** ex**.**what**()** **<<** "\n"**;**

***// nullptr***

**}**

**}**

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

**{**

Car**\*** ptr**{};**

car\_game**(**ptr**);**

***// ptr null olduğu için expection verir***

**}**

Taban sınıfların dtor'ları ya public virtual ya da protected non virtual olmalı. Ancak NVI kullanarakta tanımlayabiliriz.

**Template Method**

***// template method***

**class** Figther

**{**

**virtual** **void** attack**();**

**public:**

**void** fight**()**

**{**

attack**();** ***// savaşçıyı göre customize edilecek***

**}**

**};**

Non Virtual Interface ile:

* Taban sınıf kendi kontrollerini dayatabiliyor
* Ortak kodları bi yere toplayabiliyoruz
* implemantasyon ile interfacesi ayırmış oluyoruz

**class** Figther

**{**

**public:**

**void** attack**()**

**{**

***//invariant'lar***

attack\_impl**();**

***//invariant'lar***

**};**

**private:**

**virtual** **void** attack\_impl**()** **=** 0**;**

**}**