Observer Design Pattern

Calculator’ım var diyelim. Birisi calculate fonksiyonunun sonucunu observe etse güzel olur. Böylece sonuç çıktığında display edilecekse display eder, ağdan yollanacaksa yollar, gerekli işlemleri yaparım.

void Calculator::calculate(){

…

display.set(result);

network.set(result);

}

ICalculateObserver{

public:

virtual void set(int res) = 0;

}

class Calculator{

public:

void addObserver(ICalculateObserver \*pObs){

m\_observer.add(pObs);

}

void calculate(){

…

for (auto pObs: m\_observer)

pObs.set(result);

}

}

CRTP

Sınıf türediği base class’a kendini template olarak verir.

class NetReader: public Exec<NetReader>



Aynı olması CRTP

***C++20 Concepts***

template <class Info>

class MyClass{

…

info.setData(5);

Info info;

}

setData yoksa (ismi hatalıysa, typo varsa vs.) 20 satır compile hatası alırsın.

template<INFOClass Info>

concept InfoClass{

setData(int) var olmalı!!!

}



Concept içerisine bunu dersen şimdi 1 satır compile hatası alırsın.

A screenshot of a computer

Description automatically generated

<https://www.youtube.com/watch?v=ZQ-8laAr9Dg>

<https://www.youtube.com/watch?v=Srx4eiBdpdQ>

In base class, you already know what type the class will have that will be deriving from you. It’s like a genie looking into the future.

We want to reduce run-time polymorphism cost.