override keyword 🡪 Kodun anlaşılmasını sağlar. override demezsen farklı parametre kullanır veya harf vs. yanlış yazarsan yeni fonksiyon yaratılır.

final keyword 🡪 Yine override ediyorum ama aşağıda sınıfların override etmesini de engelliyorum.

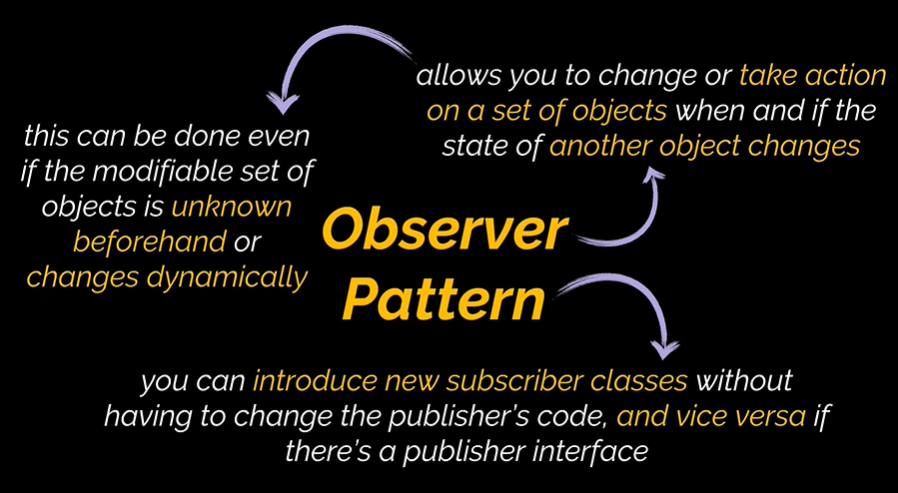
Foo x{4}; 🡪 constructor  
Foo y = x; 🡪 copy constructor  
Foo z = 5; 🡪 constructor başına “explicit” demezsen cast etmeye çalışır

C++ RTTI (typeID()) 🡪 mecbur kalmadıkça asla kullanma. Hem overhead var hem de tasarım kötü olur.

Observer Pattern

Behavioral design pattern.

Notifies multiple objects, or subscribers, about any events that happen to the object they are observing or publisher.



**UML – SYSML**

*Class Diagram*

Composition

A black arrow pointing to a white background

Description automatically generated

Foo Bar’ın parçasıdır.

Foo Bar varken vardır yokken yoktur.

Foo Bar’ın içinde pointer da olabilir. Constructorda yaratıp desctrutorda öldürürsün.

Aggregation

A black arrow pointing to the right

Description automatically generated

Foo Bar’ın parçasıdır.

Foo’nun lifetime’ı Bar’dan bağımsızdır.

Foo Bar’ın içinde pointer olabilir.

Dependency

A line of lines on a white background

Description automatically generated

void presentFoo(Foo foo);

Foo ile ilgili fonksiyon var, Foo member değil.

Lifetime söz konusu değil.

Association

A long thin line on a white background

Description automatically generated

İlişkinin ne olacağından emin değilsin. Genel bir ilişki.

Ok yönü kimin kimi include ettiğiyle alakalıdır. Foo değişirse Bar recompile olacak.

Oklar karşılıklı olabilir ama tercih etmeyiz.

Is\_A

A long thin line of metal

Description automatically generated with medium confidence

Bar Foo’dan türemiş.

Bar Foo’ya bağımlı.

A black arrow pointing to a white background

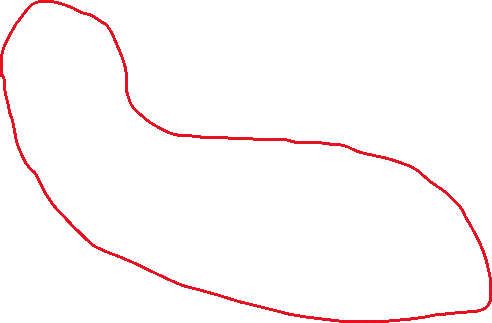
Description automatically generated

Bar’ın içerisinde birbirinden bağımsız 3 Foo member’ı var.

Hiç olmayabilir, sonsuz olabilir (Bar’ın içerisinde Foo vektörü vs. olabilir).

A diagram of a process

Description automatically generated



Ok tarafı değişirse diğer taraf compile etmek zorunda.

Kırmızı bölge 🡪 Yeni velocity message eklendiğinde oluşan bağımlılıklar.

*Sequence Diagram*

Zaman aşağıya doğru akar.

Creation Sequence

A diagram of a message factory

Description automatically generated

Transmit Sequence

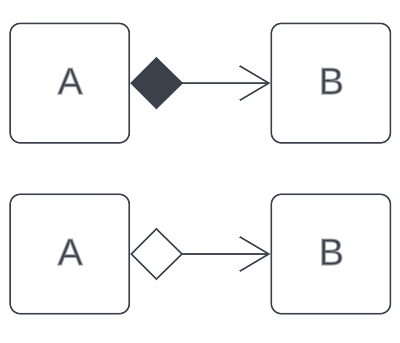
A diagram of a computer program

Description automatically generated

There is also activity diagram and state diagram.

SUMMARY

Relationships among classes 🡪 UML

* Composition
* Aggregation
  + These two are ownership
  + 
  + These two means A has B
  + Üsttekine örnek:

struct A{

B b;

};

* + Alttakine örnek:

struct A{

B \*pB;

};

* + A a; dediğinde üsttekinde B otomatik instantiate oluyor, diğerinde olmuyor.
* Dependency
  + A diagram of a diagram

    Description automatically generated with medium confidence

void A::process(const B &b){

…

}

* + B include edilmeli.
* Is\_A
  + This is inheritance
  + A diagram of a flowchart

    Description automatically generated

struct A::public B{

…

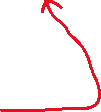
};

* + A, B’den türüyor.
* Association

A diagram of a flowchart

Description automatically generatedClass B needs functionality from class B. Options:

public  
 veya void print()  
private



B::somefunction(){  
 print();  
}

A diagram of a diagram

Description automatically generated

public void print()

B::somefunction(){  
 m\_a.print();  
}  
NOTE 🡪 m\_a means member a

Liskov Is\_A’yi tercih etmemeli diyor.

Bir sınıftan başka sınıf türediğinde, B’yi bilmeden bütün B nesneleri yerine A’mışcasına davranabilirim.A diagram of a flowchart

Description automatically generated

struct Square::public Rectangle{  
 public:  
 void setWidth(int w){  
 height = width = w;  
 }  
 void setHeight(int h){  
 height = width = h;  
 }  
 private:  
 int m\_width;  
 int m\_height;  
};

struct Rectangle{  
 public:  
 void setWidth(int w);  
 void setHeight(int h);  
 int getArea() const;  
 private:  
 int m\_width;  
 int m\_height;  
};

Bu düzgün olmadı, Liskov principle’a uygun değil.

Rectangle’ın bir kenarı 2 katına çıkarsa alan 2x olur.  
Square’in bir kenarı 2 katına çıkarsa alan 4x olur.

r->setWidth(r->getWidth\*2);

class Square{  
 void setSide(s){  
 m\_side = s;  
 }  
};

getArea yerine draw olsaydı o zaman sorun yoktu.