**#15 OOP DASAR pada PHP Interface**

Kita kembali ke kasus kita sebelum nya di bagian pemabahasan setter dan getter, kita punya 4 buah class, class utama nya class Produk dan ada 2 buah class turunan nya yaitu Game dan Komik, satu nya lagi class CetakInfoProduk.

**INTERFACE (1)**

* **Kelas abstrak** yang sama sekali tidak memiliki implementasi. Nah ketika kita punya method abstrak method tersebut hanya kita tulis nama nya saja seperti deklarasi variable tapi ini deklarasi method gitu tapi kalo **di class interface** semua nya hanya deklarasi nya saja di class interface.
* **Murni merupakan** template untuk kelas turunannya.
* **Tidak boleh** memiliki property, hanya deklarasi method saja.
* Semua method harus dideklarasikan dengan visibility **public.**
* Boleh mendeklarasikan **\_\_construct().** Untuk mewajibkan class turunan nya \_\_construct() juga.
* Satu kelas boleh mengimplementasikan banyak interface. Jadi kalian boleh punya banyak interface yg di implements ke sebuah class.

Contoh nya :

Kita harus inget mengenai kelas abstrak di video sebelum nya ginikan :

abstract class Buah {

    private $warna;

    abstract public function makan();

    public function setWarna($warna) {

**$this**->warna = $warna;

    }

}

Jadi kita tidak bisa membuat instance dari class buah, kita hanya bisa membuat instance dari class turunan nya, cara kita mengubah class buah menjadi interface ;

interface Buah {

    public function makan();

    public function setWarna($warna);

}

Nah kita tinggal nambahkan interface tanpa ada class nya dan tidak ada property didalam nya juga, dan kita boleh punya deklarasi method nya saja tanpa isi dari method nya. Jadi betul-betul murni template nanti nya class turunan nya wajib ada 2 implementasi dari method makan() dan setWarna().

Jadi itu murni class template nya dan class buah itu tidak bisa kita instasiasi jadi wajib punya class turunan nya ya, nah class turunan nya kita ubah menjadi seperti ini :

class Apel implements Buah {

    protected $warna;

    abstract public function makan(){

*// kunyah*

*// sampai bagian tengah*

    }

    public function setWarna($warna) {

**$this**->warna = $warna;

    }

}

Jika di class interface nya ada 4 function maka di seluruh class turunan nya harus ada 4 function implementasi nya tersebut gitu.

Nah sekarang misal saya tambahkan 1 interface lagi jadi saya punya 2 interfacegini :

interface Buah {

    public function makan();

    public function setWarna($warna);

}

interface Benda {

    public function setUkuran($ukuran);

}

Misal kita implementasikan Benda itu ke class Apel maka kita tulis nya gini :

class Apel implements Buah, Benda{

    protected $warna;

    protected $ukuran;

    abstract public function makan(){

*// kunyah*

*// sampai bagian tengah*

    }

    public function setWarna($warna) {

**$this**->warna = $warna;

    }

    public function setUkuran($ukuran) {

**$this**->ukuran = $ukuran;

    }

}

Nah jadi nya kita tinggal tambahkan koma saja, tapi jadi kita harus mengimplementasikan semua method yg ada didalam interface Buah dan Benda itu wajib ya.

INTERFACE (2) :

* Dengan menggunakan type-hinting dapat melakukan **‘Dependency Injection’**. Type-hiting udah kita lakukan ketika kita membahas object type ya, dimana sebuah object bisa kita jadikan sebagai parameter itu disebut nya type-hinting. Isitilah **‘Dependency Injection’** banyak sekali digunakan jadi kita memaksakan sebuah method parameter nya adalah object begitu.
* Pada akhirnya dengan kita melakukan interface ini kita akan mencapai **Polymorphism**.

**Studi Kasus**

Kita lanjutkan dari class abstrak ya, kita tidak bisa mengganti class langsung dengan interface karena dia hanya boleh berisi method kita teliti dulu ya. Gimana kalo getInfoLengkap()itu akan kita tarik menjadi interface.

Sehingga jika nanti kelas turunan nya ingin menggunakan getInfoLengkap() dia tinggal implements saja interface itu.

interface infoProduk {

    public function getInfoLengkap();

}

Karena kita sudah menarik method abstrak nya jadi class Produk sudah tidka bisa menjadi class abstrak karena syarat dari sebuah class abstrak harus mempunyai minimal 1 method abstrak, kan sudah kita ambil method nya jadi tidak ada lagi dia.

class Game extends Produk implements InfoProduk {

    public $waktuMain;

    public function \_\_construct($judul = "judul", $penulis = "penulis", $penerbit = "penerbit", $harga = 0, $waktuMain = 0) {

        parent::\_\_construct($judul, $penulis , $penerbit, $harga);

**$this**->waktuMain = $waktuMain;

    }

    public function getInfoLengkap() {

        $str = "Game : ". **$this**->getInfo() . " ~ {**$this**->waktuMain} Jam";

        return $str;

    }

}

Nah cara nya tinggal tambahin implements ke class turunan nya jadi bisa ya 2 gitu inheritence ( extends ) dan interface ( implements ) jika extends atau implements nya dari 2 class tinggal tambahin koma aja ( misal implements InfoProduk, InfoToko ) jadi dia dari 2 interface giu.

Nah dengan begitu saja sedikit merubah hasil nya sama persis dari studi kasus sebelum nya Cuma beda cara penulisan nya aja, inget ini bukan benar atau salah tapi tentang perancangan desain class nya.

Nah tapi sekarang problem nya kita bisa melakukan instasiasi Class Produk yg sebelumnya dia itu Class abstract karena tidak ada lagi method abstract jadi Class Produk biasa aja. Kalo kita kasih saja abstract tapi tidak ada method didalam class nya bisa aja tapi agak aneh ya.

Sebener nya kita bisa akalin salah satu method disitu jadi abstract misal nya yg getInfo itu deklarasi nya aja nah implementasi nya kita pindah kan ke tiap tiap class turunan nya gini :

abstract class Produk{

    abstract public function getInfo();

}

class Game extends Produk implements InfoProduk {

    public $waktuMain;

    public function getInfo() {

        $str = "{**$this**-> getJudul()} | {**$this**->getLabel()}, (Rp. {**$this**-> getHarga()})";

            return $str;

       }

    public function \_\_construct($judul = "judul", $penulis = "penulis", $penerbit = "penerbit", $harga = 0, $waktuMain = 0) {

        parent::\_\_construct($judul, $penulis , $penerbit, $harga);

**$this**->waktuMain = $waktuMain;

    }

    public function getInfoLengkap() {

        $str = "Game : ". **$this**->getInfo() . " ~ {**$this**->waktuMain} Jam";

        return $str;

    }

}

Nah ada yg kita ubah di getInfo() yaitu yg tadi nya kita panggil variable harga dan judul nya langsung sekarang kita ganti jadi method getJudul() dan getHarga() ya.