Introduction

ဒီစာအုပ်လေးမှာ OOP အကြောင်းပြောမှာဆိုတော့ OOP ဆိုတာကိုအရင် မိတ်ဆက်ပေးပါမယ်။ သူ့ရဲ့ long form ကတော့ object-oriented programming ဖြစ်ပါတယ်။ Wiki ကတော့ ဒီလိုပြောထားပါတယ်။

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects"

OOP ဆိုတာ objects ဆိုတဲ့ သဘောတရားတွေပေါ်မှာ အခြေခံထားတဲ့ programming paradigm တစ်ခုဖြစ်ပါတယ်ဆိုပြီးတော့ ပြောထားပါတယ်။ ဒီနေရာမှာ programming paradigm ဆိုတဲ့ အသုံးအနှုန်းကို ဆက်ပြီးတော့ ကျနော်ရှင်းပြပါမယ်။ နားလည်ရလွယ်တဲ့ sequential နဲ့ procedural programming အကြောင်းကို ဥပမာပေးပြီးပြောပါမယ်။

Sequential programming ဆိုတာက code လိုင်းတွေကို statement တစ်ခုခြင်းဆီ execution လုပ်သွားတာမျိုးကိုပြောတာ၊ ဥပမာ နမူနာပေးရမယ်ဆို

```
1 $a = 1;
2 $b = 2;
3 $plus = $a + $b;
4 echo $plus;
```

ဒါဆိုလို့ရှိရင် အပေါ်က code block ထဲမှ variables တွေပါတယ်၊ ပြီးတော့ arithmetic + operator ကိုသုံးပြီးတော့ ပေါင်းခြင်းလုပ်သွားတယ်၊ နောက်တစ်ကြောင်းမှာ echo ဆိုပြီးရလဒ်ကို print ထုတ်ပြတယ်။ အခုပြထားတာ sequential programming ရဲ့ နမူနာတစ်ခု၊ ဒီနေရာမှာပြောစရာရှိတာက ပေါင်းရမယ့် number operation နောက်တစ်ကြိမ်ထပ်လုပ်ရမယ်ဆိုလဲ ဒီလိုမျိုး code ထပ်ရေးရဦးမှာပဲ။ ဒီနေရာမှာ procedural programming ဆိုတာက အသုံးဝင်လာတယ်။

Procedural programming ကတော့ sequential ရဲ့အားနည်းချက်တွေက cover လုပ်ပေးသွားနိုင်တယ်။ sequential နဲ့ရေးလာမယ်ဆို နောက်ပိုင်းမှာ ကိုယ့် project code တွေကတစ်အားရှုပ်လာမယ်။ code line တွေမလိုအပ်ဘဲတစ်အားများလာလိမ့်မယ်။ အဲ့လို ပြဿနာတွေကို procedural နဲ့ဖြေရှင်းလို့ရမယ်၊ ဘယ်လိုလဲဆိုတော့ procedural မှာ function ဆိုတာပါလာတယ်၊ function ကို သုံးပြီးတော့ sequence code တွေကိုစုစုစည်းစည်းထားလိုက်မယ်ဆို နောက်တစ်ကြိမ် အလားတူ logic တွေလာပြီဆို functionလေးကို reuse လုပ်လိုက်ရုံပဲ၊ မလိုအပ်တဲ့ code တွေလျော့ချနိုင်သလို complexity လည်းနည်းသွားမယ်။ ဥပမာ အပေါ်က sequential ကို procedural အဖြစ်ပြောင်းမယ်ဆို

```php
function addNum($a,$b)
{
  $plus = $a+$b;
  echo $plus;
}

//addNum(1,2);
```

ခေါ်သုံးချင်ပြီဆို addNum(1,2) ဆိုပြီးခေါ်လိုက်ရုံပဲ၊ ဒါဆိုလို့ရှိရင် နောက် plus operation တွေရှိတိုင်းမှာ sequence မှာတုန်းကလို လိုင်းငှကြောင်းရေးစရာမလိုတော့ဘဲ တစ်ကြောင်းတည်းနဲ့ ပြတ်သွားလိုက်မယ်။ နောက်တစ်ခုတွေ့မှာက function အနောက်မှာပါလာတဲ့ param လေးတွေပဲ၊ နောက်ပိုင်း calculation လုပ်တဲ့အချိန် မတူညီတဲ့ digit တွေအတွက် addNum လို့ခေါ်တဲ့အချိန်မှာတည်းက တစ်ခါတည်းထည့်ပေးလိုက်လို့ရအောင်လုပ်ပေးထားတယ်။ ဒါကြောင့်မို့လို့ procedural နဲ့ဆိုလို့ရှိရင် sequence မှာကြုံလာရမယ့် code ထပ်မယ့် ပြဿနာတွေကိုရှောင်လို့ရသွားလိုက်မယ်။

အပေါ်ကပြောခဲ့တဲ့ sequential နဲ့ procedural ဆိုတာ programming paradigm တွေပဲ၊ paradigm ကိုမြန်မာလိုဘာသာပြန်ရရင် ရေးထုံးပုံစံတွေပေါ့။ OOP ဆိုတာကလည်း object သဘောတရားကို အခြေခံထားတဲ့ paradigm တစ်ခုပဲ၊ သူမှာပါတဲ့ ရေးထုံးတွေကိုတော့ ကျနော်တို့ တစ်ဖြည်းဖြည်းခြင်းလေ့လာသွားကြတာပေါ့။ အဓိကကတော့ OOP မှာရှိတဲ့ ရေးထုံးပုံစံတွေကြောင့် code ရေးရတဲ့နေရာမှာ ပိုပြီးတော့ သပ်သပ်ရပ်ရပ်ဖြစ်သွားမယ်၊ ထိန်းသိမ်းရတာပိုလွယ်ပြီးတော့

ပြန်လည်အသုံးပြုရတဲ့နေရာမှာ ကောင်းမွန်သွားမယ့်အပြင် တစ်ခြားသော ကောင်းကျိုးတွေလည်း အများကြီးရှိပါတယ်။ ဒါကြောင့်လည်း programming language အားလုံးတိုင်းလိုလိုမှာ OOP ဆိုတဲ့ ရေးထုံးကို support လုပ်လာခဲ့ကြတာဖြစ်ပါတယ်။

Class and Object

အပေါ်မှာ ကျနော်ပြောခဲ့သလိုပဲ OOP ဆိုတာ object သဘောတရားတွေနဲ့ ဖွဲ့စည်းထားတယ်ဆိုတဲ့အတွက် class နဲ့ object ဆိုတာ OOP ရဲ့အဓိက ပင်မထောက်တိုင်တွေလို့ သတ်မှတ်လို့ရပါတယ်။ Class ဆိုတာက object ကို ခြုံငုံထားတဲ့ template တစ်ခုလို့ဆိုလို့ရပါတယ်။ မြင်သာအောင်ရှင်းပြရရင် နိုင်ငံ ဆိုတဲ့ ဝေါဟာရက class ဖြစ်ပြီးတော့ မြန်မာ၊ ဂျပန်၊ အမေရိကန် စတဲ့ အရာတွေက object အဖြစ်သတ်မှတ်နိုင်ပါတယ်။ တစ်နည်းအားဖြင့် ဆိုရရင် class ဆိုတာက logicial entity ဖြစ်ပြီးတော့ object ကတော့ physical entity အဖြစ်တည်ရှိပါတယ်။ နိုင်ငံ ဆိုတဲ့ဝေါဟာရဟာ အပြင်မှာ လက်တွေ့မရှိပါဘူး၊ စကားလုံးဝေါဟာရအဖြစ်သာ logical entity အဖြစ်သတ်မှတ်ထားတာဖြစ်ပြီးတော့ မြန်မာ ဆိုတဲ့ အရာကတော့ လက်တွေ့မှာ တစ်ကယ်တည်ရှိတဲ့ physical entity ဖြစ်ပါတယ်။ နောက်တစ်ခု ဥပမာ ထပ်ပေးရရင် Animal ဆိုတဲ့ class မှာ dog, cat, bird အစရှိတဲ့ object တွေရှိသလိုမျိုးပေါ့။ အောက်က code example လေးကိုဆက်ကြည့်ရအောင်။

```php
<?php

class Country {
  // property
  public $name = 'Myanmar';

  // method
  public function getName() {
    return $this->name;
  }
}
?>
```

ကျနော် country ဆိုပြီး class လေးတစ်ခုဆောက်ထားပါတယ်။ အဲ့ဒီ class ထဲမှာ name ဆိုတဲ့ property လေးထည့်ထားတယ်၊ value ကို Myanmar ဆိုပြီးတစ်ခါတည်းပေးထားပါတယ်။ နောက်တစ်ခု getName ဆိုတဲ့ method လေးရေးထားတယ်၊ သူကတော့ class ထဲမှာရှိတဲ့ name property ကို return ပြန်ပေးပါမယ်။ ဒါကရိုးရိုးရှင်းရှင်း class လေးတစ်ခုဖြစ်ပါတယ်။ အဲ့ဒီ class ကနေ object ဆိုတာဘယ်လိုဖြစ်လာလဲဆက်ကြည့်ရအောင်။

```php
<?php

class Country {
  // property
  private $name = 'Myanmar';

  // method
  public function getName() {
    return $this->name;
  }
}

$myanmar = new Country();
var_dump($myanmar);
//output = object(Country)#1 (1) { ["name"]=> string(7) "Myanmar" }
var_dump($myanmar->getName());
//output = string(7) "Myanmar"

?>
```

Country class ရဲ့ အောက်မှာ Myanmar ဆိုတဲ့ variable လေးတစ်ခုဆောက်ထားပြီး new ဆိုတဲ့ keyword နဲ့ country class ကိုလှမ်းခေါ်ထားပါတယ်။ ဒီလိုမျိုးလုပ်တဲ့ process ကို instantiate လုပ်တယ်လို့ခေါ်ပါတယ်။ country class ကို instantiate လုပ်ပြီးလို့ ရလာတဲ့ myanmar ဆိုတဲ့ variable သည် object ဖြစ်ပါတယ်။ var_dump နဲ့ ထုတ်ကြည့်မယ်ဆို country object တစ်ခုရလာမှာဖြစ်ပါတယ်။ နောက်တစ်ဆင့်အနေနဲ့ class ထဲမှာရှိနေတဲ့ getName ဆိုတဲ့ method ကို myanmar object ကနေတစ်ဆင့်ခေါ် ကြည့်မယ်ဆို return value အနေနဲ့ myanmar ဆိုတဲ့ string output ပြန်ရလာမှာဖြစ်ပါတယ်။ ဒါကတော့ ရိုးရိုးရှင်းရှင်းရှင်း class and object ပဲဖြစ်ပါတယ်။

အပေါ်က code လေးထဲမှာ property ကို private လို့ပေးထားပြီး method ကို public function လို့ပေးထားတာမြင်ပါလိမ့်မယ်။ လက်ရှိအခြေအနေမှာ myanmar object ထဲကနေ function ကိုလှမ်းခေါ်သလို property ကို myanmar->name လို့လှမ်းခေါ်မယ်ဆိုရင် error တတ်ပါလိမ့်မယ်။ ဘာလို့လဲဆိုတာ နောက်တစ်ပိုင်းမှာရှင်းပြပေးပါမယ်။

Visibility & Access Modifiers

ကျနော်တို့ အပေါ်က code example တွေကနေတစ်ဆင့် တွေ့ခဲ့ရတာက private နဲ့ public ပဲတွေ့ခဲ့ရပါသေးတယ်။ ဒါလေးတွေကို access modifiers တွေလို့ခေါ် ပြီးတော့ public, protected & private ဆိုပြီးသုံးမျိုးရှိပါတယ်။

Public ကိုသုံးထားတယ်ဆို သူ့ရဲ့ property တွေ method တွေကို သူ့ကို create လုပ်ထားတဲ့ class ရဲ့ ပြင်ပကနေလည်း လှမ်းပြီးတော့ ခေါ်ယူအသုံးပြုလို့ရတယ်။

Private ကိုသုံးထားတယ်ဆိုရင်တော့ဖြင့် သူ့ကို create လုပ်ထားတဲ့ class အတွင်းမှာပဲ ခေါ်ယူအသုံးပြုနိုင်မယ်။ class ရဲ့ပြင်ပကနေ လှမ်းခေါ်သုံးလို့မရဘူး။

Public & private ကိုပဲအရင် ဥပမာ ကြည့်ရအောင်။

```php
<?php
class Person {
  public $name = 'Hlaing';
  private $age = '23';
}

$man = new Person();
echo $man->name; // output = Hlaing
echo $man->age; // can't access private property error
?>
```

Protected ကလည်း private နဲ့သဘောတရား အတူတူပဲ။ ဒါပေမယ့် protected ကတော့ မူလသူ့ကို create လုပ်ခဲ့တဲ့ class ကို ပြန် inheritance လုပ်ထား class ထဲမှာတော့ လှမ်းခေါ်သုံးခွင့်ရှိတယ်။ မလုပ်ထားရင်တော့ ခေါ်သုံးခွင့်မရှိဘူး။ ဒီနေရာမှာ ကျနော် inheritance ဆိုတာကိုပြောသွားတယ်။ လောလောဆယ်မှာ inheritance ဆိုတာ အမွေဆက်ခံယူထားတဲ့ class အဖြစ်ဘဲ ကျနော်တို့ သတ်မှတ်ထားရအောင်၊ ဥပမာ Parent Class ကို Child Class ကနေ လှမ်းပြီး inheritance လုပ်သလိုမျိုးပေါ့။

```php
<?php
class ParentClass {
  public $name = 'Hlaing';
}

// Inheritance (From Parent to child)
class ChildClass extends ParentClass {

}

$child = new ChildClass();
echo $child->name; //output Hlaing
?>
```

အပေါ်က ဥပမာ ကိုကြည့်မယ်ဆို childclass မှာ property name ကမရှိပေမယ့် ParentClass ဆီကနေတစ်ဆင့် အမွေဆက်ခံ(Inheritance) ထားတဲ့အတွက် Parent ထဲမှာရှိတဲ့ property ကိုပါ လှမ်းပြီးတော့ အသုံးပြုနိုင်နေတာပဲဖြစ်ပါတယ်။ နောက်ပိုင်းမှာ ဒီအတွက် အပိုင်းတစ်ပိုင်း သက်သက်လာမှာဆိုတော့ ဒီလောက်ပဲသိထားရင် အဆင်ပြေပါတယ်။ protected ကို inheritance နဲ့ ဘယ်လိုလှမ်းခေါ်လို့ရနိုင်မလဲဆိုတာ code example လေးနဲ့ တစ်ချက်ဆက်ကြည့်ရအောင်။

```php
<?php
class Person {
  public $name = 'Hlaing';
  protected $age = '23';

}

// Inheritance
class Hlaing extends Person {
  public function output() {
    //calling protected property `age` using `this` keyword
    echo $this->age;
  }
}

$man = new Hlaing();
$man->output(); // output = 23
$man->age; // protected property can't access outside of the class
?>
```

Hlaing ဆိုတဲ့ class ကနေပြီးတော့ Person class ဆီကနေ inheritance လှမ်းလုပ်လိုက်ပါတယ်။ Hlaing class ထဲမှာ public function တစ်ခုဆောက်ပြီးတော့ Person class က protected property ဖြစ်တဲ့ age ကို ***this*** ဆိုတဲ့ keyword လေးသုံးပြီးတော့ echo ထုတ်လိုက်ပါတယ်။ inheritance လုပ်ထားတဲ့အတွက်ကြောင့်သာ ဒီလို လှမ်းခေါ်သုံးနိုင်တာဖြစ်ပါတယ်။ အောက်က client code မှာ Hlaing class ကို instantiate လုပ်ပြီးတော့ public function ဖြစ်တဲ့ output ကိုလှမ်းခေါ်လိုက်မယ်ဆို Person class မှာ protected ဖြစ်နေတဲ့ age value ကိုရလာမှာဖြစ်ပြီးတော့ person class က protected property age ကိုတိုက်ရိုက်လှမ်းခေါ်မယ်ဆိုရင်တော့ error တတ်မှာဖြစ်ပါတယ်။

Inheritance

Inheritance အကြောင်းကို နောက်မှပြောဖို့လုပ်ထားပေမယ့် နောက်လာမယ့် အပိုင်းတွေမှာ inheritance အကြောင်းပါ သိမှရမှာလေးတွေရှိလာတော့ Inheritance ကို စောစောစီးစီးပဲ ပြောပြထားပါမယ်။ Inheritance လုပ်တယ်ဆိုတာ တစ်ခြားတော့ မဟုတ်ဘူး၊ ကျနော် အပေါ်မှာလည်း ပြောခဲ့တာတော့ ရှိပါတယ်။ မြန်မာမှုပြုရင် အမွေလက်ခံတာပေါ့၊ လက်တွေ့ဘဝနဲ့ ဆက်စပ်ပြီး ကြည့်မယ်ဆို သားသမီးက မိဘဆီကနေ အမွေလက်ခံရယူမယ်ဆို မိဘတွေပိုင်ဆိုင်တဲ့ အိမ်၊ခြံ၊ကား အစရှိတာတွေကို ပိုင်ဆိုင်ခွင့်ရမယ်။ OOP မှာလည်း ထိုနည်းလည်းကောင်းပဲ၊ Child Class ကနေ ပြီးတော့ Parent Class ကို အမွေလက်ခံ (Inheritance) လုပ်လိုက်မယ်ဆိုရင် Parent Class မှာရှိနေတဲ့ properties တွေ methods တွေကို လှမ်းပြီးတော့ ခေါ်ယူအသုံးပြုနိုင်မှာဖြစ်ပါတယ်။ အသစ်မှတ်ထားရမှာကတော့ inheritance လုပ်တော့မယ်ဆို ***extends*** ဆိုတဲ့ keyword လေးကို သုံးပါတယ်။

```php
<?php

// Parent Class
class Person {
  public $name = "Hlaing";

  public function getName() {
    echo $this->name;
  }
}

// Child class (inheritance using extends keyword)
class Man extends Person {
}

$man = new Man();
echo $man->name; //output - Hlaing
$man->getName(); //output - Hlaing

?>
```

အထက်ပါ code နမူနာလေးကိုကြည့်မယ်ဆိုရင် Man ဆိုတာက child class ဖြစ်ပြီးတော့ Person ဆိုတာကတော့ Parent Class ဖြစ်ပါတယ်။ Parent class မှာ name ဆိုတဲ့ property နဲ့ getName

ဆိုတဲ့ method ရှိပါတယ်။ Child class ဖြစ်တဲ့ Man ထဲမှာတော့ ဘာမှမရှိပါဘူး၊ ဒါပေမယ့် extends keyword လေးကို သုံးထားပြီး Person class ဆီကနေ inheritance လုပ်ထားပါတယ်။ ဒါကြောင့် Person ဆိုတဲ့ Parent Class ထဲမှာရှိတဲ့ property ဖြစ်တဲ့ name နဲ့ method ဖြစ်တဲ့ getName တို့ကို လှမ်းပြီးတော့ ခေါ်ယူအသုံးပြုနိုင်ခြင်းဖြစ်ပါတယ်။

နောက်တစ်ခုထပ်သိထားဖို့ကောင်းတာက ခေါ်ယူအသုံးပြုရုံပဲသာမကပဲ Parent Class ထဲမှာရှိတဲ့ method တွေကို child class က overwrite လုပ်လိုက်လို့ရပါတယ်၊ ကိုယ်လိုသလို ထပ်ပြီးပြင်ဆင်လို့ ရတယ်လို့ ဆိုလိုချင်တာဖြစ်ပါတယ်။

```php
<?php

// Parent Class
class Person {
  public $name = "Hlaing";

  public function getName() {
    echo $this->name;
  }
}

// Child class (inheritance using extends keyword)
class Man extends Person {
  //overwriting the parent class's method.
  public function getName(){
    echo "Hlaing Tin Htun";
  }
}

$man = new Man();
echo $man->name; //output - Hlaing
$man->getName(); //output - Hlaing Tin Htun (because we overwrote it)

?>
```

အပေါ်က နမူနာလေးကိုထပ်ကြည့်ရမယ်ဆို ပုံမှန် parent class ထဲမှာရှိတဲ့ getName ဆိုတဲ့ method ဟာ အရင်အတိုင်းဆို Hlaing ဆိုတာကိုပဲ output ပြန်ပေးမှာဖြစ်ပါတယ်၊ ဒါပေမယ့် Man ဆိုတဲ့ child class ထဲမှာ getName ဆိုတဲ့ method ကို overwrite လုပ်ပြီး Hlaing Tin Htun ဆိုပြီး return ပြန်လိုက်တဲ့အတွက် ထွက်လာတဲ့ output သည်လည်း Hlaing သာမဟုတ်တော့ဘဲ Hlaing Tin Htun ဖြစ်သွားပါလိမ့်မယ်။

တစ်ခု သတိချပ်ရမှာက inheritance လုပ်လိုက်ခြင်းသည် public နဲ့ protected ဖြစ်တဲ့ properties & methods တွေကိုသာ လှမ်းပြီးတော့ access လုပ်နိုင်မယ်ဆိုတာ သတိချပ်ထားရပါမယ်။ ကျနော် အပေါ်မှာ access modifiers အကြောင်းရှင်းပြတုန်းက protected တွေကို ဘယ်လိုပြန်ခေါ် သုံးနိုင်တယ်ဆိုတာ ရှင်းပြထားပြီးသားဖြစ်တဲ့အတွက်ကြောင့် နောက်တစ်ခေါက်ထပ်ပြီးတော့ ဒီမှာမရေးပြတော့ပါဘူး။ ပြန်ကြည့်ရလွယ်အောင် code sample တော့ ပြန်ထည့်ပေးထားပါမယ်။

```php
<?php
class Person {
  public $name = 'Hlaing';
  protected $age = '23';

}

// Inheritance
class Hlaing extends Person {
  public function output() {
    //calling protected property `age` using `this` keyword
    echo $this->age;
  }
}

$man = new Hlaing();
$man->output(); // output = 23
$man->age; // protected property can't access outside of the class
?>
```

Inheritance ကတော့ ဒီလောက်ပါပဲ၊ တစ်ခုလေးပဲထည့်ပြောပါရစေ။ ကျနော်တို့အခုဆို Person class ကို parent class အဖြစ်သဘောထားပြီး man class ကနေလှမ်းပြီး inheritance လုပ်နေတယ်။ ဒါပေမယ့် Person class ကိုဘယ်သူကမှ လာပြီး inherit မလုပ်စေချင်ရင်ရော ? ဒီနေရာမှာ ***final*** ဆိုတဲ့ keyword ကိုသုံးနိုင်ပါတယ်။ final ဆိုတဲ့ keyword ကိုသုံးလိုက်မယ်ဆို နောက်ထပ် ဘယ် class ကမှ Person class ကို လာပြီး inherit လုပ်လို့မရတော့ပါဘူး။ ဒီလိုမျိုးပေါ့။

```php
<?php

final class Person {
}

class Man extends Person {
//this will output error
//Class Man may not inherit from final class (Person)
}

?>
```

Class ကို inherit မလုပ်နိုင်အောင် ထိန်းလိုက်တာကတော့ ဟုတ်ပါပြီ၊ အပေါ်မှာပြောခဲ့တဲ့ parent class မှာရှိတဲ့ method ကို child class က overwrite မလုပ်နိုင်အောင် ထိန်းချင်တယ်ဆိုရင်ရော ? ဒါဆိုရင်လည်း final keyword ကို အသုံးပြုပြီး ကာကွယ်နိုင်ပါတယ်။

```php
<?php

// Parent Class
class Person {
  public $name = "Hlaing";
  //prevent method overwriting using final keyword
  final public function getName() {
    echo $this->name;
  }
}

// Child class (inheritance using extends keyword)
class Man extends Person {
  //overwriting the parent class's method.
  //But can't overwrite this time because of final keyword
  //Cannot override final method Person::getName()
  public function getName(){
    echo "Hlaing Tin Htun";
  }
}

?>
```

Parent class မှာ method ကို ကြေညာကတည်းက final ခံပြီးကြေညာခဲ့မယ်ဆို child class တွေက overwrite လုပ်နိုင်မှာ မဟုတ်တော့ဘဲ error တတ်မှာဖြစ်ပါတယ်။ ဒီလောက်ဆို Inheritance ကို သဘောပေါက်ပြီထင်ပါတယ်။ နောက်တစ်ခုဆက်သွားကြတာပေါ့။

Static Properties & Methods

ပုံမှန်အပေါ်မှာ ကျနော်တို့ လေ့လာခဲ့တဲ့အရာတွေအရ class ထဲမှာရှိတဲ့ properties တွေ methods တွေကို လှမ်းခေါ်သုံးချင်ပြီဆို ***new*** ဆိုတဲ့ keyword နဲ့ instantiate လုပ်ပြီးမှ ခေါ်သုံးဖြစ်ခဲ့ကြပါတယ်။ static properties တွေ methods တွေရဲ့ ထူးခြားတဲ့အချက်ကတော့ အဲ့လိုမျိုး class ကို instantiate လုပ်စရာမလိုဘဲ တိုက်ရိုက်ခေါ်ယူအသုံးပြုနိုင်မှာ ဖြစ်ပါတယ်။ ကျနော်တို့ property အကြောင်းအရင် လေ့လာကြပါမယ်၊ ပြီးမှ methods တွေကိုဆက်ကြတာပေါ့။ နမူနာ static property လေးတစ်ခု အရင်ဆုံး ကြည့်ရအောင်။

```php
<?php

class Country {
  //static property
  public static $name = 'Myanmar';
}

//using double colon to call static property
echo Country::$name;
//output - Myanmar

?>
```

အပေါ်က နမူနာ country class ထဲမှာ static property တစ်ခုကြေညာထားပါတယ်။ ထူးထူးခြားခြား ဘာမှလုပ်ပေးစရာမလိုဘူး။ ***static*** ဆိုတဲ့ keyword လေးထည့်ပြီး ကြေညာပေးရုံပဲ၊ ပြန်လည် ခေါ်သုံးလိုတဲ့အချိန်မှာတော့ country class ကို new ဆိုပြီး instantiate လုပ်စရာမလိုတော့ဘဲ classname, double_colon, property_name (***Country::$name***) ဆိုပြီး တိုက်ရိုက် ခေါ်ယူ အသုံးပြုနိုင်သွားမှာဖြစ်ပါတယ်။ ဒီနေရာမှာ အသစ်မှတ်ထားရမှာက static property ကိုခေါ်တော့မယ်ဆို **double colon (::)** သုံးရမယ်ဆိုတာပါပဲ။

Static properties တွေကို သူ့ရဲ့မူလ class ထဲမှာပဲ ပြန်ခေါ်သုံးချင်ပြီဆိုရင် ***self*** ဆိုတဲ့ keyword နဲ့ double colon ပေါင်းပြီးတော့ ***(self::)*** ပြန်လည်ခေါ်ယူအသုံးပြုနိုင်မှာဖြစ်ပြီးတော့ inherit

လုပ်ထားတဲ့ child class ကပြန်ခေါ်သုံးချင်တယ်ဆိုရင်တော့ self အစား ***parent*** ဆိုတဲ့ keyword နဲ့ ခေါ်ယူအသုံးပြုနိုင်မှာဖြစ်ပါတယ်။

အရင်ဆုံး မူလ class ထဲမှာပဲ static property ကိုပြန်ခေါ်သုံးကြည့်ရအောင်။

```php
<?php

class Country {
  //static property
  public static $name = 'Myanmar';

  //normal function calling static property using self::
  public function getName()
  {
    echo self::$name;
  }
}

// have to instantiate the class as we are calling normal function getName
$country = new Country();
$country->getName(); //output - Myanmar

?>
```

အရင်အတိုင်း static property name ရှိမယ်၊ ပြီးတော့ ပုံမှန် getName ဆိုတဲ့ method တစ်ခုရှိမယ်၊ အဲ့ဒီ method ထဲမှာ class ထဲမှာကြေညာထားတဲ့ static property name ကို ***self*** ဆိုတဲ့ keyword နဲ့ လှမ်းခေါ်ထားတာကို မြင်ရမှာဖြစ်ပါတယ်။ အောက်မှာတော့ (client code) country ဆိုတဲ့ class ကို instantiate လုပ်ထားတယ်၊ ခေါ်သုံးမယ့် getName က static method မဟုတ်တဲ့အတွက် class ကို instantiate လုပ်ဖို့ လိုအပ်တာဖြစ်ပါတယ်။

နောက်ထပ် scenario တစ်ခုကတော့ inherit လုပ်ထားတဲ့ child class ကနေ parent class မှာရှိတဲ့ static property ကို လှမ်းခေါ်မှာဖြစ်ပါတယ်။

```php
<?php

class Country {
  //static property
  public static $name = 'Myanmar';
}

class Myanmar extends Country{
  public function getName()
  {
    echo parent::$name;
  }
}

$country = new Myanmar();
$country->getName(); //output - Myanmar

?>
```

သဘောတရားက အပေါ်မှာ ကျနော်တို့ လေ့လာခဲ့တဲ့ အရာနဲ့ အတူတူပါပဲ၊ ကွာသွားတာကတော့ child class က parent class မှာရှိနေတဲ့ static property ကို လှမ်း access လုပ်တော့မယ်ဆို self ဆိုတဲ့ keyword အစား ***parent*** ဆိုတဲ့ keyword ကို သုံးရမှာဖြစ်ပါတယ်၊ အနောက်မှာ **double colon (::)** လိုက်ရမယ်ဆိုတာလဲ သတိချပ်ထားရပါမယ်။

အပေါ်က ပြောခဲ့တဲ့ static property အလုပ်လုပ်ပုံကိုနားလည်သွားပြီဆို method အတွက်လည်း အလိုအလျောက်နားလည်သွားမှာပါ၊ ကွာခြားချက်မရှိပါဘူး၊ အသုံးအနှုန်းက အတူတူပါပဲ။ method ကိုခေါ်တော့မယ်ဆို classname, double_colon, method_name လာမယ်။ ဒီအတွက် ဥပမာ တစ်ခု တစ်ခါတည်း ကြည့်ရအောင်။

```php
<?php

class Country {
  //static method
  public static function getName()
  {
    echo 'Myanmar';
  }
}

Country::getName(); //output - Myanmar

?>
```

အထူးတစ်လည်ရှင်းပြစရာမလိုလောက်တော့ဘူးထင်ပါတယ်၊ country class ထဲမှာ static method တစ်ခုကြေညာထားပြီးတော့ client code မှာ class name ကို double colon ခံပြီးတော့ method ကို တိုက်ရိုက်ခေါ်ယူအသုံးပြုနိုင်မှာဖြစ်ပါတယ်။

Property တွေလိုပဲ ***self*** နဲ့ ***parent*** အသုံးချပုံကအတူတူပါပဲ၊ မူလ class ထဲမှာပဲ method ကိုပြန်လည်အသုံးပြုချင်တယ်ဆို ***self*** ကိုသုံးပြီးတော့ inherit လုပ်ထားတဲ့ child class ကနေ အသုံးပြုချင်ရင်တော့ ***parent*** ဆိုတဲ့ keyword ကိုအသုံးပြုရမှာဖြစ်ပါတယ်။

မူလ class ထဲမှာပဲ static method ကိုပြန်လည်အသုံးပြုပုံ

```php
<?php

class Country {
  //static method
  public static function getName()
  {
    echo 'Myanmar';
  }

  public function return()
  {
    echo self::getName();
  }
}

$country = new Country();
$country->return();

?>
```

Static property မှာသုံးခဲ့တာနဲ့ အတူတူပါပဲ၊ မူလ class ထဲမှာပဲဆို **self** ဆိုတဲ့ keyword ကိုသုံးပြီးတော့ double colon ခံကာ method name ကိုလှမ်းခေါ်နိုင်မှာဖြစ်ပါတယ်။ *(self::getName())*

Inherit လုပ်ထားတဲ့ child class ကနေ parent class ထဲမှာရှိနေတဲ့ method ကိုပြန်လည်အသုံးပြုပုံ

```php
<?php

class Country {
  //static method
  public static function getName()
  {
    echo 'Myanmar';
  }
}

class Myanmar extends Country{
    public function return()
    {
        echo parent::getName();
    }
}

$country = new Myanmar();
$country->return();

?>
```

သုံးတဲ့ပုံစံ အတူတူပါပဲ၊ self နေရာမှာ **parent** အဖြစ် အစားထိုးပြောင်းလဲသွားတာကိုပဲ မြင်ရမှာဖြစ်ပါတယ်။

Interfaces

Class ဆိုတာကို ရင်းနှီးပြီးသားဆိုရင် Interface ကိုလဲ နားလည်ရလွယ်မှာပါ။ Interface ဆိုတာ ဟာလည်း တစ်နည်းအားဖြင့် class တစ်ခုပါပဲ၊ Interface ဆိုတဲ့ class ထဲမှာ ဘယ် methods တွေ properties တွေရှိမယ်ဆိုတာကို သတ်မှတ်ပေးရပါတယ်။

အဲ့လို သတ်မှတ်လိုက်ခြင်းအားဖြင့် type တူတဲ့ class တွေကို group လုပ်ပေးနိုင်ပါတယ်။ ဥပမာနဲ့ မြင်အောင် ပြောရရင် Myanmar , Japan အစရှိတဲ့ class တွေသည် Country ဆိုတဲ့ Interface အောက်မှာရှိပြီးတော့ တူညီတဲ့ methods & properties တွေရှိနိုင်ပါတယ်၊ Myanmar မှာလည်း populations ဆိုတဲ့ property ရှိနိုင်သလို Japan မှာလည်း populations ဆိုတဲ့ property ရှိပါတယ်။ နှစ်ခုလုံးက နိုင်ငံတွေဖြစ်တဲ့အတွက် နိုင်ငံတစ်ခုမှာရှိသင့်တဲ့ properties & methods တွေက တူညီနိုင်တဲ့အတွက် (Type တူတဲ့အတွက်) Country ဆိုတဲ့ Interface တစ်ခုအောက်ကနေ implement လုပ်သွားနိုင်မှာဖြစ်ပါတယ်။ နောက်ထပ် ဥပမာ တစ်ခုထပ်ပေးရရင် cat & dog class တွေမှာ Animal ဆိုတဲ့ interface ထုတ်လို့ ရတယ်ဆိုတဲ့ သဘောပေါ့၊ Cat ရော dog ရော animal တွေဖြစ်ပြီးတော့ လမ်းလျှောက်နိုင်မယ်၊ အစာစားနိုင်မယ် ဆိုတဲ့ Animal တစ်ကောင်ရဲ့ ရှိသင့်တဲ့ methods & properties တွေရှိမှာဖြစ်ပြီး Class Type ကလည်း တူမှာဖြစ်ပါတယ်။ Interface တစ်ခုဆောက်ကြည့်ရအောင်။

```php
<?php
//declaring a interface using `interface` keyword
interface Country {
  public function currentPopulation();
  public function isAsianCountry();
}

//implementing interface using `implements` keyword
class Myanmar implements Country {

}

$myanmar = new Myanmar();

?>
```

အပေါ်မှာ ကျနော် interface တစ်ခုဆောက်ပြထားပါတယ်။ interface ဆောက်တဲ့အချိန်မှာ သုံးပေးရမယ့် keyword ကလဲ ***interface*** ပဲဖြစ်ပါတယ်။ အနောက်မှာတော့ interface ရဲ့နာမည်လိုက်ပါတယ်။ လက်ရှိမှာတော့ interface နာမည်ကို country လို့ပေးထားပြီးတော့ အထဲမှာ currentPopulation နဲ့ isAsianCountry ဆိုတဲ့ methods နှစ်ခုကို တွေ့ရမှာဖြစ်ပါတယ်။ တစ်ခု သတိထားမိမှာက အဲ့ဒီနှစ်ခုလုံးကို ကြေညာပဲ ကြေညာထားပြီးတော့ implement လုပ်ထားခြင်း မရှိပါဘူး။ အဲ့လိုမျိုး implementation မရှိဘဲ declaration ဘဲရှိတဲ့ methods တွေကို abstract methods တွေလို့ခေါ်ပါတယ်။ Interface ထဲမှာ abstract methods တွေပဲ ကြေညာလို့ရမယ်ဆိုတာကိုလည်း သတိထပ်ချပ်ထားသင့်ပါတယ်။

Implementation လုပ်ရမယ့် code တွေက သူ့ကို implements လုပ်မယ့် class တွေ (concrete classes) မှာပဲရှိပါမယ်။ လက်ရှိအခြေအနေမှာ Myanmar ဆိုတဲ့ concrete class က Country ဆိုတဲ့ interface ကို implement လုပ်ထားတာကိုတွေ့ရပါမယ်။ ***implements*** ဆိုတဲ့ keyword လေးကို သုံးပါတယ်။ ဒါပေမယ့်လက်ရှိအခြေအနေမှာ Myanmar ဆိုတဲ့ concrete class ထဲမှာ Country Interface ထဲက ကြေညာထားတဲ့ abstract methods တွေကို implement လုပ်ထားခြင်း မရှိသေးပါဘူး။ ဒါကြောင့် အခုလက်ရှိ code ကို run လိုက်မယ်ဆို implement မလုပ်ရသေးဘူးဆိုတဲ့ error exception တတ်မှာဖြစ်ပါတယ်။ ဒါကြောင့် Implementation လုပ်လိုက်ရအောင်။

```php
<?php

class Myanmar implements Country {
  public function currentPopulation() {
    echo "53.71 million";
  }

  public function isAsianCountry() {
    echo "true";
  }
}

$myanmar = new Myanmar();
$myanmar->currentPopulation();// 53.71 million
$myanmar->isAsianCountry();// true

?>
```

Myanmar ဆိုတဲ့ concrete class ကို interface ထဲမှာရှိတဲ့ methods တွေအတိုင်း implementation လုပ်ပေးလိုက်ပြီဖြစ်ပါတယ်။

နောက်ထပ် country အသစ်တစ်ခု (concrete class) ထပ်ထည့်ကြည့်ရအောင်။

```php
<?php

class Japan implements Country {
  public function currentPopulation() {
    echo "126.5 million";
  }

  public function isAsianCountry() {
    echo "true";
  }
}

$japan = new Japan();
$japan->currentPopulation();//126.5 million
$japan->isAsianCountry();//true

?>
```

Japan ဆိုတဲ့ concrete class အသစ်တစ်ခုထပ်ထည့်ထားပြီးတော့ ထုံးစံအတိုင်း Country interface ထဲမှာရှိတဲ့ methods တွေကို implement လုပ်ပေးထားပါတယ်။ နောက်ထပ်လည်း လိုအပ်သလို country type တူတဲ့ concrete class တွေကို country interface ကိုသုံးပြီးတော့ implementation လုပ်လို့ရနိုင်သွားမှာပဲဖြစ်ပါတယ်။ ဒါဆိုရင် interface ရဲ့အသုံးဝင်ပုံနဲ့ အသုံးပြုပုံကို နားလည်သွားမယ်ထင်ပါတယ်၊ သူနဲ့ အခြေခံ သဘောတရားခြင်းဆင်တဲ့ Abstract class ကိုဆက်ပြီးတော့ လေ့လာကြပါမယ်။

Abstract Classes

Abstract class ရဲ့အခြေခံသဘောတရားက Interface နဲ့ အတူတူဖြစ်ပေမယ့် ကွာခြားတဲ့အချက်တွေ ရှိနေပါသေးတယ်။ အဓိက ကွာခြားတဲ့အချက်က Interface မှာတုန်းက implementation မရှိတဲ့ abstract methods ပဲ ထည့်လို့ရပေမယ့် abstract class မှာတော့ implementation မရှိတဲ့ abstract method ရော implementation လုပ်ထားလို့ရတဲ့ concrete method ရောထည့်သုံးလို့ရနိုင်ပါတယ်။ နောက်ထပ်ကွာခြားချက်တဲ့ အချက်တွေကိုတော့ နောက်တစ်ပိုင်းမှာ ထပ်ရှင်းပြပေးပါမယ်။

အရင်ဦးဆုံး ကျနော်တို့ Interface မှာရေးခဲ့တဲ့ Myanmar နဲ့ Japan ဆိုတဲ့ concrete class တွေမှာ တူညီတဲ့ method တစ်ခုရှိနေပါတယ်။ ဟုတ်ပါတယ်၊ isAsianCountry ဆိုတဲ့ method ဖြစ်ပါတယ်၊ နှစ်ခုလုံးက Asian country ဖြစ်တဲ့အတွက် return “true” ပြန်ထားပါတယ်။ ဒီနေရာမှာ အဲ့ဒီ method က concrete class နှစ်ခုလုံးမှာထပ်နေတဲ့အတွက် Interface မှာတစ်ခါတည်း ရေးထားလို့ရရင် code duplication issue ကိုရှင်းနိုင်သွားမှာဖြစ်ပါတယ်။ ဒါပေမယ့် Interface ထဲမှာ abstract method ပဲရှိခွင့်ရတဲ့အတွက် Interface အစား Abstract Class ကိုပြောင်းလဲပြီး ဒီလိုမျိုးအသုံးပြုနိုင်ပါတယ်။

```php
<?php
//abstract class declaration
abstract class Country {
  //abstract method (have to implement from concrete classes)
  public abstract function currentPopulation();
  //concrete method (already implemented)
  public function isAsianCountry(){
    echo "true";
  }
}
?>
```

Abstract class ကြေညာတဲ့နေရာမှာ ***abstract class*** ဆိုတဲ့ keywords လေးတွေကို သုံးပြီးတော့ ကြေညာပါမယ်။ ဒါဆိုရင် နောက်တစ်ဆင့်အနေနဲ့ abstract class ကို ဘယ်လိုပြန်ခေါ်သုံးနိုင်မလဲ ဆိုတာကိုဆက်ကြည့်ကြရအောင်။

```php
<?php
//abstract class declaration
abstract class Country {
  //abstract method (have to implement from concrete classes)
  public abstract function currentPopulation();
  //concrete method (already implemented)
  public function isAsianCountry(){
    echo "true";
  }
}

//using extends keyword to extend the abstract class
class Myanmar extends Country {
  public function currentPopulation() {
    echo "53.71 million";
  }
}

class Japan extends Country {
  public function currentPopulation() {
    echo "126.5 million";
  }
}

$myanmar = new Myanmar();
$myanmar->currentPopulation();// 53.71 million
$myanmar->isAsianCountry();// true

$japan = new Japan();
$japan->currentPopulation();//126.5 million
$japan->isAsianCountry();//true
?>
```

Abstract class ကိုပြန်ခေါ်သုံးတဲ့ နေရာမှာတော့ Interface ကိုခေါ်သုံးသလိုမျိုး ***implements*** ဆိုတဲ့ keyword မဟုတ်တော့ဘဲ ***extends*** ဆိုတဲ့ keyword နဲ့ ခေါ်ရမှာဖြစ်ပါတယ်။ နောက်တစ်ခု ထူးခြားသွားတာက isAsianCountry ဆိုတဲ့ method မှာ abstract class ထဲကိုရောက်သွားပြီးတော့ Myanamr & Japan ဆိုတဲ့ concrete class တွေထဲမှာ code weight လျော့ချနိုင်သွားမှာဖြစ်ပါတယ်။ result ကတော့ တူတူပဲရနေဦးမှာပဲဖြစ်ပါတယ်။ လာမယ့်အပိုင်းမှာ Interface နဲ့ Abstract class ရဲ့ အဓိက ကွာခြားချက်တွေကိုရေးသွားပါမယ်၊ Interface နဲ့ abstract class ကိုပါ နောက်တစ်ကြိမ် real world code example တွေနဲ့ မတူညီတဲ့ ရှုထောင့်ကနေ ထပ်ပြီးရှင်းလင်းသွားပါမယ်။

Interface VS Abstract Class

အပေါ်ကအပိုင်းမှာတုန်းကတော့ Interface & Abstract class ကိုအခြေခံရှုထောင့်ကနေ ရေးသားထားပါတယ်။ အခုတစ်ခေါက်တော့ သူတို့ဘာကြောင့်အသုံးဝင်လဲဆိုတဲ့ ရှုထောင့်အပေါ် ပိုဦးစားပေးပြီး ရှင်းပြပေးသွားပါမယ်။ Interface တစ်ခု implement လုပ်တော့မယ်ဆို၊ interface ဆိုတဲ့ keyword သုံးတယ်၊ Interface ထဲမှာရှိတဲ့ method တွေက abstract method တွေဖြစ်ရမယ်၊ method တွေရဲ့ declaration ပဲရှိမယ်၊ implementation မရှိရဘူး။ နောက်တစ်ချက်က method တွေရဲ့ visibility က public ဖြစ်မယ်။ဘာကြောင့် interface ကအသုံးဝင်ရတာလဲကို အောက်ကနမူနာ code လေးတွေ ကြည့်ရအောင်။

```php
<?php

class useGmail
{
  public function send()
  {
    echo "sending using gmail";
  }
}

class useHotmail
{
  public function send()
  {
    echo "sending using hotmail";
  }
}

class Client
{
    protected $service;

    public function __construct(useGmail $service)
    {
        $this->service = $service;
    }

    public function submit()
    {
        $this->service->send();
    }
}
```

usegmail နဲ့ usehotmail ဆိုပြီး ကျနော့်မှာ service class နှစ်ခုရှိပါတယ်၊ Client class တစ်ခု ရှိပါမယ်။ Client class code ထဲမှာ construct ဆိုတဲ့ function က ကျနော်တို့အတွက် စိမ်းနေသေး ပါတယ်၊ နောက်လာမယ့် အပိုင်းတွေမှာရှင်းပြပေးသွားပါမယ်။ လက်ရှိမှာတော့ Client Class ကို instantiate လုပ်မယ်ဆိုရင် construct ထဲမှာရှိတဲ့ parameter အတွက်ပါ တစ်ခါတည်း ထည့်ပေးဖို့ လိုတယ်လို့ လောလောဆယ် temporary အနေနဲ့မှတ်ထားလိုက်ပါမယ်။ ဒါဆိုရင် client code ကို ခေါ်သုံးမယ်ဆို ဒီလိုလေး သုံးရပါမယ်။

```
1 $client = new Client(new useGmail);
2 $client->submit();
```

client ထဲက submit function ကို တစ်ခါတည်းလှမ်းခေါ်လိုက်တယ်။ ဒါဆိုရင် sending using gmail ဆိုတဲ့ output ကျနော်တို့ရပါတယ်၊ ဟုတ်ပြီ၊ ဒါဆို Gmail မဟုတ်ဘဲ Hotmail ကိုသုံးပြီး ပို့ချင်တယ်ဆိုပါစို့၊ ဒါဆိုရင် client class က constructor ထဲမှာသွားပြောင်းပေးဖို့လိုပါတယ်။ လက်ရှိမှာ class ကတစ်ခုတည်း ရှိပေမယ့် real world project တွေမှာ class တွေအများကြီး ထပ်ရှိလာနိုင်ပါတယ်။ ဒါကြောင့် တစ်ခုခုပြောင်းချင်တိုင်းမှာ constructor ကိုတိုက်ရိုက် သွားထိပြီးပြင်နေရပါမယ်။ ဒီလို swapping issue ကို interface သုံးပြီး ဖြေရှင်းကြည့်ရအောင်။

```
1  <?php
2
3  interface mailService
4  {
5      public function send();
6  }
7
8  class useGmail implements mailService
9  {
10   public function send()
11   {
12     echo "sending using gmail";
13   }
14 }
15
16 class useHotmail implements mailService
17 {
18   public function send()
19   {
20     echo "sending using hotmail";
21   }
22 }
```

mailService ဆိုတဲ့ interface တစ်ခုဆောက်ပြီး usegmail နဲ့ usehotmail ဆိုတဲ့ concrete class တွေကနေ implement လုပ်ထားလိုက်မယ်။ ပြီးတော့ အရှေ့က scenario မှာကျနော်တို့ကို အလုပ်ရှုပ်စေနိုင်မယ့် client class ထဲမှာ usegmail usehotmail ဆိုတဲ့ concrete class တွေကို manual မထည့်တော့ဘဲ ဒီလိုမျိုး interface ကို constructor ထဲမှာတိုက်ရိုက်ထည့်ထားလိုက်ပါမယ်။

```php
class Client
{
    protected $service;

    public function __construct(mailService $service)
    {
        $this->service = $service;
    }

    public function submit()
    {
        $this->service->send();
    }
}
// just have to change this if we want to modify all the time
$client = new Client(new useHotmail);
$client->submit();
```

ဒီလိုမျိုး interface ကိုသုံးပြီး modify လုပ်ပြီးတဲ့နောက်မှာ ကျနော်တို့အနေနဲ့ mail service ကိုပြောင်းချင်တဲ့အချိန်တိုင်းမှာ client class ကို instantiate လုပ်တဲ့နေရာမှာ ကိုယ်သုံးချင်တဲ့ concrete class ကို ထည့်ပေးလိုက်ရုံပါပဲ၊ result ကတူတူပဲ၊ ဒါပေမယ့် constructor ကို manually သွားထိနေစရာမလိုတော့ဘူး။

Abstract class က Interface နဲ့ မတူတာက abstract မှာ concrete method တွေပါရေးလို့ရတယ်။ class ထဲမှာ define လုပ်ထားတဲ့ abstract method တွေကို child class တွေက မဖြစ်မနေ implement လုပ်ပေးရမယ်။ Abstract class ရဲ့ main purpose က child class တွေအတွက် template လေးလိုပုံစံပဲ၊ inherit လုပ်မယ်၊ abstract method တွေ implement လုပ်မယ်။ abstract class ကဘယ်လိုအသုံးဝင်လဲဆိုတာ အောက်က code example လေးတွေကြည့်ရအောင်။

```php
<?php

class ChickenCurry
{
    public function addChicken()
    {
        var_dump('Add raw chicken');
        return $this;
    }
    protected  function  addSalt()
    {
        var_dump('Add some salt');
        return $this;
    }

    protected  function addPepper()
    {
        var_dump('Add some pepper');
        return $this;
    }

    public function cook()
    {
        return $this
            ->addChicken()
            ->addSalt()
            ->addPepper();
    }
}

$chickenCurry = new ChickenCurry();
$chickenCurry->cook();
```

chickenCurry ဆိုတဲ့ ကြက်သားဟင်းချက်တဲ့ class တစ်ခုရှိတယ်။ Normal class တွေပဲနော်၊ အထဲမှာ funciton 4 ခုရှိမယ်။ chickenCurry လိုမျိုး အလားတူ နောက်ထပ် class တစ်ခုထပ်မံ တည်ဆောက်ပါမယ်။

```php
class PorkCurry
{
    public function addPork()
    {
        var_dump('Add raw pork');
        return $this;
    }
    protected  function  addSalt()
    {
        var_dump('Add some salt');
        return $this;
    }

    protected  function addPepper()
    {
        var_dump('Add some pepper');
        return $this;
    }

    public function cook()
    {
        return $this
            ->addPork()
            ->addSalt()
            ->addPepper();
    }
}

$PorkCurry = new PorkCurry();
$PorkCurry->cook();
```

နောက်တစ်ခု ဝက်သားဟင်းချက်တဲ့ porkCurry ဆိုတဲ့ class တစ်ခုရှိတယ်။ class နှစ်ခုကို ယှဉ်ကြည့်လိုက်မယ်ဆို addSalt ရယ် addPepper ရယ်ဆိုတဲ့ တူတဲ့ methods 2 ခုကိုတွေ့ရမယ်၊ အဓိကပါဝင်ပစည်းဖြစ်တဲ့ addPork နဲ့ addChicken ပဲကွာသွားမယ်။ ဒီလိုနေရာမှာ abstract class ကိုသုံးပြီးတော့ addSalt & addPepper ကို implement လုပ်ထားမယ်၊ main ingredients ဖြစ်တဲ့ ကြက်နဲ့ဝက်ကို abstract method လုပ်ပြီးတော့ concrete class တွေကို implement လုပ်ခိုင်းမယ်။ ဒါဆိုရင် အောက်က လိုဖြစ်သွားမယ်။

```php
abstract class Recipe
{
    protected  function  addSalt()
    {
        var_dump('Add some salt');
        return $this;
    }

    protected  function addPepper()
    {
        var_dump('Add some pepper');
        return $this;
    }

    public function cook()
    {
        return $this
            ->addMainIngredient ()
            ->addSalt()
            ->addPepper();
    }

    protected abstract function addMainIngredient();
}
```

Recipe ဆိုတဲ့ abstract class လေးတစ်ခုဆောက်လိုက်တယ်။ အဲ့ထဲမှာ ထပ်နေပြီး ပြန်သုံးလို့ရတဲ့ method ၃ခုကိုတစ်ခါတည်း implement လုပ်ထားလိုက်တယ်၊ ပြန်သုံးလို့မရတဲ့ method ကိုတော့ abstract method အဖြစ်ရေးထားတယ်။ ဒါဆိုရင် chickenCurry နဲ့ porkCurry class ကိုဆောက်တော့မယ်ဆို Recipe abstract class ကို extends လုပ်ပြီး ဆောက်ရုံပဲ။ extends လုပ်လိုက်ပြီဆို abstract class မှာရှိတဲ့ concrete method တွေကို access လုပ်ခွင့်ရှိမယ်။ abstract method ကိုလည်း ကိုယ်လိုချင်တဲ့ code တွေ implement လုပ်ပြီးရေးလိုက်ရုံပဲ။ အောက်ကလိုပေါ့။

```php
class ChickenCurry extends Recipe
{
    public function addMainIngredient()
    {
      var_dump('Add raw chicken');
      return $this;
    }
}
$chickenCurry = new ChickenCurry();
$chickenCurry->cook();
```

Porkcurry ကိုလည်း အပေါ်က chickenCurry လိုပဲ ထပ်မံအသုံးပြုနိုင်မှာ ဖြစ်ပါတယ်။

```php
class PorkCurry extends Recipe
{
    public function addMainIngredient()
    {
      var_dump('Add raw pork');
      return $this;
    }
}
$porkCurry = new PorkCurry();
$porkCurry->cook();
```

အရင်တုန်းက ရေးခဲ့တဲ့ class အကွဲနှစ်ခုအတိုင်းလိုမျိုး result ကအတူတူပဲရတယ်။ ဒါဆိုရင် ဘယ်လိုအခြေအနေမျိုးမှာ abstract ကိုသုံးရမယ်ဆိုတာနားလည်မယ်လို့ထင်ပါတယ်။

Interface နဲ့ abstract class ကိုတော်တော်လေးလည်းရှင်းပြီးသွားပြီဆိုတော့ အဓိကအားဖြင့် သူတို့နှစ်ခုဘာကွာသွားလဲဆိုတာကို နိဂုံးချုပ်ရရင်

- Interface မှာ child class တွေက interface တစ်ခုထက်မက implements လုပ်လို့ရတယ်။ abstract မှာတော့ child က abstract class တစ်ခုပဲ extends လုပ်လို့ရမယ်။
- Interface အတွက်ဆို implements keyword သုံးတယ်။ abstract class အတွက်ဆို extends keyword သုံးတယ်။
- Interface မှာ data member တွေနဲ့ constructor တွေပါခွင့်မရှိဘူး။ abstract class မှာတော့ data member ရော constructor ရောပါလို့ရတယ်။
- Interface မှာ implementation မရှိတဲ့ abstract method တွေပဲပါခွင့်ရှိမယ်။ abstract class မှာတော့ abstract method ရော၊ concrete method ရောပါခွင့်ရှိမယ်။
- Interface မှာ default အနေနဲ့ public ကလွဲလို့ ကျန်တဲ့ access modifiers တွေပါခွင့်မရှိဘူး။ abstract မှာတော့ methods တွေ properties တွေမှာ access modifiers တွေထည့်လို့ရတယ်။
- Interface မှာ ပါတဲ့ member တွေက static type ဖြစ်ဖွင့်မရှိဘူး။ abstract မှာတော့ concrete ဖြစ်ထားတဲ့ member တွေက static type ဖြစ်လို့ရတယ်။

Traits

Trait အကြောင်းကိုနားလည်ဖို့အတွက် Inheritance ရဲ့အစိတ်အပိုင်းလေးတစ်ခုကို ပြန်ွေးဖို့ လိုပါတယ်။ Inheritance မှာ child class တွေက parent class ဆီကနေ inheritance (အမွေလက်ခံ) ယူလို့ရပါတယ်။ ဒါပေမယ့် ခြွင်းချက်တစ်ခုရှိတာက child class တွေက တစ်ခုထက်ပိုတဲ့ parent class တွေကို inheritance လုပ်လို့မရပါဘူး၊ inheritance လုပ်လို့ရမယ့် parent class ကို တစ်ခုပဲခွင့်ပြုထားပါတယ်။ ဒါကြောင့် ဒီနေရာမှာ trait ဆိုတာကို အသုံးပြုလို့ ရနိုင်ပါတယ်။

တစ်ခုထက်ပိုတဲ့ class တွေကို inherit လုပ်ချင်ပြီဆို ပုံမှန် inheritance မလုပ်ဘဲ trait ကို အသုံးပြုပြီးတော့ class တွေအများကြီးကို တစ်ပြိုင်တည်း inherit လုပ်နိုင်မှာဖြစ်ပါတယ်။ Trait ထဲမှာ ပုံမှန် class တစ်ခုလိုပဲ methods & properties တွေကို လိုသလိုကြေညာလို့ရနိုင်တဲ့အပြင် visibility ကိုလည်း သတ်မှတ်နိုင်မှာဖြစ်ပါတယ်။ ကျနော်တို့ trait တစ်ခု စပြီးတော့ ဆောက်ကြည့်ရအောင်။

```php
<?php
//creating trait using `trait` keyword
trait trait1 {
  public function call() {
    echo "this is from trait1";
  }
}

class Client {
  //calling the trait name with `use` keyword to inherit
  use trait1;
}

$obj = new Client();
$obj->call(); //output - this is from trait1
?>
```

Trait ဆောက်တော့မယ်ဆို ***trait*** ဆိုတဲ့ keyword လေးကိုသုံးပြီးတော့ ဆောက်ပါတယ်။ Client class မှာကတော့ inherit လုပ်ချင်တဲ့ trait ကို ***use*** ဆိုတဲ့ keyword လေးသုံးပြီးတော့ ပြန်ခေါ်သုံးရပါမယ်။ အပေါ်က example ကိုကြည့်မယ်ဆို ရိုးရိုးရှင်းရှင်းပဲ trait တစ်ခု ဆောက်ထားပြီးတော့ client class မှာ အဲ့ဒီ trait ကို ယူသုံးထားပါတယ်။

နောက်ဆုံးအနေနဲ့ကတော့ client code က သုံးချင်တာကို Client class ကို instantiate လုပ်ပြီးသုံးရုံပါပဲ၊ ဒါတွေကတော့ သိပြီးသားဖြစ်တဲ့ အတွက် ကျနော်အထူးတစ်လည် မရှင်းပြတော့ပါဘူး။ နောက်တစ်ဆင့် အနေနဲ့ တစ်ခုထက်ပိုတဲ့ trait တွေဆောက်ကြည့်ပြီး ခေါ်သုံးကြည့်ပါမယ်။

```php
<?php
trait trait1 {
  public function call1() {
    echo "this is from trait1";
  }
}

trait trait2 {
  public function call2() {
    echo "this is from trait2";
  }
}

class Client {
  use trait1;
  use trait2;
}

$obj = new Client();
$obj->call1(); //output - this is from trait1
$obj->call2(); //output - this is from trait2
?>
```

အရမ်းကြီးပြောင်းလဲသွားတာမရှိတာကို တွေ့ရပါလိမ့်မယ်။ ထပ်ထည့်ချင်တဲ့ trait ကိုကြေညာတယ်၊ ပြီးတော့ class ထဲမှာ use နဲ့ပြန်ခေါ်သုံးထားမယ်ဆိုရင် client code ကနေ trait ထဲမှာရှိတဲ့ methods တွေကို access လုပ်ခွင့်ရသွားမှာဖြစ်ပါတယ်။

Trait ရဲ့သဘောတရားက လွယ်ကူရိုးရှင်းတဲ့အပြင် အသုံးဝင်တဲ့အတွက် အရေးကြီးပါတယ်။ PHP Framework တစ်ခုဖြစ်တဲ့ Laravel လိုမျိုးမှာဆိုရင်လည်း trait တွေ အမြောက်အမြား သုံးထားတာကို တွေ့ရမှာဖြစ်ပါတယ်။

Magic Methods In PHP

ဒီတစ်ပိုင်းမှာတော့ PHP မှာရှိတဲ့ magic methods တွေအကြောင်းရေးသွားမှာဖြစ်ပါတယ်။ OOP အခြေခံအတွက်ရေးနေတာဖြစ်ပေမယ့် PHP နဲ့ရေးနေတာဆိုတော့ magic methods လေးတွေအကြောင်းပါ တစ်ပါတည်းသိရအောင် ထည့်ပေးထားခြင်းဖြစ်ပါတယ်။ တစ်ခြား programming အသုံးပြုနေသူတွေအတွက် ဒီတစ်ပိုင်းကျော်ဖတ်သွားလို့ရပါတယ်။

Magic Methods တွေဆိုတာ PHP က build-in support လုပ်ပေးထားတဲ့ special functions လေးတွေဖြစ်ပါတယ်၊ အခေါ်အဝေါ်အရဆိုရင်တော့ magic methods တွေဖြစ်ပါတယ်။ magic methods တွေတိုင်းရဲ့အရှေ့မှာ __(double underscore) ပါပြီးတော့ အဲ့ဒီ magic methods တွေဟာ သတ်မှတ်ထားတဲ့ PHP events တိုင်းမှာ လာပြီး execute လုပ်ခြင်းခံရပါတယ်။ အပေါ်က code examples တွေထဲမှာ ကျနော်တို့တစ်ခုကို သိပြီးခဲ့ပါပြီ။ (__construct) ဆိုတဲ့ PHP magic method လေးပါ။ အရှေ့မှာ double underscore ခံထားတယ်၊ သူ့ကို trigger ဖြစ်မယ့်အချိန်က class ကို instantiate လုပ်မယ့်အချိန်ဖြစ်ပါတယ်။ ဥပမာလေးတစ်ခုတန်းကြည့်ရအောင်။

__construct

```php
<?php

class Person{

  //constructor magic method
  public function __construct() {
    $this->name = "Hlaing";
  }

}

//construct magic method is triggered on instantiation of person class
$person = new Person;
echo $person->name; //output - Hlaing
```

Person ဆိုတဲ့class လေးထဲမှာ magic method တစ်ခုဖြစ်တဲ့ construct ကိုတည်ဆောက်ထားပြီးတော့ property တစ်ခု assign လုပ်ပေးထားပါတယ်။ client code မှာ person class ကို instantiate လုပ်လိုက်တာနဲ့ construct magic method ကို trigger ဖြစ်ထားတဲ့အတွက် construct method ထဲမှာ ကြေညာထားတဲ့ name property ကိုလည်း အလိုအလျောက် access လုပ်နိုင်မှာပဲဖြစ်ပါတယ်။

ဒါကတော့ magic method တစ်ခုကို အမြည်းကျွေးထားရုံပဲရှိပါသေးတယ်။ တစ်ကယ်တော့ PHP မှာတစ်ခြားသော magic methods တွေအများကြီးရှိပါသေးတယ်။ ကျနော် ဒီ series လေးမှာတော့ အကုန်လုံးကို မဖော်ပြနိုင်တဲ့အတွက် အသုံးများတဲ့ magic methods တစ်ချို့ပဲရေးပြသွားပါမယ်။ မှတ်ထားရမှာက PHP magic methods တွေက ဘယ်လိုသတ်မှတ်ထားတယ်၊ ဘယ်အချိန်တွေမှာ trigger ဖြစ်တယ်ဆိုတဲ့ သဘောတရားပဲဖြစ်ပါတယ်။ trigger လုပ်မယ့်အချိန်တွေက documentation ဖတ်လိုက်ရင် နားလည်နိုင်တဲ့အတွက် သဘောတရားကိုသာ မှတ်ထားနိုင်ရင် အဆင်ပြေပါတယ်။

__destruct

Destruct ကတော့ __construct နဲ့ ပြောင်းပြန်သဘောတရားလို့မှတ်လို့ရပါတယ်။ construct က class ကို instantiate လုပ်တဲ့အချိန် trigger ဖြစ်ပေမယ့် destruct ကတော့ class ကို destroy/close ဖြစ်တဲ့အချိန်မှာ trigger ဖြစ်ပါတယ်။ ဘယ်လိုအချိန်တွေမှာ အသုံးဝင်လဲဆိုတော့ ကိုယ့် class ထဲမှာ class ကိုမပိတ်သွားခင်မှာ shutdown လုပ်စရာရှိတဲ့ task တွေကို ရေးတဲ့နေရာမှာ အသုံးဝင်ပါတယ်။ ဥပမာ

```php
<?php

class Sample{

  public function __construct() {
    $this->file = fopen('file.csv', 'w');
  }

  //file close operation on class destroys.
  public function __destruct() {
    fclose($this->file);
  }
}
```

Sample class ထဲက construct ထဲမှာ fopen operation လုပ်ထားခဲ့ပြီးတော့ class ကို terminate ဖြစ်မယ့်အချိန်မှာ trigger ဖြစ်မယ့် destruct magic method ထဲမှာ close လုပ်တဲ့ operation ကို ရေးထားခဲ့လို့ရပါတယ်။ ထိုနည်းလည်းကောင်းပဲ၊ တစ်ခြားသော ပုံစံတူ operations တွေကိုလည်း destruct ထဲမှာဝင်ရေးထားလို့ရပါတယ်။

__set

Set magic method ကတော့ define လုပ်ထားခြင်းမရှိတဲ့ property တွေ၊ access လုပ်လို့မရနိုင်တဲ့ (protected or private)ဖြစ်နေတဲ့ property တွေကို value assign လုပ်ပေးချင်တဲ့ အချိန်မှာ အသုံးပြုနိုင်ပါတယ်။ code sample ကိုကြည့်ရင် ပိုမြင်သွားပါမယ်။

```php
<?php

class Person {
    public function __set($name, $value)
    {
        $this->name = $value;
    }
}

$person = new Person();

//  __set() is triggered.
echo $person->name = 'Hlaing';//output - Hlaing
?>
```

ပုံမှန်ဆို person class ထဲမှာ name ဆိုတဲ့ property ကမရှိပေမယ့်လည်း လိုသလို value ကို assign လုပ်ပေးနိုင်မှာဖြစ်ပါတယ်။ ဒီနေရာမှာ name property က protected တို့ private တို့ဖြစ်နေမယ်ဆိုရင်လည်း ပုံမှန်ဆို inaccessible ဖြစ်တဲ့ property ကိုလှမ်းခေါ်မယ်ဆိုရင် error တတ်နိုင်ပေမယ့် set magic method ကိုသုံးထားမယ်ဆိုရင် အလုပ်လုပ်နိုင်မှာဖြစ်ပါတယ်။

__get

```php
<?php

class Person {

    public function __set($name, $value)
    {
        $this->name = $value;
    }

    public function __get($name)
    {
        return $this->name;
    }
}

$person = new Person();

//  __set() is triggered.
$person->name = 'Hlaing';

//  __get() is triggered.
echo $person->name;//output - Hlaing
?>
```

Set မှာတုန်းကတော့ inaccessible or undefined property တွေကို value assign လုပ်တဲ့နေရာမှာ သုံးခဲ့ပါတယ်။ get မှာတော့ ပြောင်းပြန်သဘောတရားဖြစ်ပြီးတော့ inaccessible or undefined ဖြစ်တဲ့ property value တွေကို access လုပ်တဲ့နေရာမှာသုံးပါတယ်။ ခုနက set လုပ်ထားတဲ့ example လေးကိုပဲ get magic method ထည့်ပြီး example ကြည့်ကြည့်ရအောင်။

Undefined property ဖြစ်တဲ့ name ကို access လုပ်လိုက်တာနဲ့ get magic method ကို trigger ဖြစ်သွားပြီး return လုပ်ပေးသွားမှာပဲဖြစ်ပါတယ်။

__toString

Object တစ်ခုကို string လို display လုပ်ချင်တဲ့အချိန်မှာ toString ဆိုတဲ့ magic method ကို အသုံးပြုပါတယ်။ ပုံမှန်ဆို object ကို ဒီလို string အတိုင်းထုတ်မယ်ဆို error တတ်မှာဖြစ်ပါတယ်။

```php
<?php
class Person {
}

$person = new Person();
echo $person; // will get an error.
?>
```

Person class ကို instantiate လုပ်ထားတဲ့ person object ကို string လိုမျိုး echo နဲ့ display လုပ်မယ်ဆို error တတ်မှာဖြစ်ပါတယ်။ ဒီအချိန်မှာ toString magic method ထည့်သုံးပြီး object ကို string လိုမျိုးအသုံးပြုနိုင်မှာဖြစ်ပါတယ်။ for example

```php
<?php
class Person {
    public function __toString()
    {
        return "Actions happen here";
    }
}

$person = new Person();
echo $person; //output - Actions happen here
?>
```

toString magic method ကိုကြည့်ပြီး အထဲမှာကိုယ်သလို operations တွေကို handle လုပ်ထားပြီး return ပြန်ထားနိုင်ပါတယ်။ ဒါဆိုရင် Person class ကို instantiate လုပ်ထားတဲ့ ဘယ် object မဆို string output ထုတ်မယ်ဆို toString ထဲက output ကို ရရှိမှာဖြစ်ပါတယ်။

ကျနော် magic methods အပိုင်းကိုတော့ ဒီမှာပဲ အဆုံးသတ်ပါရစေ၊ အခုကျနော်ပြောပြခဲ့တာတွေက အသုံးများတဲ့ magic methods တွေဖြစ်ပါတယ်။ တစ်ခြားသော real world မှာအသုံးဝင်တဲ့ magic methods တွေလည်းရှိပါသေးတယ်။ PHP Official Documentation မှာ ထပ်မံဝင်ရောက်ဖတ်ရှုနိုင်ပါတယ်။

https://www.php.net/manual/en/language.oop5.magic.php

အဓိက ရည်ရွယ်ချက်ကတော့ ဘယ်အချိန်မှာ ဘယ်လို magic methods တွေက အသုံးဝင်နိုင်တယ် ဆိုတာကို နားလည်ပြီးတော့ အသုံးချနိုင်ဖို့ပဲဖြစ်ပါတယ်။

Polymorphism

Polymorphism ဆိုတာစကားလုံးအခေါ်အဝေါ်အရသာရှုပ်သယောင်ရှိပေမယ့် တစ်ကယ်တမ်းတော့ အဲ့လောက်မရှုပ်ပါဘူး။ သူ့ရဲ့အဓိက ရည်ရွယ်ချက်သည် client code က interaction ပေါ်မူတည်ပြီး adaptable (အဆင်ပြေအောင်လုပ်ဆောင်နိုင်စွမ်း) ဖြစ်အောင် လုပ်ဆောင်ပေးပါတယ်။ ကျနော်တို့ အပေါ်မှာသင်ခဲ့တဲ့ inheritance အပေါ်မူတည်ပြီး polymorphism example တစ်ခု တစ်ခါတည်း ကြည့်ကြရအောင်။

```php
<?php

interface Country
{
    public function talk();
}

class Myanmar implements Country
{
    public function talk()
    {
        return "Mingalarpr";
    }
}

class Japan implements Country
{
    public function talk()
    {
        return "konichiwa";
    }
}
```

ကျနော်ပထမဦးဆုံး Country ဆိုတဲ့ interface တစ်ခုဆောက်ထားပြီးတော့ အထဲမှာ talk ဆိုတဲ့ method ကြေညာထားပါတယ်။ Myanmar နဲ့ Japan ဆိုတဲ့ concrete class နှစ်ခုက interface ကို ပဲ share သုံးပြီး implements လုပ်ထားကြပါတယ်။ ထိုနည်းလည်းကောင်းပဲ talk ဆိုတဲ့ method ကိုလည်း implements လုပ်ထားပါတယ်။ Myanmar ရော Japan ရောက နိုင်ငံတွေဖြစ်တဲ့

အားလျော်စွာ အမျိုးအစားတူတဲ့အတွက် country interface ကို share သုံးထားပေမယ့် အထဲမှာပါတဲ့ talk ဆိုတဲ့ function ကတော့ နိုင်ငံကိုလိုက်ပြီး ကွဲပြားနိုင်ပါတယ်။

```
$myanmar = new Myanmar();
$japan = new Japan();

echo $myanmar->talk(); //output - Mingalarpr
echo $japan->talk(); //output - konichiwa
```

Myanmar object ကနေ output ထွက်လာမယ့် talk function နှင့် Japan object က ထွက်လာမယ့် talk function ဟာကွဲပြားသွားမှာပဲဖြစ်ပါတယ်။ ဒီလိုမျိုး scenario ကိုpolymorphism လို့ခေါ်ပါတယ်။
Interface တစ်ခုကို အမျိုးအစားတူလို့ share သုံးပေမယ့် ပါဝင်တဲ့ functions တွေက မတူကွဲပြားတာကို ဆိုလိုခြင်းဖြစ်ပါတယ်။ အခုတွေ့ခဲ့ရတာကို Dynamic Polymorphism လို့ခေါ် ပြီးတော့ နောက်ထပ် static polymorphism ဆိုတာလည်း ရှိပါသေးတယ်။

Static polymorphism ကိုတော့ inheritance အကူအညီမလိုဘဲ function overloading လုပ်ခြင်းဖြင့်တည်ဆောက်နိုင်ပါတယ်။

```
<?php
   class Talk {
      //function overloading using magic method __call
      function __call($name,$arg){
        switch(count($arg)){
          case 1 : return $arg[0];
          case 2 : return $arg[0] .'&'. $arg[1];
        }
      }
   }
   $talk = new Talk();
   echo $talk->lang("Myanmar"); //output - Myanmar
   echo $talk->lang("Myanmar","Japan"); //output - Myanmar & Japan
?>
```

Talk ဆိုတဲ့ class ထဲမှာ magic method __call ကိုအသုံးပြုပြီး arguments မတူတဲ့ function call တွေအတွက် မတူညီတဲ့ results တွေ return ပြန်ပေးထားပါတယ်။ lang ဆိုတဲ့ function ကိုခေါ်ရင် argument တစ်ခုပါတယ်ဆို return result ကတစ်မျိုး၊ argument နှစ်ခုဆို return result ကတစ်မျိုး ရလာမှာဖြစ်ပါတယ်။ ဒီလိုမျိုး function arguments ပေါ်မူတည်ပြီး ပြောင်းလဲသွားနိုင်တာကို static polymorphism လို့ခေါ်ပါတယ်။

Magic method __call reference

https://www.php.net/manual/en/language.oop5.overloading.php#object.call

Dependency injection

Dependency Injection ကို wiki ကတော့ ဒီလိုပြောထားပါတယ်။

dependency injection is a technique in which an object receives other objects that it depends on.

Dependency injection (DI) ဆိုတာ object တစ်ခုက သူလိုအပ်တဲ့ တစ်ခြား object dependency တစ်ခုကို လက်ခံထားတာကို DI လို့ခေါ်ပါတယ်တဲ့ (I know it's a crazy translation :3)။ ပြေလည်အောင်ဘာသာပြန်ရရင် object တစ်ခုက လိုအပ်နေတဲ့ dependency ကို တစ်ခြား object က supply လုပ်ပေးမယ်၊ ဒါကို dependency injection လို့ခေါ်ပါတယ်။

ရှင်းအောင်ထပ်ပြောရရင် ပင်မ original class က သူလိုအပ်တဲ့ dependency object တွေကို ကိုယ်တိုင် create မလုပ်ဘဲ တစ်ခြား resources တစ်ခုဆီကနေ တိုက်ရိုက်ယူသုံးတာမျိုးကို ဆိုလိုခြင်းဖြစ်ပါတယ်။ မြင်သာသွားအောင် code examples တွေနဲ့ ချည်းကပ်ကြည့်ရအောင်။ ကျနော်တို့ dinner စားတော့မယ်ဆိုပါစို့၊ Dinner ဆိုတဲ့ class ရှိမယ်၊ စားတာကဟုတ်ပြီ၊ ဘာစားမှာလဲပေါ့။ ဒါဆိုရင် food ဆိုတဲ့ အစားအသောက် dependency တစ်ခုထပ်လိုလာပါပြီ။ code နဲ့ချရေးကြည့်ရအောင်။

```php
<?php

class Food
{
    protected $food;

    public function __construct($food)
    {
        $this->food = $food;
    }

    public function get()
    {
        return $this->food;
    }
}

$food = new Food("Mala Shan Guo");
echo $food->get(); //output - Mala Shan Guo
```

Food ဆိုတဲ့ class စဆောက်လိုက်ပါပြီ။ Food ကတော့ Dinner ဆိုတဲ့ class ကလိုအပ်တဲ့ food ကို supply လုပ်ပေးမယ့် class ပဲဖြစ်ပါတယ်။ အထဲမှာ construct တစ်ခုပါတယ်၊ get ဆိုတဲ့ method ပါမယ်၊ get ကိုခေါ်မယ်ဆို Class ကို instantiate လုပ်တုန်းက ထည့်လိုက်တဲ့ အစားအသောက်ကို return ပြန်လုပ်ပေးမှာဖြစ်ပါတယ်။ ဒါဆို နောက်ထပ် dinner class ကို ဆောက်ရအောင်။

```php
class Dinner
{
    public function eat()
    {
        $food = new Food("Mala Shan Guo");
        return $food->get();
    }
}

$dinner = new Dinner();
echo $dinner->eat(); //output - Mala Shan Guo
```

လောလောဆယ်မှာတော့ ဘာ dependency injection မှမလုပ်ထားသေးပါဘူး၊ ပုံမှန်အတိုင်းပဲ ရေးထားပါသေးတယ်။ Dinner ဆိုတဲ့ class ထဲမှာ eat ဆိုတဲ့ method ရှိမယ်၊ အထဲမှာ Food class ကို instantiate လုပ်ထားပါတယ်။ parameter နေရာမှာလည်း hardcode ထည့်ပေးထားပါတယ်။ ဒီနေရာမှာဘာတွေ မကောင်းဘူးလဲဆိုတော့ Dinner class ထဲမှာတင် Food class ကို new နဲ့ instantiate လုပ်ထားတဲ့အတွက် Tight coupling ဖြစ်နေပါတယ်၊ ဆိုလိုချင်တာက food class ကိုအသေချိတ်ထားသလိုဖြစ်နေတယ်။ နောက်တစ်ခုက parameter နေရာမှာလည်း class ထဲမှာတင် hardcode အသေထည့်ပေးထားရပါတယ်။ ဒီလို issues တွေကို dependency injection ကို အသုံးပြုပြီးတော့ ဖြေရှင်းနိုင်ပါတယ်။ DI သုံးပြီးတော့ Dinner class ကို ပြန်ပြင်ရေးကြည့်ပါမယ်။

```php
class Dinner
{
    protected $dinner;

    //receiving the dependency object food.
    public function __construct(Food $food)
    {
        $this->dinner = $food;
    }

    public function eat()
    {
        return $this->dinner->get();
    }
}

$food = new Food("Mala Shan Guo");
//supplying the food object.
$dinner = new Dinner($food);
echo $dinner->eat(); //output - Mala Shan Guo
```

Dinner class ထဲမှာ food class ကို hardcode လုပ်ထားတာတွေဖယ်ထုတ်လိုက်ပြီး construct တစ်ခုဆောက်လိုက်ပါတယ်၊ parameter ထဲမှာတော့ food object ကိုလက်ခံဖို့ ထည့်ပေးထားပါတယ်။

ဒါဆိုရင် client code ကနေပြီးတော့ လိုအပ်တဲ့ food class ကို instantiate လုပ်ပြီးတော့ dinner class ထဲကို pass လုပ်ပေးလိုက်ရုံပါပဲ။ ဒီအတွက်ကြောင့် Dinner class က food class ကို သူကိုယ်တိုင် create လုပ်စရာမလိုတော့တဲ့အတွက် tight coupling မဖြစ်တော့တဲ့အပြင် hardcode declaring တွေကိုပါ reduce လုပ်နိုင်သွားပြီပဲဖြစ်ပါတယ်။ အစမှာပြောခဲ့တဲ့ definition ကိုပြန်ကောက်ရရင် dinner class ကသူလိုအပ်တဲ့ food dependency class ကို ကိုယ်တိုင် create လုပ်ခြင်းမရှိဘဲ တစ်ခြားသော object က create လုပ်ပေးထားတာကို တိုက်ရိုက်ယူသုံးလိုက်ခြင်းဖြစ်ပါတယ်။ ဒါကို dependency inject လုပ်လိုက်တယ်လို့ဆိုလိုခြင်းဖြစ်ပါတယ်။

အခုလက်ရှိရေးလိုက်တဲ့ example မှာတော့ dependency injection ကို constructor ကနေတစ်ဆင့်လုပ်လိုက်ပေမယ့် constructor တင်မဟုတ်ဘဲ တစ်ခြားသော setter methods တွေ interface တွေမှတစ်ဆင့်လည်း လုပ်ဆောင်နိုင်ပါတယ်။

Conclusion

OOP Basic Series လေးကို ဒီမှာပဲအဆုံးသတ်ထားလိုက်ပါတယ်ခင်ဗျာ။ ကျနော်ကိုယ်တိုင်လည်း လေ့လာဆဲ လူတစ်ယောက်ဖြစ်တဲ့အတွက် လိုအပ်တာလေးတွေတွေ့ရင်လည်း ဝင်ရောက်ဖြည့်စွက်ပေးနိုင်ပါတယ်။ series လေးက အများကြီးမဟုတ်ပေမယ့် OOP ကိုစလေ့လာမယ့်သူတွေ အတွက် အကျိုးရှိမယ်လို့ မျှော်လင့်ပါတယ်။

ကျေးဇူးတင်ပါတယ်။

GitHub Repository

https://github.com/HlaingTinHtun/OOP-Basic-Myanmar