



Introduction	2
Class and Object	6
Visibility & Access Modifiers	9
Inheritance	12
Static Properties & Methods	17
Interfaces	23
Abstract Classes	28
Interface VS Abstract Class	31
Traits	40
Magic Methods In PHP	44
__construct	44
__destruct	45
__set	46
__get	47
__toString	48
Polymorphism	50
Dependency Injection in PHP	53
Conclusion	57

စာရေးသူ၏အမှာစာ

ကျနော့်နာမည် လှိုင်တင်ထွန်းဖြစ်ပါတယ်။ လက်ရှိ ဤစာအုပ်တိုလေးရေးဖြစ်ချိန်တွင် Japan, Osaka မှာ Web Engineer တစ်ဦးအဖြစ်လုပ်ကိုင်လျက်ရှိပြီး A Programmer ဆိုသော Page တွင်လည်း Programming ဘလော့များရေးသားကာ online သင်တန်းများသင်ကြားပေးလျက်ရှိပါတယ်။

OOP ဆိုတာကိုလွယ်လင့်တကူလေ့လာနိုင်ရန် မြန်မာလို Resource ကောင်းလေးတစ်ခုဖြစ်ပေါ်လာစေဖို့အတွက် ရည်ရွယ်ပြီးရေးသားထားခြင်းဖြစ်ပါသည်။ နမူနာ sample code များကို PHP နှင့်ရေးသားထားပါသည်။ လိုအပ်ချက်များရှိပါက Github တွင် issue လာရောက်တင်ပေးနိုင်ပါသည်။

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 ရဲ့နမူနာတစ်ခု၊ ဒီနေရာမှာပြောစရာရှိတာက ပေါင်းရမယ့် number operation နောက်တစ်ကြိမ်ထပ်လုပ်ရမယ်ဆိုလဲ ဒီလိုမျိုး code ထပ်ရေးရဦးမှာပဲ။ ဒီနေရာမှာ procedural programming ဆိုတာက အသုံးဝင်လာတယ်။

Procedural programming ကတော့ sequential ရဲ့ အားနည်းချက်တွေက 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

```
1 function addNum($a,$b)
2 {
3     $plus = $a+$b;
4     echo $plus;
5 }
6
7 //addNum(1,2);
```

ခေါ်သုံးချင်ပြီဆို addNum(1,2) ဆိုပြီးခေါ်လိုက်ရုံပဲ၊ ဒါဆိုလို့ရှိရင် နောက် plus operation တွေရှိတိုင်း မှာ sequence မှာတုန်းကလို လိုင်းငှကြောင်းရေးစရာမလိုတော့ဘဲ တစ်ကြောင်းတည်းနဲ့ပြတ်သွားလိုက်မယ်။ နောက်တစ်ခုတွေ့မှာက function အနောက်မှာပါလာတဲ့ param လေးတွေပဲ၊ နောက်ပိုင်း calculation လုပ်တဲ့အချိန် မတူညီတဲ့ digit တွေအတွက် addNum လို့ခေါ်တဲ့အချိန်မှာတည်းက တစ်ခါတည်း

ထည့်ပေးလိုက်လို့ရအောင်လုပ်ပေးထားတယ်။ ဒါကြောင့်မို့လို့ procedural နဲ့ဆိုလို့ရှိရင် sequence မှာ ကြိုလာရမယ့် code ထပ်မယ့် ပြဿနာ တွေကိုရှောင်လို့ရသွားလိုက်မယ်။

အပေါ်ကပြောခဲ့တဲ့ sequential နဲ့ procedural ဆိုတာ programming paradigm တွေပဲ၊ paradigm ကိုမြန်မာလိုဘာသာပြန်ရရင် ရေးထုံးပုံစံတွေပေါ့။ OOP ဆိုတာကလည်း object သဘောတရားကို အခြေခံထားတဲ့ paradigm တစ်ခုပဲ၊ သူ့မှာပါတဲ့ ရေးထုံးတွေကိုတော့ ကျနော်တို့ တစ်ဖြည်းဖြည်းခြင်း လေ့လာသွားကြတာပေါ့။ အဓိကကတော့ OOP မှာရှိတဲ့ ရေးထုံးပုံစံတွေကြောင့် code ရေးရတဲ့နေရာ မှာပိုပြီးတော့ သပ်သပ်ရပ်ရပ်ဖြစ်သွားမယ်၊ ထိန်းသိမ်းရတာပိုလွယ်ပြီးတော့ ပြန်လည်အသုံးပြုရတဲ့ နေရာမှာကောင်းမွန်သွားမယ့်အပြင် တစ်ခြားသောကောင်းကျိုးတွေလည်းအများကြီးရှိပါတယ်။ ဒါကြောင့် လည်း programming language အားလုံးတိုင်းလိုလိုမှာ OOP ဆိုတဲ့ ရေးထုံးကို support လုပ်လာခဲ့ ကြတာဖြစ်ပါတယ်။

Class and Object

အပေါ်မှာ ကျနော်ပြောခဲ့သလိုပဲ OOP ဆိုတာ object သဘောတရားတွေနဲ့ ဖွဲ့စည်းထားတယ်ဆိုတဲ့ အတွက် class နဲ့ object ဆိုတာ OOP ရဲ့ အဓိက ပင်မထောက်တိုင်တွေလို့ သတ်မှတ်လို့ရပါတယ်။ Class ဆိုတာက object ကို ခြုံငုံထားတဲ့ template တစ်ခုလို့ဆိုလို့ရပါတယ်။ မြင်သာအောင်ရှင်းပြရရင် နိုင်ငံ ဆိုတဲ့ ဝေါဟာရက class ဖြစ်ပြီးတော့ မြန်မာ၊ ဂျပန်၊ အမေရိကန် စတဲ့ အရာတွေက object အဖြစ် သတ်မှတ်နိုင်ပါတယ်။ တစ်နည်းအားဖြင့် ဆိုရရင် class ဆိုတာက logical 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။

```

1 <?php
2
3 class Country {
4     // property
5     public $name = 'Myanmar';
6
7     // method
8     public function getName() {
9         return $this->name;
10    }
11 }
12 ?>

```

ကျနော် country ဆိုပြီး class လေးတစ်ခုဆောက်ထားပါတယ်။ class ထဲမှာ name ဆိုတဲ့ property လေးထည့်ထားတယ်၊ value ကို Myanmarဆိုပြီး တစ်ခါတည်းပေးထားပါတယ်။ နောက်တစ်ခု getNa-

me ဆိုတဲ့ method လေးရေးထားတယ်။ သူကတော့ class ထဲမှာရှိတဲ့ name property ကို return ပြန်ပေးပါမယ်။ ဒါကရိုးရိုးရှင်းရှင်း class လေးတစ်ခုဖြစ်ပါတယ်။ အဲဒီ class ကနေ object ဆိုတာဘယ်လိုဖြစ်လာလဲဆက်ကြည့်ရအောင်။

```

1 <?php
2
3 class Country {
4     // property
5     private $name = 'Myanmar';
6
7     // method
8     public function getName() {
9         return $this->name;
10    }
11 }
12
13 $myanmar = new Country();
14 var_dump($myanmar);
15 //output = object(Country)#1 (1) { ["name"]=> string(7) "Myanmar" }
16 var_dump($myanmar->getName());
17 //output = string(7) "Myanmar"
18
19 ?>

```

Country class ရဲ့အောက်မှာ 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 ကနေ တစ်ဆင့်ခေါ်ကြည့်မယ်ဆို 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။ ဘာလို့လဲဆိုတာနောက်တစ်ပိုင်းမှာရှင်းပြပေးပါမယ်။

Visibility & Access Modifiers

ကျနော်တို့အပေါ်က code example တွေကနေတစ်ဆင့် တွေ့ခဲ့ရတာက private နဲ့ public ပဲတွေ့ခဲ့ရပါသေးတယ်။ ဒါလေးတွေကို access modifiers တွေလို့ခေါ်ပြီးတော့ 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။

```
1 <?php
2 class Person {
3     public $name = 'Hlaing';
4     private $age = '23';
5 }
6
7 $man = new Person();
8 echo $man->name; // output = Hlaing
9 echo $man->age; // can't access private property error
10 ?>
```

Protected ကလည်း private နဲ့သဘောတရား အတူတူပဲ။ ဒါပေမယ့် protected ကတော့ မူလသူကို 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။

```

1 <?php
2 class ParentClass {
3     public $name = 'Hlaing';
4 }
5
6 // Inheritance (From Parent to child)
7 class ChildClass extends ParentClass {
8
9 }
10
11 $child = new ChildClass();
12 echo $child->name; //output Hlaing
13 ?>

```

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

```

1 <?php
2 class Person {
3     public $name = 'Hlaing';
4     protected $age = '23';
5
6 }
7
8 // Inheritance
9 class Hlaing extends Person {
10     public function output() {
11         //calling protected property `age` using `this` keyword
12         echo $this->age;
13     }
14 }
15
16 $man = new Hlaing();
17 $man->output(); // output = 23
18 $man->age; // protected property can't access outside of the class
19 ?>

```

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၊ လက်တွေ့ဘဝနဲ့ဆက်စပ်ပြီးကြည့်မယ်ဆို 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။

```
1 <?php
2
3 // Parent Class
4 class Person {
5     public $name = "Hlaing";
6
7     public function getName() {
8         echo $this->name;
9     }
10 }
11
12 // Child class (inheritance using extends keyword)
13 class Man extends Person {
14 }
15
16 $man = new Man();
17 echo $man->name; //output - Hlaing
18 $man->getName(); //output - Hlaing
19
20 ?>
```

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 ထပ်ပြီးပြင်ဆင်လို့ရတယ်လို့ ဆိုလိုချင်တာဖြစ်ပါတယ်။

```

1 <?php
2
3 // Parent Class
4 class Person {
5     public $name = "Hlaing";
6
7     public function getName() {
8         echo $this->name;
9     }
10 }
11
12 // Child class (inheritance using extends keyword)
13 class Man extends Person {
14     //overwriting the parent class's method.
15     public function getName(){
16         echo "Hlaing Tin Htun";
17     }
18 }
19
20 $man = new Man();
21 echo $man->name; //output - Hlaing
22 $man->getName(); //output - Hlaing Tin Htun (because we overwrote it)
23
24 ?>

```

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။ ကျနော်အပေါ် မှာ access modifiers အကြောင်းရှင်းပြတုန်းက protected တွေကို ဘယ်လိုပြန်ခေါ်သုံးနိုင်တယ်ဆိုတာရှင်းပြထားပြီးသားဖြစ်တဲ့အတွက်ကြောင့် နောက်တစ်ခေါက်ထပ်ပြီးတော့ ဒီမှာမရေးပြတော့ပါဘူး။ ပြန်ကြည့်ရလွယ်အောင် code sample တော့ ပြန်ထည့်ပေးထားပါမယ်။

```

1 <?php
2 class Person {
3     public $name = 'Hlaing';
4     protected $age = '23';
5
6 }
7
8 // Inheritance
9 class Hlaing extends Person {
10     public function output() {
11         //calling protected property `age` using `this` keyword
12         echo $this->age;
13     }
14 }
15
16 $man = new Hlaing();
17 $man->output(); // output = 23
18 $man->age; // protected property can't access outside of the class
19 ?>

```

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။

```

1 <?php
2
3 final class Person {
4 }
5
6 class Man extends Person {
7 //this will output error
8 //Class Man may not inherit from final class (Person)
9 }
10
11 ?>

```

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

```
1 <?php
2
3 // Parent Class
4 class Person {
5     public $name = "Hlaing";
6     //prevent method overwriting using final keyword
7     final public function getName() {
8         echo $this->name;
9     }
10 }
11
12 // Child class (inheritance using extends keyword)
13 class Man extends Person {
14     //overwriting the parent class's method.
15     //But can't overwrite this time because of final keyword
16     //Cannot override final method Person::getName()
17     public function getName(){
18         echo "Hlaing Tin Htun";
19     }
20 }
21
22 ?>
```

Parent class မှာ method ကို ကြေညာကတည်းက final ခံပြီးကြေညာခဲ့မယ်ဆို child class တွေက overwriteလုပ်နိုင်မှာ မဟုတ်တော့ဘဲ errorတတ်မှာဖြစ်ပါတယ်။ ဒီလောက်ဆို Inheritanceကို သဘောပေါက်ပြီထင်ပါတယ်။ နောက်တစ်ခုဆက်သွားကြတာပေါ့။

Static Properties & Methods

ပုံမှန်အပေါ်မှာ ကျနော်တို့ လေ့လာခဲ့တဲ့အရာတွေအရ class ထဲမှာရှိတဲ့ properties တွေ methods တွေကို လှမ်းခေါ်သုံးချင်ပြီဆို **new** ဆိုတဲ့ keyword နဲ့ instantiate လုပ်ပြီးမှခေါ်သုံးဖြစ်ခဲ့ကြပါတယ်။ static properties တွေ methods တွေရဲ့ ထူးခြားတဲ့အချက်ကတော့ အဲလိုမျိုး class ကို instantiate လုပ်စရာမလိုဘဲ တိုက်ရိုက်ခေါ်ယူအသုံးပြုနိုင်မှာဖြစ်ပါတယ်။ ကျနော်တို့ property အကြောင်းအရင်လေ့လာကြပါမယ်၊ ပြီးမှ methods တွေကိုဆက်ကြတာပေါ့။ နမူနာ static property လေးတစ်ခု အရင်ဆုံး ကြည့်ရအောင်။

```
1 <?php
2
3 class Country {
4     //static property
5     public static $name = 'Myanmar';
6 }
7
8 //using double colon to call static property
9 echo Country::$name;
10 //output - Myanmar
11
12 ?>
```

အပေါ်က နမူနာ country class ထဲမှာ static property တစ်ခုကြေညာထားပါတယ်။ ထူးထူးခြားခြား ဘာမှလုပ်ပေးစရာမလိုဘူး။ **static** ဆိုတဲ့ keyword လေးထည့်ပြီး ကြေညာပေးရုံပဲ၊ ပြန်လည်ခေါ်သုံးလိုတဲ့အချိန်မှာတော့ country class ကို new ဆိုပြီး instantiate လုပ်စရာမလိုတော့ဘဲ classname, double-colon, property_name (**Country::\$name**) ဆိုပြီး တိုက်ရိုက် ခေါ်ယူအသုံးပြုနိုင်သွားမှာ

ဖြစ်ပါတယ်။ ဒီနေရာမှာ အသစ်မှတ်ထားရမှာက static property ကိုခေါ်တော့မယ်ဆို **double colon (::)** သုံးရမယ်ဆိုတာပါပဲ။

Static properties တွေကို သူ့ရဲ့ မူလ 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။

```

1 <?php
2
3 class Country {
4     //static property
5     public static $name = 'Myanmar';
6
7     //normal function calling static property using self::
8     public function getName()
9     {
10         echo self::$name;
11     }
12 }
13
14 // have to instantiate the class as we are calling normal function getName
15 $country = new Country();
16 $country->getName(); //output - Myanmar
17
18 ?>

```

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။

```

1 <?php
2
3 class Country {
4     //static property
5     public static $name = 'Myanmar';
6 }
7
8 class Myanmar extends Country{
9     public function getName()
10    {
11        echo parent::$name;
12    }
13 }
14
15 $country = new Myanmar();
16 $country->getName(); //output - Myanmar
17
18 ?>

```

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။ ဒီအတွက်ဥပမာတစ်ခု တစ်ခါတည်းကြည့်ရအောင်။

```

1 <?php
2
3 class Country {
4     //static method
5     public static function getName()
6     {
7         echo 'Myanmar';
8     }
9 }
10
11 Country::getName(); //output - Myanmar
12
13 ?>

```

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

```
1 <?php
2
3 class Country {
4     //static method
5     public static function getName()
6     {
7         echo 'Myanmar';
8     }
9
10    public function return()
11    {
12        echo self::getName();
13    }
14 }
15
16 $country = new Country();
17 $country->return();
18
19 ?>
```

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 ကိုပြန်လည်အသုံးပြု
ပုံ

```

1 <?php
2
3 class Country {
4     //static method
5     public static function getName()
6     {
7         echo 'Myanmar';
8     }
9 }
10
11 class Myanmar extends Country{
12     public function return()
13     {
14         echo parent::getName();
15     }
16 }
17
18 $country = new Myanmar();
19 $country->return();
20
21 ?>

```

သုံးတဲ့ပုံစံ အတူတူပါပဲ၊ self နေရာမှာ **parent** အဖြစ် အစားထိုးပြောင်းလဲသွားတာကိုပဲ မြင်ရမှာ ဖြစ်ပါတယ်။

Interfaces

Class ဆိုတာကို ရင်းနှီးပြီးသားဆိုရင် Interface ကိုလဲ နားလည်ရလွယ်မှာပါ။ Interface ဆိုတာ ဟာ လည်း တစ်နည်းအားဖြင့် class တစ်ခုပါပဲ။ Interface ဆိုတဲ့ class ထဲမှာ ဘယ် methods တွေ proper-ties တွေရှိမယ်ဆိုတာကို သတ်မှတ်ပေးရပါတယ်။

အဲ့လို သတ်မှတ်လိုက်ခြင်းအားဖြင့် type တူတဲ့ class တွေကို group လုပ်ပေးနိုင်ပါတယ်။ ဥပမာနဲ့ မြင်အောင် ပြောရရင် Myanmar , Japan အစရှိတဲ့ class တွေသည် Country ဆိုတဲ့ Interface အောက် မှာရှိပြီးတော့ 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 လုပ်သွားနိုင်မှာဖြစ် ပါတယ်။ နောက်ထပ် ဥပမာ တစ်ခုထပ်ပေးရရင် cat & dog class တွေမှာ Animal ဆိုတဲ့ interface ထုတ်လို့ ရတယ်ဆိုတဲ့သဘောပေါ့။ Cat ရော dog ရော animal တွေဖြစ်ပြီးတော့ လမ်းလျှောက်နိုင်မယ်၊ အစာစားနိုင်မယ်ဆိုတဲ့ Animal တစ်ကောင်ရဲ့ရှိသင့်တဲ့ methods & properties တွေရှိမှာဖြစ်ပြီး Class Type ကလည်း တူမှာဖြစ်ပါတယ်။ Interface တစ်ခုဆောက်ကြည့်ရအောင်။


```

1 <?php
2 //declaring a interface using `interface` keyword
3 interface Country {
4     public function currentPopulation();
5     public function isAsianCountry();
6 }
7
8 //implementing interface using `implements` keyword
9 class Myanmar implements Country {
10
11 }
12
13 $myanmar = new Myanmar();
14
15 ?>

```

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။ တစ်ခုသတိထားမိမှာက အဲဒီနှစ်ခုလုံးကို ကြေညာပဲကြေညာထားပြီးတော့ implement လုပ်ထားခြင်းမရှိပါဘူး။ အဲလိုမျိုး implementation မရှိဘဲ declaration ဘဲရှိတဲ့ methods တွေကို abstract methods တွေလို့ခေါ်ပါတယ်။ Interface ထဲမှာ abstract methods တွေပဲ ကြေညာလို့ ရမယ်ဆိုတာကိုလည်း သတိထပ်ချပ်ထားသင့်ပါတယ်။

Implementation လုပ်ရမယ့် code တွေက သူ့ကို implements လုပ်မယ့် class တွေ (concrete class) 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 တတ်မှာ ဖြစ်ပါတယ်။ ဒါကြောင့် Implementation လုပ်လိုက်ရအောင်။

```
1 <?php
2
3 class Myanmar implements Country {
4     public function currentPopulation() {
5         echo "53.71 million";
6     }
7
8     public function isAsianCountry() {
9         echo "true";
10    }
11 }
12
13 $myanmar = new Myanmar();
14 $myanmar->currentPopulation();// 53.71 million
15 $myanmar->isAsianCountry();// true
16
17 ?>
```

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

နောက်ထပ် country အသစ်တစ်ခု (concrete class) ထပ်ထည့်ကြည့်ရအောင်။

```

1 <?php
2
3 class Japan implements Country {
4     public function currentPopulation() {
5         echo "126.5 million";
6     }
7
8     public function isAsianCountry() {
9         echo "true";
10    }
11 }
12
13 $japan = new Japan();
14 $japan->currentPopulation();//126.5 million
15 $japan->isAsianCountry();//true
16
17 ?>

```

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 နားလည်သွားမယ် ထင်ပါတယ်။ သူနဲ့အခြေခံ သဘောတရားခြင်းဆင်တဲ့ Abstract class ကိုဆက်ပြီးတော့လေ့လာကြ ပါမယ်။

Abstract Classes

Abstract class ရဲ့ အခြေခံသဘောတရားက 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။

```
1 <?php
2 //abstract class declaration
3 abstract class Country {
4     //abstract method (have to implement from concrete classes)
5     public abstract function currentPopulation();
6     //concrete method (already implemented)
7     public function isAsianCountry(){
8         echo "true";
9     }
10 }
11 ?>
```

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

```
1 <?php
2 //abstract class declaration
3 abstract class Country {
4     //abstract method (have to implement from concrete classes)
5     public abstract function currentPopulation();
6     //concrete method (already implemented)
7     public function isAsianCountry(){
8         echo "true";
9     }
10 }
11
12 //using extends keyword to extend the abstract class
13 class Myanmar extends Country {
14     public function currentPopulation() {
15         echo "53.71 million";
16     }
17 }
18
19 class Japan extends Country {
20     public function currentPopulation() {
21         echo "126.5 million";
22     }
23 }
24
25 $myanmar = new Myanmar();
26 $myanmar->currentPopulation();// 53.71 million
27 $myanmar->isAsianCountry();// true
28
29 $japan = new Japan();
30 $japan->currentPopulation();//126.5 million
31 $japan->isAsianCountry();//true
32 ?>
```

Abstract class ကိုပြန်ခေါ်သုံးတဲ့ နေရာမှာတော့ Interface ကိုခေါ်သုံးသလိုမျိုး **implements** ဆိုတဲ့ keyword မဟုတ်တော့ဘဲ **extends** ဆိုတဲ့ keyword နဲ့ ခေါ်ရမှာဖြစ်ပါတယ်။ နောက်တစ်ခုထူးခြားသွားတာက isAsianCountry ဆိုတဲ့ method မှာ abstract class ထဲကိုရောက်သွားပြီးတော့ Myanmar & Japan ဆိုတဲ့ concrete class တွေထဲမှာ code weight လျော့ချနိုင်သွားမှာဖြစ်ပါတယ်။ result ကတော့တူတူပဲရနေဦးမှာပဲဖြစ်ပါတယ်။ လာမယ့်အပိုင်းမှာ Interface နဲ့ Abstract class ရဲ့ အဓိကကွာခြားချက်တွေကိုရေးသွားပါမယ်။ 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 ကအသုံးဝင်တာလဲကို အောက်ကနမူနာ code လေးတွေကြည့်ရအောင်။

```

1 <?php
2
3 class useGmail
4 {
5     public function send()
6     {
7         echo "sending using gmail";
8     }
9 }
10
11 class useHotmail
12 {
13     public function send()
14     {
15         echo "sending using hotmail";
16     }
17 }
18
19 class Client
20 {
21     protected $service;
22
23     public function __construct(useGmail $service)
24     {
25         $this->service = $service;
26     }
27
28     public function submit()
29     {
30         $this->service->send();
31     }
32 }

```

usegmail နဲ့ usehotmail ဆိုပြီး ကျနော်တို့မှာ service class နှစ်ခုရှိပါတယ်။ Client class တစ်ခု ရှိပါမယ်။ Client class code ထဲမှာ construct ဆိုတဲ့ function ကကျနော်တို့အတွက်စိမ်းနေသေးပါတယ်။ နောက် လာမယ့် အပိုင်းတွေမှာရှင်းပြပေးသွားပါမယ်။ လက်ရှိမှာတော့ Client Class ကို instantiate လုပ်မယ် ဆိုရင် construct ထဲမှာရှိတဲ့ parameter အတွက်ပါ တစ်ခါတည်း ထည့်ပေးဖို့ လိုတယ်လို့ လောလော ဆယ် temporary အနေနဲ့မှတ်ထားလိုက်ပါမယ်။ ဒါဆိုရင် client code ကို ခေါ်သုံးမယ်ဆို ဒီလိုလေးသုံး ရပါမယ်။

```
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 ကိုသုံးပြီးပို့ချင်တယ် ဆိုပါစို့၊ ဒါဆိုရင် client class က constructor ထဲမှာသွားပြောင်းပေးဖို့လိုပါတယ်။ လက်ရှိမှာ class က တစ်ခုတည်း 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 မှာကျနော်တို့ကိုအလုပ်ရှုပ်စေနိုင်မယ့် client class ထဲမှာ usegmail usehotmail ဆိုတဲ့ concrete class တွေကို manual မထည့်တော့ဘဲ ဒီလိုမျိုး interface ကို constructor ထဲမှာတိုက်ရိုက်ထည့်ထားလိုက်ပါမယ်။

```

1 class Client
2 {
3     protected $service;
4
5     public function __construct(mailService $service)
6     {
7         $this->service = $service;
8     }
9
10    public function submit()
11    {
12        $this->service->send();
13    }
14 }
15 // just have to change this if we want to modify all the time
16 $client = new Client(new useHotmail);
17 $client->submit();

```

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

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။

```
1 <?php
2
3 class ChickenCurry
4 {
5     public function addChicken()
6     {
7         var_dump('Add raw chicken');
8         return $this;
9     }
10    protected function addSalt()
11    {
12        var_dump('Add some salt');
13        return $this;
14    }
15
16    protected function addPepper()
17    {
18        var_dump('Add some pepper');
19        return $this;
20    }
21
22    public function cook()
23    {
24        return $this
25            ->addChicken()
26            ->addSalt()
27            ->addPepper();
28    }
29 }
30
31 $chickenCurry = new ChickenCurry();
32 $chickenCurry->cook();
```

chickenCurry ဆိုတဲ့ ကြက်သားဟင်းချက်တဲ့ class တစ်ခုရှိတယ်။ Normal class တွေပဲနော်၊ အထဲမှာ function 4 ခုရှိမယ်။ chickenCurry လိုမျိုးအလားတူ နောက်ထပ် class တစ်ခုထပ်မံတည်ဆောက်ပါမယ်။

```

1 class PorkCurry
2 {
3     public function addPork()
4     {
5         var_dump('Add raw pork');
6         return $this;
7     }
8     protected function addSalt()
9     {
10        var_dump('Add some salt');
11        return $this;
12    }
13
14    protected function addPepper()
15    {
16        var_dump('Add some pepper');
17        return $this;
18    }
19
20    public function cook()
21    {
22        return $this
23            ->addPork()
24            ->addSalt()
25            ->addPepper();
26    }
27 }
28
29 $PorkCurry = new PorkCurry();
30 $PorkCurry->cook();

```

နောက်တစ်ခု ဝက်သားဟင်းချက်တဲ့ porkCurry ဆိုတဲ့ class တစ်ခုရှိတယ်။ class နှစ်ခုကိုယှဉ်ကြည့် လိုက်မယ်ဆို addSalt ရယ် addPepper ရယ်ဆိုတဲ့ တူတဲ့ methods 2 ခုကိုတွေ့ရမယ်။ အဓိကပါဝင် ပစ္စည်းဖြစ်တဲ့ addPork နဲ့ addChicken ပဲကွာသွားမယ်။ ဒီလိုနေရာမှာ abstract class ကိုသုံးပြီးတော့ addSalt & addPepper ကို implement လုပ်ထားမယ်။ main ingredients ဖြစ်တဲ့ ကြက်နွံဝက်ကို abstract method လုပ်ပြီးတော့ concrete class တွေကို implement လုပ်ခိုင်းမယ်။ ဒါဆိုရင်အောက် ကလိုဖြစ်သွားမယ်။

```

1 abstract class Recipe
2 {
3     protected function addSalt()
4     {
5         var_dump('Add some salt');
6         return $this;
7     }
8
9     protected function addPepper()
10    {
11        var_dump('Add some pepper');
12        return $this;
13    }
14
15    public function cook()
16    {
17        return $this
18            ->addMainIngredient ( )
19            ->addSalt()
20            ->addPepper();
21    }
22
23    protected abstract function addMainIngredient();
24 }

```

Recipe ဆိုတဲ့ abstract class လေးတစ်ခုဆောက်လိုက်တယ်။ အဲ့ထဲမှာ ထပ်နေပြီး ပြန်သုံးလို့ရတဲ့ method ခုခုကိုတစ်ခါတည်း 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။

```

1 class ChickenCurry extends Recipe
2 {
3     public function addMainIngredient()
4     {
5         var_dump('Add raw chicken');
6         return $this;
7     }
8 }
9 $chickenCurry = new ChickenCurry();
10 $chickenCurry->cook();

```

Porkcurry ကိုလည်း အပေါ်က chickenCurry လိုပဲ ထပ်မံအသုံးပြုနိုင်မှာ ဖြစ်ပါတယ်။

```

1 class PorkCurry extends Recipe
2 {
3     public function addMainIngredient()
4     {
5         var_dump('Add raw pork');
6         return $this;
7     }
8 }
9 $porkCurry = new PorkCurry();
10 $porkCurry->cook();

```

အရင်တုန်းကရေးခဲ့တဲ့ class အကွဲနှစ်ခုအတိုင်းလိုမျိုး result ကအတူတူပဲရတယ်။ ဒါဆိုရင် ဘယ်လို အခြေအနေမျိုးမှာ abstract ကိုသုံးရမယ်ဆိုတာနားလည်မယ်လို့ထင်ပါတယ်။

Interface နဲ့ abstract class ကိုတော်တော်လေးလည်းရှင်းပြီးသွားပြီဆိုတော့ အဓိကအားဖြင့်သူတို့ နှစ်ခုဘာကွာသွားလဲဆိုတာကို 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 တွေပါခွင့်ရှိမယ်။ 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 ဖြစ်ဖို့မရှိဘူး။ abstract မှာတော့ concrete ဖြစ်ထားတဲ့ member တွေက static type ဖြစ်လို့ရတယ်။

Traits

Trait အကြောင်းကိုနားလည်ဖို့အတွက် Inheritance ရဲ့ အစိတ်အပိုင်းလေးတစ်ခုကို ပြန်နွှေးဖို့လိုပါတယ်။ 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။

```

1 <?php
2 //creating trait using `trait` keyword
3 trait trait1 {
4     public function call() {
5         echo "this is from trait1";
6     }
7 }
8
9 class Client {
10     //calling the trait name with `use` keyword to inherit
11     use trait1;
12 }
13
14 $obj = new Client();
15 $obj->call(); //output - this is from trait1
16 ?>

```


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 အဲဒီ trait ကို ယူသုံးထားပါတယ်။

နောက်ဆုံးအနေနဲ့ကတော့ client code က သုံးချင်တာကို Client class ကို instantiate လုပ်ပြီးသုံး ရုံပါပဲ၊ ဒါတွေကတော့သိပြီးသားဖြစ်တဲ့အတွက်ကျနော်အထူးတစ်လည်မရှင်းပြတော့ပါဘူး။ နောက်တစ် ဆင့်အနေနဲ့ တစ်ခုထက်ပိုတဲ့ trait တွေဆောက်ကြည့်ပြီး ခေါ်သုံးကြည့်ပါမယ်။

```
1 <?php
2 trait trait1 {
3     public function call1() {
4         echo "this is from trait1";
5     }
6 }
7
8 trait trait2 {
9     public function call2() {
10        echo "this is from trait2";
11    }
12 }
13
14 class Client {
15     use trait1;
16     use trait2;
17 }
18
19 $obj = new Client();
20 $obj->call1(); //output - this is from trait1
21 $obj->call2(); //output - this is from trait2
22 ?>
```

အရမ်းကြီးပြောင်းလဲသွားတာမရှိတာကို တွေ့ရပါလိမ့်မယ်။ ထပ်ထည့်ချင်တဲ့ trait ကိုကြည့်တယ်၊ 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 သုံးထားတာကိုတွေ့ရမှာဖြစ်ပါတယ်။

Magic Methods In PHP

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

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

[__construct](#)

```
<?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 ဆိုတဲ့ class လေးထဲမှာ magic method တစ်ခုဖြစ်တဲ့ `construct` ကိုတည်ဆောက်ထားပြီး တော့ 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
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 လုပ်ပေးချင်တဲ့အချိန်မှာအသုံးပြု နိုင်ပါတယ်။ code sample ကိုကြည့်ရင် ပိုမြင်သွားပါမယ်။

```
<?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 တို့ဖြစ်နေမယ် ဆိုရင်လည်း ပုံမှန်ဆို inaccessible ဖြစ်တဲ့ property ကိုလှမ်းခေါ်မယ်ဆိုရင် error တတ်နိုင်ပေမယ့် set magic method ကိုသုံးထားမယ်ဆိုရင် အလုပ်လုပ်နိုင်မှာဖြစ်ပါတယ်။

__get

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

```
<?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
?>
```

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
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
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>

အဓိကရည်ရွယ်ချက်ကတော့ ဘယ်အချိန်မှာဘယ်လို magic methods တွေကအသုံးဝင်နိုင်တယ်ဆိုတာကိုနားလည်ပြီးတော့ အသုံးပြုနိုင်ဖို့ပဲဖြစ်ပါတယ်။

Polymorphism

Polymorphism ဆိုတာစကားလုံးအခေါ်အဝေါ်အရသာရှုပ်သယောင်ရှိပေမယ့် တစ်ကယ်တမ်းတော့ အဲ့ လောက်မရှုပ်ပါဘူး။ သူ့ရဲ့ အဓိက ရည်ရွယ်ချက်သည် client code က interaction ပေါ်မူတည်ပြီး adaptable (အဆင်ပြေအောင်လုပ်ဆောင်နိုင်စွမ်း) ဖြစ်အောင် လုပ်ဆောင်ပေးပါတယ်။ ကျနော်တို့ အပေါ် မှာသင်ခဲ့တဲ့ inheritance အပေါ်မူတည်ပြီး polymorphism example တစ်ခု တစ်ခါတည်း ကြည့်ကြ ရအောင်။

```
<?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 အားလျှော့စွာ 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 functionဟာကွဲပြားသွားမှာပဲဖြစ်ပါတယ်။ ဒီလိုမျိုး scenario ကို polymorphism လို့ခေါ်ပါတယ်။ Interface တစ်ခုကို အမျိုးအစားတူလို့ share သုံးပေမယ့် ပါဝင်တဲ့ functions တွေက မတူကွဲပြားတာကို ဆိုလိုခြင်းဖြစ်ပါတယ်။ အခုတွေ့ခဲ့ရတာကို Dynamic Polymorphism လို့ခေါ်ပြီးတော့ 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 in PHP

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 ဆိုလိုခြင်း ဖြစ်ပါတယ်။ မြင်သာသွားအောင် code examples တွေနဲ့ ချည်းကပ်ကြည့်ရအောင်။ ကျနော်တို့ dinner စားတော့မယ်ဆိုပါစို့၊ Dinner ဆိုတဲ့ class ရှိမယ်၊ စားတာကဟုတ်ပြီ၊ ဘာစားမှာလဲပေါ့။ ဒါဆိုရင် food ဆိုတဲ့ အစားအသောက် dependency တစ်ခုထပ်လိုလာပါပြီ။ code နဲ့ချရေးကြည့်ရအောင်။

```
<?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။ အဲဒါမှာ construct တစ်ခုပါတယ်၊ get ဆိုတဲ့ method ပါမယ်၊ get ကိုခေါ်မယ်ဆို Class ကို instantiate လုပ်တုန်းက ထည့်လိုက်တဲ့ အစားအသောက်ကို return ပြန်လုပ်ပေးမှာဖြစ်ပါတယ်။ ဒါဆို နောက်ထပ် dinner class ကို ဆောက်ရအောင်။

```
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၊ ပုံမှန်အတိုင်းပဲရေးထားပါသေးတယ်။ Dinner ဆိုတဲ့ class ထဲမှာ eat ဆိုတဲ့ method ရှိမယ်၊ အဲဒါမှာ 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 အသုံးပြုပြီးတော့ဖြေရှင်းနိုင်ပါတယ်။ DI သုံးပြီးတော့ Dinner class ကို ပြန်ပြင်ရေးကြည့်ပါမယ်။

```
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 သူ့ကိုယ်တိုင် create လုပ်စရာမလိုတော့တဲ့အတွက် tight coupling မဖြစ်တော့တဲ့အပြင် hardcode-declaring တွေကိုပါ reduce လုပ်နိုင်သွားပြီပဲဖြစ်ပါတယ်။ အစမှာပြောခဲ့တဲ့ definition ကိုပြန်ကောက်ရရင် dinner class ကသူလိုအပ်တဲ့ food dependency class ကိုကိုယ်တိုင် create လုပ်ခြင်းမရှိဘဲတစ်ခြားသော object က create လုပ်ပေးထားတာကို တိုက်ရိုက်ယူသုံးလိုက်ခြင်းဖြစ်ပါတယ်။ ဒါကို dependency inject လုပ်လိုက်တယ်လို့ဆိုလိုခြင်းဖြစ်ပါတယ်။

အခုလက်ရှိရေးလိုက်တဲ့ example မှာတော့ dependency injection ကို constructor ကနေ တစ်ဆင့် လုပ်လိုက်ပေမယ့် constructor တင်မဟုတ်ဘဲ တစ်ခြားသော setter methods တွေ interface တွေမှ တစ်ဆင့်လည်း လုပ်ဆောင်နိုင်ပါတယ်။

