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

```php
$a = 1;
$b = 2;
$plus = $a + $b;
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။