PHP
OBJECT ORIENTED
PROGRAMING

- ◢ Class & Object
- ◢ Constructor Method
- ◢ Access Modifiers
- ◢ Inheritance
- ◢ Method Overriding, Overloading
- ◢ Abstract Classes
- ◢ Static Methods,Properties

PHP OOP

Object-oriented programming has several advantages over procedural programming.

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the PHP code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

Extra Reward ভিডিও সম্পর্কে

◢ Extra Reward টপিক পড়তে গিয়ে, কোর্সের মুল প্রবাহ থেকে সরে যাবেন না।

◢ Extra Reward টপিক গুলোতে স্ট্যাক হয়ে, দীর্ঘ সময় সেখানেই পার করে দিবেন না।

◢ Reward ভিডিও আপনি আপনার সুযোগ সময় মত, ফিউচার লার্নিং এর ক্ষেত্রে সেগুলো ব্যাবহার করবেন ।

◢ Extra Reward ভিডিও কোর্স ধারাবাহিক প্রবাহের সাথে যুক্ত নয়

◢ Extra Reward টপিকের সাথে লাইভ ক্লাস- সাপোর্ট ক্লাস যুক্ত নয়

◢ কোর্সের মুল সিলেবাস মডিউল, আপনার লারাভেল ডেভেলপার হয়ে ওঠার জন্য যথেষ্ট

◢ যারা Extra Reward ভিডিও আনলক করতে পারবেন না, বিষয়টি নিয়ে হতাশ হওয়ার কিছু নেই। একেবারে বিগিনার অবস্থায় , আনলক করতে না পারাটাই আপনার জন্য মঙ্গল জনক । পরবর্তিতে আবার আনলক করার আরও সুযোগ আসবে, Don't Worry.

MESSAGE FROM
Your Well Wishers

Class & Object

- A class is a template for objects
- An object is an instance of class.

```php
index.php

3   class MyClass{
4
5   }
6
7   $obj = new MyClass;
8
```

Constructor

A constructor allows you to initialize an object's properties upon creation of the object.

```php
3    class MyClass{
4
5        function __construct() {
6            echo "I am constructor";
7        }
8    }
9
10   $obj = new MyClass();
11
```

index.php

Constructor Parameter

Passing constructor parameter as same as you function/method parameter you passed before

```php
index.php

3   class MyClass{
4       function __construct($msg) {
5           echo $msg;
6       }
7   }
8
9   $obj = new MyClass("I am constructor");
```

PHP - Access Modifiers

Properties and methods can have access modifiers which control where they can be accessed.

◢ public

The property or method can be accessed from everywhere. This is default

◢ protected

The property or method can be accessed within the class and by classes derived from that class

◢ private

The property or method can ONLY be accessed within the class

Access Modifier

Example 01

◢ public

The property or method can be accessed from everywhere.

```php
index.php

3    class MyClass{
4        public $name="Rabbil";
5        protected $age="33";
6        private $city="Dhaka";
7    }
8
9    $Obj = new MyClass();
10   echo $Obj->name;  // OK
11   echo $Obj->age;  // ERROR
12   echo $Obj->city; // ERROR
```

Access Modifiers

Example 02

◢ public

The property or method can be accessed from everywhere.

```php
index.php

 3    class MyClass{
 4
 5        public function name($name) {
 6            echo $name;
 7        }
 8        protected function age($age) {
 9            echo $age;
10        }
11        private function city($city) {
12            echo $city;
13        }
14    }
15
16    $obj = new MyClass();
17    $obj->name("Jack");    // Working Fine
18    $obj->age("30");       // Error
19    $obj->city("London");  // Error
```

Access Modifiers

◢ private

The property or method can ONLY be
accessed within the class

```php
index.php

3    class MyClass{
4
5        private $num1=20;
6        private $num2=30;
7
8        public function addTwo() {
9            $num3=$this->num1+$this->num2;
10           echo $num3;
11       }
12   }
13
14   $obj = new MyClass();
15   $obj->addTwo();
16
```

Access Modifiers

◢ protected

The property or method can be accessed within the class and by classes derived from that class

```php
class MyClass{

    protected $num1=20;
    protected $num2=30;

    public function addTwo() {
        $num3=$this->num1+$this->num2;
        echo $num3;
    }
}

$obj = new MyClass();
$obj->addTwo();
```

index.php

Static Methods

Static methods can be called directly - without creating an instance of the class first.

```php
index.php

3    class MyClass{
4        static function addTwo() {
5            $num1=20; $num2=30;
6            $num3=$num1+$num2;
7            echo $num3;
8        }
9    }
10
11   MyClass::addTwo();
12
```

Static Properties

Static properties can be called directly - without creating an instance of a class.

```php
index.php

3    class MyClass{
4        public static $cityList = array(
5            'Dhaka',
6            'Rangpur',
7            'Rajshahi'
8        );
9    }
10   echo MyClass::$cityList[1]
```

Why Static

▲ 01

A static method belongs to the class rather than the object of a class.

▲ 02

When we no need to use OOP features like Inheritance/ Overriding/ Overloading etc.

▲ 03

Static is used for utility action preposes

▲ 03

Static is used for utility action preposes

Static Features

- Static methods can be accessed directly in static and non-static methods.

- We can call other static methods inside another static methods.

- We can access static methods directly inside a non static methods.

- We can only access a static variables inside a static function.

- Non static methods and variables cannot be accessed inside a static method, it throws a compile time error

  to change the modifier to static.

- We can also access static methods inside a static block.

- Cannot use 'this' keyword in static.

- Static methods cannot be overridden , but static methods can be overloaded.

Inheritance Concepts

It is a concept of accessing the features of one class from another class. If we inherit the class features into another class, we can access both class properties. We can extends the features of a class by **using 'extends' keyword.**

- It supports the concept of hierarchical classification.

- Inheritance has three types, single, multiple and multilevel Inheritance.

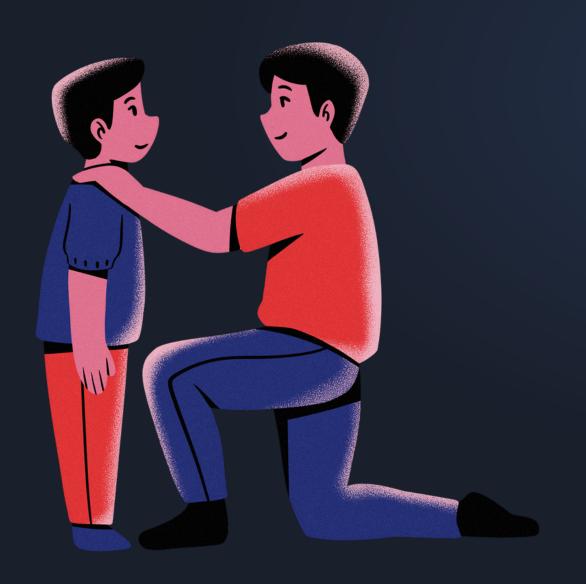- PHP supports only single inheritance, where only one class can be derived from single parent class.

Inheritance Concepts

```php
index.php

3    class Father{

4

5    }

6

7

8    class Son extends Father{

9

10   }
```

Inheritance

Example 01

How father's method can use his son directly


```php
3   class Father{
4       function addTwo(){
5           $num1=10;
6           $num2=10;
7           $num3= $num1+$num2;
8           echo $num3;
9       }
10  }
11
12  class Son extends Father{
13
14  }
15
16  $obj=new Son();
17  $obj->addTwo();
```

`index.php`

Inheritance Override

Son can modify or change father's properties

```php
3   class Father{
4       function addTwo(){
5           $num1=10; $num2=10;
6           $num3= $num1+$num2;
7           echo $num3;
8       }
9   }
10  class Son extends Father{
11      // Method overriding
12      function addTwo(){
13          $num1=30; $num2=40;
14          $num3= $num1+$num2;
15          echo $num3;
16      }
17  }
18  $obj=new Son();
19  $obj->addTwo();
```

Abstract Class

We can create direct object when any class declared as abstract

```php
// Class Abstract
abstract class Father{
    function addTwo(){
      $num1=10; $num2=10;
      $num3= $num1+$num2;
      echo $num3;
    }
}
class Son extends Father{


}
// This is correct
$obj2=new Son();
$obj2->addTwo();

// This will error
$obj1=new Father();
$obj1->addTwo();
```

index.php

PHP Namespaces

Namespaces are qualifiers that solve two different problems:
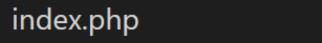
◢ 01

They allow for better organization by grouping classes that work together to perform a task

◢ 02

They allow the same name to be used for more than one class

PHP Namespaces

PHP Namespaces are the way of encapsulating items so that same names can be reused without name conflicts

```php
4    namespace Namespace1 {
5
6        class MyClass{
7
8            function addNumber(int $num1,int $num2){
9             echo $num1+$num2;
10           }
11
12       }
13   }
14
15
16   namespace Namespace2 {
17
18       class MyClass{
19
20           function addNumber(int $num1,int $num2){
21            echo $num1+$num2;
22          }
23
24       }
25   }
```

Learn with Rabbil

PHP Interface

This is a blue print class. Interfaces allow you to specify what methods a class should implement.

```php
interfaceClass.php

1   <?php
2   interface ProductInterface {
3       public function ProductList();
4       public function ProductListByCategory();
5       public function ProductListByBrand();
6       public function ProductByID();
7   }
8
9   class Product implements ProductInterface {
10      public function ProductList() {
11          echo "Product List Here";
12      }
13  }
14
15  $ProductObj= new Product();
16  $ProductObj->ProductList();
17  ?>
```

PHP Method Overloading

- __call() – triggered while invoking overloaded methods in the object context

- __callStatic() – triggered while invoking overloaded methods in static context.

```php
OverLoaded.php

class Product
{
    public function __call($name,$param)
    {
        print_r($param);
    }
    public static function __callStatic($name, $param)
    {
        print_r($param);
    }
}
$objProduct = new Product();
$objProduct->OverLoadedMethod(1,2,3,3);
$objProduct::OverLoadedStaticMethod("A","B","C");
?>
```

PHP Method Overloading

- __call() – triggered while invoking overloaded methods in the object context

- __callStatic() – triggered while invoking overloaded methods in static context.

```php
class Product
{
    public function __call($name,$param)
    {
        print_r($param);
    }
    public static function __callStatic($name, $param)
    {
        print_r($param);
    }
}
$objProduct = new Product();
$objProduct->OverLoadedMethod(1,2,3,3);
$objProduct::OverLoadedStaticMethod("A","B","C");
?>
```

OverLoaded.php