PHP Object Oriented Programming

# What Is Object - Oriented Programming?

**অবজেক্ট অরিয়েন্টেড প্রোগ্রামিং কি :** অবজেক্ট অরিয়েন্টেড কোডিং সাধারন প্রসিডিউরাল (Procedural) কোডিং এর মতই শুধু পার্থক্য হচ্ছে এখানে একই ধরনের ফাংশন এবং এর সাথে কোডগুলিকে এক একটি অবজেক্ট এর মধ্যে রাখা হয়। কিন্তু প্রসিডিউরাল পদ্ধতিতে একই সাথে সব ফাংশন বা কোড রাখা হয় এতে অনেক অসুবিধা হয় যেমন

\* একই নামের অনেক গুলি ভেরিয়েবল বা ফাংশন তৈরী করতে পারবেননা (যেখানে প্রতিটি ভেরিয়েবলের মান এবং প্রতিটি ফাংশনের কাজ অলাদা হবে।

 \* কোড অনেক বড় হবে ফলে পরে এই কোডে কোন সমস্যা হলে বা নতুন কোড যোগের প্রয়োজন হলে পুরো কোড আবার পড়ে দেখতে হবে যে, নতুন যে ফাংশন বা ভেরিয়েবল (গ্লোবাল) যোগ হচ্ছে সেই নামে আগে থেকে এখানে কোন ভেরিয়েবল বা ফাংশন আছে কিনা।

 আরও অনেক সমস্যা হয় প্রসিডিউরাল কোড লিখলে এবং এই সমস্যাগুলির সমাধানের জন্য অবজেক্ট অরিয়েন্টেড প্রোগামিং নতুন কনসেপ্ট নিয়ে এসেছে। উপরের সমস্যাগুলির সমাধান ছাড়াও আরও অনেক সুবিধা আছে নিচে সংক্ষেপে আলোচনা করা হল:

\*\* একটা স্ক্রিপ্টে যদি ১০ টা অবজেক্ট লিখেন এবং প্রতিটি অবজেক্টে একই নামের ফাংশন বা ভেরিয়েবল থাকে তবু সমস্যা হবেনা যদিও ফাংশনগুলি এক একটা অবজেক্টে এক এক রকম কাজ করবে।এরুপ ভেরিয়েবলেরও ক্ষেত্রেও প্রযোজ্য।

 \*\* আপনার সাইটের প্রতিটি অংশের জন্য আলাদা আলাদা মডিউল তৈরী করে সেখানে প্রাসংগিক অবজেক্ট গুলি ভাগ ভাগ করে রাখতে পারবেন।

 যেমন আপনার সাইট যদি অনেক বড় হয় ধরুন ফেসবুকের মত তাহলে ইউজার ম্যানেজমেন্ট নামে একটা মডিউল বানাতে পারেন। সেখানে ইউজারের লগিন এবং রেজিস্ট্রেশনের জন্য একটা অবজেক্ট বানাতে পারেন, একটা প্রোফাইল অবজেক্ট থাকতে পারে। এরপর মেসেজের জন্য একটা মডিউল বানাতে পারেন সেখানে মেসেজ হ্যান্ডলিং এর জন্য একটা অবজেক্ট থাকতে পারে ইত্যাদি ইত্যাদি।

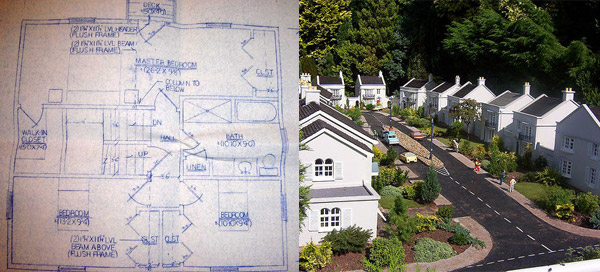
 \*\* ফলে স্ক্রিপ্টগুলি ছোট ছোট হবে এবং পরে নতুন কোড বা অবজেক্ট যোগ করতে চাইলে প্রাসংগিক কোন মডিউলে ঢুকে প্রয়োজনীয় কোডটুকু যোগ করে দিতে পারবেন।

 \*\* আপনার তৈরী অবজেক্টগুলি একটি ফ্রেমওয়ার্কের মত হয়ে থাকবে, ধরুন অন্য আরেকটা সাইট তৈরী করছেন সেখানে একই ধরনের এপ্লিকেশন আছে তাহলে নতুন করে কোড না লিখে এখান থেকে অবজেক্ট তুলে নিয়ে গিয়ে সেই প্রজেক্টে ব্যবহার করতে পারেন।

\*\* কোডগুলি মডিউলার হয়ে থাকবে, কোন অবজেক্ট আরেকটা অবজেক্টে কোন প্রভাব ফেলবেনা। ধরুন সাইটে ইউজার রেজিস্ট্রেশনে সমস্যা হচ্ছে তাহলে পুরো কোড আর পড়ে দেখতে হবেনা শুধু ইউজারের রেজিস্ট্রেশনের জন্য যে অবজেক্ট তৈরী করেছেন সেখানে গিয়ে ঠিক করলেই কাজ হয়ে যাবে।

**Understanding Objects and Classes**

Before you can get too deep into the finer points of OOP, a basic understanding of the differences between **objects** and **classes** is necessary. This section will go over the building blocks of classes, their different capabilities, and some of their uses.



A class, for example, is like **a blueprint for a house**. It defines the shape of the house on paper, with relationships between the different parts of the house clearly defined and planned out, even though the house doesn't exist.

An object, then, is like **the actual house** built according to that blueprint. The data stored in the object is like the wood, wires, and concrete that compose the house: without being assembled according to the blueprint, it's just a pile of stuff. However, when it all comes together, it becomes an organized, useful house.

**Classes form the structure of data and actions and use that information to build objects.** More than one object can be built from the same class at the same time, each one independent of the others. Continuing with our construction analogy, it's similar to the way an entire subdivision can be built from the same blueprint: 150 different houses that all look the same but have different  
families and decorations inside.

# Class:

**ক্লাস (Class) :**অবজেক্ট অরিয়েন্টেড প্রোগ্রমিং এ সবার আগে ক্লাস (class) সম্পর্কে ধারনা নিতে হবে।ক্লাস হচ্ছে অবজেক্টের জন্য টেমপ্লেট।অবজেক্ট তৈরী করার আগে একটা ক্লাস তৈরী করে নিতে হয় এবং এই ক্লাসটিকেই instantiate করলে এটা একটা অবজেক্ট হয়ে যায় (new শব্দ দিয়ে instantiate করতে হয়->নিচে বিস্তারিত আছে)।ক্লাস তৈরীর জন্য প্রথমেই class এই শব্দটি লিখে এরপর যেকোন নাম দিতে হয় আর এরপর দ্বিতীয় বন্ধনির (curly braces) ভিতর সব কোড লিখতে হয়।যেমন

In the real world, objects have characteristics and behaviors. A car has a color, a weight, a manufacturer, and a gas tank of a certain volume. Those are its characteristics. A car can accelerate, stop, signal for a turn, and sound the horn. Those are its behaviors. Those characteristics and behaviors are common to all cars. Although different cars may have different colors, all cars have a color.

class myCar

{

// Property declarations defined here

// Method declarations defined here

}

একটা ক্লাসের ভিতর প্রোপর্টিজ এবং মেথড থাকবে।একটা মেথডে পিএইচপির যেকোন কোড লিখতে পারবেন।

# Object:

**অবজেক্ট (Object) :** অবজেক্ট হচ্ছে অবজেক্ট অরিয়েন্টেড প্রোগ্রামিং এর মুল মেশিন।একটা ক্লাস তৈরী করে তাকে আগে অবজেক্ট বানিয়ে নিতে হয় এরপর এই অবজেক্ট এর প্রোপার্টিজ এবং মেথডে একসেস নিয়ে কাজ করা হয়।ধরুন উপরে যে ক্লাসটি তৈরী করেছি সেটি যদি অবজেক্ট বানাতে চান তাহলে new শব্দটি দিয়ে নিচের মত করে কোড লিখতে হবে।

An *object* is a specific instance of a class. For example, if you create a Car class, you might then go on to create an object called myCar that belongs to the Car class. You could then create a second object, your myCar, also based on the Car class. Think of a class as a blueprint, or factory, for constructing an object. A class specifies the characteristics that an object will have, but not necessarily the specific values of those characteristics. Meanwhile, an object is constructed using the blueprint provided by a class, and its characteristics have specific values.

$car = new myCar ();

# Properties

**প্রোপার্টিজ (Properties) :** প্রোপার্টি হচ্ছে সাধারন পিএইচপি ভেরিয়েবলের মত,ক্লাসের ভিতরে থাকলে এদেরকে প্রোপার্টি বা মেম্বার ভেরিয়েবল বলে।ক্লাসের ভিতর প্রয়োজন হলে এই মান পরিবর্তন করতে পারেন,সাধারন ভেরিয়েবলের মত।

\*\*প্রোপার্টিজকে কোথাও কোথাও attributes বা মেম্বার ভেরিয়েবল বলতে শুনতে পারেন।একই জিনিস।

ক্লাস ডিক্লেয়ার করার পর দ্বিতীয় বন্ধনীর ভিতর স্ক্রিপ্টের শুরুতেই প্রোপার্টিজ লিখতে হয়।যেমন

Properties are much like regular variables, in that they have a name and a value (which can be of any type). Some properties allow their value to be changed and others do not. For example, the Car class might have properties such as color and weight. Although the color of the car can be changed by giving it a new paint job, the weight of the car (without cargo or passengers) is a fixed value.

class Employee {

public $name; // This is a public property

private $age; // This is a private property

protected $wage; // This is a protected property }

# Method:

**মেথড (Method):** ক্লাসের ভিতর ফাংশনকে মেথড বলা হয়।প্রথমে এ function শব্দটি এবং এরপর মেথডের নাম,এভাবে মেথড লেখা হয়।প্রোপার্টিজ ডেটা সংরক্ষন করার জন্য আর মেথড কোন একটা কাজ পরিচালনা করার জন্য।ধরুন আপনি একটা অবজেক্ট তৈরী করলেন যেটা ইউজারের নিবন্ধন সম্পর্কিত, তাহলে এখানে ইউজারের অথেনটিকেশন (যাচাইকরন) এর জন্য একটা মেথড লিখতে পারেন।

\*\*মেথডকে কোথাও কোথাও মেম্বার ফাংশন বা behaviour বলতে শুনতে পারেন।একই জিনিস।

প্রোপার্টিজ এর মত মেথডকেও public, protected, private শব্দগুলি দ্বারা এনক্যাপসুলেট করা হয়। যেমন

Encapsulation

**এনক্যাপসুলেশন (Encapsulation) :** এটা হচ্ছে অবজেক্ট অরিয়েয়েন্টেড প্রোগ্রামিং এর একটা অন্যতম প্রধান বৈশিষ্ট্য।বড় বড় স্ক্রিপ্ট যখন লিখবেন তখন এনক্যাপসুলেশন অনেক সুবিধা দেবে।স্ক্রিপ্টের এক একটা অংশকে আপনি এমনভাবে মুড়িয়ে (এনক্যাপসুলেট) রাখতে পারবেন যে এই অংশ অন্য কোন অংশ দ্বারা প্রভাবিত হবেনা যদিও অংশগুলির মধ্যে একই নামের প্রোপার্টিজ বা মেথড ব্যবহার করে ফেলেন। আরও সহজভাবে বলতে গেলে এনক্যাপসুলেশন এমন একটা পদ্ধতি যেটা দ্বারা স্ক্রিপ্টে প্রোপার্টিজ,মেথড ইত্যাদির স্কোপ ঠিক করে দেয়া যায় (আবার এটা মনে করবেননা যে,মেথড/প্রোপার্টিজকে একসেস মডিফায়ার দিয়ে সীমানা ঠিক করে দেওয়া শুধুমাত্র এটাই এনক্যাসুলেশন।বরং অবজেক্টে ডেটাগুলি বিভিন্নভাবে মোড়ানো থাকার কারনে যে সুরক্ষিত, এটাই এনক্যাপসুলেশন)।

# Property Scopes or Access modifier:

**একসেস মডিফায়ার** (public, private এবং protected) দিয়ে এই কাজ করা যায়।যখনি এই কিওয়ার্ডগুলি দিয়ে কোন মেথড বা প্রোপার্টিজ এর স্কোপ ঠিক করা হবে তখনি এটা নির্দিষ্ট হয়ে যাবে যে এই মেথড এবং প্রোপার্টিজ স্ক্রিপ্টের কোথায় ব্যবহার করা যাবে আর কোথায় যাবেনা। ধারনা আসছেনা?অর্থ্যাৎ এই সুবিধা শুধু কোডাররা পাবে যখন নাকি একই প্রজেক্টে অনেক প্রোগ্রামার কাজ করবেন।

**public :**কোন মেথড বা প্রোপার্টিজ এর সামনে যদি এই কিওয়ার্ডটি ব্যবহার করা হয় তাহলে সেই মেথড এবং প্রোপার্টিজ স্ক্রিপ্টের যেকোন জায়গা থেকে যেকোন কেউ (একজন প্রোগ্রামার যে হয়ত পরে এই স্ক্রিপ্ট টি নিয়ে কাজ করবে) একসেস নিতে পারবে।যেকোন জায়গা বলতে এই ক্লাসের ভিতরে এবং এই ক্লাস থেকে প্রতিপাদিত ক্লাসে (child ক্লাসে) অথবা এই ক্লাসের বাইরে থেকেও এখানে একসেস নেয়া যাবে।কোন মেথড বা প্রোপার্টিজের সামনে যদি কিছু না থাকে তাহলে বাই ডিফল্ট এটি public থাকে।

**private :**কোন ক্লাসের কোন মেথড বা প্রোপার্টিজ এর সামনে যদি এই কিওয়ার্ডটি ব্যবহার করা হয় তাহলে সেই মেথড এবং প্রোপার্টিজকে শুধুমাত্র সেই ক্লাসের ভিতর একসেস নেয়া যাবে।

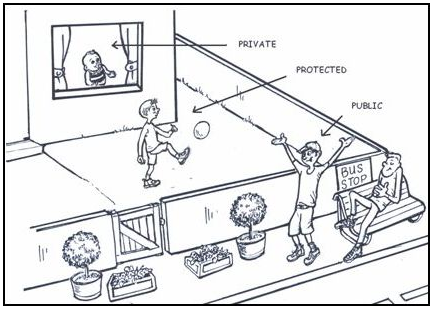
**protected :**কোন ক্লাসের কোন মেথড বা প্রোপার্টিজ এর সামনে যদি এই কিওয়ার্ডটি ব্যবহার করা হয় তাহলে সেই মেথড এবং প্রোপার্টিজকে সেই ক্লাসের ভিতর এবং এই ক্লাস থেকে প্রতিপাদিত ক্লাসে একসেস নেয়া যাবে।

\*\*কোথাও এগুলিকে visibility বলে শুনতে পারেন।অর্থ্যাৎ public মেথড বা প্রোপার্টিজকে স্ক্রিপ্টের যেকোন জায়গা থেকে দেখা যাবে এবং একসেস বা মডিফাই করা যাবে।

private মেথড বা প্রোপার্টিজকে শুধুমাত্র এই ক্লাস থেকে দেখা যাবে এবং একসেস বা মডিফাই করা যাবে।

protected মেথড বা প্রোপার্টিজকে এই ক্লাস এবং এর প্রতিপাদিত ক্লাস থেকে একসেস বা মডিফাই করা যাবে।

PHP supports five class property scopes: *public, private, protected, final*, and *static*



Public:

When you declare a method (function) or a property (variable) as public, those methods and properties can be accessed by:

* The same class that declared it.
* The classes that inherit the above declared class.
* Any foreign elements outside this class can also access those things.

**Example:**

<?php

class GrandPa

{

public $name='Mark Henry'; // A public variable

}

class Daddy extends GrandPa // Inherited class

{

function displayGrandPaName()

{

return $this->name; // The public variable will be available to the inherited class

}

}

// Inherited class Daddy wants to know Grandpas Name

$daddy = new Daddy;

echo $daddy->displayGrandPaName(); // Prints 'Mark Henry'

// Public variables can also be accessed outside of the class!

$outsiderWantstoKnowGrandpasName = new GrandPa;

echo $outsiderWantstoKnowGrandpasName->name; // Prints 'Mark Henry'

2. Private:

When you declare a method (function) or a property (variable) as private, those methods and properties can be accessed by:

* The same class that declared it.

Outsider members cannot access those variables. Outsiders in the sense that they are not **object instances of the declared class itself** and even the classes that **inherit the declared class.**

**Example:**

<?php

class GrandPa

{

private $name = 'Mark Henry';

}

class Daddy extends GrandPa

{

function displayGrandPaName()

{

return $this->name;

}

}

$daddy = new Daddy;

echo $daddy->displayGrandPaName(); // Results in a Notice

$outsiderWantstoKnowGrandpasName = new GrandPa;

echo $outsiderWantstoKnowGrandpasName->name; // Results in a Fatal Error

The exact error messages will be:

Notice: Undefined property: Daddy::$name  
Fatal error: Cannot access private property GrandPa::$name

3. Protected:

When you declare a method (function) or a property (variable) as protected, those methods and properties can be accessed by

* The same class that declared it.
* The classes that inherit the above declared class.

Outsider members cannot access those variables. "Outsiders" in the sense that they are not **object instances of the declared class itself.**

**Example:**

<?php

class GrandPa

{

protected $name = 'Mark Henry';

}

class Daddy extends GrandPa

{

function displayGrandPaName()

{

return $this->name;

}

}

$daddy = new Daddy;

echo $daddy->displayGrandPaName(); // Prints 'Mark Henry'

$outsiderWantstoKnowGrandpasName = new GrandPa;

echo $outsiderWantstoKnowGrandpasName->name; // Results in a Fatal Error

The exact error will be this:

PHP Fatal error: Cannot access protected property GrandPa::$name

## 4. Static

Sometimes it’s useful to create properties and methods that are not invoked by any particular object but rather are pertinent to and are shared by all class instances. For example, suppose that you are writing a class that tracks the number of web page visitors. You wouldn’t want the visitor count to reset to zero every time the class is instantiated, so you would set the property to be of the static scope:

<?php

class Visitor

{

private static $visitors = 0;

function \_\_construct()

{

self::$visitors++;

}

static function getVisitors()

{

return self::$visitors;

}

}

// Instantiate the Visitor class.

$visits = new Visitor();

echo Visitor::getVisitors()."<br />";

// Instantiate another Visitor class.

$visits2 = new Visitor();

echo Visitor::getVisitors()."<br />";

?>

## 5. Final

Marking a method as *final* prevents it from being overridden by a subclass. A finalized method is declared like this:

class Employee

{

final function getName() {

...

}

}

# PHP property Overloading:

# PHP property overloading allows us to create dynamic properties in object context. For creating those properties no separate line of code is needed. A property associated with class instance, and it is not declared within the scope of the class, is considered as overloaded property.

<?php

class Employee

{

public $name;

// Setter

public function setName($name) {

$this->name = $name;

}

// Getter

public function getName() {

return $this->name;

} }

$employee=new Employee();

$employee->name="Hafizur Rahman";

echo $employee->name;

?>

# Setting Properties with the \_\_set() Method

<?php

class Employee

{

public $name;

function \_\_set($propName, $propValue)

{

$this->$propName = $propValue;

}

}

$employee = new Employee();

$employee->name = "Hafizur";

$employee->title = "Lecturer";

echo "Name: ".$employee->name;

echo "<br />";

echo "Title: ".$employee->title;

?>

# Getting Properties with the \_\_get() Method

The *accessor*, or *mutator* method, is responsible for encapsulating the code required for retrieving a class variable. Its prototype follows:

boolean \_\_get([string *property\_name*])

It takes as input one parameter, the name of the property whose value you’d like to retrieve. It

should return the value TRUE on successful execution and FALSE otherwise. An example follows:

<?php

class Employee

{

public $name;

public $city;

protected $wage;

function \_\_get($propName)

{

echo "\_\_get called!<br />";

$vars = array("name","city");

if (in\_array($propName, $vars))

{

return $this->$propName;

} else {

return "No such variable!";

}

}

}

$employee = new Employee();

$employee->name = "Hafizur";

echo $employee->name."<br />";

echo $employee->age;

?>

# Methods

Methods are special functions declared within a class. As you might expect, a method declaration resembles a function declaration. The function keyword precedes a method name, followed by an optional list of argument variables in parentheses. The method body is enclosed by braces:

public function myMethod( $argument, $another )

{

// ...

}

## Calling Methods

To call an object ’ s method, simply write the object ’ s name, then the same arrow used for accessing properties ( - > ), then the method name followed by parentheses:

$object-> method();

class MyClass {

public function hello()

{

echo “Hello, World!”;

}

}

$obj = new MyClass;

$obj-> hello(); // Displays “Hello, World!”

Example:

<?php

class Employee {

private $name;

function setName($name) {

if ($name == "") echo "Name cannot be blank!";

else $this->name = $name;

}

function getName() {

return "My name is ".$this->name."<br />";

}

}

class Executive extends Employee {

function pillageCompany() {

echo "I'm selling company assets to finance my yacht!";

}

}

// Create a new Executive object

$exec = new Executive();

$exec->setName("Hafizur");

echo $exec->getName();

$exec->pillageCompany();

?>

Inheritance.

Once you have written a class, you can derive subclasses from it. This can save lots of  
painstaking code rewriting: you can take a class similar to the one you need to write,  
extend it to a subclass, and just modify the parts that are different. You achieve this  
using the extends operator.

<?php

class User  
{  
public $name, $password;  
function save\_user()  
{  
echo "Save User code goes here";  
}  
}  
class Subscriber extends User  
{  
public $phone, $email;  
function display()  
{  
echo "Name: " . $this->name . "<br>";  
echo "Pass: " . $this->password . "<br>";  
echo "Phone: " . $this->phone . "<br>";  
echo "Email: " . $this->email;  
}  
}

$object = new Subscriber;  
$object->name = "Fred";  
$object->password = "pword";  
$object->phone = "012 345 6789";  
$object->email = "fred@bloggs.com";  
$object->display();  
?>

Or

<?php

// Define a base Employee class

class Employee {

private $name;

// Define a setter for the private $name property.

function setName($name) {

if ($name == "") echo "Name cannot be blank!";

else $this->name = $name;

}

// Define a getter for the private $name property

function getName() {

return "My name is ".$this->name."<br />";

}

} // end Employee class

// Define an Executive class that inherits from Employee

class Executive extends Employee {

// Define a method unique to Employee

function pillageCompany() {

echo "I'm selling company assets to finance my yacht!";

}

} // end Executive class

// Create a new Executive object

$exec = new Executive();

// Call the setName() method, defined in the Employee class

$exec->setName("Richard");

// Call the getName() method

echo $exec->getName();

// Call the pillageCompany() method

$exec->pillageCompany();

?>

মেথড ওভাররাইডিং (Method Overriding) :

পিএইচপিতে প্যারেন্ট ক্লাসের মেথড এবং প্রোপার্টিজ চাইল্ড ক্লাসে একসেস পাওয়া যায় এটাতো সবার জানা আছে।প্যারেন্ট ক্লাসের কোন মেথড যদি চাইল্ড ক্লাসে লিখেন এবং এখানে নতুন কিছু যোগ করে এই চাইল্ড ক্লাসকে instantiate করেন তাহলে যখন এই চাইল্ড ক্লাসের এই মেথড কল করবেন তখন আর প্যারেন্ট ক্লাসের মেথড এক্সিকিউট হবেনা।বরং একই নাম হওয়া স্বত্তেও চাইল্ড ক্লাসের মেথডটিই এক্সিকিউট হবে।যেমন:

*Overriding a method and using the parent operator*

*<?php*

class Dad  
{  
function test()  
{  
echo "[Class Dad] I am your Father<br>";  
}  
}  
class Son extends Dad  
{  
function test()  
{  
echo "[Class Son] I am Luke<br>";  
}  
function test2()  
{  
parent::test();  
}  
}

$object = new Son;  
$object->test();  
$object->test2();

*?>*

This code creates a class called Dad and then a subclass called Son that inherits its  
properties and methods, and then overrides the method test. Therefore, when line 2  
calls the method test, the new method is executed. The only way to execute the over‐  
ridden test method in the Dad class is to use the parent operator, as shown in func‐  
tion test2 of class Son. The code outputs the following:

**[Class Son] I am Luke**  
 **[Class Dad] I am your Father**

If you wish to ensure that your code calls a method from the current class, you can  
use the self keyword, like this:  
self::method();  
Subclass constructors  
When you extend a class and declare your own constructor, you should be aware that  
PHP will not automatically call the constructor method of the parent class. If you  
want to be certain that all initialization code is executed, subclasses should always call  
the parent constructors, as in

**Calling the parent class constructor**

<?php  
class Wildcat  
{  
public $fur; // Wildcats have fur  
function \_\_construct()  
{  
$this->fur = "TRUE";  
}  
}  
class Tiger extends Wildcat  
{  
public $stripes; // Tigers have stripes  
function \_\_construct()  
{  
parent::\_\_construct(); // Call parent constructor first  
$this->stripes = "TRUE";  
}  
}

$object = new Tiger();  
echo "Tigers have...<br>";  
echo "Fur: " . $object->fur . "<br>";  
echo "Stripes: " . $object->stripes;  
  
?>  
This example takes advantage of inheritance in the typical manner. The Wildcat class  
has created the property $fur, which we’d like to reuse, so we create the Tiger class to  
inherit $fur and additionally create another property, $stripes. To verify that both  
constructors have been called, the program outputs the following:

# Constructor

**কনস্ট্রাক্টর মেথড (Constructor Method) :** পিএইচপি ক্লাসে কনস্ট্রাক্টর একটা বিশেষ মেথড।ইচ্ছে করলে ক্লাস তৈরীর সময় কনস্ট্রাক্টর মেথড নাও দিতে পারেন।যদি দেন তাহলে এই ক্লাসটিকে instantiation করার সময় (অর্থ্যাৎ অবজেক্ট তৈরীর সময়) এই মেথডটি অটোমেটিক কল হবে।

আর সহজভাবে বলতে গেলে কনস্ট্রাক্টর হচ্ছে এমন একটি কোডব্লক যেটা অবজেক্ট তৈরীর সময়ই অটোমেটিক এক্সিকিউট হয়।কনস্ট্রাক্টর মেথড লিখতে হয় দুটি আন্ডারস্কোর (\_\_) এবং এরপর construct শব্দটি দিয়ে।যেমন

PHP recognizes constructors by the name \_\_construct (a double underscore precedes the

constructor keyword). The general syntax for constructor declaration follows:

*function \_\_construct([argument1, argument2, ..., argumentN])*

*{*

*// Class initialization code*

*}*

<?php

class Employee{

public $name;

public $title;

public function \_\_construct($name,$title){

$this->name = $name;

$this->title = $title;

}

public function getInfo(){

echo "Employee Name is : ".$this->name."<br/>"."Title : ". $this->title ."<br/>";

}

}

$productObject = new Employee("Tanvir", "Web developer");

$productObject->getInfo();

?>

# Destructors

Just as you can use constructors to customize the object creation process, so can you use destructors to modify the object destruction process. Destructors are created like any other method but must be titled **\_\_destruct().** An example follows:

<?php

class Book

{

private $title;

private $isbn;

private $copies;

function \_\_construct($isbn)

{

echo "<p>Book class instance created.</p>";

}

function \_\_destruct()

{

echo "<p>Book class instance destroyed.</p>";

}

}

$book = new Book("0615303889");

?>

# Object Cloning

Object cloning is the act of making a copy of an object. As [Cody](http://stackoverflow.com/questions/2144506/what-is-object-cloning-in-php/2144523#2144523) pointed out, cloning in PHP is done by making a shallow copy of the object. This means that internal objects of the cloned object will **not**be cloned, unless you explicitly instruct the object to clone these internal objects too, by defining the magic method \_\_clone().

*Cloning an object*  
<?php  
  
class User  
{  
public $name;  
}

$object1 = new User();  
$object1->name = "Alice";  
$object2 = clone $object1;  
$object2->name = "Amy";  
echo "object1 name = " . $object1->name . "<br>";  
echo "object2 name = " . $object2->name;  
?>  
Voilà! The output from this code is what we initially wanted:  
**object1 name = Alice**  
**object2 name = Amy**

<?php

class Corporate\_Drone {

private $employeeid;

private $tiecolor;

// Define a setter and getter for $employeeid

function setEmployeeID($employeeid) {

$this->employeeid = $employeeid;

}

function getEmployeeID() {

return $this->employeeid;

}

// Define a setter and getter for $tiecolor

function setTieColor($tiecolor) {

$this->tiecolor = $tiecolor;

}

function getTieColor() {

return $this->tiecolor;

}

}

// Create new Corporate\_Drone object

$drone1 = new Corporate\_Drone();

// Set the $drone1 employeeid property

$drone1->setEmployeeID("12345");

// Set the $drone1 tiecolor property

$drone1->setTieColor("red");

// Clone the $drone1 object

$drone2 = clone $drone1;

// Set the $drone2 employeeid property

$drone2->setEmployeeID("67890");

// Output the $drone1 and $drone2 employeeid properties

printf("Drone1 employeeID: %d <br />", $drone1->getEmployeeID());

printf("Drone1 tie color: %s <br />", $drone1->getTieColor());

printf("Drone2 employeeID: %d <br />", $drone2->getEmployeeID());

printf("Drone2 tie color: %s <br />", $drone2->getTieColor());

?>

# Abstract Method and Class:

এবস্ট্রাক্ট ক্লাস instantiate করা যায়না তবে inherit করা যায়।এখানে থেকে বের হওয়া চাইল্ড ক্লাসের instantiate করা যায়।এবস্ট্রাক্ট ক্লাস বানানো হয়েছে এর চাইল্ড ক্লাসে কি কি থাকবে সেই নির্দেশনা যাতে প্রোগ্রামার প্যারেন্ট ক্লাসটি (এবস্ট্রাক্ট ক্লাস) দেখে বুঝতে পারে।যেমন:

<?php

abstract class DepartmentInfo {

abstract public function departmentActivity();

}

class StatisticsDept extends DepartmentInfo {

public function departmentActivity() {

echo "Statistics department arrange a tour yearly<br/>";

}

}

class CseDept extends DepartmentInfo {

public function departmentActivity() {

echo "CSE department should arrange programming contest";

}

}

$sdept = new StatisticsDept();

$sdept->departmentActivity();

$cdept = new CseDept();

$cdept->departmentActivity();

?>

আউটপুট:

Statistics department arrange a tour yearly

CSE department should arrange programming contest

ব্যাখ্যা:এখানে দেখুন প্যারেন্ট ক্লাসটি এবস্ট্রাক্ট এবং এখানে একটি এবস্ট্রাক্ট মেথড আছে।তাই এখানে থেকে প্রতিপাদিত প্রতিটি চাইল্ড ক্লাসেও এই মেথডটি রাখতে হয়েছে।যদি না রাখতাম তাহলে এরর দেখাত।

**ইন্টারফেস ক্লাস (Interface Class) :**

এর আগে এবস্ট্রাক্ট ক্লাস সম্পর্কে আলোচনা হয়েছে।এবস্ট্রাক্ট ক্লাস আর ইন্টারফেস ক্লাস হুবহু একই শুধু পার্থক্য হচ্ছে এবস্ট্রাক্ট ক্লাসে এবস্ট্রাক্ট মেথড ছাড়াও অন্য মেথড থাকতে পারে এবং এই মেথডের ভিতর কোড থাকতে পারে (সাধারনত এটাকে মেথড বডি বলে) আর ইন্টারফেস ক্লাসে কোন মেথডে বডি থাকতে পারবেনা।

\*ইন্টারফেস ক্লাস অন্যান্য ক্লাসের মতই লিখতে হয়,এখানে class শব্দের পরিবর্তে interface শব্দ ব্যবহৃত হয়।

\*ইন্টারফেস ক্লাস থেকে ইন্টারফেস ক্লাস extend করা যায়।

\*ইন্টারফেস ক্লাস extends (ইনহেরিট) এবং instantiate করা যায়না বরং (চাইল্ড ক্লাসে) implements করা যায়।যেমন

\*\*চাইল্ড ক্লাসে মেথডগুলি ডিফাইন করার পর ইচ্ছেমত বডি দিতে পারবেন (এই মেথডের ভিতর কোড দিতে পারবেন)।তবে প্যারেন্ট ক্লাসটিতে (এবস্ট্রাক্ট ক্লাসটিতে) এবস্ট্রাক্ট মেথডে কোন বডি থাকবেনা।

\*এবস্ট্রাক্ট প্যারেন্ট ক্লাসে এবস্ট্রাক্ট মেথড ছাড়া অন্য সাধারন মেথড রাখতে পারেন এবং এসব মেথডের বডি থাকতে পারে।এটাই ইন্টারফেস ক্লাসের সাথে এবস্ট্রাক্ট ক্লাসের পার্থক্য

\*\*যদি কোন ক্লাসে একটা এবস্ট্রাক্ট মেথড থাকে তাহলে সেই ক্লাসকেও অবশ্যই এবস্ট্রাক্ট ডিক্লেয়ার করতে হবে।

\*\*প্যারেন্ট ক্লাসে যতগুলি এবস্ট্রাক্ট মেথড থাকবে,এখান থেকে বের হওয়া সব চাইল্ড ক্লাসেও সেই মেথডগুলি থাকতে হবে।

\*\*এবস্ট্রাক্ট ক্লাস তৈরীর জন্য ক্লাসের নামের আগে শুধু abstract শব্দটি দিতে হয় আর এবস্ট্রাক্ট মেথড তৈরীর জন্য একসেস মডিফায়ার এর আগে abstract শব্দটি দিতে হয়।

<?php

interface Moveable {

public function move();

}

class Man implements Moveable {

public function move() {

echo "A man can move<br/>";

}

}

class Vehicle implements Moveable {

public function move() {

echo "A Vehicle also can move";

}

}

$mans = new Man();

$mans->move();

$machine = new Vehicle();

$machine->move();

?>

আউটপুট:

A man can move

A Vehicle also can move

ব্যাখ্যা:এখানে দেখুন Moveable ইন্টারফেসকে implements করেছে Man এবং Vehicle ক্লাসে।ইন্টারফেস ক্লাসে যেসব মেথড থাকবে সেই মেথডগুলি এর implement করা ক্লাসে অবশ্যই থাকতে হবে এবং এখানে প্রয়োজন অনুযায়ী মেথড বডি থাকবে।এটা নিয়ম।

\*\*ইন্টারফেস ক্লাসে যেসব মেথড থাকবে সেগুলি অবশ্যই পাবলিক হতে হবে।

\*\*দুটি অবেজক্টের মধ্যে যদি কোন সম্পর্ক না থাকে শুধুমাত্র ২/১টি মেথডে মিল আছে এমন হলে সাধারনত সেই অবজেক্টগুলির জন্য একটা ইন্টারফেস ক্লাস বানানো হয়।

যেমন উপরের উদাহরনে দেখুন Man এবং Vehicle এর মধ্যে তেমন কোন মিল নেই তবে দুটোই চলতে পারে তাই এখানে এদের জন্য Moveable ইন্টারফেস নেয়া হয়েছে।যদি খুব মিল থাকত তাহলে তো ইনহেরিটেন্স দিয়েই কাজ হয়ে যেত।

\*\*ইন্টারফেস ব্যবহারের আরকেটি সুবিধা হচ্ছে যখন একসাথে অনেক প্রোগ্রামার কাজ করবে তখন হয়ত প্রজেক্ট ম্যানেজার একটা ইন্টারফেস তৈরী করে জুনিয়র প্রোগ্রামার কে দিয়ে দেবে।এতে জুনিয়র প্রোগ্রামারকে আর এপ্লিকেশন নিয়ে ভাবতে হবেনা,সে শুধু ইন্টারফেস দেখে দেখে তার ক্লাসগুলিতে মেথডগুলি পূর্ন (মেথডে কোড লিখবে) করবে।

\*\*একসাথে একাধিক ইন্টারফেস একটা ক্লাসে implements করা যায়।যেমন

<?php

interface Moveable {

public function move();

}

interface Test {

public function testMove();

}

class Man implements Moveable, Test {

public function move() {

echo "A man can move<br/>";

}

public function testMove() {

echo "Multiple Interface multiple inheritence<br/>";

}

}

class Vehicle implements Moveable {

public function move() {

echo "A Vehicle also can move";

}

}

$mans = new Man();

$mans->move();

$mans->testMove();

$machine = new Vehicle();

$machine->move();

?>

আউটপুট:

A man can move

Multiple Interface multiple inheritance

A Vehicle also can move

# ক্লাস অটোলোড করা (Autoloading Class) :

একটা ক্লাসকে ব্যবহার করার আগে তাকে সংযুক্ত করতে হয়।সাধারনত আমরা require\_once বা include ইত্যাদি ফাংশন দ্বারা আগে প্রয়োজনীয় ক্লাসগুলি সংযুক্ত করি এরপর এগুলি দিয়ে কাজ শুরু করি।যেমন

<?php

require\_once 'class.WebcoachbdProduct.php';

require\_once 'class.UserLogin.php';

$productObject = new WebcoachbdProduct();

$productObject->getTutorial(' OOP');

?>

কিন্তু যদি আপনার প্রজেক্টে ১৫০/২০০ টি ক্লাস থাকে এবং কোন স্ক্রিপ্টে যদি ১০০টি ক্লাস লাগে তাহলে সবগুলি এভাবে require\_once দিয়ে যোগ করবেন?

উত্তরে বলুন না,কারন এই বিরক্তিকর কাজ থেকে মুক্তির জন্য এবার ক্লাস অটোলোড ফাংশনটির কাজ শিখবেন।যাইহোক ক্লাস অটোলোড ফাংশন কাজ করানোর জন্য দুটি বিষয় অবশ্যই অনুসরন করতে হবে

-একটা ফাইলে একটা ক্লাস রাখতে হবে

-ফাইল এবং ক্লাসের নাম দেয়ার সময় কোন একটা naming convention অনুসরন করতে হবে।

যেমন দেখুন আমার ফাইলগুলি আমি নাম দিয়েছি এভাবে class.WebcoachbdProduct.php, class.DepartmentInfo.php ইত্যাদি এবং এই ফাইলগুলিতে ক্লাসের নাম দিয়েছি এভাবে class WebcoachbdProduct, class DepartmentInfo ..

এখন এই ক্লাসগুলি আমি অটোলোড করাতে পারি এভাবে

1. <?php
2. function \_\_autoload($class\_name){
3. require\_once 'class.' . $class\_name . '.php';
4. }
5. $prod = new WebcoachbdProduct();
6. $prod->getTutorial(' Framework');
7. ?>

২,৩,৪ নম্বর লাইনে যে ফাংশনটি আছে এটাই ক্লাস অটোলোড করবে।যে ক্লাসকেই instantiate করুন না কেন,সাথে সাথে সেই ক্লাসকে এই ফাংশন অটোমেটিক লোড করবে।যেমন আমি যখনই ৫ নম্বর লাইনে new এর পর ক্লাসের নামটি লিখেছি সাথে সাথে এই নামটি ফাংশনটির $class\_name এর জায়গায় গিয়ে বসে গেছে এবং এরপর ৩ নম্বর লাইন দিয়ে ক্লাস টি যোগ হয়েছে।এভাবে যদি new UserLogin দিতাম তাহলে UserLogin ক্লাসটি অটোলোড হতো যেটা class.UserLogin.php ফাইলে আছে।

\*\*যদি পিএইচপি ফাইলগুলি অন্য কোন ফোল্ডারে থাকে তাহলে require\_once এ উল্লেখ করে দিলেই হবে।যেমন আমি আমার ক্লাস ফাইলগুলি যদি test ফোল্ডারে রাখি তাহলে তখন নিচের মত করে অটোলোড করাতে হবে।

<?php

function \_\_autoload($class\_name){

require\_once 'test/class.' . $class\_name . '.php';

}

$prod = new WebcoachbdProduct();

$prod->getTutorial(' Framework');

?>

# নেমস্পেস (Namespace) :

নেমস্পেস মূলত ব্যবহার হয় ডেটা এনক্যাপসুলেট করার জন্য। একটা স্ক্রিপ্টে ক্লাস, ফাংশন, ভেরিয়েবল ইত্যাদির নাম যদি একই হয় (মানে একই নামের একাধিক মেথড) তখন এই কোড এক্সিকিউট করার সাথে সাথেই এরর দেখাবে অথবা ঠিকমত কাজ করবেনা। এখন ভাবতে পারেন যে একই স্ক্রিপ্টে কেন কেউ একই নামের ক্লাস/মেথড ইত্যাদি ব্যবহার করবে? আসলে এরকম কেউ করেনা, তবে এমন হয়ে যেতে পারে। ধরুন আপনি একটা datetime নামে ক্লাস তৈরী করেছেন এবং আপনার সাইটে একটা 3rd party লাইব্রেরি ব্যবহার করেছেন ধরুন একটা ফ্রেমওয়ার্ক। এখন এই ফ্রেমওয়ার্কেও এই নামে যদি কোন ক্লাস থাকে তাহলে সমস্যার শুরু।

নেমস্পেস আসার আগে অনেক লম্বা লম্বা নাম ব্যবহার করা হত। এই নাম conflict সমস্যা সমাধানের জন্য পিএইচপি ৫.৩.০ ভার্সন থেকে নেমস্পেস সুবিধা চালু করেছে। একটা কোডব্লককে একটা নাম দিতে পারেন এবং পরে সেই নাম ধরে সেই কোডব্লকের কোড ব্যবহার করতে পারেন। যদি স্ক্রিপ্টের অন্য কোন কোডে এই একই নামের ফাংশন বা ক্লাস থাকে তবুও এবার সমস্যা হবেনা। নেমস্পেস ধরে যখন কল করবেন তখন পিএইচপি ঠিক বুঝে নেবে যে এখন কোন্ কোডকে এক্সিকিউট করাতে হবে।

যেমন class. Webcoachbd.php নামে ফাইল তৈরী করে সেখানে নিচের কোড লিখুন।

<?php

namespace Framework;

class Webcoachbd {

  public $title = "default value";

  public function getTutorial($name) {

  echo "Webcoachbd provide massive tutorial on" . $name;

  }

}

?>

এবং class.TestDemo.php ফাইল তৈরী করে সেখানে নিচের কোড লিখুন। দেখুন ২টি কোডেই ক্লাসের নাম একই এমনকি getTutorial() নামের একই নাম সম্বলিত একাধিক মেথডও আছে। শুধু নেমস্পেসের নাম আলাদা

<?php

namespace OOPHP;

class Webcoachbd {

   public $title = "different value";

   public function getTutorial($name) {

   echo "Webcoachbd provide massive tutorial on" . $name . "<br/>";

  }

}

?>

কোডদুটি একই শুধু namespace আলাদা। যাইহোক এবার আরেকটি পিএইচপি ফাইলে এই ফাইলদুটি সংযুক্ত করে নিচের মত কোড লিখে রান করান।

1. <?php
2. require\_once 'class.TestDemo.php';
3. require\_once 'class.Webcoachbd.php';
4. $objWeb = new OOPHP\Webcoachbd();
5. $objWeb->getTutorial(' OOPHP');
6. $objFrm = new Framework\Webcoachbd();
7. $objFrm->getTutorial(' Framework');
8. ?>

আউটপুট:

Webcoachbd provide massive tutorial on OOPHP

Webcoachbd provide massive tutorial on Framework

যদি namespace উঠিয়ে দিয়ে এই কোড রান করান তাহলে এরর আসবে।

\*namespace স্ক্রিপ্টের শুরুতে ডিফাইন করতে হবে আর কল করার সময় namespace এর নামের পর ব্যাকস্ল্যাশ দিয়ে ক্লাসের নাম দিতে হবে (যেমন আমি ৪ এবং ৬ নম্বর লাইনে দিয়েছি)

\*ইচ্ছে করলে একই ফাইলে একাধিক নেমস্পেস ব্যবহার করতে পারেন। সেক্ষেত্রে namespace এরপর নেমস্পেসের নাম এবং এরপর দ্বিতীয় বন্ধনীর ভিতর এক একটা কোডব্লক রাখতে হবে। যেমন

<?php

namespace Framework {

class Webcoachbd {

    public $title = "default value";

    public function getTutorial($name) {

    echo "Webcoachbd provide massive tutorial on" . $name;

    }

  }

}

namespace OOPHP {

class Webcoachbd {

   public $title = "different value";

   public function getTutorial($name) {

   echo "Webcoachbd provide massive tutorial on" . $name . "<br/>";

   }

  }

}

?>

## মডেল ভিউ কন্ট্রোলার স্ট্রাকচার টিউটোরিয়াল (MVC Structure Tutorial in Bangla)

মডেল ভিউ কন্ট্রোলার স্ট্রাকচার (MVC) :  এটা একটা সফটওয়্যার ডিজাইন প্যাটার্ন বা পদ্ধতি। এরুপ আরো অনেক ডিজাইন প্যাটার্ন আছে। কোডিং করার সময় এসব প্যাটার্ন অনুসরন করলে কোড দেখতে পরিষ্কার এমনকি পরে মেইনটিনেন্সে (কোড রিফ্যাক্টরিং) করা সহজ হয়। যেমন raw PHP দিয়ে কোড করার সময় আমরা সাধারনত একই পেজে এইচটিএমএল, লজিক, ডেটাবেসের একসেস কোড, কোয়েরি ইত্যাদি লিখে থাকি। এই কোডে পরে আরেকজন ডেভেলপার কাজ করা খুবই দু:সাধ্য। আসলে নিজেই পরে এই কোড দেখলে বুঝতে পারবেন না, বিশেষ করে যখন বড় প্রজেক্ট হবে এবং স্ক্রিপ্টগুলি অনেক বড় হবে।

যেমন নিচের কোড দেখুন এখানে কোন ডিজাইন প্যাটার্ন অনুসরন করা হয়নি

ভাবে কোড লিখলে পরবর্তীতে এই কোড সম্পাদন করা বা আপডেট করা ইত্যাদি বেশ কঠিন হয়ে যায় অনেকটা অসম্ভব। মডেল ভিউ কন্ট্রোলার ডিজাইন প্যাটার্ন এই সমস্যার সমাধান নিয়ে এসেছে। এই MVC Structure কোডগুলিকে ৩ ভাগে ভাগ করে লেখার সুবিধা দেয় কিন্তু কাজ সব একসাথেই করবে যেমন একসাথে কোডগুলি লিখলে কাজ হবে।এই ভাগ ৩টি হল

মডেল (Model)

ভিউ (View)

কন্ট্রোলার (Controller)

### মডেল (Model):

ধরুন আপনি কোডইগনাইটারে কোড লিখে লিখে একটা এপ্লিকেশন তৈরী করতেছেন, এই এপ্লিকেশনের সব কোডগুলির মধ্যে মডেল অংশে শুধুমাত্র ঐ কোডগুলি (অথবা বলতে পারেন লজিকগুলি বা ফাংশনগুলি) থাকবে যেকোডগুলির দ্বারা আপনি ডেটাবেসে ডেটা ইনসার্ট, ডিলিট, আপডেট ইত্যাদি করবেন। অর্থ্যাৎ মডেল অংশ আপনার ডেটা স্ট্রাকচার রিপ্রেজেন্ট করবে। এই লজিকগুলিকে বলা হয় বিজনেস লজিক (business logic)

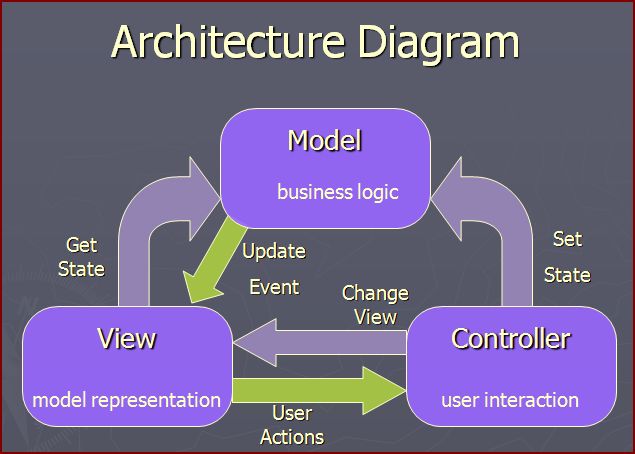
### ভিউ (View):

এখানে ঐ সমস্ত কোডগুলি থাকবে যেগুলির দ্বারা ইউজার একটা পেজ দেখবে। অর্থ্যাৎ এটা একটা সাধারন ওয়েব পেজ বা বলতে পারেন এইচটিএমএল পেজ। কোডইগনাইটারে এখানে একটা পেজের একটা অংশও থাকতে পারে যেমন পেজের হেডার, ফুটার, আরএসএস পেজ ইত্যাদি।

### কন্ট্রোলার (Controller):

কন্ট্রোলারে ঐ সমস্ত কোডগুলি থাকবে যার দ্বারা মডেল এবং ভিউ এর লজিকগুলি একে অপরের সাথে যোগাযোগ করবে। অর্থ্যাৎ মডেল এবং ভিউ কিভাবে কাজ করবে এই লজিকগুলি এখানে লেখা হয়ে থাকে।

কোডইগনাইটার ফ্রেমওয়ার্ক MVC অর্থ্যাৎ মডেল ভিউ কন্ট্রোলার স্ট্রাকচার এর ভিত্তি করে কাজ করে অর্থ্যাৎ এখানে এভাবে কোড লেখা হয়ে থাকে।



**push-MVC বনাম pull-MVC (ওয়েব এপ্লিকেশন)**  
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

MVC (Model View Controller) একটি সফটওয়্যার ডিজাইন প্যাটার্ন এটা অনেকেই জানেন তবে MVC যে ২ ধরনের সেটা অনেকের আজানা। যাইহোক সাধারনত আমরা যারা ওয়েব এপ্লিকেশন নিয়ে কাজ করি তারা push-MVC নিয়ে কাজ করি। কেননা বেশির ভাগ ফ্রেমওয়ার্ক এই আর্কিটেকচার অনুসরন করে।

**push-MVC:** এই পদ্ধতিতে মেথড/action ব্যবহার করে প্রসেসিং হয় (সাধারনত URL থেকে একশন চিনে নেয় কিংবা রাউটিং থেকে) এবং এরপর কন্ট্রোলার থেকে ডেটা ভিউতে (View) পুশ করে। push-MVC এর আর্টিটেকচার action-based. উদাহরন: Symfony, CodeIgniter এসব ফ্রেমওয়ার্ক এই আর্কিটেকচার ব্যবহার করে।

**pull-MVC:** এই পদ্ধতিতে ভিউ লেয়ার থেকে সিদ্ধান্ত হয় কোন কোন কন্ট্রোলার থেকে ডেটা আনবে, কন্ট্রোলার থেকে আবার সিদ্ধান্ত হয় কোন মডেল থেকে ডেটা আসবে। প্রয়োজনে ভিউ থেকে ঠিক করা যায় যে, একাধিক কন্ট্রোলার থেকে ডেটা আনবে কিনা (মানে একাধিক কন্ট্রোলার থেকে ডেটা আনা যায়)। ডেটা pull করে তাই নাম pull-MVC. জুমলা ফ্রেমওয়ার্ক ([framework.joomla.org](http://framework.joomla.org/) ) এই আর্কিটেকচার ব্যবহার করে। এটার আর্কিটেকচার component-based

তবে কিছু কিছু ফ্রেমওয়ার্ক push এবং pull দুটোই ব্যবহার করে যেমন জেন্ড, ওয়াই আই আই ইত্যাদি।