Classes can have variables within it. Those variables are called properties. A property is a normal PHP variable which is in any data type (integer, string, array, object, etc).

In classes, before declaring a variable, we should add the visibility keyword to define where the variable is available. You will learn more about this later. For now, keep in mind that adding the public keyword in front of the variable will make the variable available everywhere.

```
<?php
classHouse {
      public $primaryColor = 'black';
      public $secondaryColors = [
            'bathroom' =>'white',
            'bedroom' =>'light pink',
            'kitchen' =>'light blue'
      ];
      public $hasPool = false;
      public $extra;
}
```

As in the above example, we can declare any type of variable as a property. We can add default values for those properties. (Black is the default for $primaryColor) Also, note that the property $extra does not have a default value.
Remind the blueprint explanation. A blueprint can have default widths and heights in it. But, also some houses will need some additional features that are not defined in the blueprint. So, those additional features do not have a default value. But, they can be added to a house. (Ex: Having a pool)

In Object Oriented Programming in PHP, methods are functions inside classes. Their declaration and behavior are almost similar to normal functions, except their special uses inside the class.

How to declare a method?

Let's declare a method inside a class named Example class to echo out a simple string that we give.

```
<?php
classExample {
      publicfunctionecho($string) {
            echo $string;
      }
}
```

We use the public keyword to make the method available inside and outside the class. You will learn more about this in the visibility chapter.

How to call a method?

```
$example = newExample();
$example ->echo('Hello World');
```

Result: Hello World

Explained:

First, we create an object ($example) from the class Example

Next, we call the method echo with -> (object operator) and () (parentheses)

The parentheses contain the arguments as usual

Changing a property value using methods

Let's implement the things we learned in the above example to our House class. Now we are going to change the color of the house. For ease, all the properties are removed from the House class, except $primaryColor.

By default the color of the house is black. We need to change it to another one.

```
<?php
classHouse{
      public $primaryColor = 'black';
      publicfunctionchangeColor($color){
            $this ->primaryColor = $color;
      }
}

// creates an object from the class
$myHouse = newHouse();

# black (default value)
echo $myHouse ->primaryColor;

// change the color of the house
$myHouse ->changeColor('white');

# white
echo $myHouse ->primaryColor;

?>
```

We can create multiple objects from a class. These objects are called instances of the class. This process is called instantiation.

```
<?php
classHouse {
      public $primaryColor = 'black';
      public $secondaryColors = [
            'bathroom' =>'white',
            'bedroom' =>'light pink',
            'kitchen' =>'light blue'
      ];
      public $hasPool = false;
      public $extra;
}
$myHouse = newHouse();
$friendHouse = newHouse();
```

```
echo $myHouse ->primaryColor;
echo $friendHouse ->primaryColor;
```

PHP OOP $this Keyword

$this is a pseudo-variable (also a reserved keyword) which is only available inside methods. And, it refers to the object of the current method.

```
<?php
classHouse{
      public $name;
      public $color;
      publicfunctionsetData($name, $color){
            $this -> name = $name;
            $this -> color = $color;
      }
      publicfunctionechoData(){
            echo"The color of the {$this -> name} is {$this ->
color}";
      }
}
```

Constructors

Constructors: Constructors are called when an object is created from a class.

Destructors: Destructors are called when an object destructs. Usually, it is when the script ends.

A constructor is a public method which is named as __construct.

```
classExample{
      publicfunction__construct(){
            // your constructor function
            echo"Hello World";
      }
}
```

The __construct() method will be called once when you create an object from the class.

```
<?php
classHouse{
      public $name;
      public $color;
      publicfunction__construct($name, $color){
            $this -> name = $name;
            $this -> color = $color;
      }
      publicfunctionechoData(){
            echo"The color of the {$this -> name} is {$this ->
color}";
      }
}
$blackHouse = newHouse("John's House", "black");
$blackHouse ->echoData();
```

Destructor

A destructor is called when the object is destructed or the script is stopped or exited.

```
<?php
classHouse{
      public $name;
      public $color;
      publicfunction__construct($name, $color){
            $this -> name = $name;
            $this -> color = $color;
      }
      publicfunction__destruct(){
            echo"The color of the {$this -> name} is {$this -> color}";
      }
}
$blackHouse = newHouse("John's House", "black");
```

Types of Visibility

Visibility of a class member (property, method or constant) is where it can be accessed. (Eg: inside the class, outside the class)

- Public - Can be accessed from everywhere
- Private - Can only be accessed within the class
- Protected - Can be accessed by the class declared it and by the classes that inherit the above declared class.

Public vs Private

Consider the following example.

```
classExample{
      public $name;
      private $age;
}
$example  =new Example;
```

Here we have two properties $name and $age. We have added public and private keywords to each property respectively.

What does this do? Adding public or private will change the visibility of the property.

A public property can be accessed everywhere. (Within the class and from the outside)

```
$example -> name; // this is valid
```

A private property can only be accessed by the class that defines the property.

```
$example -> age; // this is invalid - will thow an error
```

Public Visibility

A public property or method can be declared adding the public keyword in front of its declaration. (If you followed the previous chapters, you will remember that we used this.)

```
classExample{
      public $property = 'property'; // a public property
      publicfunctionmyMethod(){ // a public method
            echo'Hello';
      }
}
```

A method declared without a visibility keyword will be public.

```
classExample{
      functionmyMethod(){
            // a public method
      }
}
```

Properties cannot be declared without a visibility keyword.

```
<?php
classUser {
      public $name = 'Hyvor';
      publicfunctionchangeName($name) {
            $this -> name = $name;
      }
}
$user = newUser();
// Accesing public property from outside
echo $user -> name;
echo'<br>'; // a line break

// Accessing public method from outside
$user ->changeName('Hyvor Developer');
echo $user -> name;
```

Private Visibility

A private property or method can be declared adding the private keyword in front of its declaration.

```
classExample {
      private $property = 'property'; // a private property
      privatefunctionprivateMethod() { // a private method
            echo$this -> property; // this is valid
      }
}
```

A private class member can only be accessed from the methods in the class.

Assume that you have a private property $name in the User class. Any method inside the class has access to this property. But, you cannot access it from outside the class. See this example:

```
classUser {
      private $name = 'Hyvor'; // a private property
      publicfunctionechoName() { // a private method
            echo$this -> name; // this is valid
      }
}
$user = newUser();
$user ->echoName(); // valid
echo $user -> name; // this will show an error
```

Protected Visibility

A protected property or method can be declared adding the protected keyword in front of its declaration.

```
classExample {
      protected $property = 'This is a protected property';
      protectedfunctionmyMethod() {
            // I'm protected!
      }
}
```

Protected properties and methods can be accessed by,

The class which declared the variable

The classes which *inherits the above declared class

```
<?php
classHouse {
      // color of the house
      private $color;
      // only these colors are allowed
      private $allowedColors = [
            'black', 'blue', 'red', 'green'
      ];
      publicfunctionsetColor($color) {
            // Black to black (lowercase)
            $color = strtolower($color);
            if ( in_array( $color, $this ->allowedColors ) ) {
                  // if $color is in the $allowedColors array
                  // we can set the color property
                  $this -> color = $color;
            }
      }
      publicfunctiongetColor() {
            if ($this -> color) {
                  // if color is set
                  return$this -> color;
            } else {
```

```
                // show an error message
                return'No color is set. May be you have set a color
which is not allowed';
            }
      }
}

// Example 1
$house1 = newHouse();
$house1 ->setColor('black');
echo $house1 ->getColor();

echo'<br>'; // a HTML line break to make it readable

$house2 = newHouse();
$house2 ->setColor('yellow'); // a not allowed color
echo $house2 ->getColor();
```

The Concept of Inheritance

The derived class is the child, and the other class which the child derived from is the parent class. (Sometimes they are called sub class and super class respectively)

In another way, the child class extends the parent class.

The child class inherits all the public and protected properties and methods from the parent class. Additionally, it can have its own properties and methods.

Note: A child class can also be inherited by another class.

In PHP, the extends keyword is used to declare an inherited class.

```
<?php
classPerson {
      public $name;
      public $age;
      publicfunction__construct($name, $age) {
            $this -> name = $name;
            $this -> age = $age;
      }
      publicfunctionintroduce() {
            echo"My name is {$this -> name}. My age is {$this ->
age}";
      }
}
/* Tom is inherited from Person */
classTomextendsPerson {
      # __construct() is inherited
      # introduce() is inherited
      publicfunctionsayHello() {
            echo"Hello, World <br>";
```

```
	}
}
$tom = newTom('Tom', 29);
$tom ->sayHello();
$tom ->introduce();
```

Note these things in the example

We inherited the Tom class from the Person class.

Even we didn't define the __construct() and introduce() methods in the Tom class, we could use it because of inheritance.

Also, we could add new methods to the Tom class (sayHello()).

$name and $age were inherited as they are public properties.

Using Protected Visibility

We discussed the public and private visibility with several examples in the visibility chapter, but protected visibility was not fully described.

The main usage of protected is in inheritance. Protected variables are only available in:

The class where it is defined

The classes which inherit the defined class

By declaring a property or method protected, you can restrict it to the above-mentioned two classes.

```
<?php
classParentClass {
	protected $protectedProperty = 'Protected';
	private $privateProperty = 'Private';
	protectedfunctionprotectedMethod() {
		echo$this ->protectedProperty;
	}
	privatefunctionprivateMethod() {
		// cannot call this function
		// from Child
	}
}
classChildextendsParentClass {
	publicfunctiondoSomething() {
		$this ->protectedMethod();
		// invalid
		// $this ->privateMethod();
	}
}
$child = newChild();
$child ->doSomething();
```

```
// all of following are invalid (only public visibility is accessible from outside)
// echo $child ->protectedProperty;
// echo $child ->privateProperty;
// $child ->protectedMethod();
// $child ->privateMethod();
```

Overriding Inherited Methods

The inherited methods can be overridden by redefining the method with the same name.

```
<?php
classPerson {
	public $name;
	public $age;
	publicfunction__construct($name, $age) {
		$this -> name = $name;
		$this -> age = $age;
	}
	publicfunctionintroduce() {
		echo"My name is {$this -> name}. My age is {$this -> age}";
	}
}
classTomextendsPerson {
	public $school;
	// overriden method
	publicfunction__construct($name, $age, $school) {
		$this -> name = $name;
		$this -> age = $age;
		$this -> school = $school;
	}
	publicfunctionintroduce() {
		echo"My name is {$this -> name}. My age is {$this -> age}. My school is {$this -> school}";
	}
}
$tom = newTom('Tom', 29, 'Foothill School');
$tom ->introduce();
```

Calling the Parent's Methods

From the child class, we can call parent's methods using the parent keyword and :: (Scope Resolution Operator).

```
<?php
classPerson {
	public $name;
	public $age;
	publicfunction__construct($name, $age) {
		$this -> name = $name;
```

```
            $this -> age = $age;
      }
      publicfunctionintroduce() {
            echo"My name is {$this -> name}. My age is {$this ->
age}";
      }
}
classTomextendsPerson {
      public $school;
      publicfunction__construct($name, $age, $school) {
            # $this -> name and $this -> age will be set by the
parent's constructor
            parent::__construct($name, $age);
            $this -> school = $school;
      }
      publicfunctionintroduce() {
            echo"My name is {$this -> name}. My age is {$this ->
age}. My school is {$this -> school}";
      }
}
$tom = newTom('Tom', 29, 'Foothill School');
$tom ->introduce();
```

Using the Final Keyword

There are two uses of the final keyword.

1. Preventing Class Inheritance

```
finalclassNonParent{
      // class code
}
// will throw an error
classChildextendsNonParent{...}
```

2. Preventing Method Overriding

```
classParentClass{
      finalpublicfunctionmyMethod(){

      }
}
classChildextendsParentClass{
      // Fatal Error: Overriding final methods
      publicfunctionmyMethod(){

      }
}
```