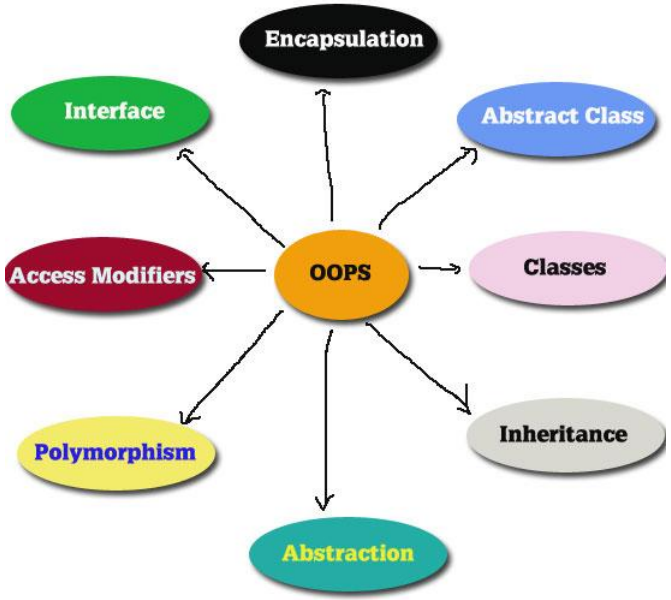
Web Development III

Block: III

Prepared By: Soniya Suthar

Block: III - Agenda

- OOP in PHP, Access Modifiers
- Class, Object
- Constructor, Destructor
- Inheritance, Traits
- Abstract, Interface, Static
- Exception Handling



OOP in PHP

- Object-Oriented Programming, also known as OOP is a special way of programming. It is considered to be more powerful and fast for certain tasks than the normal way of programming in PHP. OOP helps you to create and manage tasks easily.
- OOP is harder to understand compared to other programming techniques. But, if you understand the following 4 terms you are almost done!
    - Class
    - Object
    - Properties
    - Methods (or Functions)

What is a Class?

- A class is a blueprint. It is a piece of code describing how to manage a topic or task in the way we want.
- For instance, a class is like a blueprint of a house.You can build more than one house from a blueprint. In the same way, you can create more Objects from a class.

What is an Object?

- An Object is an instance of a class. It is like a house built from the blueprint. You can build more than one object from a class like creating multiple houses from a blueprint.

OOP in PHP

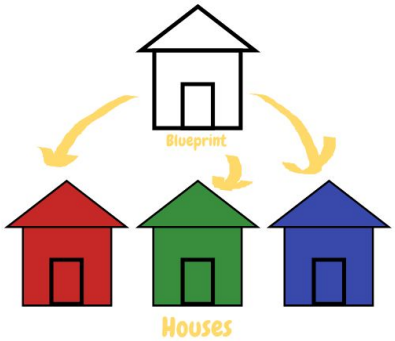
What are Properties?

- Properties are variables of an object. They are the values associated with the object. They describe the appearance of the Object.
- Properties can be added, changed, removed. Some can also be read-only.
- Color is a property of our house. We can also perform actions on our house (like changing the color).

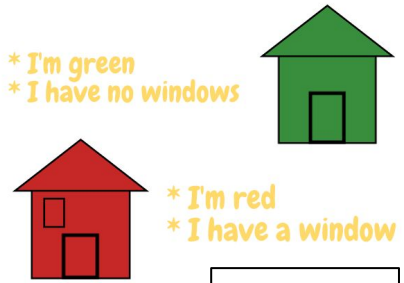
What are Methods?

- Methods are actions that are performed on Objects. Changing the color of my house is a method performed on my house object.

Class



Object



Properties

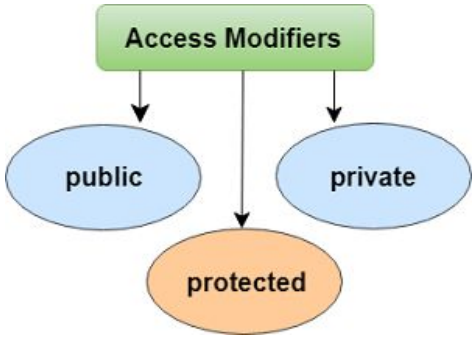
Methods

Prepared By: Soniya Suthar

PHP - Access Modifiers

- There are three access modifiers:
  - `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.

```php
<?php
	class Fruit {
	  public $name;
	  protected $color;
	  private $weight;
	}
	$mango = new Fruit();
	$mango->name = 'Mango'; // OK
	$mango->color = 'Yellow'; // ERROR
	$mango->weight = '300'; // ERROR
?>
```



Class

```php
<?php
class Fruit {
  // Properties
  public $name,$color;
  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
?>
```

Object

```php
<?php
$apple = new Fruit();
$banana = new Fruit();
$apple->set_name('Apple');
$banana->set_name('Banana');

echo $apple->get_name();
echo "<br>";
echo $banana->get_name();
?>
```

PHP OOP - Constructor & Destructor

- Constructors are special member functions for initial settings of newly created object instances from a class.
- Constructors are the very basic building blocks that define the future object and its nature. You can say that the Constructors are the blueprints for object creation providing values for member functions and member variables.
- Once the object is initialized, the constructor is automatically called. Destructors are for destroying objects and automatically called at the end of execution.
- Both are special member functions of any class with different concepts but the same name except destructors are preceded by a ~ Tilde operator.
- Syntax:
- __construct():

```
function __construct()
        {
        // initialize the object and its properties by assigning
        //values
        }
```

- __destruct():

```
function __destruct()

        {

        // destroying the object or clean up resources here

        }
```

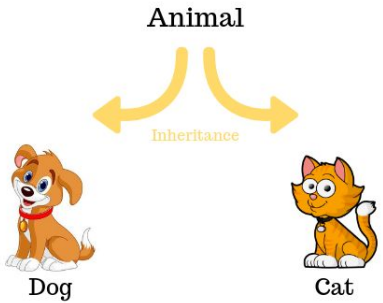
PHP OOP - Constructor & Destructor

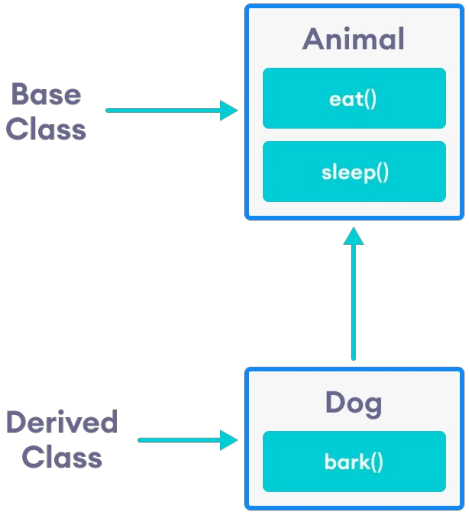
```php
<?php
      class SomeClass
     {
         function __construct()
         {
             echo "In constructor, " ;
             $this->name = "Class object! " ;
         }
         function __destruct()
         {
             echo "destroying "  . $this->name .  "\n";
         }
     }
      $obj  = new Someclass();
?>
```

PHP OOP - Inheritance

- Inheritance is one of the key features of Object-oriented programming. It allows us to create a new class (derived class) from an existing class (base class).
- The derived class inherits the features from the base class and can have additional features of its own.
- An inherited class is defined by using the `extends` keyword.

```
class Animal {

      // eat() function

      // sleep() function

}

class Dog extends Animal {

      // bark() function

}
```





PHP OOP - Inheritance

```php
<?php

class Fruit {

  public $name;

  public $color;

  public function __construct($name, $color) {

      $this->name = $name;

      $this->color = $color;

  }

  public function intro() {

    echo "The fruit is {$this->name} and the color
is {$this->color}.";

  }

}

// Mango is inherited from Fruit

class Mango extends Fruit {

  public function msg() {

    echo "Am I your favorite fruit? ";

  }

}

$Mango  = new Mango ("Mango", "Yellow");

$Mango ->msg();

$Mango ->intro();

?>
```

PHP OOP - Inheritance with Access Modifiers

```php
<?php

class Fruit {

  public $name;

  public $color;

  public function __construct($name, $color) {

      $this->name = $name;

      $this->color = $color;

  }

  protected function intro() {

    echo "The fruit is {$this->name} and the color
is {$this->color}.";

  }

}

// Mango is inherited from Fruit

class Mango extends Fruit {

  public function msg() {

    echo "Am I your favorite fruit? ";

  }

}

$Mango  = new Mango ("Mango", "Yellow");

$Mango ->msg();

$Mango ->intro(); // ERROR. intro() is protected

?>
```

PHP OOP - Final Keyword

- The `final` keyword can be used to prevent class inheritance or to prevent method overriding.

```php
<?php
final class Fruit {
  // some code
}
// will result in error
class Mango extends Fruit {
  // some code
}
?>
```

PHP OOP - Traits

- PHP only supports single inheritance: a child class can inherit only from one single parent.
- So, what if a class needs to inherit multiple behaviors? OOP traits solve this problem.
- Traits are used to declare methods that can be used in multiple classes. Traits can have methods and abstract methods that can be used in multiple classes, and the methods can have any access modifier (public, private, or protected).
- Traits are declared with the `trait` keyword.
- Syntax:

```php
<?php
trait TraitName {
  // some code...
}
?>
```

- To use a trait in a class, use the `use` keyword.
- Syntax:

```php
<?php
class MyClass {
  use TraitName;
}
?>
```

PHP OOP - Traits

```php
<?php
  trait Reader{
    public function add($a,$b){
      return $a+$b;
    }
  }
  trait writer {
    public function sub($a,$b){
      return $a-$b;
    }
  }
  class File {
    use Reader;
    use writer;
    public function calc($a,$b){
        echo "Result of addition:".$this->add($a,$b) ."";
        echo "Result of multiplication:".$this->sub($a,$b);
    }
  }
  $obj = new File();
  $obj->calc(5,3);
?>
```

PHP OOP - Abstract Classes

- Abstract classes and methods are when the parent class has a named method, but need its child class(es) to fill out the tasks.
- An abstract class is a class that contains at least one abstract method. An abstract method is a method that is declared, but not implemented in the code.
- An abstract class or method is defined with the abstract keyword:
- Syntax:

```php
<?php
abstract class ParentClass {
  abstract public function someMethod1();
  abstract public function someMethod2($name, $color);
}
?>
```

- When inheriting from an abstract class, the child class method must be defined with the same name, and the same or a less restricted access modifier.
- So, if the abstract method is defined as protected, the child class method must be defined as either protected or public, but not private.
- Also, the type and number of required arguments must be the same. However, the child classes may have optional arguments in addition.

PHP OOP - Abstract Classes

```php
<?php
	abstract class Base{
	  abstract protected function prefixName($name);
	}
	class Child extends Base {
	  public function prefixName($name) {
	    if ($name == "Sunny") {
	          $prefix = "Mr.";
	     }
	     elseif ($name == "Soniya") {
	          $prefix = "Mrs.";
	     }
	     else {
	          $prefix = "";
	     }
	     return "{$prefix} {$name}";
	     }
	}
```

```php
$class = new ChildClass;
echo
$class->prefixName("Sunny");
echo "<br>";
echo
$class->prefixName("Soniya");
?>
```

PHP OOP - Interfaces

- Interfaces allow you to specify what methods a class should implement.
- Interfaces make it easy to use a variety of different classes in the same way. When one or more classes use the same interface, it is referred to as "polymorphism".
- Interfaces are declared with the `interface` keyword
- Syntax:

```
<?php
interface InterfaceName {
  public function someMethod1();
  public function someMethod2($name, $color);
?>
```

- Interface are similar to abstract classes. The difference between interfaces and abstract classes are:
  - Interfaces cannot have properties, while abstract classes can
  - All interface methods must be public, while abstract class methods is public or protected
  - All methods in an interface are abstract, so they cannot be implemented in code and the abstract keyword is not necessary
  - Classes can implement an interface while inheriting from another class at the same time

PHP OOP - Interfaces

```php
<?php
  interface Person{
    public function learn();
  }
  class Programmer implements Person{
      public function learn(){
        echo "Learn Programming" . "\n";
    }
  }
  class Dancer implements Person{
      public function learn(){
        echo "Learn Dance" . "\n";
    }
  }
$P = new Programmer;
$D = new Dancer;
$P->learn();
$D->learn();

?>
```

PHP OOP - Static Methods

- Static methods can be called directly - without creating an instance of the class first.
- Static methods are declared with the `static` keyword.
- Static methods can also be called from methods in other classes. To do this, the static method should be public.
- To call a static method from a child class, use the `parent` keyword inside the child class.

```php
<?php
class greeting {
  public static function welcome() {
    echo "Hello World!";
  }
}
// Call static method
greeting::welcome();
?>
```

PHP OOP - Static Methods

```php
<?php
/* Counter without Static */
class solution {

    $count=0;

    public function getCount() {
        return self::$count++;
    }
}

$s = new solution();
for($i = 0; $i < 5; ++$i) {
     $s->getCount();
     echo "\n";
}
?>
```

```php
<?php
/* Use static function as a counter */
class solution {

    static $count;

    public static function getCount() {
        return self::$count++;
    }
}
solution::$count = 1;
for($i = 0; $i < 5; ++$i) {
     solution::getCount();
     echo "\n";

}
?>
```

PHP OOP - Static Properties

- Static properties can be called directly - without creating an instance of a class.
- Static properties are declared with the `static` keyword.
- A class can have both static and non-static properties. A static property can be accessed from a method in the same class using the self keyword and double colon (::).
- To call a static property from a child class, use the parent keyword inside the child class:

```php
<?php
class pi {
  public static $value = 3.14159;
}

// Get static property
echo pi::$value;
?>
```

```php
<?php
class pi {
  public static $value=3.14159;
  public function staticValue() {
    return self::$value;
  }
}

$pi = new pi();
echo $pi->staticValue();
?>
```

PHP OOP - Exceptions

- An exception is an unexpected program result that can be handled by the program itself. Exception Handling in PHP is almost similar to exception handling in all programming languages.
- PHP provides the following specialized keywords for this purpose.
  - try: It represents a block of code in which exceptions can arise.
  - catch: It represents a block of code that will be executed when a particular exception has been thrown.
  - throw: It is used to throw an exception. It is also used to list the exceptions that a function throws, but doesn't handle itself.
  - finally: It is used in place of a catch block or after a catch block basically it is put for cleanup activity in PHP code.
- Syntax:

```
try {
  code that can throw exceptions
} catch(Exception $e) {
  code that runs when an exception is caught
} finally {
  code that always runs regardless of whether an exception was caught
}
```

PHP OOP - Exceptions

```php
<?php
    function divide($a, $b) {
      if($b == 0) {
        throw new Exception("Division by zero");
      }
      return $a / $b;
    }

    try {
      echo divide(5, 0);
    }
    catch(Exception $e) {
      echo "Unable to divide. ";
    }
    finally {
      echo "Process complete.";
    }
?>
```

PHP OOP - The Exception Object

- The Exception Object contains information about the error or unexpected behaviour that the function encountered.
- Syntax:

```
new Exception(message, code, previous)
```

- Parameter Values:

Parameter	Description
message	Optional. A string describing why the exception was thrown
code	Optional. An integer that can be used used to easily distinguish this exception from others of the same type
previous	Optional. If this exception was thrown in a catch block of another exception, it is recommended to pass that exception into this parameter

- Methods:

Method	Description
getMessage()	Returns a string describing why the exception was thrown
getPrevious()	If this exception was triggered by another one, this method returns the previous exception. If not, then it returns *null*
getCode()	Returns the exception code
getFile()	Returns the full path of the file in which the exception was thrown
getLine()	Returns the line number of the line of code which threw the exception

PHP OOP - The Exception Object

```php
<?php
    function divide($dividend, $divisor) {
      if($divisor == 0) {
        throw new Exception("Division by zero", 1);
      }
      return $dividend / $divisor;
    }

    try {
      echo divide(5, 0);
    }
    catch(Exception $ex) {
      $code = $ex->getCode();
      $message = $ex->getMessage();
      $file = $ex->getFile();
      $line = $ex->getLine();
      echo "Exception thrown in $file on line $line: [Code $code]
      $message";
    }
?>
```

Prepared By: Soniya Suthar