Session 13

Methods in PHP And Other OOP Concepts

Session Overview

In this session, learners would be able to:

- Explain calling of member methods
- Outline public, private, and protected members
- Explain how to use method overriding
- Elaborate the scope of methods within a class
- Explain inheritance, interfaces, and abstract classes
- Outline static and final keyword
- Explain static method
- Describe method overloading

Calling Member Methods [1-2]

Example:

```
...
$lion->setTitle(" Alex in the Madagascar ");
$tiger->setTitle(" Joshua in the central zoo ");
$zebra->setTitle(" Marty in the central zoo ");
$lion->setQuantity(150);
$tiger->setQuantity(100);
$zebra->setQuantity(550);
```

Modifier	Methods	Variables	Classes
public	Applicable	Applicable	Not Applicable
private	Applicable	Applicable	Not Applicable
protected	Applicable	Applicable	Not Applicable
abstract	Applicable	Not Applicable	Applicable
final	Applicable	Not Applicable	Applicable

Table: List of PHP Access Modifiers

Calling Member Methods [2-2]

Code Snippet:

```
<?php
class AnimalBooks {
  // Properties
  public $title;
  public $quantity;
  // Methods
  function setTitle($title) {
    $this->title = $title;  }
  function getTitle() {
    return $this->title;  }
  function setQuantity($quantity) {
    $this->quantity = $quantity;
  }
  function getQuantity() {
    return $this->quantity;
  }
}
$lion = new AnimalBooks();
$tiger = new AnimalBooks();
$zebra = new AnimalBooks();
$lion->setTitle("Alex in the Madagascar");
$tiger->setTitle("Joshua in the central zoo");
$zebra->setTitle("Marty in the central zoo");
$lion->setQuantity(150);
$tiger->setQuantity(100);
$zebra->setQuantity(550);
$lion->getTitle();
$tiger->getTitle();
$zebra->getTitle();
```

```
$lion->getQuantity();
$tiger->getQuantity();
$zebra->getQuantity();
echo " Title: " . $lion->getTitle();
echo " Quantity: " . $lion->getQuantity();
echo "<br>";
echo " Title: " . $tiger->getTitle();
echo " Quantity: " . $tiger->getQuantity();
echo "<br>";
echo " Title: " . $zebra->getTitle();
echo " Quantity: " . $zebra->getQuantity();
echo "<br>";
?>
```

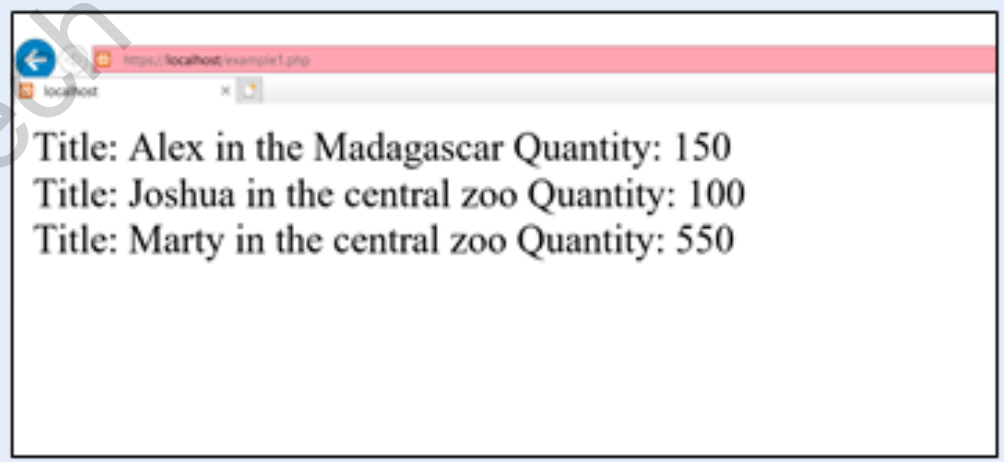

Figure: Output for Code Snippet

Inheritance [1-2]

Following are the things to remember while using inheritance:

- The child class has access to only the non-private methods and properties on the parent class.
- The methods of the child class are not available to the parent class.
- The methods defined in the parent class can be overridden by the child class with its own implementation.

Inheritance [2-2]

Example:

```
...
    class Bank {
        // parent class code
    }
     class Transaction extends Bank {
        // child class code
    }
...
```

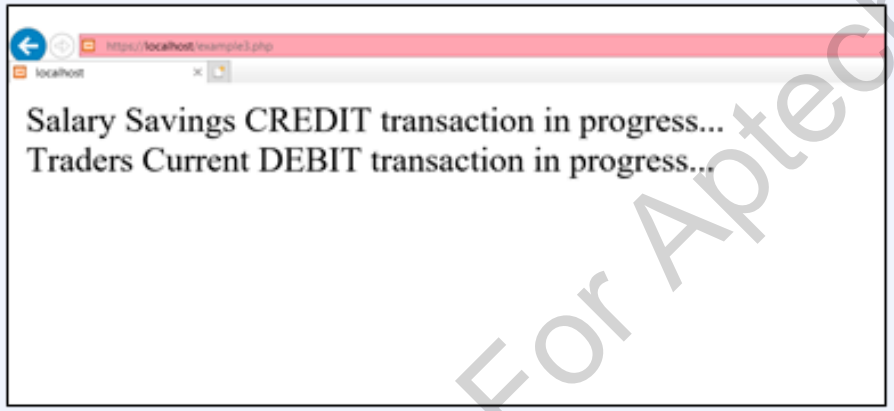

Figure: Output for Code Snippet

Code Snippet:

```
<?php
    // parent class
    class BankAccount {
        // public property type
        public $type;
     // public function credit
        public function credit() {
            echo $this->type. "  CREDIT transaction in
progress...<br/>";
        }
       // public function debit
        public function debit() {
            echo $this->type. "  DEBIT transaction in
progress...<br/>";
        }
    }
        // child class
    class Savings extends BankAccount {
        // No code in child class
    }
    // child class
    class Currentaccnt extends BankAccount {
        // No code in child class
    }
    // Instantiate the derived classes to create objects
    $savings = new Savings();
    $savings->type = "Salary Savings Account ";

    $currentaccnt = new Currentaccnt();
    $currentaccnt->type = "Trading Current Account ";

   // call class methods
    $savings->credit();
    $currentaccnt->debit();
?>
```

Child Class with its Own Methods and Properties

Code Snippet:

```
<?php
    // parent class
    class Television {
        // public property name
        public $name;

        // public function transmit_sound
        public function transmit_sound() {
        echo $this->name. " - does transmit sound
successfully <br/>";
        }
      // public function transmit_video
        public function transmit_video() {
           echo $this->name. " - does transmit video
successfully...<br/>";
        }
    }
    // child class
    class Lcdtv extends Television {

        public function control_feature() {
            echo "Manufacturer Name: " . $this->name.
"<br/>";
            echo " Our best feature: Consuming much
less
power than plasma displays<br/>";
        }
    }
```

```
// child class
    class Ledtv extends Television {
        // specific category of Television
        // and methods
public function control_feature() {
            echo "Manufacturer Name: " .
$this->name.
"<br/>";
            echo " Our best feature: The
picture quality of
an LED display is far better than an
LCD<br/>";
        }
   }
```

```
    $ledtv = new Ledtv();
    $ledtv->name = "Samsung";
    // calling parent class method
    $ledtv->transmit_sound();
    // calling child class method
    $ledtv->control_feature();
    echo "<br/>";

$lcdtv = new Lcdtv();
    $lcdtv->name = "Sony";
    // calling parent class method
    $lcdtv->transmit_sound();
    // calling child class method
    $lcdtv->control_feature();
    ?>
```

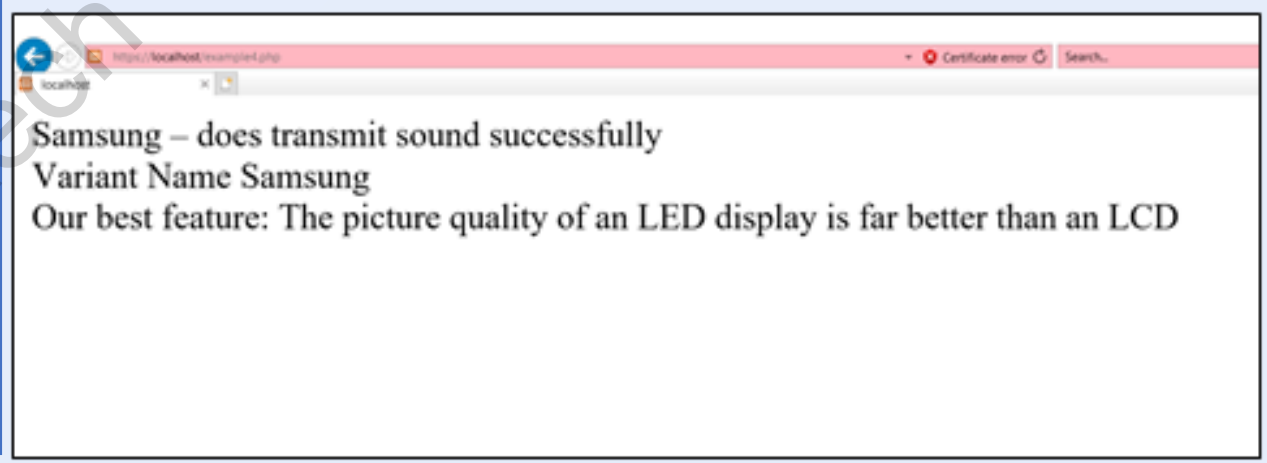

Figure: Output for Code Snippet

Public Members

By defining the classes and their members as public, they can be accessed from any of the following:

- Outside the scope of a class
- Within the scope of a class
- Another class within the declared class

Private Members

Code Snippet:

```
<?php
// Define the base class
class Footwear {
    private $price = "We have a fixed
price of 3000";
    private function show()
    {
        echo "This is a private method
of a Footwear(base class)";
    }
}

//  Define the derived classes
class Sneakers extends Footwear {
    function printPrice()
    {
        echo $this->price;
    }
}
```

```
// Create the object of the derived
class
$obj= new Sneakers;

// The given line is trying to call a
private method. Hence, trying to access
private function show() throws error

$obj->show();

// The given line also throws error
since $price is private member of
parent class Footwear
$obj->printPrice();

?>
```

Protected Members

Code Snippet:

```
<?php
// Define the base class
class Footwear {
    protected $price1 = 5000;
    protected $price2 = 15000;
    function total()    {
        echo $sum = $this->price1 + $this->price2;
        echo "<br>";

    }
}
// Define the derived classes
class Sneakers extends Footwear {
    function printBill() {
        $tax = 100;
//Calculating the total + tax
        echo $sub = $this->price1 + $this->price2
+ $tax;
        echo "<br>";
    }
}
```

```
$obj= new Sneakers;
$obj->total();
//This code accesses the function
printBill which has calculation with
protected members of base class namely
$price1 and $price2
$obj->printBill();
?>
```

Method Overriding

Example:

```
...
function getQuantity() {
    echo $this->quantity. "<br/>";
    return $this->quantity;
}
function getTitle(){
    echo $this->title. "<br/>";
    return $this->title;
}
```

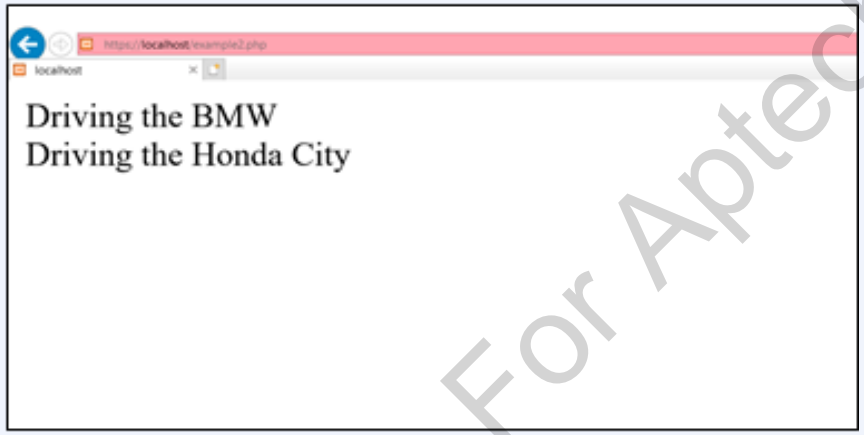

Figure: Output for Code Snippet

Code Snippet:

```
<?php
// parent class
class Car {
        // public property name
        public $title;
        // public method
        public function drive() {
            echo "Driving the Car<br/>";
        }
}
// child class
class BMW extends Car {
        public function drive() {
            echo "Driving the BMW<br/>";
        }
}
// child class
class HondaCity extends Car {
        public function drive() {
            echo "Driving the Honda City<br/>";
        }
}
$bmw = new BMW();
$bmw->title = "BMW 3 Series (G20/G21)";
     // calling child class method
    $bmw->drive();
$hondacity = new HondaCity();
    $hondacity->title = "Honda City V MT";
// calling child class method
    $hondacity->drive();

?>
```

Scope of Functions within Class

Functions are by default global. Following is the syntax to define a PHP function:

Syntax:

```
function <FunctionName>()
{
code for the function
}
```

Inner Functions

Example:

```
. . .
function Function1()
{
 echo("Function1");
 function InnerFunction2()
 {
   echo("Inner Function 2");
 }
}
```

Example:

```
function Function1()
{
 echo("My Function");
 InnerFunction2();
 if(!function_exists("InnerFunction2")){
   function InnerFunction2()
   {
    echo("InnerFunction2 ");
   }
 }
}
```

Example:

```
. . .
function Function1()
{
 echo("Function1");
 if(!function_exists("InnerFunction2")){
  function InnerFunction2()
  {
   echo("Inner Function 2");
  }
 }
}
```

Interfaces [1-2]

Following is the syntax for defining a PHP interface:

Syntax:

```
<?php
    // interface declaration
    interface NameOfInterface {
  }
 ?>
```

Interfaces [2-2]

Example:

```
. . .
    // interface declaration
    interface ClothingApp {
     // methods declaration
        public function login($phone,
$password);
public function register($phone, $password,
$username);
        public function logout();
        }
```

Example:

```
. . .
    // class declaration
    class BestClothing implements ClothingApp
{
  // methods definition
        public function login($mobile,
$password) {
     echo "Login the Customer with phone
number: " .
$mobile;
        }
        public function register($mobile,
$password,
$username) {
            echo "Customer registered: Phone
Number=".$mobile." and Username=".$username;
        }
        public function logout() {
            echo "Customer logged out!";
        }
}
```

Implementing Multiple Interfaces

Example:

```
// interface declaration
interface Reward {
     // methods declaration
        public function earnReward($post);
}
```

Example:

```
// class declaration
class BestClothing implements ClothingApp,
Reward {
          // methods definition
        public function login($mobile,
$password) {
            echo "Login with phone number: " .
$mobile;
        }
         public function register($mobile,
$password, $username) {
            echo "Customer registered: Phone
Number
=".$mobile." and Username=".$username;
        }
          public function logout() {
            echo " Customer logged out!";
        }
        public function earnReward ($post) {
            echo $post." rewards earned!";
        }
}
```

Abstract Classes [1-2]

Following are some important points to know about abstract classes and methods:

- Abstract classes can have methods and properties similar to other classes. However, they cannot be instantiated.
- A child class has to be created to instantiate an abstract class.
- The class should be abstract to have an abstract method.
- Abstract method is a declaration with an empty body. The name of the method and argument is provided by the user.

Abstract Classes [2-2]

Following is an example to show definition of an abstract class:

Example:

```
. . .
    // abstract class
    abstract class Microwave {

        // abstract function bake, with no implementation
        abstract public function bake();
    }
```

Non-Abstract Method in Abstract Class

Following example shows how to include the non-abstract method in the Microwave class defined earlier:

Example:

```
. . .
    // abstract class
    abstract class Microwave {
            // protected variable
        protected $degree;
    // non-abstract public function start
        public function start() {
            echo $this->degree. " - Microwave start...<br/>";
        }
        // non-abstract public function stop
        public function stop() {
            echo $this->degree. " - Microwave stop...<br/>";
        }
        // non-abstract public function setDegree
        public function setDegree ($degree) {
            $this->degree= $degree;
        }
        // abstract function bake, with no implementation
        abstract public function bake ();
    }
```

Inheriting Abstract Classes [1-2]

Example:

```
. . .
    // child class
    class Onida extends Microwave {
           public function bake() {
            echo "This version baking
temperature is  - 180
to 300 degrees celsius";
        }
    }
```

Example:

```
. . .
// child class
    class Samsung extends Microwave {
      public function bake() {
       echo " This version baking temperature
is  -
200 to 400 degrees celsius ";
        }
}
```

Example:

```
. . .
    $samsung = new Samsung ();
    $samsung ->setDegree("300");
    $samsung ->bake();
```

Inheriting Abstract Classes [2-2]

Code Snippet:

```
<?php
    // abstract class
    abstract class Microwave {
     // protected variable
        protected $degree;
    // non-abstract public function start
        public function start() {
            echo $this->degree. " - Microwave
start...<br/>";
        }
   // non-abstract public function stop
        public function stop() {
            echo $this->degree. " - Microwave
stop...<br/>";
        }
   // non-abstract public function setDegree
        public function setDegree($degree) {
            $this->degree = $degree;
        }
        // abstract function bake
        abstract public function bake() ;
    }
```

```
// child class
    class Samsung extends Microwave {
       public function bake() {
           echo "This version baking
temperature is  - 200 to
400 degrees celsius ";
        }
      }
$samsung = new Samsung();
     $samsung->setDegree("400");
     $samsung->bake();
   ?>
```

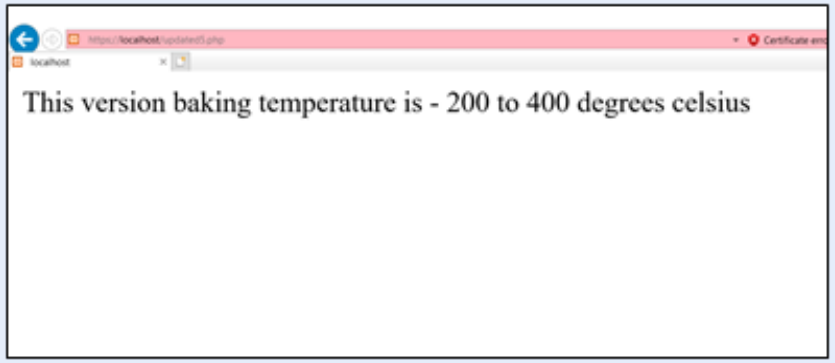

Figure: Output for Code Snippet

Interface vs Abstract Class

Following are some differences between an interface and an abstract class:

While an interface cannot have concrete methods and have only abstract methods, an abstract class can have both.

In an interface, the user can only declare the methods as public. However, in an abstract class, the user can declare methods as public, private, or protected.

In a class, though multiple interfaces can be implemented, only one abstract class can be implemented.

final Keyword

Code Snippet:

```
<?php
   class Bill {
      public function check() {
         echo "Bill::check() called<br>";
      }
      final public function applytax() {
         echo "Bill::applyingtax()
called<br>";
      }
   }
   class  RentBill extends Bill {
      public function applytax() {
         echo "ChildClass::applyingtax()
called<br>";
      }
   }
    $rentBill = new RentBill();
    $rentBill->applytax();
?>
```

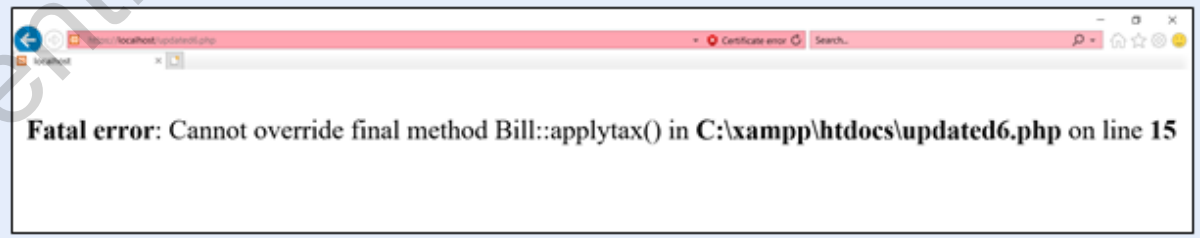

Figure: Output for Code Snippet

Static Members [1-2]

Following is the syntax to create a static method:

Syntax:

```
<?php
class ClassName {
  public static function <staticMethod>() {
  <code>
  }
}
?>
```

To access a static method use the class name, a double colon (::), and the method name. Following is the syntax:

Syntax:

```
ClassName::staticMethod();
```

Static Members [2-2]

Example:

```
. . .
   class Test {
      public static $my_static =
"test";
      public function staticValue() {
         return self::$my_static;
      }
   }
   print Test::$my_static. "<br>";
   $test = new Test();
   print $test->staticValue() . "<br>";
?>
```

Figure: Output for Code Snippet

Code Snippet:

```
<!DOCTYPE html>
<html>
<body>
<?php
class greeting {
  public static function welcome() {
     echo "Warm wishes for the season..";
  }
}
// Call static method
greeting::welcome();
?>
</body>
</html>
```

Method Overloading

Following example shows method overloading:

Example:

```
...
class Model {
        public function __call($name, $arguments) {
        echo "This is object method '$name' "
            . implode(', ', $arguments). "<br>";
    }
    public static function __callStatic($name, $arguments) {
        echo "This is static method '$name' "
            . implode(', ', $arguments). "<br>";
    }
}
 // Create new object
$obj = new Model;
$obj->runModel("Object context");
Model::runModel("Static context");
```

Property Overloading [1-2]

Following are some functions used in property overloading:

`__set()`	• To initialize overloaded properties
`__get()`	• To read data from inaccessible properties
`__isset()`	• To check the overloaded properties
`__unset()`	• To be invoked when used for overloaded properties

Property Overloading [2-2]

Following example shows property overloading:

Example:

```
...
//
class Model {
    private $data = array();
    public $declared = 1;
    // Overloading used when accessed outside the class
    private $hidden = 2;
    // Function definition
    public function __set($name, $value) {
        echo "Set members '$name' to '$value'<br>";
        $this->data[$name] = $value;
    }
    // Function definition
    public function __get($name) {
        echo "Get members '$name: ";
        if (array_key_exists($name, $this->data)) {
            return $this->data[$name];
        }
        $trace = debug_backtrace();
        return null;
    }
    // Function definition
    public function __isset($name) {
        echo "Is '$name' trying to set?<br>";
        return isset($this->data[$name]);
    }
    // Definition of __unset function
    public function __unset($name) {
        echo "Trying to Unset '$name'<br>";
        unset($this->data[$name]);
    }
    // getHidden function definition
    public function getHidden() {
        return $this->hidden;
    }
}
// Create an object
$obj = new Model;
// Set value 1 to the object variable
$obj->a = 1;
echo $obj->a . "<br>";
// Use isset function to check 'a' is set or not var_dump(isset($obj->a));
// Unset 'a'
unset($obj->a);
var_dump(isset($obj->a));
echo $obj->declared. "<br><br>";
echo "Private properties are allowed to access inside the class ";
echo $obj->getHidden(). "<br><br>";
echo "Private properties are not allowed to access outside of class<br>";
echo $obj->hidden. "<br>";
```

Magic Methods

`__set()` is triggered when overloaded properties are initialized.

`__get()` is triggered when overloaded properties are used with PHP print statements.

`__isset()` is a magic method that is invoked when overloaded properties are checked with the `isset()` function.

`__unset()` is a function that is invoked on using PHP `unset()` for overloaded properties.

Summary

- Variables and member methods of a class can be accessed through an object of the class by using the → operator.
- Access modifiers are keywords that decide the accessibility for various class methods, variables, and other member methods.
- Inheritance allows users to reduce code duplication by creating a new class with properties and methods inherited from other classes. The child class inherits the properties and methods of the parent class.
- An interface can be considered a blueprint of a class and allows the user to specify what all methods and properties a class should implement.
- Abstract class is a class that defines a structure for other classes to extend. An abstract class cannot be instantiated, has at least one abstract method, and can contain method definitions.
- Overloading is an important feature in PHP that allows users to dynamically create and use methods and properties.