Introduction to Web Development (4340704)

Computer Department

AVPTI Rajkot

Unit – III
Object Oriented Concepts in PHP

Unit Outcomes

- Define class, object, constructor and destructor for a given problem.
- Implement Inheritance to extend the base class.
- Use polymorphism to solve the given problem.
- Clone the given object.

OOP

- OOP stands for Object-Oriented Programming.
- Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.
- 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

PHP with OOP

- PHP is a server-side scripting language, mainly used for web development and also used as a general-purpose programming language.
- Object-Oriented Programming concept is added to php5, that helps in building complex, reusable web applications.
- OOP in PHP provides a number of benefits over traditional procedural programming. It allows for modularity, code reuse, encapsulation, inheritance, polymorphism, and better error handling.
- By using OOP, developers can create more organized, maintainable, and scalable web applications.

The Object Oriented concepts in PHP

- **Class** – This is a programmer-defined data type, which includes local functions as well as local data. You can think of a class as a template for making many instances of the same kind (or class) of object.
- **Object** – An individual instance of the data structure defined by a class. You define a class once and then make many objects that belong to it. Objects are also known as instance.
- **Inheritance** – When a class is defined by inheriting existing function of a parent class then it is called inheritance. Here child class will inherit all or few member functions and variables of a parent class.
- **Polymorphism** – This is an object oriented concept where same function can be used for different purposes. For example function name will remain same but it make take different number of arguments and can do different task.

Continue…

- **Overloading** – a type of polymorphism in which some or all of operators have different implementations depending on the types of their arguments. Similarly functions can also be overloaded with different implementation.
- **Data Abstraction** – Any representation of data in which the implementation details are hidden (abstracted).
- **Encapsulation** – refers to a concept where we encapsulate all the data and member functions together to form an object.
- **Constructor** – refers to a special type of function which will be called automatically whenever there is an object formation from a class.
- **Destructor** – refers to a special type of function which will be called automatically whenever an object is deleted or goes out of scope.

Class,Object,Properties&Methods

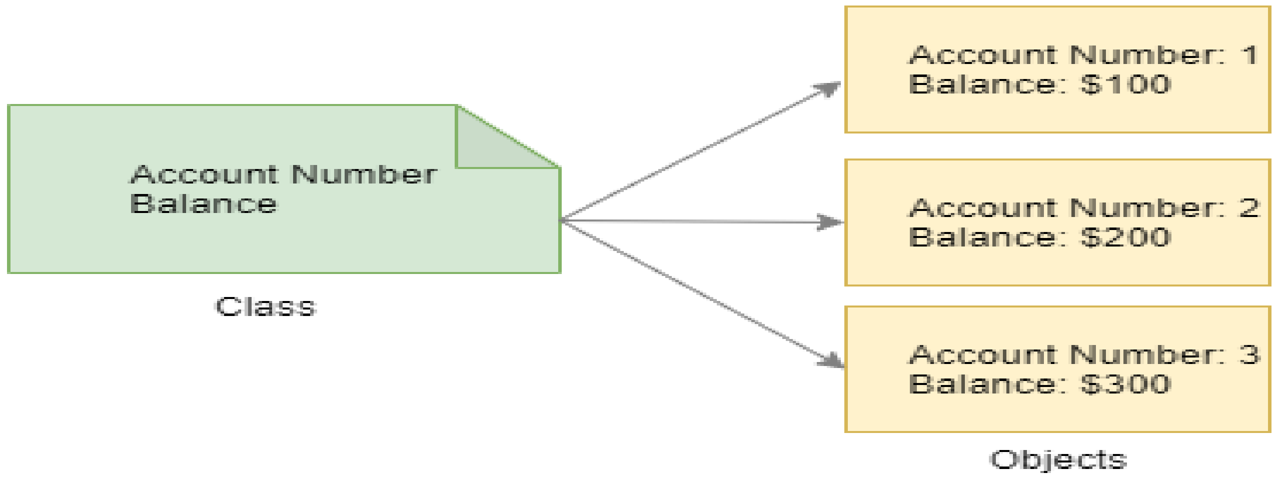
- In the real world, you can find many same kinds of objects. For example, a bank has many bank accounts. All of them have account numbers and balances.
- These bank accounts are created from the same blueprint. In object-oriented terms, we say that an individual bank account is an **Object(instance**) of a Bank Account class.
- By definition, a **class** is the blueprint of objects. For example, from the Bank Account class, you can create many bank account objects.

Continue…

- All the objects share the two common key characteristics:
- State
- Behavior
- For example, a bank account has the state that consists of:
- Account number
- Balance
- A bank account also has the following behaviors:
- Deposit
- Withdraw
- An object holds its state in variables that are often referred to as **properties.** An object also exposes its behavior via functions which are known as **methods.**

Continue…

- The following illustrates the relationship between the **Bank Account class** and its objects. From the Bank Account class you can create many Bank Account objects. And each object has its own account number and balance.



Define a class & Object

-
  ```
  <?php
  class MyClass
      {
          // Class properties and methods go here
      }
  ?>
  ```

-
  ```
  <?php
  class MyClass
  {
          // Class properties and methods go here
  }
  $obj = new MyClass;  //define object , new keyword to
                                  create an object from a class.
  var_dump($obj);  // display the structured information
                             (type and value) about one or more
                              variables.
  ?>
  ```

Example…

```
<?php
class BankAccount
{           public $accountNumber;
            public $balance;
            public function deposit($amount)
            {          if ($amount > 0) {
                                $this->balance += $amount;
                                echo "$this->balance"."</br>";
            }             }
            public function withdraw($amount)
            {             if ($amount <= $this->balance) {
                                $this->balance -= $amount;
                                echo $this->balance; }   }
}
$account = new BankAccount();
$account->accountNumber = 1;
$account->balance = 100;
$account->deposit(100);  // Output is 200
$account->withdraw(200); // Output is 0
 var_dump($account) ;   ?>   // Output is… object(BankAccount)#1 (2)
                                 {  ["accountNumber"]=> int(1)  ["balance"]=> int(0) }
```

$this in PHP

- In PHP $this keyword references the current object of the class.
- The $this keyword allows you to access the properties and methods of the current object within the class using the object operator ->

$this->property

$this->method()

- The $this keyword is only available within a class. It doesn't exist outside of the class.
- When you access an object property using the $this keyword, use the $ with the this keyword only and don't use the $ with the property name.
- **$this->balance**

Access Modifiers

- There are three access modifiers(specifier) in PHP
   - Public
   - Protected
   - Private
- **Public:** Members of the class declared as public are accessible everywhere.
- **Protected:** Members of the class declared as protected are accessible only within the base class and the derived class which extends the base class.
- **Private:** Members of the class declared as private are accessible with the class that defines it.

Constructor

- A constructor allows you to initialize an object's properties upon creation of the object.
- Constructors are special member functions for initially assign value to newly created object instances from a class.
- PHP will automatically call constructor function when you create an object from a class.
- The constructor is defined in the public section of the Class.
- You can create a constructor in PHP using *__construct()*

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

Types of Constructor in PHP

- **Default Constructor**: It doesn't take any parameters, When an object of the class is created, this default constructor is called by default.
- **Parameterized Constructor**: It takes the parameters, and also you can pass different values to the data members. It allows you to pass values during object creation and use those values to set properties.
- **Copy Constructor:** It accepts the address of the other objects as a parameter.

Default Constructor

```
<?PHP
class Stock
{
      function Stock()
      {
          echo "Its a User-defined Constructor of the class Stock";
      }

      function __construct()
      {
         echo "Its a Pre-defined Constructor of the class Stock";
      }
}

$obj= new Stock();
//Output: Its a Pre-defined Constructor of the class Stock
?>
```

Parameterized Constructor

```
<?php
class Employee
{       Public $name;
        Public $address;
        function __construct($name,$address)  {
                        // This is initializing the class properties
                        $this->name=$name;
                        $this->address=$address; }
            function show_details() {
                        echo $this->name;
                        echo $this->address; }
}
$employee_obj= new Employee("Raj","Baroda");
$employee_obj->show_details();
$employee2= new Employee("Ajay","Rajkot");
$employee2->show_details(); ?>
```

Copy of object with clone

- The **clone** keyword is used to create a copy of an object.
- $copy_object_name = **clone** $object_to_be_copied

```
<?php
class MyClass {
  public $college;
  public $branch;
}
$S1 = new MyClass();
$S1->college = 'AVPTI';
$S1->branch = 'COMPUTER';
$S2 = clone $S1;
print_r($S1);
echo "</br>";
print_r($S2);
?>
```

Output:

```
MyClass Object ( [college] => AVPTI [branch] => COMPUTER )
MyClass Object ( [college] => AVPTI [branch] => COMPUTER )
```

Continue…

```
<?php
class Person1
{
public $name;
public $gender;
public function __construct($name,$gender){
$this->name = "$name";
$this->gender = $gender;
}

}
$perObj2 = new Person1("Raj","Male");
$perObj3 = clone $perObj2;
echo "<br/>".$perObj3->name;
echo "<br/>".$perObj3->gender;
?>
```

Destructor

- Destructor is also a special member function which is exactly the reverse of constructor method and is called when an instance of the class is deleted from the memory.
- They don’t have any types or return value.
- PHP will automatically call the function destructor at the end of the script.
- You can create a destructor in PHP using *__ destruct()*

```
function __ destruct()
{
 // destroying the object
}
```

Example…

```
<?php
class class1
   {
      function __construct()
      {
         echo "In constructor, "."</br>";
         $this->name = "Class object! ";
      }
      function __destruct()
      {
         echo "destroying " . $this->name . "\n";
      }
   }
$obj = new class1();
?>
```

Output:

```
In constructor,
destroying Class object!
```

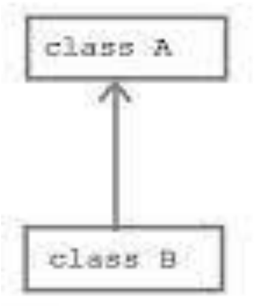
Inheritance

- A class should be closed for modification but should be open for extension.
- We should not modify any class. We can add additional features to an existing class without modifying it.
- Inheritance is one concept to achieve this.
- Inheritance is a concept of inheriting the properties of one class (super class) into another class (sub class).
- It is a process of reusing existing class functionality in new class.
- The existing class is called super class/parent class/base class and the new class is called subclass/child class/derived class.
- All the members of super class will be inherited into subclass including __construct and __destruct functions (except private members) and can be accessed directly in subclass.
- Super classes are generalized classes, subclasses are specialized classes.
- The idea behind using inheritance is all about better management of the code and the code reusability.

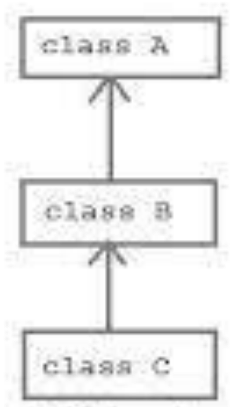
Types of Inheritance in PHP

- **Single Inheritance :** It is the most basic and simple inheritance type. In a single inheritance, there is only one base class and one sub or derived class. It directly inherits the subclass from the base class.
- **Multilevel inheritance :** In this type of inheritance, we will have more than 2 classes. In this type of inheritance, a parent class will be inherited by a child class then that child class will be inherited by the another child class.
- **Hierarchical inheritance** : It is the type of inheritance in which a program consists of a single parent and more than one child class.

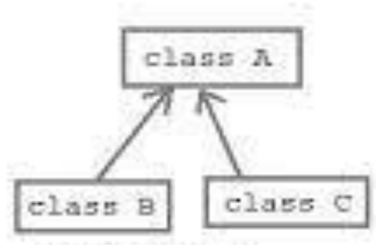
```
class bclass
{
    //Members of Base class
}

class dclass extends bclass
{
    //Members of deriveclass
}
```



Simple Inheritance



Multilevel inheritance



Heirarchical inheritance

Single Inheritance Example

```
<?php
   class demo
   {
       public function display()
       {
           echo "example of inheritance  ";
       }
   }
   class demo1 extends demo
   {
       public function view()
       {
           echo "in php";
       }
   }
   $obj= new demo1();
   $obj->display();
   $obj->view();    // o/p is  : example of inheritance in php
?>
```

Multilevel Inheritance Example

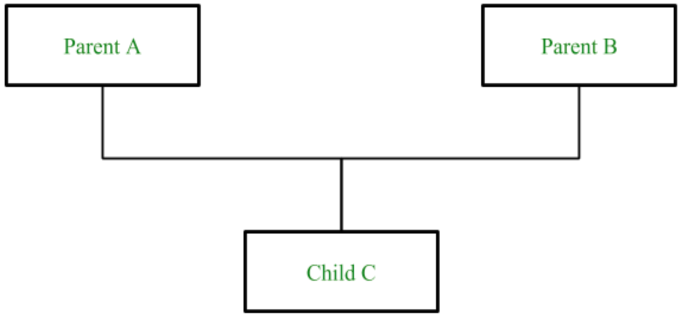
```
<?php
class Base
{       function BaseFun() {
        echo "BaseFun() called<br>"; }
}
class Derived1 extends Base
{       function Derived1Fun() {
        echo "Derived1Fun() called<br>";  }
}
class Derived2 extends Derived1
{       function Derived2Fun() {
        echo "Derived2Fun() called<br>"; }
}
$dObj = new Derived2();
$dObj->BaseFun();                       // o/p is :  BaseFun() called
$dObj->Derived1Fun();                                Derived1Fun() called
$dObj->Derived2Fun();   ?>                           Derived2Fun() called
```

Hierarchical Inheritance Example

```
<?php
class Base
{       function BaseFun()  {
        echo "BaseFun() called<br>";  }
}
class Derived1 extends Base
{     function Derived1Fun() {
        echo "Derived1Fun() called<br>";  }
}
class Derived2 extends Base
{      function Derived2Fun() {
        echo "Derived2Fun() called<br>";   }
}
$Obj1 = new Derived1();
$Obj2 = new Derived2();
$Obj1->BaseFun();                        // o/p is :  BaseFun() called
$Obj1->Derived1Fun();                                 Derived1Fun() called
echo "<br>";
$Obj2->BaseFun();                                     BaseFun() called
$Obj2->Derived2Fun();                                 Derived2Fun() called
?>
```

Multiple Inheritance in PHP

- Multiple Inheritance is the property of the Object Oriented Programming languages in which child class or sub class can inherit the properties of the multiple parent classes or super classes.
- PHP doesn’t support multiple inheritance but by using **Interfaces** in PHP or using **Traits (type of class which enables multiple inheritance)** in PHP instead of classes, we can implement it.



Interface

- Interfaces allow you to specify what methods a class should implement.
- A PHP interface defines a contract which a class must fulfill. If a PHP class is a blueprint for objects, an interface is a blueprint for classes.
- An Interface allows the users to create programs, specifying the public methods that a class must implement, without involving the complexities and details of how the particular methods are implemented.
- It is generally referred to as the next level of abstraction.

Continue…

- An Interface is defined just like a class is defined but with the class keyword replaced by the interface keyword and just the function prototypes. The interface contains no data variables.
- define an interface using the *interface* keyword.

```
<?php
interface MyInterfaceName {
   public  function methodA();
   public  function methodB();
}
?>
```

Continue…

- To implement an interface,
  use the **implements** operator as follows:

```
<?php
class MyClassName implements MyInterfaceName
{
public function methodA() {
// method A implementation
}
public function methodB(){
// method B implementation
}
}
?>
```

Characteristics of an Interface

- interface consists of methods that have no implementations, which means the interface methods are abstract methods.
- All the methods in interfaces must have public visibility scope.
- Interfaces are different from classes as the class can inherit from one class only whereas the class can implement one or more interfaces.
- An interface can model multiple inheritances because a class can implement more than one interface whereas it can extend only one class.
- An interface allows unrelated classes to implement the same set of methods, regardless of their positions in the class inheritance hierarchy.

Example (2 interface 1 class)

```
<?php
    interface a
    {  public function fun1();  }
    interface b
    {  public function fun2();  }
class demo implements a,b
{
    public function  fun1()
    {  echo "method 1...";  }
    public function  fun2()
    {  echo "method2...";  }
}
$obj= new demo();
$obj-> fun1();            // method 1...
$obj-> fun2();           // method 2...
?>
```

Example (1 interface 2 class)

```
<?php
interface Animal {
  public function makeSound(); }
class Cat implements Animal
 {
      public function makeSound() {
      echo " Meow ";  }
 }
class Dog implements Animal
 {
    public function makeSound() {
     echo " Bark ";  }
}
$cat = new Cat();
$dog = new Dog();
$cat->makeSound();    // Meow
$dog->makeSound();   // Bark
?>
```

Abstract Class

- 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 can contain abstract as well as non abstract methods.
- An abstract class or method is defined with the abstract keyword.
- Instance of abstract class can not be created.
- abstract class in PHP can contain constructor also

Continue...

```
<?php
abstract class base
{    // This is abstract function
    abstract function printdata();
    // This is not abstract function
    function pr()
    {  echo "Base class";  }
} ?>
<?php
abstract class Base
{   function __construct()
    {    echo "this is abstract class constructor ";  }
    abstract function printdata();
} ?>
```

Example...

```
<?php
abstract class Base
{
    abstract function printdata();
}
class Derived extends base
{
    function printdata()
    {
        echo "Derived class";
    }
}
 // $b = new Base();   // give an error as you can not create
                          instance of abstract class
$b1 = new Derived;
$b1->printdata();   // Derived Class
?>
```

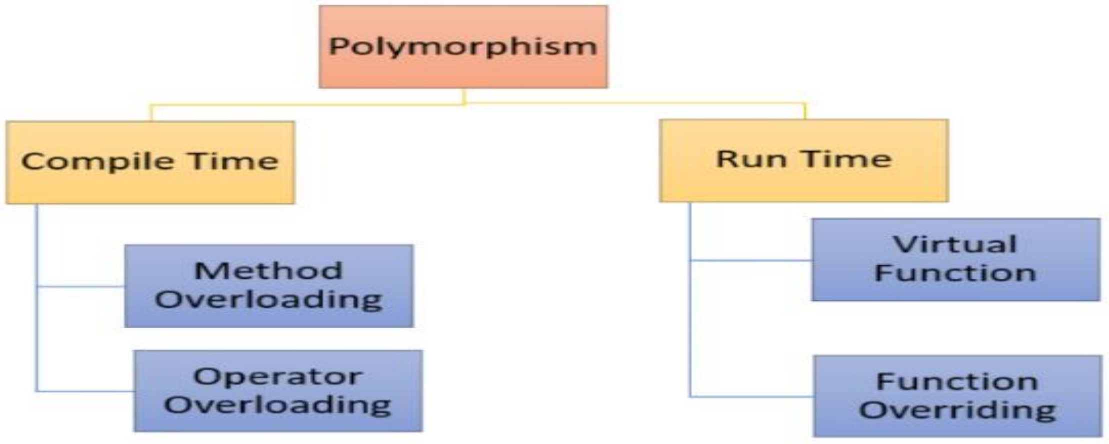
Interface vs Abstract class in PHP

- 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, and the abstract keyword is not necessary.
- Abstract classes can include both abstract methods (without implementations) and regular methods (with implementations).
- A class can implement multiple interfaces.
- A class can only extend one parent class (abstract or concrete).
- Interfaces cannot have constructors.
- Abstract class can contain constructor.

Polymorphism

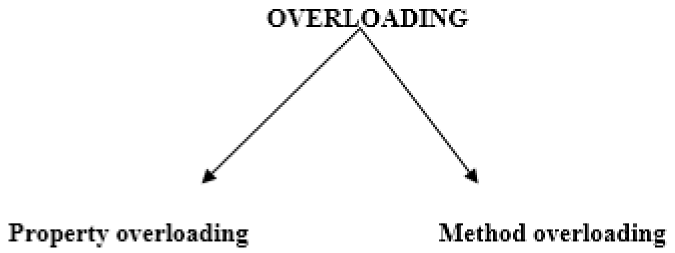
- Polymorphism is the process of using a function or an operator for more than one purpose. In other words, we can also say that polymorphism refers to the fact that the same entity can serve us in different ways in different scenarios.
- The word "polymorphism" is derived from two words "poly" and "morphism" which means multiple forms or multiple states.
- We can understand polymorphism with the help of an example, consider the '+' operator, which can be used to add two integers (6 + 4 = 10), and we can also use the same operator to concatenate two strings ("Hello" + "Boss!" = "Hello Boss!").
- Here, the '+' operator is behaving differently in two different situations and is serving two different purposes of adding and concatenation.

Types of Polymorphism



Polymorphism in PHP

- Compile time polymorphism, which is also known as function overloading



- Run time polymorphism, which is also called function overriding

Function (Method) Overloading

- Method overloading is a concept that allows you to have a method that can perform differently based on its number of parameters.
- It allows you have multiple definitions for a same method in the same class.
- This method will have the same name for all its uses, but might produce different output in different situations.

Traditional method Overloading

- For example, you have an add() that you want to use to calculate sum of two digits and three digits.

```
function add(int $a, int $b)
        {
                return $a + $b;
        }

        function add(int $a, int $b, int $c)
        {
                return $a + $b + $c ;
        }
```

The first definition takes two number and give sum while second definition takes three number and give sum.

Method Overloading in PHP

- In PHP you can not redefine a method multiple times like this.

```
<?php
class SampleClass
{
        function add(int $a, int $b)
        {
                return $a + $b;
        }

        function add(int $a, int $b, int $c)
        {
                return $a + $b + $c ;
        }
}
$sampleObject = new SampleClass;
echo $sampleObject->add(12, 12);
echo $sampleObject->add(12, 2, 6);
?>
```

The above code will give an error **Cannot redeclare SampleClass::add().**

Continue…

- PHP supports method overloading using a magic keyword __call().
- __call() is a magic method that PHP calls when it tries to execute a method of a class and it doesn't find it.
- This magic keyword takes in two arguments: a function name and other arguments to be passed into the function. Its definition looks like this:
- **function __call(string $function_name, array $arguments) {}**
- Using this magic method, you can create as many methods and as many variations of each of these methods as you like.
- Overloading in PHP provides means to **dynamically create properties and methods.** These dynamic entities are **processed via magic methods** one can establish in a class for various action types.

Example1…

```
<?php
class SampleClass
{         function __call($function_name, $arguments)
          {           $count = count($arguments);
                      // Check function name
                      if ($function_name == 'add') {
                                  if ($count == 2) {
                                  return array_sum($arguments);
                                  } else if ($count == 3) {
                                  return array_sum($arguments); } }
          }
}
$sampleObject = new SampleClass;
echo $sampleObject->add(12, 12);   // 24
echo "</br>";
echo $sampleObject->add(12, 2, 6);  // 20
?>
```

Example2…

```
<?php
  class Shape {
  function __call($name,$arg){
        if($name == 'area')
           switch(count($arg)){
              case 0 : return 0 ;
              case 1 : return 3.142 * $arg[0]* $arg[0];
              case 2 : return $arg[0] * $arg[1];
           }
     }
  }
  $circle = new Shape();
  echo $circle->area(3);    //28.278
  echo "</br>";
  $rect = new Shape();
  echo $rect->area(8,6);     //48
?>
```

Property Overloading

- PHP property overloading is used to create dynamic properties in the object context.
- A property associated with a class instance, and if it is not declared within the scope of the class, it is considered as overloaded property.
- Following operations are performed with overloaded properties in PHP.
- Setting and getting overloaded properties.
- Evaluating overloaded properties setting.
- Undo such properties setting.
- Before performing the operations, we should define appropriate magic methods. which are, **__set() , __get(),__isset(),__unset().**

Example…

```
<?php
class Toys
{    private $str;
     public function __set($name, $value)
    {  $this->str[$name] = $value;  }
    public function __get($name)
    {  echo "Overloaded Property name = " . $this->str[$name] . "<br/>";}

    public function __isset($name)
    {
        if (isset($this->str[$name])) {
            echo "Property \$$name is set.<br/>";
        } else {  echo "Property \$$name is not set.<br/>"; }  }
   public function __unset($name)
    {   unset($this->str[$name]);
        echo "\$$name is unset <br/>“; }
}
```

Continue…

```
$objToys = new Toys();

/* setters and getters on dynamic properties */
$objToys->overloaded_property = "new";
echo $objToys->overloaded_property . "\n\n";

/* Operations with dynamic properties values */
isset($objToys->overloaded_property);
unset($objToys->overloaded_property);
isset($objToys->overloaded_property);
?>
```

Output is :

```
Overloaded Property name = new
Property $overloaded_property is set.
$overloaded_property is unset
Property $overloaded_property is not set.
```

Function Overriding

- Method overriding allows a child class to provide a specific implementation of a method already provided by its parent class.
- To override a method, you redefine that method in the child class with the same name, parameters, and return type.
- The method in the parent class is called **overridden method,** while the method in the child class is known as the **overriding method**.
- The code in the overriding method overrides (or replaces) the code in the overridden method.
- PHP will decide which method (overridden or overriding method) to call based on the object used to invoke the method.
- If an object of the parent class invokes the method, PHP will execute the overridden method.
- But if an object of the child class invokes the method, PHP will execute the overriding method.

Example...

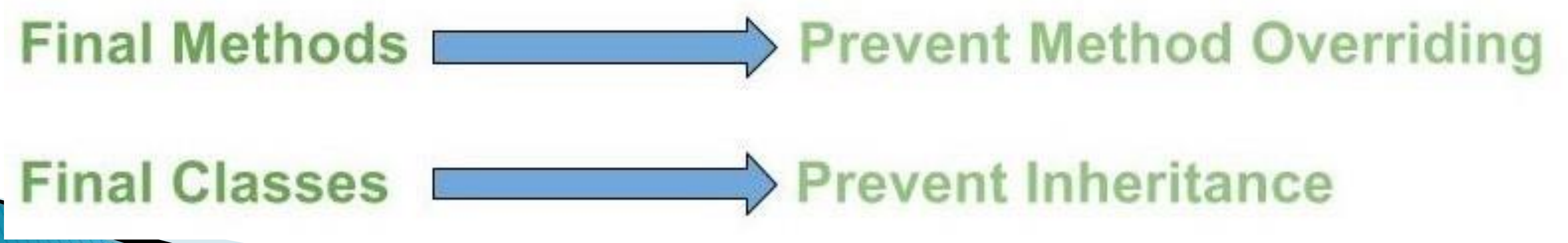
```
<?php
class Robot
{           public function greet()
          {   return 'Hello!'; }
}
class Android extends Robot
{          public function greet()
          {    return 'Hi';     }
}
$robot = new Robot();
echo $robot->greet(); // Hello
$android = new Android();
echo $android->greet(); // Hi!
?>
```

Function Overloading vs Function Overriding

Function Overloading	Function Overriding
Function Overloading provides multiple definitions of the function by changing signature.	Function Overriding is the redefinition of base class function in its derived class with same signature.
An example of compile time polymorphism.	An example of run time polymorphism.
Function signatures should be different.	Function signatures should be the same.
Overloading is used when the same function has to behave differently depending upon parameters passed to them.	Overriding is needed when derived class function has to do some different job than the base class function.
A function has the ability to load multiple times.	A function can be overridden only a single time.
In function overloading, we don’t need inheritance.	In function overriding, we need an inheritance concept.

final keyword in php

- Final keyword in PHP is used in different context. The final keyword is used only for methods and classes.
- **Final methods:** When a method is declared as final then overriding on that method can not be performed.
- **Final Classes:** A class declared as final can not be extended in future. A final class can contain final as well as non final methods.



Method with final

```
<?php
    class base
    {
        final public function fun1()
        {
            echo "Base class..";
        }
    }
    class derived extends base
    {
        public function fun1()
        {
            echo "derived class";
        }
    }
    $obj = new derived();
    $obj->fun1();  // error…can not override final method
?>
```

Class with final

```
<?php
final class Base
 {
    final function printdata() {
        echo "final base class final method";
    }
    function nonfinal() {
        echo "\nnon final method of final base class";
    }
}
class Derived extends Base
{

}
?>   // Error .... Class Derived may not inherit from final
                    class (Base)
```

Example…

```
<?php
class Base
 {         final function printdata() {
                   echo " Base class final printdata function"; }
           function nonfinal() {
                   echo "\n This is nonfinal function of base class"; }
}
class Derived extends Base
{          // Inheriting method nonfinal
           function nonfinal() {
                   echo "\n Derived class non final function";  }
           // Here printdata function can not be overridden
}
$obj = new Derived;
$obj->printdata();   // Base class final printdata function
$obj->nonfinal();     // Derived class non final function
?>
```