OOPs [Object-Oriented Programming System]

+91 9638601537 | virsing.vasava255@gmail.com | Java Developer, Ahmedabad, Gujarat

Summary

- **Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts :-**
  - Object
  - Class
  - Inheritance
  - Polymorphism
  - Abstraction
  - Encapsulation
- **Apart from these concepts, there are some other terms which are used in Object-Oriented design:**
  - Coupling
  - Cohesion
  - Association
  - Aggregation
  - Composition

OBJECT FROM OOPS

__destruct Function

- A destructor is called when the object is destructed or the script is stopped or exited.
- If you create a __destruct() function, PHP will automatically call this function at the end of the script.
- Notice that the destruct function starts with two underscores (__)!
- The example below has a __construct() function that is automatically called when you create an object from a class,
- and a __destruct() function that is automatically called at the end of the script:

__construct Function

- A constructor allows you to initialize an object's properties upon creation of the object.
- If you create a __construct() function, PHP will automatically call this function when you create an object from a class.
- We see in the example below, that using a constructor saves us from calling the set_name() method which reduces the amount of code:
- Notice that the construct function starts with two underscores (__)!.

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

Static Methods

- Static methods can be called directly - without creating an instance of the class first.
- Static methods are declared with the static keyword:

```
class ClassName {
  public static function staticMethod() {
    echo "Hello World!";
  }
}
```

Static Properties

- Static properties can be called directly - without creating an instance of a class.
- Static properties are declared with the static keyword:

```
class ClassName {
  public static $staticProp = "W3Schools";
}
```

Iterable

- An iterable is any value which can be looped through with a foreach() loop.
- Arrays : All arrays are iterables, so any array can be used as an argument of a function that requires an iterable.
- Iterators : Any object that implements the Iterator interface can be used as an argument of a function that requires an iterable.
- An iterator contains a list of items and provides methods to loop through them. It keeps a pointer to one of the elements in the list. Each item in the list should have a key which can be used to find the item.
- An iterator must have these methods:
- **current() -** Returns the element that the pointer is currently pointing to. It can be any data type
- **key()** Returns the key associated with the current element in the list. It can only be an integer, float, boolean or string
- **next()** Moves the pointer to the next element in the list
- **rewind()** Moves the pointer to the first element in the list
- **valid()** If the internal pointer is not pointing to any element.

Class

- A class is the blueprint objects.
- Number of object.
  - class Virsing { } *example - Student*
  - Properties :- name, birth of date, address
  - Methods/Task :- read(), play(), write()

Object

- Real word entity, Properties, Task performed, Instance of class.
- Example :- **Human**
- Properties : - Name, Color, Height, Weight
- Method : - Walk(), Run(), Read(), Write()

Abstraction

- Showing only essential parts and hiding the implementation detail is known as abstraction.
  - Example :- Android Application => apk , exe.

Encapsulation

- Binding the variable and method in single unit is known as encapsulation.
  - Example :- Name => Variable => Method

Inheritance

- Acquired the properties of one class to another class is known as Inheritance.
  - Super class -> Parent class -> Base class
  - Subclass -> Child class -> Derived class
  - **Types of Inheritance**
  - Single-level inheritance.
  - Multi-level Inheritance.
  - Hierarchical Inheritance.
  - Multiple Inheritance.
  - Hybrid Inheritance.

Polymorphism

- Performing same method or task in diffrent ways is known as polymorphism.
  - Add ()
  - Add(int a , int b)
  - **Method Overloading**
    - Two or more methods have the same name if they differ in parameters (different number of parameters, different types of parameters, or both).
      - class Dog{
      - public void bark(){
      - System.out.println("woof ");
      - }
      - //overloading method
      - public void bark(int num){
      - for(int i=0; i

        - System.out.println("woof ");
        - }
        - }
    - **Method Overriding**
        - method overriding occurs when a subclass (child class) has the same method as the parent class.
            - class Dog{
            - public void bark(){
            - System.out.println("woof ");
            - }
            - }
            - class Hound extends Dog{
            - public void sniff(){
            - System.out.println("sniff ");
            - }
            -
            - public void bark(){
            - System.out.println("bowl");
            - }
            - }
            -
            - class OverridingTest{
            - public static void main(String [] args){
            - Dog dog = new Hound();
            - dog.bark();
            - }
            - }

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:
    - interface Animal {
    - public function makeSound();
    - }
    - class Cat implements Animal {
    - public function makeSound() {
    - echo "Meow";
    - }
    - }
    - $animal = new Cat();
    - $animal->makeSound();

Traits

- Traits allow you to reuse various methods freely in many different classes that do not need to be in the same class hierarchy.
- Inheritance allows classes to reuse the code vertically while the traits allow classes reuse the code horizontally.
    - trait Preprocessor{
    - public function preprocess(){
    - echo 'Preprocess...done' ;
    - }
    - }
    - trait Compiler{
    - public function compile() {
    - echo 'Compile code... done' . ";
    - }
    - }
    - class IDE {
    - use Preprocessor, Compiler;
    - public function run() {

- $this->preprocess();
- $this->compile();
- echo 'Execute the file...done' . ';
- }
- }
- $ide = new IDE();
- $ide->run();

Skills

Object, Class, Inheritance, Overloading, Polymorphism, Abstraction, Overriding, Encapsulation

Education

Xavier's
OOPs provides the ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language