CET218

Advanced Web Programming

04 - PHP Objects

Dr. Ahmed Said

Start →

PHP Object Oriented Programming (OOP)

PHP Objects

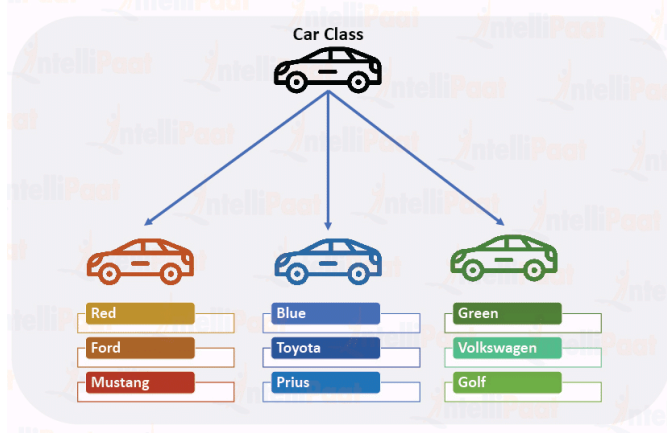
- In much the same way that functions represent a huge increase in programming power over the early days of computing.
- **O**bject-**O**riented **P**rogramming (OOP) takes the use of functions in a different direction.
- Once you get the hang of condensing **reusable** bits of code into functions, it's not that great a leap to consider bundling the functions and their data into objects.
- **OOP** is a way of organizing your code so that it is easier to understand and maintain.

Terminology

- Imagine our universe made of different objects like **sun**, **earth**, **moon** etc.
- Similarly, we can imagine a car made of different objects like **wheel**, **steering**, **gear** etc, some behaviors (functions) **move forward**, **turn right**, **turn left**, etc, and some properties (variables) like **color**, **speed**, **weight** etc.
- Same way there is object oriented programming concepts which assume **everything as an object of certain class**.

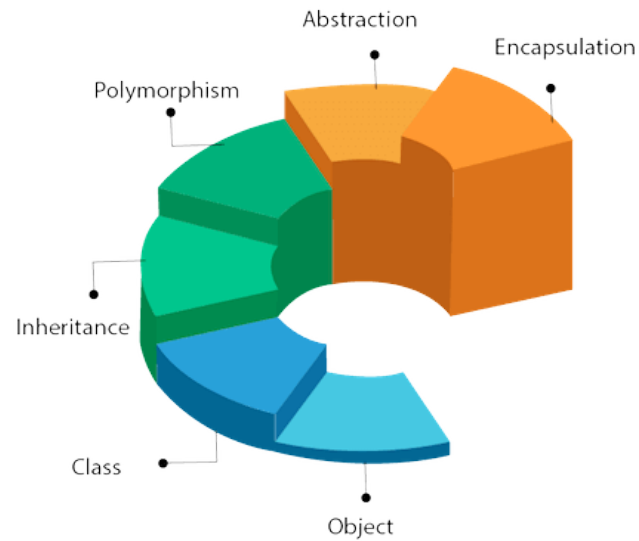
Classes and Objects

- **Class**: A class is a blueprint for objects, used to hold the objects along with their behaviors (functions) and properties.
- **Object**: An object in a class is an instance that has its own behaviors and properties. Objects can be related to the entities in real life.
- **Examples**:
  - A class `Car` can have objects like `Ali BMW`, `Sama Audi`, `Ahmed Toyota` etc.
  - A class `Person` can have objects like `Ali`, `Sama`, `Ahmed` etc.



Classes and Objects, cont.

- The data associated with an object is called its properties; the functions it uses are called methods.
- In defining a class, you supply the names of its properties and the code for its methods.
- OOP Concepts:
  - **Encapsulation**: Wrapping up of data and functions into a single unit.
  - **Inheritance**: A class can inherit properties and methods from another class.
  - **Polymorphism**: The ability to use a single interface for different data types.

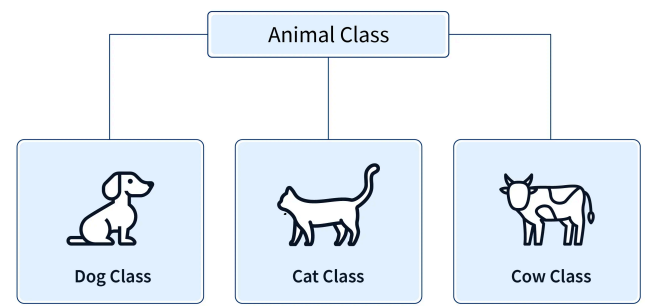


Classes and Objects, cont.

- When you're creating objects, it is best to use encapsulation, or writing a class in such a way that only its methods can be used to manipulate its properties (variables).
- In other words, you deny outside code direct access to its data. The methods you supply are known as **the object's interface**.
- This approach makes debugging easy: you have to fix faulty code only within a class.
- Additionally, when you want to upgrade a program, if you have used proper encapsulation and maintained the same interface, **you can simply develop new replacement classes**.

Classes and Objects, cont.

- Once you have created a class, you may find that you need another class that is similar to it but not quite the same.
- The quick and easy thing to do is to define a new class using **inheritance**.
- When you do this, your new class has all the properties of the one it has inherited from.
- The original class is now called the **parent** (or occasionally the superclass), and the new one is the **subclass** (or derived class).



Tip

- An excellent benefit of this system is that if you improve the speed or any other aspect of the `superclass`, its `subclasses` will receive the same benefit.
- On the other hand, any change made to the `parent/superclass` could break the `subclass`.

Declaring a Class

- Before you can use an object, you must define a class with the `class` keyword.
- Class definitions contain the class name (which is case-sensitive), its properties, and its methods.

```php
class User
{
  public $name, $password;
  function save_user()
  {
    echo "Save User code goes here";
  }
}
```

```php
$object = new User;
print_r($object);
```

```html
User Object
(
  [name] =>
  [password] =>
)
```

Creating an Object

- Once you have defined a class, you can create an object of that class using the `new` keyword.
- `$object = new Class`

```php
$object = new User;
$temp = new User('name', 'password');
```

Accessing Properties and Methods

- Extends the previous code by setting object properties and calling a method.

```php
class User
{
  public $name, $password;
  function save_user()
  {
    echo "Save User code goes here";
  }
}

$object = new User;
print_r($object); echo "<br>";
```

```html
User Object
(
[name] =>
[password] =>
)
```

Accessing Properties and Methods, cont.

```php
class User
{
  ...
}

$object = new User;
$object->name = "Joe";
$object->password = "mypass";
print_r($object); echo "<br>";
$object->save_user();
```

```html
User Object
(
[name] => Joe
[password] => mypass
)
Save User code goes here
```

Cloning Objects

```php
$object1 = new User();
$object1->name = "Alice";
$object2 = $object1;
$object2->name = "Amy";
echo "object1 name = " . $object1->name . "<br>";
echo "object2 name = " . $object2->name;
class User {
  public $name;
}
```

- What is the output?

```html
object1 name = Amy
object2 name = Amy
```

- Once you have created an object, it is passed by reference when you pass it as a parameter (**Shallow Copy**).
  - In other words, making object assignments **does not copy objects in their entirety**.
  - Both `$object1` and `$object2` refer to the same object, so changing the `name` property of `$object2` to `Amy` also sets that property for `$object1`.

Cloning Objects, cont.

- To avoid this confusion, you can use the clone operator, which creates a new instance of the class and copies the property values from the original instance to the new instance.

```php
$object1 = new User();
$object1->name = "Alice";
$object2 = clone $object1;
$object2->name = "Amy";
echo "object1 name = " . $object1->name . "<br>";
echo "object2 name = " . $object2->name;
class User
{
  public $name;
}
```

```html
object1 name = Alice
object2 name = Amy
```

- This is called a **deep copy**.

Constructors

- A **constructor** is a special method that is called when an object is created.
- When creating a new object, you can pass a list of arguments to the class being called.
- To do this you use the function name `__construct` (that is, construct preceded by two underscore characters).

```php
class User
{
  function __construct($param1, $param2)
  {
  //   Constructor statements go here
  }
}
```

Destructors

- A **destructor** is a special method that is called when an object is destroyed.
- This ability is useful when code has made the last reference to an object or when a script reaches the end.
  - The destructor can do cleanup such as releasing a connection to a database or some other resource that you reserved within the class.
  - Because you reserved the resource within the class, you have to release it here, or it will stick around indefinitely.
  - Many system-wide problems are caused by programs reserving resources and forgetting to release them.

```php
class User
{
  function __destruct()
  {
    // Destructor code goes here
  }
}
```

Writing Methods

- Methods are **functions that are defined within a class**.
- Declaring a method is similar to declaring a function, but there are a few differences.
  - For example, method names beginning with a double underscore (__) are reserved (for example, for __construct and __destruct), and you should not create any of this form.
  - You also have access to a special variable called `$this`, which can be used to access the current object's properties.

```php
class User {
  public $name, $password;
  function get_password()
  {
    return $this->password;
  }
}
```

```php
$object = new User;
$object->password = "secret";
echo $object->get_password();
```

Note

How the preceding `$` of the property `$password` is omitted when we use the `->` operator. Leaving the `$` in place is a typical error you may run into, particularly when you first use this feature.

Declaring Properties

- Properties are **variables that are defined within a class**.
- It is not necessary to explicitly declare properties within classes, as they can be implicitly defined when first used.

```php
$object1 = new User();
$object1->name = "Alice";
echo $object1->name;
class User { ... }
```

- Because PHP implicitly declares the property `$object1->name` for you.
- But this kind of programming can lead to bugs that are infuriatingly difficult to discover, because name was declared from outside the class.

Declaring Properties, cont.

- To avoid this problem (To help yourself and anyone else who will maintain your code), I advise that you get into the habit of always declaring your properties explicitly within classes.
- Also, when you declare a property within a class, you may assign a default value to it.
- The value you use must be a constant and not the result of a function or expression.

```php
class Test
{
  public $name = "Paul Smith"; // Valid
  public $age = 42; // Valid
  public $time = time(); // Invalid - calls a function
  public $score = $level * 2; // Invalid - uses an expression
}
```

Declaring Constants

- Constants are similar to properties, but they are declared with the `const` keyword.
  - In the same way that you can create a global constant with the define function, you can define constants inside classes.
  - The generally accepted practice is to use **UPPERCASE** letters to make them stand out
  - You can reference constants directly, using the `self` keyword and double colon operator.

```php
Translate::lookup();
class Translate
{
  const ENGLISH = 0;
  const SPANISH = 1;
  const FRENCH = 2;
  const GERMAN = 3;
  // ...
  static function lookup()
  {
    echo self::SPANISH;
  }
}
```

Note

- Note that this code calls the class directly, using the double colon operator (`Scope resolution operator`) at line 1, without creating an instance of it first.

Static Methods

- A static method is a method that is **called on a class rather than an object**.
- You can call a static method without creating an object of the class.
- To declare a static method, you use the `static` keyword.

```php
User::pwd_string();
class User
{
  static function pwd_string()
  {
    echo "Please enter your password";
  }
}
```

Note

- How we call the class itself, along with the static method, using a double colon (also known as the **scope resolution operator**), not `->`.
- Static functions are useful for performing actions relating to the class itself but not to specific instances of the class.

Static Properties

- Most data and methods apply to instances of a class.
  - These facts and operations apply separately to each user and therefore use instance-specific properties and methods.
- A static property is a property that is **shared by all instances of a class**.
- But occasionally you'll want to maintain data about a whole class.
  - For instance, to report how many users are registered, you will store a variable that applies to the whole User class.

Static Properties, cont.

```php
$temp = new Test();
echo "Test A: " . Test::$static_property . "<br>";
echo "Test B: " . $temp->get_sp() . "<br>";
echo "Test C: " . $temp->static_property . "<br>";
class Test
{
  static $static_property = "I'm static";
  function get_sp()
  {
    return self::$static_property;
  }
}
```

Note

- How the method `get_sp` accesses `$static_property` using the keyword `self`. This is how a static property or constant can be directly accessed within a class.

- A property declared static cannot be directly accessed within an instance of a class, but a static method can.

```html
Test A: I'm static
Test B: I'm static
Notice: Undefined property: Test::$static_property
Test C:
```

Inheritance

- **Inheritance** is a way to create a new class that is based on an existing class.
- The new class inherits all the properties and methods of the existing class.
- Inheritance is useful because it allows you to create a new class that is based on an existing class, but with some modifications.
- You achieve this using the `extends` keyword.

Inheritance: Example.

```php
class User {
  public $name, $password;
  function save_user() { echo "Save User code goes here"; }
}
class Subscriber extends User {
  public $phone, $email;
  function display() {
    echo "Name: " . $this->name . "<br>";
    echo "Pass: " . $this->password . "<br>";
    echo "Phone: " . $this->phone . "<br>";
    echo "Email: " . $this->email;
  }
}
$object = new Subscriber;
$object->name = "Fred";
$object->password = "pword";
$object->phone = "012 345 6789";
$object->email = "fred@bloggs.com";
$object->display();
```

```html
Name: Fred
Pass: pword
Phone: 012 345 6789
Email: fred@bloggs.com
```

Property and Method Scope

- PHP provides three keywords for controlling the scope of properties and methods (members):
  - **Public**: Can be accessed from outside the class.
  - **Private**: Can only be accessed from within the class.
  - **Protected**: Can be accessed from within the class and any subclasses.
- Here's how to decide which you need to use:
  - Use public when outside code should access this member and extending classes should also inherit it.
  - Use protected when outside code should not access this member but extending classes should inherit it.
  - Use private when outside code should not access this member and **extending classes also should not inherit it**.

```php
class Example
{
  var $name = "Michael"; // Same as public but deprecated
  public $age = 23; // Public property
  protected $usercount; // Protected property
  private function admin() { /* ... */ }
}
```

Inheritance: The `parent` Keyword

- When you use inheritance, you can call a method from the parent class using the `parent` keyword.
- This is useful when you want to add functionality to a method in the subclass without having to rewrite the entire method.

```php
class Dad {
  function test() {
    echo "[Class Dad] I am your Father<br>";
  }
}
class Son extends Dad {
  function test() {
    echo "[Class Son] I am Luke<br>";
  }
  function test2() {
    parent::test();
  }
}
$object = new Son;
$object->test();
$object->test2();
```

Inheritance: The `parent` Keyword, cont.

```
[Class Son] I am Luke
[Class Dad] I am your Father
```
html

- If you wish to ensure that your code calls a method from the current class, you can use the self keyword, like this: `self::method();`

Inheritance: Subclass Constructors

- When you create a subclass, you can define a constructor for it.
- You should be aware that PHP will not automatically call the constructor method of the parent class not like C++.
- If you want to be certain that all initialization code is executed, subclasses should always call the parent constructors

Inheritance: Subclass Constructors, cont.

```php
class Wildcat {
  public $fur; // Wildcats have fur
  function __construct()
  {
    $this->fur = "TRUE";
  }
}
class Tiger extends Wildcat {
  public $stripes; // Tigers have stripes
  function __construct()
  {
    parent::__construct(); // Call parent constructor first
    $this->stripes = "TRUE";
  }
}
$object = new Tiger();
echo "Tigers have...<br>";
echo "Fur: " . $object->fur . "<br>";
echo "Stripes: " . $object->stripes;
```

```html
Tigers have...
Fur: TRUE
Stripes: TRUE
```

Inheritance: The `final` Keyword

- When you wish to prevent a subclass from overriding a superclass method, you can use the `final` keyword.

```php
class User
{
  final function copyright()
  {
    echo "This class was written by Joe Smith";
  }
}
```

PHP Version Compatibility

- PHP is in an ongoing process of development, and there are multiple versions.
- If you need to check whether a particular function is available to your code, you can use the `function_exists` function, which **checks all predefined and user-created functions**.

```php
if (function_exists("array_combine"))
{
  echo "Function exists";
}
else
{
  echo "Function does not exist - better write your own function";
}
```

- This code checks for `array_combine`, a function specific to only some versions of PHP `>5`.

PHP Version Compatibility, cont.

- Using code such as this, you can take advantage of features in newer versions of PHP and yet still have your code run on earlier versions where the newer features are unavailable, as long as you replicate any features that are missing.
- **Your functions may be slower than the built-in ones, but at least your code will be much more portable**.
- You can also use the `phpversion` function to determine which version of PHP your code is running on.

```php
echo phpversion();
```

```text
8.0.0
```