OOP'S – PART:2

What are Abstract Classes and Methods?

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
abstract class ParentClass {
  abstract public function someMethod1();
  abstract public function someMethod2($name, $color);
  abstract public function someMethod3() : string;
}
?>
```

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.

What are 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);
  public function someMethod3() : string;
}
?>
```

PHP - Interfaces vs. Abstract Classes

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

Using Interfaces

To implement an interface, a class must use the implements keyword.

A class that implements an interface must implement **all** of the interface's methods.

Example

```php
<?php
interface Animal {
  public function makeSound();
}

class Cat implements Animal {
  public function makeSound() {
    echo "Meow";
  }
}

$animal = new Cat();
$animal->makeSound();
?>
```

What are 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
class MyClass {
  use TraitName;
}
?>
```

Static Methods

Static methods can be called directly - without creating an instance of the class first.

Static methods are declared with the static keyword:

Syntax

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

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

Syntax

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

Let's look at an example:

Example

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

// Call static method
greeting::welcome();
?>
```

Static Properties

Static properties can be called directly - without creating an instance of a class.

Static properties are declared with the static keyword:

Syntax

```
<?php
class ClassName {
  public static $staticProp = "W3Schools";
}
?>
```

To access a static property use the class name, double colon (::), and the property name:

Syntax

```
ClassName::$staticProp;
```

Let's look at an example:

Example

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

// Get static property
```

```
echo pi::$value;
?>
```

More on Static Properties

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 (::):

Example

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

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

To call a static property from a child class, use the parent keyword inside the child class:

Example

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

class x extends pi {
  public function xStatic() {
    return parent::$value;
  }
}
```

```
// Get value of static property directly via child class
echo x::$value;

// or get value of static property via xStatic() method
$x = new x();
echo $x->xStatic();
?>
```

PHP Namespaces

Namespaces are qualifiers that solve two different problems:

1. They allow for better organization by grouping classes that work together to perform a task
2. They allow the same name to be used for more than one class

For example, you may have a set of classes which describe an HTML table, such as Table, Row and Cell while also having another set of classes to describe furniture, such as Table, Chair and Bed. Namespaces can be used to organize the classes into two different groups while also preventing the two classes Table and Table from being mixed up.

Declaring a Namespace

Namespaces are declared at the beginning of a file using the namespace keyword:

Syntax

Declare a namespace called Html:

```
<?php
namespace Html;
?>
```

Using Namespaces

Any code that follows a namespace declaration is operating inside the namespace, so classes that belong to the namespace can be instantiated without any qualifiers. To access classes from outside a namespace, the class needs to have the namespace attached to it.

Example

Use classes from the Html namespace:

```php
<?php
$table = new Html\Table();
$row = new Html\Row();
?>
```

What is an Iterable?

An iterable is any value which can be looped through with a foreach() loop.

The iterable pseudo-type was introduced in PHP 7.1, and it can be used as a data type for function arguments and function return values.

Using Iterables

The iterable keyword can be used as a data type of a function argument or as the return type of a function:

Example

Use an iterable function argument:

```php
<?php
function printIterable(iterable $myIterable) {
  foreach($myIterable as $item) {
    echo $item;
  }
}
```

```php
$arr = ["a", "b", "c"];
printIterable($arr);
?>
```

Example

Return an iterable:

```php
<?php
function getIterable():iterable {
  return ["a", "b", "c"];
}

$myIterable = getIterable();
foreach($myIterable as $item) {
  echo $item;
}
?>
```