PHP – Interfaces

Just as a class is a template for its objects, an **interface** in PHP can be called as a template for classes. We know that when a class is instantiated, the properties and methods defined in a class are available to it. Similarly, an interface in PHP declares the methods along with their arguments and return value. These methods do not have any body, i.e., no functionality is defined in the interface.

A **concrete** class has to implement the methods in the interface. In other words, when a class implements an interface, it must provide the functionality for all methods in the interface.

An interface is defined in the same way as a class is defined, except that the keyword "**interface**" is used in place of class.

```
interface myinterface {
   public function myfunction(int $arg1, int $arg2);
   public function mymethod(string $arg1, int $arg2);
}
```

Note that the methods inside the interface have not been provided with any functionality. Definitions of these methods must be provided by the class that implements this interface.

When we define a child class, we use the keyword "**extends**". In this case, the class that must use the keyword "**implements**".

All the methods declared in the interface must be defined, with the same number and type of arguments and return value.

```
class myclass implements myinterface {
   public function myfunction(int $arg1, int $arg2) {
      ## implementation of myfunction;
   }
   public function mymethod(string $arg1, int $arg2) {
      # implementation of mymethod;
   }
}
```

Note that all the methods declared in an interface must be public.

Example

Let us define an interface called **shape**. A shape has a certain area. You have shapes of different geometrical appearance, such as rectangle, circle etc., each having an area, calculated with different formula. Hence the shape interface declares a method area() that returns a float value.

```
interface shape {
   public function area(): float;
}
```

Next, we shall define a circle class that implements shape interface, to implement, the class must provide a concrete implementation of the functions in the interface. Here, the area() function in circle class calculates the area of a circle of a given radius.

```
class circle implements shape {
   var $radius;
   public function __construct($arg1) {
      $this->radius = $arg1;
   }
   public function area(): float {
      return pow($this->radius,2)*pi();
   }
}
```

We can now declare an object of circle class, and call the area() method.

```
$cir = new circle(5);
echo "Radius : " . $cir->radius .  " Area of Circle: " . $cir->area(). PHP_EOL;
```

An interface can be implemented by any number of classes (which may be unrelated to each other) provided the implementing class provides functionality of each method in the interface.

Here is a Square class that implements shape. The area() method returns the square of the side property.

```
class square implements shape {
   var $side;
   public function __construct($arg1) {
      $this->side = $arg1;
   }
   public function area(): float {
      return pow($this->side, 2);
```

```
    }
}
```

Similarly, create a Square object and call the area() method.

Example

Given below is the complete code for a shape interface, implemented by circle and Square classes –

</> Open Compiler

```
<?php
   interface shape {
      public function area(): float;
   }

   class square implements shape {
      var $side;
      public function __construct($arg1) {
         $this->side = $arg1;
      }
      public function area(): float {
         return pow($this->side, 2);
      }
   }
   class circle implements shape {
      var $radius;
      public function __construct($arg1) {
         $this->radius = $arg1;
      }
      public function area(): float {
         return pow($this->radius,2)*pi();
      }
   }

   $sq = new square(5);
   echo "Side: " . $sq->side .  " Area of Square: ". $sq->area() . PHP_EOL;

   $cir = new circle(5);
```

```
   echo "Radius: " . $cir->radius .  " Area of Circle: " . $cir->area(). PHP_EOL;
?>
```

It will produce the following **output** –

```
Side: 5 Area of Square: 25
Radius: 5 Area of Circle: 78.539816339745
```


Multiple Inheritance in PHP

PHP doesn't have the provision to build a child class that extends two parent classes. In other words, the statement –

```
class child extends parent1, parent2
```

is not accepted. However, PHP does support having a child class that extends one parent class, and implementing one or more interfaces.

Let use look at the following example that shows a class that extends another and implements an interface.

First, the parent class marks. It has three instance variables or properties $m1, $m2, $m3 representing the marks in three subjects. A **constructor** is provided to initialize the object.

```
class marks {
   protected int $m1, $m2, $m3;
   public function __construct($x, $y, $z) {
      $this->m1 = $x;
      $this->m2 = $y;
      $this->m3 = $z;
   }
}
```

We now provide an interface called percent that declares a method percent(), which should return a float but doesn't have a function body.

```
interface percent {
   public function percent(): float;
}
```

We now develop a class that extends marks class and provides implementation for percent() method in the interface.

```
class student extends marks implements percent {
   public function percent(): float {
      return ($this->m1+$this->m2+$this->m3)*100/300;
   }
}
```

The student class inherits the parent constructor, but provides implementation of parent() method that returns the percentage of marks.

Example

The complete code is as follows –

</> Open Compiler

```
<?php
   class marks {
      protected int $m1, $m2, $m3;
      public function __construct($x, $y, $z) {
         $this->m1 = $x;
         $this->m2 = $y;
         $this->m3 = $z;
      }
   }
   interface percent {
      public function percent(): float;
   }

   class student extends marks implements percent {
      public function percent(): float {
         return ($this->m1+$this->m2+$this->m3)*100/300;
      }
   }

   $s1 = new student(50, 60, 70);
```

```
    echo "Percentage of marks: ". $s1->percent() . PHP_EOL;
?>
```

It will produce the following **output** –

```
Percentage of marks: 60
```

The interface in PHP defines a framework of methods that classes use to provide a different but concrete implementation of their own.