PHP - Overloading

In C++ or Java, the term means a class can a class method of same name more than once but with different arguments and/or return type. In PHP, the term overloading has a different interpretation. It is a feature with which properties and methods can be created dynamically. PHP's magic methods (method names starting with double underscore) are used to set up dynamic properties and methods.

The magic methods used for the purpose of overloading are invoked when interacting with properties or methods that have not been declared or are not visible in the current scope.

Property Overloading

The examples of PHP's magic methods are __construct(), __destruct(), __tostring(), etc. PHP uses the following magic methods for overloading properties.

```
public __set ( string $name , mixed $value ) : void
public __get ( string $name ) : mixed
public __isset ( string $name ) : bool
public __unset ( string $name ) : void
```

Here,

- **__set()** is run for writing data to inaccessible properties that are protected or private or non-existing.
- **__get()** reads data from inaccessible properties.
- **__isset()** calls isset() or empty() on inaccessible properties.
- **__unset()** is invoked when unset() is called on inaccessible properties.

The **$name** argument used above is the name of the property to be set or retrieved. The **$value** argument of __set() method specifies the value to be assigned to the property.

The **__isset()** method checks if a certain property has been set or not. The **__unset()** method removes the property.

Property overloading works only in **object context**. In any **static context**, these magic methods will not be triggered. Hence they should not be declared static.

Example

In the following code, a dynamic property named myprop, which is not declared in the class, is set and retrieved.

</> Open Compiler

```
<?php
   class myclass {
      public function __set($name, $value) {
         echo "setting $name property to $value \n";
         $this->$name = $value;
      }

      public function __get($name) {
         echo "value of $name property is ";
         return $this->$name;
      }
   }

   $obj = new myclass();

   # This calls __set() method
   $obj->myproperty="Hello World!";

   # This call __get() method
   echo "Retrieving myproperty: " . $obj->myproperty . PHP_EOL;
?>
```

It will produce the following **output** –

```
setting myproperty property to Hello World!
Retrieving myproperty: Hello World!
```

The **__set()** and **__get()** magical methods also set and retrieve a property which is declared as private. Add the following statement inside myclass (before the function definitions)

```
private $myproperty;
```

You can check if the property, define __isset() method in **myclass** –

```
public function __isset($name) {
   return isset($this->$name);
```

```
}
```

Check if the property is set with this statement –

```
var_dump (isset($obj->myproperty));
```

Which in this case returns **true**.

To unset the dynamically created property with the __unset() method defined in **myclass** –

```
public function __unset($name) {
   unset($this->$name);
}
```

The following code would return **false** –

```
var_dump (isset($obj->myproperty));
```

Method Overloading

Two magic methods used to set methods dynamically are **__call()** and **__callStatic()**.

```
public __call (string $name , array $arguments) : mixed
public static __callStatic (string $name , array $arguments) : mixed
```

The __call() is triggered when invoking inaccessible (not defined or private) methods in an object context. On the other hand, the __callStatic() is triggered when invoking inaccessible methods in a static context.

Example

The following example demonstrates method overloading in PHP

</> Open Compiler

```
<?php
   class myclass {
      public function __call($name, $args) {

         // Value of $name is case sensitive.
         echo "Calling object method $name with " . implode(" ", $args). "\n";
```

```
      }
      public static function __callStatic($name, $args) {
         echo "Calling static method $name with " . implode(" ", $args). "\n";
      }
   }
   $obj = new myclass();

   # This invokes __call() magic method
   $obj->mymethod("Hello World!");

   # This invokes __callStatic() method
   myclass::mymethod("Hello World!");
?>
```

It will produce the following **output** –

```
Calling object method mymethod with Hello World!
Calling static method mymethod with Hello World!
```

Note that the use of "**->**" operator implies that the method is an **instance method**, and "::" operator means that the method is a **static method**.