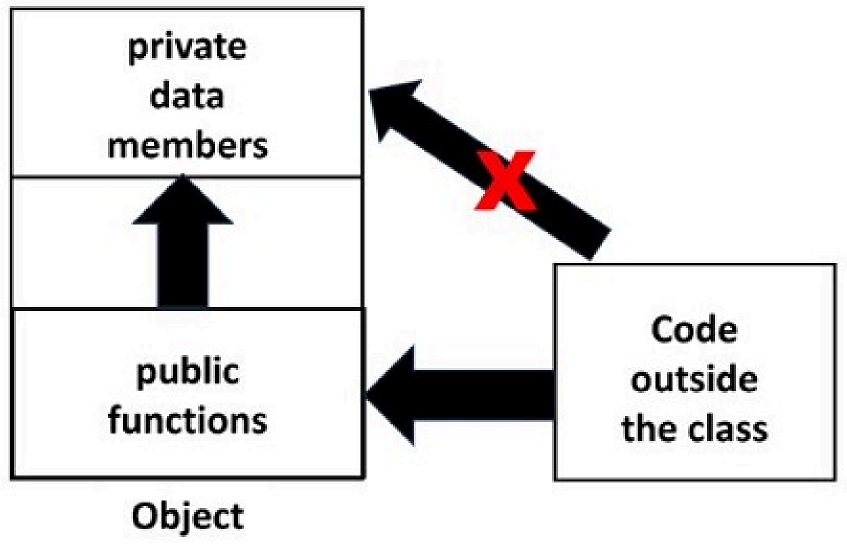
PHP - Encapsulation

PHP implements **encapsulation**, one of the important principles of OOP with access control keywords: **public, private** and **protected**.

Encapsulation refers to the mechanism of keeping the data members or properties of an object away from the reach of the environment outside the class, allowing controlled access only through the methods or functions available in the class.

The following diagram illustrates the principle of encapsulation in object-oriented programming methodology.



PHP’s keywords list contains the following keywords that determine the accessibility of properties and methods of an object, which is an instance of a class in PHP −

- **Public** − Class members are accessible from anywhere, even from outside the scope of the class, but only with the object reference.
- **Private** − Class members can be accessed within the class itself. It prevents members from outside class access even with the reference of the class instance.
- **Protected** − Members can be accessed within the class and its child class only, nowhere else.

These three keywords "**public, private** and **protected**" are often called access modifiers. They are also referred as visibility modes, as they decide upto what extent a certain class member is available.

Public Members

In PHP, the class members (both member variables as well as member functions) are public by default.

Example

In the following program, the member variables title and price of the object are freely accessible outside the class because they are public by default, if not otherwise specified.

Open Compiler

```php
<?php
   class Person {

      /* Member variables */
      var $name;
      var $age;

      /*Constructor*/
      function __construct(string $param1="Ravi", int $param2=28) {
         $this->name = $param1;
         $this->age = $param2;
      }

      function getName() {
         echo "Name: $this->name" . PHP_EOL;;
      }
      function getAge() {
         echo "Age: $this->age" . PHP_EOL;;
      }
   }
   $b1 = new Person();
   $b1->getName();
   $b1->getAge();
   echo "Name : $b1->name Age: $b1->age" . PHP_EOL;
?>
```

It will produce the following **output** −

```
Name: Ravi
Age: 28
Name : Ravi Age: 28
```

Note that the properties all the class members are public by default, you can explicitly declare them as public if desired. As a result, the instance methods getName() and getAge() can be called from outside the class.

Since properties name and age are also public, hence they can also be accessed outside the class, something which is not desired as per the principle of encapsulation.

Private Members

As mentioned above, the principle of encapsulation requires that the member variables should not be accessible directly. Only the methods should have the access to the data members. Hence, we need to make the member variables private and methods public.

Example

Let us change the declaration of name and age properties to private and run the following PHP script −

Open Compiler

```
<?php
   class Person {

      /* Member variables */
      private $name;
      private $age;

      /*Constructor*/
      function __construct(string $param1="Ravi", int $param2=28) {
         $this->name = $param1;
         $this->age = $param2;
      }

      public function getName() {
         echo "Name: $this->name" . PHP_EOL;;
      }

      public function getAge(){
         echo "Age: $this->age" . PHP_EOL;;
      }
   }

   $b1 = new Person();
   $b1->getName();
   $b1->getAge();
   echo "Name : $b1->name Age: $b1->age" . PHP_EOL;
?>
```

It will produce the following **output** −

```
Name: Ravi
Age: 28
PHP Fatal error:  Uncaught Error: Cannot access private property Person::$name in perso
```

The error message tells the reason that a private property cannot be accessed from a public scope.

Protected Members

The effect of specifying protected access to a class member is effective in case of class inheritance. We know that public members are accessible from anywhere outside the class, and private members are denied access from anywhere outside the class.

The **protected** keyword grants access to an object of the same class and an object of its inherited class, denying it to any other environment.

Example

Let us inherit the person class and define a student class. We shall change the name property from private to protected. The student class has a new public method getDetails() that prints the values of name and age properties.

Person class

```
<?php
class Person {

   /* Member variables */
   protected $name;
   private $age;
```

```
    /*Constructor*/
    function __construct(string $param1="Ravi", int $param2=28) {
       $this->name = $param1;
       $this->age = $param2;
    }

    public function getName(){
       echo "Name: $this->name" . PHP_EOL;;
    }

    public function getAge() {
       echo "Age: $this->age" . PHP_EOL;;
    }
 }
```

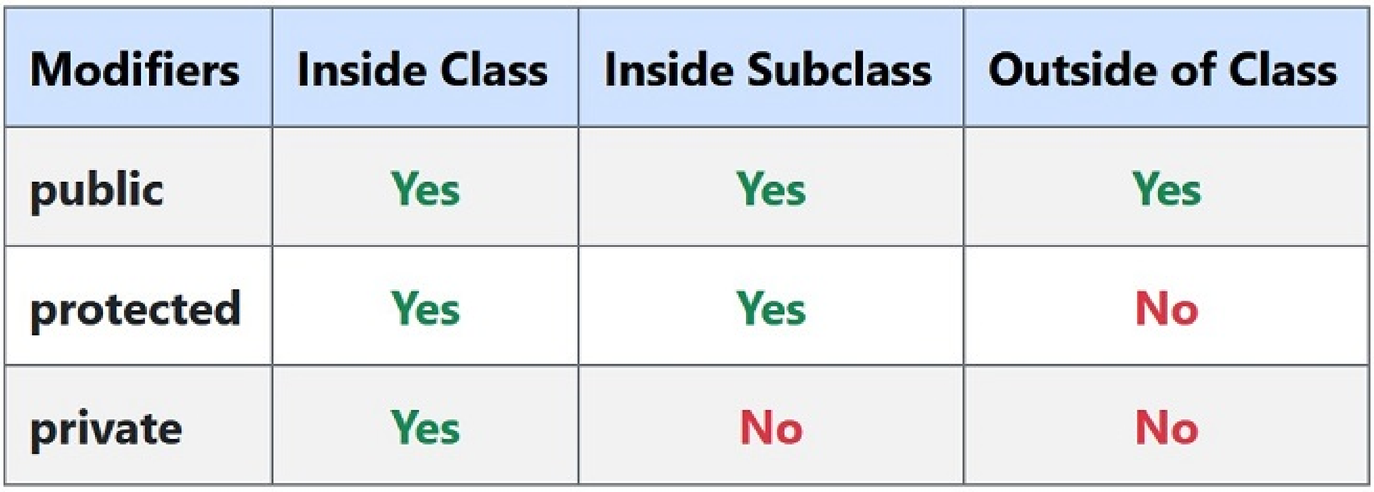
Student class

```
class student extends Person {
   public function getDetails() {
      echo "My Name: $this->name" . PHP_EOL;
      echo "My age: $this->age" . PHP_EOL;
   }
}
$s1 = new student();
$s1->getDetails();
?>
```

It will produce the following **output** –

```
My Name: Ravi
PHP Warning:  Undefined property: student::$age in person.php on line 28
My age:
```

The following table illustrates the rules of accessibility of class members in PHP –



Modifiers	Inside Class	Inside Subclass	Outside of Class
public	Yes	Yes	Yes
protected	Yes	Yes	No
private	Yes	No	No