Method and Property Visibility

This lesson discusses access modifiers like public, private and protected and how they affect the visibility of members.

We'll cover the following

Access modifiers provide access to the variables of a class. In this lesson, we will discuss the **three** visibility types that you can apply to **methods** (*class/object functions*) and **properties** (*class/object variables*) within a class. Access modifiers provide access control for the *method* or *property* to which they are applied.

Public

Declaring a method or a property as `public` allows the method or property to be accessed by:

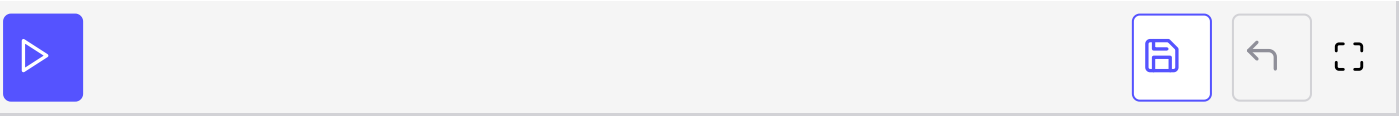
- The class that declared it.
- The classes that inherits from the declared class.
- Any external objects, classes, or code outside the class hierarchy.

Example

Run the code below to see how the `public` keyword works:

```
<?php
class Car
{
    public $name = " ";

    public function display()
```

```
    {
        echo "Name: $this->name" . "\n";
    }


    public function __construct($name)
    {
        $this->name = $name;

    }
}

$obj1 = new Car("BMW"); //creating an object of car and setting its name as BMW
echo "Name: " . $obj1->name; //accessing the "name" property of obj1 directly outside of class
echo "\n";

$obj2 = new Car("Mercedes"); //creating an object of car and setting its name as Mercedes
echo $obj2->display(); //accessing the "display" method of obj1 directly outside of class
?>
```



The example above shows that the `public` *property*, `$name` and the `public` *method* `display()`, can be accessed outside directly by the objects of the class.

Private

Declaring a method or a property as `private` allows the method or property to be accessed by:

- **Only** the class that declared it.

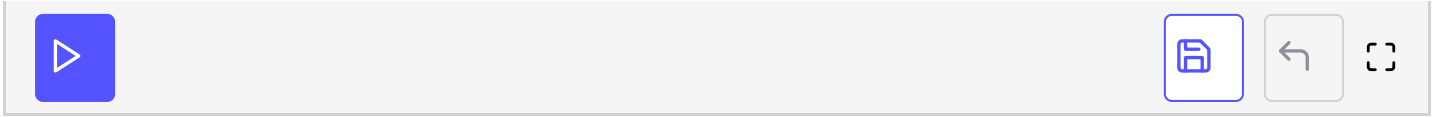
A `private` *method* or *property* is only visible and accessible within the class that created it.

Example

Run the code below to see how the `private` keyword works:

```
<?php
class Car
{
    public $name = " ";
    private $plateNumber;

    public function display()
    {
        echo "Name: $this->name" . "\n";
    }

    public function setPlateNumber($number)
    { //sets value of property plateNumber
        $this->plateNumber = $number;
```

```
    }

    public function getPlateNumber()
    { //returns the property "plateNumber"
        return $this->plateNumber;
    }

    public function __construct($name, $number)
    {
        $this->name = $name;
        $this->plateNumber = $number;

    }
}

$obj1 = new Car("BMW", 68775); //making a car object with values of name and platenumber set
echo $obj1->display(); //displaying name of car
echo "Plate number: " . $obj1->getPlateNumber(); //accessing plateNumber by calling getPlateNumber
echo "\n";
$obj1->setPlateNumber(47798); //changing PlateNumber value using setPlateNumber
echo "Plate number: " . $obj1->getPlateNumber(); //accessing plateNumber by calling getPlateNumber
echo "\n";

$obj2 = new Car("Mercedes", 89976);
//uncomment the line below and try running the code
//you will get an error as you cannot directly access a private member outside of the class it is
//echo $obj2->plateNumber;
?>
```



The example above shows that the methods `display()`, `getPlateNumber()` and `setPlateNumber()` in class `Car` can access the `private` property `$plateNumber`. The get and set functions are used to access as well as set values of the **private** members in a class.

In the code above, when you run the code accessing the `private` member `$plateNumber` directly, an error of *undefined property* is thrown as private property is accessed out of its scope.

Protected

Declaring a method or a property as `protected` allows the method or property to be accessed by:

- the class that declared it
- the classes that inherits from the declared class

This **does not allow** external objects, classes, or code outside the class hierarchy to access these methods or properties. An error occurs if they try to access it.

Example

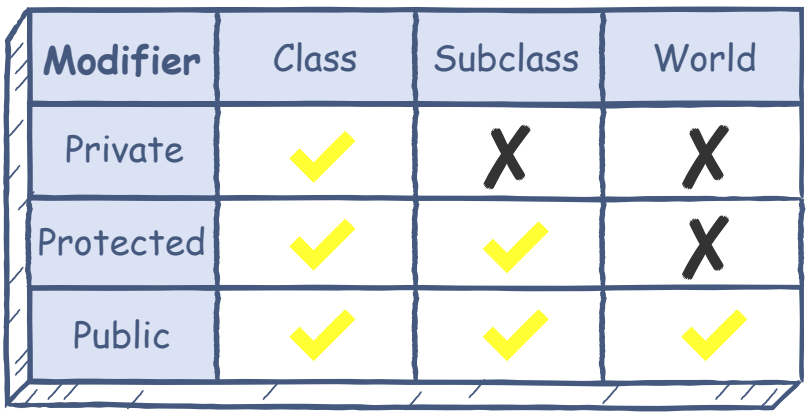
Run the code below to see how the `protected` keyword works:

```
<?php
class Car
{
    public $name = " ";
    protected $power = 2500;

    public function display()
    {
        echo "Name: $this->name" . "\n";
    }

    public function __construct($name)
    {
        $this->name = $name;

    }
}

$obj = new Car("Blue");
echo $obj->display();
echo $obj->power; //comment out this line to prevent an error
?>
```

The example above shows that when the main script tries to access the `protected` variable `$power`, an error: *cannot access protected property* is thrown.

> Removing `line 20`, in the above code, will allow the code to run successfully.

The table below illustrates the difference between all three access modifiers:



> **Note:** Objects of the same class will have access to each other's `private` and `protected` members even though they are not the same instances.

Run the code below to see an example:

```
<?php

class Test
{
        private $privateM=1;
        protected $protectedM=2;

        public function increase(Test $test)
        {
                $test-> privateM *= 10;
                $test-> protectedM *= 10;
        }

        public function __toString()
        {
                return "privateMember = ".$this->privateM.", protectedMember = ".$this->protectedM
        }
}

$test1 = new Test();
$test2 = new Test();
echo "before test1: $test1\n";

//call $test2 method on another instance of the Test class - $test1
$test2->increase($test1);

echo "after test1: $test1\n";
?>
```

This marks the end of our discussion on *classes*. In the next lesson we'll discuss the concept of *inheritance*.