Inheritance in object oriented programming

One of the main advantages of object-oriented programming is the ability to reduce code duplication with inheritance. Code duplication occurs when a programmer writes the same code more than once, a problem that inheritance strives to solve. In inheritance, we have a parent class with its own methods and properties, and a child class (or classes) that can use the code from the parent. By using inheritance, we can create a reusable piece of code that we write only once in the parent class, and use again as much as we need in the child classes.

How can a class inherit the code of another class?

Inheritance allows us to write the code only once in the parent, and then use the code in both the parent and the child classes.

class Parent {

// The parent’s class code

}

class Child extends Parent {

// The child can use the parent's class code

}

**Example:**

// The parent class

class Car {

// Private property inside the class

private $model;

// Public setter method

public function setModel($model)

{

$this -> model = $model;

}

public function hello()

{

return "beep! I am a <i>" . $this -> model . "</i><br />";

}

}

// The child class inherits the code

// from the parent class

class SportsCar extends Car {

// No code in the child class

}

// Create an instance from the child class

$sportsCar1 = new SportsCar();

// Set the value of the class’ property.

// For this aim, we use a method that we created in the parent

$sportsCar1 -> setModel('Mercedes Benz');

// Use another method that the child class inherited

// from the parent class

echo $sportsCar1 -> hello();

Result:

beep! I am a Mercedes Benz

How can a child class have its own methods and properties?

The parent class has its properties and methods

class Car {

// A private property or method can be used only by the parent

private $model;

// Public methods and properties can be used

// by both the parent and the child classes

public function setModel($model)

{

$this -> model = $model;

}

public function getModel()

{

return $this -> model;

}

}

// The child class can use the code it inherited from the parent class,

// and it can also have its own code

class SportsCar extends Car {

private $style = 'fast and furious';

public function driveItWithStyle()

{

return 'Drive a ' . $this -> getModel() . ' <i>' .

$this -> style . '</i>';

}

}

// Create an instance from the child class

$sportsCar1 = new SportsCar();

// Use a method that the child class inherited from the parent class

$sportsCar1 -> setModel('Ferrari');

// Use a method that was added to the child class

echo $sportsCar1 -> driveItWithStyle();

Result:

Drive a Ferrari fast and furious.

The protected access control modifier

When we declare a property or a method as protected, we can approach it from both the parent and the child classes.

// The parent class

class Car {

// The $model property is private, thus it can be accessed

// only from inside the class

**private $model;**

// Public setter method

public function setModel($model)

{

$this -> model = $model;

}

}

// The child class

class SportsCar extends Car{

// Tries to get a private property

// that belongs to the parent

public function hello()

{

return "beep! I am a <i>" . $this -> model . "</i><br />";

}

}

//Create an instance from the child class

$sportsCar1 = new SportsCar();

// Set the class model name

$sportsCar1 -> setModel('Mercedes Benz');

// Get the class model name

echo $sportsCar1 -> hello();

Result:

Notice: Undefined property: SportsCar::$model

**Example 2.**

// The parent class

class Car {

// The $model property is now protected, so it can be accessed

// from within the class and its child classes

**protected $model;**

// Public setter method

public function setModel($model)

{

$this -> model = $model;

}

}

// The child class

class SportsCar extends Car {

// Has no problem to get a protected property

// that belongs to the parent

public function hello()

{

return "beep! I am a <i>" **. $this -> model** . "</i><br />";

}

}

// Create an instance from the child class

$sportsCar1 = new SportsCar();

// Set the class model name

$sportsCar1 -> setModel('Mercedes Benz');

// Get the class model name

echo $sportsCar1 -> hello();

Result:

beep! I am a Mercedes Benz

Now it works, because we can access a protected code that belongs to a parent from a child class.