Late Static Bindings

represents a feature which can be used to reference the called class in a context of static inheritance;

it boils down to the fact that the ***self*** word does not follow the same rules of inheritance (self always resolves to the class in which it is used);

this means that if you make a method in a parent class and call it from a child class, ***self*** will not reference the child as you might expect (but the parent);

late static binding introduces a new use for the static keyword, which addresses this particular shortcoming. When you use ***static***, it represents the class where you first use it, ie. it 'binds' to the runtime class (the class where the method is called from)

e.g. (using ***self***)

class Model

{

    protected static $name = 'Model';

    public static function find()

    {

        echo self::$name;

    }

}

class Product extends Model

{

    protected static $name = 'Product';

}

Product::find();  // prints Model

Example above prints ‘Model’, because ***self*** refers to the class where the method test() is defined;

e.g. (using ***static***)

class A {

    public static function who() {

        echo \_\_CLASS\_\_;

    }

    public static function test() {

        static::who();

    }

}

class B extends A {

    public static function who() {

        echo \_\_CLASS\_\_;

    }

}

B::test(); //prints B

Example above prints B, because ***static*** refers to the class where the method test() is called from (it ‘binds’ to the runtime class);

**Note:**

Late static bindings' resolution will stop at a fully resolved static call with no fallback. On the other hand, static calls using keywords like ***parent::*** or ***self::*** will forward the calling information to the class where the call has been made from.

class A {

    public static function foo() {

        static::who();

    }

    public static function who() {

        echo \_\_CLASS\_\_."\n";

    }

}

class B extends A {

    public static function test() {

        A::foo();

        parent::foo();

        self::foo();

    }

    public static function who() {

        echo \_\_CLASS\_\_."\n";

    }

}

class C extends B {

    public static function who() {

        echo \_\_CLASS\_\_."\n";

    }

}

C::test(); //prints A C C

//C::test calls the test() in B which calls:

    //A::foo() -> A (static refers to the class from which we call the method -> A in this case)

    //parent:foo() -> (parent:foo() calls the foo() in A,where static refers to the class from which we call the method -> C in this case)

    //self:foo() -> self:foo() refers to the class where foo() is defined ->A, foo() in a calls 'static'::who() and static here refers to the class where the method is called from (again C)

\*remember to start back at C::test() after each output print\*

**Note:**

We cannot have a stronger level of encapsulation in a child class than in parent

class Model

{

    //protected static $name = 'Model';

    public static function find()

    {

        echo static::$name;

    }

}

class Product extends Model

{

    protected static $name = 'Product';

}

Product::find();  // prints Product  (calls the find() in Model class, which echoes the $name property of the Product class -note Model has no $name property)

However, note the code acts exactly the same, even if we add the protected static $name prop to Model class. Note that if we replace static in find() with self, and leave the $name prop to Model class, it prints Model (as self references the current class in which the function is being defined):

class Model

{

    protected static $name = 'Model';

    public static function find()

    {

        echo self::$name;

    }

}

class Product extends Model

{

    protected static $name = 'Product';

}

Product::find();  // prints Model

**Note:** we cannot have a higher level of encapsulation in a child class than in parent (see below)

class A {

    private function foo() {

        echo "success!\n";

    }

    public function test() {

        $this->foo();

        static::foo();

    }

}

class B extends A {

}

class C extends A {

    private function foo() {

        /\* original method is replaced; the scope of the new one is C \*/

     echo 'success2';

    }

}

$b = new B(); // success success

$b->test();

$c = new C();

$c->test();  //success

//$b->test() prints success, success ($this->foo() prints success by calling foo() in A, static::foo() prints success because it refers to the class that called the method, in this case A so calls foo() from A once more);

//c$->test() // prints success only once (when it calls $this->foo() as it refers to A class, when it calls static::foo(), static refers to the class that called the method so C which has no foo method)

$c🡪test() prints ‘success’ only once because of ***self***, as it calls foo() from A (foo() from A, prints ‘success’ once because of $this🡪foo(), when it reaches self::foo(), as ***self*** refers to the class that called the function (in this case C and as C has a higher level of encapsulation foo() method (private instead of public), that method cannot be accessed. However, if we make foo() from C public, and we echo something inside of it, that will get printed on the screen when self::foo() is called from A;