EX

```php
<?php

class People {
    // These are the class properties
    public $name;
    public $country;

    // These are the class methods
    function set_name($name) {
        $this->name = $name;
    }
    function get_name() {
        return $this->name;
    }
    function set_country($country) {
        $this->country = $country;
    }
    function get_country() {
        return $this->country;
    }
}

// new instance of the People() class
$alex = new People();
// called the methods of the class
$alex->set_name('Alexander');
$alex->set_country('UK');
// accessing the object properties
echo "Name: " . $alex->get_name();
echo "<br>";
echo "Country: " . $alex->get_country();
```

constructor

When creating a new object, you can pass a list of arguments to the class being called. These are passed to a special method within the class, called the constructor, which initializes various properties. This is done using the `__construct()` function.

```php
<?php

class People {
    // These are the class properties
    public $name;
    public $country;

    // The constructor function
    function __construct($name, $country) {
        $this->name = $name;
        $this->country = $country;
    }
    // More class methods
    function get_name() {
        return $this->name;
    }
    function get_country() {
        return $this->country;
    }
}

// new instance of the People() class
$alex = new People('Alexander','UK');
// accessing the object properties
echo "Name: " . $alex->get_name();
echo "<br>";
echo "Country: " . $alex->get_country();
```

Destructors

PHP also has the ability to create destructor methods. This ability is useful when code has made the last reference to an object or when a script reaches the end. This is done by creating a `__destruct()` function. Let's take the example above and make a destructor function.

```php
<?php

class People {
    // These are the class properties
    public $name;
    public $country;

    // The constructor function
    function __construct($name, $country) {
        $this->name = $name;
        $this->country = $country;
    }
    function __destruct() {
        echo "Name: " . $this->name;
        echo "<br>";
        echo "Country: " . $this->country;
    }
}

// new instance of the People() class
$alex = new People('Alexander','UK');
```

constants

Class constants provide a mechanism for holding fixed values in a program. Class constants can only be defined with the `const` keyword - the `define` function cannot be used in this context.

Class constants may be accessed by using the double colon operator (so-called the scope resolution operator) on a class, much like static variables.

It is recommended to use uppercase letters for constants. Also note that constants are case-sensitive.

```php
<?php



class MultiplyBy10 {

    public $num;

    const MULT = 10;

    function __construct($num){

        $this->num = $num;

    }

    function __destruct() {

        echo $this->num * self::MULT;

    }

}



$mynum = new MultiplyBy10(5);
```

- Access Modifiers

PHP provides three keywords for controlling the scope of properties and methods:

- `public` - These properties are the default when declaring a variable using the `var` or `public` keywords, or when a variable is implicitly declared the first time it is used. The keywords `var` and `public` are interchangeable because, although deprecated, var is retained for compatibility with previous versions of PHP. Methods are assumed to be public by default.
- `protected` - These properties and methods (members) can be referenced only by the object's class methods and those of any subclasses.
- `private` - These members can be referenced only by methods within the same class—not by subclasses.

To help you to decide which you need to use:

- Use `public` when outside code should access this member and extending classes should also inherit it.
- Use `protected` when outside code should not access this member but extending classes should inherit it.
- Use `private` when outside code should not access this member and extending classes also should not inherit it.

```php
<?php


class Car {
```

```
    // These are the class properties

    public $name;

    protected $country;

    private $model;

}



$car = new Car();

$car->name = 'Ford';

$car->country = 'USA';

$car->model = '1995';
```

The result shows that `protected` and `private` properties of the object is not accessible and PHP returns an error.

- Inheritance

Once you have written a class, you can derive subclasses from it. This can save lots of painstaking code rewriting: you can take a class similar to the one you need to write, extend it to a subclass, and just modify the parts that are different. This is achieved using the extends operator.

```php
<?php

class Person
{
    public $name;
    public $age;
    public function __construct($name, $age) {
        $this->name = $name;
        $this->age = $age;
    }
    public function get_age() {
        return $this->age;
    }
}

// Student is inherited from Person
class Student extends Person
```

```
{
    public function sayHello() {
        echo "Hello I am a student.";
    }
}

class Employee extends Person
{
    public function sayHi() {
        echo "Hello I am an Employee.";
    }
}

$std = new Student('Frank', 17);
$std->sayHello();
echo "<br>";
echo "I am " . $std->get_age() . "years old.";
echo "<br>";
```

```php
$worker = new Employee('John', 35);

$worker->sayHi();

echo "<br>";

echo "I am " . $worker->get_age() . "years old.";
```

Inherited methods can be overridden by redefining the method in the child class. Note that the same method name should be used as in the example below:

```php
<?php


class Person

{

    public $name;

    public $age;

    public function __construct($name, $age) {

        $this->name = $name;

        $this->age = $age;
```

```
    }

    public function sayHello() {

        echo "Hello I am " . $this->name . " and I am " . $this->age . "
years old.";

    }

}



// Student is inherited from Person

class Student extends Person

{

    public $name;

    public $age;

    public $course;

    public function __construct($name, $age, $course) {

        $this->name = $name;

        $this->age = $age;

        $this->course = $course;

    }
```

```
    public function sayHello() {

        echo "Hello I am " . $this->name . " and I am " . $this->age . "
years old. I study " . $this->course;

    }

}



$std = new Student('Frank', 17, 'Biology');

$std->sayHello();
```

To prevent overriding methods of the parent class, the `final` keyword is used. the following script will result to an error.

```
<?php



class Person

{

    public $name;

    public $age;
```

```
    public function __construct($name, $age) {

        $this->name = $name;

        $this->age = $age;

    }

    final public function sayHello() {

        echo "Hello I am " . $this->name . " and I am " . $this->age . "
years old.";

    }

}



// Student is inherited from Person

class Student extends Person

{

    public $name;

    public $age;

    public $course;

    public function __construct($name, $age, $course) {

        $this->name = $name;
```

```
        $this->age = $age;

        $this->course = $course;

    }

    public function sayHello() {

        echo "Hello I am " . $this->name . " and I am " . $this->age . "
years old. I study " . $this->course;

    }

}

$std = new Student('Frank', 17, 'Biology');

$std->sayHello();
```

- Abstract Classes

An abstract class is a class that cannot be instantiated. Abstract classes can define abstract methods, which are methods without any body, only a definition:

```
abstract class MyAbstractClass {

    abstract public function doSomething($a,
$b);

}
```

Abstract classes should be extended by a child class which can then provide the implementation of these abstract methods. The following is an example of an implementation of the abstract class.

```php
<?php



abstract class Person {

    public $name;

    // the constructor function is public by default

    public function __construct($name) {

        $this->name = $name;

    }

    abstract public function sayHello() : string;

}



// Child classes

class Student extends Person {
```

```
    // inherited from parent class , needs to be defined but should follow
the same parameters

    public function sayHello() : string {

        return "Hello! I'm a student and my name is $this->name!";

    }

}



class Employee extends Person {

    // inherited from parent class , needs to be defined but should follow
the same parameters

    public function sayHello() : string {

        return "Hi! I'm an employee and my name is $this->name!";

    }

}



// create instances of the child classes

$std = new Student("Frank");

echo $std->sayHello();

echo "<br>";
```

```php
$worker = new Employee("Mark");

echo $worker->sayHello();
```