Web application development with PHP

Anis JEMMALI

EPITA Paris

April 2024

Define a Class

A class is defined by using the `class` keyword, followed by the name of the class and a pair of curly braces

```php
<?php
class MyClass {
        // code goes here...
}
?>
```

Define a Class

```php
<?php
class Student
{
        public readonly string $fname;
        public ?string $country;
        public function __construct(string $fn)
        {
                $this->fname = $fn;
        }
        public function setContry(string $c)
        {
                $this->country = $c;
        }

}
?>
```

Define a Class

- `readonly` modifier prevents modification of the property after initialization
- `?` (nullable) signifies that as well as the specified type, null can be passed as an argument
- `$this` keyword refers to the current object, and is only available inside methods
- If you create a `__construct()` function, PHP will automatically call this function when you create an object from a class.
- If you create a `__destruct()` function, PHP will automatically call this function at the end of the script.

Access Modifiers

- `public` - the property or method can be accessed from everywhere. This is default
- `protected` - the property or method can be accessed within the class and by classes derived from that class
- `private` - the property or method can ONLY be accessed within the class

Define Objects

Objects of a class are created using the `new` keyword.

```php
<?php
$me = new Student('Anis');
?>
```

PHP OOP - Inheritance

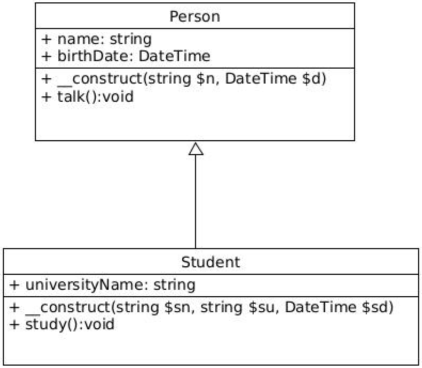
Inheritance in OOP = When a class derives from another class.

The child class will inherit all the public and protected properties and methods from the parent class. In addition, it can have its own properties and methods.

The child class will inherit all the public and protected properties and methods from the parent class. In addition, it can have its own properties and methods.

An inherited class is defined by using the `extends` keyword.

Inheritance - Example



Inheritance - Example

In the file Person.php we create the Person class

```
<?php
class Person
{
        public string $name;
        public DateTime $birthDate;
        public function __construct(string $n, DateTime $d)
        {
                $this->name = $n;
                $this->birthDate = $d;
        }
        public function talk():void{
                echo 'Hello␣world<br>';
        }
}
?>
```

Inheritance - Example

In the file `Student.php` we create the `Student` class

```php
<?php
require 'Person.php';
class Student extends Person
{

 public string $universityName;

 public function __construct(string $sn, string $su, DateTime $sd)
 {
         parent::__construct($sn, $sd);
         $this->universityName = $su;
 }

 public function study():void{
         echo "I'm student at $this->universityName <br>";
 }
}
```

Inheritance - Example

In the file using_class.php we will use Person and Student classes

```php
<?php
require 'Student.php';

$student = new Student('Abinash','EPITA', new DateTime('2004-01-01

$student->talk();
$student->study();
$student->name = 'Zelda';
$student->universityName = 'Sorbonne';
echo "$student->name<br>";
echo "$student->universityName<br>";
?>
```

Overriding Inherited Methods

Inherited methods can be overridden by redefining the methods (**use the same name**) in the child class.

Overriding Inherited Methods

Look at the example below. The `talk()` method in the child class (Student) will override the `talk()` method in the parent class (Person):

```php
<?php
class Person
{
        public function talk():void{
                echo 'Hello world<br>';
        }
}

class Student extends Person
{
        public function talk():void{
                echo "Hello students <br>";
        }
}
```

The final Keyword

The final keyword can be used to prevent class inheritance or to prevent method overriding.

The following example shows how to prevent class inheritance:

```php
<?php
final class Person {
        // some code
}

// will result in error
class Student extends Person {
        // some code
}
?>
```

The final Keyword

The following example shows how to prevent method overriding:

```php
<?php
class Person {
        final public function talk() {
                // some code
        }
}

class Student extends Person {
        // will result in error
        public function talk() {
                // some code
        }
}
?>
```

Class Constants

A class constant is declared inside a class with the `const` keyword.

⚠ A constant cannot be changed once it is declared.

Class constants are case-sensitive. However, it is recommended to name the constants in all uppercase letters.

We can access a constant from outside the class by using the class name followed by the scope resolution operator (`::`) followed by the constant name

Class Constants

The following example shows how to access a constant from outside the class:

```php
<?php
class Goodbye {
  const LEAVING_MESSAGE = "Thanks for attending this course!";
}

echo Goodbye::LEAVING_MESSAGE;
?>
```

Class Constants

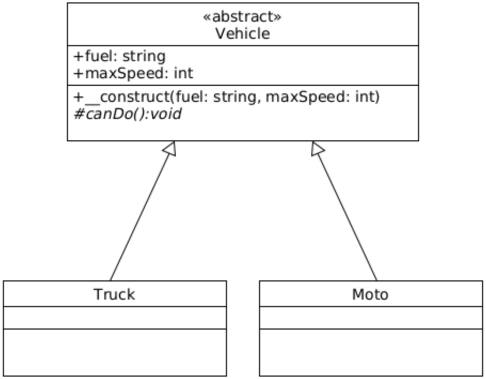
Or, we can access a constant from inside the class by using the self keyword followed by the scope resolution operator (::) followed by the constant name, like here:

```php
<?php
class Goodbye {
  const LEAVING_MESSAGE = "Thanks␣for␣attending␣this␣course!";

  public function byebye() {
      echo self::LEAVING_MESSAGE;
  }
}

$goodbye = new Goodbye();
$goodbye->byebye();
?>
```

Abstract Classes

- An abstract class is a class that contains at least one abstract method
- An abstract method is a method that is declared, but not implemented in the code
- The child class of an abstract class should implement the code of the abstract method

Abstract Classes



Abstract Classes

In the Vehicle.php file, we implement the Vehicle class.

```php
<?php
abstract class Vehicle
{
        public string $fuel;
        public int $maxSpeed;

        public function __construct(string $f, int $ms)
        {
                $this->fuel = $f;
                $this->maxSpeed = $ms;
        }

        abstract protected function canDo();
}
?>
```

Abstract Classes

In the Truck.php file, we implement the Truck class.

```php
<?php
require_once 'Vehicle.php';

class Truck extends Vehicle
{
        public function canDo()
        {
                echo 'I can carry loads!';
        }
}
?>
```

Abstract Classes

In the Moto.php file, we implement the Moto class.

```
<?php
require_once 'Vehicle.php';

class Moto extends Vehicle
{
        public function canDo(){
                echo 'I␣can␣do␣wheelies!';
        }
}
?>
```

Abstract Classes

```php
<?php
include 'Moto.php';

$m = new Moto('Petrol', 280);
$m->canDo();
?>
```

Abstract Classes

```php
<?php
include 'Truck.php';

$t = new Truck('Diesel', 180);
$t->canDo();
?>
```

Abstract Classes

When a child class is inherited from an abstract class, we have the following rules:

- The child class method must be defined with the same name and it redeclares the parent abstract method
- The child class method must be defined with the same or a less restricted access modifier
- The number of required arguments must be the same

Interfaces

Interfaces allow you to specify what methods a class should implement.

Interfaces are declared with the interface keyword

```
<?php
interface InterfaceName {
        public function someMethod1();
        public function someMethod2($name, $color);
        public function someMethod3() : string;
}
?>
```

Interfaces vs. Abstract Classes

Interface are similar to abstract classes.

The difference between interfaces and abstract classes are:

- Interfaces cannot have properties, while abstract classes can
- All interface methods must be public, while abstract class methods is public or protected
- All methods in an interface are abstract, so they cannot be implemented in code and the abstract keyword is not necessary
- Classes can implement an interface while inheriting from another class at the same time

Using Interfaces

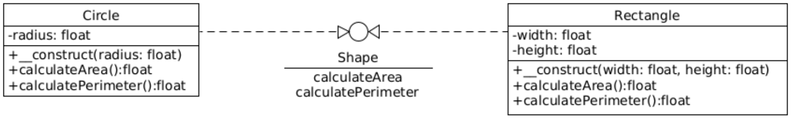
To implement an interface, a class must use the `implements` keyword.

A class that implements an interface must implement **all** of the interface's methods.

Example:
Let's suppose we're developing an application for managing geometric shapes. We have different types of shapes such as circles, rectangles, and triangles. Each shape needs to be able to calculate its area and perimeter, but the formula for each type of shape is different.

To model this in object-oriented programming, we can use interfaces.

Using Interfaces



Using Interfaces

In the file Shape.php we write the coode of the interface Shape

```php
<?php
interface Shape
{
        public function calculateArea():float;
        public function calculatePerimeter():float;
}
?>
```

Using Interfaces

In the file Circle.php we write the code of the class Circle which implement the interface Shape

```
<?php
require_once 'Shape.php';
class Clircle implements Shape
{
        private float $radius;
        public function __construct(float $radius)      {
                $this->radius = $radius;
        }
        public function calculateArea(): float  {
                return pi() * pow($this->radius, 2);
        }
        public function calculatePerimeter(): float     {
                return 2 * M_PI * $this->radius;
        }
}
```

Using Interfaces

In the file Rectangle.php we write the code of the class Rectangle

```php
<?php
require_once 'Shape.php';
class Rectangle implements Shape
{
        private float $height, $width;
        public function __construct(float $height, float $width){
                $this->height = $height;
                $this->width = $width;
        }
        public function calculateArea(): float{
                return $this->height * $this->width;
        }
        public function calculatePerimeter(): float     {
                return ($this->height+$this->width)*2;
        }
}
?>
```

Using Interfaces

Using the Circle class

```php
<?php
include 'Circle.php';

$C = new Clircle(5);
echo $C->calculateArea();
echo '<br>';
echo $C->calculatePerimeter();

?>
```

Using Interfaces

Using the Rectangle class

```php
<?php
include 'Rectangle.php';

$R = new Rectangle(5, 6);
echo $R->calculateArea();
echo '<br>';
echo $R->calculatePerimeter();
?>
```

Traits

⚠ PHP only supports single inheritance: a child class can inherit only from one single parent.

So, what if a class needs to inherit multiple behaviors?

OOP traits solve this problem.

Traits

- Traits are used to declare methods that can be used in multiple classes.
- Traits can have methods and abstract methods
- Traits methods can have any access modifier (public, private, or protected)

Traits

Traits are declared with the trait keyword:

```
<?php
trait TraitName {
        // some code...
}
?>
```

Example

Let's suppose we are going to develop an employee management application.
We distinguish full-time employees, part-time employees, and human resource managers.
Employees have common functionalities such as salary calculation and report generation.
The human resource manager also manages leaves and performance evaluations.

Example

To model this application, we will create:

- The "EmployeeTrait" trait, which will contain methods common to all types of employees.
- The "LeaveManagementTrait" trait to manage leaves.
- The "PerformanceEvaluationTrait" trait to manage performance evaluations.

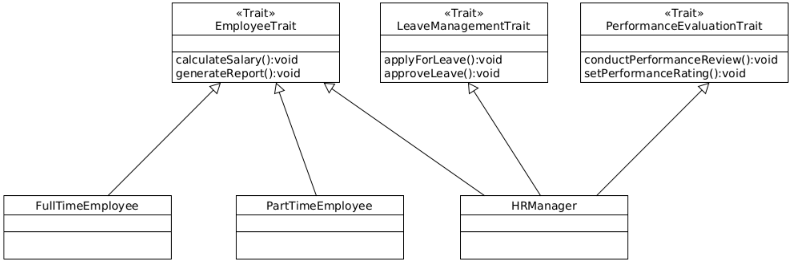
Example

We will then create the following classes:

- "FullTimeEmployee" to represent full-time employees.
- "PartTimeEmployee" to represent part-time employees.
- "HRManager" to represent human resource managers.

Example

Here is the corresponding UML diagram:



Example

```php
<?php
trait EmployeeTrait {
	public function calculateSalary() {
		//here comes code for calculating salary
	}

	public function generateReport() {
		//here comes code for generating report
	}
}
?>
```

Example

```php
<?php
trait LeaveManagementTrait {
        public function applyForLeave() {
                //Code of request for leave
        }

        public function approveLeave() {
                //Code of leave approval
        }
}
?>
```

Example

```php
<?php
trait PerformanceEvaluationTrait {
        public function conductPerformanceReview() {
                // Code of performance review
        }

        public function setPerformanceRating() {
                //Code to set performance rating
        }
}
?>
```

Example

```php
<?php
class FullTimeEmployee {
        use EmployeeTrait;
        //Other properties and methods specific
        //to a full-time employee
}
class PartTimeEmployee {
        use EmployeeTrait;
        //Other properties and methods specific
        //to a part-time employee.
}
class HRManager {
        use EmployeeTrait, LeaveManagementTrait,
                PerformanceEvaluationTrait;
        //Other properties and methods specific to an HR manager
}
?>
```

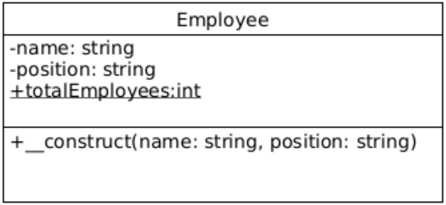
Static Methods

Static methods can be called directly - without creating an instance of the class
Static methods are declared with the static keyword
To access a static method use the class name, double colon (::), and the method name:

```
<?php
class MyClass
{
        public static function myMethod()
        {
                echo 'Hello␣world';
        }
}

MyClass::myMethod();
?>
```

Example

Let's suppose we're developing an employee management application where we need to track the total number of created employees.
Here the UML diagram of the class Employee



Example

```php
<?php
class Employee
{
  private string $name;
  private string $position;
  public static int $totalEmployee = 0;

  public function __construct(string $name, string $position)
  {
    $this->name = $name;
    $this->position = $position;
    self::$totalEmployee++;
  }
}
?>
```

Example

```php
<?php
$employee1 = new Employee("Mark␣Abramenko", "Developer");
$employee2 = new Employee("Natthanicha␣Vongjarit", "Manager");

echo Employee::$totalEmployee;
?>
```

PDO

PDO class represents a connection between PHP and a database server.

For more details check
`https://www.php.net/manual/en/class.pdo.php`

In this course, we will be using a MySql database.

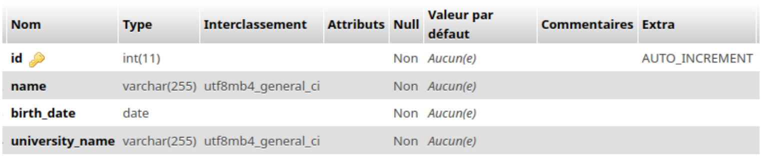
Connection

```php
<?php
$server      = 'localhost';
$db_name     = 'epita';
$user        = 'root';
$password    = '';

//Data Source Name, contains the information required
//to connect to the database.
$dsn = "mysql:dbname=$db_name;host=$server";

//Creates a PDO instance to represent a connection
//to the requested database.
$conn = new PDO($dsn, $user, $password);

//Close the Connection
$conn = null;
?>
```

Connection

```php
<?php
$server      = 'localhost';
$db_name     = 'epita';
$user        = 'root';
$password    = '';

$dsn = "mysql:dbname=$db_name;host=$server";

try {
    $conn = new PDO($dsn, $user, $password);
    echo "Connected␣successfully";
} catch (PDOException $ex) {
    echo 'Connection␣failed:␣'.$ex->getMessage();
}
//Close the Connection
$conn = null;
?>
```

We consider the 'students' table depicted below:



Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
id 🔑	int(11)			Non	*Aucun(e)*		AUTO_INCREMENT
name	varchar(255)	utf8mb4_general_ci		Non	*Aucun(e)*		
birth_date	date			Non	*Aucun(e)*		
university_name	varchar(255)	utf8mb4_general_ci		Non	*Aucun(e)*		

Insert Data

```
<?php
$user       = 'root';
$password   = '';
$dsn = "mysql:dbname=epita;host=localhost";
try {
    $conn = new PDO($dsn, $user, $password);
    $sql = "INSERT INTO
    students (name, birth_date , university_name)
    VALUES ('Galip Ata Hamdan', '2004-01-01', 'EPITA')";
  // use exec() because no results are returned
  $conn->exec($sql);
  $last_id = $conn->lastInsertId();
  echo "New record created successfully.
        Last inserted ID is: $last_id";
} catch (PDOException $ex) {
    echo $sql . "<br>" .$ex->getMessage();
}
//Close the Connection
```

Prepared Statements

```php
<?php
$user         = 'root';
$password     = '';
$dsn  = "mysql:dbname=epita;host=localhost";
$conn = new PDO($dsn, $user, $password);
// prepare sql and bind parameters
$stmt = $conn->prepare("INSERT INTO
  students (name, birth_date , university_name)
  VALUES (:name, :birth_date, :university)");
$stmt->bindParam(':name', $name);
$stmt->bindParam(':birth_date', $date);
$stmt->bindParam(':university', $university);
// insert a row
$name = "Assile Zeidan";
$date = "2005-01-01";
$university = "EPITA";
$stmt->execute();
?>
```

Select Data

```php
<?php
$user        = 'root';
$password    = '';
$dsn = "mysql:dbname=epita;host=localhost";
$conn = new PDO($dsn, $user, $password);

$stmt = $conn->prepare("select * from students");
$stmt->execute();
$stmt->setFetchMode(PDO::FETCH_ASSOC);

$allStudents = $stmt->fetchAll();
foreach($allStudents as $oneStudent){
    foreach($oneStudent as $k=>$v){
        echo $k .'= '.$v.' | ';
    }
    echo '<br>';
}
?>
```

Delete Data

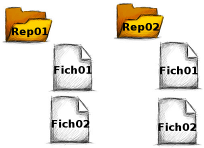
```php
<?php
$user       = 'root';
$password   = '';
$dsn = "mysql:dbname=epita;host=localhost";
$conn = new PDO($dsn, $user, $password);

$sql = "delete from students where id =2";

// use exec() because no results are returned
$conn->exec($sql);

echo "Record deleted successfully";
?>
```

Update Data

```php
<?php
$user       = 'root';
$password   = '';
$dsn = "mysql:dbname=epita;host=localhost";
$conn = new PDO($dsn, $user, $password);

$sql = "update␣students
set␣university_name='Sorbonne'␣where␣id>3";

// Prepare statement
$stmt = $conn->prepare($sql);

// execute the query
$stmt->execute();

// echo a message to say the UPDATE succeeded
echo $stmt->rowCount() . "␣records␣UPDATED␣successfully";
?>
```

Namespaces overview

In any operating system, two files with the same name cannot exist in the same directory



In addition, to access the file `Fich01` outside of the directory `/Rep01`, we must prepend the directory name to the file name using the directory separator to get `/Rep01/Fich01`
This same principle extends to namespaces in the programming world.

PHP Namespaces

Namespaces are qualifiers that solve two different problems:

- They allow for better organization by grouping classes that work together to perform a task
- They allow the same name to be used for more than one class

For example, you may have a set of classes which describe an HTML table, such as Table, Row and Cell while also having another set of classes to describe furniture, such as Table, Chair and Bed.

PHP Namespaces

In file1.php

```php
<?php
namespace Html;
class Table
{
  public static function generateTable(int $rows, int $cols)
  {
        echo '<table style="border:1px solid">';
        for ($i=1; $i <= $rows; $i++) {
          echo '<tr>';
          for ($j=1; $j <= $cols ; $j++) {
                echo "<td style='border:1px solid'>[$i,$j]</td>";
          }
          echo '</tr>';
        }
        echo '</table>';
  }
}
```

PHP Namespaces

In `file2.php`

```
<?php
namespace Furniture;

class Table
{
 private int $height;
 private int $width;
 private int $length;

 public function __construct(int $height, int $width, int $length)
 {
   $this->height = $height;
   $this->width = $width;
   $this->length = $length;
   echo $height.'cm␣Hx'.$width.'cm␣Wx'.$length.'cm␣L␣created';
  }
}
```

PHP Namespaces

In test.php

```php
<?php
include 'file1.php';
include 'file2.php';

use Furniture\Table;
use Html\Table as HT;

$t = new Table(1,2,3);

HT::generateTable(2,3);
?>
```