Magic methods

بص يا صديقي، في PHP، في حاجات اسمها ميثودات سحرية (Magic Methods)، اللي بتعمل حاجات خاصة لما ال object بيعمل حاجة معينة.

بتبدأ أسماء الميثودات السحرية بـ double underscore (__)، واللي بتبدأ بالـ double underscore دي بتبقى خاصة بال magic methods في PHP.

من قبل كدا، اتعرفت على الميثود السحرية __construct() و __destruct()، اللي بتتشغل تلقائياً لما تبقى تخلق object أو تحذفه.

بجانب الميثودات دي، في PHP في ميثودات سحرية تانية زي:

- __get(): بيتشغل لما تحاول تجيب قيمة attribute من object بس مفيش attribute باسم اللي اتحاول أجيب قيمته.
- __set(): بيتشغل لما تحاول تغير قيمة attribute في object.
- __call(): بيتشغل لما تحاول تستدعي method مش موجود في ال object بتاعك.
- __toString(): بيتشغل لما تحاول تحول object إلى string، مثلاً لما تجيب قيمة معينة في echo statement.
- __clone(): بيتشغل لما تحاول تنسخ object باستخدام الـ clone keyword.
- __sleep() و __wakeup(): بيتشغلوا لما تحاول تحفظ وتحمل object على الملف أو على الـ session.

PHP __set() method

لما بتحاول تغير قيمة attribute في الـ object بتاعك، وال attribute ده إما مش موجود أو مش متاح ليك تغير قيمته، بيتم استدعاء الميثود __set() تلقائياً.

الميثود دي بتستقبل اسم ال attribute اللي بتحاول تغير قيمته، والقيمة الجديدة اللي عايز تحطها فيه. وده شكل الميثود اللي بتكتبه:

void : (mixed $value , public __set (string $name

مثال على استخدام الميثود دي:

```
public function __set($name, $value) {
 if ($name === 'color') {
```

```
    $this->color = $value;
  } else {
    echo "Can't set property $name";  }    }
```

PHP __get() method

لما بتحاول تقرأ قيمة attribute في الـ object بتاعك، وال attribute ده إما مش موجود أو مش متاح ليك تقرأ قيمته، بيتم استدعاء الميثود السحرية __get() تلقائياً.

الميثود دي بتستقبل اسم ال attribute اللي بتحاول تقرأ قيمته. وده شكل الميثود اللي بتكتبه:

mixed : (public __get (string $name

مثال على استخدام الميثود دي:

```
public function __get($name) {
  if ($name === 'color') {
    return $this->color;
  } else {
    echo "Can't get property $name";   }  }
```

PHP __call() method

الـ method السحريّة __call() بتتشغل تلقائيًا لما بتحاول تستدعي method مش موجودة في الـ class بتاعك، أو الـ method دي private أو protected ومش متاحة ليك.

الـ method دي بتستقبل اسم الـ method اللي بتحاول تستدعيه، و array فيه ال arguments اللي بتبعتها لل method اللي بتحاول تستدعيه.

والميثود دي مفيدة جدًا في حالة عاوز تكتب wrapper class تعمل wrap ل API موجود براها.

PHP __callStatic() method

الـ method السحرية __callStatic() بتشتغل تلقائيًا لما تحاول تستدعي static method مش موجودة في الـ class بتاعك أو متاحة فقط للـ instance methods.

الـ method دي بتستقبل اسم الـ static method اللي بتحاول تستدعيه، و array فيه arguments اللي بتبعتها للـ method.

```
public static function __callStatic(string $method, array $parameters)
{
   if (!array_key_exists($method, self::$methods)) {
      throw new Exception('The ' . $method . ' is not supported.');
   }
   return call_user_func_array(self::$methods[$method], $parameters);
} }
```

Procedural Programming

يا شباب، في برمجة اسمها Procedural Programming. الفكرة إننا بنستخدم كتل من الكود اسمها procedures أو routines، اللي بيحتوي على مجموعة من التعليمات المتتالية اللي بتنفذ في وقت التنفيذ.

في مميزات للـ Procedural Programming زي:

- الـ Scoping اللي بيضبط مكان الـ (objects entities) داخل البرنامج عشان يمنع الوصول للمتغيرات من غير المكان المحدد ليها.
- الـ Modularity اللي بينظم الكود وبيخليه reusable وبيسهل تبادل الكود.
- وكمان الـ Parameter Passing اللي بيحدد إزاي نمرر البارامترز للـ procedures.

بس فيه أيضًا عيوب للـ Procedural Programming، زي:

- ان الـ data ممكن تبقى exposed للكل ومحدش يعرف يحميها، فبتقدر تعدل فيها بدون إذن ودي بتسبب مشاكل كبيرة.
- وكمان فيه تركيز كبير على الـ procedures، وبيبقى صعب تخيل التركيز على تنظيم الـ data في البرنامج.

Object-Oriented Programming (OOP)

في الـ OOP بنستخدم حاجات اسمها objects عشان تمثل حاجات في الدنيا الحقيقية زي العربية اللي بتمشي وتقف وعندها اسم وسرعة.

وفي المبادئ الرئيسية بتاعته، التجريد بيخلي الناس بره الـ class يتعاملوا معاه من غير ما يعرفوا تفاصيل التنفيذ. والـ encapsulation بيجمع الكود والـ variables المرتبطة بيه في مكان واحد وبيتم الوصول لها بس عن طريق getter و setter methods، وده بيساعد على اخفاء بعض المعلومات عن الناس اللي مش لازم يشوفوها.

وبعدين فيه حاجة اسمها التوريث، اللي بيسمح لـ subclass اللي هي الـ child class بأخذ الـ properties والـ behaviors من الـ superclass اللي هي الـ parent class. فـ subclass بيقدر يوصل لـ methods والـ attributes اللي في الـ superclass وكمان يضيف حاجات جديدة فيه. على سبيل المثال، الـ BMW car بيورث خصائص الـ CAR، والـ CAR بيورث خصائص الـ VEHICLE.

وبعدين فيه حاجة اسمها الـ polymorphism اللي بيسمح لنفس الـ object انه يسلك سلوك مختلف على حسب السياق. وفي الـ OOP بتقدر تستخدم نوعين من الـ polymorphism، الأول هو الـ runtime polymorphism اللي بيسمح لـ function انه يعدل على تنفيذ method في الـ superclass، والثاني هو الـ compile-time polymorphism اللي بيسمح لـ function انه يكون عنده أكتر من نسخة على حسب signature. على سبيل المثال، الـ person في نفس الوقت هو ابن وأب وزوج، فـ نفس الـ person بيتصرف بطريقتين مختلفين وفقاً للسياق.

Functional Programming

الفكرة في البرمجة الوظيفية إن الوحدات الأساسية بتكون الـ "functions"، وده لأن في البرمجة الوظيفية بيعاملوا الـ functions زي أي متغير تاني، يعني بيقدروا يعرفوا الـ function داخل متغير، ويبعتوها كوسيط، ويرجعوها من function تانية زي أي متغير تاني. الفكرة إنهم بيشيلوا الفرق بين الـ functions والـ data.

وفي مسطلحات :

- Side effect: دا لما function بتعدل على متغير خارج نطاقها، برامج البرمجة الوظيفية اللي بتقدم دعم لل side effects بتتسمى impure functional languages.
- Immutability: لما تربط قيمة بمتغير ميتقدرش يتغير في البرنامج كله.
- Functional composition: الفكرة إنك تطبق functions على functions تانية وتربطهم مع بعض، زي مثلا .f(g(x))
- Ad Hoc Polymorphism: دا آلية بتسمح للكومبايلر/الإنتربريتر إنه يتفرق بين إصدارات function مختلفة على حسب نوع البارامترات اللي بيتم إستدعاؤها بيها. دا بيتعرف أيضًا بال overloading، اللي بيعني إن في functions ممكن تكون فيها إصدارات مختلفة بناءً على نوع البارامترات أو التوقيعات.
- Subtyping: طريقة بتحدد العلاقة بين نوع ونوع تاني، مثلاً ال Cat وال Dog بتكونوا أنواع فرعية لل Animal.
- First-class functions: دول functions بيتعاملوا زي أي متغير، يعني مثلاً بتقدر تمرر function كبارامتر لـ function تانية، وترجع function من function تانية، وتعرف function داخل متغير، زي الأمثلة اللي في النص.
- Higher-order functions: دول functions بيتأخذوا functions تانية كبارامترات أو بيسترجعوا functions

Cloning in php (shallow copy and deep copy)

في PHP، النسخ (Cloning) هي عملية إنشاء نسخة جديدة من الكائن (Object) الأصلي. يمكن استخدام النسخ لإنشاء نسخة مستقلة من الكائن الأصلي، لتعديل النسخة دون التأثير على الأصل.

هناك نوعين من النسخ في PHP: النسخ الضحل (Shallow Copy) والنسخ العميق Deep Copy).)

- النسخ الضحل (Shallow Copy): يتم فيها إنشاء نسخة جديدة من الكائن الأصلي، ولكن تشير المتغيرات المرجعية في الكائن الجديد إلى نفس الكائنات المرجعية في

الكائن الأصلي. بمعنى آخر، فإنه يتم إنشاء نسخة جديدة فارغة من الكائن، ويتم تعبئتها بالمتغيرات المرجعية الخاصة بالكائن الأصلي.

لإجراء (Shallow Copy) ، يمكن استخدام الدالة المدمجة "clone()" والتي تنشئ نسخة ضحلة من الكائن الأصلي.

- النسخ العميق (Deep Copy): يتم فيها إنشاء نسخة جديدة من الكائن الأصلي، ولكن المتغيرات المرجعية في الكائن الجديد تشير إلى نسخ جديدة من الكائنات المرجعية في الكائن الأصلي. يتم إنشاء نسخ جديدة من جميع الكائنات المرجعية داخل الكائن الأصلي، بحيث يتم إنشاء شجرة كاملة من النسخ الجديدة.

لإجراء النسخ العميق، يتعين عليك تحديد الخواص التي يجب نسخها، وإنشاء نسخة جديدة من كل خاصية تحتوي على كائن.

مثال (:Shallow Copy

```
class Person {
    public $name;
    public $age;
  }
  $person1 = new Person();
  $person1->name = "John";
  $person1->age = 30;
   $person2 = clone $person1;
  $person2->name = "Jane";
   echo $person1->name; // output: Jane
```

مثال (:Shallow Copy

```
class Person {
  public $name;
  public $age;
}
$person1 = new Person();
```

```
$person1->name = "John";
$person1->age = 30;
$person2 = unserialize(serialize($person1));
$person2->name = "Jane";
echo $person1->name; // output: John
```

Self vs This vs Static

طيب يلا نشوف الفرق بين self و this و static في البرمجة:

- self: دي بتشير إلى الـ class نفسها اللي بيشتغل فيها ال method، يعني ال object اللي بيتم إنشاؤه من ال class دي هيكون ليه نفس الـ properties والـ methods اللي بيستخدمهم الـ method اللي بيشتغل فيها.
- this: دي بتشير إلى الـ object الحالي اللي بيتم استخدامها في الـ method اللي بيشتغل فيها، يعني الـ properties والـ methods اللي بيستخدمهم الـ method ده هيكونوا للـ object الحالي بتاعه.
- static: دي بتشير إلى الـ class نفسها اللي بيشتغل فيها الـ method، بس الفرق إن الـ properties والـ methods اللي بيستخدمهم الـ method ده مش مرتبطين بـ object معين، يعني ممكن تستخدم الـ method ده بدون ما تنشئ object من الـ class دي.