Что такое объектно-ориентированное программирование (ООП)?

Объектно-ориентированное программирование, обычно называемое ООП - это подход, который вам помогает разрабатывать сложные приложения таким образом, чтобы они легко поддерживались и масштабировались в течение длительного времени. В мире ООП реальные понятия `Person`, `Car` или `Animal` рассматриваются как объекты. В объектно-ориентированном программировании вы взаимодействуете с вашим приложением, используя объекты. Это отличается от процедурного программирования, когда вы, в первую очередь, взаимодействуете с функциями и глобальными переменными.

В ООП существует понятие **«class»**, использываемое для моделирования или сопоставления реального понятия с шаблоном данных **(свойств)** и функциональных возможностей **(методов)**. **«Object»** - это экземпляр класса, и вы можете создать несколько экземпляров одного и того же класса. Например, существует один класс `Person`, но многие объекты person могут быть экземплярами этого класса - `dan`, `zainab`, `hector` и т. д.

Например, для класса Person могут быть `name`, `age` и `phoneNumber`. Тогда у каждого объекта person для этих свойств будут свои значения.

Вы также можете определить в классе методы, которые позволяют вам манипулировать значениями свойств объекта и выполнять операции над объектами. В качестве примера вы можете определить метод `save`, сохраняющий информацию об объекте в базе данных.

Что такое класс PHP?

Класс - это шаблон, который представляет реальное понятие и определяет свойства и методы данного понятия. В этом разделе мы обсудим базовую анатомию типичного класса PHP.

Лучший способ понять новые концепции - показать это на примере. Итак, давайте рассмотрим в коде класс `Employee`, который представляет объект служащего.

```php
<?php
class Employee
{
  private $first_name;
  private $last_name;
  private $age;
```

```
7
8     public function __construct($first_name, $last_name, $age)
9     {
10      $this->first_name = $first_name;
11      $this->last_name = $last_name;
12      $this->age = $age;
13    }
14
15    public function getFirstName()
16    {
17      return $this->first_name;
18    }
19
20    public function getLastName()
21    {
22      return $this->last_name;
23    }
24
25    public function getAge()
26    {
27      return $this->age;
28    }
29  }
30  ?>
```

Оператор `class Employee` в первой строке определяет класс `Employee`. Затем мы продолжаем объявлять свойства, конструктор и другие методы класса.

Свойства класса в PHP

Вы можете думать о свойствах класса как о переменных, которые используются для хранения информации об объекте. В приведенном выше примере мы определили три свойства - `first_name`, `last_name` и `age`. В большинстве случаев доступ к свойствам класса осуществляется через созданные объекты.

Эти `private` свойства могут быть доступны только внутри класса. Данный подход - самый безопасный уровень доступа к свойствам. Позже в уроке мы обсудим различные уровни доступа к свойствам и методам класса.

Конструкторы для классов PHP

Конструктор - это специальный метод класса, который вызывается автоматически при инстанцинации объекта. В следующих разделах мы увидим, как инстанцировать объекты, но сейчас вам нужно просто знать, что конструктор используется для инициализации свойств объекта при создании объекта.

Вы можете определить конструктор с помощью метода `__construct`.

Методы для классов PHP

Давайте подумаем о методах класса как о функциях, которые выполняют определенные действия, связанные с объектами. В большинстве случаев они используются для доступа и управления свойствами объекта и выполнения связанных операций.

В приведенном выше примере мы определили метод `getLastName`, который возвращает фамилию, связанную с объектом.

В следующем разделе мы увидим, как создавать объекты класса `Employee`.

Что такое объект в PHP?

В предыдущем разделе мы обсудили базовую структуру PHP класса. Теперь, когда вы хотите использовать класс, вам нужно его инстанцировать, конечным результатом чего будет объект. Таким образом, мы можем думать о классе как о проекте, а объект - это реальная вещь, над которой вы можете работать.

В контексте класса `Employee`, созданного в предыдущем разделе, давайте посмотрим, как создать понятие объекта этого класса.

```
1  <?php
2  $objEmployee = new Employee('Bob', 'Smith', 30);
3
4  echo $objEmployee->getFirstName(); // print 'Bob'
5  echo $objEmployee->getLastName(); // prints 'Smith'
6  echo $objEmployee->getAge(); // prints '30'
7  ?>
```

Если вы хотите создать понятие объекта любого класса вместе с его именем, нужно использовать ключевое слово `new`, и в итоге вы получите новое понятие объекта этого класса.

Если класс определил метод `__construct` и ему требуются аргументы, вам нужно передать эти аргументы при создании экземпляра объекта. В нашем случае конструктор класса `Employee` требует три аргумента, и поэтому мы их передали, когда создавали объект `$objEmployee`. Как мы говорилось ранее, метод `__construct` вызывается автоматически при инстанциации объекта. Затем мы вызвали методы класса для объекта `$objEmployee`, чтобы запечатлить информацию, инициализированную во время создания объекта. Конечно же, вы можете создать несколько объектов одного класса, как это показано в следующем фрагменте.

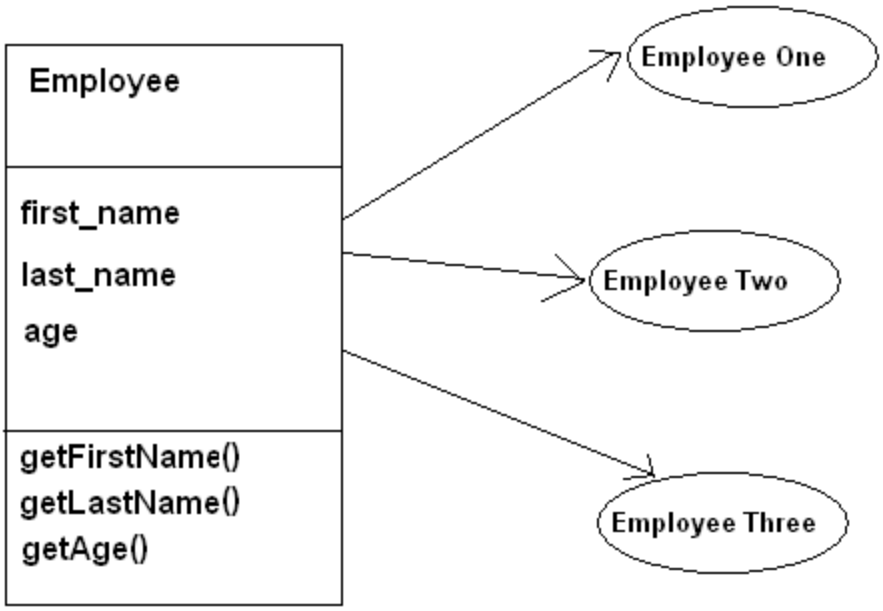
```php
<?php
$objEmployeeOne = new Employee('Bob', 'Smith', 30);

echo $objEmployeeOne->getFirstName(); // prints 'Bob'
echo $objEmployeeOne->getLastName(); // prints 'Smith'
echo $objEmployeeOne->getAge(); // prints '30'

$objEmployeeTwo = new Employee('John', 'Smith', 34);

echo $objEmployeeTwo->getFirstName(); // prints 'John'
echo $objEmployeeTwo->getLastName(); // prints 'Smith'
echo $objEmployeeTwo->getAge(); // prints '34'
?>
```

Следующее изображение является графическим представлением класса Employee и некоторых его экземпляров.



Проще говоря, класс - это проект, который вы можете использовать для создания структурированных объектов.

Инкапсуляция

В предыдущем разделе мы обсуждали, как создавать экземпляры объектов класса `Employee`. Интересно отметить, что сам объект `$objEmployee` объединяет свойства и методы класса. Другими словами, он скрывает эти детали от остальной части программы. В мире ООП это называется инкапсуляцией данных.

Инкапсуляция является важным аспектом ООП, позволяющий ограничить доступ к определенным свойствам или методам объекта. А это побуждает нас обсудить другую тему: уровень доступа.

Уровни доступа

Если вы определяете свойство или метод в классе, тогда вы можете объявить, что он имеет один из этих трех уровней доступа - *public*, `private`, или `protected`.

Доступ public

Когда вы объявляете свойство или метод как public, к нему можно получить доступ из любого места вне класса. Значение открытого свойства можно изменить из любого участка вашего кода.

Давайте рассмотрим на пример, чтобы понять как создать уровень публичного доступа.

```php
<?php
class Person
{
  public $name;

  public function getName()
  {
    return $this->name;
  }
}

$person = new Person();
$person->name = 'Bob Smith';
echo $person->getName(); // prints 'Bob Smith'
?>
```

Как вы видете в приведенном выше примере, мы объявили общедоступное свойство `name`. Следовательно, вы можете установить его из любого места вне класса, что мы и сделали.

Доступ private

В случае если вы объявляете свойство или метод `private`, доступ к ним можно получить только из класса. Это означает, что вам нужно определить методы получения и установки, чтобы получить и установить значение этого свойства. Опять же, давайте пересмотрим предыдущий пример, чтобы понять уровень частного доступа.

```php
<?php
class Person
{
  private $name;

  public function getName()
  {
    return $this->name;
```

```
9     }
10
11    public function setName($name)
12    {
13      $this->name = $name;
14    }
15  }
16
17  $person = new Person();
18  $person->name = 'Bob Smith'; // Throws an error
19  $person->setName('Bob Smith');
20  echo $person->getName(); // prints 'Bob Smith'
21  ?>
```

Если вы захотите получить доступ к private свойству вне класса, он выдаст фатальную ошибку `Cannot access private property Person::$name`. Таким образом, вам нужно установить значение private свойства с помощью метода setter, как мы это сделали с помощью метода `setName`.

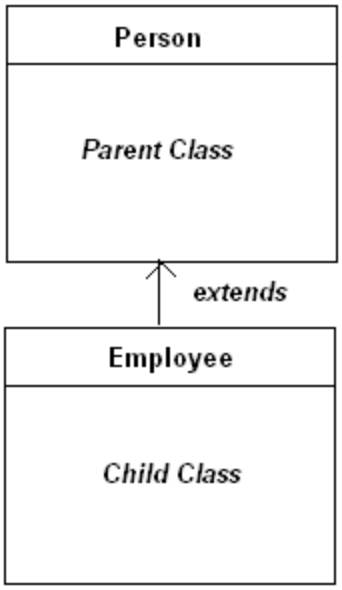
Могут возникать веские причины, из-за которых вы захотите установить private свойство. Например, возможно, для предпринятия какого-то действия (скажем, обновить базу данных или перерисовать шаблон), если это свойство меняется. В этом случае вы можете определить метод установки и управление любой специальной логикой для изменения свойства.

Доступ protected

Наконец, когда вы объявляете свойство или метод `protected`, к ним может обращаться тот же класс, который их определил, или классы, которые наследуют рассматриваемый класс. Мы обсудим наследование в следующем разделе, поэтому вернемся к уровню защищенного доступа чуть позже.

Наследование

Наследование является важным аспектом парадигмы объектно-ориентированного программирования, которая позволяет наследовать свойства и методы других классов, расширяя их. Класс, который наследуется, называется **родительским классом**, а класс, который наследует другой класс, называется **дочерним классом**. Когда вы создаете экземпляр объекта дочернего класса, он также наследует свойства и методы родительского класса. Давайте посмотрим на следующий скриншот, чтобы понять концепцию наследования.



В примере выше класс `Person` является родительским классом, а класс Employee расширяет или наследует класс `Person`, поэтому и называется дочерним классом.
Давайте попробуем разобраться на реальном примере, чтобы понять, как это работает.

```php
<?php
class Person
{
  protected $name;
  protected $age;

  public function getName()
  {
    return $this->name;
  }

  public function setName($name)
  {
```

```
14      $this->name = $name;
15    }
16
17    private function callToPrivateNameAndAge()
18    {
19      return "{$this->name} is {$this->age} years old.";
20    }
21
22    protected function callToProtectedNameAndAge()
23    {
24      return "{$this->name} is {$this->age} years old.";
25    }
26  }
27
28  class Employee extends Person
29  {
30    private $designation;
31    private $salary;
32
33    public function getAge()
34    {
35      return $this->age;
36    }
37
38    public function setAge($age)
39    {
40      $this->age = $age;
41    }
42
43    public function getDesignation()
44    {
45      return $this->designation;
```

```
46      }
47
48      public function setDesignation($designation)
49      {
50          $this->designation = $designation;
51      }
52
53      public function getSalary()
54      {
55          return $this->salary;
56      }
57
58      public function setSalary($salary)
59      {
60          $this->salary = $salary;
61      }
62
63      public function getNameAndAge()
64      {
65          return $this->callToProtectedNameAndAge();
66      }
67  }
68
69  $employee = new Employee();
70
71  $employee->setName('Bob Smith');
72  $employee->setAge(30);
73  $employee->setDesignation('Software Engineer');
74  $employee->setSalary('30K');
75
76  echo $employee->getName(); // prints 'Bob Smith'
77  echo $employee->getAge(); // prints '30'
```

```
78 echo $employee->getDesignation(); // prints 'Software Engineer'
79 echo $employee->getSalary(); // prints '30K'
80 echo $employee->getNameAndAge(); // prints 'Bob Smith is 30 years old.'
81 echo $employee->callToPrivateNameAndAge(); // produces 'Fatal Error'
82 ?>
```

Здесь важно отметить, что класс `Employee` использовал для наследования класса `Person` ключевое слово `extends`. Теперь класс `Employee` может получить доступ ко всем свойствам и методам класса `Person`, объявленные как public или protected. (Он не может получить доступ к ntv, которые объявлены как private.)

В примере выше объект `$employee` может получить доступ к методам `getName` и `setName`, которые определены в классе `Person`, поскольку они объявлены как public.

Затем мы обратились к методу `callToProtectedNameAndAge`, используя метод `getNameAndAge`, определенный в классе `Employee`, поскольку он объявлен как protected. Наконец, объект `$employee` не может получить доступ к методу `callToPrivateNameAndAge` класса Person, поскольку он объявлен как `private`.

С другой стороны, вы можете использовать объект `$employee` для установки свойства `age` класса `Person`, как мы это делали в методе `setAge`, который определен в классе `Employee`, поскольку свойство `age` объявлено как protected.

И так, это было краткое введение в наследование. Оно помогает сократить дублирование кода и, следовательно, способствует его повторному использованию.

Полиморфизм

Полиморфизм - еще одна важная концепция в мире объектно-ориентированного программирования, которая относится к способности по-разному обрабатывать объекты в зависимости от их типов данных.

Например, в контексте наследования, если дочерний класс хочет изменить поведение метода родительского класса, он может переопределить этот метод. Это называется переопределением метода. Давайте быстро рассмотрим реальный пример, чтобы понять концепцию переопределения метода.

```
1 <?php
```

```
class Message
{
  public function formatMessage($message)
  {
    return printf("<i>%s</i>", $message);
  }
}

class BoldMessage extends Message
{
  public function formatMessage($message)
  {
    return printf("<b>%s</b>", $message);
  }
}

$message = new Message();
$message->formatMessage('Hello World'); // prints '<i>Hello World</i>'

$message = new BoldMessage();
$message->formatMessage('Hello World'); // prints '<b>Hello World</b>'
?>
```

Как видите, мы изменили поведение метода `formatMessage`, переопределив его в классе `BoldMessage`. Важно то, что сообщение форматируется по-разному в зависимости от типа объекта, будь то экземпляр родительского или дочернего класса.

(Некоторые объектно-ориентированные языки также имеют своего рода перезагрузку методов, которая позволяет определять несколько методов класса с одним и тем же именем, но с разным количеством аргументов. Это не поддерживается напрямую в PHP, но существуют несколько обходных путей для достижения аналогичной функциональности.)