Классы и объекты

Основы

class

Каждое определение класса начинается с ключевого слова class, затем следует имя класса, и далее пара фигурных скобок, которые заключают в себе определение свойств и методов этого класса.

Именем класса может быть любое слово, при условии, что оно не входит в список зарезервированных слов PHP, начинается с буквы или символа подчеркивания и за которым следует любое количество букв, цифр или символов подчеркивания.

Класс может содержать собственные константы, переменные (называемые свойствами) и функции (называемые методами).

```php
<?php
    class SimpleClass {
        public $var = 'default value';

        public function displayVar() {
            echo $this->var;
        }
    }
?>
```

Псевдо-переменная $this доступна в том случае, если метод был вызван в контексте объекта. $this является ссылкой на вызываемый объект. Обычно это тот объект, которому принадлежит вызванный метод, но может быть и другой объект, если метод был вызван статически из контекста другого объекта.

new

Для создания экземпляра класса используется директива new. Новый объект всегда будет создан, за исключением случаев, когда он содержит конструктор, в котором определен вызов исключения в случае ошибки. Рекомендуется определять классы до создания их экземпляров (в некоторых случаях это обязательно).

Properties and methods

Свойства и методы класса живут в разделенных "пространствах имен", так что возможно задавать свойство и метод с одним и тем же именем. Ссылки как на свойства, так и на методы имеют одинаковую нотацию, и получается, что получите вы доступ к свойству или же вызовете метод - определяется контекстом использования.

extends

Класс может наследовать методы и свойства другого класса используя ключевое слово extends при его описании. Невозможно наследовать несколько классов, один класс может наследовать только один базовый класс.

Наследуемые методы и свойства могут быть переопределены (за исключением случаев, когда метод класса объявлен как final) путем объявления их с теми же именами, как и в родительском классе. Существует возможность доступа к переопределенным методам или статическим методам путем обращения к ним через parent::

::class

Можно использовать ключевое слово class для разрешения имени класса. С помощью конструкции ClassName::class можно получить строку с абсолютным именем класса ClassName. Обычно это довольно полезно при работе с классами, использующими пространства имен.

```php
<?php
    namespace NS {
        class ClassName {}

        echo ClassName::class;
    }

    // NS\ClassName
?>
```

Свойства

Переменные, которые являются членами класса, называются "свойства". Также их называют, используя другие термины, такие как "аттрибуты" или "поля". Они определяются с помощью ключевых слов public, protected, или private, следуя правилам правильного описания переменных. Это описание может содержать инициализацию, но инициализация должна применяться для константных значений - то есть, переменные должны быть вычислены во время компиляции и не должны зависеть от информации программы во время выполнения для их вычисления.

В пределах методов класса доступ к нестатическим свойствам может быть получен с помощью -> (объектного оператора): $this->property (где property - имя свойства). Доступ к статическим свойствам может быть получен с помощью :: (двойного двоеточия): self::$property.

Псевдо-переменная $this доступна внутри любого метода класса, когда этот метод вызывается в пределах объекта. $this - это ссылка на вызываемый объект (обычно, объект, которому принадлежит метод, но возможно и другого объекта, если метод вызван статически из контекста второго объекта).

Константы классов

Константы также могут быть объявлены и в пределах одного класса. Отличие переменных и констант состоит в том, что при объявлении последних или при обращении к ним не используется символ $. Область видимости констант, по умолчанию, public.

Значение должно быть неизменяемым выражением, не (к примеру) переменной, свойством или вызовом функции.

Константы класса задаются один раз для всего класса, а не отдельно для каждого созданного объекта этого класса.

Автоматическая загрузка классов

Большинство разработчиков объектно-ориентированных приложений используют такое соглашение именования файлов, в котором каждый класс хранится в отдельно созданном для него файле. Одной из наиболее при этом досаждающих деталей является необходимость писать в начале каждого скрипта длинный список подгружаемых файлов.

Функция spl_autoload_register() позволяет зарегистрировать необходимое количество автозагрузчиков, для автоматической загрузки ранее не определенных классов и интерфейсов. Вызов этой функции - последний шанс для интерпретатора загрузить класс прежде, чем он закончит выполнение скрипта с ошибкой.

то время как функция __autoload() также может быть использована для автоматической загрузки классов и интерфейсов, следует отдать предпочтение spl_autoload_register(), потому, что она предоставляет гораздо более гибкую альтернативу, позволяя регистрировать необходимое количество автозагрузчиков, например для корректной работы сторонних библиотек. В связи с этим, использование __autoload() не рекомендуется и она может быть объявлена устаревшей в будущем.

В этом примере функция пытается загрузить классы *MyClass1* и *MyClass2* из файлов *MyClass1.php* и *MyClass2.php* соответственно.

```php
<?php
spl_autoload_register(function ($class_name) {
    include $class_name . '.php';
});

$obj  = new MyClass1();
$obj2 = new MyClass2();
?>
```

Конструкторы и деструкторы

PHP позволяет объявлять методы-**конструкторы**. Классы, в которых объявлен метод-констуктор, будут вызывать этот метод при каждом создании нового объекта, так что это может оказаться полезным, например, для инициализации какого-либо состояния объекта перед его использованием.

Конструкторы, определенные в классах-родителях не вызываются автоматически, если класс-потомок определяет собственный конструктор. Чтобы вызвать конструктор, объявленный в родительском классе, следует обратиться к методу parent::__construct() внутри конструктора класса-потомка. Если в классе-потомке не определен конструктор, то он может наследоваться от родительского класса как обычный метод (если он не определен как приватный).

```php
<?php
    class BaseClass {
        function __construct() {
            print "BaseClass constructor";
        }
    }

    class SubClass extends BaseClass {
        function __construct() {
            parent::__construct();
            print "SubClass constructor";
        }
    }

    class OtherSubClass extends BaseClass {
        // inherits BaseClass's constructor
    }

    $obj = new BaseClass();
    $obj = new SubClass();
    $obj = new OtherSubClass();

    /*
        BaseClass constructor
        BaseClass constructor
        SubClass constructor
        BaseClass constructor
    */
?>
```

Деструктор будет вызван при освобождении всех ссылок на определенный объект или при завершении скрипта (порядок выполнения деструкторов не гарантируется).

```php
<?php
    class MyDestructableClass {
        function __construct() {
            print "Constructor";
            $this->name = "MyDestructableClass";
        }

        function __destruct() {
            print "Removing " . $this->name;
        }
    }

    $obj = new MyDestructableClass();

    /*
        Constructor
        Removing MyDestructableClass
    */
?>
```

Как и в случае с конструкторами, деструкторы, объявленные в родительском классе, не будут вызваны автоматически. Для вызова деструктора, объявленном в классе-родителе, следует обратиться к методу parent::__destruct() в теле деструктора-потомка. Также класс-потомок может унаследовать деструктор из родительского класса, если он не определен в нем.

Область видимости

Область видимости свойства, метода может быть определена путем использования следующих ключевых слов в объявлении: public, protected или private. Доступ к свойствам и методам класса, объявленным как public (общедоступный), разрешен отовсюду. Модификатор protected (защищенный) разрешает доступ наследуемым и родительским классам. Модификатор private (закрытый) ограничивает область видимости так, что только класс, где объявлен сам элемент, имеет к нему доступ.

Область видимости свойства

Свойства класса должны быть определены через модификаторы public, private, или protected. Свойства, где определение модификатора отсутствует, определяются как public.

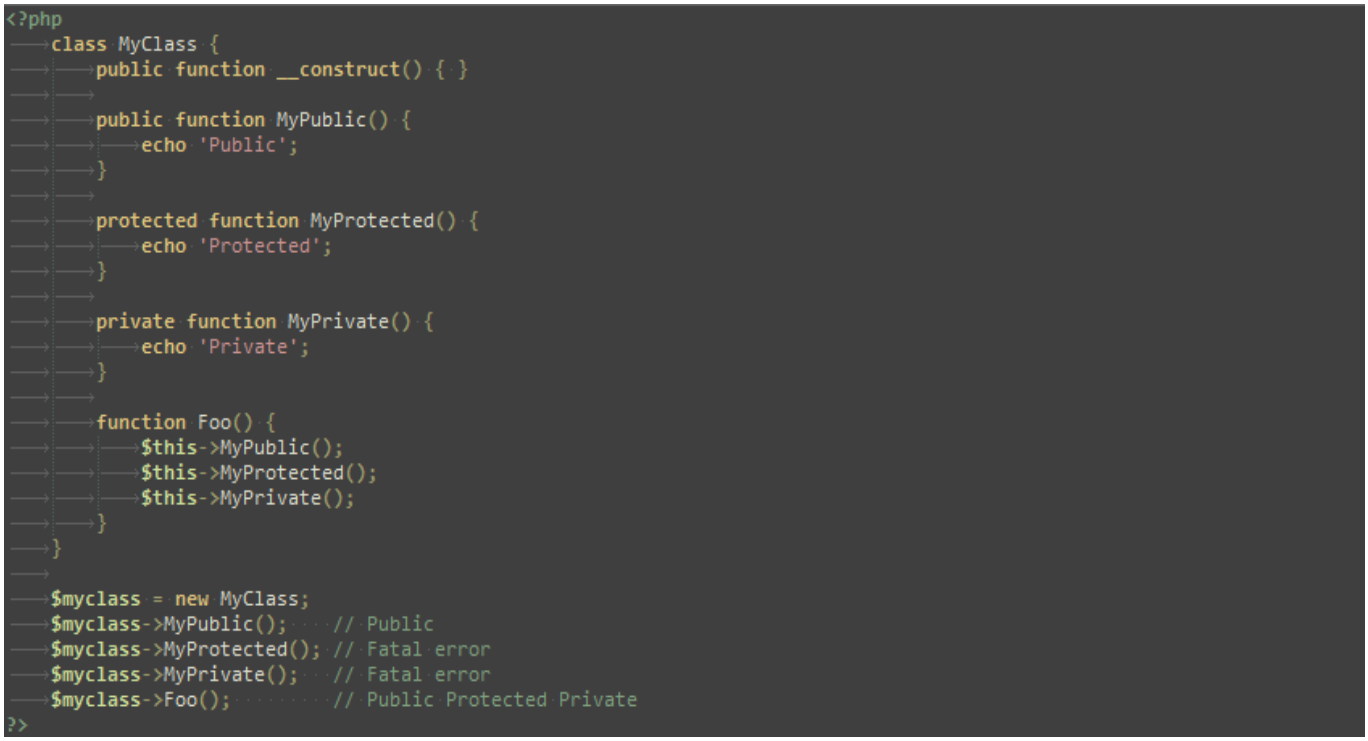
```php
<?php
class MyClass {
    public $public = 'Public';
    protected $protected = 'Protected';
    private $private = 'Private';

    function printHello() {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj = new MyClass();
echo $obj->public;    // Public
echo $obj->protected; // Fatal error
echo $obj->private;   // Fatal error
$obj->printHello();   // Public Protected Private
?>
```

Видимость из других объектов

Объекты, которые имеют общий тип (наследуются от одного класса), имеют доступ к элементам с модификаторами private и protected друг друга, даже если не являются одним и тем же экземпляром. Это объясняется тем, что реализация видимости элементов известна внутри этих объектов.

Область видимости метода

Методы класса должны быть определены через модификаторы public, private, или protected. Методы, где определение модификатора отсутствует, определяются как public.

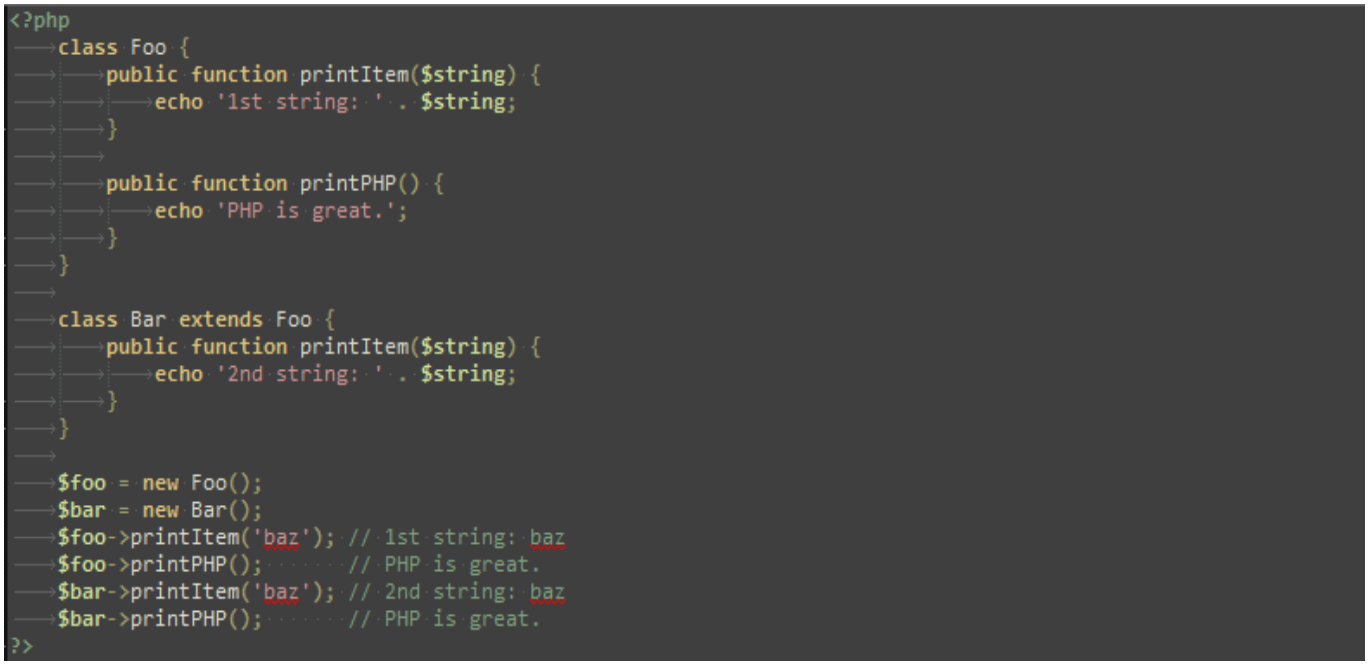


```php
<?php
    class MyClass {
        public function __construct() { }

        public function MyPublic() {
            echo 'Public';
        }

        protected function MyProtected() {
            echo 'Protected';
        }

        private function MyPrivate() {
            echo 'Private';
        }

        function Foo() {
            $this->MyPublic();
            $this->MyProtected();
            $this->MyPrivate();
        }
    }

    $myclass = new MyClass;
    $myclass->MyPublic();    // Public
    $myclass->MyProtected(); // Fatal error
    $myclass->MyPrivate();   // Fatal error
    $myclass->Foo();         // Public Protected Private
?>
```

Наследование

Когда вы расширяете класс, дочерний класс наследует все публичные и защищенные методы из родительского класса. До тех пор пока не будут эти методы переопределены, они будут сохранять свою исходную функциональность.

Пока не используется автозагрузка, классы должны быть объявлены до того, как их будут использовать. Если класс расширяет другой, то родительский класс должен быть объявлен до наследующего класса. Это правило применяется к классам, которые наследуют другие классы или интерфейсы.



```php
<?php
    class Foo {
        public function printItem($string) {
            echo '1st string: ' . $string;
        }

        public function printPHP() {
            echo 'PHP is great.';
        }
    }

    class Bar extends Foo {
        public function printItem($string) {
            echo '2nd string: ' . $string;
        }
    }

    $foo = new Foo();
    $bar = new Bar();
    $foo->printItem('baz'); // 1st string: baz
    $foo->printPHP();       // PHP is great.
    $bar->printItem('baz'); // 2nd string: baz
    $bar->printPHP();       // PHP is great.
?>
```

Оператор разрешения области видимости (::)

Оператор разрешения области видимости (также называемый "Paamayim Nekudotayim") или просто "двойное двоеточие" - это лексема, позволяющая обращаться к статическим свойствам, константам и перегруженным свойствам или методам класса. При обращении к этим элементам извне класса, необходимо использовать имя этого класса.

Возможно обратиться к классу с помощью переменной. Значение переменной не должно быть ключевым словом (например, self, parent или static).

```php
<?php
class MyClass {
    const CONST_VALUE = 'Constant value';
}

$classname = 'MyClass';
echo $classname::CONST_VALUE; // Constant value
echo MyClass::CONST_VALUE;    // Constant value
?>
```

Для обращения к свойствам и методам внутри самого класса используются ключевые слова self, parent и static.

```php
<?php
class MyClass {
    const CONST_VALUE = 'Constant value';
}

class OtherClass extends MyClass {
    public static $my_static = 'Static variable';

    public static function doubleColon() {
        echo parent::CONST_VALUE;
        echo self::$my_static;
    }
}

$classname = 'OtherClass';
$classname::doubleColon(); // Constant value Static variable
OtherClass::doubleColon(); // Constant value Static variable
?>
```

Ключевое слово "static"

Объявление свойств и методов класса статическими позволяет обращаться к ним без создания экземпляра класса. Атрибут класса, объявленный статическим, не может быть доступен посредством экземпляра класса (но статический метод может быть вызван).

Статические методы

Так как статические методы вызываются без создания экземпляра класса, то псевдо-переменная $this не доступна внутри метода, объявленного статическим.

```php
<?php
    class Foo {
        public static function aStaticMethod() {
            echo 'Hello!';
        }
    }

    Foo::aStaticMethod(); // Hello!
    $classname = 'Foo';
    $classname::aStaticMethod(); // Hello!
?>
```

Статические свойства

Статические свойства не могут использоваться с использованием оператора "->".

Как и любая другая статическая переменная PHP, статические свойства могут инициализироваться только используя литерал или константу, выражения же недопустимы. Таким образом вы можете инициализировать статическое свойство например целым числом или массивом, но не сможете указать другую переменную, результат вызова функции или объект.

Абстрактные классы

PHP поддерживает определение абстрактных классов и методов. Класс, который содержит по крайней мере один абстрактный метод, должен быть определен как абстрактный. Следует помнить, что нельзя создать экземпляр абстрактного класса. Методы, объявленные абстрактными, несут, по существу, лишь описательный смысл и не могут включать реализации.

При наследовании от абстрактного класса, все методы, помеченные абстрактными в родительском классе, должны быть определены в классе-потомке; кроме того, область видимости этих методов должна совпадать (или быть менее строгой). Например, если абстрактный метод объявлен как protected, то реализация этого метода должна быть либо protected либо public, но никак не private. Более того, сигнатуры методов должны совпадать, т.е. контроль типов (type hint) и количество обязательных аргументов должно быть одинаковым. К примеру, если в дочернем классе указан необязательный параметр, которого нет в сигнатуре абстрактного класса, то в данном случае конфликта сигнатур не будет.

Интерфейсы объектов

Интерфейсы объектов позволяют создавать код, который указывает, какие методы должен реализовать класс, без необходимости описывания их функционала.

Интерфейсы объявляются так же, как и обычные классы, но с использованием ключевого слова interface вместо class. Тела методов интерфейсов должны быть пустыми.

Все методы, определенные в интерфейсы должны быть публичными, что следует из самой природы интерфейса.

implements

Для реализации интерфейса используется оператор implements. Класс должен реализовать все методы, описанные в интерфейсе; иначе произойдет фатальная ошибка. При желании классы могут реализовывать более одного интерфейса за раз, реализуемые интерфейсы должны разделяться запятой.

Интерфейсы могут быть унаследованы друг от друга, так же как и классы, с помощью оператора extends.

constants

Интерфейсы могут содержать константы. Константы интерфейсов работают точно так же, как и константы классов, за исключением того, что они не могут быть перекрыты наследующим классом или интерфейсом.

```php
<?php
    interface iTemplate {
        public function setVariable();
        public function getHtml();
    }

    class Template implements iTemplate {
        public function setVariable() {
            //
        }

        public function getHtml() {
            //
        }
    }
?>
```

Трейты

PHP вводит инструментарий для повторного использования кода, называемый трейтом.

Трейт (англ. trait) - это механизм обеспечения повторного использования кода в языках с поддержкой единого наследования, таких как PHP. Трейт предназначен для уменьшения некоторых ограничений единого наследования, позволяя разработчику повторно использовать наборы методов свободно, в нескольких независимых классах и реализованных с использованием разных архитектур построения классов. Семантика комбинации трейтов и классов определена таким образом, чтобы снизить уровень сложности, а также избежать типичных проблем, связанных с множественным наследованием и смешиванием (mixins).

```php
<?php
    trait ezcReflectionReturnInfo {
        function getReturnType() {}
        function getReturnDescription() {}
    }

    class ezcReflectionMethod extends ReflectionMethod {
        use ezcReflectionReturnInfo;
        /* ... */
    }

    class ezcReflectionFunction extends ReflectionFunction {
        use ezcReflectionReturnInfo;
        /* ... */
    }
?>
```

Приоритет

Наследуемый член из базового класса переопределяется членом, находящимся в трейте. Порядок приоритета следующий: члены из текущего класса переопределяют методы в трейте, которые в свою очередь переопределяют унаследованные методы.

Несколько трейтов

Несколько трейтов могут быть вставлены в класс путем их перечисления в директиве use, разделяя запятыми.

Перегрузка

Перегрузка в PHP означает возможность динамически "создавать" свойства и методы. Эти динамические сущности обрабатываются с помощью "волшебных" методов, которые можно создать в классе для различных видов действий.

Все методы перегрузки должны быть объявлены как public.

Интерпретация "перегрузки" в PHP отличается от остальных объектно-ориентированных языков. Традиционно перегрузка означает возможность иметь множество одноименных методов с разным количеством или различными типами аргументов.

Перегрузка свойств

Метод __set() будет выполнен при записи данных в недоступные свойства.

Метод __get() будет выполнен при чтении данных из недоступных свойств.

Метод __isset() будет выполнен при использовании isset() или empty() на недоступных свойствах.

Метод __unset() будет выполнен при вызове unset() на недоступном свойстве.

Перегрузка методов

В контексте объекта при вызове недоступных методов вызывается метод __call().

В статическом контексте при вызове недоступных методов вызывается метод __callStatic().

Итераторы объектов

PHP предоставляет такой способ объявления объектов, который дает возможность пройти по списку элементов данного объекта, например, с помощью оператора foreach. По умолчанию, в этом обходе (итерации) будут участвовать все видимые свойства объекта.

Магические методы

Имена методов __construct(), __destruct(), __call(), __callStatic(), __get(), __set(), __isset(), __unset(), __sleep(), __wakeup(), __toString(), __invoke(), __set_state(), __clone() и __debugInfo() зарезервированы для "магических" методов в PHP. Не стоит называть свои методы этими именами, если вы не хотите использовать их "магическую" функциональность.

PHP оставляет за собой право все методы, начинающиеся с __, считать "магическими". Не рекомендуется использовать имена методов с __ в PHP, если вы не желаете использовать соответствующий "магический" функционал.

__sleep() , *__wakeup()*

Рекомендованное использование __sleep() состоит в завершении работы над данными, ждущими обработки или других подобных задач очистки. Кроме того, этот метод можно выполнять в тех случаях, когда нет необходимости сохранять полностью очень большие объекты.

Обычно __wakeup() используется для восстановления любых соединений с базой данных, которые могли быть потеряны во время операции сериализации и выполнения других операций повторной инициализации.

__toString()

Метод __toString() позволяет классу решать самостоятельно, как он должен реагировать при преобразовании в строку.

__invoke()

Метод __invoke() вызывается, когда скрипт пытается выполнить объект как функцию.

__set_state()

Этот статический метод вызывается для тех классов, которые экспортируются функцией var_export().

__debugInfo()

Этот метод вызывается функцией var_dump(), когда необходимо вывести список свойств объекта. Если этот метод не определен, тогда будут выведены все public, protected и private свойства объекта.

Ключевое слово "final"

PHP предоставляет ключевое слово final, разместив которое перед объявлениями методов класса, можно предотвратить их переопределение в дочерних классах. Если же сам класс определяется с этим ключевым словом, то он не сможет быть унаследован.

```php
<?php
    class BaseClass {
        public function test() {
            echo "Вызван метод BaseClass::test()";
        }

        final public function moreTesting() {
            echo "Вызван метод BaseClass::moreTesting()";
        }
    }

    class ChildClass extends BaseClass {
        public function moreTesting() {
            echo "Вызван метод ChildClass::moreTesting()";
        }
    }

    // Выполнение заканчивается фатальной ошибкой: Cannot override final method BaseClass::moreTesting()
?>
```

Свойства не могут быть объявлены окончательными, только классы и методы.

Клонирование объектов

Создание копии объекта с абсолютно идентичными свойствами не всегда является приемлемым вариантом.

Копия объекта создается с использованием ключевого слова clone (который вызывает метод __clone() объекта, если это возможно). Вызов метода __clone() не может быть осуществлён непосредственно.

$copy_of_object = clone $object;

При клонировании объекта, PHP выполняет неполную копию всех свойств объекта. Любые свойства, являющиеся ссылками на другие переменные, останутся ссылками.

Сравнение объектов

При использовании оператора сравнения (==), свойства объектов просто сравниваются друг с другом, а именно: два объекта равны, если они содержат одинаковые свойства с одинаковыми значениями (значения так же сравниваются через ==), и являются экземплярами одного и того же класса.

С другой стороны, при использовании оператора идентичности (===), переменные, содержание объект, считаются идентичными тогда и только тогда, когда они ссылаются на один и тот же экземпляр одного и того же класса.

Позднее статическое связывание

В PHP есть особенность, называемая позднее статическое связывание, которая может быть использована для того чтобы получить ссылку на вызываемый класс в контексте статического наследования.

Если говорить более точно, позднее статическое связывание сохраняет имя класса указанного в последнем "не перенаправленном вызове". В случае статических вызовов это явно указанный класс (обычно слева от оператора ::); в случае не статических вызовов это класс объекта. "Перенаправленный вызов" - это статический вызов, начинающийся с self::, parent::, static::

Объекты и ссылки

Одним из ключевых моментов объектно-ориентированной парадигмы PHP, которой часто обсуждается, является "передача объектов по ссылке по умолчанию".

Ссылка в PHP это псевдоним (алиас), который позволяет присвоить двум переменным одинаковое значение. Такая переменная содержит только идентификатор объекта, который позволяет найти конкретный объект при обращении к нему. Когда объект передается как аргумент функции, возвращается или присваивается другой переменной, то эти разные переменные не являются псевдонимами (алиасами): они содержат копию идентификатора, который указывает на один и тот же объект.

Сериализация объектов - сохранение объектов между сессиями

Функция serialize() возвращает строковое представление любого значения, которое может быть сохранено в PHP. Функция unserialize() использует эту строку для восстановления исходного значения переменной. Использование serialize для сериализации объекта сохранит имя класса и все его свойства, однако методы не сохраняются.

Для того, чтобы иметь возможность сделать unserialize() для объекта нужно чтобы класс этого объекта был определен заранее. То есть, если у вас есть экземпляр класса A, и вы сделаете его сериализацию, вы получите его строковое представление, которое содержит значение всех переменных описанных в нем. Для того, чтобы восстановить объект из строки в другом PHP файле класс A должен быть определен заранее. Это можно сделать сохранив определение класса A в отдельный файл и подключить этот файл или использовать функцию spl_autoload_register() для автоматического подключения.