Вопросы на модуль №3

1. Что означает понятие абстрактный класс?

В отличие от интерфейса содержит реализацию и абстрактный метод. Абстрактный класс нельзя создать, его можно только наследовать. Он носит описательный характер. Другие классы наследуют его реализацию и переопределяются согласно его методу.

1. Что буде выведено в результате работы скрипта?

**class** Myclass  
{  
 **public $a**=10;  
 **public $b**=20;  
 **private $c**=30;  
}  
$number=**new** Myclass();  
**foreach** ($number **as** $var=>$value) {  
 **echo "**$value**"**;  
}*// 10,20 Приватное свойство находится вне доступа он закрыт*

1. Какое ключевое слово используется для наследования одного класса от другого?

Extends

1. Какое из ключевых слов используется для создания объекта?

New

1. Какой метод в РНР вызывается автоматически, при попытке конвертации объекта в строку или при конкатенации?

\_\_toString

1. Какие есть в РНР модификаторы доступа и чем они отличаются?

Public – глобальная видимость

Privet – внутри самого класса можно к нему обратиться

Protect – внутри класса и класса наследника тоже можно обратиться

1. Какое название нельзя использовать в качестве имени константы класса?

Const объявление констант в классах.

1. Экземпляры классов еще называются:

Объектами

1. Что делает магический метод \_\_sleep? Для чего он может понадобиться?

Функция serialize() проверяет, присутствует ли в вашем классе метод с «магическим» именем \_\_sleep(). Если это так, то этот метод выполняется прежде любой операции сериализации. Он может очистить объект и предполагается, что будет возвращен массив с именами всех переменных объекта, который должен быть сериализован. 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 большие объекты.

Сериализация объектов.

1. Что будет выведено в результате работы скрипта?

**Class** Object  
{  
 **function** Object($entry)  
 {  
 $entry->**name**=**'John'**;  
 }  
}  
**class** Entry  
{  
 **var$name**=**"Maria"**;  
}  
$entry=**new** Entry();  
$obj = **new** Object($entry);  
**print** $entry->**name**;  
*//John*

1. Чем отличается создание объекта через new и через clone?

New - создание нового объекта его вызывает магический метод конструктор

1. Какой из след шаблонов проектирования используется, если нам необходимо в любой момент времени иметь один экземпляр какого-то класса?

Синглтон (Singlton) для базы данных

1. Какой из методов можно использовать для автоматической подгрузки классов?

\_\_autoload()

1. Какая из след возможностей позволяет расширять возможности существующих классов?

Наследование

1. Какой из типов данных в РНР передается по ссылке (по умолчанию)?

Object

1. Какая цель определения метода \_\_call?

Для отлавливания неопределенных методов

1. Если каждый из след методов возвращает объект, то как можно переписать код?

$a = new Myclass();

$b=$a->getInstance();

$c=$b->doSomthing();

Ответ $c=$a->getInstance()->doSomething();

1. Что выведет следующий код?

**Class** A {  
 **public static function** who() {  
 **echo *\_\_CLASS\_\_***;  
 }  
 **public static function** test() {  
 **self**::*who*();  
 }  
}  
**Class** B **extends** A {  
 **public static function** who()  
 {  
 **echo *\_\_CLASS\_\_***;  
 }  
}  
B::*test*(); *//A*

1. Что произойдет при объявлении методов как статических? Например – что произойдет при выполнении кода:

**class** newClass {  
 **public static** *$props*=[];  
 **static function** \_\_set($n, $v)  
 {  
 **self**::*$props*[$n]=$v;  
 }  
}  
newClass::*\_\_set*(**"prop"**.**"value"**);  
*print\_r*(newClass::*$props*);  
*//Warning: The magic method \_\_set()*

*//Array ( [propvalue] => )*

Магические методы делать статическими нельзя

1. Найдите ошибки в следующем коде:

**interface** Baselogger  
{  
 **public function** baselog($param1, $param2);  
}  
**interface** Logger **implements *Baselogger*** {  
 **private function** *log*();  
}  
**class** Mylog **implements** Logger {  
 **public function** baselog()  
 {  
  
 }  
 **public function** *log*() {  
  
 }  
}  
*//Ответ - в интерфейсе не может  
//быть закрытых методов, наследуются через extends  
//декларация baselog не соответствует*

1. Как обращаться к полям класса через ключевое слово $this?

$this->varname

1. Что будет выведено в результате работы скрипта?

**class** A {  
 **static** *$word* = **"hello"**;  
 **static function** hello()  
 {  
 **print static**::*$word*;  
 }  
}  
**class** B **extends** A {  
 **static** *$word* = **"bye"**;  
}  
B::*hello*();  
*//bye Позднее статистические связывание*

1. Какой из след магических методов запускается перед сериализацией объекта какого-то класса?

\_\_sleep

1. Что выведет следующий код?

**class** Foo {  
 **public static** *$cnt* = 0;  
 **public function** foo() {  
 ++**self**::*$cnt*;  
 }  
 **function** \_\_construct()  
 {  
 ++**self**::*$cnt*;  
 }  
}  
$f = **new** Foo;  
$d = $f;  
$c = **clone** $d;  
$f->foo();  
$d->foo();  
$c->foo();  
**echo** Foo::*$cnt*;  
*//4*

1. Что выведет следующий код?

**class** myClass  
{  
**var  
 private $property** = **"assigned"**;  
  
 **public function** getProperty()  
 {  
 **return** $this->**property**;  
 }  
}  
$obj = **new** myClass;  
$obj->getProperty();  
*//Parse error: syntax error, unexpected 'private' (T\_PRIVATE),  
// expecting variable (T\_VARIABLE) in C:\xampp\htdocs\modul\m3\25.php on line 12 не корректная синтаксическая конструкция. var архаизм*

1. Найдите ошибки в следующем коде:

**interface** iTemplate {  
 **const *MYCOST*** = 50;  
 **public function** setVariable($var);  
 **public function** getHtml($template);  
}  
**class** BadTemplate **interface** iTemplate {  
 **const *MYCONST*** = 100;  
 **public function** setVariable($var, $name)  
 {  
 *//Код* }  
}  
*//Константы нельзя переопределять в интерфейсах,  
//декларация setVariable должна быть совместима  
//с интерфейсом, нет реализации getHtml*

1. Можно ли в реализации метода дочернего класса указывать необязательный параметр которого нет в сигнатуре абстрактного класса? Например:

**abstract class** AbstractClass {  
 **abstract protected function** prefixName($name);  
}  
**class** ConcreteClass **extends** AbstractClass {  
 **public function** prefixName($name, $value = **""**)(*//код метода)* }  
 *//да*

1. Что выведет след. код?

**class** MyExeption **extends** Exception {  
 *//code*}  
**class** myClass {  
 **function** \_\_destruct()  
 {  
 **echo "Закрыть соединение с базой"**;  
 **throw new** Exception(**"Ошибка при завершении транзакции"**);  
 }  
  
}  
**class** myNewClass {  
 **function** \_\_destruct()  
 {  
 **echo "Закрываю лог фаил"**;  
 **throw new** Exception(**"Ошибка при закрытии лог фаила"**);  
 }  
}  
**try**{ $db = **new** myClass();  
 $log = **new** myNewClass();  
}  
**catch** (Exception $e) {  
 **echo** $e->getMessage();  
}  
**catch** (MyExeption $m) {  
 **echo** $m->getMessage();  
}  
*Нельзя из деструктора бросать исключения*

*//Закрываю лог фаил  
//Fatal error: Uncaught exception 'Exception'  
// with message 'Ошибка при закрытии лог фаила  
//' in C:\xampp\htdocs\modul\m3\28.php:23 Stack  
// trace: #0 [internal function]: myNewClass->\_\_destruct()  
// #1 {main} thrown in C:\xampp\htdocs\modul\m3\28.php on  
// line 23*

1. Что выведет след. код?

**class** Foo {  
 **public static** *$my\_static* = **'foo'**;  
}  
**class** Bar **extends** Foo {  
 **public function** foostatic() {  
 **return parent**::**$my\_static**;  
 }  
}  
$foo = **new** Bar();  
$foo->**my\_static** = **"Bar"**;  
**echo***var\_dump*($foo->foostatic());  
*//foo  
//strict standarts accessing static  
// property Bar::$my\_static as non stati foo*

1. Есть класс:

**class** NewClass {  
 **public function** getOne() {  
 **echo *\_\_FUNCTION\_\_***;  
 }  
 **public function** getTwo(){  
 **echo *\_\_METHOD\_\_***;  
 }  
}

Что выведет код  
$obj = **new** newClass();  
$obj->getOne();  
**echo "<br>"**;  
$obj->getTwo();  
*//getOne  
//NewClass::getTwo (имя метода)*

1. Реализуйте задачу

Создайте класс DB, в конструкторе инициализируйте свойство $connect - случайным образом принимающее false\true. Если false - бросьте исключение “Нет соединения”. Объект создает index.php, класс Дб в файле ДБ.пхп.

При загрузке  файла индекс.пхп в браузере должно выводится сообщение “Нет соединения” в случае неудачи и имя файла, в котором брошено исключение.

1. Что означает понятие «перегрузка метода» в РНР?

Полиморфизм – возможность перезаписать имя фаила

1. Что означает ключевое слово final?
2. Что в РНР может содержать внутри себя класс?

Свойства, методы, константы, трейты.

1. Что выведет следующий код?

**class** SimpleClass {  
 *//some code*}  
$instance = **new** SimpleClass();  
$assigned = $instance;  
$reference = &$instance;  
$instance->**var** = **"some value"**;  
$instance=**null**;  
*var\_dump*($instance);

*//null  
//Notice: Undefined variable: var\_dump in  
// C:\xampp\htdocs\modul\m3\35.php on line 18  
  
//Fatal error: Function name must be a string   
//in C:\xampp\htdocs\modul\m3\35.php on line 18*

1. Можно ли в РНР переопределять любые свойства и методы?

Нет – кроме финальных.

1. Что такое деструктор и когда он вызывается?
2. Что может содержать в себе интерфейс? Напишите примеры

Константы, методы

1. Реализовать класс Logger со статистическим методом save.

Реализовать несколько классов, в них минимум по одному методу, в каждом из которых вызывается save. Метод save класса Logger , в зависимости от того, из какого метода и класса вызывается – запишите в фаил «log.txt»время своего вызова и имя класса и метода из которого его вызвали.

**class** Logger {  
 **public static function** save($method) {  
 $time = *date*(**'H-i-s'**);  
 *file\_put\_contents*(**'log.txt'**, **"[**$time**]"**.$method.***PHP\_EOL***, ***FILE\_APPEND***);  
 }  
}  
**class** Caller {  
 **public function** hello()  
 {  
 Logger::*save*(***\_\_METHOD\_\_***);  
  
 }  
}  
$caller = **new** Caller();  
$caller->hello();  
$caller->hello();  
*sleep*(1);  
$caller->hello();

1. Напишите пример паттера «Singleton»

**final class** Singleton  
{  
 */\*\*  
 \** ***@var*** *Singleton $instance  
 \*/* **private static** *$instance*;  
 */\*\*  
 \* gets the instance via lazy initialization (created on first usage)  
 \*/* **public static function** getInstance() *//Singleton* {  
 **if** (**null** === **static**::*$instance*) {  
 **static**::*$instance* = **new static**();  
 }  
 **return static**::*$instance*;  
 }  
 */\*\*  
 \* is not allowed to call from outside to prevent from creating multiple instances,  
 \* to use the singleton, you have to obtain the instance from Singleton::getInstance() instead  
 \*/* **private function** \_\_construct()  
 {  
 }  
 */\*\*  
 \* prevent the instance from being cloned (which would create a second instance of it)  
 \*/* **private function** \_\_clone()  
 {  
 }  
 */\*\*  
 \* prevent from being unserialized (which would create a second instance of it)  
 \*/* **private function** \_\_wakeup()  
 {  
 }  
}

1. Напишите пример паттера «Registry»
2. **abstract class** Registry  
   {  
   **const *LOGGER*** = **'logger'**;  
   */\*\*  
   \* this introduces global state in your application which can not be mocked up for testing  
   \* and is therefor considered an anti-pattern! Use dependency injection instead!  
   \*  
   \** ***@var*** *array  
   \*/***private static** *$storedValues* = [];  
   */\*\*  
   \** ***@var*** *array  
   \*/***private static** *$allowedKeys* = [  
   **self**::***LOGGER***,  
   ];  
   */\*\*  
   \** ***@param*** *string $key  
   \** ***@param*** *mixed $value  
   \*  
   \** ***@return*** *void  
   \*/***public static function** set(string $key, $value)  
   {  
   **if** (!*in\_array*($key, **self**::*$allowedKeys*)) {  
   **throw new** \InvalidArgumentException(**'Invalid key given'**);  
   }  
   **self**::*$storedValues*[$key] = $value;  
   }  
   */\*\*  
   \** ***@param*** *string $key  
   \*  
   \** ***@return*** *mixed  
   \*/***public static function** get(string $key)  
   {  
   **if** (!*in\_array*($key, **self**::*$allowedKeys*) || !**isset**(**self**::*$storedValues*[$key])) {  
   **throw new** \InvalidArgumentException(**'Invalid key given'**);  
   }  
   **return self**::*$storedValues*[$key];  
   }  
   }