

Лабораторные работы по курсу РНР: Проектирование и разработка сложных веб – проектов

Лабораторная работа 1.1

Использование шаблонов разработки Стратегия и Фабричный метод

Упражнение 1. Использование шаблона Стратегия

1. Откройте папку в корне сайта **/patterns**
2. Создайте класс **Product**, который реализует абстрактный класс **AProduct**
3. На основе интерфейса **IRender** создайте классы **HTMLRender**, **XMLRender**, **JSONRender**, каждый из которых должен представлять продукт в соответствующем виде
4. Создайте экземпляр **Product** на основе произвольных данных
5. Выведите данные из экземпляра, используя различные стратегии вывода
6. Убедитесь в правильности работы скрипта
7. * Создайте у класс **Product** свойство, отвечающее за определение ценовой стратегии (опт, розница). Реализуйте ценовые стратегии в виде классов и повторите вывод продукта, используя разные стратегии.

Упражнение 2. Использование шаблона Фабричный метод

1. Создайте класс **ProductFactory** с методом **get()**
2. Реализуйте метод **get()**, он принимать входные данные для конструктора товара и возвращать экземпляр товара (продукта)
3. Перепишите код предыдущего упражнения так, чтобы создание экземпляра товара шло через фабричный метод
4. Проверьте работу скрипта

Упражнение 3. Для самостоятельной (домашней) работы

1. Реализуйте в рамках работы с классом **Product** шаблон проектирования Адаптер
2. Реализуйте в рамках работы с классом **Product** шаблон проектирования Декоратор
3. * Нарисуйте UML-диаграммы для выполненных шаблонов.
4. ** Используя **dia** и расширение для генерации РНР-классов, чтобы из UML-диаграммы классов получить код РНР-класса

Лабораторная работа 1.2

Использование шаблонов разработки Одиночка и Наблюдатель

Упражнение 1. Использование шаблона Одиночка

1. Откройте реализацию класса **ProductFactory**
2. Сделайте так, чтобы был доступен единственный экземпляр класса **ProductFactory**
3. Попробуйте создать второй экземпляр класса **ProductFactory**
4. Убедитесь, что существует единственный доступный экземпляр

Упражнение 2. * Использование шаблона Наблюдатель: Для самостоятельной (домашней) работы

1. Откройте реализацию класса **ProductFactory**
2. Укажите, что класс **ProductFactory** выполняет интерфейс **Observable**

```
interface Observable {  
    function attach(Observer $observer ) ;  
    function detach(Observer $observer ) ;  
    function notify();  
}
```

3. Реализуйте методы **attach()** и **detach()** у **ProductFactory**
4. В методе **get()** укажите вызов метода **notify()**. Примечание: все подписчики будут уведомляться при создании нового товара
5. Создайте интерфейс **Observer**

```
interface Observer{  
    function update( Observable $observable );  
}
```

6. Создайте класс **User**
7. Укажите, что класс **User** выполняет интерфейс **Observer**
8. Подпишите экземпляр класса **User** на создание нового объекта **Product**
9. Убедитесь, что подписчик реагирует на создание нового продукта

Лабораторная работа 2.1

Использование SPL-итераторов: ArrayIterator

Упражнение 1. Использование итератора по массиву ArrayIterator

1. Откройте корневую папку в корне сайта **/labs/iterators**
2. Откройте файл **iterators.php**
3. Создайте массив **\$products** с названиями продуктов
4. Создайте экземпляр **ArrayObject**, передав в конструктор в качестве аргумента **\$products**.
Экземпляр назовите **\$prodArrayObject**
5. Получите из **\$prodArrayObject** итератор при помощи метода **getIterator()**
6. Выведите количество элементов в итераторе при помощи метода **count()** в строку вида
«Надо пройти по записям N шт.»
7. В цикле, пока итератор еще содержит записи, выведите ключ – метод **key()** и текущий элемент – метод **current()**.
8. Примечание: в теле цикла укажите переход к следующему итерируемому элементу
9. Сохраните файл **/labs/iterators/iterators.php**

Упражнение 2. Проверка работы

1. Запустите в браузере **labs/spl/iterators.php**
2. Убедитесь, что информация о продуктах выводится корректно
3. Исправьте существующие ошибки (если они появились)

Упражнение 3. Итератор по массиву объектов

1. Измените переменную **\$products**, заполнив её экземплярами **Product**
2. Измените код цикла так, чтобы в браузер выводилась информация об объектах
3. Сохраните файл и проверьте работу скрипта

Лабораторная работа 2.2

Использование SPL-итераторов: CallbackFilterIterator

Упражнение 1. Использование итератора по callback - CallbackFilterIterator

1. Откройте корневую папку в корне сайта **/labs/iterators**
2. Откройте файл **iterators.php**
3. Перепишите код используя итератор **CallbackFilterIterator** так, чтобы выводились товары только определенного ценового диапазона.
4. (*) Можно поэкспериментировать с сортировкой товаров по другим свойствам
5. Сохраните файл **iterators.php**

Упражнение 2. Проверка работы

1. Запустите в браузере **labs/spl/iterators.php**
2. Убедитесь, что информация о продуктах выводится корректно
3. Исправьте существующие ошибки (если они появились)

Лабораторная работа 2.3

Использование SPL-итераторов: GlobIterator

Упражнение 1. Использование итератора по массиву Globliterator

1. Откройте корневую папку в корне сайта **/labs/iterators**
2. Создайте файл **globiterator.php**
3. Создайте итератор **Globliterator** на основе всех файлов текущей директории
4. Выведите список названий и размеров файлов директории на основе итератора
4. Запустите в браузере **labs/spl/iterators.php**
5. Убедитесь, что информация о продуктах выводится корректно
6. Исправьте существующие ошибки (если они появились)

Упражнение 3. Проверка работы

4. Запустите в браузере **labs/spl/globiterator.php**
5. Убедитесь, что информация о продуктах выводится корректно
6. Исправьте существующие ошибки (если они появились)

Лабораторная работа 2.4

Автозагрузка классов через `spl_autoload_register()`

Упражнение 1. Регистрация функций автозагрузки

1. Откройте корневую папку в корне сайта **/labs/iterators**
2. Откройте файл **iterators.php**
3. Создайте две функции автозагрузки файлов **load1()** и **load2()**. Примечание: названий функций не важны, главное, чтобы они «умели» подгружать классы
4. Зарегистрируйте функции автозагрузки при помощи **spl_autoload_register()**
5. Создайте объекты, классы которых будут загружены функциями
6. Сохраните файл **iterators.php**

Упражнение 2. Проверка работы

1. Запустите в браузере **labs/spl/iterators.php**
2. Убедитесь, что классы подгружены и объекты созданы корректно
3. Исправьте существующие ошибки (если они появились)

Лабораторная работа 2.5

Пространства имен (Namespace) – механизм группировки связанных констант, функций и классов, логически связанных между собой.

1. Откройте файл **/labs/namespace/lab.php**
2. Распечатайте текущее пространство имён
`echo __NAMESPACE__;`
3. Убедитесь, что в браузере ничего не вывелось
4. Над строкой с оператором `echo` впишите пространство имён **Specialist**:
`namespace Specialist`
5. Убедитесь, что в браузере отобразилось слово «Specialist». Закомментируйте вывод.
6. Внимательно изучите код

```
const SPEC_URL = "http://specialist.ru";  
function spec_func() { return __FUNCTION__; }  
class Specialist {  
    static function getClass(){
```

```

        return __CLASS__;
    }
}
echo "<p>",SPEC_URL;
echo "<p>","\Specialist\SPEC_URL;
echo "<p>",spec_func();
echo "<p>",Specialist::getClass();

```

7. Вставьте код в lab.php, сохраните файл
8. Изучите результат работы. Должен получиться результат:

```

«http://specialist.ru
http://specialist.ru
Specialist\spec_func
Specialist\Specialist»

```

9. Создайте в файле lab.php функцию substr():

```

function substr() {
    echo " <p> test";
}
substr();

```

10. Обратите внимание! Несмотря на то, что substr() – существующая функция PHP в нашем пространстве имён Specialist, она является самостоятельной функцией.
11. Добавьте перед вызовом функции символ обратного слеша «\».
12. Посмотрите результат в браузере. Функция будет требовать два параметра. Удалите слеш перед функцией.

Лабораторная работа 2.6

1. В нижней части скрипта в файле lab.php создайте экземпляр класса User (в текущем файле этот класс не описан, нам нужно просто попытаться создать экземпляр).

```

$user = new User();
echo "<p>",$user->name;

```
2. Должна показаться ошибка: «**Fatal error:** Class 'Specialist\User' not found in...». То есть в **текущем** пространстве имён нет такого класса.
3. Нам нужно подключить класс User из папки vendor. Папка с таким названием обычно содержит сторонние библиотеки/классы, которые мы можем загружать.
 Добавьте обратный слеш перед названием класса при создании объекта:

```

$user = new \User();

```
4. Должна показаться ошибка: «**Fatal error:** Class 'User' not found in...». То есть в **глобальном** пространстве имён нет такого класса.
5. В верхней части файла lab.php подключите autoload.php:

```

require __DIR__ . '/vendor/autoload.php';

```
6. Создайте поочередно экземпляры класс Foo

```

$foo = new \Some\Foo\Foo(); //а потом просто new \Some\Foo ()
$foo->getInfo();

```
7. Сравните работу в случае с \Some\Foo\Foo() и \Some\Foo(). Найдите в папке vendor файлы, соответствующие вызываемым в коде.
8. Примечание: во многих фреймворках такая организация загрузки классов упрощает построение веб-приложения

Лабораторная работа 2.7 (для самостоятельной работы)

1. Изучите способы загрузки файлов с классами в Laravel, Yii и Zend Framework 2