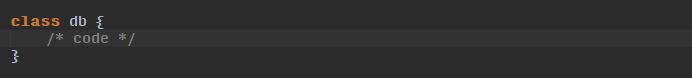
**Практическое занятие №34 Шаблоны проектирования**

**Задание №1**

Проверьте запущен ли у Вас Open Server. Папка с материалами урока (php34.loc) должна быть размещена в директории /domains сервера. Заходим на <http://php34.loc> и начинаем урок.

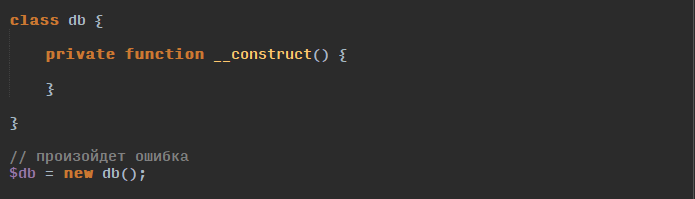
**Задание №2**

Шаблон Singleton используется, когда есть необходимость сделать так, чтобы объект класса был единственным и было бы невозможно создание еще одного такого объекта. Предположим, что у нас есть класс для работы с базой данных:

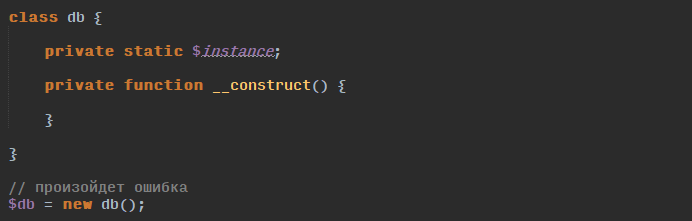


Если мы работаем с одной базой данных, то нам нужен только один объект, который будет работать с БД – то есть мы будем применять шаблон singleton.

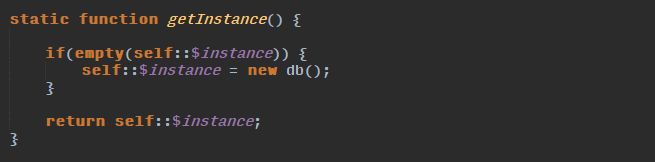
Начнем с того что нам нужно запретить создавать объекты напрямую, для этого мы определяем конструктор как приватный метод:



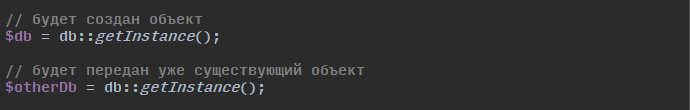
Далее создаём статическую переменную, в которой будет храниться единственный экземпляр объекта этого класса:



И наконец создаём статический метод, который и будет создавать один единственных экземпляр объекта db и записывать его в статическое свойство:



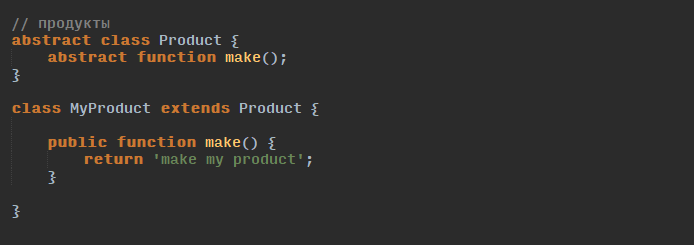
Такие образом создание объекта выглядит вот так:



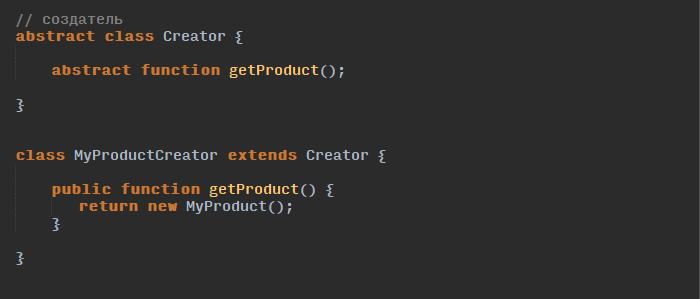
Вывод: Мы используем шаблон Singleton в тех случаях, когда нам нужен всего один объект данного класса. Это может быть объект подключения к базам данных или объект конфигурации приложения.

**Задание №3**

Шаблон «Factory Method». Это подход, при котором созданием объектов занимаются отдельные, специально для этого созданные классы и объекты. Рассмотрим пример, у нас есть абстрактный класс ***Product*** и одна из его реализаций ***MyProduct***:

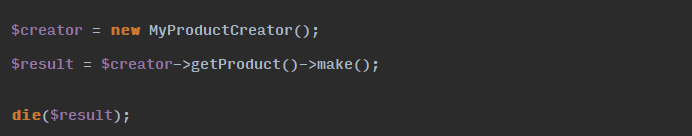


Теперь создаём классы создателей (классы, которые будут создавать наши продукты Product):



В примере выше код очень сильно упрошен. В реальной ситуации действия, которые необходимо сделать для подготовки создания объекта могут быть очень сложными.

И наконец запуск продукта может выглядеть так:

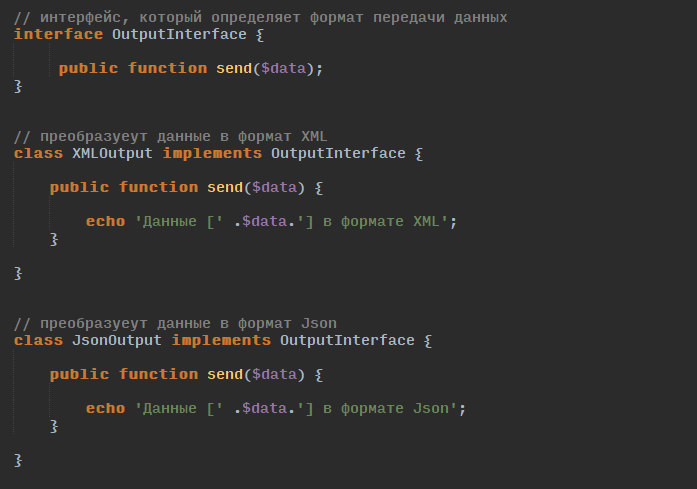


В переменной $result мы должны получить строку «*make my product*», что свидетельствует о том, что произошел вызов метода ***make***.

**Задание №4**

Паттерн «Strategy» или «Стратегия». Этот паттерн используется, когда некоторые действия нужно менять в зависимости от обстоятельств. Рассмотрим следующий пример: у нас есть класс *Data,* его задача хранить данные и преобразовывать их различные форматы. Причем по условиям ТЗ форматов данных может быть много и должна быть возможность легко добавлять новые форматы.

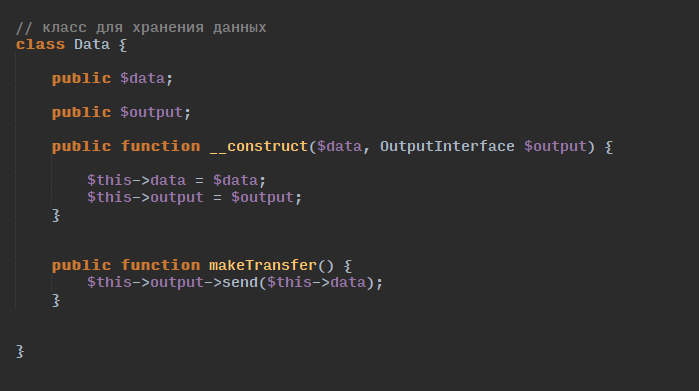
Сначала мы создаём интерфейс *OutputInterface,* который будет наследоваться классами XMLOutput и JsonOutput. Классы XMLOutput и JsonOutput могут конвертировать данные из класса Data в соответствующие свои форматы. Смотрите код ниже:



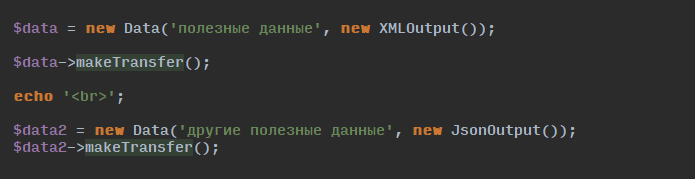
В примере выше код очень сильно упрошен. В реальной ситуации действия, которые необходимо сделать для обработки данных в нужный формат могут быть очень сложными.

Обратите внимание, что в интерфейсы мы объявили метод send, который нужно обязательно реализовать в классах потомках. Это поможет нам в дальнейшем.

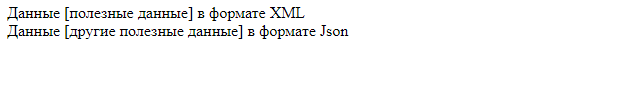
Создаём класс *Data*, так что бы он в одном из своих свойств содержал объект типа *OutputInterface* и использовал его в методе *makeTransfer*:



Вызвать и протестировать наш функционал можно следующим образом:

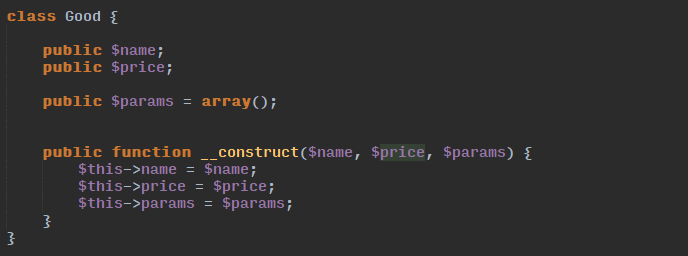


В результате мы должны получить вывод наших данных сначала в формате Json и XML:



**Задание №7**

Шаблон «Lazy Load» или ленивая загрузка. Суть этого шаблона в том, чтобы отложить получение некоторых свойств объекта до момента первого обращения к нему. Например, у нас есть класс Good описывающий товар на нашем сайте. Упрошенная версия этого класса может выглядеть вот так:



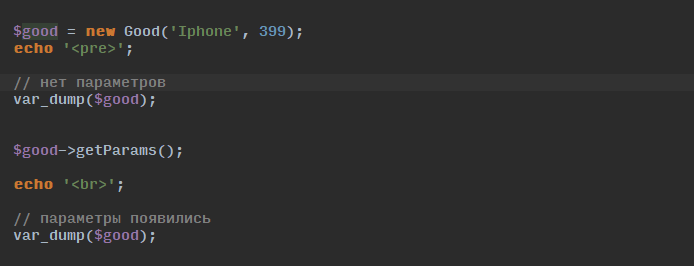
Этот класс работает так как нам нужно. Он оперирует названием товара, ценой товара и его параметрами. Но в одно прекрасное утро к нам приходит босс и говорит, что нужно оптимизировать нагрузку к базе данных и ускорить работу сайта. Мы знаем, что параметры мы запрашиваем только на странице товара, на всех других страницах наш класс Good не используют параметры товара ($this->params).

Используя шаблон «Lazy Load» переписываем класс Good следующим образом:



В коде выше мы упростили получение параметров, в реальном примере нам нужно было бы делать запросы к базе данных или другим способом получать нужные параметры.

Протестировать наш код можно следующим образом:



В итоге у нас должно быть два дампа: одни с параметрами, другой – без.

