Приветствую Вас в первом уроке мини курса по изучению паттернов проектирования на языке PHP. Хотя, справедливости ради, стоит отметить , что паттерны – это понятие актуально для любого языка поддерживающем объектно-ориентированный подход к программированию. А значит можно сделать вывод, что по большому счету паттерны описывают некое взаимодействие между классами, и в этом Вы сами убедитесь далее по ходу курса.  
Теперь давайте немного поговорим о том что такое паттерны, какие проблемы они решают и для чего вообще они применяются.

Итак, Паттерн проектирования, или шаблон проектирования — это часто встречающееся решение определённой проблемы при проектировании архитектуры веб-приложения, или любой другой программы. По сути – это способ решения периодически возникающих проблем, то есть некоторых типовых задач.  
Но Вы должны понимать, что паттерн – это не библиотека и не определенный набор функций или методов, то есть это не готовое решение, которое можно взять и скопировать. Это в первую очередь способ, или некая методика по решению той или иной проблемы, которую потребуется написать самостоятельно и возможно адаптировать под конкретную реализацию.

Так же не стоит путать паттерны с алгоритмами, потому как если алгоритм предоставляет четкий набор действий, то паттерн – это только лишь описание решения проблемы, реализация которого может быть различной.

Как правило если говорят о паттернах то выделяют следующие основные пункты:  
- проблема, которую решает паттерн;

- структуры классов, составляющих решение;

- примерная реализация

Теперь, думаю у Вас назрел вполне предсказуемый вопрос, зачем знать паттерны, если по сути это не готовое решение, а только методика? Для ответа на данный вопрос можно привести довольно банальный и стандартный пример. Согласитесь что такую простую проблему, как забить гвоздь, можно решить разными способами. Можно использовать молоток, а можно использовать пласкогубци, или скажем дрель. По сути результат будет один и тот же, но вот качество и затраченные ресурсы, будут различны. Осознанное владение инструментом как раз и отличает профессионала от любителя.

Паттерны предоставляют:  
 - Проверенные решения., что способствует значительной економии времени.

- Стандартизацию кода, что способствует уменьшению допускаемых ошибок и просчетов в реализации.

- Общий подход к решению проблемы, что дает возможность систематизировать и технически правильно описать сруктуру вашего приложения. Ну и конечно же применяя паттерны значительно проще, объяснять принцип работы определенной части системы. Достаточно просто указать что система работает на основе конкреного паттерна.

Классификация паттернов

Паттерны отличаются по уровню сложности, детализации и охвата проектируемой системы.

Самые низкоуровневые и простые паттерны — идиомы. Они не универсальны, поскольку применимы только в рамках одного языка программирования.

Самые универсальные — архитектурные паттерны, которые можно реализовать практически на любом языке. Они нужны для проектирования всей программы, а не отдельных её элементов.

Кроме того, паттерны отличаются и предназначением. Есть три основные группы паттернов

Порождающие паттерны беспокоятся о гибком создании объектов без внесения в программу лишних зависимостей.

Структурные паттерны показывают различные способы построения связей между объектами.

Поведенческие паттерны заботятся об эффективной коммуникации между объектами.

Одиночка — это порождающий паттерн проектирования, который гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.

Одиночка решает сразу две проблемы, нарушая принцип единственной ответственности класса.

Гарантирует наличие единственного экземпляра класса. Чаще всего это полезно для доступа к какому-то общему ресурсу, например, базе данных.

Представьте, что вы создали объект, а через некоторое время пробуете создать ещё один. В этом случае хотелось бы получить старый объект, вместо создания нового.

Такое поведение невозможно реализовать с помощью обычного конструктора, так как конструктор класса всегда возвращает новый объект.

Предоставляет глобальную точку доступа. Это не просто глобальная переменная, через которую можно достучаться к определённому объекту. Глобальные переменные не защищены от записи, поэтому любой код может подменять их значения без вашего ведома.

Одиночка имеет такие же преимущества и недостатки, что и глобальные переменные. Его невероятно удобно использовать, но он нарушает модульность вашего кода.

Вы не сможете просто взять и использовать класс, зависящий от одиночки в другой программе. Для этого придётся эмулировать присутствие одиночки и там. Чаще всего эта проблема проявляется при написании юнит-тестов.