1. What is the Design pattern?

Шаблоны проектирования - это многоразовые решения часто возникающих проблем при разработке программного обеспечения. Они являются одновременно захватывающей и увлекательной темой для изучения на любом языке программирования.

повторяемая [архитектурная конструкция](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F), представляющая собой решение проблемы [проектирования](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) в рамках некоторого часто возникающего [контекста](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D0%B5%D0%BA%D1%81%D1%82).

Общепринятое решение каких-то задач или проблем, концепция решения

1. Difference between Architecture and Design pattern?

архитектура приложения относится к более крупной структуре и организации приложения, в то время как шаблон проектирования относится к методу решения определенного типа проблемы, обычно гораздо более сфокусированному и более низкому уровню, чем глобальная структура системы. Например, шаблон проектирования может быть решением проблемы поведения объекта в зависимости от его внутреннего состояния, или когда поведение объекта зависит от состояния других объектов в системе. С архитектурным дизайном уровня вы не беспокоитесь о деталях того, как каждая часть будет работать, вы обеспокоены тем, как структурировать все части вместе.

MVC определяет поведение и структуру всего проекта, паттерны проектирования для решения небольших локальных задач

1. Difference between Observer (Наблюдатель)

Наблюдатель-это шаблон проектирования, в котором объект (известный как субъект) поддерживает список объектов, зависящих от него (наблюдателей), автоматически уведомляя их о любых изменениях состояния.

and Publisher Subscriber?

наблюдатель (или объект), желающий получать уведомления по теме, должен подписаться на этот интерес к объекту, инициирующему событие (субъект).

Publisher Subscriber, использует канал темы / события, который находится между объектами, желающими получать уведомления (подписчики), и объектом, запускающим событие (издатель). Эта система событий позволяет коду определять события приложения, которые могут передавать пользовательские аргументы, содержащие значения, необходимые подписчику. Идея здесь состоит в том, чтобы избежать зависимостей между подписчиком и издателем.

1. How to use singleton?

Существует один класс и в приложении может быть только один экземпляр данного класса.

может быть реализован путем создания класса с помощью метода, который создает новый экземпляр класса, если он не существует. Если экземпляр уже существует, он просто возвращает ссылку на этот объект.

1. Examples of patterns in JS

Классификация:

* + - 1. Creational patterns. Порождающие шаблоны. Создание объектов
  1. Singleton
  2. Factory
  3. Constructor pattern
     + 1. Structural patterns. Структурные шаблоны.

определяют различные сложные структуры, которые изменяют [интерфейс](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_(%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) уже существующих объектов или его реализацию, позволяя облегчить разработку и оптимизировать программу.

2.1 Facade

2.2 Module

позволяет скрывать приватную информацию в замыканиях, давая доступ лишь к тому, что разработчик решил сделать общедоступным

2.3 Decorator

* + - 1. Behavior patterns. Поведенческие шаблоны. Налаживают коммуникация (взаимодействия) между существующими объектами

3.1 Observer

3.2 Publisher

3.3 Command

Architectural pattern

1. MVC (Model-View-Controller)
2. MVP (Model-View-Presenter)
3. MVVM (Model-View-View-Model)

Constructor

|  |
| --- |
| // function Server(name, ip) { |
|  | // this.name = name |
|  | // this.ip = ip |
|  | // } |
|  | // |
|  | // Server.prototype.getUrl = function() { |
|  | // return `https://${this.ip}:80` |
|  | // } |
|  |  |
|  | class Server { |
|  | constructor(name, ip) { |
|  | this.name = name |
|  | this.ip = ip |
|  | } |
|  |  |
|  | getUrl() { |
|  | return `https://${this.ip}:80` |
|  | } |
|  | } |
|  |  |
|  | const aws = new Server('AWS German', '82.21.21.32') |
|  | console.log(aws.getUrl()) |