**Патерн**

**Патерн** – частоповторювана архітектурна конструкція, що вирішує конкретні проблеми проектування.

Категорії патернів:

**Creational Patterns** (Патерни створення) – патерни сисемного відстеження і контролю створення певних класів, об’єктів

* **Конструктор дизайн патерн** – слугує для створення нових сутностей
* **Фабрика(Factory)** – при наявності колекції подібних класів, які відрізняються мінімально і нам потрібно їх регулярно створювати, то можна створити окремий клас-конструктор по патрену Factory створюватиме нам дані сутності (екзепляри колекції цих класів в залежності від переданих параметрів)
* **Prototype** – використання об’єкта із певними властивостями і методами (який може містити посилання на поля, яких в ньому немає), і який ми прототипно прив’язуємо до нашого нового об’єкта через **Object.create()**
* **Singletop –** в усьому додатку може існувати клас і даний патерн вимагає лише одну його ініціалізацію

**Design Patterns** (Структурні патерни) – використовуються для добавлення нового функціоналу в уже існуючу систему без зміни старого коду – не порушуючи перший закон SOLID

* **Adapter** – позволяє інтегрувати старий інтерфейс в новий інретфейс для сумісної роботи, не ламаючи додаток. Патерн для рефакторінгу коду чи переходу на новішу версію
* **Decorator** – добавляємо нову поведінку чи функціонал для існуючих класів. Для цього в клас-декоратор передаємо ініціалізацію класу до якого ми повинні додати новий функціонал
* **Façade** – створення більш простий і унікальний (публічний) інтерфейс для спрощення взаємодії (JQuery - $(‘’))
* **Flyweight** – ефективна передача даних через різні типи об’єктів (загрузка різних зображень, кешування)
* **Proxy** – ставимо пастки на взаємодію із даними класу чи об’єкту.

**Поведінкові дизайн патерни** – слугують для комунікації між різними вже існуючими об’єктами, класами, сутностями (API, versions)

* **ChaninOfResponsibility** –дозволяє покращити комунікацію між об’єктами різного типу, послідовно в одного і того ж об’єкта викликати якийсь набір операцій і послідовно їх моифіковувати. Як приклад бібліотека jquery, в якій можна використовувати chaning
* **Command** – дозволяє створювати певну абстракту оболонку над функціоналом, який позволяє керувати через певний об’єкт, записуючи певний стан, який був викликано. (Redux)
* **Iterator** – створюємо клас\об’єкт в якому ми послідовно зможемо отримувати доступ до інформації
* **Mediator** – дозволяє вистроювати тісну комунікацію між різними об’єктами різного типу, і представляє централізовану абстракцію, яка позволяє взаємодіяти групі об’єктів через один одного
* **Observer** – формує залежності 1 до багатьох, є об’єкт в якого ми можемо затрігерити виклик змін і всі об’єкти, які підписані в дані зміни, виконують певний код
* **State** – дозволяє створювати різні класи, що є елементами стейта і делігірувати зміни станів цих класів на інший клас стейт і буде міняти внутрішній стан даних цих елементів
* **Strategy** – створює оболонку для різних інтерфесів, для використання різних алгоритмів і інтерфейсів в конкретній задачі, Визначає сімейство алгоритмів, які наслідують об’єкти в незмінному порядку
* **Template** – визначає скелет певного алгоритму, але при цьому делігірує створення конкретного функціоналу в конкретні класи. Він визначає структуру, а дочірні класи реалізовують конкретний функціонал