**Курс " Javascript Fullstack Developer" и "Javascript Frontend"**

# **Кожевников Сергей 2024 год**

# **Задание 6**

Опциональное дз:

Выучите принципы SOLID наизусть.

Попробуйте по памяти воспроизвести примеры JS кода для демонстрации каждого из

этих принципов.

Выполните практические задачи по алгоритмам из Хендбука Яндекса

(https://education.yandex.ru/handbook/algorithms).

# 1) Реализовать алгоритм Флойда–Уоршелла или алгоритм Беллмана-Форда в графе (на матрице смежности или на списках смежности). 2) Реализовать с использованием TypeScript паттерны Строитель, Прокси, Медиатор. Привести примеры использования собственных классов. 3) Пройти хэндбук от Яндекса: <https://education.yandex.ru/handbook/algorithms>

# **Решение**

Выучите принципы SOLID наизусть.

Вот как расшифровывается акроним SOLID:

* S: Single Responsibility Principle (Принцип единственной ответственности).
* O: Open-Closed Principle (Принцип открытости-закрытости).
* L: Liskov Substitution Principle (Принцип подстановки Барбары Лисков).
* I: Interface Segregation Principle (Принцип разделения интерфейса).
* D: Dependency Inversion Principle (Принцип инверсии зависимостей).

Попробуйте по памяти воспроизвести примеры JS кода для демонстрации каждого из

этих принципов.

## **Принцип единственной ответственности**

Класс должен быть ответственен лишь за что-то одно. Если класс отвечает за решение нескольких задач, его подсистемы, реализующие решение этих задач, оказываются связанными друг с другом. Изменения в одной такой подсистеме ведут к изменениям в другой.

class Animal {

   constructor(name: string){ }

   getAnimalName() { }

}

class AnimalDB {

   getAnimal(a: Animal) { }

   saveAnimal(a: Animal) { }

}

## **Принцип открытости-закрытости**

Программные сущности (классы, модули, функции) должны быть открыты для расширения, но не для модификации.

class Animal {

       makeSound();

       //...

}

class Lion extends Animal {

   makeSound() {

       return 'roar';

   }

}

class Squirrel extends Animal {

   makeSound() {

       return 'squeak';

   }

}

class Snake extends Animal {

   makeSound() {

       return 'hiss';

   }

}

//...

function AnimalSound(a: Array<Animal>) {

   for(int i = 0; i <= a.length; i++) {

       a[i].makeSound();

   }

}

AnimalSound(animals);

## **Принцип подстановки Барбары Лисков**

Цель этого принципа заключаются в том, чтобы классы-наследники могли бы использоваться вместо родительских классов, от которых они образованы, не нарушая работу программы. Если оказывается, что в коде проверяется тип класса, значит принцип подстановки нарушается.

function AnimalLegCount(a: Array<Animal>) {

   for(let i = 0; i <= a.length; i++) {

**a[i].LegCount();**

   }

}

AnimalLegCount(animals);

class Animal {

       makeSound();

**LegCount();**

       //...

}

class Lion extends Animal {

   makeSound() {

       return 'roar';

   }

**LegCount();**

}

## **Принцип разделения интерфейса**

*Создавайте узкоспециализированные интерфейсы, предназначенные для конкретного клиента. Клиенты не должны зависеть от интерфейсов, которые они не используют.*  
  
Этот принцип направлен на устранение недостатков, связанных с реализацией больших интерфейсов.

interface Shape {

   draw();

}

interface ICircle {

   drawCircle();

}

interface ISquare {

   drawSquare();

}

interface IRectangle {

   drawRectangle();

}

interface ITriangle {

   drawTriangle();

}

class Circle implements ICircle {

   drawCircle() {

       //...

   }

}

class Square implements ISquare {

   drawSquare() {

       //...

   }

}

class Rectangle implements IRectangle {

   drawRectangle() {

       //...

   }

}

class Triangle implements ITriangle {

   drawTriangle() {

       //...

   }

}

class CustomShape implements Shape {

  draw(){

     //...

  }

}

## **Принцип инверсии зависимостей**

1. Модули верхних уровней не должны зависеть от модулей нижних уровней. Оба типа модулей должны зависеть от абстракций.
2. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.

interface Connection {

   request(url: string, opts:any);

}

class Http {

   constructor(private httpConnection: Connection) { }

   get(url: string , options: any) {

       this.httpConnection.request(url,'GET');

   }

   post() {

       this.httpConnection.request(url,'POST');

   }

   //...

}

class XMLHttpService implements Connection {

   const xhr = new XMLHttpRequest();

   //...

   request(url: string, opts:any) {

       xhr.open();

       xhr.send();

   }

}

class NodeHttpService implements Connection {

   request(url: string, opts:any) {

       //...

   }

}

class MockHttpService implements Connection {

   request(url: string, opts:any) {

       //...

   }

}

## **Реализовал алгоритм Флойда–Уоршелла на JavaScript.**

// JavaScript реализация алгоритма Флойда–Уоршелла

var INF = 99999;

class AllPairShortestPath {

        constructor() {

        this.V = 4;

        }

        floydWarshall(graph) {

        var dist = Array.from(Array(this.V), () => new Array(this.V).fill(0));

        var i, j, k;

// инициализация

        for (i = 0; i < this.V; i++) {

            for (j = 0; j < this.V; j++) {

            dist[i][j] = graph[i][j];

            }

        }

        //Добавляем все вершины

        for (k = 0; k < this.V; k++) {

            // Перебираем все вершины от одного к другому

            for (i = 0; i < this.V; i++) {

            // Перебираем все вершины назначения для выбранных ресурсов

            for (j = 0; j < this.V; j++) {

                // если вершина k короче

                // части от i до j, тогда обновляем значение dist[i][j]

                if (dist[i][k] + dist[k][j] < dist[i][j]) {

                dist[i][j] = dist[i][k] + dist[k][j];

                }

            }

            }

        }

        // выводим матрицу кратчайших растояний

        this.printSolution(dist);

        }

        printSolution(dist) {

        document.write(

            "Следующая матрица показывает кратчайшие " +

            " расстояния между каждой парой вершин <br>"

        );

        for (var i = 0; i < this.V; ++i) {

            for (var j = 0; j < this.V; ++j) {

            if (dist[i][j] == INF) {

                document.write(" INF ");

            } else {

                document.write("  " + dist[i][j] + " ");

            }

            }

            document.write("<br>");

        }

        }

    }

    // Driver Code

    /\* Let us create the following

        weighted graph

             10

        (0)------->(3)

        |          /|\

        5 |         |

        |         | 1

        \|/         |

        (1)------->(2)

              3        \*/

    var graph = [

        [0, 5, INF, 10],

        [INF, 0, 3, INF],

        [INF, INF, 0, 1],

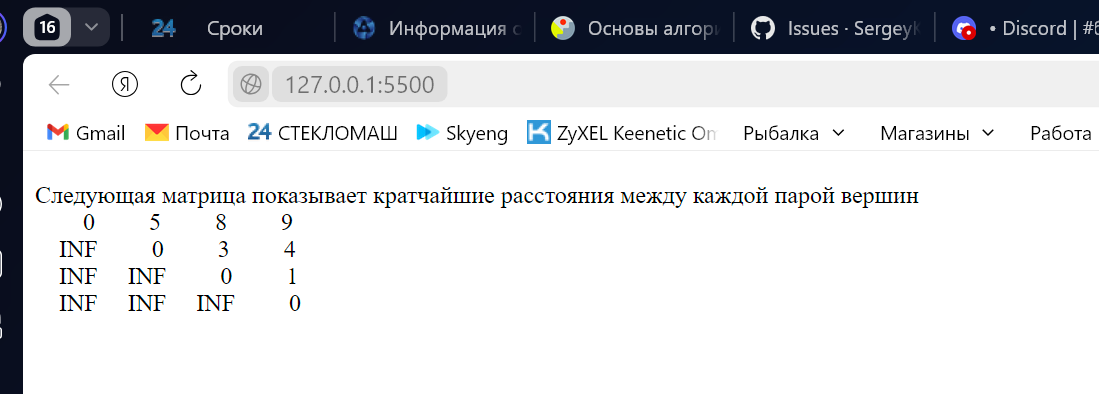
        [INF, INF, INF, 0],

    ];

    var a = new AllPairShortestPath();

    // вывод результата

    a.floydWarshall(graph);



## **Реализовал с использованием TypeScript паттерны Строитель, Прокси, Медиатор. Привел примеры использования собственных классов.**

**паттерн Строитель(Builder)**

interface User {

  name: string;

  age: number;

}

class UserBuilder {

  private readonly \_user: User;

  constructor() {

    this.\_user = {

      name: "",

      age: 0

    };

  }

  name(name: string): UserBuilder {

    this.\_user.name = name;

    return this;

  }

  age(age: number): UserBuilder {

    this.\_user.age = age;

    return this;

  }

  build(): User {

    return this.\_user;

  }

}

const userWithName: User = new UserBuilder()

  .name("John")

  .build();

document.write("user= "+ userWithName.name);

# **паттерн Прокси(Proxy)**

# 

// A Proxy Concept Example

interface ISubject {

    // An interface implemented by both the Proxy and Real Subject

    request(): void

    // A method to implement

}

class RealSubject implements ISubject {

    // The actual real object that the proxy is representing

    enormousData: number[]

    constructor() {

        // hypothetically enormous amounts of data

        this.enormousData = [1, 2, 3]

    }

    request() {

        return this.enormousData

    }

}

class ProxySubject implements ISubject {

    // In this case the proxy will act as a cache for

    // `enormous\_data` and only populate the enormous\_data when it

    // is actually necessary

    enormousData: number[]

    realSubject: RealSubject

    constructor() {

        this.enormousData = []

        this.realSubject = new RealSubject()

    }

    request() {

        // Using the proxy as a cache, and loading data into it only if

        // it is needed

        if (this.enormousData.length === 0) {

            console.log('pulling data from RealSubject')

            this.enormousData = this.realSubject.request()

            return this.enormousData

        }

        console.log('pulling data from Proxy cache')

        return this.enormousData

    }

}

// The Client

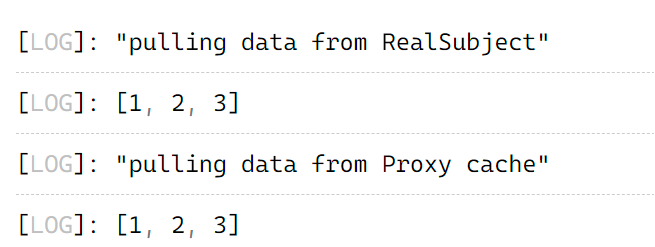
const PROXY\_SUBJECT = new ProxySubject()

// Use the Subject. First time it will load the enormous amounts of data

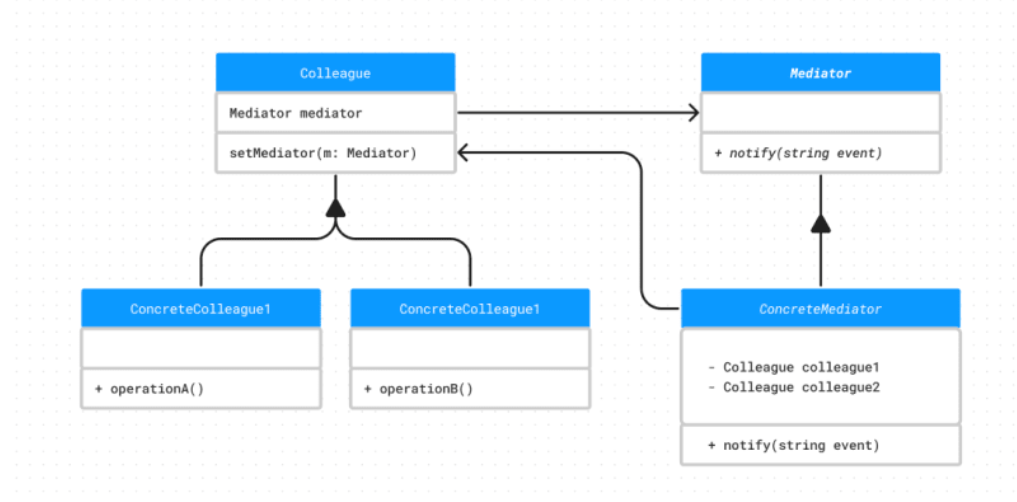
console.log(PROXY\_SUBJECT.request())

// Use the Subject again, but this time it retrieves it from the local cache

console.log(PROXY\_SUBJECT.request())

****

**паттерн Медиатор (mediator)**

****

// Mediator interface

interface ChatMediator {

  registerUser(user: User): void;

  sendMessage(message: string, sender: User): void;

}

// Concrete mediator

class ChatRoom implements ChatMediator {

  private users: User[] = [];

  registerUser(user: User) {

    this.users.push(user);

  }

  sendMessage(message: string, sender: User) {

    this.users.forEach((user) => {

      if (user !== sender) {

        user.receiveMessage(message);

      }

    });

  }

}

// User interface

interface User {

  name: string;

  mediator: ChatMediator;

  sendMessage(message: string): void;

  receiveMessage(message: string): void;

}

// Concrete user

class ChatUser implements User {

  constructor(

    public name: string,

    public mediator: ChatMediator,

  ) {

    mediator.registerUser(this);

  }

  sendMessage(message: string) {

    this.mediator.sendMessage(message, this);

  }

  receiveMessage(message: string) {

    console.log(`${this.name} received message: ${message}`);

  }

}

const chatMediator: ChatMediator = new ChatRoom();

const user1: User = new ChatUser("User1", chatMediator);

const user2: User = new ChatUser("User2", chatMediator);

user1.sendMessage("Hello!");

user2.sendMessage("Hi there!");

****

## **Мой класс пример**

type TObject = {

    name:string,

    type\_obj:string,

    stereotype:string[],

    notes:string,

    status:string,

    version:number,

    author:string,

    };

type Requirement = TObject & {

    CL\_STS\_REQ?:number,

    }

type Artifact = TObject & {

    CL\_STS\_ART?:number,

    }

const req01: Requirement = {

    name:'TestReq01',

    type\_obj:'Requirement',

    stereotype:['EPC'],

    notes:'Тестовое требование',

    status:'Proposed',

    version:1.1,

    author:'Sergey Kozhevnikov',

    CL\_STS\_REQ:1,

    };

    console.log(req01.name)

const art01: Artifact = {

    name:'TestArt01',

    type\_obj:'Artifact',

    stereotype:['EPC'],

    notes:'Тестовый артифакт ',

    status:'Proposed',

    version:1.0,

    author:'Sergey Kozhevnikov',

    CL\_STS\_ART:110,

    };

    console.log(art01.name)

//class

//abstract

class A\_TObject{

    name:string;

    notes!: string;

    status!:string;

    version!:number;

    author!:string;

    constructor(name: string){

        this.name=name;

    };

getInfo() :void {

    console.log('Object 1 = '+ this.name)

    }

};

class cl\_TObject extends A\_TObject {

    private type\_obj!:string;

    private stereotype!: string[];

constructor(name: string,type\_obj:string,stereotype:string[]){

        super(name);

        this.type\_obj=type\_obj;

        this.stereotype=stereotype;

    }

getInfo(): void {

        console.log('Object 2 = '+ this.name +','+ this.type\_obj +','+ this.stereotype)

    }

}

let \_req02 = new A\_TObject('TestReq10100');

\_req02.getInfo();

let \_req01 = new cl\_TObject('TestReq10000','Requirement',['ЕПС']);

\_req01.getInfo();