Оглавление

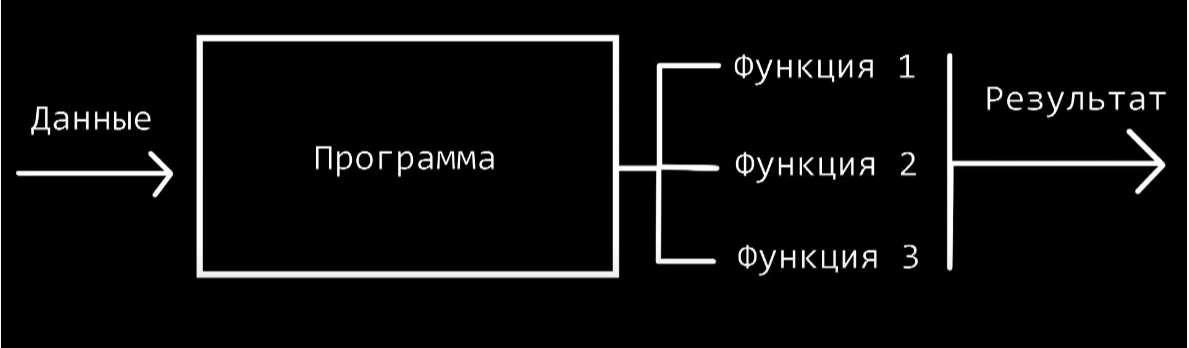
[Процедурный подход 1](#_Toc92306041)

[ООП 1](#_Toc92306042)

[Инкапсуляция и сокрытие 1](#_Toc92306043)

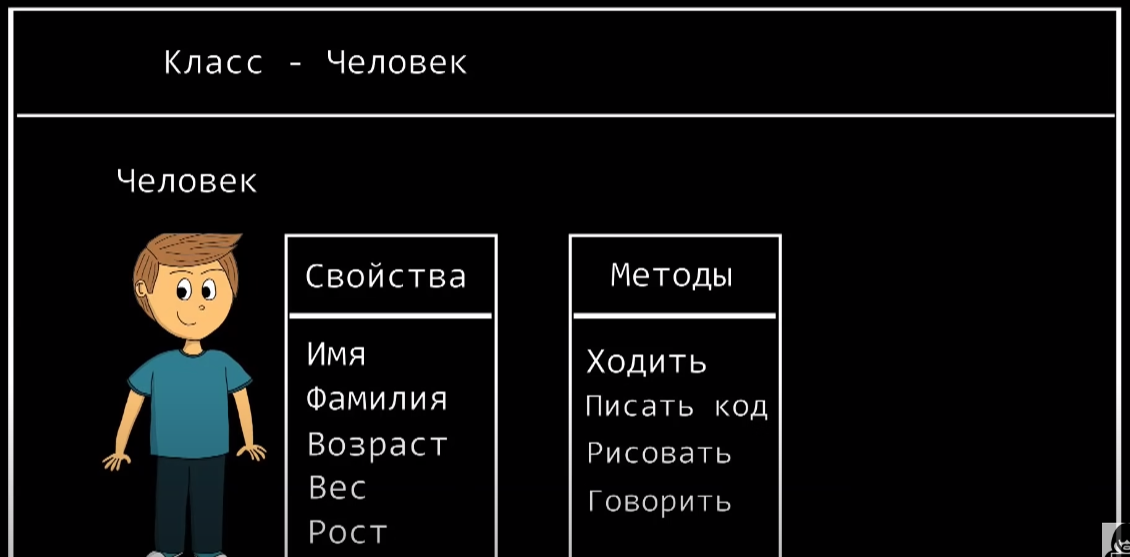
[Наследование 2](#_Toc92306044)

# Процедурный подход



Устаревший способ, плох при большом кол-ве кода.

# ООП



Разбитие на классы и методы

# Инкапсуляция и сокрытие

Есть классы со своими свойствами и методами. Класс – капсула. Модификаторы доступа: public, private. Private можно вызвать только из класса. (В js и ts в используют \_ перед названием свойства) Приватные свойства класса получают гетерами (get) и устанавливают значение сеттерами (set). Если нет необходимости в изменении свойства (например id), то set не создаем.

Пример кода:

class User {

    private \_username;

    private \_password;

    private \_id;

    constructor(username, password){

        this.\_username = username;

        this.\_password = password;

        this.\_id = Math.random();

    }

    get username() {

        return this.\_username;

    }

    set username(value) {

        this.\_username = value;

    }

    get password() {

        return this.\_password;

    }

    set password(value) {

        this.password = value;

    }

    get id() {

        return this.\_id;

    }

}

const user = new User ('leon', 'qwerty');

console.log(user);

console.log(user.username);

user.username = 'not leon';

console.log(user.username);

# Наследование

Класс человек (Имя Фам Возраст) **→** Класс работник (Инн, снилс и тд) **→** класс разработчик (яз прог, уровень и тд).

Пример:

class User {

    #\_username;

    #\_password;

    #\_age;

    constructor(username, password, age){

        this.\_username = username;

        this.\_password = password;

        this.\_age = age;

    }

    get username() {

        return this.\_username;

    }

    set username(value) {

        this.\_username = value;

    }

    get password() {

        return this.\_password;

    }

    set password(value) {

        this.password = value;

    }

    get age() {

        return this.\_age;

    }

    set age(value) {

        if (value > 0){ this.\_age = value;}

        else {this.\_age = 0;}

    }

}

class Emploer extends User {

    #\_INN

    #\_Snils

    constructor (username, password, age, INN, Snils) {

        super(username, password, age);

        this.\_INN = INN;

        this.\_Snils = Snils;

    }

    get INN() {

        return this.\_INN;

    }

    set INN(value) {

        this.\_INN = value;

    }

    get SNILS() {

        return this.\_Snils;

    }

    set SNILS(value) {

        this.\_Snils = value;

    }

}

class Developer extends Emploer {

    #\_Lang

    #\_LVL

    constructor (username, password, age, INN, Snils, lang, lvl) {

        super (username, password, age, INN, Snils);

        this.\_lang = lang;

        this.\_lvl = lvl;

    }

    get lang() {

        return this.\_lang;

    }

    set lang(value) {

        this.\_lang = value;

    }

    get lvl() {

        return this.\_lvl;

    }

    set lvl(value) {

        this.\_lvl = value;

    }

}

const developer = new Developer ('Leo', 'qwerty', 22, 620260, 112233, 'js', 'middle');

console.log(developer);

console.log(developer.username);

developer.INN = 111111;

console.log(developer.INN);

# Полиморфизм

2 вида: Параметрический (истинный) и ad-hoc (мнимый). Не оч понял о чем речь, если что вот источник: <https://habr.com/ru/post/37576/>

# Агрегация и композиция

Композиция

Класс автомобиль. Использует объект другого класса двигатель. Массив из 4-х объектов колесо. Суть в том, что они создаются неразделимыми в классе автомобиль. Что это значит я так и не понял. А не, понял, в конструкторе свойства объявляются примерно так this.engine = new Engine(), а не через переменные.

Агрегация

К классу автомобиль можно добавить объект из вне.

# Абстрактные классы и интерфейсы.

Вообще хз, дикий бэк оставил на потом.