МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра автоматизації проектування енергетичних процесів і систем

Лабораторна робота №2

з дисципліни «Системи моніторингу стану мережі об'єктів у реальному часі»

**Виконав:**

студент 5-го курсу

ТЕФ, гр. ТР-02мп

Круглий Д.В.

**Перевірив:**

проф. Сліпченко В. Г.

**Київ-2021**

**Тема:** модель функціонування ліфта.

**Мета:** побудова моделі функціонування ліфта та формалізувати умови блокування та активізації ліфта та пасажирів.

**Постановка задачі:** потрібно запрограмувати завдання, написавши монітор "Ліфт", і забезпечити справедливе виділення ліфта для пасажирів в залежності від їх появи в системі. Ліфт слід розглядати як процес сервер.

Для виконання даної лабораторної роботи мною була вибрана мова програмування JavaScript в середовищі виконання Node.js. Для моделювання роботи ліфта було обрано наступний алгоритм: при звільненні ліфта, він прямує до найближчого пасажира, після чого рухається за його напрямком. Ліфт має максимальну вантажопідйомність в п’ять людей, тому при переповненні він не підбирає людей, поки не звільниться хоча б одне вільне місце. Кінцева зупинка ліфта задається першим пасажиром, або останнім поверхом, отже для зміни напрямку руху, ліфт повинен висадати останнього пасажира.

**Представлення коду програми**

**Passenger.js**

const { getRandomArbitrary} = require('./utils');

const Elevator = require('./Elevator');

module.exports = class Passenger{

    constructor(elevator, id) {

        this.elevator = elevator;

        this.id = id;

        this.name = `client-${this.id}`;

        this.destinationFloor = null;

        this.startFloor = this.\_calculateDestinationFloor();

        this.direction = null;

        this.arrived = true;

        this.\_checkIsArrived();

    };

    CHECK\_IS\_ARRIVED\_INTERVAL = 100;

    \_checkIsArrived() {

        setInterval(() => { if (this.arrived) {this.\_goToFloor() }}, this.CHECK\_IS\_ARRIVED\_INTERNAL);

    }

    \_goToFloor(){

        this.destinationFloor = this.\_calculateDestinationFloor();

        this.direction = this.startFloor > this.destinationFloor ? Elevator.DOWN\_DIRECTION : Elevator.UP\_DIRECTION;

        this.arrived = false;

        setTimeout(() => this.elevator.passengerRequest(this), math.random \* 1000);

    }

    \_setFloor(destinationFloor){

        this.startFloor = destinationFloor;

    }

    \_calculateDestinationFloor(){

        return getRandomArbitrary(Elevator.FIRST\_FLOOR, Elevator.MAX\_FLOOR);

    }

};

**Elevator.js**

module.exports = class Elevator {

    static BUSY\_STATE = "BUSY";

    static FREE\_STATE = "FREE";

    static UP\_DIRECTION = "UP";

    static DOWN\_DIRECTION = "DOWN";

    static MAX\_PASSENGERS\_COUNT = 3;

    static FIRST\_FLOOR = 1;

    static MAX\_FLOOR = 5;

    passengersInElevator = [];

    passengersQueues = {

        1: [],

        2: [],

        3: [],

        4: [],

        5: [],

    };

    state = {

        state: Elevator.FREE\_STATE,

        currentFloor: 1,

        direction: Elevator.UP\_DIRECTION,

    };

    passengerRequest(passenger){

        this.\_addPassengerToQueue(passenger);

            if (this.\_isFree()){

                this.\_goToFloorWithoutPassengers(passenger.startFloor);

                if (passenger.direction == Elevator.UP\_DIRECTION) this.\_goUp();

                if (passenger.direction == Elevator.DOWN\_DIRECTION) this.\_goDown();

                return;

        }

    }

    \_isFree(){return this.state.state == Elevator.FREE\_STATE}

    \_isFull(){return this.passengersInElevator.length >= Elevator.MAX\_PASSENGERS\_COUNT}

    \_addPassengerToQueue(passenger){ this.passengersQueues[passenger.startFloor].push(passenger)}

    \_goDown(){

        if (this.state.currentFloor == Elevator.FIRST\_FLOOR) throw new Error('You are on the first floor already');

        this.state.state = Elevator.BUSY\_STATE;

        this.state.direction = Elevator.DOWN\_DIRECTION;

        const passengerWithSameDirection = this.passengersQueues[this.state.currentFloor].filter(p => p.direction == Elevator.DOWN\_DIRECTION);

        this.\_addPassengerToElevator(passengerWithSameDirection);

        const arrivedPassengers = this.passengersInElevator.filter(p => p.destinationFloor == this.state.currentFloor);

        this.\_printArrivedPassengers(arrivedPassengers);

        this.\_removeArrivedPassengers();

        this.\_printElevatorState();

        this.state.currentFloor -= 1;

        if (this.state.currentFloor == Elevator.FIRST\_FLOOR){

            this.state.state = Elevator.FREE\_STATE;

            return;

        }else{

            this.\_goDown();

        }

    }

    \_goUp(){

        if (this.state.currentFloor == Elevator.MAX\_FLOOR) throw new Error('You are on the last floor already');

        this.state.state = Elevator.BUSY\_STATE;

        this.state.direction = Elevator.UP\_DIRECTION;

        const passengerWithSameDirection = this.passengersQueues[this.state.currentFloor].filter(p => p.direction == Elevator.UP\_DIRECTION);

        this.\_addPassengersToElevator(passengerWithSameDirection);

        const arrivedPassengers = this.passengersInElevator.filter(p => p.destination == this.state.currentFloor);

        this.\_printArrivedPassengers(arrivedPassengers);

        this.\_removeArrivedPassengers();

        this.\_printElevatorState();

        this.state.currentFloor += 1;

        if (this.state.currentFloor == Elevator.MAX\_FLOOR){

            this.state.state = Elevator.FREE\_STATE;

        }else{

            this.\_goUp();

        }

    }

    \_addPassengersToElevator(passengerWithSameDirection){

        const freePlaces = Elevator.MAX\_PASSENGERS\_COUNT - this.passengersInElevator.length;

        const newPassengers = passengerWithSameDirection.slice(0, freePlaces);

        this.passengersInElevator = this.passengersInElevator.concat(newPassengers);

        this.\_removePassengersFromQueue(newPassengers);

    }

    \_removeArrivedPassengers(){

        this.passengersInElevator = this.passengersInElevator.filter(p => p.destinationFloor != this.state.currentFloor);

    }

    \_removePassengersFromQueue(passengers){

        passengers.forEach(pas =>{

            this.passengersQueues[pas.startFloor] = this.passengersQueues[pas.startFloor].filter(p => p.id != pas.id);

            pas.arrived = true;

            pas.startFloor = this.state.currentFloor;

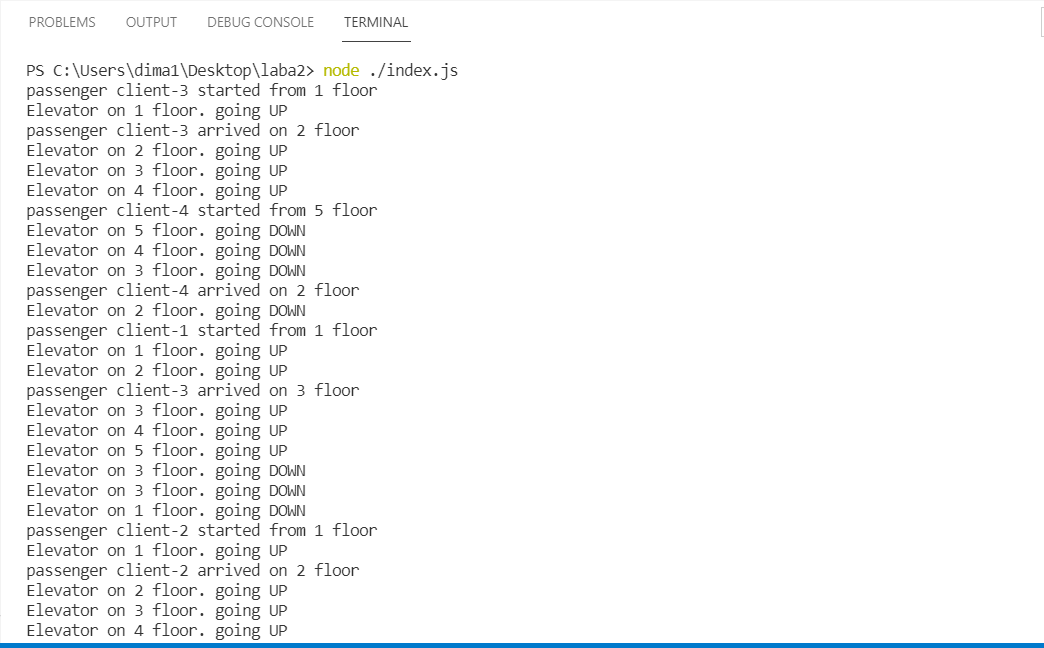
            this.\_printStartingPassenger(pas);

        })

    }

};

**Приклад роботи програми**



**Висновок:** виконавши дану лабораторну роботу, я побудував модель функціонування ліфта та формалізував умови блокування та активізації ліфта та пасажирів.