МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное образовательное учреждение высшего образования

САНКТ – ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ

КАФЕДРА № 42

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  |  |  | С. Ю. Гуков |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №7  Телефонная книга |
|  |
| по курсу: АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 4321 |  |  |  | К.А. Лебедев |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург, 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc178416272)

[2 Задание 4](#_Toc178416273)

[3 Ход разработки 5](#_Toc178416274)

[4 Исходный код программы 6](#_Toc178416275)

[5 Результаты работы программы 9](#_Toc178416276)

[6 Вывод 14](#_Toc178416277)

**1 Цель работы**

Реализовать структуру данных, эффективно обрабатывающую запросы вида add number name, del number и find number.

**2 Задание**

Реализовать простую телефонную книгу, которая будет считывать количество команд и обрабатывать заранее установленные команды, выводя очередь ответов на запросы. Дополнительные задания: команда undoи защита от дубликатов имен.

**3 Ход разработки**

Были реализованы классы очереди, телефонной книги и задачи, написана утилита для прочтения чисел из командной строки на языке TypeScript, интегрирован компилятор в JavaScript, который сразу собирает TypeScript проект без предварительной компиляции.

**4 Исходный код программы**

*// index.ts*  
  
import Task from "./lib/Task";

new Task(); *// инициализация задачи*

*// utils/readline.ts*

import readline from "node:readline";

*// readline интерфейс*

export const readlineInterface = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

*// interfaces/interfaces.ts*

*// Дженерик очереди*

export interface IQueue<T> {

value: T[];

push: (item: T) => void;

shift: () => T | undefined;

}

*// Тип для сущности контакта*

export type TContact = {

phone: string;

name: string;

};

*// Тип для книги контактов*

export interface IBook {

value: Map<TContact["phone"], TContact["name"]>;

add: (contact: TContact) => void;

find: (phone: TContact["phone"]) => TContact["name"] | undefined;

del: (phone: TContact["phone"]) => void;

}

*// config/index.ts*

*// enum с операциями*

export enum Operations {

ADD = "add",

FIND = "find",

DELETE = "del",

UNDO = "undo",

}

*// lib/Queue.ts*

import { IQueue } from "../interfaces/interfaces";

*// Класс очереди для работы с массивами дженерного типа*

class Queue<T> *implements* IQueue<T> {

*public* value: T[]; *// значение очереди*

constructor(arr: T[]) {

this.value = arr; *// создание массива*

}

*// push в очередь*

*public* push(item: T) {

this.value.push(item);

}

*// shift из очереди*

*public* shift() {

return this.value.shift();

}

}

export default Queue;

*// lib/Book.ts*

import { IBook, TContact } from "../interfaces/interfaces";

*// Класс телефонной книги*

class Book *implements* IBook {

*public* value: Map<string, string>;

constructor(arr: [string, string][]) {

this.value = new Map<string, string>(arr); *// итерируемая коллекция*

}

*// матчер дубликата по имени*

*public* duplicateNameMatcher(findName: string): boolean {

const flag = Array.from(this.value, ([\_, name]) => name).reduce(

(acc, item) => {

acc &&= item !== findName;

return acc;

},

true

);

return !flag;

}

*// добавить контакт*

*public* add(contact: TContact): void {

if (!this.duplicateNameMatcher(contact.name))

this.value.set(contact.phone, contact.name);

}

*// удалить контакт*

*public* del(phone: string): void {

this.value.delete(phone);

}

*// найти контакт*

*public* find(phone: string): TContact["name"] | undefined {

return this.value.get(phone);

}

}

export default Book;

*// lib/Task.ts*

import { Operations } from "../config";

import { readlineInterface } from "../utils/readline";

import Book from "./Book";

import Queue from "./Queue";

class Task {

*private* res: Book;

*private* resCountCommands: number;

*private* callbackQueue: Queue<() => void>;

*private* countCommands: number;

*private* prevRes: Book;

*// перечисление свойств*

constructor() {

this.init(); *// инициализация*

this.res = new Book([]); *// создание контактной книги*

this.prevRes = new Book(Array.from(this.res.value)); *// предыдущее состояние контактной книги*

this.resCountCommands = 0; *// итоговое количество команд*

this.callbackQueue = new Queue([] as (() => void)[]); *// очередь запросов*

this.countCommands = 0; *// текущее количество команд*

}

*// сохранение предыдущего состояния контактной книги*

*private* setPrevRes() {

this.prevRes = this.res;

}

*// метод прочтения команды*

readElement() {

if (this.countCommands < this.resCountCommands) {

readlineInterface.question("", element => {

this.parse(element); *// парсинг команды*

this.countCommands++; *// увеличение количества команд*

this.readElement(); *// рекурсия*

});

} else {

while (this.callbackQueue.value.length > 0) {

(this.callbackQueue.shift() as () => void)(); *// очистка очереди команд*

}

readlineInterface.close(); *// закрытие интерфейса readline*

}

}

*// парсинг команды*

parse(command: string) {

const [operation, phone, name] = command.split(" "); *// деструктуризация команды*

switch (operation) {

*// добавление контакта*

case Operations.ADD: {

if (isNaN(Number(phone)) || !name) {

console.warn("Не корректные данные!");

readlineInterface.close();

} else {

this.setPrevRes();

this.res.add({ phone, name });

}

break;

}

*// поиск контакта*

case Operations.FIND: {

if (isNaN(Number(phone))) {

console.warn("Не корректный телефон!");

readlineInterface.close();

} else {

const findValue = this.res.find(phone);

this.callbackQueue.push(() =>

console.log(findValue ?? "not found")

);

}

break;

}

*// удаление контакта*

case Operations.DELETE: {

if (isNaN(Number(phone))) {

console.warn("Не корректный телефон!");

readlineInterface.close();

} else {

this.setPrevRes();

this.res.del(phone);

}

break;

}

*// откат истории телефонной книги*

case Operations.UNDO: {

if (this.prevRes.value.entries.length === 0) {

console.warn("Контактная книга пуста!");

readlineInterface.close();

} else {

this.res = this.prevRes;

}

break;

}

*// обработка остальных случаев*

default:

break;

}

}

*// инициализация задачи*

init() {

readlineInterface.question("", (length: string) => {

this.resCountCommands = parseInt(length); *// чтение количества команд*

if (isNaN(this.resCountCommands) || this.resCountCommands <= 0) {

console.log(

"Пожалуйста, введите корректное положительное число."

);

readlineInterface.close();

return;

}

this.readElement(); *// вызов рекурсии*

});

}

}

export default Task;

**5 Результаты работы программы**

Результат работы программы включает в себя вывод очереди команд

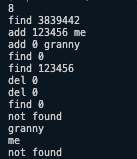


Рисунок 1 – тест программы №1

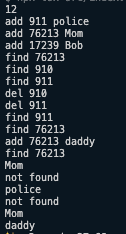


Рисунок 2 – тест программы №2

**6 Вывод**

Телефонная книга – реализация простейших CRUD операций с дополнительным функционалом. Данная задача позволяет получить ценный опыт в разработке современных систем программного обеспечения, а также получить навыки по принципам работы с базами данных и управления данными.

В результате выполнения лабораторной работы был изучен принцип работы CRUD (CREATE, READ, UPDATE, DELETE) систем, написаны утилитарные функции для обработки входящих команд.