Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Кубанский государственный технологический университет»

(ФГБОУ ВО «КубГТУ»)

Факультет информационных технологий и кибербезопасности

Кафедра информационных систем и программирования

**ОТЧЕТ**

по лабораторной работе №7.

Тема работы: «Модульное тестирование объектно-ориентированных программ».

Наименование дисциплины: «Тестирование и отладка программного обеспечения».

Подготовил:

Студент группы 23-КБ-ПР1

Булгаков В. В.

Краснодар

2025

**Цель работы**

Цель работы – изучить подход к автоматизации процесса модульного тестирования объектно-ориентированных программ.

**Задание**

1) Создать класс (в соответствии с вариантом задания из п.5), реализующий работу с АСД (абстрактная структура данных).

2) Построить диаграмму состояний класса.

3) Составить тестовые требования к методам класса не основе полученной диаграммы состояний.

4) Определить наборы тестов на основе полученных тестовых требований.

5) Создать проект для автоматизированного модульного тестирования на основе тестовых наборов.

6) Запустить тестирование и проверить результаты работы.

7) Составить отчет о результатах проведенного тестирования.

Необходимо создать множество на основе односвязного списка.

**Ход работы**

1. Создал класс и интерфейс.

class LinkedListNode {

    constructor(value) {

        this.value = value;

        this.next = null;

    }

}

class LinkedList {

    constructor() {

        this.head = null;

    }

    prepend(value) {

        const newNode = new LinkedListNode(value);

        newNode.next = this.head;

        this.head = newNode;

    }

    remove(value) {

        if (!this.head) return false;

        if (this.head.value === value) {

            this.head = this.head.next;

            return true;

        }

        let current = this.head;

        while (current.next) {

            if (current.next.value === value) {

                current.next = current.next.next;

                return true;

            }

            current = current.next;

        }

        return false;

    }

    contains(value) {

        let current = this.head;

        while (current) {

            if (current.value === value) {

                return true;

            }

            current = current.next;

        }

        return false;

    }

    toArray() {

        const result = [];

        let current = this.head;

        while (current) {

            result.push(current.value);

            current = current.next;

        }

        return result;

    }

    size() {

        let count = 0;

        let current = this.head;

        while (current) {

            count++;

            current = current.next;

        }

        return count;

    }

    clear() {

        this.head = null;

    }

    getFirst() {

        if (!this.head) {

            throw new Error('List is empty');

        }

        return this.head.value;

    }

    getLast() {

        if (!this.head) {

            throw new Error('List is empty');

        }

        let current = this.head;

        while (current.next) {

            current = current.next;

        }

        return current.value;

    }

}

class MySet {

    constructor() {

        this.list = new LinkedList();

    }

    add(item) {

        if (this.list.contains(item)) {

            return false;

        }

        this.list.prepend(item);

        return true;

    }

    contains(item) {

        return this.list.contains(item);

    }

    delete(item) {

        return this.list.remove(item);

    }

    size() {

        return this.list.size();

    }

    isEmpty() {

        return this.list.size() === 0;

    }

    clear() {

        this.list.clear();

    }

    toArray() {

        return this.list.toArray();

    }

    getFirst() {

        return this.list.getFirst();

    }

    getLast() {

        return this.list.getLast();

    }

}

module.exports = MySet;

2) Построил диаграмму состояний класса.



3) Составил тестовые требования к методам класса на основе полученной диаграммы состояний.

1. Конструктор new MySet() – состояние пустое множество (size = 0, head = null);
2. Метод Add(value) – при добавлении первого элемента – множество переходит в состояние непустое множество (size = 1, head!=null), при попытке добавить дубликат – состояние не меняется, при добавлении нового элемента – размер увеличивается на 1, состояние остаётся непустым множеством;
3. Метод Delete(value) – если элемент есть, если он единственный, то переходит в состояние пустое множество, иначе – размер уменьшиться на 1, состояние останется непустым множеством. Если элемента нет, возвращается false. Состояние не меняется;
4. Метод Contains(value) – возвращает true, если элемент есть, иначе false. Состояние не меняется;
5. Метод size() – возвращает текущий размер множества, состояние не меняется;
6. Метод isEmpty() – возвращает true, если size = 0, иначе false. Состояние не меняется;
7. Метод clear() – очищает множество, переходит в состояние пустое множество;
8. Метод ToArray() – возвращает массив всех элементов множества, не меняет состояние;
9. Метод getFirst() – если множество пустое, то ошибка, иначе возвращает первый элемент (head.value). Состояние не меняется;
10. Метод getLast() – если множество пустое, то ошибка, иначе возвращает последний элемент. Состояние не меняется.

4-6) Создал тестовый класс для автоматизированного модульного тестирования.

const MySet = require('../src/MySet');

describe('MySet', () => {

    let set;

    beforeEach(() => {

        set = new MySet();

    });

    test('createEmptySet', () => {

        expect(set.isEmpty()).toBe(true);

        expect(() => set.getFirst()).toThrow('Set is empty');

        expect(() => set.getLast()).toThrow('Set is empty');

        expect(set.size()).toBe(0);

    });

    test('addItem', () => {

        set.add(2);

        expect(set.isEmpty()).toBe(false);

        expect(set.getFirst()).toBe(2);

        expect(set.getLast()).toBe(2);

        expect(set.size()).toBe(1);

        set.add(3);

        expect(set.isEmpty()).toBe(false);

        expect(set.getFirst()).toBe(3);

        expect(set.getLast()).toBe(2);

        expect(set.size()).toBe(2);

        set.add(4);

        set.add(3);

        expect(set.getLast()).toBe(2);

        expect(set.size()).toBe(3);

    });

    test('contains', () => {

        expect(set.contains(5)).toBe(false);

        set.add(5);

        set.add(3);

        set.add(2);

        expect(set.contains(5)).toBe(true);

        expect(set.contains(2)).toBe(true);

        expect(set.contains(10)).toBe(false);

    });

    test('removeItem', () => {

        expect(set.delete(5)).toBe(false);

        set.add(5);

        expect(set.delete(5)).toBe(true);

        expect(set.size()).toBe(0);

        set.add(5);

        set.add(3);

        set.add(2);

        expect(set.delete(2)).toBe(true);

        expect(set.size()).toBe(2);

        expect(set.delete(5)).toBe(true);

        expect(set.size()).toBe(1);

    });

    test('clear', () => {

        set.clear();

        expect(() => set.getFirst()).toThrow('Set is empty');

        expect(() => set.getLast()).toThrow('Set is empty');

        expect(set.size()).toBe(0);

        set.add(5);

        set.add(2);

        set.clear();

        expect(() => set.getFirst()).toThrow('Set is empty');

        expect(() => set.getLast()).toThrow('Set is empty');

        expect(set.size()).toBe(0);

    });

    test('isEmpty', () => {

        expect(set.isEmpty()).toBe(true);

    });

    test('getFirst', () => {

        expect(() => set.getFirst()).toThrow('Set is empty');

        set.add(5);

        expect(set.getFirst()).toBe(5);

    });

    test('getLast', () => {

        expect(() => set.getLast()).toThrow('Set is empty');

        set.add(5);

        expect(set.getLast()).toBe(5);

    });

    test('size', () => {

        expect(set.size()).toBe(0);

        set.add(5);

        expect(set.size()).toBe(1);

        set.add(3);

        expect(set.size()).toBe(2);

    });

    test('toArray', () => {

        expect(set.toArray()).toEqual([]);

        set.add(5);

        set.add(3);

        const arr = set.toArray();

        expect(arr).toContain(5);

        expect(arr).toContain(3);

        expect(arr.length).toBe(2);

    });

    test('remove возвращает false в классе LinkedListNode когда число не представлено (для 100% покрытия)', () => {

        set.add(5);

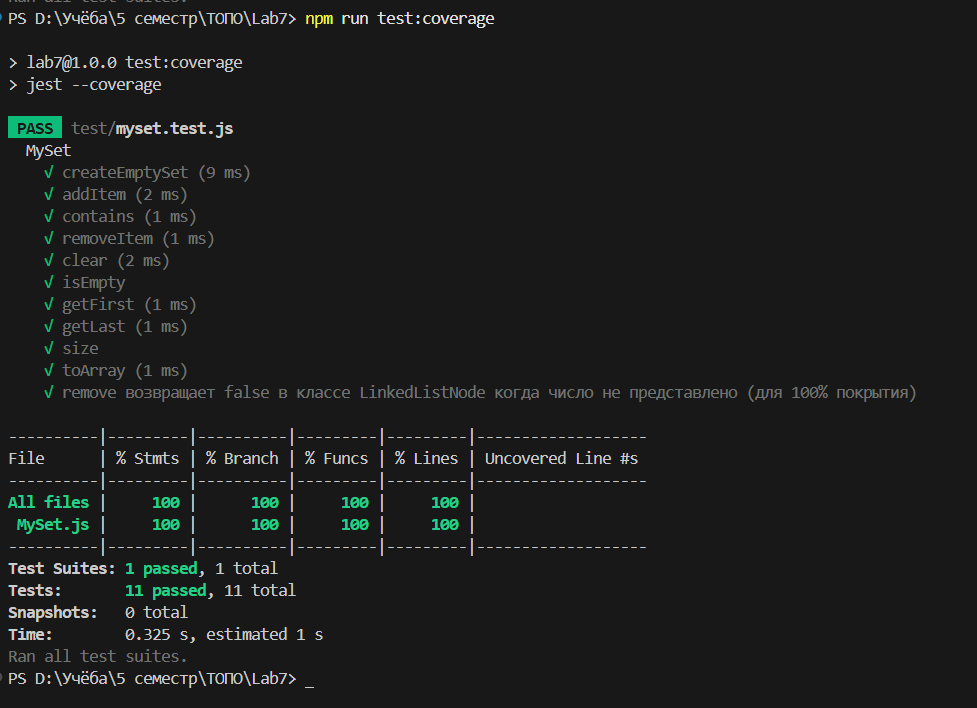
        set.add(10);

        expect(set.delete(15)).toBe(false);

        expect(set.size()).toBe(2);

    });

});



**Вывод**

В ходе выполнения данной лабораторной работы удалось изучить подход к автоматизации процесса модульного тестирования объектно-ориентированных программ.