**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра «Математическая Кибернетика и Информационные технологии»

Лабораторная работа №3

По дисциплине «Информационные технологии и программирование»

Выполнил: Студент группы

БВТ2203

Попов Александр

Москва

2023

**Цель работы:** Реализация класса HashTable на языке программирования Java.

**Задание 1:**

1. Создайте класс HashTable, который будет реализовывать хэш таблицу с помощью метода цепочек.

2. Реализуйте методы put(key, value), get(key) и remove(key), которые добавляют, получают и удаляют пары «ключ-значение» соответственно.

3. Добавьте методы size() и isEmpty(), которые возвращают количество элементов в таблице и проверяют, пуста ли она.

**Задание 2:**

*Вариант 4:* Реализация хэш-таблицы для хранения информации о книгах в библиотеке. Ключом будет ISBN книги, а значением - объект класса Book, содержащий информацию о названии, авторе и количестве копий. Необходимо реализовать операции вставки, поиска и удаления книги по ISBN.

**Ход работы**

import java.util.LinkedList;

*public* *abstract* *class* HashTable<K, V> {

*private* LinkedList<Entry<K, V>>[] table;

*private* int size;

    @SuppressWarnings("unchecked")

*public* HashTable() {

        table = (LinkedList<Entry<K, V>>[]) *new* LinkedList<?>[1000];

        size = 0;

    }

    @SuppressWarnings("unchecked")

*public* HashTable(int *initialCapacity*) {

        table = (LinkedList<Entry<K, V>>[]) *new* LinkedList<?>[*initialCapacity*];

        size = 0;

    }

*private* int hash(K *key*) {

*try*{

*return* *key*.hashCode() % (table.length);

        } *catch*(ArithmeticException *e*) {

            System.out.println("Your table is 0 length");

            System.exit(0);

*return* 0;

        }

    }

*public* void put(K *key*, V *value*) {

        int index = hash(*key*);

*if* (table[index] == null) {

            table[index] = *new* LinkedList<Entry<K, V>>();

        }

*for* (Entry<K, V> entry *:* table[index]) {

*if* (entry.getKey().equals(*key*)) {

                entry.setValue(*value*);

*return*;

            }

        }

        table[index].add(*new* Entry<K, V>(*key*, *value*));

        size++;

    }

*public* V get(K *key*) {

        int index = hash(*key*);

*if* (table[index] != null) {

*for* (Entry<K, V> entry *:* table[index]) {

*if* (entry.getKey().equals(*key*)) {

*return* entry.getValue();

                }

            }

        }

*return* null;

    }

*public* void remove(K *key*) {

        int index = hash(*key*);

*if* (table[index] != null) {

*for* (Entry<K, V> entry *:* table[index]) {

*if* (entry.getKey().equals(*key*)) {

                    table[index].remove(entry);

                    size--;

*return*;

                }

            }

        }

    }

*public* int size() {

*return* size;

    }

*public* boolean isEmpty() {

*return* size == 0;

    }

}

Файл 1. Класс HashTable.

Это основной класс HashTable, реализующий все методы, указанные в задании 1. Моя хэш-таблица является массивом с заданной длиной. Элементы этого массива – связанные списки. Такой тип данных выбран для разрешения коллизии. Сам связанный список состоит из элементов Entry<K, V> (Файл 2). Такой класс является универсальным за счет использования generic типов.

public *class* Entry<K, V> {

    K key;

    V value;

*public* Entry(K *key*, V *value*) {

        this.key = *key*;

        this.value = *value*;

    }

*public* K getKey() {

*return* this.key;

    }

*public* V getValue() {

*return* this.value;

    }

*public* void setValue(V *value*) {

        this.value = *value*;

    }

}

Файл 2. Класс Entry.

Класс Entry предназначен для создания элементов – пар «ключ-значение».

public *class* Book {

*private* String author;

*private* String title;

*private* int countCopy;

*public* Book() {

        this.author = "";

        this.title = "";

        this.countCopy = 0;

    }

*public* Book(String *author*, String *title*, int *countCopy*) {

        this.author = *author*;

        this.title = *title*;

        this.countCopy = *countCopy*;

    }

*public* String bookInfo() {

*return* this.author + "\n" + this.title + "\n" + this.countCopy;

    }

*public* String getAuthor() {

*return* this.author;

    }

*public* String getTitle() {

*return* this.title;

    }

*public* int getCountCopy() {

*return* this.countCopy;

    }

}

Файл 3. Класс Book.

Для второго задания мне понадобятся объекты класса Book. В нем находятся 3 поля, 3 геттера для этих полей, 2 конструктора (один по умолчанию) и метод bookInfo для корректоного отображения информации о книге.

public *class* BookHashTable extends HashTable<String, Book> {

*public* BookHashTable() {

        super();

    }

*public* BookHashTable(int *initialCapacity*) {

        super(*initialCapacity*);

    }

}

Файл 4. Класс BookHashtable.

Я реализовал отдельный класс для хэш таблицы книг. Он наследует стандартный класс HashTable, который мы приводим к типу HashTable<String, Book>.

public *class* Main {

*public* *static* void main(String[] *args*) {

        BookHashTable bht = *new* BookHashTable(100);

        bht.put("978-3-16-12345", *new* Book("L.N.Tolstoy", "Voyna i Mir", 100));

        Book book1 = bht.get("978-3-16-12345");

        System.out.println(book1.bookInfo());

        System.out.println(bht.isEmpty());

        System.out.println(bht.size());

        bht.remove("978-3-16-12345");

        System.out.println(bht.isEmpty());

    }

}

Файл 5. Класс Main.

В файле 5 приведено тестирование класса BookHashTable.

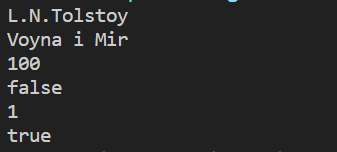
****

Рис. 1. Результат выполнения программы.

**Выводы**

1. Я освоил принцип работы хэш-таблицы, являющейся одной из самх востребованных в программировании.

2. Я написал свой класс HashTable, реализующий все основные методы стандартного класса, а также типизировал его для хранения информации о книгах.