МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

«МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И ИНФОРМАТИКИ»

(МТУСИ)

Кафедра «Математическая Кибернетика и Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

по дисциплине

«Информационные технологии и программирование»

на тему

“Хэш-таблицы”

Выполнил:

студент группы БВТ2302

Миронов А. А.

Москва, 2024 г.

Практическое задание номер 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.

Ход работы:

Начнём создание собственного класса, реализующего хэш-таблицу HashTab с импортирования встроенного класса в джаве LinkedList, который понадобится нам, чтобы справляться коллизиями (моментами, когда у разных ключей совпадает хэщкод, нам же всё равно нужно их куда то положить, будем класть в отсек хэш-таблицы с одинаковым индексом в linkedlist).

```
import java.util.LinkedList;

public class HashTab<K, V> {

    private LinkedList<KplusV<K, V>>[] indxList;
    private int size = 0;

    private class KplusV<Ke, Va> {
        Ke key;
        Va value;

        public KplusV(Ke ky, Va val) {
            this.key = ky;
            this.value = val;
        }
    }

    @SuppressWarnings("unchecked")
    public HashTab(int indxListLen) {
        indxList = (LinkedList<KplusV<K,V>>[]) new LinkedList[indxListLen];
    }

    public HashTab(){
        this(indxListLen:1000);
    }

    private int hash(K key) {

        return Math.abs(key.hashCode()) % indxList.length;
```

При указании типа данных класса таблицы, мы используем вот такие скобки < >, которые используются для обозначения неопределённых типов данных, т е при создании таблицы, мы можем назначить вместо переменных K и V разные типы данных, к которым будет относиться наш ключ и значение.

Далее создаём ссылку на массив linkedlistов, которые в качества типа данных будут принимать наш внутренний класс KplusV<K,V>, который мы создаём далее. Этот класс будет использоваться только для непосредственное хранения частной пары ключ-значение.

Реализуем конструктор для ссылки на массив ссылок списков, чтобы получить конкретное количество ссылок linkedlistов (сколько строк будет в нашей hashtab). Делаем конструктор по умолчанию на 1000 строк. Стоит отметить, что конструктор джава не разрешает использовать generic (неопределённые типы данных), поэтому обманываем его присваивая массиву linkedlistов generic тип с помощью casting, а также скрываем предупреждение «unchecked».

Теперь нам нужно написать функцию, которая рассчитывает хэшкод для каждого ключа. Используем функцию модуля, потому что в хэш таблице не может быть отрицательных строк, а также получаем остаток от деления на количетво строк хэш-таблицы, получая таким образом хэш код, совпадающий с номером определённой строки в таблице.

```
}


public void put(K k_, V v_) {
    int index = hash(k_);
    if (indxList[index] == null) {
        indxList[index] = new LinkedList<KplusV<K, V>>();
    } else {
        for (KplusV<K, V> KplusV : indxList[index]) {
            if (KplusV.key.equals(k_)) {
                KplusV.value = v_;
                return;
            }
        }
    }
    indxList[index].add(new KplusV<K,V>(k_, v_));
    size++;
}


public V get(K k_) {
    int index = hash(k_);
    if (indxList[index] != null) {
        for (KplusV<K, V> KplusV : indxList[index]) {
            if (KplusV.key.equals(k_)) {
                return KplusV.value;
            }
```

Реализуем метод put для заполнения строки таблицы новой парой-значением. Сначала метод считает хэщ ключа и потом, если значение под индексом, совпадающим с этим хэшэм пусто, то создаёт там новый линкдлист. В обратном случае, если линкдлист уже есть, мы пробегаемся по его значениям и смотрим, совпадает ли наш полученный ключ уже с каким то из имеющихся, если да, то заменяем значение на наше в элементе, в котором ключ совпадает. Если же ключ не совпадает, то просто добавляем в текущую строку нашу пару ключ-значение. При добавлении нового значения, увеличиваем счётчик элементов в таблице на 1.

В методе get всё проще, если хэш нашего ключа является индексом пустой строки в таблице, то возвращаем 0, если же в строке что-то есть, то пробегаемся по линкдлисту и возвращаем значение соответствующего ключа.

```
        }
    }
    return null;
}

public void remove(K k_){
    int index = hash(k_);
    if (indxList[index] == null){
        return;
    } else{
        for (KplusV<K,V> KplusV : indxList[index]){
            if (KplusV.key.equals(k_)){
                indxList[index].remove(KplusV);
                size--;
            }
        }
    }

}

public int sizeCon() {
    return size;
}

public boolean isEmpty(){
    return size == 0;
}
```

При создании метода remove для удаления значения из таблицы по ключу действуем похожим на предыдущий метод способом, только не возвращаем значение value, а удаляем пару значений KplusV с помощью метода линкдлиста remove() и уменьшаем счётчик элментов таблицы на 1.

Имплиментируем методы получения количества элементов таблицы и проверки условия её пустоты.

```java
public class HashtabMain {
    Run | Debug
    public static void main(String[] args) {
        HashTab<Integer,String> newtab = new HashTab<Integer,String>();
        newtab.put(k_:1,v_:"subj1");
        newtab.put(k_:2,v_:"subj2");
        newtab.put(k_:3,v_:"subj3");
        newtab.remove(k_:2);

        System.out.println(newtab.get(k_:1));
        System.out.println(newtab.get(k_:3));
        System.out.println(newtab.sizeCon());
        System.out.println(newtab.isEmpty());

    }
}
```

В отдельном файле повзаимодействуем с нашим созданным классом HashTab, создав объект класса, заполнив таблицу значениями, проверив возможность получения и удаления записей из таблицы по ключу и работоспособность методов, связанных с количеством записей в таблице.

Вот, что получим в терминале:

```
PS C:\Users\User\Desktop\MTUCI\2_course\java_subject\just_labs\j_labzzz\lab_3> java HashtabMain.java
subj1
subj3
2
false
```

Задание 2. Работа со встроенным классом HashMap по варианту.

Вариант 2: Реализация хэш-таблицы для хранения информации о товарах в интернет-магазине. Ключом является артикул товара, а значением - объект класса Product, содержащий поля наименование, описание, цена и количество на складе. Необходимо реализовать операции вставки, поиска и удаления товара по артикулу.

Для начала создадим класс Product, где укажем все необходимые поля, конструкторы и метод, который будет преобразовывать характеристики нашего товара в строку. Чтобы мы в дальнейшем могли положить эту строку в value пары-значений линкдлиста объекта нашего класса HashMap:

```
public class Product {
    public String name;
    public String description;
    public int price;
    public int stock;

    public Product(String n_, String desc, int p_, int s_){
        name = n_;
        description = desc;
        price = p_;
        stock = s_;
    }

    public Product(){
        this(n_:"name",desc:"description",p_:10,s_:1);

    }

    public String convToStr(){
        String output = "";
        output+=name+"; "+description+"; "+String.valueOf(price)+"; "+String.valueOf(stock);
        return output;
    }


}
```

Импортируем класс ХэшМап, создадим объект — хэш таблицу, и заполним её характеристиками товраров с ключами артикулам, а затем продемонстрируем получение и удаление данных из этой таблицы.

```
import java.util.HashMap;

public class ProductMain {
    Run | Debug
    public static void main(String[] args) {
        HashMap<Integer, Product> merchHT = new HashMap<Integer, Product>();
        merchHT.put(key:11, new Product(n_:"watch", desc:"shows time", p_:150, s_:5));
        merchHT.put(key:14, new Product(n_:"umbrella", desc:"protect from rain", p_:200, s_:7));
        merchHT.put(key:20, new Product(n_:"cap", desc:"just fancy thing", p_:300, s_:3));
        merchHT.remove(key:11);

        System.out.println(merchHT.get(key:14).convToStr());
        System.out.println(merchHT.get(key:20).convToStr());
    }

}
```

Вот что получим, если запустим программу в терминале:

```
PS C:\Users\User\Desktop\MTUCI\2_course\java_subject\just_labs\j_labzzz\lab_3> java ProductMain.java
umbrella; protect from rain; 200; 7
cap; just fancy thing; 300; 3
```

Вывод:

Мы разобрались со структурой хэш таблиц, логикой их работы и способах применения, реализовав свой собственный класс, описывающий хэщ-таблицы. А также мы поработали со встроенным классом Java, используемым для создания хэш-таблиц — HashMap и поняли, что он является довольно быстрым и удобным инструментом в этой сфере.