**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

Георгян Сергей

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:**

Реализация хэш-таблицы для хранения информации о сотрудниках в компании. Ключом будет идентификационный номер сотрудника, а значением - объект класса Employee, содержащий данные о имени, должности и заработной плате. Необходимо реализовать операции вставки, поиска и удаления сотрудника по ID.

**Ход работы**

**Задание 1:**

import java.util.LinkedList;  
  
public class HashTable<K, V> {  
 private static final int *TABLE\_SIZE* = 16;  
 private LinkedList<Entry<K, V>>[] table;  
  
 public HashTable() {  
 table = new LinkedList[*TABLE\_SIZE*];  
 }  
  
  
 private static class Entry<K, V> {  
 K key;  
 V value;  
  
 Entry(K key, V value) {  
 this.key = key;  
 this.value = value;  
 }  
 }

Файл 1 – создание хэш таблицы и конструктора Entry

public void put(K key, V value) {  
 int index = key.hashCode() % *TABLE\_SIZE*;  
  
 if (table[index] == null) {  
 table[index] = new LinkedList<>();  
 }  
  
  
 for (Entry<K, V> entry : table[index]) {  
 if (entry.key.equals(key)) {  
 entry.value = value;  
 return;  
 }  
 }  
  
  
 table[index].add(new Entry<>(key, value));  
}

Файл 2 – метод put для ввода новых значений в таблицу

public V get(K key) {  
 int index = key.hashCode() % *TABLE\_SIZE*;  
  
 if (table[index] != null) {  
 for (Entry<K, V> entry : table[index]) {  
 if (entry.key.equals(key)) {  
 return entry.value;  
 }  
 }  
 }  
  
 return (V) ("Работник с ключом " + key + " не найден");  
}

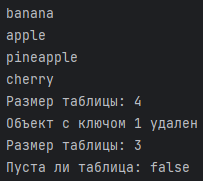
Файл 3 – метод get для получения значения из таблицы по ключу

public void remove(K key) {  
 int index = key.hashCode() % *TABLE\_SIZE*;  
  
 if (table[index] != null) {  
 for (Entry<K, V> entry : table[index]) {  
 if (entry.key.equals(key)) {  
 table[index].remove(entry);  
 System.*out*.println("Объект с ключом " + key + " удален");  
 return;  
 }  
 }  
 }  
}  
  
public int size() {  
 int size = 0;  
 for (LinkedList<Entry<K, V>> bucket : table) {  
 if (bucket != null) {  
 size += bucket.size();  
 }  
 }  
 return size;  
}

Файл 4 – методы remove и size для удаления элемента и получения размера

public boolean isEmpty() {  
 for (LinkedList<Entry<K, V>> bucket : table) {  
 if (bucket != null && !bucket.isEmpty()) {  
 return false;  
 }  
 }  
 return true;  
 }  
  
 public static void main(String[] args) {  
 HashTable<Integer, String > myTable = new HashTable<>();  
 myTable.put(1, "banana");  
 myTable.put(2, "apple");  
 myTable.put(3, "pineapple");  
 myTable.put(4, "cherry");  
  
 System.*out*.println(myTable.get(1));  
 System.*out*.println(myTable.get(2));  
 System.*out*.println(myTable.get(3));  
 System.*out*.println(myTable.get(4));  
 System.*out*.println("Размер таблицы: " + myTable.size());  
 myTable.remove(1);  
 System.*out*.println("Размер таблицы: " + myTable.size());  
 System.*out*.println("Пуста ли таблица: " + myTable.isEmpty());  
 }  
}

Файл.5 – метод isEmpty, создание таблицы и ее вывод

 Рис.1 – вывод методов таблицы для задания 1

Задание 2:

import java.util.LinkedList;  
  
public class EmployeeHashTable<K, V> {  
 private static final int *TABLE\_SIZE* = 16;  
 private LinkedList<Entry<K, V>>[] table;  
  
 public EmployeeHashTable() {  
 table = new LinkedList[*TABLE\_SIZE*];  
 }  
  
  
 private static class Entry<K, V> {  
 K key;  
 Employee employee;  
  
 Entry(K key, V employee) {  
 this.key = key;  
 this.employee = (Employee) employee;  
 }  
 }

Файл 6 – изменение конструктора Entry

public void put(K key, V employee) {  
 int index = key.hashCode() % *TABLE\_SIZE*;  
  
 if (table[index] == null) {  
 table[index] = new LinkedList<>();  
 }  
  
  
 for (Entry<K, V> entry : table[index]) {  
 if (entry.key.equals(key)) {  
 entry.employee = (Employee) employee;  
 return;  
 }  
 }  
  
 table[index].add(new Entry<>(key, employee));  
}

Файл 7 – изменение метода put

public String get(K key) {  
 int index = key.hashCode() % *TABLE\_SIZE*;  
  
 if (table[index] != null) {  
 for (Entry<K, V> entry : table[index]) {  
 if (entry.key.equals(key)) {  
 String new\_str = "Имя работника: " + entry.employee.getName() +  
 " | Должность работника: " + entry.employee.getPosition() +  
 " | Зароботная плата: " + entry.employee.getSalary() + "р.";  
 return new\_str;  
 }  
 }  
 }  
  
 return "Работник с ключом " + key + " не найден"; // Ключ не найден  
}

Файл 8 – изменение метода get

public void remove(K key) {  
 int index = key.hashCode() % *TABLE\_SIZE*;  
  
 if (table[index] != null) {  
 for (Entry<K, V> entry : table[index]) {  
 if (entry.key.equals(key)) {  
 table[index].remove(entry);  
 System.*out*.println("Объект с ключом " + key + " удален");  
 return;  
 }  
 }  
 }  
}  
  
public int size() {  
 int size = 0;  
 for (LinkedList<Entry<K, V>> bucket : table) {  
 if (bucket != null) {  
 size += bucket.size();  
 }  
 }  
 return size;  
}  
  
  
public boolean isEmpty() {  
 for (LinkedList<Entry<K, V>> bucket : table) {  
 if (bucket != null && !bucket.isEmpty()) {  
 return false;  
 }  
 }  
 return true;  
}

Файл 9 – изменение методов remove, size, isEmpty

public static class Employee {  
 private String name;  
 private String position;  
 private int salary;  
  
  
 public Employee(String name, String position, int salary) {  
 this.name = name;  
 this.position = position;  
 this.salary = salary;  
 }  
 public String getName() {  
 return name;  
 }  
  
 public String getPosition() {  
 return position;  
 }  
  
 public int getSalary() {  
 return salary;  
 }  
 }  
  
  
 public static void main(String[] args) {  
 EmployeeHashTable<Integer, Employee> myTable = new EmployeeHashTable<>();  
 myTable.put(1, new Employee("Егор Васильевич", "офисный менеджер", 65000));  
 myTable.put(2, new Employee("Олег Витальевич", "начальник отдела продаж", 93000));  
 myTable.put(3, new Employee("Сергей Викторович", "директор компании", 1000000));  
 myTable.put(4, new Employee("Артем Владимирович", "руководитель отдела разработки", 100000));  
 myTable.put(5, new Employee("Евгения Владиславовна", "главный секретарь", 60000));  
  
 System.*out*.println(myTable.get(1));  
 System.*out*.println(myTable.get(2));  
 System.*out*.println(myTable.get(3));  
 System.*out*.println(myTable.get(4));  
 System.*out*.println(myTable.get(5));  
 System.*out*.println(myTable.get(6));  
 System.*out*.println("Размер таблицы: " + myTable.size());  
 myTable.remove(1);  
 System.*out*.println("Размер таблицы: " + myTable.size());  
 System.*out*.println("Пуста ли таблица: " + myTable.isEmpty());  
  
 }  
}

Файл 10 – создание конструктора Employee и создание таблицы

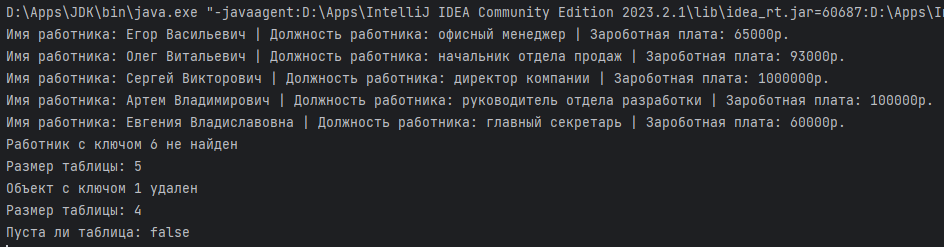


Рис.2 – вывод программы для задания 2

**Выводы**

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

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