**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»**

Кафедра Математическая кибернетика и информационные технологии

Отчет по Лабораторной работе №3

Класс Object. Работа с хэш-таблицами

По дисциплине: Информационные технологии и программирование

Выполнил: студент группы БВТ-2401

Язалиев Тимур Исламович

Руководитель: Харрасов Камиль Раисович

Москва, 2025

Вариант: 1

Цель работы:

Освоить принципы работы с хэш-таблицами в Java, научиться создавать собственные структуры данных и использовать встроенные коллекции HashMap.

Задание 1. Реализация обобщённой хэш-таблицы HashTable<K, V>

Цель:

Понять внутреннее устройство хэш-таблиц и реализовать собственную структуру данных с основными операциями:

добавление (put),

поиск (get),

удаление (remove),

получение размера (size),

проверка пустоты (isEmpty).

Содержание:

package Lab\_3;  
  
import java.util.LinkedList;  
import java.util.Objects;  
import java.util.Iterator;  
  
public class HashTable<K, V> {  
 private static final int *DEFAULT\_CAPACITY* = 16;  
 private LinkedList<Entry<K,V>>[] table;  
 private int size = 0;  
  
 @SuppressWarnings("unchecked")  
 public HashTable(int capacity) {  
 int cap = Math.*max*(1, capacity);  
 table = (LinkedList<Entry<K,V>>[]) new LinkedList[cap];  
 }  
  
 public HashTable() {  
 this(*DEFAULT\_CAPACITY*);  
 }  
  
 private int indexFor(Object key) {  
 int h = (key == null) ? 0 : Objects.*hashCode*(key);  
 // ensure non-negative  
 return Math.*abs*(h) % table.length;  
 }  
  
 public void put(K key, V value) {  
 int idx = indexFor(key);  
 if (table[idx] == null) table[idx] = new LinkedList<>();  
 for (Entry<K,V> e : table[idx]) {  
 if (Objects.*equals*(e.key, key)) {  
 e.value = value; // replace  
 return;  
 }  
 }  
 table[idx].add(new Entry<>(key, value));  
 size++;  
 }  
  
 public V get(K key) {  
 int idx = indexFor(key);  
 if (table[idx] == null) return null;  
 for (Entry<K,V> e : table[idx]) {  
 if (Objects.*equals*(e.key, key)) return e.value;  
 }  
 return null;  
 }  
  
 public boolean remove(K key) {  
 int idx = indexFor(key);  
 if (table[idx] == null) return false;  
 Iterator<Entry<K,V>> it = table[idx].iterator();  
 while (it.hasNext()) {  
 Entry<K,V> e = it.next();  
 if (Objects.*equals*(e.key, key)) {  
 it.remove();  
 size--;  
 return true;  
 }  
 }  
 return false;  
 }  
  
 public int size() {  
 return size;  
 }  
  
 public boolean isEmpty() {  
 return size == 0;  
 }  
  
 // Простой внутренний класс записи  
 private static class Entry<K,V> {  
 K key;  
 V value;  
 Entry(K k, V v) { key = k; value = v; }  
 }  
}

HashTable

Созданная структура HashTable демонстрирует принцип работы встроенного класса HashMap.

Она позволяет понять, как данные распределяются по хэшам и как решаются коллизии.

Задание 2. Использование класса HashMap (вариант 1)

Цель:

Использовать встроенный класс HashMap для хранения и поиска пользовательских объектов.

Содержание:

package Lab\_3;  
  
import java.util.HashMap;  
import java.util.Map;  
  
public class Lab3HashMapDemo {  
 public static void main(String[] args) {  
 // Используем номер зачетной книжки (String) в качестве ключа  
 Map<String, Student> students = new HashMap<>();  
  
 // Вставка студентов (put)  
 students.put("2023-001", new Student("Иван", "Иванов", 20, 4.50));  
 students.put("2023-002", new Student("Мария", "Петрова", 21, 4.75));  
 students.put("2023-003", new Student("Алексей", "Сидоров", 19, 3.90));  
  
 System.*out*.println("=== Все студенты (после вставки) ===");  
 students.forEach((k,v) -> System.*out*.println(k + " -> " + v));  
  
 // Поиск студента по номеру зачетки (get)  
 String searchKey = "2023-002";  
 Student found = students.get(searchKey);  
 System.*out*.println("\nПоиск по номеру " + searchKey + ":");  
 System.*out*.println(found != null ? found : "Студент не найден.");  
  
 // Удаление студента (remove)  
 String removeKey = "2023-001";  
 Student removed = students.remove(removeKey);  
 System.*out*.println("\nУдаление номера " + removeKey + ": " + (removed != null ? "удален: " + removed : "не найден"));  
  
 System.*out*.println("\n=== Результат после удаления ===");  
 students.forEach((k,v) -> System.*out*.println(k + " -> " + v));  
  
 // Демонстрация использования собственной HashTable (optional)  
 System.*out*.println("\n=== Демонстрация собственной HashTable ===");  
 HashTable<String, Student> myTable = new HashTable<>();  
 myTable.put("2023-002", new Student("Мария", "Петрова", 21, 4.75));  
 myTable.put("2023-003", new Student("Алексей", "Сидоров", 19, 3.90));  
  
 System.*out*.println("myTable.get(2023-002) -> " + myTable.get("2023-002"));  
 System.*out*.println("myTable.size() -> " + myTable.size());  
 myTable.remove("2023-003");  
 System.*out*.println("after remove, myTable.size() -> " + myTable.size());  
 }  
}

Lab3HashMapDemo

package Lab\_3;  
  
import java.util.Objects;  
  
public class Student {  
 private String firstName;  
 private String lastName;  
 private int age;  
 private double gpa; // средний балл  
  
 public Student(String firstName, String lastName, int age, double gpa) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.age = age;  
 this.gpa = gpa;  
 }  
  
 // геттеры/сеттеры (по необходимости можно добавить)  
 public String getFirstName() { return firstName; }  
 public String getLastName() { return lastName; }  
 public int getAge() { return age; }  
 public double getGpa() { return gpa; }  
  
 @Override  
 public String toString() {  
 return String.*format*("%s %s, age=%d, GPA=%.2f",  
 firstName, lastName, age, gpa);  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (!(o instanceof Student)) return false;  
 Student s = (Student) o;  
 return age == s.age  
 && Double.*compare*(s.gpa, gpa) == 0  
 && Objects.*equals*(firstName, s.firstName)  
 && Objects.*equals*(lastName, s.lastName);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(firstName, lastName, age, gpa);  
 }  
}

Student

В отличие от нашей собственной HashTable, встроенная HashMap автоматически управляет хэшированием и расширением таблицы, а также эффективно решает коллизии.

**Сравнение HashTable и HashMap**

Критерий HashTable (своя реализация) HashMap (встроенная)

Реализация Ручная, учебная Встроенная в Java

Коллизии Метод цепочек Цепочки + балансировка

Производительность Средняя Высокая

Потокобезопасность Нет Нет (для этого есть ConcurrentHashMap)

**Вывод**

В ходе лабораторной работы:

Изучены принципы хэширования и структура хэш-таблиц;

Реализована собственная версия HashTable;

Освоено использование встроенного класса HashMap;

Проведено сравнение эффективности и принципов работы обеих реализаций.

Результат: получен практический опыт работы с ассоциативными контейнерами и хэш-функциями в языке Java.

*Ссылка на GitHub: https://github.com/Maggistr/LABWORK\_2\_sem.git*