**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.

Вариант 3.

Выполнил: студент группы БПИ2401

Костиль Антон Валерьевич

Проверил: Харрасов Камиль Раисович

Москва, 2025

Цель работы: **изучение и практическая реализация хэш-таблиц в языке Java, закрепление навыков объектно-ориентированного программирования.**

Ход работы:

**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.

**Шаг 1**.

Задаем класс HashTable. В нем описываем внутренний обобщенный класс Entry, объект которого будет хранить одну пару ключ-значение. В нем реализуем методы put(), get(), remove()

public class HashTable<K, V> {

    private static class Entry<K,V> {

        private K key;

        private V value;

        public Entry(K key, V value) {

            this.key = key;

            this.value = value;

        }

        public K getKey() {

            return key;

        }

        public V getValue() {

            return value;

        }

        public void setValue(V value) {

            this.value = value;

        }

    }

**Шаг 2.**

Заполняем поля HashTable (массив связанных списков и количество элементов в таблице) и прописываем конструктор

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

    private int size;

    public HashTable(int capacity) {

        table = new LinkedList[capacity];

        size = 0;

    }

**Шаг 3.**

Прописываем хэш-функцию и методы добавления пар, получения значения по ключу, удаления пары, методы size() и isEmpty()

private int hash(K key) {

        return Math.abs(key.hashCode() % table.length);

    }

    public void put(K key, V value) {

        int index = hash(key);

        if (table[index] == null) {

            table[index] = new LinkedList<>();

        }

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

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

                entry.setValue(value);

                return;

            }

        }

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

        size++;

    }

    public V get(K key) {

        int index = hash(key);

        if (table[index] == null) {

            return 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) {

            return;

        }

        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;

    }

}

**Задание 2. Работа с встроенным классом HashMap.**

Реализация хэш-таблицы для хранения информации о заказах в интернет-магазине. Ключом является номер заказа, а значением — объект класса Order, содержащий поля дата заказа, список товаров и статус заказа. Необходимо реализовать операции вставки, поиска и удаления заказа по номеру. Также необходимо реализовать метод для изменения статуса заказа.

**Шаг 1.**

Описываем класс Order, содержащий поля дата заказа, список товаров и статус заказа. Реализуем геттеры и сеттеры

package lab3;

import java.util.ArrayList;

import java.util.List;

public class Order {

    private String date;

    private List<String> items;

    private String status;

    public Order(String date, List<String> items, String status) {

        this.date = date;

        this.items = new ArrayList<>(items);

        this.status = status;

    }

    public String getDate() {

        return date;

    }

    public List<String> getItems() {

        return items;

    }

    public String getStatus() {

        return status;

    }

    public void setStatus(String status) {

        this.status = status;

    }

    @Override

    public String toString() {

        return "Order{" +

                "date='" + date + '\'' +

                ", items=" + items +

                ", status='" + status + '\'' +

                '}';

    }

}

**Шаг 2.**

Прописываем класс OrderManager, который будет являться нашей хэш-таблицей. Реализуем операции вставки, удаления заказа, получения значения по ключу, обновления статуса заказа.

package lab3;

import java.util.HashMap;

public class OrderManager {

    private HashMap<Integer, Order> orders;

    public OrderManager() {

        orders = new HashMap<>();

    }

    public void addOrder(int orderNumber, Order order) {

        orders.put(orderNumber, order);

    }

    public Order getOrder(int orderNumber) {

        return orders.get(orderNumber);

    }

    public void removeOrder(int orderNumber) {

        orders.remove(orderNumber);

    }

    public void updateStatus(int orderNumber, String newStatus) {

        Order order = orders.get(orderNumber);

        if (order != null) {

            order.setStatus(newStatus);

        } else {

            System.out.println("Order " + orderNumber + " is not found");

        }

    }

}

**Шаг 3.**

Пишем Main.java, в котором проверяем работу нашей хэш-таблицы. Вывод программы представлен на рисунке 1.

package lab3;

import java.util.List;

public class Main {

    public static void main(String[] args) {

        OrderManager manager = new OrderManager();

        Order order1 = new Order("19.10.2025", List.of("Laptop", "Mouse"), "In progress");

        Order order2 = new Order("19.10.2025", List.of("Iphone"), "In delivery");

        manager.addOrder(1001, order1);

        manager.addOrder(1002, order2);

        System.out.println(manager.getOrder(1001));

        manager.updateStatus(1001, "Canceled");

        manager.removeOrder(1002);

        System.out.println(manager.getOrder(1001));

        System.out.println(manager.getOrder(1002));

    }

}

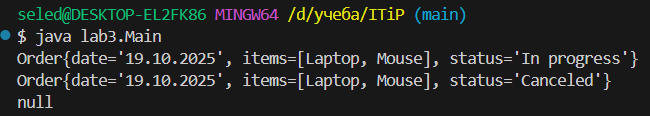


Рисунок 1 - вывод программы

**Ответы на контрольные вопросы:**

**1. Для чего нужен класс Object?**

Object — корневой класс всех классов в Java. Любой класс в Java неявно наследуется от Object, если не указано другое. Содержит базовые методы, доступные для всех объектов. позволяет работать с любыми объектами через общие методы.

**2. Для чего нужно переопределять методы equals() и hashCode()?**

equals() проверяет логическое равенство объектов. По умолчанию проверяет только ссылки на объекты, поэтому нужно переопределение.

hashCode() возвращает хеш-код объекта. Переопределение нужно, чтобы в хэш-таблицах логически равные элементы имели одинаковый хеш-код.

**3. Какие есть правила переопределения методов equals() и hashCode()?**

• два объекта, которые считаются равными методом equals(), должны иметь одинаковое значение хеш-кода

• коллекции, основанные на хэш-таблицах (например, HashMap, HashSet), используют хеш-коды для эффективного хранения и поиска данных. Несогласованность между методами equals() и hashCode() приведет к неправильной работе этих структур данных.

**4. Что делает метод toString()? Почему его часто переопределяют?**

toString() возвращает строковое представление объекта. По умолчанию выводит: ИмяКласса@хэшкод (например, Order@1a2b3c). Переопределяют, чтобы получить читаемое и информативное представление,

**5. Что делает метод finalize()? Почему его использование считается устаревшим (deprecated)?**

finalize() — вызывается перед сборкой мусора, когда объект уничтожается. Позволяет выполнять очистку ресурсов. Устарел, потому что использование может привести к утечкам памяти или непредсказуемому поведению.

**6. Что такое коллизия?**

Коллизия — ситуация, когда два разных ключа имеют одинаковый хэш-код и попадают в один и тот же индекс массива хэш-таблицы.

**7. Какие есть способы разрешения коллизий?**

Метод цепочек - каждый индекс массива хранит список всех элементов с этим хэш-кодом.

Открытая адресация - ищем свободный индекс по определённой стратегии: линейное или квадратичное пробирование.

Двойное хеширование - используем второй хэш для поиска нового индекса.

**8. Как хранятся данные в хэш-таблице?**

Данные хранятся в массиве бакетов (индексов). В каждом бакете может быть список элементов (если используется метод цепочек). Каждый элемент — это пара ключ-значение

**9. Что происходит, если в хэш-таблицу добавить элемент с одинаковым значением ключа?**

Старое значение заменяется новым. Количество элементов не увеличивается.

**10. Что происходит, если в хэш-таблицу добавить элемент с таким же хэш-кодом ключа, но разными исходными значениями?**

Происходит коллизия:

* При цепочках элемент добавляется в список по индексу.
* При открытой адресации ищется свободный индекс.

**11. Как изменяется HashMap при достижении порогового значения?**

HashMap имеет load factor (коэффициент заполнения по умолчанию 0.75). Когда количество элементов превышает вместимость(capacity) \* load factor, происходит расширение. Удваивается размер массива. Все элементы рехешируются в новый массив.

**Вывод:** в ходе выполнения работы были освоены основные принципы работы с хэш-таблицами в Java. Были разработаны и реализованы классы HashTable, Order и OrderManager, продемонстрированы операции добавления, поиска, удаления и изменения статуса элементов. Работа позволила закрепить навыки работы с коллекциями, генерализацией типов (generics), а также использование методов toString(), инкапсуляцию полей и взаимодействие объектов. Работа размещена на GitHub - <https://github.com/Antosha044/ITiP>