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

Лабораторная работа №3

Класс Object. Работа с хэш-таблицами

по дисциплине
«Информационные технологии и программирование»

Выполнил: студент гр. БВТ2403
Толибов Р.Ш.

Руководитель:

Москва, 2025 г.

Цель работы:
Изучить особенности базового класса Object в Java и его ключевые методы, а также рассмотреть принципы работы хэш-таблиц и их реализацию в виде классов HashMap и Hashtable.

Ход работы:
Класс Object является базовым классом всей иерархии классов в Java**.** Все остальные классы, включая пользовательские, неявно наследуют Object. Это означает, что любой объект в Java имеет методы, определённые в этом классе. Среди них:
toString() — возвращает строковое представление объекта;
equals(Object obj) — проверяет равенство объектов;
hashCode() — возвращает целое значение хэш-кода объекта;
getClass() — возвращает метаданные класса;
clone() — создаёт копию объекта;
wait(), notify(), notifyAll() — используются при синхронизации потоков.
Особое значение имеют методы **equals()** и **hashCode()**, поскольку они определяют, как объекты сравниваются и хранятся в коллекциях, основанных на хэш-таблицах, таких как HashMap и Hashtable.
По умолчанию метод equals() сравнивает **ссылки на объекты**, а не их содержимое. Чтобы сравнение было логическим (по данным), этот метод необходимо переопределить. При этом важно также переопределить hashCode(), чтобы два равных объекта имели одинаковые хэш-коды. Нарушение этого контракта может привести к некорректной работе хэш-коллекций — например, объект не будет найден в HashMap, даже если он логически равен другому элементу.

В рамках выполнения лабораторной работы была реализована система для демонстрации работы с хэш-таблицами. Первым шагом стал создание класса Entry, который представляет собой узел хэш-таблицы. Этот класс содержит пару "ключ-значение" и ссылку на следующий узел, что позволяет организовать цепочки для разрешения коллизий методом цепочек. Класс включает конструктор для инициализации полей и переопределение метода toString() для удобного вывода содержимого.

```
      import java.util.*;


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;
        }
    }
```

Далее был разработан основной класс HashTable, реализующий функционал хэш-таблицы. Класс содержит массив объектов Entry и предоставляет основные операции: добавление элемента (put), получение значения по ключу (get), удаление элемента (remove) и вывод всей таблицы (print). Особенностью реализации является использование метода цепочек для обработки коллизий - при совпадении хэш-кодов новые элементы добавляются в начало цепочки.

```
private LinkedList<Entry<K, V>>[] table;
    private int size;
    private static final int capacity = 10;

    public HashTable() {
        table = new LinkedList[capacity];
        size = 0;
    }

    private int hash(K key) {
        return Math.abs(key.hashCode() % capacity);
    }

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;
    }
    public void printTable() {
        for (int i = 0; i < table.length; i++) {
            System.out.print("[" + i + "]: ");
            if (table[i] != null) {
                for (Entry<K, V> entry : table[i]) {
```

```
                    System.out.print("(" + entry.getKey() + " ; " +
entry.getValue() + ") ");
                }
            }
            System.out.println();
        }
    }
    public static void main(String[] args) {
        HashTable<String, Integer> map = new HashTable<>();
        map.put("one", 1);
        map.put("two", 2);
        map.put("three", 3);
        map.put("two", 22);

        System.out.println("Значение по ключу 'two': " + map.get("two"));
        System.out.println("Размер таблицы: " + map.size());
        System.out.println("Таблица пуста? " + map.isEmpty());
        map.printTable();
    }
}
```

Вывод в терминале:

```
Размер таблицы: 3

Таблица пуста? false

[0]:

[1]:

[2]: (one ; 1)

[3]:

[4]:

[5]:

[6]: (two ; 22) (three ; 3)

[7]:

[8]:

[9]:
```

Для демонстрации работы хэш-таблицы был создан класс Order, содержащий информацию о заказах: номер, дата, товар, статус заказа

```
import java.util.*;

class Order {
    private String date;
    private String items;
    private String status;


    public Order(String date, String items, String status) {
        this.date = date;
        this.items = items;
        this.status = status;
    }

    public String getDate() {
        return date;
    }

    public String getItems() {
        return items;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    @Override
    public String toString() {
        return "Дата: " + date +
               ", Товары: " + items +
               ", Статус: " + status;
    }
}
```

В демонстрационной программе была создана хэш-таблица, где ключами выступали номера заказов, а значениями - объекты класса Order. Были последовательно продемонстрированы все операции: добавление нескольких заказов, поиск по ключу, удаление элемента и вывод содержимого таблицы.

```
public class Main {
    public static void main(String[] args) {

        HashMap<Integer, Order> orders = new HashMap<>();

        orders.put(1001, new Order("2025-09-23", "Ноутбук", "В обработке"));
        orders.put(1002, new Order("2025-09-22", "Наушники", "В обработке"));
        orders.put(1003, new Order("2025-09-21", "Монитор", "Доставлен"));

        System.out.println("\nВсе заказы в системе:");
            for (Map.Entry<Integer, Order> entry : orders.entrySet()) {
                System.out.println("Номер: " + entry.getKey() + " ; " +
entry.getValue());
        }
         System.out.println();

        int searchId = 1002;
        if (orders.containsKey(searchId)) {
            System.out.println("Найден заказ №" + searchId + ": " +
orders.get(searchId));
        } else {
            System.out.println("Заказ №" + searchId + " не найден.");
        }

        int updateId = 1001;
        if (orders.containsKey(updateId)) {
            orders.get(updateId).setStatus("Доставлен");
            System.out.println("Статус заказа №" + updateId + " обновлён: " +
orders.get(updateId));
        }

        int deleteId = 1003;
        if (orders.containsKey(deleteId)) {
            orders.remove(deleteId);
            System.out.println("Заказ №" + deleteId + " удалён.");
        }

        System.out.println("\nВсе заказы в системе:");
        for (Map.Entry<Integer, Order> entry : orders.entrySet()) {
            System.out.println("Номер: " + entry.getKey() + " ; " +
entry.getValue());
        }
    }
}
```

Отдельно проверялась обработка коллизий путем добавления элементов с разными ключами, но дающими одинаковый хэш-код. Все операции выполнялись корректно, что подтверждает правильность реализации

структуры данных и соблюдение контракта между методами equals() и hashCode().

Вывод в терминале:

```
Все заказы в системе:

Номер: 1001 ; Дата: 2025-09-23, Товары: Ноутбук, Статус: В обработке

Номер: 1002 ; Дата: 2025-09-22, Товары: Наушники, Статус: В обработке

Номер: 1003 ; Дата: 2025-09-21, Товары: Монитор, Статус: Доставлен



Найден заказ №1002: Дата: 2025-09-22, Товары: Наушники, Статус: В обработке

Статус заказа №1001 обновлён: Дата: 2025-09-23, Товары: Ноутбук, Статус:
Доставлен

Заказ №1003 удалён.



Все заказы в системе:

Номер: 1001 ; Дата: 2025-09-23, Товары: Ноутбук, Статус: Доставлен

Номер: 1002 ; Дата: 2025-09-22, Товары: Наушники, Статус: В обработке
```

Вывод:

В ходе работы я освоил принципы работы хэш-таблиц и научился корректно переопределять методы equals() и hashCode(), соблюдая их контракт для обеспечения правильной работы с хэш-коллекциями.

https://github.com/TRuslan666/ITiP/tree/main/лаб3