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»

Отчет по лабораторной работе №3

по дисциплине «Информационные технологии и программирование»

Выполнил: студент группы БПИ2401

Юлдашев Всеволод

Проверил: Харрасов Камиль Раисович

Москва

2025

1. **Цель работы:** изучить принципы работы хеш-таблиц и механизм хеширования данных в языке Java.

   В ходе работы необходимо освоить:

   a. структуру данных Hash Table и её реализацию на основе массива списков (метод цепочек);
   b. понятия ключа, значения, хеш-функции и коллизии;
   c. методы добавления, поиска и удаления элементов в хеш-таблице;
   d. использование дженериков (обобщений) для создания универсальных структур данных;
   e. работу с классом Object и методами hashCode() и equals() при организации сравнения объектов.

2. **Ход работы:**

```java
import java.util.LinkedList;

public class HashTable<K, V> {
    private LinkedList<Entry<K, V>>[] table;
    private int capacity = 10;
    private int size = 0;

    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; }
```

```
    }
    @SuppressWarnings("unchecked")
    public HashTable() {
        table = new LinkedList[capacity];
    }

    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) {
```

```
            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) {
            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 static void main(String[] args) {
        HashTable<String, Integer> table = new HashTable<>();
        table.put("MTUCI", 10);
        table.put("MEI", 20);
        table.put("MTU", 30);

        System.out.println("MTUCI -> " + table.get("MTUCI"));
        table.remove("MEI");
        System.out.println("Size = " + table.size());
        System.out.println("Is empty? " + table.isEmpty());
    }
}
```

```
> java HashTable
MTUCI -> 10
Size = 2
Is empty? false
```

```
import java.util.HashMap;

class Car {
    private String brand;
    private String model;
    private int year;

    public Car(String brand, String model, int year) {
        this.brand = brand;
        this.model = model;
        this.year = year;
    }

    @Override
```

```
    public String toString() {
        return brand + " " + model + " (" + year + ")";
    }
}

public class CarRegistry {
    private HashMap<String, Car> cars = new HashMap<>();

    public void addCar(String plate, Car car) {
        cars.put(plate, car);
    }

    public Car findCar(String plate) {
        return cars.get(plate);
    }

    public void removeCar(String plate) {
        cars.remove(plate);
    }

    public void printAll() {
        for (String plate : cars.keySet()) {
            System.out.println(plate + " -> " + cars.get(plate));
        }
    }

    public static void main(String[] args) {
        CarRegistry registry = new CarRegistry();
        registry.addCar("A777MP", new Car("Dodge", "Challenger",
2020));
        registry.addCar("B888OP", new Car("BMW", "X5", 2022));
```

```
        System.out.println("Поиск: " +
registry.findCar("A777MP"));
        registry.printAll();


        registry.removeCar("B888OP");
        System.out.println("После удаления:");
        registry.printAll();
    }
}
```

```
> java CarRegistry
Поиск: Dodge Challenger (2020)
B888OP -> BMW X5 (2022)
A777MP -> Dodge Challenger (2020)
После удаления:
A777MP -> Dodge Challenger (2020)
```

Контрольные вопросы:

1. Для чего нужен класс Object?
Класс Object — это базовый класс всех классов в Java. Он определяет общие методы (equals(), hashCode(), toString(), clone(), и т.д.), доступные каждому объекту.

2. Для чего нужно переопределять методы equals() и hashCode()?
Чтобы корректно сравнивать объекты по содержимому, а не по ссылке, и чтобы объекты правильно работали в коллекциях типа HashMap, HashSet.

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

- Если переопределён equals(), нужно переопределить и hashCode().
- Равные объекты должны иметь одинаковый хэш-код.
- hashCode() должен возвращать одинаковое значение при неизменных данных объекта.

4. Что делает метод toString()? Почему его часто переопределяют?
Возвращает строковое представление объекта.
Его переопределяют, чтобы удобно выводить информацию об объекте (например, для отладки).

5. Что делает метод finalize()? Почему его использование считается устаревшим?
finalize() вызывался перед удалением объекта сборщиком мусора для освобождения ресурсов.
Он устарел, потому что работает ненадёжно и непредсказуемо — вместо него используют try-with-resources или AutoCloseable.

6. Что такое коллизия?
Коллизия — это ситуация, когда разные ключи имеют одинаковый хэш-код.

7. Какие есть способы разрешения коллизий?

- Метод цепочек (chaining) — хранение элементов в списке в одной ячейке.
- Открытая адресация — поиск другой свободной ячейки по определённому правилу.

8. Как хранятся данные в хэш-таблице?
Данные хранятся в массиве (бакетах), где каждый элемент массива содержит список (цепочку) пар «ключ–значение» с одинаковыми хэшами.

9. Что происходит, если в хэш-таблицу добавить элемент с одинаковым значением ключа?
Старое значение заменяется новым.

10. Что происходит, если в хэш-таблицу добавить элемент с таким же хэш-кодом, но разными исходными значениями?
Возникает коллизия — элемент добавляется в ту же ячейку (цепочку).

11. Как изменяется HashMap при достижении порогового значения?
Происходит расширение (rehashing) — создаётся новый массив большего размера, и все элементы перераспределяются по новым индексам.

Вывод: В ходе работы реализована хеш-таблица с использованием метода цепочек и изучены принципы хеширования в Java. Реализованы основные операции добавления, поиска и удаления элементов, а также рассмотрено применение класса HashMap. Цель работы достигнута.

https://github.com/zevy3/java-labs/tree/main/java-3