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.

Выполнил: студент группы ___БВТ2402__
Скопа Михаил Вячеславович

Москва, 2025

Цель работы: познакомить с основами работы с хэш-таблицами.

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.

```
import java.util.LinkedList;

public class HashTable<K, V> {
    private static class Entry<K, V> {
        K key;
        V value;

        Entry(K key, V value) {
            this.key = key;
            this.value = value;
        }
    }

    private LinkedList<Entry<K, V>>[] table;
    private int size;
    private int capacity;
    private final double loadFactor = 0.75;

    @SuppressWarnings("unchecked")
    public HashTable() {
        this.capacity = 16;
        this.table = new LinkedList[capacity];
        this.size = 0;
        initializeTable();
    }
```

Класс HashTable реализует структуру данных хэш-таблицы с использованием дженериков для типов ключей и значений. Он предоставляет основные операции для работы с ассоциативным массивом. Внутренний статический класс Entry хранит пары ключ-значение. Каждый объект Entry

содержит ключ типа K и значение типа V. Создаю конструктор по умолчанию.

```java
@SuppressWarnings("unchecked")
public HashTable(int initialCapacity) {
    this.capacity = initialCapacity;
    this.table = new LinkedList[capacity];
    this.size = 0;
    initializeTable();
}

private void initializeTable() {
    for (int i = 0; i < capacity; i++) {
        table[i] = new LinkedList<>();
    }
}

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

Создал конструктор с параметрами, создал метод инициализации хэш-таблицы, создал метод хэширования ключа.

```java
public void put(K key, V value) {
    if ((double) size / capacity >= loadFactor) {
        rehash();
    }

    int index = hash(key);
    LinkedList<Entry<K, V>> bucket = table[index];

    for (Entry<K, V> entry : bucket) {
        if (entry.key.equals(key)) {
            entry.value = value;
            return;
        }
    }

    bucket.add(new Entry<>(key, value));
    size++;
}
```

Создал метод добавления в таблицу пары ключ-значение. Если пара с таким ключом уже существует, то меняю значение value.

```java
public V get(K key) {
    int index = hash(key);
    LinkedList<Entry<K, V>> bucket = table[index];

    for (Entry<K, V> entry : bucket) {
        if (entry.key.equals(key)) {
            return entry.value;
        }
    }

    return null;
}
```

Метод возвращения значения Value.

```java
public V remove(K key) {
    int index = hash(key);
    LinkedList<Entry<K, V>> bucket = table[index];

    for (Entry<K, V> entry : bucket) {
        if (entry.key.equals(key)) {
            V value = entry.value;
            bucket.remove(entry);
            size--;
            return value;
        }
    }

    return null;
}

public int size() {
    return size;
}

public boolean isEmpty() {
    return size == 0;
}
```

Реализовал метод удаления из таблицы, если ключ найден, иначе, возвращает null. Также создал методы size и isEmpty, возвращающие количество элементов и проверяет на пустоту таблицу.

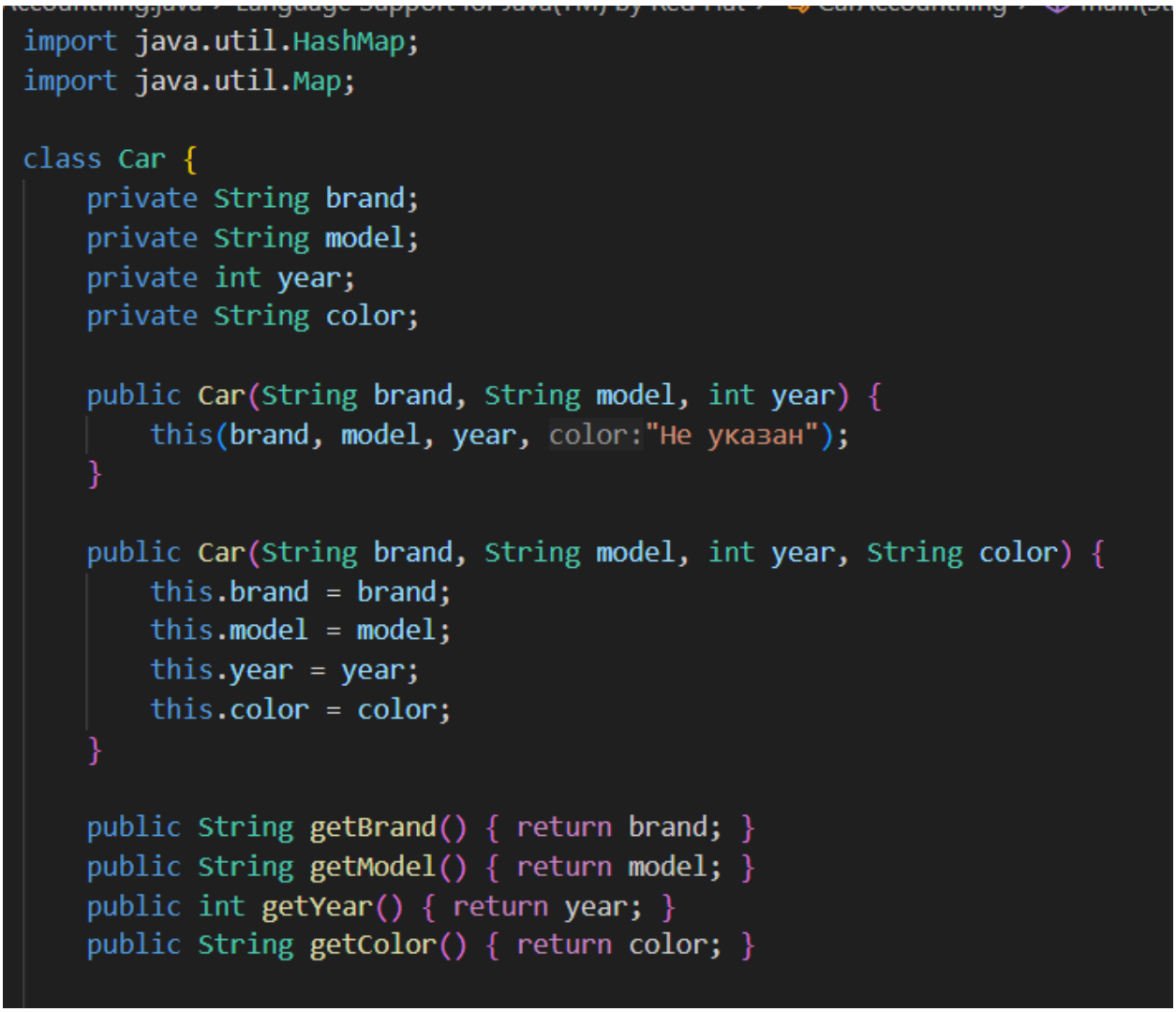
```java
@SuppressWarnings("unchecked")
private void rehash() {
    int newCapacity = capacity * 2;
    LinkedList<Entry<K, V>>[] oldTable = table;

    table = new LinkedList[newCapacity];
    capacity = newCapacity;
    size = 0;
    initializeTable();

    for (LinkedList<Entry<K, V>> bucket : oldTable) {
        for (Entry<K, V> entry : bucket) {
            put(entry.key, entry.value);
        }
    }
}
```

Реализовал метод rehash, который увеличивает размер таблицы в два раза, когда она заполняется больше, чем на 75%. При этом индексы ключей меняются.

```
    public void display() {
        for (int i = 0; i < capacity; i++) {
            System.out.print("Bucket " + i + ": ");
            LinkedList<Entry<K, V>> bucket = table[i];
            for (Entry<K, V> entry : bucket) {
                System.out.print("[" + entry.key + "=" + entry.value + "] ");
            }
            System.out.println();
        }
    }

    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        HashTable<String, Integer> hashTable = new HashTable<>();

        hashTable.put(key:"apple", value:1);
        hashTable.put(key:"banana", value:2);
        hashTable.put(key:"orange", value:3);
        hashTable.put(key:"apple", value:4);

        System.out.println("apple: " + hashTable.get(key:"apple"));
        System.out.println("banana: " + hashTable.get(key:"banana"));
        System.out.println("grape: " + hashTable.get(key:"grape"));

        System.out.println("Removed: " + hashTable.remove(key:"banana"));
        System.out.println("After removal - banana: " + hashTable.get(key:"banana"));

        System.out.println("Size: " + hashTable.size());

        System.out.println(x:"\nTable contents:");
        hashTable.display();
    }
}
```

Метод вывода информации на экран и метод main.

Задание 2: Реализация хэш-таблицы для учета автомобилей в авто парке. Ключом будет номерной знак автомобиля, а значением — объект класса Car, содержащий информацию о марке, модели и годе выпуска. Необходимо реализовать операции вставки, поиска и удаления автомобиля по знаку.



```java
import java.util.HashMap;
import java.util.Map;

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

    public Car(String brand, String model, int year) {
        this(brand, model, year, color:"Не указан");
    }

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

    public String getBrand() { return brand; }
    public String getModel() { return model; }
    public int getYear() { return year; }
    public String getColor() { return color; }
```

Создал класс Car, конструкторы по умолчанию и с параметрами, методы get.

```
    public void setColor(String color) { this.color = color; }

    @Override
    public String toString() {
        return String.format(brand, model, year, color);
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Car car = (Car) obj;
        return year == car.year &&
                brand.equals(car.brand) &&
                model.equals(car.model) &&
                color.equals(car.color);
    }

    @Override
    public int hashCode() {
        int result = brand.hashCode();
        result = result % 16;
        return result;
    }
}
```

Реализовал метод set, переписал методы toString, equals и hashCode.

```
public class CarAccounthing {
    private Map<String, Car> carAccount;

    public void CarAccInitialization() {
        this.carAccount = new HashMap<>();
    }

    public CarAccounthing() {
        this.carAccount = new HashMap<>(initialCapacity:16);
    }

    public CarAccounthing(int initialCapacity) {
        this.carAccount = new HashMap<>(initialCapacity);
    }

    public CarAccounthing(int initialCapacity, float loadFactor) {
        this.carAccount = new HashMap<>(initialCapacity, loadFactor);
```

Создал класс CarAccounthing, в нем создал хэш-таблицу типа Map, метод инициализации таблицы, конструкторы.

```
public String addCar(String NumberSign, Car car) {
    if (NumberSign == null || NumberSign.trim().isEmpty()) {
        return "Номерной знак не может быть пустым";
    }
    if (car == null) {
        return "Автомобиль не введен";
    }

    carAccount.put(NumberSign.toUpperCase(), car);
    return ("Автомобиль " + NumberSign + " добавлен в автопарк");
}

public Car findCar(String NumbeSign) {
    if (NumbeSign == null) return null;

    Car car = carAccount.get(NumbeSign.toUpperCase());
    if (car != null) {
        System.out.println("Найден автомобиль: " + NumbeSign + " - " + car);
    } else {
        System.out.println("Автомобиль с номером " + NumbeSign + " не найден");
    }
    return car;
}
```

Реализовал методы добавления и поиска авто.

```
public Car removeCar(String NumberSign) {
    if (NumberSign == null) return null;

    Car removedCar = carAccount.remove(NumberSign.toUpperCase());
    if (removedCar != null) {
        System.out.println("Автомобиль " + NumberSign + " удален из автопарка");
    } else {
        System.out.println("Автомобиль с номером " + NumberSign + " не найден для удаления");
    }
    return removedCar;
}

public boolean containsCar(String NumberString) {
    boolean exists = carAccount.containsKey(NumberString.toUpperCase());
    if (exists){System.out.println("Автомобиль с номером" + NumberString  + " в автопарке");}
    else {System.out.println("Автомобиль с номером" + NumberString  + " не в автопарке");}

    return exists;
}
```

Методы удаления авто из автопарк и проверка, состоит ли в автопарке авто с заданным номером.

```java
public void updateCar(String NumberSign, Car newCar) {
    if (NumberSign == null || newCar == null) return;

    if (carAccount.containsKey(NumberSign.toUpperCase())) {
        carAccount.put(NumberSign.toUpperCase(), newCar);
        System.out.println("Информация об автомобиле " + NumberSign + " обновлена");
    } else {
        System.out.println("Автомобиль с номером " + NumberSign + " не найден для обновления");
    }
}

public int getCarCount() {
    return carAccount.size();
}

public boolean isEmpty() {
    return carAccount.isEmpty();
}
```

Методы обновления информации об авто и получения размеров, проверка на наличие элементов в таблице.

```java
public void displayAllCars() {
    if (carAccount.isEmpty()) {
        System.out.println(x:"Автопарк пуст");
        return;
    }

    System.out.println("=== АВТОПАРК (всего автомобилей: " + carAccount.size() + ") ===");
    int counter = 1;
    for (Map.Entry<String, Car> entry : carAccount.entrySet()) {
        System.out.println(counter + ". " + entry.getKey() + " - " + entry.getValue());
        counter++;
    }
    System.out.println(x:"========================================\n");
}

public void clearAcc() {
    carAccount.clear();
    System.out.println(x:"Автопарк полностью очищен");
}

Run main | Debug main | Run | Debug
public static void main(String[] args) {
    CarAccounthing fleet = new CarAccounthing();

    fleet.addCar(NumberSign:"A123BC", new Car(brand:"Toyota", model:"Camry", year:2020, color:"Черный"));
    fleet.addCar(NumberSign:"B456DE", new Car(brand:"Honda", model:"Civic", year:2019, color:"Белый"));
    fleet.addCar(NumberSign:"C789FG", new Car(brand:"BMW", model:"X5", year:2022, color:"Синий"));
    fleet.addCar(NumberSign:"D012HI", new Car(brand:"Mercedes", model:"E-Class", year:2021, color:"Серый"));
    fleet.addCar(NumberSign:"E345JK", new Car(brand:"Toyota", model:"RAV4", year:2023, color:"Красный"));

    fleet.displayAllCars();

    fleet.findCar(NumbeSign:"A123BC");
    fleet.findCar(NumbeSign:"X999YY");

    fleet.containsCar(NumberString:"B456DE");
    fleet.containsCar(NumberString:"Z000ZZ");

    fleet.updateCar(NumberSign:"A123BC", new Car(brand:"Aboba", model:"Camry Hybrid", year:2023, color:"Черный"));
    fleet.findCar(NumbeSign:"A123BC");

    fleet.removeCar(NumberSign:"C789FG");
    fleet.removeCar(NumberSign:"NONEXISTENT");

    fleet.displayAllCars();

    System.out.println("   Всего автомобилей: " + fleet.getCarCount());
    System.out.println("   Автопарк пуст: " + fleet.isEmpty());
}
```

Метод выовда информации об автопарке, очищения автопарка. Реализовал функцию main.

Вывод: ознакомился с основыми работы с хэщ-таблицами.

https://github.com/Mirphon/ITIP/tree/main/Lab%203