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 информационным технологиям и программированию

на тему: «**Класс Object. Работа с хэш-таблицами**»

Выполнил: студент группы БПИ2403

ФИО: Сон Владимир Сергеевич

Руководитель: Рыбаков Егор Дмитриевич

Москва, 2025

Цель работы: Изучить роль класса Object как базового класса в Java, разобрать основные его методы (toString(), equals(), hashCode() и др.), а также понять принципы работы с хэш-таблицами, включая взаимосвязь методов equals() и hashCode().

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.

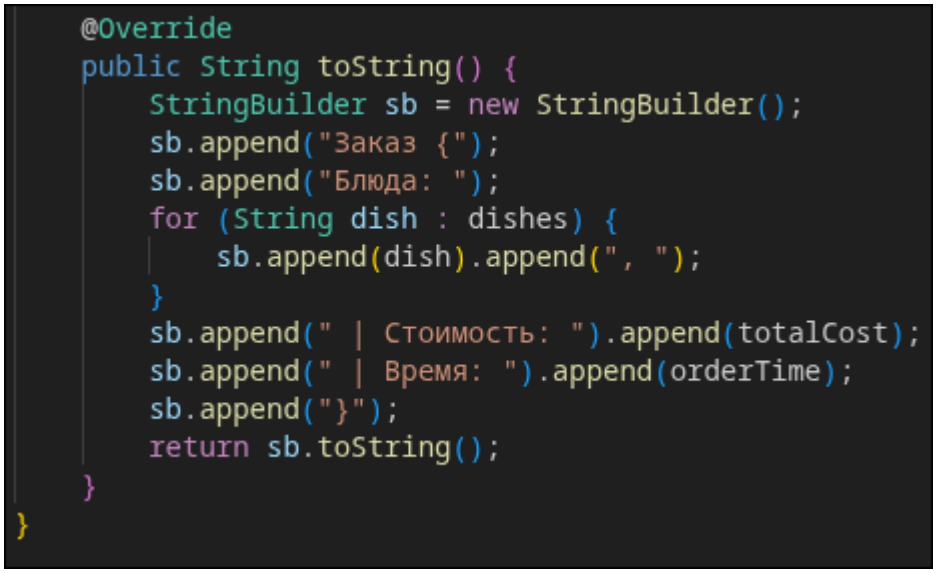
Задание 2. Работа с встроенным классом HashMap.

```java
import java.util.LinkedList;


public class HashTable<K, V> {
    private LinkedList<Entry<K, V>>[] table;
    private int size;
    private static final int DEFAULT_CAPACITY = 16;

    @SuppressWarnings("unchecked")
    public HashTable() {
        table = new LinkedList[DEFAULT_CAPACITY];
        size = 0;
    }

    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<Entry<K, V>>();
        }

        for (Entry<K, V> entry : table[index]) {
            if (entry.getKey().equals(key)) {
                entry.setValue(value);
                return;
            }
        }

        table[index].add(new Entry<K, V>(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;
    }
```

```java
public V remove(K key) {
    int index = hash(key);
    if (table[index] == null) {
        return null;
    }

    for (Entry<K, V> entry : table[index]) {
        if (entry.getKey().equals(key)) {
            V value = entry.getValue();
            table[index].remove(entry);
            size--;
            return value;
        }
    }

    return null;
}
```

```java
        Order order3 = new Order(dishes3, 2200.0, "20:00");

        orderTable.put(5, order1);
        orderTable.put(12, order2);
        orderTable.put(8, order3);

        orderTable.display();

        Order foundOrder = orderTable.get(12);
        if (foundOrder != null) {
            System.out.println("Заказ для столика 12: " + foundOrder);
        } else {
            System.out.println("Заказ для столика 12 не найден");
        }

        Order removedOrder = orderTable.remove(5);
        if (removedOrder != null) {
            System.out.println("Удален заказ: " + removedOrder);
        }

        orderTable.display();

        System.out.println("Количество заказов: " + orderTable.size());
        System.out.println("Таблица пуста: " + orderTable.isEmpty());

        String[] updatedDishes = {"Паста", "Суп", "Вино", "Десерт"};
        Order updatedOrder = new Order(updatedDishes, 2100.0, "19:15");
        orderTable.put(12, updatedOrder);

        System.out.println("Обновленный заказ для столика 12: " + orderTable.get(12));
    }
}

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 V getValue() {
        return value;
    }

    public void setValue(V value) {
        this.value = value;
    }
}

class Order {
    private String[] dishes;
    private double totalCost;
    private String orderTime;

    public Order(String[] dishes, double totalCost, String orderTime) {
        this.dishes = dishes;
        this.totalCost = totalCost;
        this.orderTime = orderTime;
    }

    public String[] getDishes() {
        return dishes;
    }

    public void setDishes(String[] dishes) {
        this.dishes = dishes;
    }

    public double getTotalCost() {
        return totalCost;
    }

    public void setTotalCost(double totalCost) {
        this.totalCost = totalCost;
    }

    public String getOrderTime() {
        return orderTime;
    }

    public void setOrderTime(String orderTime) {
        this.orderTime = orderTime;
    }
```



```
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("Заказ {");
        sb.append("Блюда: ");
        for (String dish : dishes) {
            sb.append(dish).append(", ");
        }
        sb.append(" | Стоимость: ").append(totalCost);
        sb.append(" | Время: ").append(orderTime);
        sb.append("}");
        return sb.toString();
    }
}
```

Вывод: В ходе работы было рассмотрено значение класса Object в Java и его ключевые методы, которые определяют поведение объектов. Особое внимание уделено методам equals() и hashCode() для корректной работы в хэш-таблицах (например, в HashMap, HashSet). Установлено, что переопределение этих методов необходимо для корректного сравнения содержимого объектов и обеспечения консистентности между равенством объектов и их хэш-кодами.