

**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 БПИ2401
Трухина Анастасия Александровна
Проверил: Харрасов Камиль Раисович

Москва,
2025

Оглавление

Цель работы:	3
Задание:	3
Основная часть.....	3
Задание 1:	3
Задание 2	6
Ответы на контрольные вопросы:	9
Заключение	12

Цель работы:

Изучить базовый класс Object в Java, принципы работы его методов (в частности, equals и hashCode), а также структуру и применение хэш-таблиц, включая реализацию собственной хэш-таблицы и использование встроенных коллекций типа HashMap.

Задание:

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.

Реализация хэш-таблицы для учета заказов в интернет магазине. Ключом будет номер заказа, а значением — объект класса Order, содержащий информацию о товарах, адресе доставки и стоимости заказа. Необходимо реализовать операции вставки, поиска и удаления заказа по номеру заказа

Основная часть

Задание 1:

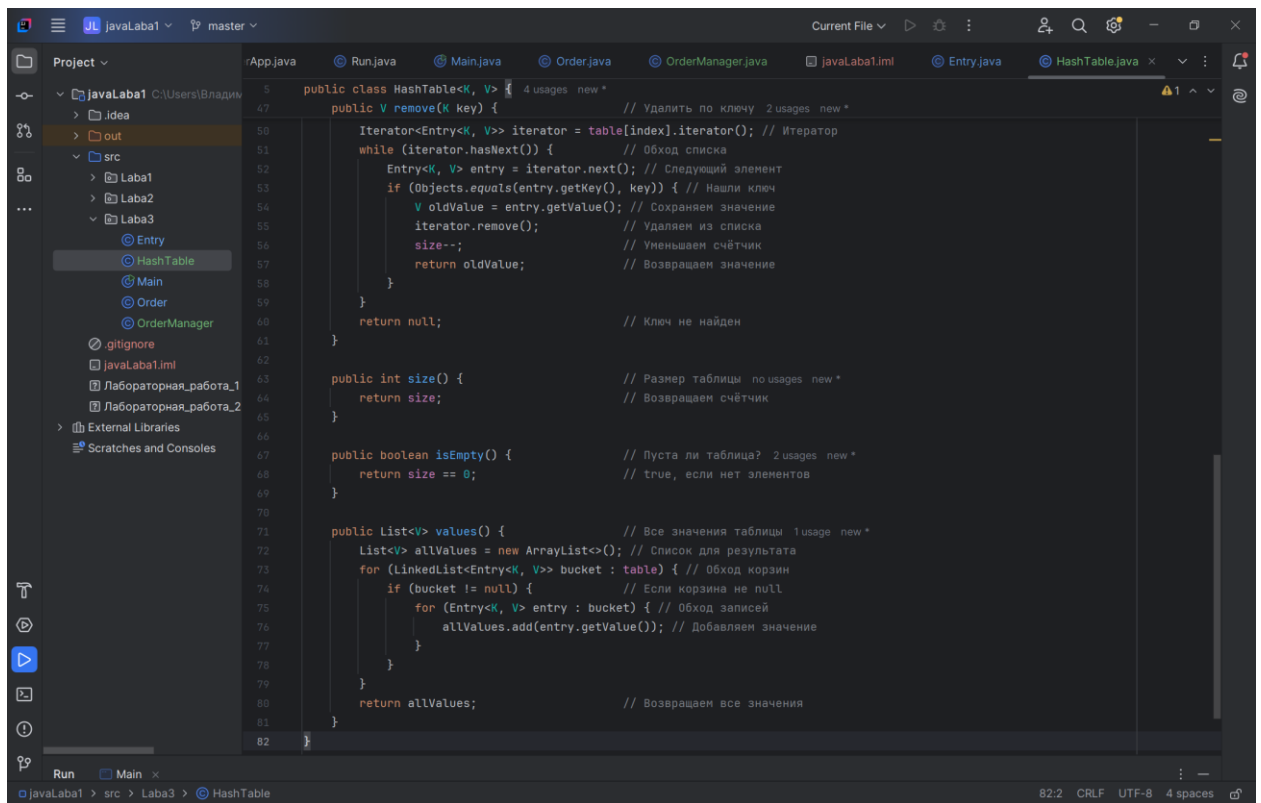
Хэш-таблица:

The screenshot shows the IntelliJ IDEA IDE with the 'HashTable.java' file open. The project structure on the left includes 'javaLab1' with subdirectories 'src' and 'out'. The 'src' directory contains files 'Entry', 'HashTable', 'Main', 'Order', and 'OrderManager'. The 'HashTable.java' file contains the following code:

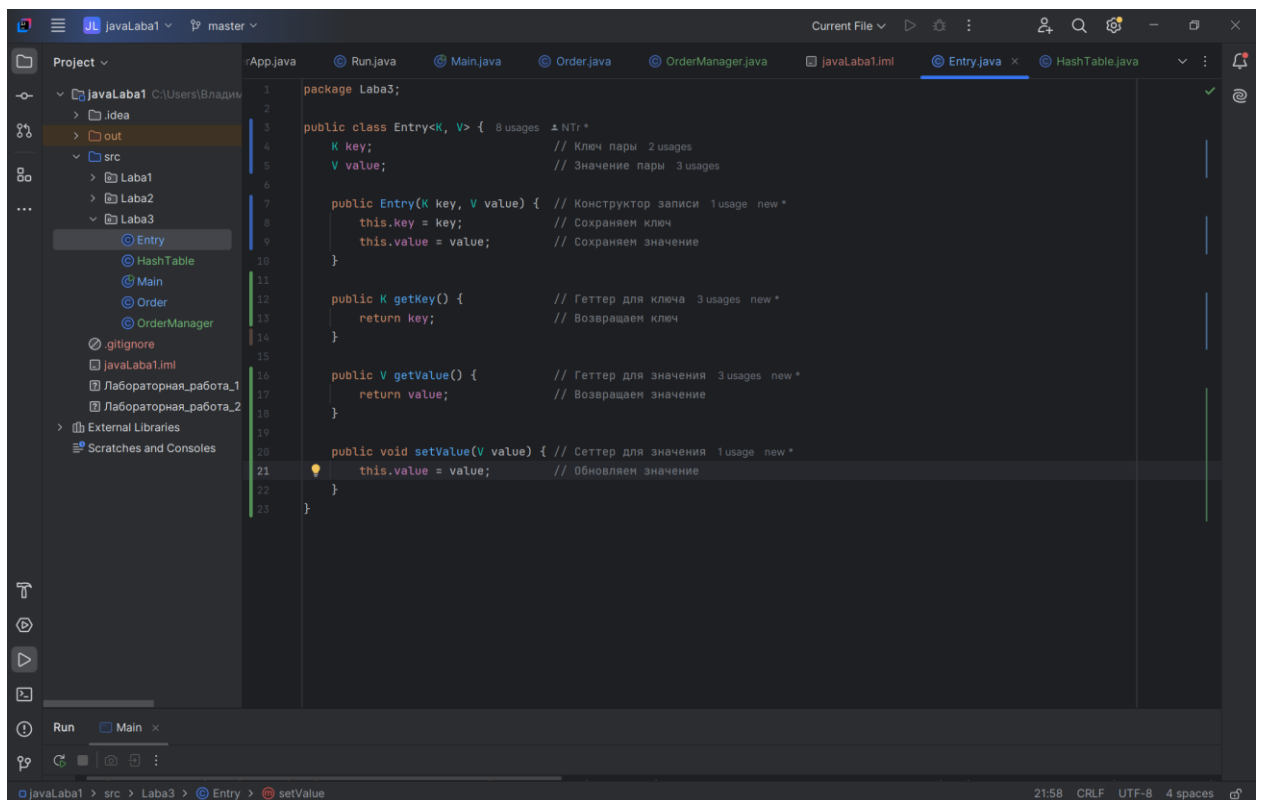
```
1 package Laba3;
2
3 import java.util.*;
4
5 public class HashTable<K, V> { // 4 usages new *
6     private LinkedList<Entry<K, V>>[] table; // Массив корзины (списков) 11 usages
7     private int size; // Количество элементов 5 usages
8     private static final int INITIAL_CAPACITY = 16; // Начальный размер 1 usage
9
10    @SuppressWarnings("unchecked") // 2 usages new *
11    public HashTable() { // Конструктор
12        table = new LinkedList<Entry<K, V>>[INITIAL_CAPACITY]; // Создаём массив
13        size = 0; // Изначально пусто
14    }
15
16    private int hash(K key) { // Хэш-функция 3 usages new *
17        if (key == null) return 0; // null → индекс 0
18        return Math.abs(key.hashCode()) % table.length; // Индекс в пределах
19    }
20
21    public void put(K key, V value) { // Добавить/обновить 5 usages new *
22        int index = hash(key); // Вычисляем индекс
23        if (table[index] == null) { // Если корзина пуста
24            table[index] = new LinkedList<>(); // Создаём список
25        }
26        for (Entry<K, V> entry : table[index]) { // Ищем ключ
27            if (Objects.equals(entry.getKey(), key)) { // Найден
28                entry.setValue(value); // Обновляем значение
29                return; // Выходим
30            }
31        }
32        table[index].add(new Entry<>(key, value)); // Добавляем новую запись
33        size++; // Увеличиваем счётчик
34    }
35}
```

The screenshot shows the IntelliJ IDEA IDE with the 'HashTable.java' file open. The project structure on the left is the same as the previous screenshot. The 'HashTable.java' file contains the following code:

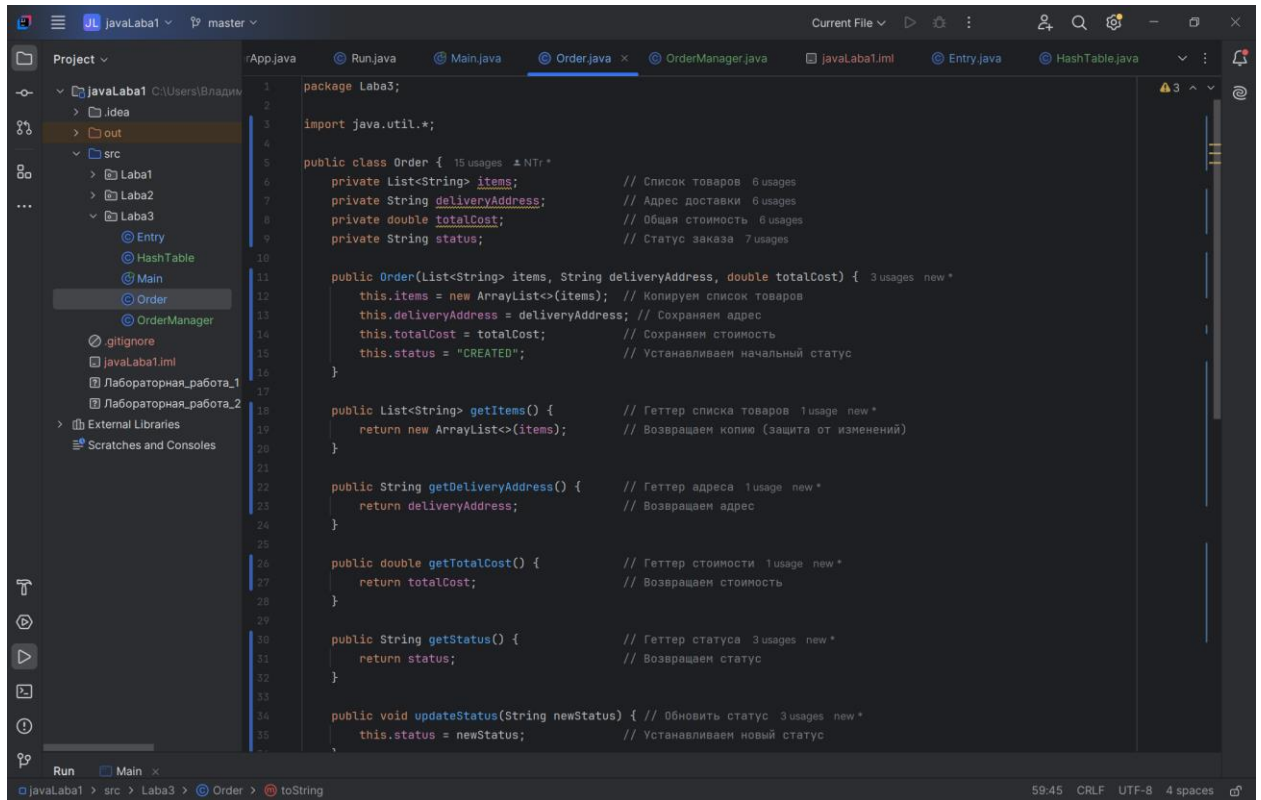
```
5 public class HashTable<K, V> { // 4 usages new *
6     public void put(K key, V value) { // Добавить/обновить 5 usages new *
7         size++; // Увеличиваем счётчик
8     }
9
10    public V get(K key) { // Получить значение 9 usages new *
11        int index = hash(key); // Индекс по ключу
12        if (table[index] == null) return null; // Корзина пуста
13        for (Entry<K, V> entry : table[index]) { // Ищем в списке
14            if (Objects.equals(entry.getKey(), key)) { // Ключ найден
15                return entry.getValue(); // Возвращаем значение
16            }
17        }
18        return null; // Не найден
19    }
20
21    public V remove(K key) { // Удалить по ключу 2 usages new *
22        int index = hash(key); // Индекс
23        if (table[index] == null) return null; // Нет корзины
24        Iterator<Entry<K, V>> iterator = table[index].iterator(); // Итератор
25        while (iterator.hasNext()) { // Обход списка
26            Entry<K, V> entry = iterator.next(); // Следующий элемент
27            if (Objects.equals(entry.getKey(), key)) { // Нашли ключ
28                V oldValue = entry.getValue(); // Сохраняем значение
29                iterator.remove(); // Удаляем из списка
30                size--; // Уменьшаем счётчик
31                return oldValue; // Возвращаем значение
32            }
33        }
34        return null; // Ключ не найден
35    }
36
37    public int size() { // Размер таблицы по usages new *
38        return size; // Возвращаем счётчик
39    }
40}
```



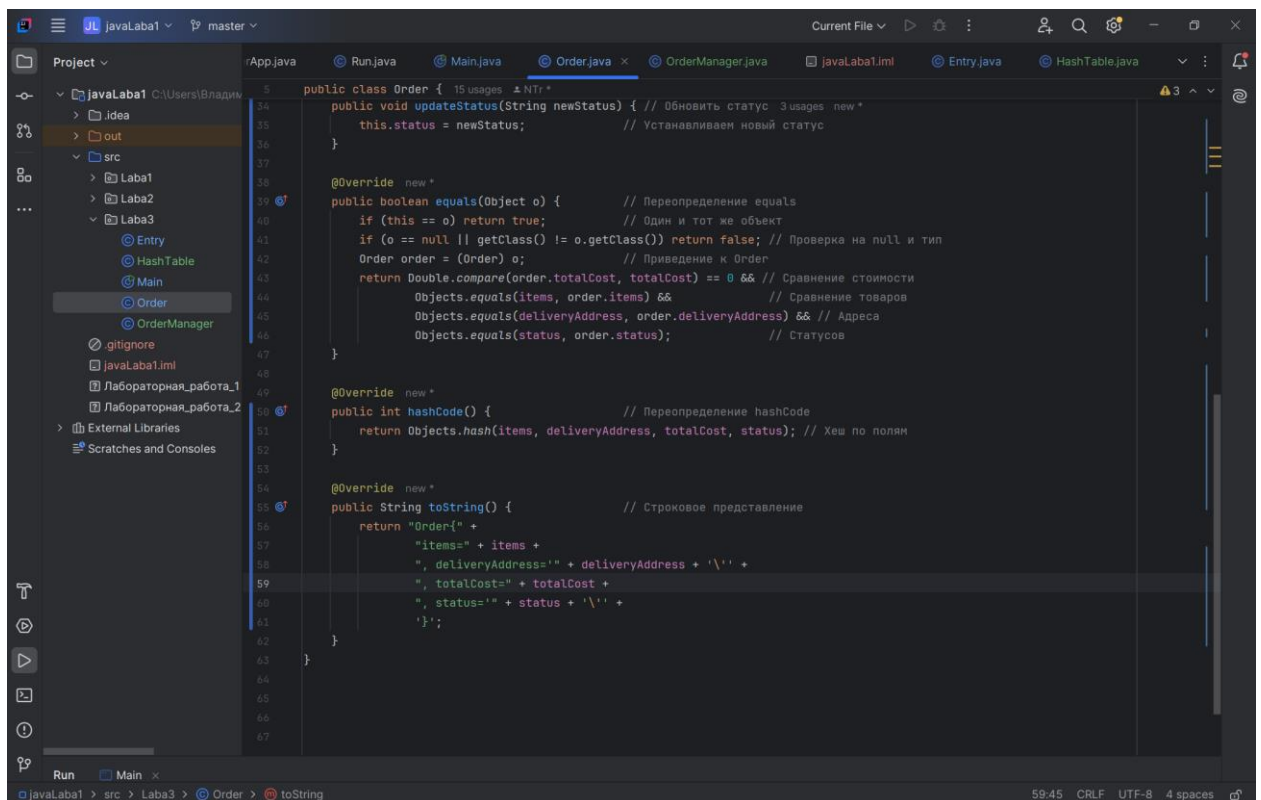
Entry класс с generics:



Задание 2



```
1 package Laba3;
2
3 import java.util.*;
4
5 public class Order {
6     private List<String> items; // Список товаров
7     private String deliveryAddress; // Адрес доставки
8     private double totalCost; // Общая стоимость
9     private String status; // Статус заказа
10
11     public Order(List<String> items, String deliveryAddress, double totalCost) {
12         this.items = new ArrayList<>(items); // Копируем список товаров
13         this.deliveryAddress = deliveryAddress; // Сохраняем адрес
14         this.totalCost = totalCost; // Сохраняем стоимость
15         this.status = "CREATED"; // Устанавливаем начальный статус
16     }
17
18     public List<String> getItems() {
19         return new ArrayList<>(items); // Возвращаем копию (защита от изменений)
20     }
21
22     public String getDeliveryAddress() {
23         return deliveryAddress; // Возвращаем адрес
24     }
25
26     public double getTotalCost() {
27         return totalCost; // Возвращаем стоимость
28     }
29
30     public String getStatus() {
31         return status; // Возвращаем статус
32     }
33
34     public void updateStatus(String newStatus) {
35         this.status = newStatus; // Обновить статус
36     }
37 }
```



```
38
39 @Override
40 public boolean equals(Object o) {
41     if (this == o) return true; // Проверка на null и тип
42     if (o == null || getClass() != o.getClass()) return false;
43     Order order = (Order) o; // Приведение к Order
44     return Double.compare(order.totalCost, totalCost) == 0 &&
45         Objects.equals(items, order.items) &&
46         Objects.equals(deliveryAddress, order.deliveryAddress) &&
47         Objects.equals(status, order.status);
48 }
49
50 @Override
51 public int hashCode() {
52     return Objects.hash(items, deliveryAddress, totalCost, status);
53 }
54
55 @Override
56 public String toString() {
57     return "Order{" +
58         "items=" + items +
59         ", deliveryAddress=" + deliveryAddress + '\n' +
60         ", totalCost=" + totalCost +
61         ", status=" + status + '\n' +
62         '}';
63 }
64
65
66
67 }
```

```
1 package Laba3;
2
3 import java.util.*;
4
5 public class OrderManager {
6     private HashTable<Long, Order> orders = new HashTable<>(); // Собственная хэш-таблица
7
8     public void addOrder(Long orderId, Order order) { // Добавить заказ
9         orders.put(orderId, order); // Используем put из HashTable
10    }
11
12    public Order findOrder(Long orderId) { // Найти заказ
13        return orders.get(orderId); // Используем get из HashTable
14    }
15
16    public Order removeOrder(Long orderId) { // Удалить заказ
17        return orders.remove(orderId); // Используем remove из HashTable
18    }
19
20    public boolean updateOrderStatus(Long orderId, String newStatus) { // Изменить статус
21        Order order = orders.get(orderId); // Получаем заказ
22        if (order != null) { // Если существует
23            order.updateStatus(newStatus); // Меняем статус
24            return true; // Успешно
25        }
26        return false; // Заказ не найден
27    }
28
29    public List<Order> getAllOrders() { // Все заказы
30        return orders.values(); // Используем values() из HashTable
31    }
32}
```

Run Main x

javaLaba1 > src > Laba3 > OrderManager > findOrder 13:84 CRLF UTF-8 4 spaces

```
1 package Laba3;
2
3 import java.util.*;
4
5 public class Main {
6     public static void main(String[] args) {
7         System.out.println("=".repeat(60));
8         System.out.println("ДЕМОНСТРАЦИЯ РАБОТЫ HashTable");
9         System.out.println("=".repeat(60));
10
11         HashTable<String, Integer> ht = new HashTable<>();
12
13         // isEmpty()
14         System.out.println("1. isEmpty() -> " + ht.isEmpty()); // true
15
16         // put()
17         ht.put("apple", 5);
18         ht.put("banana", 3);
19         ht.put("orange", 7);
20         System.out.println("2. Добавлены: apple " + ht.get("apple") + ", banana " + ht.get("banana") + ", orange " + ht.get("orange"));
21
22         // size()
23         System.out.println("3. size() -> " + ht.size()); // 3
24
25         // get()
26         System.out.println("4. get('apple') -> " + ht.get("apple")); // 5
27         System.out.println("   get('grape') -> " + ht.get("grape")); // null
28
29         // put() с обновлением
30         ht.put("apple", 10);
31         System.out.println("5. Обновили 'apple' на 10 -> get('apple') = " + ht.get("apple"));
32
33         // remove()
34         Integer removed = ht.remove("banana");
35         System.out.println("6. remove('banana') -> " + removed); // 3
36    }
37}
```

Run Main x

javaLaba1 > src > Laba3 > Main > main 21:1 CRLF UTF-8 4 spaces

```
Project ▾ javaLab1 ▾ master ▾
  ▾ javaLab1 C:\Users\Владим
    ▾ .idea
    ▾ out
    ▾ src
      ▾ Laba1
      ▾ Laba2
      ▾ Laba3
        Entry
        HashTable
        Main
        Order
        OrderManager
      .gitignore
      javaLab1.iml
      Лабораторная_работа_1
      Лабораторная_работа_2
    ▾ External Libraries
    ▾ Scratches and Consoles

App.java Run.java Main.java Order.java OrderManager.java javaLab1.iml Entry.java HashTable.java
5 public class Main { @NTr+
6   public static void main(String[] args) { @NTr+
33   // remove()
34   Integer removed = ht.remove(key: "banana");
35   System.out.println("6. remove('banana') → " + removed); // 3
36   System.out.println("   get('banana') после удаления → " + ht.get("banana")); // null
37   System.out.println("   size() после удаления → " + ht.size()); // 2
38
39   // isEmpty() после добавления
40   System.out.println("7. isEmpty() после операций → " + ht.isEmpty()); // false
41
42   System.out.println("\n" + "=".repeat(count: 60));
43   System.out.println("ДЕМОНСТРАЦИЯ РАБОТЫ Order и OrderManager (Вариант 7)");
44   System.out.println("=".repeat(count: 60));
45
46   OrderManager manager = new OrderManager();
47
48   // Создаём заказы
49   List<String> items1 = Arrays.asList("Нюгтбук", "Мышь");
50   Order order1 = new Order(items1, deliveryAddress: "Москва, Тверская 1", totalCost: 95000.0);
51   Order order2 = new Order(
52     Arrays.asList("Книга"),
53     deliveryAddress: "СПб, Невский 10",
54     totalCost: 1200.0
55   );
56
57   // toString()
58   System.out.println("1. order1.toString() → " + order1);
59
60   // Геттеры
61   System.out.println("2. order1.getItems() → " + order1.getItems());
62   System.out.println("   order1.getDeliveryAddress() → " + order1.getDeliveryAddress());
63   System.out.println("   order1.getTotalCost() → " + order1.getTotalCost());
64   System.out.println("   order1.getStatus() → " + order1.getStatus());
65
Run Main ×
javaLab1 ▸ src ▸ Laba3 ▸ Main ▸ main 21:1 CRLF UTF-8 4 spaces
```

```
Project ▾ javaLab1 ▾ master ▾
  ▾ javaLab1 C:\Users\Владим
    ▾ .idea
    ▾ out
    ▾ src
      ▾ Laba1
      ▾ Laba2
      ▾ Laba3
        Entry
        HashTable
        Main
        Order
        OrderManager
      .gitignore
      javaLab1.iml
      Лабораторная_работа_1
      Лабораторная_работа_2
    ▾ External Libraries
    ▾ Scratches and Consoles

App.java Run.java Main.java Order.java OrderManager.java javaLab1.iml Entry.java HashTable.java
5 public class Main { @NTr+
6   public static void main(String[] args) { @NTr+
63   System.out.println("   order1.getTotalCost() → " + order1.getTotalCost());
64   System.out.println("   order1.getStatus() → " + order1.getStatus());
65
66   // updateStatus()
67   order1.updateStatus(newStatus: "SHIPPED");
68   System.out.println("3. После updateStatus('SHIPPED') → статус: " + order1.getStatus());
69
70   // equals() и hashCode()
71   Order order1Copy = new Order(items1, deliveryAddress: "Москва, Тверская 1", totalCost: 95000.0);
72   order1Copy.updateStatus(newStatus: "SHIPPED"); // чтобы статус совпадал
73   System.out.println("4. order1.equals(order1Copy) → " + order1.equals(order1Copy)); // true
74   System.out.println("   order1.hashCode() == order1Copy.hashCode() → " + (order1.hashCode() == order1Copy.hashCode())); // true
75
76   // addOrder()
77   manager.addOrder(orderId: 1001L, order1);
78   manager.addOrder(orderId: 1002L, order2);
79   System.out.println("5. Заказы 1001 и 1002 добавлены в OrderManager");
80
81   // findOrder()
82   System.out.println("6. findOrder(1001) → " + manager.findOrder(orderId: 1001L));
83   System.out.println("   findOrder(999) → " + manager.findOrder(orderId: 999L)); // null
84
85   // updateOrderStatus()
86   boolean updated = manager.updateOrderStatus(orderId: 1002L, newStatus: "PROCESSING");
87   System.out.println("7. updateOrderStatus(1002, 'PROCESSING') → " + updated); // true
88   System.out.println("   Статус заказа 1002 теперь: " + manager.findOrder(orderId: 1002L).getStatus());
89
90   // getAllOrders()
91   System.out.println("8. Всего заказов: " + manager.getAllOrders().size()); // 2
92
93   // removeOrder()
94   Order removedOrder = manager.removeOrder(orderId: 1001L);
95   System.out.println("9. removeOrder(1001) → удалён: " + (removedOrder != null));
96
Run Main ×
javaLab1 ▸ src ▸ Laba3 ▸ Main ▸ main 21:1 CRLF UTF-8 4 spaces
```



```
5 public class Main {
6     public static void main(String[] args) {
7         // addOrder()
76         manager.addOrder( orderId: 1001L, order1);
77         manager.addOrder( orderId: 1002L, order2);
78         System.out.println("5. Заказы 1001 и 1002 добавлены в OrderManager");
79
80
81         // findOrder()
82         System.out.println("6. findOrder(1001) -> " + manager.findOrder( orderId: 1001L));
83         System.out.println("7. findOrder(999) -> " + manager.findOrder( orderId: 999L)); // null
84
85         // updateOrderStatus()
86         boolean updated = manager.updateOrderStatus( orderId: 1002L, newStatus: "PROCESSING");
87         System.out.println("7. updateOrderStatus(1002, 'PROCESSING') -> " + updated); // true
88         System.out.println("Статус заказа 1002 теперь: " + manager.findOrder( orderId: 1002L).getStatus());
89
90         // getAllOrders()
91         System.out.println("8. Всего заказов: " + manager.getAllOrders().size()); // 2
92
93         // removeOrder()
94         Order removedOrder = manager.removeOrder( orderId: 1001L);
95         System.out.println("9. removeOrder(1001) -> удалён: " + (removedOrder != null));
96         System.out.println("findOrder(1001) после удаления -> " + manager.findOrder( orderId: 1001L)); // null
97         System.out.println("Всего заказов после удаления: " + manager.getAllOrders().size()); // 1
98
99         // Попытка обновить несуществующий заказ
100         boolean failedUpdate = manager.updateOrderStatus( orderId: 9999L, newStatus: "DELIVERED");
101         System.out.println("10. updateOrderStatus(9999, 'DELIVERED') -> " + failedUpdate); // false
102     }
103 }
```

```
=====
1. isEmpty() -> true
2. Добавлены: apple 5, banana 3, orange 7
3. size() -> 3
4. get('apple') -> 5
   get('grape') -> null
5. Обновили 'apple' на 10 -> get('apple') = 10
6. remove('banana') -> 3
   get('banana') после удаления -> null
   size() после удаления -> 2
7. isEmpty() после операций -> false

=====
ДЕМОНСТРАЦИЯ РАБОТЫ Order и OrderManager (Вариант 7)
=====
1. order1.toString() -> Order{items=[Ноутбук, Мышь], deliveryAddress='Москва, Тверская 1', totalCost=95000.0, status='CREATED'}
2. order1.getItems() -> [Ноутбук, Мышь]
   order1.getDeliveryAddress() -> Москва, Тверская 1
   order1.getTotalCost() -> 95000.0
   order1.getStatus() -> CREATED
3. После updateStatus('SHIPPED') -> статус: SHIPPED
4. order1.equals(order1Copy) -> true
   order1.hashCode() == order1Copy.hashCode() -> true
5. Заказы 1001 и 1002 добавлены в OrderManager
6. findOrder(1001) -> Order{items=[Ноутбук, Мышь], deliveryAddress='Москва, Тверская 1', totalCost=95000.0, status='SHIPPED'}
   findOrder(999) -> null
7. updateOrderStatus(1002, 'PROCESSING') -> true
   Статус заказа 1002 теперь: PROCESSING
8. Всего заказов: 2
9. removeOrder(1001) -> удалён: true
   findOrder(1001) после удаления -> null
   Всего заказов после удаления: 1
10. updateOrderStatus(9999, 'DELIVERED') -> false
```

Ответы на контрольные вопросы:

1. Класс Object является корнем всей иерархии классов в языке Java. Это означает, что каждый класс, включая пользовательские, автоматически наследует класс Object, даже если явного указания на это нет. Благодаря этому, каждый объект в Java обладает набором

базовых методов, определенных в классе Object. Это включает такие методы, как equals(), hashCode(), toString(), getClass(), clone(), а также методы для синхронизации wait(), notify(), notifyAll(). Наследование от Object обеспечивает единообразие и предоставляет универсальные возможности для всех объектов в системе.

2. Метод equals() по умолчанию сравнивает ссылки на объекты, то есть проверяет, являются ли две переменные ссылками на один и тот же объект в памяти. Это поведение часто недостаточно, когда нужно сравнить содержимое объектов. Например, два разных объекта Person с одинаковыми полями name и age логично считать равными. Для этого equals() нужно переопределить. Метод hashCode() переопределяют, потому что хэш-коллекции (HashMap, HashSet) используют хеш-коды для эффективного хранения и поиска элементов. Согласно контракту, если два объекта равны по equals(), они обязаны возвращать одинаковый хеш-код. Несоблюдение этого правила приведет к неправильной работе коллекций, например, объект может быть добавлен, но не найден позже.

3. При переопределении equals() нужно соблюдать несколько важных правил, известных как контракт equals(). Метод должен быть рефлексивным: x.equals(x) всегда должно возвращать true. Он должен быть симметричным: если x.equals(y) возвращает true, то y.equals(x) тоже должен возвращать true. Также требуется транзитивность: если x.equals(y) и y.equals(z) возвращают true, то и x.equals(z) должно возвращать true. Метод должен быть согласованным: при отсутствии изменений в объектах, результат вызова equals() должен оставаться постоянным. Наконец, для любого ненулевого объекта x, вызов x.equals(null) должен возвращать false. При переопределении hashCode() необходимо строго следовать контракту: равные объекты (согласно equals()) обязаны возвращать одинаковый хеш-код. Неравные объекты могут возвращать одинаковый хеш-код (коллизия), но для эффективности лучше, чтобы разные объекты возвращали разные хеш-коды.

4. Метод toString() возвращает строковое представление объекта. По умолчанию он возвращает имя класса и шестнадцатеричное представление хеш-кода объекта, например, Person@15db9742. Такое строковое представление не несёт полезной информации о состоянии объекта. Его часто переопределяют, чтобы получить человеко-читаемую и информативную строку, описывающую текущие значения полей объекта. Это чрезвычайно удобно для отладки кода, логирования действий программы и вывода информации о состоянии объекта в консоль или в интерфейс.

5. Метод finalize() вызывался сборщиком мусора перед тем, как объект физически удалялся из памяти. Его можно было использовать для выполнения очистки, например, закрытия файловых дескрипторов или сетевых соединений, если объект владел ими. Однако использование finalize() считается устаревшим, потому что поведение и время его вызова непредсказуемы. Это может привести к утечкам ресурсов, проблемам с

производительностью и непредсказуемому поведению программы. Современные альтернативы, такие как `try-with-resources` и `API Cleaner`, предоставляют более надежные и предсказуемые способы управления ресурсами.

6. Коллизия — это ситуация, которая возникает в хэш-таблице, когда два или более разных ключа при вычислении хеш-функции получают один и тот же индекс в массиве хранения. Это неизбежный побочный эффект работы хэш-функций, так как количество возможных ключей обычно гораздо больше, чем количество возможных индексов в массиве. Коллизии не означают ошибку, но требуют специального способа разрешения, чтобы оба (или более) значения могли быть корректно сохранены и найдены.

7. Существует несколько способов разрешения коллизий. Один из наиболее распространённых — метод цепочек (`chaining`). В этом методе каждая ячейка массива хэш-таблицы (корзина) содержит структуру данных, например, связный список, для хранения всех пар "ключ-значение", которые попадают в эту ячейку. При коллизии новая пара добавляется в список. Другой способ — открытая адресация (`open addressing`). При коллизии в открытой адресации алгоритм ищет другую свободную ячейку в массиве по определённой стратегии (например, линейный пробинг, квадратичный пробинг, двойное хеширование) и помещает пару туда.

8. Хэш-таблица — это структура данных, предназначенная для хранения пар "ключ-значение". Внутри она использует массив "корзин" (`buckets`). Ключ передаётся в хеш-функцию, которая вычисляет индекс массива, где должна храниться пара. Значение сохраняется по этому индексу. Если два разных ключа получают одинаковый индекс (коллизия), используется один из методов разрешения коллизий, например, цепочки, где в одной корзине хранится список пар. Это позволяет эффективно находить, добавлять и удалять элементы по ключу.

9. Если в хэш-таблицу (например, в `HashMap`) добавить элемент с ключом, который уже существует в таблице, старое значение, связанное с этим ключом, будет заменено новым значением. Это стандартное поведение метода `put()`. Сравнение ключей при этом производится с помощью метода `equals()` объекта ключа. Таким образом, таблица всегда содержит только одно значение для каждого уникального ключа (с точки зрения `equals()`).

10. Если в хэш-таблицу добавить элемент с ключом, который имеет такой же хеш-код, что и у другого ключа, но при этом фактически это разные ключи (сравнение по `equals()` возвращает `false`), это также приведёт к коллизии. Эта новая пара "ключ-значение" будет добавлена в ту же "корзину", что и пара с одинаковым хеш-кодом (например, в список, если используется метод цепочек). При поиске значение будет найдено правильно, потому

что после определения корзины по хеш-коду, происходит перебор элементов в корзине, и ключи сравниваются с помощью метода equals(), а не только по хеш-коду.

11. HashMap изменяется при достижении порогового значения, которое вычисляется как `capacity * loadFactor`. LoadFactor (коэффициент загрузки) — это мера того, насколько заполнена таблица, прежде чем произойдёт увеличение размера. Когда количество элементов (size) превышает порог, происходит процесс, называемый resize (изменение размера). Внутри HashMap увеличивает свою внутреннюю ёмкость (capacity), обычно в 2 раза. Затем все существующие элементы перехэшируются и помещаются в новую, большую таблицу. Это делается для поддержания эффективности операций (чтобы средняя длина цепочек или количество коллизий оставалось низким).

Заключение

Вывод:

В ходе лабораторной работы изучены особенности базового класса Object и реализована собственная хэш-таблица с методом цепочек. Рассмотрено переопределение методов equals и hashCode. Выполнена реализация системы учёта заказов с использованием разработанной структуры данных.

Ссылка на ГитХаб с файлами кода: [NT-005-TN/ITiP_LabWorks_Trukhina](https://github.com/NT-005-TN/ITiP-LabWorks-Trukhina)