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»

(МТУСИ)

Кафедра «Математическая кибернетика и информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

по дисциплине

«Информационные технологии и программирование»

на тему

«Хэш-таблица»

Выполнил: студент группы БВТ2302

Никитин Андрей Александрович

Проверил:

Харрасов Камиль Раисович

Москва, 2024

Цель

Понять, как работают хеш-таблицы в языке программирования Java и научиться ими пользоваться.

Задачи

Задание 1:

1. Создать класс HashTable, который будет реализовывать хэш-таблицу с помощью метода цепочек.

2. Реализовать методы put(key, value), get(key) и remove(key), которые добавляют, получают и удаляют пары «ключ-значение» соответственно.

3. Добавить методы size() и isEmpty(), которые возвращают количество элементов в таблице и проверяют, пуста ли она.

Задание 2: Работа с встроенным классом HashMap

Вариант 3: Реализация хэш-таблицы для хранения информации о заказах в интернет-магазине. Ключом является номер заказа, а значением - объект класса Order, содержащий поля дата заказа, список товаров и статус заказа. Необходимо реализовать операции вставки, поиска и удаления заказа по номеру. Также необходимо реализовать метод для изменения статуса заказа.

Ход работы

Задание 1

Импортируем *LinkedList* и создадим публичный класс *HashTable*, с параметризированными параметрами K и V. Внутри класс *HashTable* задаем приватные поля:

LinkedList<Entry<K, V>> [] buckets - создаем массив списков *LinkedList*, в которых будут храниться пары “ключ-значение” типа *Entry<K, V>;*

int size - поле для отслеживания количества элементов в таблице;

private static final int INITIAL_CAPACITY = 16 - изначальная емкость таблицы.

```
import java.util.LinkedList;

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

Рис. 1 Создание полей класса HashTable

Далее создадим внутренний статический класс Entry, который используется в классе *HashTable* для представления пары "ключ-значение". Каждый экземпляр класса *Entry* содержит поле *key* типа *K* и поле *value* типа *V*, представляющие ключ и значение соответственно.

Конструктор класса Entry принимает два параметра: *key* и *value*, и инициализирует соответствующие поля экземпляра.

Класс *Entry* также содержит два геттера и сеттер:

- *getKey()* возвращает значение поля key.
- *getValue()* возвращает значение поля value.
- *setValue(V value)* устанавливает новое значение для поля value.

```
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;
    }
}
```

Рис. 2 Внутренний статический класс Entry

Далее зададим конструктор для инициализации нового экземпляра класса HashTable.

@SuppressWarnings("unchecked") - это аннотация, которая подавляет предупреждение о неявном приведении типа при создании массива buckets.

Это предупреждение возникает из-за того, что Java не может гарантировать безопасность приведения типа от Object[] к LinkedList<Entry<K, V>>[].

Создается новый массив списков с инициализацией размера INITIAL_CAPACITY. Каждый элемент массива инициализируется значением null.

size = 0; - инициализируется поле size, представляющее количество элементов в хеш-таблице, значением 0.

```
@SuppressWarnings("unchecked")
public HashTable() {
    buckets = new LinkedList[INITIAL_CAPACITY];
    size = 0;
}
```

Рис. 3 Конструктор класса HashTable

Далее создадим метод *hash()*, который используется для вычисления индекса в массиве buckets, где будет храниться пара "ключ-значение" с данным ключом. Метод выполняет следующие действия:

key.hashCode() - вызывает метод *hashCode()* объекта *key*, который возвращает хеш-код ключа.

Math.abs(key.hashCode()) - применяет метод *Math.abs()* для получения абсолютного значения хеш-кода ключа. Это гарантирует, что результат будет неотрицательным, даже если хеш-код ключа отрицателен.

% buckets.length - применяет оператор остатка от деления для получения индекса в массиве *buckets*. Это гарантирует, что индекс будет в пределах длины массива *buckets*.

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

Рис. 4 Метод hash()

Далее создадим метод *put()*, который используется для добавления новой пары "ключ-значение" в хеш-таблицу или обновления значения существующей пары с данным ключом. Метод выполняет следующие действия:

int index = hash(key); - вычисляет индекс в массиве buckets, где будет храниться пара "ключ-значение" с данным ключом, с помощью метода *hash()*.

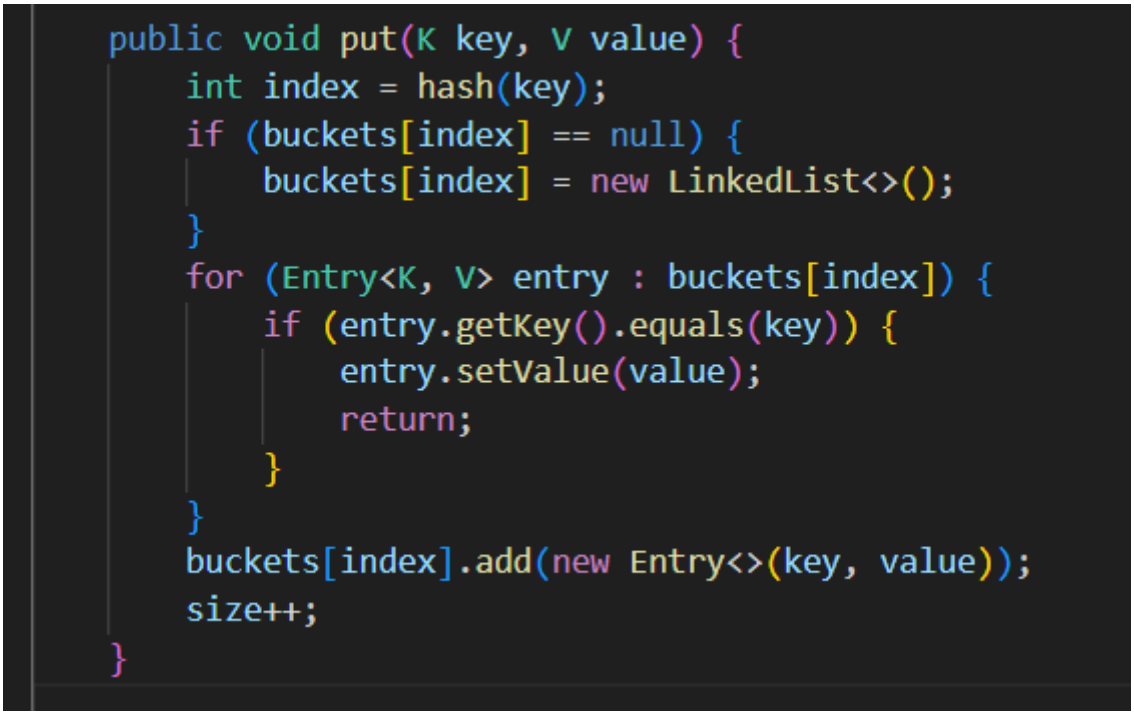
if (buckets[index] == null) { - проверяет, существует ли список в массиве buckets по индексу index. Если списка нет, создается новый список и помещается в массив buckets по индексу *index*.

for (Entry<K, V> entry : buckets[index]) { - проходит по всем входам (парам "ключ-значение") в списке, связанном с индексом index.

if (entry.getKey().equals(key)) { - проверяет, существует ли пара "ключ-значение" с данным ключом в списке. Если пара найдена, значение этой пары обновляется значением value, переданным в метод put, и метод завершает свою работу, возвращаясь из него.

buckets[index].add(new Entry<>(key, value)); - если пара "ключ-значение" с данным ключом не найдена в списке, создается новая пара с ключом *key* и значением *value*, и добавляется в конец списка, связанного с индексом *index*.

size++; - увеличивает счетчик количества элементов в хеш-таблице на 1.



```
public void put(K key, V value) {
    int index = hash(key);
    if (buckets[index] == null) {
        buckets[index] = new LinkedList<>();
    }
    for (Entry<K, V> entry : buckets[index]) {
        if (entry.getKey().equals(key)) {
            entry.setValue(value);
            return;
        }
    }
    buckets[index].add(new Entry<>(key, value));
    size++;
}
```

Рис. 5 Метод put()

Далее создаем метод *get()*, который используется для получения значения по заданному ключу из хеш-таблицы. Метод выполняет следующие действия:

int index = hash(key); - вычисляет индекс в массиве *buckets*, где может находиться пара "ключ-значение" с данным ключом, с помощью метода *hash()*.

if (buckets[index] != null) { - проверяет, существует ли список в массиве buckets по индексу index. Если списка нет, метод возвращает null, так как пара "ключ-значение" с данным ключом не найдена.

for (Entry<K, V> entry : buckets[index]) { - проходит по всем входам (парам "ключ-значение") в списке, связанном с индексом *index*.

if (entry.getKey().equals(key)) { - проверяет, существует ли пара "ключ-значение" с данным ключом в списке. Если пара найдена, метод возвращает значение этой пары.

Если метод не находит пару "ключ-значение" с данным ключом в списке, он возвращает *null*.

```
public V get(K key) {
    int index = hash(key);
    if (buckets[index] != null) {
        for (Entry<K, V> entry : buckets[index]) {
            if (entry.getKey().equals(key)) {
                return entry.getValue();
            }
        }
    }
    return null;
}
```

Рис. 6 Метод get()

Далее создаем метод *remove()*, используется для удаления пары "ключ-значение" с заданным ключом из хеш-таблицы. Метод выполняет следующие действия:

int index = hash(key); - вычисляет индекс в массиве buckets, где может находиться пара "ключ-значение" с данным ключом, с помощью метода *hash()*.

if (buckets[index] != null) { - проверяет, существует ли список в массиве buckets по индексу index. Если списка нет, метод ничего не делает, так как пара "ключ-значение" с данным ключом не найдена.

for (Entry<K, V> entry : buckets[index]) { - проходит по всем входам (парам "ключ-значение") в списке, связанном с индексом *index*.

if (entry.getKey().equals(key)) { - проверяет, существует ли пара "ключ-значение" с данным ключом в списке. Если пара найдена, она удаляется из списка с помощью метода *remove()*, и счетчик количества элементов в хеш-таблице уменьшается на 1.

Если метод не находит пару "ключ-значение" с данным ключом в списке, он ничего не делает.

```
public void remove(K key) {
    int index = hash(key);
    if (buckets[index] != null) {
        for (Entry<K, V> entry : buckets[index]) {
            if (entry.getKey().equals(key)) {
                buckets[index].remove(entry);
                size--;
                return;
            }
        }
    }
}
```

Рис. 7 Метод remove()

Далее создаем методы:

public int size() { - возвращает текущее количество элементов в хеш-таблице;

public boolean isEmpty() { - проверяет, пуста ли хеш-таблица. Он возвращает true, если текущее количество элементов в хеш-таблице равно 0 (то есть, если хеш-таблица пуста), и false в противном случае.

```java
public int size() {
    return size;
}

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

Рис. 8 Методы size() и isEmpty()

Проверим работу HashTable:

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

    System.out.println("Таблица пустая - " + table.isEmpty());

    table.put(key:101, value:"Ivan");
    table.put(key:305, value:"Petr");
    table.put(key:555, value:"Anna");

    System.out.println("Размер таблицы - " + table.size());

    System.out.println("Получим пользователя 101 - " + table.get(key:101));
    System.out.println("Получим пользователя 305 - " + table.get(key:305));
    System.out.println("Получим пользователя 555 - " + table.get(key:555));
    System.out.println("Получим пользователя 770 - " + table.get(key:770));

    table.put(key:101, value:"Nikolay");

    System.out.println("Обновили ник пользователя 101 - " + table.get(key:101));

    table.remove(key:305);
    System.out.println("Размер таблицы после удаления элемента - " + table.size());
    System.out.println("Номера 305 теперь нет - " + table.get(key:305));

    table.remove(key:101);
    table.remove(key:555);
    System.out.println("Таблица пустая - " + table.isEmpty());
}
```

Рис. 9 Метод main()

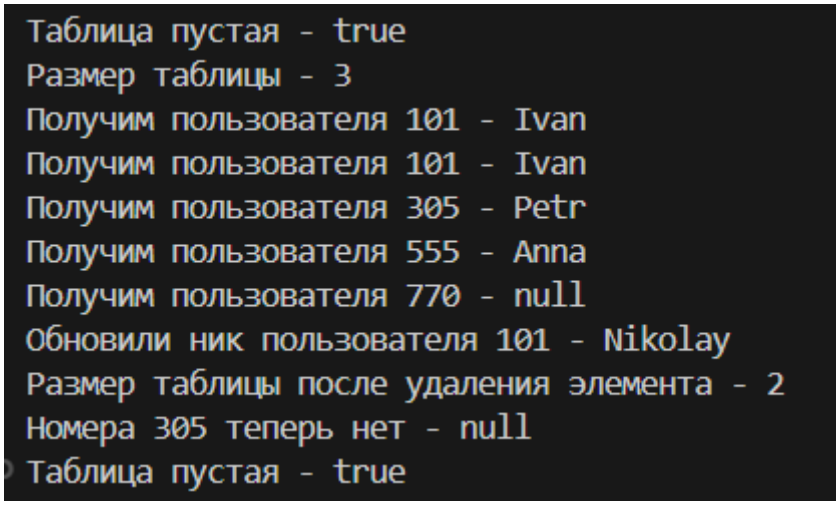


Рис. 10 Вывод метода main()

Задание 2

Создадим класс Orders, который представляет собой класс для управления заказами и содержит вложенный класс Order.

Класс *Order* представляет заказ, содержащий дату, список продуктов и статус заказа. Описание полей и методов класса *Order*:

date (private String) - дата заказа.

products (private List<String>) - список продуктов в заказе. Инициализируется новым экземпляром *ArrayList.*

status (private String) - текущий статус заказа.

Order(String date, List<String> products, String status) - конструктор, который инициализирует поля date, products и status переданными значениями.

setDate(String date) - устанавливает новое значение поля date

getDate() - возвращает текущее значение поля date

setList(List<String> products) - устанавливает новый список товаров заказа

getList() - возвращает текущий список товаров

setStatus(String status) - метод, который устанавливает новое значение поля status.

getStatus() - возвращает текущее значение поля status

info() - метод, который возвращает строковое представление заказа в формате "дата список_продуктов статус".

```
public class Orders {
    protected class Order {
        private String date;
        private List<String> products = new ArrayList<>();
        private String status;

        public Order(String date, List<String> products, String status) {
            this.date = date;
            this.products = products;
            this.status = status;
        }

        protected void setDate(String date) {
            this.date = date;
        }

        protected String getDate() {
            return date;
        }

        protected void setList(List<String> products) {
            this.products = products;
        }

        protected String getList() {
            return products.toString();
        }

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

        protected String getStatus(){
            return status;
        }

        protected String info() {
            return "Дата заказа: " + date + " || Содержание заказа: " + products + " || Статус заказа: " + status;
        }
```

Рис. 11 Вложенный класс Order

Методы класса *Orders* используются для управления заказами, хранящимися в карте *orders* типа *HashMap<Integer, Order>*. Каждый заказ

представлен экземпляром вложенного класса *Order* и имеет уникальный номер заказа в качестве ключа в карте. Методы класса *Orders* выполняют следующие действия:

addOrder(int number, Order order) - добавляет заказ в карту *orders* по указанному номеру заказа. Если заказ с таким же номером уже существует в карте, он будет перезаписан новым заказом.

getOrder(int number) - получает заказ из карты *orders* по указанному номеру заказа и возвращает строковое представление заказа с помощью метода *info()* вложенного класса *Order*. Если заказа с таким номером не существует в карте, метод возвращает *null*.

getOrderDate(int number) - возвращает дату заказа с указанным номером. Если заказа с данным номером нет, метод возвращает сообщение "Нет заказа с номером [*number*]".

setOrderDate(int number, String date) - получает заказ из карты *orders* по указанному номеру заказа, устанавливает новое значение поля *date* заказа с помощью метода *setDate()* вложенного класса *Order*, и добавляет обновленный заказ обратно в карту *orders*. Если заказа с таким номером не существует в карте, метод ничего не делает.

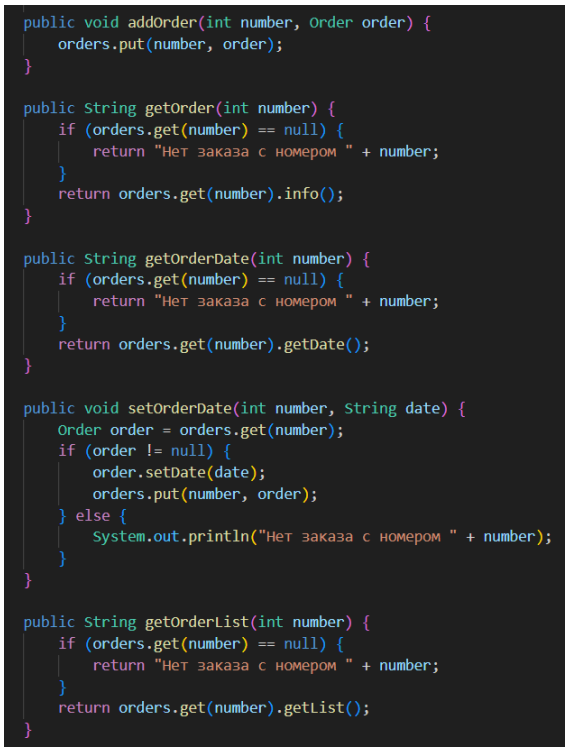
getOrderList(int number) - этот метод возвращает строковое представление списка товаров заказа с указанным номером. Если заказа с данным номером нет, метод возвращает сообщение "Нет заказа с номером [*number*]".

setOrderList(int number, List<String> products) - получает заказ из карты *orders* по указанному номеру заказа, устанавливает новое значение поля *products* заказа с помощью метода *setList()* вложенного класса *Order*, и добавляет обновленный заказ обратно в карту *orders*. Если заказа с таким номером не существует в карте, метод ничего не делает.

getOrderStatus(int number) - этот метод возвращает текущий статус заказа с указанным номером. Если заказа с данным номером нет, метод возвращает сообщение "Нет заказа с номером [*number*]".

setOrderStatus(int number, String status) - получает заказ из карты *orders* по указанному номеру заказа, устанавливает новое значение поля status заказа с помощью метода *setStatus()* вложенного класса *Order*, и добавляет обновленный заказ обратно в карту *orders*. Если заказа с таким номером не существует в карте, метод ничего не делает.

removeOrder(int number) - удаляет заказ из карты *orders* по указанному номеру заказа. Если заказа с таким номером не существует в карте, метод ничего не делает.



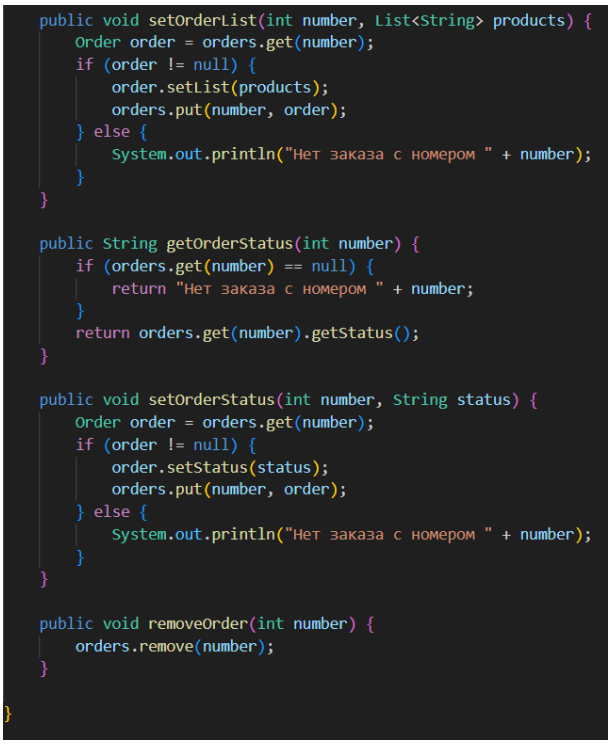
```
public void addOrder(int number, Order order) {
    orders.put(number, order);
}

public String getOrder(int number) {
    if (orders.get(number) == null) {
        return "Нет заказа с номером " + number;
    }
    return orders.get(number).info();
}

public String getOrderDate(int number) {
    if (orders.get(number) == null) {
        return "Нет заказа с номером " + number;
    }
    return orders.get(number).getDate();
}

public void setOrderDate(int number, String date) {
    Order order = orders.get(number);
    if (order != null) {
        order.setDate(date);
        orders.put(number, order);
    } else {
        System.out.println("Нет заказа с номером " + number);
    }
}

public String getOrderList(int number) {
    if (orders.get(number) == null) {
        return "Нет заказа с номером " + number;
    }
    return orders.get(number).getList();
}
```



```
public void setOrderList(int number, List<String> products) {
    Order order = orders.get(number);
    if (order != null) {
        order.setList(products);
        orders.put(number, order);
    } else {
        System.out.println("Нет заказа с номером " + number);
    }
}

public String getOrderStatus(int number) {
    if (orders.get(number) == null) {
        return "Нет заказа с номером " + number;
    }
    return orders.get(number).getStatus();
}

public void setOrderStatus(int number, String status) {
    Order order = orders.get(number);
    if (order != null) {
        order.setStatus(status);
        orders.put(number, order);
    } else {
        System.out.println("Нет заказа с номером " + number);
    }
}

public void removeOrder(int number) {
    orders.remove(number);
}
}
```

Рис. 10, 11 Методы класса Orders

Проверим работу Orders:

```
public static void main(String[] args){
    List<String> products = new ArrayList<>();
    products.add(e:"Apple");
    products.add(e:"Banana");
    products.add(e:"Cherry");

    List<String> products2 = new ArrayList<>();
    products2.add(e:"Laptop");
    products2.add(e:"Mouse");
    products2.add(e:"Screen");

    List<String> products3 = new ArrayList<>();
    products3.add(e:"Jacket");
    products3.add(e:"Cap");
    products3.add(e:"scarf");


    Orders orders = new Orders();

    orders.addOrder(number:123, orders.new Order(date:"11.10.2024", products, status:"выполнен"));
    orders.addOrder(number:215, orders.new Order(date:"15.10.2024", products2, status:"активный"));
    orders.addOrder(number:347, orders.new Order(date:"25.10.2024", products3, status:"активный"));

    System.out.println("Выведем заказ под номером 123 - " + orders.getOrder(number:123));
    orders.removeOrder(number:123);
    System.out.println("Заказ под номером 123 удален - " + orders.getOrder(number:123));
    System.out.println("Статус заказа 215 до изменения - " + orders.getOrder(number:215));
    orders.setOrderStatus(number:215, status:"выполнен");
    System.out.println("Статус заказа 215 после изменения - " + orders.getOrderStatus(number:215));
    orders.setOrderDate(number:123, date:"12.12.2012");
    System.out.println("Попытаемся получить статус удаленного заказа - " + orders.getOrderStatus(number:123));
    System.out.println("Дата заказа 347 до изменения - " + orders.getOrderDate(number:347));
    orders.setOrderDate(number:347, date:"23.10.2024");
    System.out.println("Дата заказа 347 после изменения - " + orders.getOrderDate(number:347));
    System.out.println("Список товаров заказа 347 - " + orders.getOrderList(number:347));
    orders.setOrderList(number:347, products);
    System.out.println("Список товаров заказа 347 после изменения - " + orders.getOrderList(number:347));
}
```

Рис. 11 Метод main()

```
Выведем заказ под номером 123 - Дата заказа: 11.10.2024 || Содержание заказа: [Apple, Banana, Cherry] || Статус заказа: выполнен
Заказ под номером 123 удален - Нет заказа с номером 123
Статус заказа 215 до изменения - Дата заказа: 15.10.2024 || Содержание заказа: [Laptop, Mouse, Screen] || Статус заказа: активный
Статус заказа 215 после изменения - выполнен
Нет заказа с номером 123
Попытаемся получить статус удаленного заказа - Нет заказа с номером 123
Дата заказа 347 до изменения - 25.10.2024
Дата заказа 347 после изменения - 23.10.2024
Список товаров заказа 347 - [Jacket, Cap, scarf]
Список товаров заказа 347 после изменения - [Apple, Banana, Cherry]
```

Рис. 12 Вывод метода main()

Вывод

Мы разобрались с тем, как работает встроенный класс HashMap и на основе этих знаний создали свой класс HashTable, в котором смогли успешно реализовать необходимый функционал.