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»

Лабораторная работа № 3

Хэш-таблица

Выполнил: Студент группы

БВТ2303

Ситникова Дарья

Вариант 3

Москва

2024

Цель работы: Изучить и реализовать хэш-таблицу с использованием метода цепочек для хранения пар «ключ-значение», а также освоить работу с встроенным классом HashMap в Java.

Задачи:

1. Создать класс HashTable, который будет реализовывать хэш-таблицу с помощью метода цепочек.
2. Работа с встроенным классом HashMap. Реализовать хэш-таблицу для хранения информации о заказах в интернет-магазине.

Краткая теория:

Хэш-таблица — это способ хранения данных, который позволяет быстро находить значения по уникальным ключам.

Хэш-таблица — это структура, где мы храним пары «ключ-значение». Ключ — это уникальный идентификатор, а значение — это информация, которую мы хотим сохранить.

Чтобы найти, где хранится значение, мы используем хэш-функцию. Эта функция берет ключ и преобразует его в число, которое указывает на место (индекс) в массиве, где хранится значение. Например, если у нас есть пары:

- «apple» — 5
- «banana» — 3
- «orange» — 7

Хэш-функция преобразует ключи в индексы массива, и мы можем сохранить значения по этим индексам.

Пример: Допустим, после обработки хэш-функцией у нас получается следующее:

- Индекс 2: («banana», 3)
- Индекс 3: («orange», 7)

- Индекс 5: («apple», 5)

Это значит, что значение 3 хранится по индексу 2, значение 7 — по индексу 3, а значение 5 — по индексу 5.

Коллизия. Иногда два разных ключа могут преобразоваться в один и тот же индекс. Это называется коллизией. Например, если «pear» также попадает в индекс 2, где уже есть «banana», то мы должны как-то решить эту проблему.

Как решать коллизии? Один из способов — метод цепочек. Это значит, что каждый индекс массива может хранить не одно значение, а список значений. Если происходит коллизия, новое значение добавляется в конец списка. Например, если мы добавим пару («pear», 2), то индекс 2 будет выглядеть так:

- Индекс 2: («banana», 3) -> («pear», 2)

В Java хэш-таблицы реализованы в классах HashMap и Hashtable. Оба класса позволяют хранить пары «ключ-значение», но HashMap не защищен от одновременного доступа из разных потоков, а Hashtable — защищен.

Ход работы

Задание 1.

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

```
import java.util.LinkedList; // Импортируем класс LinkedList для использования связанного списка, который будет использоваться для хранения записей в хэш-таблице

// Определяем класс HashTable с обобщенными типами K (ключ) и V (значение)
public class HashTable<K, V> {

    // Внутренний класс Entry для хранения пары "ключ-значение"
    private static class Entry<K, V> {
        private K key; // Поле для хранения ключа
        private V value; // Поле для хранения значения
```

```
        // Конструктор класса Entry, который принимает ключ и значение
        public Entry(K key, V value) {
            this.key = key; // Инициализируем поле ключа
            this.value = value; // Инициализируем поле значения
        }

        // Метод для получения ключа
        public K getKey() {
            return key; // Возвращаем ключ
        }

        // Метод для получения значения
        public V getValue() {
            return value; // Возвращаем значение
        }

        // Метод для установки нового значения
        public void setValue(V value) {
            this.value = value; // Обновляем значение
        }
    }

    // Массив, который будет хранить списки пар "ключ-значение"
    private LinkedList<Entry<K, V>>[] table;
    private int size; // Переменная для хранения количества элементов в хэш-
таблице

    // Конструктор класса HashTable, который принимает размер массива
    @SuppressWarnings("unchecked") // Отключаем предупреждения компилятора о
необобщенных типах
    public HashTable(int capacity) {
        table = new LinkedList[capacity]; // Инициализируем массив списков с
заданной емкостью
        size = 0; // Устанавливаем начальный размер в 0
    }

    // Метод для вычисления индекса в массиве на основе ключа
    private int hash(K key) {
        return Math.abs(key.hashCode()) % table.length; // Возвращаем индекс,
используя хэш-код ключа
    }
```

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

```
// Метод для добавления пары "ключ-значение" в хэш-таблицу
    public void put(K key, V value) {
        int index = hash(key); // Вычисляем индекс для данного ключа
```

```
        if (table[index] == null) { // Проверяем, существует ли уже список по
этому индексу
            table[index] = new LinkedList<>(); // Если нет, создаем новый
связанный список
        }
        // Проходим по всем элементам списка, чтобы проверить, существует ли уже
такой ключ
        for (Entry<K, V> entry : table[index]) {
            if (entry.getKey().equals(key)) { // Если ключ найден
                entry.setValue(value); // Обновляем значение
                return; // Выходим из метода
            }
        }
        // Если ключ не найден, добавляем новую пару "ключ-значение" в конец
списка
        table[index].add(new Entry<>(key, value));
        size++; // Увеличиваем размер хэш-таблицы
    }

    // Метод для получения значения по ключу
    public V get(K key) {
        int index = hash(key); // Вычисляем индекс для данного ключа
        if (table[index] != null) { // Проверяем, существует ли список по этому
индексу
            // Проходим по всем элементам списка, чтобы найти нужный ключ
            for (Entry<K, V> entry : table[index]) {
                if (entry.getKey().equals(key)) { // Если ключ найден
                    return entry.getValue(); // Возвращаем соответствующее
значение
                }
            }
        }
        return null; // Если ключ не найден, возвращаем null
    }

    // Метод для удаления пары "ключ-значение" по ключу
    public void remove(K key) {
        int index = hash(key); // Вычисляем индекс для данного ключа
        if (table[index] != null) { // Проверяем, существует ли список по этому
индексу
            // Проходим по всем элементам списка, чтобы найти нужный ключ
            for (Entry<K, V> entry : table[index]) {
                if (entry.getKey().equals(key)) { // Если ключ найден
                    table[index].remove(entry); // Удаляем элемент из списка
                    size--; // Уменьшаем размер хэш-таблицы
                    return; // Выходим из метода
                }
            }
        }
    }
```

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

```
    // Метод для получения текущего количества элементов в хэш-таблице
    public int size() {
        return size; // Возвращаем размер
    }

    // Метод для проверки, пуста ли хэш-таблица
    public boolean isEmpty() {
        return size == 0; // Возвращаем true, если размер равен 0, иначе false
    }
}
```

Добавляем метод main в класс MainHashTable для запуска программы и ввода значений.

```
public class MainHashTable {
    public static void main(String[] args) {
        HashTable<String, Integer> hashTable = new HashTable<>(10);
        hashTable.put("apple", 5); // Добавление пары ключ-значение
        hashTable.put("banana", 3);
        hashTable.put("orange", 7);

        System.out.println("HashTable: " + hashTable); // Вывод текущего
состояния хеш-таблицы
        System.out.println("Get apple: " + hashTable.get("apple")); // Получение
значения по ключу "apple" и вывод его на экран
        System.out.println("Size: " + hashTable.size()); // Вывод текущего
размера хеш-таблицы

        hashTable.remove("banana"); // Удаление элемента с ключом "banana" из
хеш-таблицы
        System.out.println("After removing banana: " + hashTable); // Вывод
состояния хеш-таблицы после удаления "banana"
        System.out.println("Is empty: " + hashTable.isEmpty()); // Проверка,
пуста ли хеш-таблица, и вывод результата на экран
    }
}
```

Вывод в терменале:

```
PS C:\Users\manul\Desktop\ITiP\Lab 3> java MainHashTable.java
HashTable: HashTable@6c130c45
Get apple: 5
Size: 3
After removing banana: HashTable@6c130c45
Is empty: false
PS C:\Users\manul\Desktop\ITiP\Lab 3>
```

Код создает хэштаблицу и выполняет несколько операций с ней, но когда выводится объект hashTable, он отображает не содержимое хэштаблицы, а стандартное представление объекта Java, которое включает имя класса и хэш-код. Чтобы исправить это и получить более информативный вывод, нужно переопределить метод toString() в классе HashTable.

Добавим метод toString() в класс HashTable, чтобы он возвращал строковое представление содержимого хэштаблицы.

Измененный код:

```
    @Override
    public String toString() {
        // Создаем новый объект StringBuilder для построения строки
        StringBuilder sb = new StringBuilder();
        //Добавляем открывающую фигурную скобку для представления хэштаблицы
        sb.append("{");
        // Проходим по всем "ведрам" (спискам) в массиве table
        for (LinkedList<Entry<K, V>> bucket : table) {
            // Проверяем, не является ли текущее "ведро" null
            if (bucket != null) {
                // Если "ведро" не пустое, проходим по всем элементам в нем
                for (Entry<K, V> entry : bucket) {
                    //Добавляем строку, представляющую пару "ключ-значение", в
StringBuilder
                    sb.append(entry.getKey()).append(":
").append(entry.getValue()).append(", ");
                }
            }
        }

       //Удаляем последнюю запятую и пробел, если они есть

        if (sb.length() > 1) {
            // Устанавливаем длину StringBuilder на 2 символа меньше, удаляя
последние два символа
            sb.setLength(sb.length() - 2);
        }
        // Добавляем закрывающую фигурную скобку
        sb.append("}");
        // Преобразуем StringBuilder в строку и возвращаем ее
        return sb.toString();
    }
}
```

Вывод в терменале с измененным кодом:

```
PS C:\Users\manul\Desktop\ITiP\Lab 3> java MainHashTable.java
HashTable: {apple: 5, orange: 7, banana: 3}
Get apple: 5
Size: 3
After removing banana: {apple: 5, orange: 7}
Is empty: false
PS C:\Users\manul\Desktop\ITiP\Lab 3>
```

Задание 2.

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.

Создаем два класса Order и OrderManager. Класс Order будет представлять информацию о заказе, а класс OrderManager будет использовать встроенный класс HashMap для управления заказами.

Класс Order будет содержать поля для даты заказа, списка товаров и статуса заказа. Мы также добавим методы для доступа к этим полям.

```
import java.util.Date;
import java.util.List;

public class Order {
    private String orderNumber; // Номер заказа
    private Date orderDate; // Дата заказа
    private List<String> items; // Список товаров
    private String status; // Статус заказа

    // Конструктор класса Order
    public Order(String orderNumber, Date orderDate, List<String> items, String
status) {
        this.orderNumber = orderNumber; // Инициализируем номер заказа
        this.orderDate = orderDate; // Инициализируем дату заказа
        this.items = items; // Инициализируем список товаров
        this.status = status; // Инициализируем статус заказа
    }
```

```
    // Метод для получения номера заказа
    public String getOrderNumber() {
        return orderNumber;
    }

    // Метод для получения даты заказа
    public Date getOrderDate() {
        return orderDate;
    }

    // Метод для получения списка товаров
    public List<String> getItems() {
        return items;
    }

    // Метод для получения статуса заказа
    public String getStatus() {
        return status;
    }

    // Метод для изменения статуса заказа
    public void setStatus(String status) {
        this.status = status; // Обновляем статус заказа
    }

    // Переопределяем метод toString для удобного отображения информации о заказе
    @Override
    public String toString() {
        return "Order{" +
                "orderNumber='" + orderNumber + '\'' +
                ", orderDate=" + orderDate +
                ", items=" + items +
                ", status='" + status + '\'' +
                '}';
    }
}
```

Класс OrderManager будет использовать HashMap для хранения заказов, где ключом будет номер заказа, а значением — объект Order.

```
import java.util.HashMap;

public class OrderManager {
    private HashMap<String, Order> orders; // Хранит заказы, ключ - номер заказа

    // Конструктор класса OrderManager
    public OrderManager() {
        orders = new HashMap<>(); // Инициализируем HashMap для хранения заказов
```

```
    }

    // Метод для добавления нового заказа
    public void addOrder(Order order) {
        orders.put(order.getOrderNumber(), order); // Вставляем заказ в HashMap
    }

    // Метод для получения заказа по номеру
    public Order getOrder(String orderNumber) {
        return orders.get(orderNumber); // Возвращаем заказ по номеру
    }

    // Метод для удаления заказа по номеру
    public void removeOrder(String orderNumber) {
        orders.remove(orderNumber); // Удаляем заказ из HashMap
    }

    // Метод для изменения статуса заказа
    public void updateOrderStatus(String orderNumber, String newStatus) {
        Order order = orders.get(orderNumber); // Получаем заказ по номеру
        if (order != null) {
            order.setStatus(newStatus); // Обновляем статус заказа
        }
    }

    // Метод для отображения всех заказов
    public void displayOrders() {
        for (Order order : orders.values()) {
            System.out.println(order); // Выводим информацию о каждом заказе
        }
    }
}
```

Класс с методом Main, чтобы протестировать реализацию.

```
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        OrderManager orderManager = new OrderManager(); // Создаем экземпляр
OrderManager

        // Создаем несколько заказов
        Order order1 = new Order("001", new Date(), new
ArrayList<>(List.of("Товар1", "Товар2")), "В обработке");
        Order order2 = new Order("002", new Date(), new
ArrayList<>(List.of("Товар3")), "Доставлен");
```

```
        // Добавляем заказы в менеджер
        orderManager.addOrder(order1);
        orderManager.addOrder(order2);

        // Отображаем все заказы
        System.out.println("Все заказы:");
        orderManager.displayOrders();

        // Получаем заказ по номеру
        System.out.println("\nПолучаем заказ 001:");
        System.out.println(orderManager.getOrder("001"));

        // Изменяем статус заказа
        orderManager.updateOrderStatus("001", "Доставлен");
        System.out.println("\nПосле изменения статуса заказа 001:");
        System.out.println(orderManager.getOrder("001"));

        // Удаляем заказ
        orderManager.removeOrder("002");
        System.out.println("\nПосле удаления заказа 002:");
        orderManager.displayOrders();
    }
}
```

Терминал:

```
PS C:\Users\manul\Desktop\ITiP\Lab 3> java Main.java
Все заказы:
Order{orderNumber='001', orderDate=Sun Oct 06 21:46:55 MSK 2024, items=[Товар1, Товар2], status='В об
работке'}
Order{orderNumber='002', orderDate=Sun Oct 06 21:46:55 MSK 2024, items=[Товар3], status='Доставлен'}

Получаем заказ 001:
Order{orderNumber='001', orderDate=Sun Oct 06 21:46:55 MSK 2024, items=[Товар1, Товар2], status='В об
работке'}

После изменения статуса заказа 001:
Order{orderNumber='001', orderDate=Sun Oct 06 21:46:55 MSK 2024, items=[Товар1, Товар2], status='Дост
авлен'}

После удаления заказа 002:
Order{orderNumber='001', orderDate=Sun Oct 06 21:46:55 MSK 2024, items=[Товар1, Товар2], status='Дост
авлен'}
PS C:\Users\manul\Desktop\ITiP\Lab 3>
```

Вывод: Было изучено создание и реализация хэш-таблицы с использованием метода цепочек, а также работа с встроенным классом HashMap в Java. Освоены основные операции, такие как вставка, поиск и удаление элементов.

Приложение:

GitHub: https://github.com/Daria0w0/ITiP

1. HashTable.java

```
import java.util.LinkedList;

public class HashTable<K, V> {
    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;
        }

        public void setKey(K key) {
            this.key = key;
        }
    }

    private LinkedList<Entry<K, V>>[] table;
    private int size;

    @SuppressWarnings("unchecked")
    public HashTable(int capacity) {
        table = new LinkedList[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<>();
        }
        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;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("{");
        for (LinkedList<Entry<K, V>> bucket : table) {
```

```
            if (bucket != null) {
                for (Entry<K, V> entry : bucket) {
                    sb.append(entry.getKey()).append(":
").append(entry.getValue()).append(", ");
                }
            }
        }
        if (sb.length() > 1) {
            sb.setLength(sb.length() - 2);
        }
        sb.append("}");
        return sb.toString();
    }
}
```

2. MainHashTable.java

```
public class MainHashTable {
    public static void main(String[] args) {
        HashTable<String, Integer> hashTable = new HashTable<>(10);
        hashTable.put("apple", 5);
        hashTable.put("banana", 3);
        hashTable.put("orange", 7);

        System.out.println("HashTable: " + hashTable);
        System.out.println("Get apple: " + hashTable.get("apple"));
        System.out.println("Size: " + hashTable.size());

        hashTable.remove("banana");
        System.out.println("After removing banana: " + hashTable);
        System.out.println("Is empty: " + hashTable.isEmpty());
    }
}
```

3. Order.java

```
import java.util.Date;
import java.util.List;

public class Order {
    private String orderNumber;
    private Date orderDate;
    private List<String> items;
    private String status;

    public Order(String orderNumber, Date orderDate, List<String> items, String
status) {
        this.orderNumber = orderNumber;
        this.orderDate = orderDate;
```

```
        this.items = items;
        this.status = status;
    }

    public String getOrderNumber() {
        return orderNumber;
    }

    public Date getOrderDate() {
        return orderDate;
    }

    public List<String> getItems() {
        return items;
    }

    public String getStatus() {
        return status;
    }

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

    @Override
    public String toString() {
        return "Order{" +
                "orderNumber='" + orderNumber + '\'' +
                ", orderDate=" + orderDate +
                ", items=" + items +
                ", status='" + status + '\'' +
                '}';
    }
}
```

4. OrderManager.java

```
import java.util.HashMap;

public class OrderManager {
    private HashMap<String, Order> orders;

    public OrderManager() {
        orders = new HashMap<>();
    }

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

```
    public Order getOrder(String orderNumber) {
        return orders.get(orderNumber);
    }

    public void removeOrder(String orderNumber) {
        orders.remove(orderNumber);
    }

    public void updateOrderStatus(String orderNumber, String newStatus) {
        Order order = orders.get(orderNumber);
        if (order != null) {
            order.setStatus(newStatus);
        }
    }

    public void displayOrders() {
        for (Order order : orders.values()) {
            System.out.println(order);
        }
    }
}
```

5. Main.java

```
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        OrderManager orderManager = new OrderManager();

        Order order1 = new Order("001", new Date(), new
ArrayList<>(List.of("Товар1", "Товар2")), "В обработке");
        Order order2 = new Order("002", new Date(), new
ArrayList<>(List.of("Товар3")), "Доставлен");

        orderManager.addOrder(order1);
        orderManager.addOrder(order2);

        System.out.println("Все заказы:");
        orderManager.displayOrders();

        System.out.println("\nПолучаем заказ 001:");
        System.out.println(orderManager.getOrder("001"));

        orderManager.updateOrderStatus("001", "Доставлен");
        System.out.println("\nПосле изменения статуса заказа 001:");
        System.out.println(orderManager.getOrder("001"));
```

```
        orderManager.removeOrder("002");
        System.out.println("\nПосле удаления заказа 002:");
        orderManager.displayOrders();
    }
}
```