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 «Программная инженерия»

Отчет по лабораторной работе № 3.

по дисциплине «Информационные технологии и программирование»

по разделу:

«3. Класс Object.

Работас хэш-таблицами»

Выполнила: студентка группы БПИ2401 Рябова Екатерина

Проверил: Харрасов Камиль Раисович

Москва

2025

Содержание

Цель работы:

Познакомиться с хэш-таблицами в Java, изучить и практически реализовать хэш-таблицу с методом цепочек. Ознакомиться с методами класса Object.

Ход работы:

Задание 1 для выполнения лабораторной работы:

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.*

Рассмотрим какие поля у нас должны быть у класса HashTable: размер таблицы, кол-во элементов в ней и сама таблица – массив из связного списка пар вида (ключ, значение). Так как у нас абстрактная таблица, то и тип данных у ключа и значения может быть любым. Поэтому обзовем их типы K и V, а тип

данных, в котором записана пара ключ-значение – Entry. Соответственно создадим класс Entry, вложенный в класс HashTable с полями key и value. Создадим для него геттеры – получение ключа и значения и сеттер для значения (ключ менять нельзя). Полный код для этого класса представлен на листинге 1.1.

Листинг 1.1 вложенный класс Entry

```
    private 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; }
    }
```

Для класса Hashtable пропишем конструктор, напишем функцию хэширования – то есть превращения ключа в индекс в списке. Для этого воспользуемся встроенной функцией хэширования, возьмем ее модуль и остаток при делении на размер таблицы. Воспользуемся листингом 3.3 из методички для написания метода put. Принцип работы этого метода заключается в том, что она сначала с помощью функции хэширования получает индекс, где МОГ БЫ находится объект, затем мы проверяем, что лежит в списке по этому индексу. Если ничего – то создаем пустой связанный список из энтриешек – пар ключ-значение. Затем по этому списку (существующему или только что созданному) проходимся циклом. У каждого entry вытаскиваем ключ уже написанным методом getKey() и сравниваем его с нашим ключом. Если где-то совпало, значит, нам нужно просто поменять значение методом setValue. Если мы прошли по всему списку и ничего не нашли (и, соответственно, не ретёрнулись), то просто добавляем в список наш

объект типа Entry. Не забываем увеличить переменную, которая считает кол-во объектов в таблице.

Аналогично пишем метод remove – только если в цикле мы нашли пару с тем же ключом, то мы удаляем эту entry из списка и уменьшаем счетчик объектов в таблице, если в цикле ничего не нашлось, то ничего не делаем. Так же, метод get, только если ничего не нашлось, то возвращаем null.

Пропишем еще два требуемых метода – size и isEmpy, работающие с переменной, считающей кол-во объектов. См. полностью на листинге 1.2.

Листинг 1.2 полный код для класса HashTable

```
package hashtable;

import java.util.LinkedList;

public class HashTable<K, V> {
    private 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; }
    }

    private LinkedList<Entry<K, V>>[] table;
    private int size = 0;
    final private int tableSize = 10;

    public HashTable() {
        table = new LinkedList[tableSize];
    }

    public int hash(K key){
        return Math.abs(key.hashCode()) % tableSize;
    }

    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 void remove(K key){
        int index = hash(key);
        if (table[index] == null) {
            return;
        }

        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;
    }

    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;
    }
}
```

Напишем класс Main, тестирующий работу с этим классом. Создадим две хэш-таблицы: книга-рейтинг и книга-автор, где книга – ключ, а рейтинг (в этом примере целое число) и автор (строка) – значения. Важно, что в одной хэш-таблице типы данных для ключа и значения, хоть и не прописаны, фиксируются при создании. Полный код для тестирования представлен на листинге 1.3.

Листинг 1.3 код класса Main для тестирования хэш-таблиц.

```
package hashtable;

public class Main {
 public static void main(String[] args) {
  HashTable<String, Integer> bookRatings = new HashTable<>();

  bookRatings.put("Война и мир", 6);
  bookRatings.put("Властелин Колец", 9);
  bookRatings.put("Медвежий Угол", 10);
  bookRatings.put("1984", 8);
  bookRatings.put("Гарри Поттер", 9);

  System.out.println("Рейтинг 'Медвежий Угол': " + bookRatings.get("Медвежий
Угол"));
  System.out.println("Рейтинг 'Гарри Поттер': " + bookRatings.get("Гарри
Поттер"));
  System.out.println("Всего книг: " + bookRatings.size());

  System.out.println("Старый рейтинг 'Гарри Поттер': " + bookRatings.get("Гарри
Поттер"));
  bookRatings.put("Гарри Поттер", 10);
  System.out.println("Новый рейтинг 'Гарри Поттер': " + bookRatings.get("Гарри
Поттер"));

  bookRatings.remove("1984");
  System.out.println("'1984' после удаления: " + bookRatings.get("1984"));
  System.out.println("Книг после удаления: " + bookRatings.size());

  HashTable<String, String> bookAuthors = new HashTable<>();
```

```
  bookAuthors.put("Война и мир", "Лев Толстой");
  bookAuthors.put("Властелин Колец", "Джон Рональд Руэл Толкин");
  bookAuthors.put("Медвежий Угол", "Фредрик Бакман");
  bookAuthors.put("1984", "Джордж Оруэлл");

  System.out.println("Автор 'Властелин Колец': " + bookAuthors.get("Властелин
Колец"));
  System.out.println("Автор '1984': " + bookAuthors.get("1984"));

  System.out.println("Автор 'Гарри Поттер': " + bookAuthors.get("Гарри Поттер"));
 }
}
```

Результат работы Main.java представлен на рис. 1:

Рисунок 1. Тестирование работы написанной хэш-таблицы.

```
PS C:\Users\user\.vscode\mtuci\java\labs\thirdlab> java hashtable.Main
Рейтинг 'Медвежий Угол': 10
Рейтинг 'Гарри Поттер': 9
Всего книг: 5
Старый рейтинг 'Гарри Поттер': 9
Новый рейтинг 'Гарри Поттер': 10
'1984' после удаления: null
Книг после удаления: 4
Автор 'Властелин Колец': Джон Рональд Руэл Толкин
Автор '1984': Джордж Оруэлл
Автор 'Гарри Поттер': null
```

Задание 2 для выполнения лабораторной работы:

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.

Для начала напишем класс Order с полями «дата заказа» (создается автоматически как текущая дата при инициализации), список товаров и строка-статус. В конструкторе прописываем дату, копируем список товаров, поступивший на вход и автоматически ставим статус «В обработке».

Прописываем геттеры (для даты, списка товаров и статуса) и сеттеры для статуса (больше ничего нельзя менять): стандартный сеттер, который меняет статус на то, что задал пользователь, и автоматические, которые меняют статус при его подтверждении, отправлении, доставке и отмене. Во всех них используем уже написанный раннее стандартный сеттер. Все эти методы package-private. См. на листинге 2.1.

Листинг 2.1. Класс Order

```
package hashmap;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

public class Order {
    private LocalDate orderDate;
    private List<String> products;
    private String status;

    public Order(List<String> products){
        this.orderDate = LocalDate.now();
        this.products = new ArrayList<>(products);
        this.status = "В обработке";
    }

    LocalDate getOrderDate() {
        return orderDate;
    }

    List<String> getProducts() {
        return new ArrayList<>(products);
    }

    String getStatus() {
        return status;
    }

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

    void confirm(){
        setStatus("Подтвержден");
    }
```

```
    void send(){
        setStatus("Отправлен");
    }

    void deliver(){
        setStatus("Доставлен");
    }

    void cancel(){
        setStatus("Отменен");
    }
}
```

Теперь напишем класс HashMap, использующий объект Order как значение, а его код – как ключ. Внутри него так же, как и в задании 1 (листинг 1.1) есть вложенный класс Entry – класс-связка для кода и значения, единственное отличие в том, что теперь нам точно известны типы данных – int вместо K и Order вместо V, реализованы те же методы. У внешнего класса есть поля с размерами таблицы и кол-во объектов в карте, а также связный список – сама карта. Напишем конструктор, который создает этот список заданной длины и хэш-функцию, которая преобразует код заказа в индекс в этой карте (проделаем различные математические операции с кодом, следя, чтобы число было неотрицательным и меньше размера карты).

Напишем методы put, get, remove как в задании 1 (листинг 1.2) по такой же логике: хэширование кода в индекс – проверка того, что лежит по этому индексу на пустоту (создание списка из Entry если там ничего нет) – цикл по списку из энтриешек по указанному индексу и поиск среди них такого же ключа. Здесь сравнение происходит с помощью оператора ==, потому что мы знаем, что ключ – это код, то есть, целое число. Далее – создание нового списка или возврат null (в зависимости от метода), если не нашли. Напишем методы получения кол-ва объектов в карте и проверка на пустоту. Продублируем написанные в Order геттеры и сеттеры, но уже как паблик. В каждом из них с помощью метода get получим заказ, если он не null то выполним соответствующий метод для этого заказа (мы в том же пакете), если null –

выведем сообщение об ошибке и вернем тоже null или просто вернемся, зависит от типа метода. Полный код представлен в листинге 2.2.

Листинг 2.2 Код класса HashMap

```
package hashmap;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public class HashMap {
    private class Entry {
        private int key;
        private Order value;

        public Entry(int key, Order value) {
            this.key = key;
            this.value = value;
        }

        public int getKey() { return key; }
        public Order getValue() { return value; }
        public void setValue(Order value) { this.value = value; }
    }
    private LinkedList<Entry>[] table;
    private int size = 0;
    final private int tableSize = 10000;

    public HashMap() {
        table = new LinkedList[tableSize];
    }

    public int hash(int key){
        return ((key + 13) * 47 / (31 + key % 100)) % tableSize;
    }

    public void put(int key, Order value) {
        int index = hash(key);
        if (table[index] == null) {
            table[index] = new LinkedList<Entry> ();
        }

        for (Entry entry : table[index]) {
            if (entry.getKey() == key) {
                entry.setValue(value);
                return;
            }
```

```
        }

        table[index].add(new Entry(key, value));
        size ++;
    }

    public void remove(int key){
        int index = hash(key);
        if (table[index] == null) {
            return;
        }

        for (Entry entry : table[index]) {
            if (entry.getKey() == key) {
                table[index].remove(entry);
                size --;
                return;
            }
        }
    }

    public int size(){
        return size;
    }

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

    public Order get(int key){
        int index = hash(key);
        if (table[index] == null) {
            return null;
        }

        for (Entry entry : table[index]) {
            if (entry.getKey() == key) {
                return entry.getValue();
            }
        }
        return null;
    }

     public LocalDate getOrderDate(int key) {
        Order order = get(key);
        if (order == null) {
            System.out.println("Error, order wasn`t found");
            return null;
        }
        return order.getOrderDate();
```

```
    }

    public List<String> getProducts(int key) {
        Order order = get(key);
        if (order == null) {
            System.out.println("Error, order wasn`t found");
            return null;
        }
        return order.getProducts();
    }

    public String getStatus(int key) {
        Order order = get(key);
        if (order == null) {
            System.out.println("Error, order wasn`t found");
            return null;
        }
        return order.getStatus();
    }

    public void setStatus(int key, String status){
        Order order = get(key);
        if (order == null) {
            System.out.println("Error, order wasn`t found");
            return;
        }
        order.setStatus(status);
    }

    public void confirm(int key){
        Order order = get(key);
        if (order == null) {
            System.out.println("Error, order wasn`t found");
            return;
        }
        order.confirm();
    }

    public void send(int key){
        Order order = get(key);
        if (order == null) {
            System.out.println("Error, order wasn`t found");
            return;
        }
        order.send();
    }

    public void deliver(int key){
        Order order = get(key);
        if (order == null) {
```

```
            System.out.println("Error, order wasn`t found");
            return;
        }
        order.deliver();
    }

    public void cancel(int key){
        Order order = get(key);
        if (order == null) {
            System.out.println("Error, order wasn`t found");
            return;
        }
        order.cancel();
    }
}
```

Затем напишем класс Main, где протестируем работу нашей хэш-карты. Создадим хэш-карту менеджер заказов, создадим три списка товаров и соответственно три заказа. Добавим их в карту. Для каждого заказа выведем какую-то иноформацию. Создадим новый список товаров для заказа 1, обновим его статус. С помощью метода put положим его в карту и выведем о нем информацию. Подробнее см. листинг 2.3.

Листинг 2.3 Код для тестирования.

```
package hashmap;

import java.util.Arrays;
import java.util.List;

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

        List<String> products1 = Arrays.asList("Ноутбук ASUS", "Мышь
беспроводная", "Чехол для ноутбука");
        List<String> products2 = Arrays.asList("Смартфон Samsung", "Защитное
стекло", "Чехол");
        List<String> products3 = Arrays.asList("Наушники JBL", "Коврик для
мыши");

        Order order1 = new Order(products1);
        Order order2 = new Order(products2);
        Order order3 = new Order(products3);

        orderManager.put(1001, order1);
        orderManager.put(1002, order2);
```

```
        orderManager.put(1003, order3);

        System.out.println("Добавлено заказов: " + orderManager.size());

        Order foundOrder = orderManager.get(1001);
        if (foundOrder != null) {
            System.out.println("Заказ 1001 найден: " + foundOrder.getStatus());
        } else {
            System.out.println("Заказ 1001 не найден");
        }

        System.out.println("Дата заказа: " + orderManager.getOrderDate(1002));
        System.out.println("Статус: " + orderManager.getStatus(1002));
        System.out.println("Товары: " + orderManager.getProducts(1002));

        orderManager.confirm(1001);
        System.out.println("Статус заказа 1001: " +
orderManager.getStatus(1001));
        System.out.println("Дата заказа 1001: " +
orderManager.getOrderDate(1001));

        orderManager.send(1002);
        System.out.println("Статус заказа 1002: " +
orderManager.getStatus(1002));

        orderManager.deliver(1003);
        System.out.println("Статус заказа 1003: " +
orderManager.getStatus(1003));

        System.out.println("Поиск заказа 9999: " + orderManager.get(9999));
        orderManager.confirm(9999);
        System.out.println("Статус заказа 9999: " +
orderManager.getStatus(9999));

        System.out.println("Заказов до удаления: " + orderManager.size());
        orderManager.remove(1002);
        System.out.println("Заказов после удаления: " + orderManager.size());
        System.out.println("Поиск удаленного заказа 1002: " +
orderManager.get(1002));

        List<String> newProducts = Arrays.asList("Ноутбук ASUS", "Мышь
беспроводная", "Сумка для ноутбука", "Охлаждающая подставка");
        Order updatedOrder = new Order(newProducts);
        updatedOrder.confirm();

        orderManager.put(1001, updatedOrder);
        System.out.println("Обновленные товары заказа 1001: " +
orderManager.getProducts(1001));
        System.out.println("Статус заказа 1001: " +
orderManager.getStatus(1001));
```

```
        System.out.println("Всего заказов: " + orderManager.size());
    }
}
```

Результат работы Main.java представлен на рис. 2

Рисунок 2. Тестирование хэш-карты.

```
PS C:\Users\user\.vscode\mtuci\java\labs\thirdlab> java hashmap.Main
Добавлено заказов: 3
Заказ 1001 найден: В обработке
Дата заказа: 2025-10-22
Статус: В обработке
Товары: [Смартфон Samsung, Защитное стекло, Чехол]
Статус заказа 1001: Подтвержден
Дата заказа 1001: 2025-10-22
Статус заказа 1002: Отправлен
Статус заказа 1003: Доставлен
Поиск заказа 9999: null
Error, order wasn`t found
Error, order wasn`t found
Статус заказа 9999: null
Заказов до удаления: 3
Заказов после удаления: 2
Поиск удаленного заказа 1002: null
Обновленные товары заказа 1001: [Ноутбук ASUS, Мышь беспроводная, Сумка для ноутбука, Охлаждающая подставка]
Статус заказа 1001: Подтвержден
Всего заказов: 2
```

Работа загружена на гитхаб по ссылке: https://github.com/Kateriabova/Riaboava_Kate_BPI2401_IT2.git

Ответы на контрольные вопросы:

1. **Для чего нужен класс Object?**

Класс Object является базовым классом всей иерархии в Java. Все классы автоматически наследуют от Object, что обеспечивает наличие основных методов (toString, equals, hashCode и др.) у любого объекта.

2. **Для чего нужно переопределять методы equals() и hashCode()?**
   - Чтобы сравнивать объекты по содержимому, а не по ссылкам
   - Для корректной работы с коллекциями (HashMap, HashSet)
   - Чтобы обеспечить консистентность: равные объекты должны иметь одинаковые хэш-коды
3. **Какие есть правила переопределения методов equals() и hashCode()?**
   - Если equals() == true то и hashCode() == true. Отсюда следует, что если hashCode() == false, то и equals() == false
   - При этом, отсюда не следует, что если hashCode() == true то и equals() == true.
   - Методы должны быть согласованы: одинаковые объекты → одинаковый хэш-код
4. **Что делает метод toString()? Почему его часто переопределяют?**

Возвращает строковое представление объекта. Переопределяют для получения читаемой информации вместо стандартного вывода (имя_класса и хэш_код).

5. **Что делает метод finalize()? Почему его использование считается устаревшим?**

Вызывается сборщиком мусора перед удалением объекта. Устарел из-за непредсказуемости работы и лучших альтернатив.

6. **Что такое коллизия?**

Когда разные ключи имеют одинаковый хэш-код и попадают в одну ячейку массива.

7. **Какие есть способы разрешения коллизий?**
   - Метод цепочек (chaining) - связные списки в ячейках
   - Открытая адресация - поиск следующей свободной ячейки
   - Двойное хэширование
8. **Как хранятся данные в хэш-таблице?**

   В массиве, где каждая ячейка может содержать:

   - null
   - Один элемент
   - Связный список элементов (при коллизиях)
9. **Что происходит, если в хэш-таблицу добавить элемент с одинаковым значением ключа?**

Происходит обновление значения для существующего ключа (старое значение заменяется новым).

10. **Что происходит, если в хэш-таблицу добавить элемент с таким же хэш-кодом ключа, но разными исходными значениями?**

Происходит коллизия. В случае метода цепочек пары ключ-значение добавляются в одну ячейку как разные элементы связного списка.

11. **Как изменяется HashMap при достижении порогового значения?**

Происходит рехэширование - увеличение размера массива и перераспределение всех элементов.

Вывод:

В ходе работы мы изучили принципы использования и создания хэш-структур в Java, методы класса Object. Научились писать хэш-таблицы с использованием метода связанных цепочек.