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 «Программная инженерия»

Отчет по лабораторной работе 3

Класс Object. Работа с хэш-таблицами

Выполнила: студентка группы БПИ2401

Садовникова Евгения

Проверил: Харрасов Камиль Раисович

Москва, 2025

Содержание

Цель работы: изучение принципов организации и функционирования хэш-таблиц, а также приобретение практических навыков их реализации и использования в языке Java.

Задания

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.

Пример реализации метода put(key, value) представлен на листинге 1.

Листинг 1. Реализация метода put(key, value)

```
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++;
}
```

Задание 2. Работа с встроенным классом HashMap.

4. Реализация хэш-таблицы для хранения информации о книгах в библиотеке. Ключом будет ISBN книги, а значением — объект класса Book, содержащий информацию о названии, авторе и количестве копий. Необходимо реализовать операции вставки, поиска и удаления книги по ISBN.

Ход работы:

Создадим класс HashTable с вложенным статическим классом Entry (листинг 1). Протестируем его в классе Main (листинг 2)

Листинг 1. class HashTable

```
import java.util.LinkedList;

public class HashTable<K, V> {

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

        Entry(K key, V value) {
            this.key = key;
            this.value = value;
        }

        K getKey() {
            return key;
        }

        /*
         * V getValue() {
         * return value;
         * }
         */

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

        @Override
        public String toString() {
            return "(" + key + ", " + value + ")";
        }
    }

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

    @SuppressWarnings("unchecked")
    public HashTable(int capacity) {
        this.capacity = capacity;
        table = new LinkedList[capacity];
        size = 0;
    }

    public int hash(K key) {
        return Math.abs(key.hashCode() % capacity);
```

```
    }

    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 String get(K key) {
        int index = hash(key);
        if (table[index] == null) {
            return "";
        }
        for (Entry<K, V> entry : table[index]) {
            if (entry.getKey().equals(key)) {
                return entry.toString();
            }
        }
        return "";
    }

    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() {
        if (size == 0) {
            return true;
        }
        /*
         * for (int i = 0; i < capacity; i++) {
         * if (table[i] != null) {
         * for (Entry<K, V> entry : table[i]) {
```

```
             * if (entry.getValue() != null) {
             * return false;
             * }
             * }
             * }
             * }
             */
            return true;

        }
    }
```

Создадим абстрактный класс Goblin с полями: noseLength, height, footSize. Переопределяет методы getMovement() и printDescription(). (листинг 2)

Листинг 2. class Main

```
    public class Main {
        public static void main(String[] args) {
            // Задание 1
            System.out.println("Task 1");
            HashTable<String, Integer> hashTable = new HashTable<>(10);
            System.out.println(hashTable.size());
            System.out.println(hashTable.isEmpty());

            System.out.println();

            hashTable.put("apple", null);
            System.out.println(hashTable.get("apple"));
            System.out.println(hashTable.size());
            System.out.println(hashTable.isEmpty());
            System.out.println();

            hashTable.remove("apple");
            System.out.println(hashTable.size());
            System.out.println(hashTable.isEmpty());
            System.out.println();

            hashTable.put("banana", 3);
            System.out.println(hashTable.get("banana"));

            hashTable.put("pear", 2);
            System.out.println(hashTable.get("pear"));

            hashTable.put("orange", 7);
            System.out.println(hashTable.get("orange"));
            System.out.println(hashTable.size());
            System.out.println(hashTable.isEmpty());
            System.out.println();
```

Создадим класс Book (листинг 3) и класс Library, где реализуем HashMap (листинг 4). Протестируем его в классе Main (листинг 5)

Листинг 3. class Book

```
public class Book {
    private String name;
    private String author;
    private int copies = 0;

    Book(String name, String author, int copies) {
        this.name = name;
        this.author = author;
        this.copies = copies;
    }

    public String getName() {
        return name;
    }

    public String getAuthor() {
        return author;
    }

    public int getCopies() {
        return copies;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public void setCopies(int copies) {
        this.copies = copies;
    }

    @Override
    public String toString() {
        return "Book{'name='" + name + "', author='" + author + "',
copies=" + copies + "}";
    }

}
```

Листинг 4. class HashMap

```
import java.util.HashMap;

public class Library {
    HashMap<String, Book> books;

    public Library() {
        books = new HashMap<>();
    }

    public void add(String isbn, Book book) {
        books.put(isbn, book);
```

```
            System.out.println("The book has been added: " + book);
        }

        public Book search(String isbn) {
            Book book = books.get(isbn);
            if (book != null) {
                System.out.println("A book has been found: " + book);
            } else {
                  System.out.println("Book with isbn " + isbn + " not
found");
            }
            return book;
        }

        public void del(String isbn) {
            Book removed = books.remove(isbn);
            if (removed != null) {
                System.out.println("The book was deleted: " + removed);
            } else {
                  System.out.println("Book with isbn " + isbn + " not
found");
            }
        }

    }
```

Листинг 5. class Main

```
    public class Main {
        public static void main(String[] args) {
            // Задание 2
            System.out.println("Task 2");
            Library library = new Library();

            // Создание книг
            Book book1 = new Book("War and Peace", "Leo Tolstoy", 5);
              Book book2 = new Book("Crime and Punishment", "Fyodor
Dostoevsky", 3);

            library.add("1234", book1);
            library.add("123456", book2);
            library.search("1234");
            library.del("1234");
            library.search("1234");

        }
    }
```

Выходные данные:

Изображены на рисунке 1

Рисунок 1. Результат программы Main.java

Контрольные вопросы

1. Для чего нужен класс Object?

Класс Object является базовым классом всей иерархии классов в Java. Все остальные классы, включая пользовательские, автоматически наследуют этот класс, даже если явного указания нет. Это значит, что каждый объект в Java имеет методы, определенные в классе Object.

В этом классе определены важные методы getClass(), toString(), equals(), hashCode(), clone(), finalize()

2. Для чего нужно переопределять методы equals() и hashCode()?

По умолчанию метод equals() проверяет идентичность ссылок (==), что зачастую недостаточно. Например, у вас есть два объекта одного класса, содержащие одинаковые данные, но имеющие разные ссылки. В этом случае логично считать такие объекты равными. Для этого метод equals() следует переопределить так, чтобы сравнивать содержимое объектов.

Листинг 3.1. Переопределение метода equals()

```
1.  @Override
2.  public boolean equals(Object obj) {
3.      if (this == obj) return true;
4.     if (obj == null || getClass() != obj.getClass()) return false;
5.
6.      YourClass other = (YourClass) obj;
7.      return this.field1.equals(other.field1)
8.          && this.field2 == other.field2;
9.  }
```

3. Какие есть правила переопределения методов equals() и hashCode()?

Когда вы переопределяете метод equals(), важно также переопределить метод hashCode(). Причина заключается в следующем:

• два объекта, которые считаются равными методом equals(), должны иметь одинаковое значение хеш-кода;

• коллекции, основанные на хэш-таблицах (например, HashMap, HashSet), используют хеш-коды для эффективного хранения и поиска данных. Несогласованность между методами equals() и hashCode() приведет к неправильной работе этих структур данных.

Листинг 3.2. Переопределение метода hashCode()

```
1.  @Override
2.  public int hashCode() {
3.      int result = field1.hashCode();
4.      result = 31 * result + field2;
5.      return result;
6.  }
```

Важно соблюдать контракт между этими двумя методами:

• если два объекта равны согласно equals(), то их хеш-коды должны совпадать;

• если два объекта имеют одинаковый хеш-код, это не обязательно означает, что они равны.

4. Что делает метод toString()? Почему его часто переопределяют?

Возвращает строковое представление объекта. По умолчанию возвращает имя класса и хеш-код объекта, но часто его переопределяют для получения более информативного результата;

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

Вызывается сборщиком мусора перед удалением объекта. Этот метод редко используется, поскольку поведение сборщика мусора непредсказуемо.

6. Что такое коллизия?

Хэш-таблица — это структура данных, которая используется для хранения пар «ключ-значение». Она основана на идее хэш-функции, которая преобразует ключ в индекс массива, где хранится значение.

Хэш-функция должна быть быстрой и однозначной, то есть каждому ключу должен соответствовать уникальный индекс. Если два разных ключа преобразуются в один и тот же индекс, то это называется коллизией.

7. Какие есть способы разрешения коллизий?

Коллизии могут быть разрешены различными способами, например с помощью метода цепочек или открытой адресации. В методе цепочек каждый индекс массива содержит связанный список пар «ключ-значение». Если происходит коллизия, то новая пара добавляется в конец списка. При поиске значения по ключу нужно просмотреть все элементы списка, начиная с первого.

8. Как хранятся данные в хэш-таблице?

Данные в хэш-таблице (например, HashMap) хранятся в виде пар ключ значение. Каждый ключ сначала преобразуется в хэш-код с помощью метода hashCode(). Хэш-код используется для определения индекса в массиве (внутренний массив называется бакетами). В каждом бакете могут храниться один или несколько элементов. Итог: HashMap — это массив бакетов, где каждый бакет хранит список элементов с одинаковым хэш-кодом.

9. Что происходит, если в хэш-таблицу добавить элемент с одинаковым значением ключа?

В HashMap ключи являются уникальными. При попытке вставки элемента с ключом, который уже присутствует в таблице, старое значение, связанное с этим ключом, заменяется новым. Общее количество элементов

при этом не увеличивается. Такая стратегия позволяет поддерживать уникальность ключей и предотвращает дублирование информации.

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

Если два различных ключа обладают одинаковым хэш-кодом, возникает коллизия. В этом случае оба элемента размещаются в одном бакете. Для различения ключей используется метод equals(). Следовательно, хэш-таблица допускает наличие нескольких ключей с одинаковым хэш-кодом, при этом корректная идентификация ключей обеспечивается проверкой на равенство.

11. Как изменяется HashMap при достижении порогового значения?

Каждая HashMap имеет пороговое значение (threshold), которое определяет максимально допустимое количество элементов в таблице до необходимости расширения. При достижении этого порога осуществляется рехеширование: создаётся новый массив бакетов увеличенного размера (обычно в два раза), после чего все существующие элементы перераспределяются по новым индексам. Это обеспечивает сохранение высокой производительности операций вставки, поиска и удаления, поддерживая амортизированное время доступа на уровне

Вывод

В лабораторной работе изучены принципы работы хэш-таблиц и методы обработки коллизий. Реализован класс HashTable с операциями вставки, поиска и удаления, а также освоено использование HashMap для хранения информации о книгах.

Ссылка на GitHub: https://github.com/Evgeshhha/Information-technology-and-programming