МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Лабораторная работа № 3

«Создание хэш-таблиц»

Выполнила: Студентка группы БВТ2306

Максимова Мария

Москва 2024

Гитхаб: https://github.com/Mashaaaaa7/Lab3

Задачи:

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: Работа с встроенным классом HashMap.

Вариант 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.

При выполнении задания 1 реализуем методы put, get, remove (key) и добавим методы Size и isEmpty.

Код к заданию 1:

```
import java.util.LinkedList;
import java.util.Objects;

public class HashTable<K, V> {
    //объявляет вложенный класс.
    private static class Entry<K, V> {

        private K key;
        private V value;

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

        public K getKey() {

            return key;
        }
        public V getValue() {
```

```
            return value;
        }

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

        @Override
        public boolean equals(Object obj) { // определяет, равны ли два
объекта

            if (this == obj) { //Проверяет, сравниваются ли один и тот же
объект
                return true;
            }

            if (!(obj instanceof Entry<?, ?> entry)) { //Проверяет, является
ли передаваемый объект obj экземпляром класса Entry
                return false;
            }

            return Objects.equals(getKey(), entry.getKey()); //Сравнивает
ключи двух объектов
        }
        //реализует метод hashCode для класса Entry, который является
вложенным классом в классе HashTable
        @Override
        public int hashCode() {
            return Objects.hash(getKey());
        }

    }
    //объявляет массив ссылок на LinkedList
    private LinkedList<Entry<K, V>>[] table;
    private int size; //поле целого типа, хранит количество элементов в
HashTable

    public HashTable(int capacity) { //задает начальную емкость хэш-таблицы
        table = new LinkedList[capacity];
        size = 0;
    }

    private int hash(K key) { //принимает ключ и возвращает его хеш-код
        return Math.abs(Objects.hashCode(key)) % table.length; // Возвращает
абсолютное значение хеш-кода, чтобы получить неотрицательное число
    }

    public void put(K key, V value) { //добавляет

        int index = hash(key);

        if (table[index] == null) {
            table[index] = new LinkedList<>();
        }

        //обновление значения по ключу
        for (Entry<K, V> entry : table[index]) { //список элементов, которые
хранятся в определенной ячейке массива table.
            if (entry.getKey().equals(key)) { //получает ключ текущего
элемента Entry.
                entry.setValue(value); //обновление значения
                return;
            }
```

```
        }
//прибавление нового элемента
        table[index].add(new Entry<>(key, value));
        size++;
    }

    public V get(K key) { //возвращает значение

        int index = hash(key);

        LinkedList<Entry<K, V>> bucket = table[index];

        if (bucket != null) { //bucket-список элементов Entry
            for (Entry<K, V> entry : bucket) {
                if (entry.getKey().equals(key)) {
                    return entry.getValue();
                }
            }
        }

        return null;
    }

    public V remove(K key) {

        int index = hash(key);

        LinkedList<Entry<K, V>> bucket = table[index];

        if (bucket != null) {
            for (Entry<K, V> entry : bucket) {
                if (entry.getKey().equals(key)) {
                    V removedValue = entry.getValue();
                    bucket.remove(entry);
                    size--;
                    return removedValue;
                }
            }
        }

        return null;
    }

    public int size() {

        return size;
    }

    public boolean isEmpty() {

        return size == 0;
    }

}
```

Этот код представляет простую имитацию хэш-таблицы с использованием класса Entry.

В этом коде реализована простая хэш-таблица, использующая отдельные цепочки для разрешения коллизий.

1. Импорт:

import java.util.LinkedList;: Импортирует класс LinkedList, который используется для обработки коллизий в хэш-таблице (отдельная цепочка). Когда несколько ключей хэшируются по одному и тому же индексу, они сохраняются в связанном списке по этому индексу.

import java.util.Objects;: Импортирует класс Objects, который предоставляет служебные методы для работы с объектами, такие как hashCode() и equals().

2. Класс HashTable<K, V>:

Это универсальный класс (использующий <K, V>), который реализует хэш-таблицу. K представляет тип ключа, а V - тип значения.

Запись<K, V> Внутренний класс: Этот вложенный класс представляет собой отдельную запись в хэш-таблице. В нем хранятся ключ (key) и значение (value). Он переопределяет методы equals() и hashCode(), что имеет решающее значение для правильной работы хэш-таблицы. Метод equals() сравнивает записи на основе их ключей; метод hashCode() вычисляет хэш-код на основе ключа.

таблица: LinkedList<Запись<K, V>>[]: Это основная структура данных хэш-таблицы. Это массив связанных списков. Каждый индекс в массиве представляет собой сегмент, и каждый сегмент может содержать несколько записей (в случае коллизий).

размер: целочисленная переменная, которая отслеживает количество пар ключ-значение, хранящихся в данный момент в хэш-таблице.

Это универсальный класс (использующий <K, V>), который реализует хэш-таблицу. K представляет тип ключа, а V - тип значения.

Запись<K, V> Внутренний класс: Этот вложенный класс представляет собой отдельную запись в хэш-таблице. В нем хранятся ключ (key) и значение (value). Он переопределяет методы equals() и hashCode(), что имеет решающее значение для правильной работы хэш-таблицы.

Метод equals() сравнивает записи на основе их ключей; метод hashCode() вычисляет хэш-код на основе ключа. таблица: LinkedList<Запись<K, V>>[]: Это основная структура данных хэш-таблицы. Это массив связанных списков. Каждый индекс в массиве представляет собой сегмент, и каждый сегмент может содержать несколько записей (в случае коллизий). размер: целочисленная переменная, которая отслеживает количество пар ключ-значение, хранящихся в данный момент в хэш-таблице.

Конструктор HashTable(int capacity): Этот конструктор инициализирует хэш-таблицу с заданной емкостью. Он создает массив LinkedLists указанного размера.

метод хэширования (K ключей): Этот метод вычисляет хэш-код для данного ключа, используя Objects.hashCode(ключ), и умножает его на длину таблицы, чтобы получить индекс в массиве. Он использует Math.abs() для обеспечения неотрицательного индекса.

метод put(ключ K, значение V): Этот метод добавляет новую пару ключ-значение в хэш-таблицу. Сначала вычисляется хэш-код для определения индекса. Если корзина пуста, создается новый список ссылок. Затем он выполняет итерацию по связанному списку корзины, чтобы проверить, существует ли ключ. Если это так, он обновляет значение; в противном случае он добавляет новую запись.

метод получения (K-ключа): Этот метод извлекает значение, связанное с данным ключом. Выполняется поиск в связанном списке по вычисленному индексу.

Метод удаления(K-ключа): Этот метод удаляет пару ключ-значение, связанную с данным ключом. Если элемент найден, он удаляет его из LinkedList и уменьшает размер.

Методы size() и isEmpty(): Эти методы возвращают количество элементов в хэш-таблице и проверяют, пуста ли она.

Этот код предоставляет базовую, но функциональную реализацию хэш-таблицы, использующую отдельные цепочки для обработки коллизий. В нем используется массив связанных списков для хранения записей. Несмотря на свою функциональность, встроенная в Java хэш-карта, как правило, предпочтительнее для производственного кода из-за эффективности и лучшей обработки крайних случаев. Однако эта реализация полезна для понимания основных концепций реализации хэш-таблиц.

Далее, перейдем к заданию 2.

Мой код:

```
import java.util.HashMap;

class Book { //определяет класс, который представляет книгу
    private String title;
    private String author;
    private int copies;

    public Book(String title, String author, int copies) { //инициализация
        this.title = title;
        this.author = author;
        this.copies = copies;
    }


    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    public int getCopies() {

        return copies;
    }

    // Метод для изменения количества копий
    public void setCopies(int copies) {

        this.copies = copies;
```

```
    }
}

public class Library { //определяет класс Библиотека
    private HashMap<String, Book> bookMap; // Явное объявление HashMap

    public Library() {
        bookMap = new HashMap<>(); //создает поле bookMap типа HashMap,
которое будет хранить книги с ключом ISBN.
    }

    public Book addBook(String isbn, Book book) {

        bookMap.put(isbn, book); // Добавление книги в хэш-таблиц
        return book;
    }

    public Book findBook(String isbn) {

        return bookMap.get(isbn);
    }

    public Book removeBook(String isbn) {

        bookMap.remove(isbn);
        return null;
    }
}
```

```
public class Main {

    public static void main(String[] args) {
        Library library = new Library(); //создает объект Library

        // Добавление книг
        library.addBook("978-0143034231", new Book("1984", "George Orwell",
5));
        library.addBook("978-0141439501", new Book("Pride and Prejudice",
"Jane Austen", 2));

        // Поиск книги
        Book foundBook = library.findBook("978-0143034231");
        if (foundBook != null) { //проверяет, найдена ли книга.
            System.out.println("Найдена книга: " + foundBook.getTitle() + "
by " + foundBook.getAuthor() + ", " + foundBook.getCopies() + " copies");
        } else {
            System.out.println("Книга не найдена." + foundBook.getTitle() + "
by " + foundBook.getAuthor() + ", " + foundBook.getCopies() + " copies");
        }

        Book foundBook2 = library.findBook("978-0141439501");
        if (foundBook2 != null) {
            System.out.println("Найдена книга: " + foundBook2.getTitle() + "
by " + foundBook2.getAuthor() + ", " + foundBook2.getCopies() + " copies");
        } else {
            System.out.println("Книга не найдена.");
        }

        // Удаление книги
        library.removeBook("978-0141439501");

        // Поиск удаленной книги
        foundBook = library.findBook("978-0143034231");
```

```
        if (foundBook != null) {
            System.out.println("Книга найдена: " + foundBook.getTitle() + "
by " + foundBook.getAuthor() +  ", " + foundBook.getCopies() + " copies");
        } else {
            System.out.println("Книга не найдена.");
        }


        foundBook2 = library.findBook("978-0141439501");
        if (foundBook2 != null) {
            System.out.println("Книга найдена: " + foundBook2.getTitle() + "
by " + foundBook2.getAuthor() +  ", " + foundBook.getCopies() + " copies");
        } else {
            System.out.println("Книга не найдена.");
        }
    }
}
```

Этот код представляет собой пример использования HashMap в Java для хранения книг в библиотеке.

В методе main создается объект класса Library и добавляются две книги. Затем находится книга по ISBN, выводится информация о ней. Книга удаляется по ISBN и затем снова производится поиск этой книги, результат которого выводится в консоль.

Класс Book:

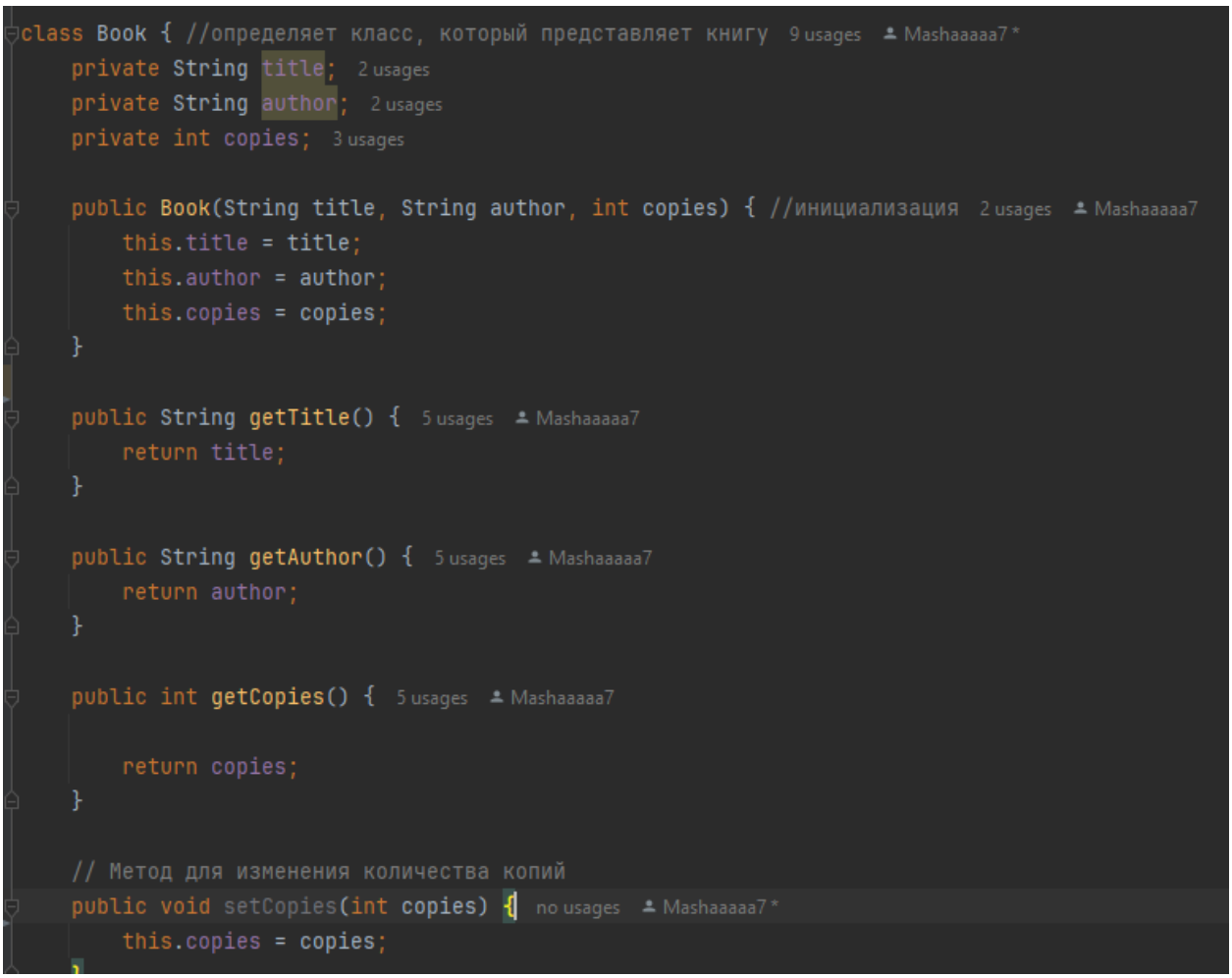

Рисунок 1

Класс Book представляет книгу с полями заголовок, автор и количество копий. В классе Library создается HashMap bookMap, где ключом является ISBN книги, а значением объект книги.

- class Book: определяет класс, который представляет книгу с названием, автором и количеством копий.
- private String title;: хранит название книги.
- private String author;: хранит имя автора.
- private int copies;: хранит количество копий книги.
- Book(String title, String author, int copies): конструктор, который инициализирует объект Book с указанными названием, автором и количеством копий.

- getTitle(), getAuthor(), getCopies(): геттеры для получения информации о книге.
- setCopies(int copies): сеттер для изменения количества копий.

Класс Library:

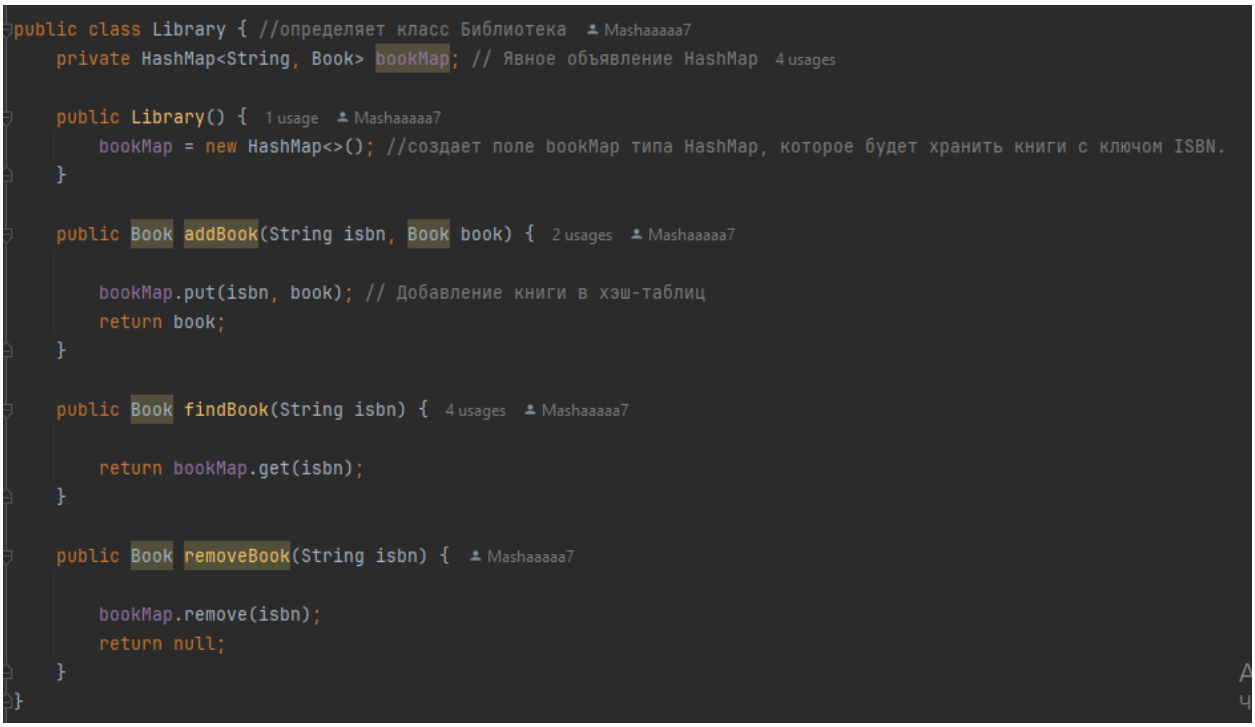

Рисунок 2

- public class Library: определяет класс Library.
- private HashMap<String, Book> bookMap;: создает поле bookMap типа HashMap, которое будет хранить книги с ключом ISBN.
- Library(): конструктор, который инициализирует bookMap пустым HashMap.

Методы Library:

- public Book addBook(String isbn, Book book): добавляет книгу в библиотеку по ее ISBN.
- public Book findBook(String isbn): ищет книгу по ее ISBN и возвращает ее объект, если она найдена, иначе возвращает null.
- public Book removeBook(String isbn): удаляет книгу из библиотеки по ее ISBN.

main() метод:

```java
public class Main {

    public static void main(String[] args) {
        Library library = new Library(); //создает объект Library

        // Добавление книг
        library.addBook("978-0143034231", new Book("1984", "George Orwell", 5));
        library.addBook("978-0141439501", new Book("Pride and Prejudice", "Jane Austen", 2));

        // Поиск книги
        Book foundBook = library.findBook("978-0143034231");
        if (foundBook != null) { //проверяет, найдена ли книга.
            System.out.println("Найдена книга: " + foundBook.getTitle() + " by " + foundBook.getAuthor() + ", " + foundBook.getCopies() + " copies");
        } else {
            System.out.println("Книга не найдена." + foundBook.getTitle() + " by " + foundBook.getAuthor() + ", " + foundBook.getCopies() + " copies");
        }

        Book foundBook2 = library.findBook("978-0141439501");
        if (foundBook2 != null) {
            System.out.println("Найдена книга: " + foundBook2.getTitle() + " by " + foundBook2.getAuthor() + ", " + foundBook2.getCopies() + " copies");
        } else {
            System.out.println("Книга не найдена.");
        }
```

Рисунок 3

- public static void main(String[] args): точка входа в программу.
- Library library = new Library();: создает объект Library.

- library.addBook("978-0143034231", new Book("1984", "George Orwell", 5));: добавляет книгу “1984” в библиотеку.
- library.addBook("978-0141439501", new Book("Pride and Prejudice", " Jane Austen", 5));: добавляет книгу “Pride and Prejudice” в библиотеку.

- Book foundBook = library.findBook("978-0143034231");: ищет книгу по ISBN и сохраняет ее в переменную foundBook.
- Book foundBook2 = library.findBook("978-0141439501");: ищет книгу по ISBN и сохраняет ее в переменную foundBook.

- if (foundBook != null) { ... }: проверяет, найдена ли книга. Если да, то выводит информацию о ней, иначе выводит сообщение о том, что книга не найдена.

Рисунок 4

- library.removeBook("978-0141439501");: удаляет книгу по ISBN.

- foundBook = library.findBook("978-0143034231");: ищеткнигу.

- if (foundBook != null) { ... }: проверяет, найдена ли книга. Если да, то выводит сообщение о том, что книга найдена, иначе выводит сообщение о том, что книга не найдена.

Вывод в консоли:

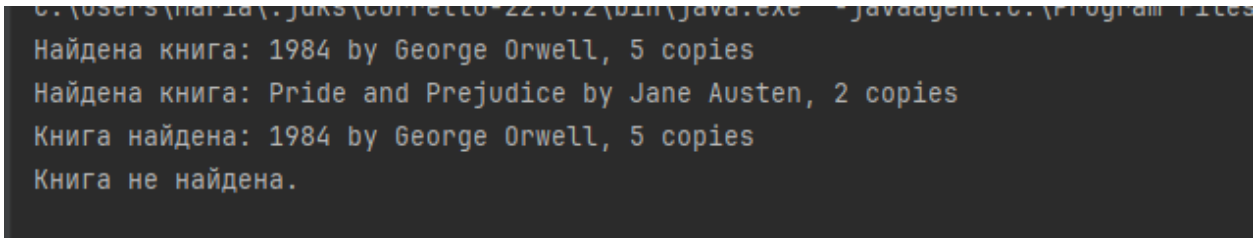
Найдена книга: 1984 by George Orwell, 5 copies
Найдена книга: Pride and Prejudice by Jane Austen, 2 copies
Книга найдена: 1984 by George Orwell, 5 copies
Книга не найдена.

Рисунок 5