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)

Факультет
Информационные технологии
Искусственный интеллект и машинное обучение

по теме:

Класс Object. Работа с хэш-таблицами

Студент:
БВТ 2401 *А.И. Меланич*

Предподаватель:

Москва 2025

СОДЕРЖАНИЕ

1 ВВЕДЕНИЕ

1.1 Цель работы

Разработка и реализация хэш-таблицы для эффективного хранения и обработки информации о продуктах, с освоением принципов работы хэш-структур данных и алгоритмов их обработки.

Конкретные задачи для достижения цели

- **Реализация структуры данных:** создание хэш-таблицы с использованием номера продуктах в качестве ключа
- **Разработка класса:** создание объекта для хранения информации о продукте (дата, список товаров, статус)
- **Программирование основных операций:**
  - вставка нового продукта в таблицу
  - поиск продукта по номеру
  - удаление продукта из таблицы
  - изменение статуса существующего продукта
- **Отработка практических навыков:**
  - применение хэш-функций
  - обработка коллизий
  - обновление продукта по ключу

1.2 Описание 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. Пример реализации метода put(key, value) представлен на ли- стинге 3.3.

Листинг 3.3. 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++;
}
```

Реализация хэш-таблицы для учета продуктов на складе. Ключом будет штрихкод товара, а значением — объект класса Product, содержащий информацию о названии, цене и доступном количестве. Необходимо реализовать операции вставки, поиска и удаления продукта по штрихкоду

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Хеш-таблицы являются одной из наиболее эффективных структур данных для быстрого поиска, добавления и удаления элементов. В Java они реализованы в классах `Hashtable` и `HashMap`.

2.1 Основные концепции

2.1.1 Принцип работы

Хеш-таблица представляет собой структуру данных, которая хранит пары ключ-значение. Для определения позиции элемента используется хеш-функция.

2.1.2 Хеш-функция

Хеш-функция преобразует ключ в числовой индекс массива. Формально:

$$f : K \to \{0, 1, ..., N - 1\} \tag{1}$$

где K — множество ключей, N — размер таблицы.

2.2 Структура хеш-таблицы

Хеш-таблица состоит из:

- Массива бакетов (корзин)
- Механизма разрешения коллизий
- Хеш-функции

2.3 Разрешение коллизий

Коллизия возникает, когда разные ключи имеют одинаковый хеш-код. В Java используется метод цепочек:

```
public class Hashtable<K, V> {
    private Entry<K, V>[] table;

    private static class Entry<K, V> {
        final K key;
        V value;
        Entry<K, V> next;

        Entry(K key, V value, Entry<K, V> next) {
            this.key = key;
            this.value = value;
            this.next = next;
        }
    }
}
```

2.4 Основные операции

2.4.1 Добавление элемента

```
public void put(K key, V value) {
    int hash = hash(key);
    int index = indexFor(hash, table.length);

    for (Entry<K, V> e = table[index]; e != null; e = e.next) {
        if (e.hash == hash && (e.key == key || key.equals(e.key))) {
            e.value = value;
            return;
        }
    }
    table[index] = new Entry<>(key, value, table[index]);
}
```

2.4.2 Поиск элемента

```
public V get(Object key) {
    int hash = hash(key);
    int index = indexFor(hash, table.length);

    for (Entry<K, V> e = table[index]; e != null; e = e.next) {
        if (e.hash == hash && (e.key == key || key.equals(e.key))) {
            return e.value;
        }
    }
    return null;
}
```

2.5 Характеристики производительности

- Средняя сложность операций: $O(1)$
- В худшем случае: $O(n)$ (при большом количестве коллизий)

2.6 Заключение

Хеш-таблицы являются мощным инструментом для работы с данными в Java, обеспечивая быстрый доступ и эффективное использование памяти при правильном использовании.

3 РЕАЛИЗАЦИЯ

3.1 Создание вспомогательного класса для хранения типа данных в хэш-таблице

```
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;
  }
}
```

Класс **Entry** представляет собой вспомогательный класс-обёртку для хранения пар ключ-значение в хэш-таблице. Данный класс является универсальным (generic) и может работать с любыми типами данных для ключа и значения.

3.1.1 Назначение

Класс предназначен для:

- Хранения пары ключ-значение
- Использования в структурах данных на основе хэширования
- Обеспечения инкапсуляции данных
- Поддержки типизации через дженерики

3.1.2 Структура класса

Параметры типа

- **K** - тип ключа (Key)
- **V** - тип значения (Value)

Поля класса

- **key** - приватное поле для хранения ключа типа K
- **value** - приватное поле для хранения значения типа V

Конструктор

```
public Entry(K key, V value)
```

- Принимает два параметра: ключ и значение
- Инициализирует соответствующие поля объекта

Методы доступа

```
public K getKey()
```

- Возвращает сохранённый ключ
- Обеспечивает доступ только для чтения

```
public V getValue()
```

- Возвращает сохранённое значение
- Обеспечивает доступ только для чтения

```
public void setValue(V value)
```

- Позволяет изменить значение
- Сохраняет новое значение в поле value

3.1.3 Особенности реализации

- Класс использует инкапсуляцию для защиты данных
- Все поля являются приватными
- Предоставляет контролируемый доступ через публичные методы
- Поддерживает работу с любыми типами данных благодаря дженерикам

3.1.4 Пример использования

```
Entry<String, Integer> entry = new Entry<>("name", 42);
String key = entry.getKey(); // Получаем ключ
Integer value = entry.getValue(); // Получаем значение
entry.setValue(100); // Изменяем значение
```

Данный класс является базовым строительным блоком для реализации более сложных структур данных, таких как HashMap, HashTable и других хэш-ориентированных коллекций.

3.2 Создание класса для хранения данных о продукте

```
class Product {
  private String name;
  private double price;
  private int quantity;

  public Product(String name, double price, int quantity) {
    this.name = name;
    this.price = price;
    this.quantity = quantity;
  }

  public String getName() {
    return name;
  }

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

  public double getPrice() {
    return price;
  }

  public void setPrice(double price) {
    this.price = price;
  }
```

```
  public int getQuantity() {
    return quantity;
  }

  public void setQuantity(int quantity) {
    this.quantity = quantity;
  }

  @Override
  public String toString() {
    return "Product{name='" + name + "', price=" + price + ", quantity="
        + quantity + "}";
  }
}
```

Класс **Product** представляет собой сущность для хранения информации о товаре. Класс содержит основные характеристики продукта: название, цену и количество.

3.2.1 Назначение

Класс предназначен для:

- Хранения информации о товарах
- Обеспечения инкапсуляции данных
- Предоставления методов доступа к полям
- Форматированного вывода информации о товаре

3.2.2 Структура класса

Поля класса

- **name** - приватное поле типа String для хранения названия товара
- **price** - приватное поле типа double для хранения цены
- **quantity** - приватное поле типа int для хранения количества товара

Конструктор

```
public Product(String name, double price, int quantity)
```

- Инициализирует все поля объекта
- Принимает параметры для установки начальных значений

Геттеры и сеттеры

Для каждого поля определены методы доступа:

```
public String getName()
public void setName(String name)
```

```
public double getPrice()
public void setPrice(double price)
```

```
public int getQuantity()
public void setQuantity(int quantity)
```

Метод toString

```
@Override
public String toString() {
    return "Product{name='" + name + "', price=" + price + ", quantity=" + quantity + "}";
}
```

Предоставляет строковое представление объекта в формате: `Product{name='название', price=цена, quantity=количество}`

3.2.3 Особенности реализации

- Полная инкапсуляция данных
- Контроль доступа через геттеры и сеттеры
- Корректное строковое представление объекта
- Простота расширения функциональности

3.2.4 Пример использования

```
Product product = new Product("Смартфон", 29999.99, 10);
String name = product.getName();
double price = product.getPrice();
int quantity = product.getQuantity();

System.out.println(product);
// Выведет: Product{name='Смартфон', price=29999.99, quantity=10}
```

3.3 Создание класса HashTable

```
import java.util.LinkedList;

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

  @SuppressWarnings("unchecked")
  public HashTable() {
    table = new LinkedList[DEFAULT_CAPACITY];
    size = 0;
  }

  @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<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 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;
  }

  public V remove(K key) {
    int index = hash(key);
```

```
    if (table[index] == null) {
      return null;
    }

    for (Entry<K, V> entry : table[index]) {
      if (entry.getKey().equals(key)) {
        V value = entry.getValue();
        table[index].remove(entry);
        size--;
        return value;
      }
    }

    return null;
  }

  public int size() {
    return size;
  }

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

  public void printTable() {
    for (int i = 0; i < table.length; i++) {
      System.out.print("Bucket " + i + ": ");
      if (table[i] != null) {
        for (Entry<K, V> entry : table[i]) {
          System.out
              .print("[" + entry.getKey() + " -> " + entry.getValue() + "] ");
        }
      }
      System.out.println();
    }
  }
}
```

Класс **HashTable** представляет собой реализацию хэш-таблицы с использованием обобщенных типов (generics). Хэш-таблица хранит пары ключ-значение и использует связанные списки для разрешения коллизий.

3.3.1 Назначение

Класс предназначен для:

- Хранения пар ключ-значение
- Обеспечения быстрого доступа к данным по ключу
- Управления коллекцией элементов с помощью хэширования
- Разрешения коллизий через связанные списки

3.3.2 Структура класса

Параметры типа

- **K** - тип ключа
- **V** - тип значения

Поля класса

- **DEFAULT_CAPACITY** - константа, задающая начальную емкость таблицы
- **table** - массив связанных списков для хранения элементов
- **size** - количество элементов в таблице

Конструкторы

```
public HashTable()
```

Создает таблицу с емкостью по умолчанию (16)

```
public HashTable(int capacity)
```

Создает таблицу с заданной емкостью

Основные методы

Хэширование

```
private int hash(K key)
```

Вычисляет индекс для заданного ключа

Добавление элемента

```
public void put(K key, V value)
```

Добавляет или обновляет элемент в таблице

Получение элемента

```
public V get(K key)
```

Возвращает значение по заданному ключу

Удаление элемента

```
public V remove(K key)
```

Удаляет элемент по заданному ключу и возвращает его значение

Вспомогательные методы

- **size()** - возвращает количество элементов
- **isEmpty()** - проверяет пустоту таблицы
- **printTable()** - выводит содержимое таблицы

3.3.3 Особенности реализации

- Использование обобщенных типов для гибкости
- Разрешение коллизий через связанные списки
- Инкапсуляция данных
- Контроль размера коллекции
- Возможность вывода содержимого таблицы

3.3.4 Пример использования

```
HashTable<String, Integer> table = new HashTable<>();
table.put("key1", 10);
table.put("key2", 20);

Integer value = table.get("key1"); // Получаем значение
table.remove("key2"); // Удаляем элемент

table.printTable(); // Выводим содержимое таблицы
```

3.3.5 Применение

Класс может использоваться в различных приложениях, где требуется:

– Быстрый доступ к данным по ключу **Хранение пар ключ-значение**
– Реализация кэширования данных
– Создание ассоциативных массивов
– Обработка больших объемов данных

4 ВЫВОД

В ходе выполнения лабораторной работы была успешно реализована структура данных **хэш-таблица** с использованием обобщенных типов (generics) и связанных списков для разрешения коллизий.

4.1 Основные результаты работы

1. Разработан вспомогательный класс **Entry**, который:

– Обеспечивает хранение пар ключ-значение
– Реализует инкапсуляцию данных
– Предоставляет методы доступа к полям

2. Создан основной класс **HashTable**, который:

– Реализует базовую функциональность хэш-таблицы
– Использует массив связанных списков для хранения данных
– Обеспечивает эффективное добавление, поиск и удаление элементов

3. В процессе реализации были достигнуты следующие цели:

– Реализована **хэш-функция** для вычисления индекса элемента
– Реализовано **разрешение коллизий** через связанные списки
– Обеспечен **контроль размера** коллекции
– Реализована возможность **визуализации** содержимого таблицы

4.2 Практическая значимость работы

Практическая значимость работы заключается в следующем:

– Получен опыт работы с обобщенными типами (generics)
– Изучены принципы работы хэш-таблиц
– Освоены методы разрешения коллизий
– Получена практика реализации сложных структур данных

4.3 Решение поставленных задач

В ходе работы были решены следующие задачи:

- Создана эффективная структура хранения данных
- Реализованы основные операции с элементами
- Обеспечена безопасность типов через использование generics
- Реализована обработка особых случаев (пустые значения, коллизии)

4.4 Анализ результатов

Полученные результаты показывают, что разработанная хэш-таблица:

- Обеспечивает быстрый доступ к данным
- Эффективно использует память
- Легко масштабируется
- Может быть использована в реальных приложениях

4.5 Заключение

Таким образом, цель лабораторной работы достигнута, основные задачи выполнены, а полученная реализация хэш-таблицы может служить основой для создания более сложных приложений, требующих эффективного хранения и обработки данных.