**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Ордена Трудового Красного Знамени федеральное государственное бюджетное учреждение высшего образования

**«Московский технический университет связи и информатики»**

**ФАКУЛЬТЕТ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ**

Кафедра «МКиИТ»

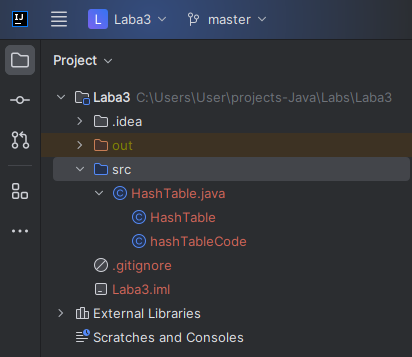
**Информационные технологии и**

**программирование**

**Лабораторная работа по ИТП №3**

Выполнил: студент группы БВТ2205

Белов Егор Олегович



Изображение 1 – Структура файлов.

Сначала мы импортируем библиотеки. Затем создаём точку входа в программу в виде main. В конце мы пропишем все методы для проверки работы hashTableCode.

Далее идет жесткий каркас, на котором будет выполнено первое и второе задания:

**Основной класс hashTableCode<K, V> имеет следующие методы и поля:**

**table: поле типа ArrayList<LinkedList<Entry<K, V>>>, используется для хранения списков и связывания этих списков между друг другом.**

**Вложенный статический класс Entry<K, V> представляет элемент хэш-таблицы, содержащий ключ (key) и значение (value). Он имеет следующие методы и поля:**

**key: поле для хранения ключа.**

**value: поле для хранения значения.**

**Entry(K key, V value): конструктор для создания экземпляра класса Entry, который принимает ключ и значение в качестве параметров.**

**getKey(): метод для получения ключа.**

**setValue(V values): метод для установки значения.**

**toString(): переопределенный метод, который возвращает строковое представление объекта Entry, включающее ключ и значение.**

**Методы класса hashTableCode<K, V>:**

**put(K keys, V values): метод для добавления элемента в хэш-таблицу. Он вычисляет хэш ключа, определяет индекс в массиве table, создает новый список в случае отсутствия списка по указанному индексу и добавляет элемент в список. Если ключ уже присутствует, то обновляет значение.**

**get(K key): метод для получения значения элемента по ключу. Он вычисляет хэш ключа, определяет индекс в массиве table, ищет элемент в связанном списке по указанному индексу и возвращает его значение, либо выводит сообщение о том, что элемент не найден.**

**remove(K key): метод для удаления элемента по ключу. Он вычисляет хэш ключа, определяет индекс в массиве table, ищет элемент по указанному индексу в связанном списке и удаляет его.**

**size(): метод для получения количества элементов в хэш-таблице. Он перебирает все связанные списки в массиве table и подсчитывает размер каждого списка.**

**isEmpty(): метод для проверки, пуста ли хэш-таблица. Он возвращает true, если размер массива table равен 0, иначе возвращает false.**

**println(): метод для вывода элементов хэш-таблицы на консоль. Он перебирает все связанные списки в массиве table и выводит их содержимое в виде строкового представления.**

**Общий функционал этой реализации хэш-таблицы включает добавление, получение, удаление элементов, подсчет размера и проверку на пустоту. При возникновении коллизий (когда несколько элементов имеют одинаковый хэш) используется метод цепочек - элементы с одинаковым хэшем хранятся в связанных списках.**

**Задание 1:**

Код файла HashTable:

import java.util.ArrayList;  
import java.util.LinkedList;  
import static java.util.Objects.*hash*;  
  
public class HashTable {  
 public static void main(String[] args) {  
 /\* ТЕСТИРОВКА КОДА. --------------------------------------------------------------------------------------------  
  
 hashTableCode<Object, Object> table = new hashTableCode<>();  
  
 table.put("ab", 5);  
 table.put('a', 5);  
 table.put(97, 19);  
 table.put("Banana", 105);  
  
 table.println();  
  
 Object value = table.get("Banana");  
 System.out.println("\n" + value + "\n");  
 Object value2 = table.get("B");  
 System.out.println("\n" + value2 + "\n");  
  
 System.out.println("\n" + table.size() + "\n");  
  
 table.remove('a');  
  
 table.println();  
  
 System.out.println("\n" + table.size());  
  
 System.out.println("\n" + table.isEmpty());  
 ----------------------------------------------------------------------------------------------------------------  
 \*/  
 }  
}  
  
class hashTableCode<K, V> {  
 private final ArrayList<LinkedList<Entry<K, V>>> table;  
  
 hashTableCode() {  
 this.table = new ArrayList<>();  
 }  
  
 static class Entry<K, V> {  
 K key;  
 V value;  
  
 public Entry(K key, V value) {  
 this.key = key;  
 this.value = value;  
 }  
  
 public K getKey() {  
 return key;  
 }  
  
 public void setValue(V values) {  
 value = values;  
 }  
  
 @Override  
 public String toString() {  
 return "Entry[key = " + key + ", value = " + value + "]";  
 }  
 }  
  
 public void put(K keys, V values) {  
 int index = *hash*(keys) % 16;  
  
 if (index >= table.size()) {  
 for (int i = 0; i <= index; i++) {  
 table.add(null);  
 }  
 }  
  
 if (table.get(index) == null) {  
 table.set(index, new LinkedList<>());  
 }  
  
 for (Entry<K, V> entry : table.get(index)) {  
 if (entry.getKey().equals(keys)) {  
 entry.setValue(values);  
 return;  
 }  
 }  
 table.get(index).add(new Entry<>(keys, values));  
  
 }  
  
 public V get(K key) {  
 int index = *hash*(key) % 16;  
  
 if (index >= table.size()) {  
 System.*out*.println("Такого значения нет.");  
 return null;  
 }  
  
 LinkedList<Entry<K, V>> value = table.get(index);  
  
 if (value == null) {  
 System.*out*.println("Такого значения нет.");  
 return null;  
 }  
  
 for (Entry<K, V> entry : value) {  
 if (entry.getKey().equals(key)) {  
 return entry.value;  
 }  
 }  
  
 System.*out*.println("Такого значения нет.");  
 return null;  
 }  
  
 public void remove(K key) {  
 int index = *hash*(key) % 16;  
  
 LinkedList<Entry<K, V>> value = table.get(index);  
  
 for (int i = 0; i < value.size(); i++) {  
 if (value.get(i).getKey().equals(key)) {  
 value.remove(i);  
 }  
 }  
  
 }  
  
 public int size() {  
 int size = 0;  
  
 for (LinkedList<Entry<K, V>> entry : table) {  
 if (entry != null) {  
 size += entry.size();  
 }  
 }  
  
 return size;  
 }  
  
 public boolean isEmpty() {  
 if (table.size() != 0) {  
 return false;  
 }  
  
 return true;  
 }  
  
 public void println() {  
 for (int i = 0; i < table.size(); i++) {  
 if (table.get(i) != null) {  
 System.*out*.println(table.get(i).toString());  
 }  
 }  
 }  
}

**Задание 2, Вариант 3:**

Теперь мы должны реализовать хэш-таблицу для хранения информации о товарах в интернет-магазине. Как раз объяснение того, почему прошлый код я назвал жестким каркасом. Очевидно, что ничего менять не надо, чтобы выполнить задание 2.

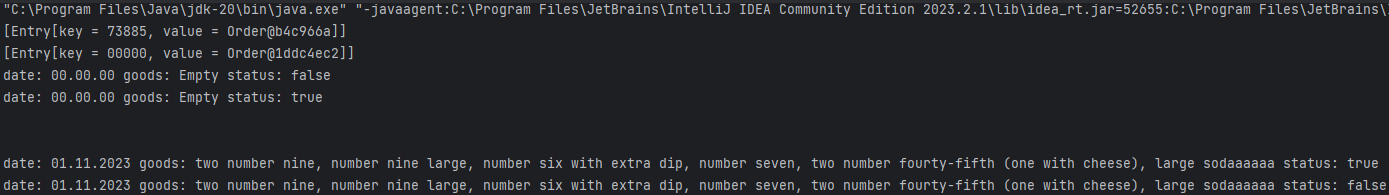
**Мы всего лишь создаем класс Order. Этот класс представляет объект заказа и содержит следующие поля и методы:**

1. **date: поле для хранения даты заказа.**
2. **goods: массив товаров в заказе.**
3. **status: поле, указывающее на статус заказа (завершен или нет).**
4. **Конструкторы Order(), Order(String date, String[] goods, boolean status): конструкторы для создания объекта заказа с заданными значениями полей.**
5. **changeStatus(): метод для изменения статуса заказа.**
6. **getInfo(): метод для вывода информации о заказе на консоль.**

Код:

import java.util.ArrayList;  
import java.util.LinkedList;  
import static java.util.Objects.*hash*;  
  
public class HashTable2 {  
 public static void main(String[] args) {  
 hashTableCode2<Object, Order> table = new hashTableCode2<>();  
  
 Order order = new Order();  
 Order order2 = new Order("01.11.2023", new String[] {"two number nine", "number nine large",  
 "number six with extra dip", "number seven",  
 "two number fourty-fifth (one with cheese)",  
 "large sodaaaaaa"}, true);  
  
 table.put("00000", order);  
 table.put("73885", order2);  
  
 table.println();  
  
 Order value = table.get("00000");  
 Order value2 = table.get("73885");  
  
 value.getInfo();  
  
 value.changeStatus();  
  
 value.getInfo();  
  
 System.*out*.println("\n");  
  
 value2.getInfo();  
  
 value2.changeStatus();  
  
 value2.getInfo();  
 }  
}  
  
class hashTableCode2<K, V> {  
 private final ArrayList<LinkedList<Entry<K, V>>> table;  
  
 hashTableCode2() {  
 this.table = new ArrayList<>();  
 }  
  
 static class Entry<K, V> {  
 K key;  
 V value;  
  
 public Entry(K key, V value) {  
 this.key = key;  
 this.value = value;  
 }  
  
 public K getKey() {  
 return key;  
 }  
  
 public void setValue(V values) {  
 value = values;  
 }  
  
 @Override  
 public String toString() {  
 return "Entry[key = " + key + ", value = " + value + "]";  
 }  
 }  
  
 public void put(K keys, V values) {  
 int index = *hash*(keys) % 16;  
  
 if (index >= table.size()) {  
 for (int i = 0; i <= index; i++) {  
 table.add(null);  
 }  
 }  
  
 if (table.get(index) == null) {  
 table.set(index, new LinkedList<>());  
 }  
  
 for (Entry<K, V> entry : table.get(index)) {  
 if (entry.getKey().equals(keys)) {  
 entry.setValue(values);  
 return;  
 }  
 }  
 table.get(index).add(new Entry<>(keys, values));  
  
 }  
  
 public V get(K key) {  
 int index = *hash*(key) % 16;  
  
 if (index >= table.size()) {  
 System.*out*.println("Такого значения нет.");  
 return null;  
 }  
  
 LinkedList<Entry<K, V>> value = table.get(index);  
  
 if (value == null) {  
 System.*out*.println("Такого значения нет.");  
 return null;  
 }  
  
 for (Entry<K, V> entry : value) {  
 if (entry.getKey().equals(key)) {  
 return entry.value;  
 }  
 }  
  
 System.*out*.println("Такого значения нет.");  
 return null;  
 }  
  
 public void remove(K key) {  
 int index = *hash*(key) % 16;  
  
 LinkedList<Entry<K, V>> value = table.get(index);  
  
 for (int i = 0; i < value.size(); i++) {  
 if (value.get(i).getKey().equals(key)) {  
 value.remove(i);  
 }  
 }  
  
 }  
  
 public int size() {  
 int size = 0;  
  
 for (LinkedList<Entry<K, V>> entry : table) {  
 if (entry != null) {  
 size += entry.size();  
 }  
 }  
  
 return size;  
 }  
  
 public boolean isEmpty() {  
 if (table.size() != 0) {  
 return false;  
 }  
  
 return true;  
 }  
  
 public void println() {  
 for (int i = 0; i < table.size(); i++) {  
 if (table.get(i) != null) {  
 System.*out*.println(table.get(i).toString());  
 }  
 }  
 }  
}  
  
class Order {  
 String date;  
 String[] goods;  
 boolean status;  
  
 public Order() {  
 this ("00.00.00", new String[] {"Empty"}, false);  
 }  
  
 public Order(String date, String[] goods, boolean status) {  
 this.date = date;  
 this.goods = goods;  
 this.status = status;  
 }  
  
 public void changeStatus() {  
 if (!status) {  
 this.status = true;  
 return;  
 }  
  
 this.status = false;  
 };  
  
 public void getInfo() {  
 StringBuilder goodsString = new StringBuilder();  
  
 for (int i = 0; i < this.goods.length; i++) {  
 goodsString.append(this.goods[i]);  
  
 if (i != this.goods.length - 1) {  
 goodsString.append(", ");  
 }  
 }  
  
 System.*out*.println("date: " + this.date + " goods: " + goodsString + " status: " + this.status);  
 }  
}

Вывод программы:



**Вывод**

Мы выполнили все задания лабораторной работы №3. Мы также смогли разобраться с тем, как устроена хэш-таблица, узнали метод для борьбы с коллизией, реализовали своими руками таблицу ключ-значение. Приобретённые знания теперь будут сопутствовать дальнейшим нашим успехам в выполнении следующих работ.