Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа № 1

По дисциплине ЯПИС

За шестой семестр

Тема: «Работа с таблицей идентификаторов»

Вариант 7

Выполнил:

Студент 3 курса

Группы ИИ-16(2)

Пешко А.С.

Проверил:

Слинко Е.В.

Брест, 2021

**Цель работы:** изучить основные методы организации таблиц идентификаторов, получить представление о преимуществах и недостатках, присущих различным методам организации таблиц символов (идентификаторов).

**Ход работы**

**Задание:** для выполнения лабораторной работы требуется написать программу, которая получает на входе набор идентификаторов, организует таблицу по заданному методу и позволяет осуществить поиск идентификатора в этой таблице.

Список идентификаторов задан в виде текстового файла. Длина идентификаторов ограничена 32 символами.

**Вариант 7:** Таблица строится с использованием хеш-функции из варианта №6. Одинаковые элементы помещаются в одну ячейку, внутри которой организуется упорядоченный список.

**Хэш-функция(в варианте №6)** - сумма трех первых букв идентификатора. При этом все буквы переводятся в заглавные (большие).

**Поиск в хэш-таблице**

1. Вычисляется хэш-код n для элемента в соответвесвии с вариантом.
2. Если ячейка хэш-таблицы по адресу n пустая, то элемент не найден и алгоритм завершен, иначе Выбрать из хэш-таблицы адрес ячейки с упорядоченным списком.
3. В списке сравнить имя элемента в ячейке с именем искомого элемента. Если они совпадают, то искомый элемент найден и алгоритм завершен, иначе элемент не найден и алгоритм завершен.

Засчет того, что список изначально упорядоченный, поиск по нему будет происходить быстрее, чем обычно.

Проект с исходным кодом можно найти в репозитории по ссылке: **<https://github.com/anapiashko/university/tree/master/YPIS/lab1>**

**Класс содержащий функцию main**  
public class Main {  
  
 // хэш-таблица  
 static HashTable hashTable = new HashTable();  
  
 public static void main(String[] args) {  
  
 // чтение строк из файла  
 List<String> input = read();  
  
 for (int i = 0; i < input.size(); i++) {  
  
 // функция вставки в таблицу элемента  
 hashTable.insert(input.get(i));  
 }  
  
 // функция вывода элементов таблицы с одинаковым хэшем, таким как у передаваемого элемента  
 hashTable.showListByHash("string");  
 System.out.println();  
  
 // функция search Для поиска элемента в таблице  
 // возвращает boolean, который означает присутсвие элемента в таблице  
 boolean exists = hashTable.search("string");  
 System.out.println("Search \"string\" = " + exists);  
 if (exists){  
  
 System.out.println("Deleting...");  
  
 // функция удаления элемента из таблицы  
 hashTable.delete("string");  
 }  
  
 // опять ищем тот же элемент, только теперь ожидаем результат false  
 System.out.println("Search \"string\" = " + hashTable.search("string"));  
 System.out.println();  
  
 // функция вывода элементов таблицы с одинаковым хэшем, таким как у передаваемого элемента  
 hashTable.showListByHash("string");  
 }  
  
 // построчное считывание файла  
 public static List<String> read() {  
 List<String> input = new ArrayList<>();  
 try {  
 File file = new File("input.txt");  
  
 FileReader fr = new FileReader(file);  
  
 BufferedReader reader = new BufferedReader(fr);  
  
 String line = reader.readLine();  
 while (line != null) {  
 // System.out.println(line);  
 input.add(line);  
  
 line = reader.readLine();  
 }  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return input;  
 }  
}

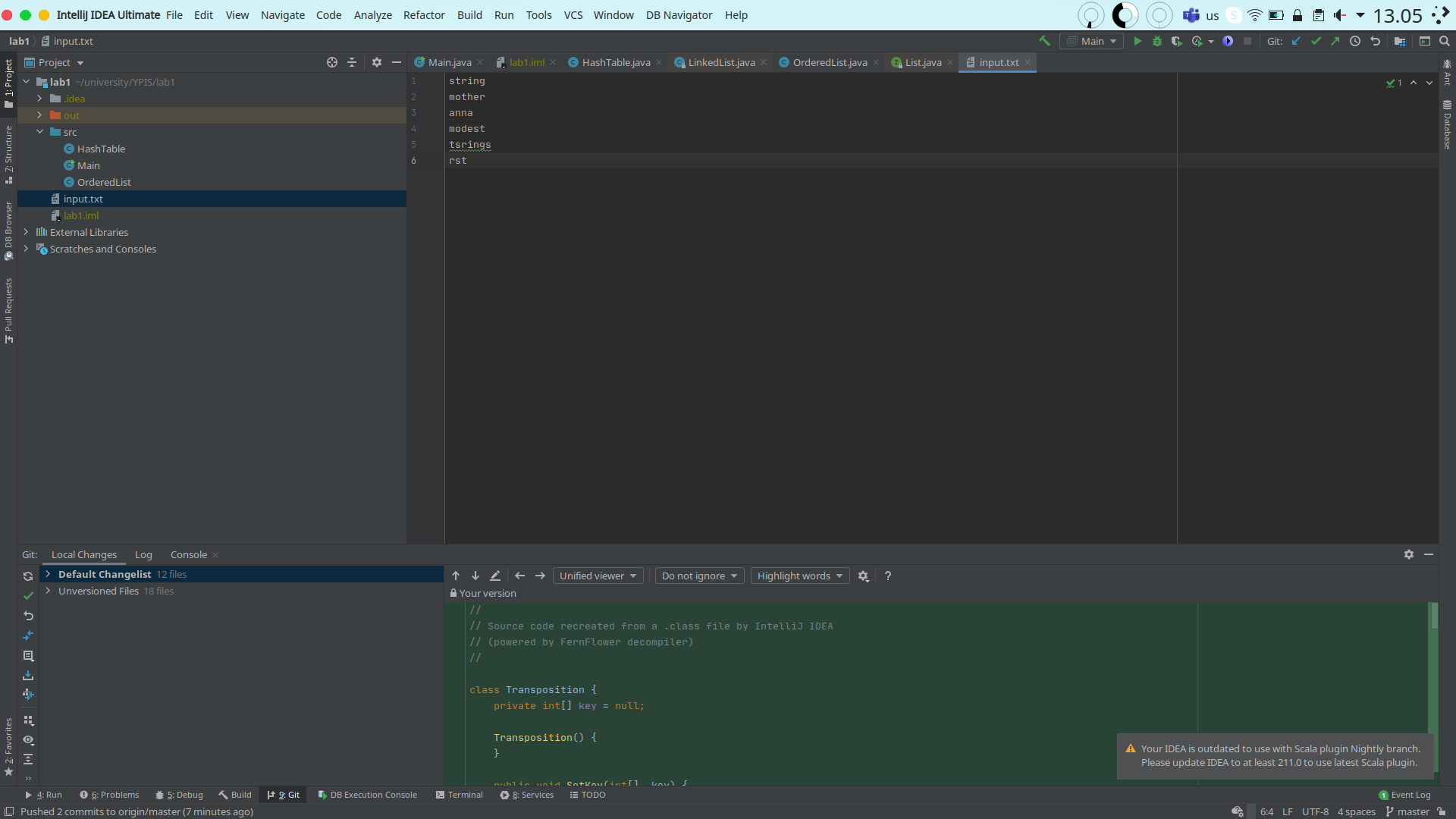
**Класс реализующий хэш-таблицу**

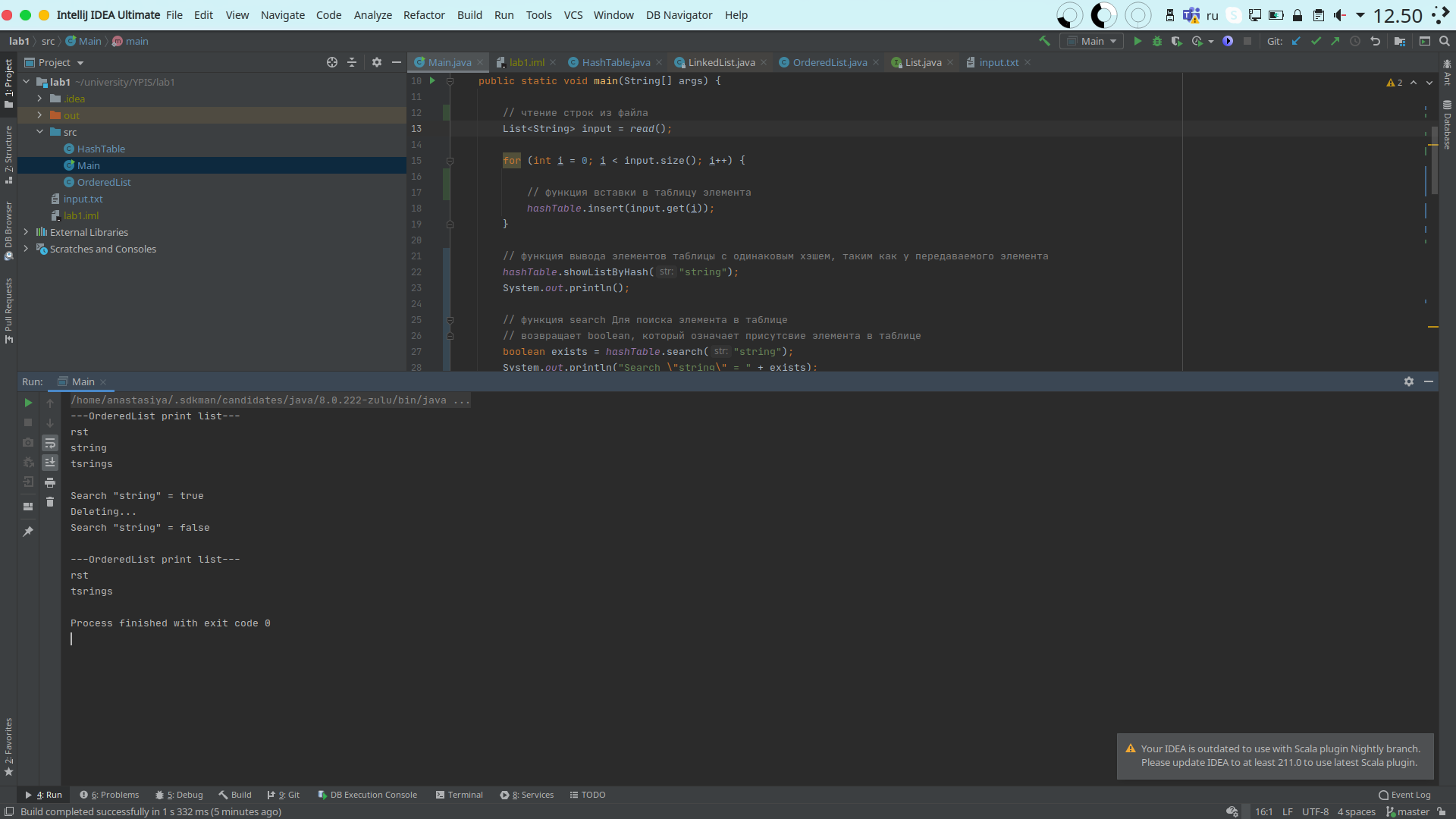
public class HashTable {  
  
 private final int capacity = 50;  
  
 // массив упорядоченых списков  
 private final OrderedList[] table = new OrderedList[capacity];  
  
 public HashTable() {  
 // начальная инициализация пустыми списками  
 for (int i = 0; i < capacity; i++) {  
 table[i] = new OrderedList();  
 }  
 }  
  
 // вставка элемента в таблицу  
 // метод add реализует класс OrderedList  
 public void insert(String str) {  
 int h = hash(str);  
 table[h].add(str);  
 }  
  
 // поик элемента в таблице  
 // возвращает boolean, который означает присутсвие/отсутствие элемента в таблице  
 // метод search реализует класс OrderedList  
 public boolean search(String str) {  
  
 int hash = hash(str);  
  
 int index = table[hash].search(str);  
  
 return index >= 0;  
 }  
  
 // удаление элемента  
 public void delete(String str) {  
  
 int hash = hash(str);  
  
 table[hash].delete(str);  
 }  
  
 // вывод элементов таблицы с одинаковым хэшем, таким как у элемента str  
 public void showListByHash(String str){  
 int hash = hash(str);  
  
 table[hash].print();  
 }  
  
 // хэш-функция  
 int hash(String str) {  
 str = str.toUpperCase();  
 return (str.charAt(0) + str.charAt(1) + str.charAt(2)) % capacity;  
 }  
}

**Класс реализующий упорядоченный список**  
public class OrderedList {  
  
 private Node first, last;  
 private int size = 0;  
  
 public OrderedList() {  
  
 }  
  
 public void add(String data) {  
  
 if (first == null) { // if empty list  
 Node newNode = new Node(data);  
 first = last = newNode;  
 size = size + 1;  
 return;  
 }  
  
 Node node = first;  
  
 // цикл вставки элемента в упорядоченный список  
 int i;  
 for (i = 0; i < size; i++) {  
  
 if (data.charAt(0) <= node.data.charAt(0)) {  
  
 addElement(node.previous, node, data);  
  
 break;  
 }  
  
 if(node.next != null) {  
 node = node.next;  
 }  
 }  
  
 // если необходима вставка в конец  
 if (i == size) {  
 addElement(node, node.previous, data);  
 }  
 size = size + 1;  
 }  
  
 // возвращает индекс элемента в списке  
 public int search(String data) {  
 Node node = first;  
  
 // если список пустой  
 if (first == null) {  
 return -1;  
 }  
  
 for (int i = 0; i < size; i++) {  
 if (data.equals(node.data)) {  
 return i;  
 }  
 node = node.next;  
 }  
 return -1;  
 }  
  
 public void delete(String str) {  
 int index = search(str);  
  
 if (index < 0) {  
 return;  
 }  
  
 // перемещаемся по списку до нужного индекса  
 Node node = first;  
 for (int i = 0; i < index; i++) {  
 node = node.next;  
 }  
  
 // если один элемент в списке  
 if (node.previous == null && node.next == null) {  
 first = last = null;  
 }  
 // если это первый элемент  
 else if (node.previous == null && node.next != null) {  
 first = node.next;  
 node.next.previous = null;  
 }  
 // если это последний элемент  
 else if (node.previous != null && node.next == null) {  
 node.previous.next = null;  
 }  
 // в остальных случаях  
 else {  
 Node temp = node.previous;  
 temp.next = node.next;  
 node.next = temp;  
 }  
 size = size - 1;  
 }  
  
 // добавление элемента в список между prev и next узлами  
 private void addElement(Node prev, Node next, String data) {  
  
 final Node newNode = new Node(prev, next, data);  
  
 if(next != null) {  
 next.previous = newNode;  
 }  
  
 if (prev != null) {  
 prev.next = newNode;  
 } else {  
 first = newNode;  
 }  
 }  
  
 public void print() {  
 System.out.println("---OrderedList print list---");  
 Node node = first;  
 for (int i = 0; i < size; i++) {  
 System.out.println(node.data);  
 node = node.next;  
 }  
 }

**// приватный класс класса OrderedList , описывающий  
 // один элемент списка**  
 private class Node {  
 String data;  
 Node next;  
 Node previous;  
  
 public Node() {  
  
 }  
  
 public Node(String data) {  
 this.data = data;  
 next = null;  
 previous = null;  
 }  
  
 public Node(Node aPrev, Node aNext, String data) {  
 this.data = data;  
 next = aNext;  
 previous = aPrev;  
 }  
 }  
}

**Файл с входными строками input.txt**





**Вывод:** научились строить хэш-таблицы. Разобрал и реализовал метод при котором одинаковые элементы помещаются в одну ячейку, внутри которой организуется упорядоченный список.