|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| *Федеральное государственное бюджетное образовательное учреждение высшего образования* ***«МИРЭА – Российский технологический университет»***  **РТУ МИРЭА** |

**Институт информационных технологий (ИТ)**

**Кафедра инструментального и прикладного программного обеспечения (ИиППО)**

**Дисциплина «Программирование на языке Джава»**

**ОТЧЕТ**

**ПО ПРАКТИЧЕСКОМУ ЗАНЯТИЮ №15, №16**

Выполнил студент группы ИНБО-02-20 Самойленко М. А.

Принял Степанов П.В.

Практические работы выполнены «\_\_\_»\_\_\_\_\_\_\_2020г.

«\_\_\_\_\_\_\_\_\_\_\_\_\_\_» «\_\_\_»\_\_\_\_\_\_\_2020г.

Отметка о выполнении

**Москва – 2020 г.**

**Практическое занятие №15**

**Задание 1**

Разработайте класс, который получает строковое представление телефонного номера в одном из двух возможных строковых форматов:

+<Код страны><Номер 10 цифр>, например “+79175655655” или

“+104289652211” или

8<Номер 10 цифр> для России, например “89175655655”

и преобразует полученную строку в формат:

+<Код страны><Три цифры>–<Три цифры>–<Четыре цифры>

**Задание 2**

В методе main считай с консоли имя файла, который содержит слова, разделенные пробелом. В методе getLine() используя StringBuilder расставьте все слова в таком порядке, чтобы последняя буква данного слова совпадала с первой буквой следующего не учитывая регистр. Каждое слово должно участвовать 1 раз.

**Код программы:**

**Задание 1:**

**Класс Main**

package ru.mirea.java.practice15.ex1w;  
  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 Scanner in = new Scanner(System.*in*);  
 StringBuffer a = new StringBuffer("+104289652211");  
 PhoneNumber k = new PhoneNumber();  
 k.createPhoneNumber(a);  
 System.*out*.println(k.toString());  
 a.delete(0,a.length());  
 a.append("+79175655655");  
 k.createPhoneNumber(a);  
 System.*out*.println(k.toString());  
 a.delete(0,a.length());  
 a.append("89175565565");  
 k.createPhoneNumber(a);  
 System.*out*.println(k.toString());  
 }  
}

**Класс phoneNumber**

package ru.mirea.java.practice15.ex1w;  
import java.lang.StringBuffer;  
public class PhoneNumber {  
 StringBuffer outputString = new StringBuffer("");  
  
 void createPhoneNumber(StringBuffer key){  
 outputString.delete(0,outputString.length());  
 if (key.charAt(0) == '+')  
 key.deleteCharAt(0);  
 outputString.append("+<").append(key.substring(0,key.length()-10)).append("><").append(key.substring(key.length()-10,key.length()-7)).append("><").append(key.substring(key.length()-7,key.length()-4)).append("><").append(key.substring(key.length()-4,key.length())).append(">");  
 }  
  
 @Override  
 public String toString() {  
 return "PhoneNumber{" +  
 outputString +  
 '}';  
 }  
}

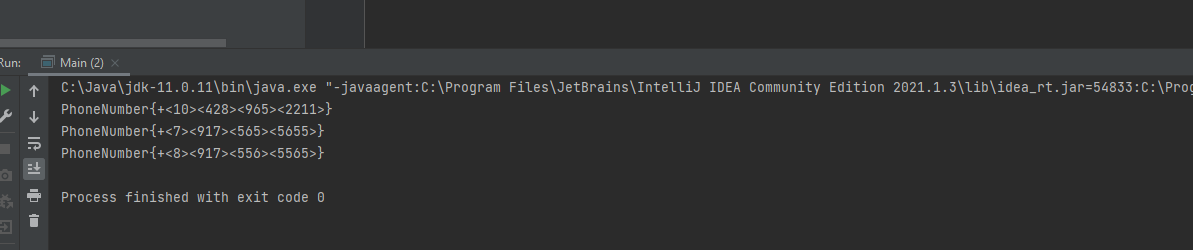
**Задание 2**

**Класс Main**

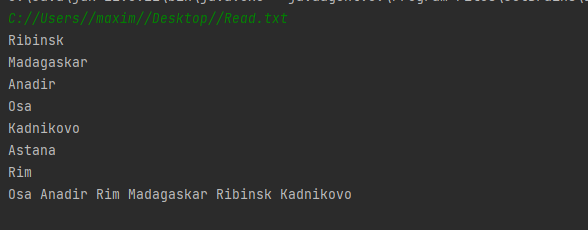
package ru.mirea.java.practice15.ex2;  
  
import java.io.\*;  
import java.util.Scanner;  
  
public class Main {  
 public static StringBuilder getLine(String[] words){  
 StringBuilder ret = new StringBuilder();  
 if(words==null || words.length==0)  
 return ret;  
 ret.append(words[0]);  
 for (int j = 0; j < words.length; j++) {  
  
 for (int i = 1; i < words.length; i++) {  
 if(words[i].length()==0)  
 continue;  
 if(ret.toString().toLowerCase().charAt(0) == words[i].toLowerCase().charAt(words[i].length()-1)){  
 ret.insert(0, words[i] + " ");  
 words[i] = "";  
 }  
 else if(ret.toString().toLowerCase().charAt(ret.length()-1) == words[i].toLowerCase().charAt(0)){  
 ret.append(" " + words[i]);  
 words[i] = "";  
 }  
 }  
  
 }  
 return ret;  
 }  
 public static void main(String[] args) throws IOException {  
 Scanner in = new Scanner(System.in);  
 try (FileInputStream fileInput = new FileInputStream(in.nextLine())) {  
 // C://Users//maxim//Desktop//Read.txt  
 int i = -1;  
 StringBuilder a = new StringBuilder();  
  
 while ((i = fileInput.read()) != -1)  
 a.append((char) i);  
  
 String[] w = a.toString().split(" ");  
 for (int k = 0; k < w.length; k++)  
 System.out.println(w[k]);  
 a = getLine(w);  
 System.out.println(a.toString());  
 } catch (IOException ex) {  
 System.out.println("Не открылся файл");  
 }  
  
 }  
}

**Тесты**

**Задание 1**



**Задание 2**



**Практическое занятие №16**

**Задание**

Создайте класс Drink – напитка. Класс описывает сущность – напиток и характеризуется следующими свойствами - стоимостью, названием и описанием**. Класс должен быть определен как неизменяемый (Immutable class).**

*Конструкторы*:  
−принимающий два параметра – название и описание. Стоимость при этом инициализируется значением 0;  
−принимающий три параметра – стоимость, название и описание.

*Методы*:

−возвращающий стоимость.

− возвращающий название.

−возвращающий описание.

***Дополнительные требования*:**

Вместо литералов в коде (магических констант) необходимо использовать константы класса, содержащие эти значения. Пояснение: в этом случае вы локализуете изменения этих значений в одном месте, а имя константы скажет нам о сути литерала. Этот класс должен быть неизменяемым (правила проектирования таких классов приводятся в лекциях).

Создайте интерфейс Item – для работы с позициями заказа. Интерфейс определяет 3 метода:

−возвращает стоимость.

− возвращает название.

−возвращает описание позиции.

Класс Drink и Dish должны реализовывать этот интерфейс.  
Класс Dish сделайте неизменяемым (аналогично Drink). Order должен хранить (удалять и добавлять) не только экземпляры класса Dish, но и Drink (Для этого разработайте классы Order и TablesOrderManager).

Создайте класс InternetOrder, который моделирует сущность интернет заказ в ресторане или кафе. Класс основан на циклическом двусвязном списке с выделенной головой и может хранить как блюда, так и напитки. Внимание: список реализуется самостоятельно.

*Конструкторы*:

−не принимающий параметров (для списка создается только головной элемент, сам список пуст).

−принимающий массив позиций заказа ( созхдаем список из N позиций.

*Методы*:

−добавляющий позицию в заказ (принимает ссылку типа Item). Пока этот метод возвращает истину после выполнения операции добавления элемента.  
−удаляющий позицию из заказа по его названию (принимает название блюда или напитка в качестве параметра). Если позиций с заданным названием несколько, всегда удаляются последние. Возвращает логическое значение (true, если элемент был удален).

−удаляющий все позиции с заданным именем (принимает название в качестве параметра). Возвращает число удаленных элементов.

−возвращающий общее число позиций заказа (повторяющиеся тоже считаются) в заказе.

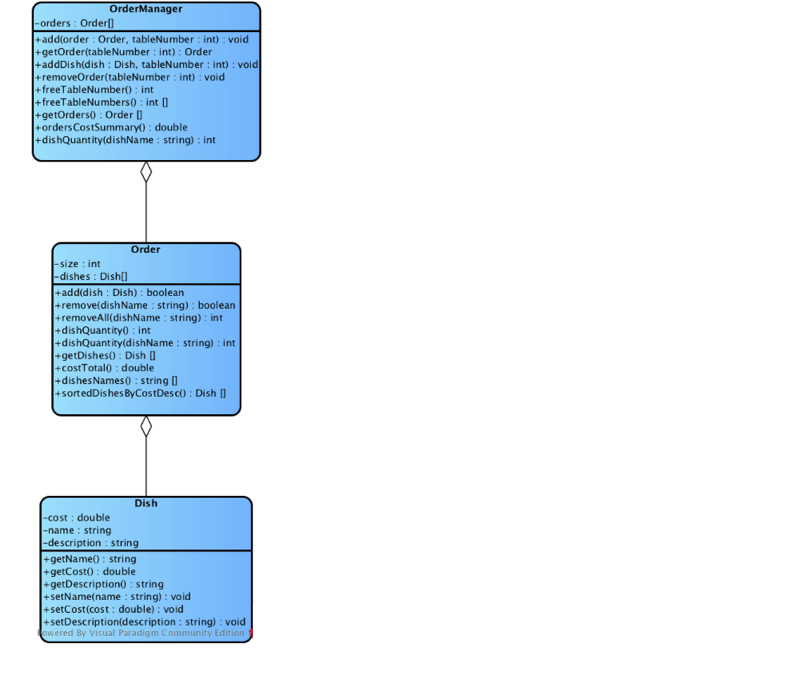
−возвращающий массив заказанных блюд и напитков (*значений null в массиве быть не должно*).

−возвращающий общую стоимость заказа.

−возвращающий число заказанных блюд или напитков (принимает название блюда или напитка в качестве параметра).

− возвращающий массив названий заказанных блюд и напитков (без повторов).

− возвращающий массив позиций заказа, отсортированный по убыванию цены. *Дополнительные требования*:



Переименуйте класс Order из предыдущего задания в RestaurantOrder.  
Создайте интерфейс Order – позиции заказа.

Интерфейс должен определять следующие методы:

−добавления позиции в заказ (принимает ссылку типа *Item*), при этом возвращает логическое значение.

−удаляет позицию из заказа по его названию (принимает название блюда или напитка в качестве параметра). Возвращает логическое значение.

−удаляет все позиции с заданным именем (принимает название в качестве параметра). Возвращает число удаленных элементов.

−возвращает общее число позиций заказа в заказе.

−возвращает массив позиций заказа.

−возвращает общую стоимость заказа.

−возвращает число заказанных блюд или напитков (принимает название в качестве параметра).

−возвращает массив названий заказанных блюд и напитков (без повторов).

−возвращает массив позиций заказа, отсортированный по убыванию цены.

**Замечание:** Классы *InternetOrder* и *RestaurantOrder* должны реализовывать интерфейс *Order*.

Переименуйте класс *TablesOrderManager* в *OrderManager*. Добавьте ему еще одно поле типа *java*.*util*.*HasMap*<*String*, *Order*>, которое содержит пары адрес-заказ, и методы (работающие с этим полем):

Методы класса:

−перегрузка метода добавления заказа. В качестве параметров принимает строку – адрес и ссылку на заказ.

−перегрузка метода получения заказа. В качестве параметра принимает строку – адрес.

−перегрузка метода удаления заказа. В качестве параметра принимает строку – адрес заказа.

−перегрузка метода добавления позиции к заказу. В качестве параметра принимает адрес и *Item*.

−возвращающий массив имеющихся на данный момент интернет-заказов.

−возвращающий суммарную сумму имеющихся на данный момент интернет-заказов.

−возвращающий общее среди всех интернет-заказов количество заказанных порций заданного блюда по его имени. Принимает имя блюда в качестве параметра. Методы должны работать с интерфейсными ссылками *Order* и *Item*.

Создайте объявляемое исключение *OrderAlreadyAddedException*, выбрасываемое при попытке добавить заказ столику или по адресу, если со столиком или адресатом уже связан заказ.

Конструктор классов *Drink* и *Dish* должен выбрасывать исключение java.lang.IllegalArgumentException при попытке создать блюдо или напиток со стоимостью меньше 0, без имени или описания (если параметры имя и описание - пустые строки).

**Код программы**

**Класс Main**

package ru.mirea.java.practice16;  
  
import ru.mirea.java.practice16.Item.Dish;  
import ru.mirea.java.practice16.Item.Drink;  
import ru.mirea.java.practice16.order.InternetOrder;  
import ru.mirea.java.practice16.order.OrderManager;  
import ru.mirea.java.practice16.order.RestaurantOrder;  
  
public class Main {  
 public static void main(String[] args) {  
 OrderManager k = new OrderManager();  
 Dish[] d1 = new Dish[] {new Dish("pervoe", "Вкусная"),  
 new Dish(56,"lecho", "Вкусная"),  
 new Dish(78,"sup", "Вкусная"),  
 new Dish(89,"kartoshka", "Вкусная"),  
 };  
 RestaurantOrder od1 = new RestaurantOrder(d1);  
 k.add(od1,1);  
 k.add("1 стол",od1);  
 String[] result = k.getOrders(1).dishesNames();  
 for (String it: result)  
 if (it != null)  
 System.*out*.println(it);  
 System.*out*.println(k.ordersCostSummary());  
 System.*out*.println("-------------HashMap-------------------");  
 String[] result2 = k.getOrders("1 стол").dishesNames();  
 for (String it: result2)  
 if (it != null)  
 System.*out*.println(it);  
 System.*out*.println(k.ordersCostSummary());  
 System.*out*.println("-------------Drink-------------------");  
 Drink[] d2 = new Drink[] {new Drink("Cola", "Вкусная"),  
 new Drink(50,"Pepsi", "Вкусная"),  
 new Drink(50,"Fanta", "Вкусная"),  
 new Drink(50,"Shveps", "Вкусная"),  
 };  
 InternetOrder od2 = new InternetOrder(d2);  
 String[] result3 = od2.dishesNames();  
 for (String it: result3)  
 if (it != null)  
 System.*out*.println(it);  
 System.*out*.println(od2.costTotal());  
 }  
}

**Класс IllegalTableNumber**

package ru.mirea.java.practice16.exeption;  
  
public class IllegalTableNumber extends Exception{  
}

**Класс OrderAlreadyAddedException**

package ru.mirea.java.practice16.exeption;  
  
public class OrderAlreadyAddedException extends Exception{  
}

**Класс Dish**

package ru.mirea.java.practice16.Item;  
  
public class Dish implements Item {  
 private final int price;  
 private final String name;  
 private final String description;  
 private final int priceZero = 0;  
 public Dish(int price, String name, String description) throws IllegalArgumentException{  
 if(price < 0 || name.equals("") || description.equals(""))  
 throw new IllegalArgumentException();  
 this.price = price;  
 this.name = name;  
 this.description = description;  
 }  
 public Dish(String name, String description) throws IllegalArgumentException{  
 if(name.equals("") || description.equals("")) throw new IllegalArgumentException();  
 this.price = priceZero;  
 this.name = name;  
 this.description = description;  
 }  
  
 @Override  
 public int getPrice() {  
 return price;  
 }  
  
 @Override  
 public String getDescription() {  
 return description;  
 }  
  
 @Override  
 public String getName() {  
 return name;  
 }  
}

**Класс Drink**

package ru.mirea.java.practice16.Item;  
  
public class Drink implements Item {  
 private final int price;  
 private final String name;  
 private final String description;  
 private final int priceZero = 0;  
 public Drink(int price, String name, String description) throws IllegalArgumentException{  
 if(price < 0 || name.equals("") || description.equals(""))  
 throw new IllegalArgumentException();  
 this.price = price;  
 this.name = name;  
 this.description = description;  
 }  
 public Drink(String name, String description) throws IllegalArgumentException{  
 if(name.equals("") || description.equals(""))  
 throw new IllegalArgumentException();  
 this.price = priceZero;  
 this.name = name;  
 this.description = description;  
 }  
  
 @Override  
 public int getPrice() {  
 return price;  
 }  
  
 @Override  
 public String getDescription() {  
 return description;  
 }  
  
 @Override  
 public String getName() {  
 return name;  
 }  
}

**Класс Item**

package ru.mirea.java.practice16.Item;  
  
public interface Item {  
 int getPrice();  
 String getDescription();  
 String getName();  
}

**Класс InternetOrder**

package ru.mirea.java.practice16.order;  
  
import ru.mirea.java.practice16.Item.Item;  
  
import java.util.LinkedList;  
  
public class InternetOrder implements Order {  
 LinkedList<Item> a = new LinkedList<Item>();  
  
 public InternetOrder() {  
 }  
 public InternetOrder(Item[] k) {  
 for (int i = 0; i < k.length;i++)  
 a.add(k[i]);  
 }  
 public boolean add(Item item) {  
 a.add(item);  
 return true;  
 }  
 public boolean remove(String itemName){  
 for (Item it: a) {  
 if (it.getName().equals(itemName)) {  
 a.remove(it);  
 return true;  
 }  
 }  
 return false;  
 }  
 public int itemQuantity(){  
 return a.size();  
 }  
  
 public Item[] getItems(){  
 Item[] items = new Item[a.size()];  
 int i = 0;  
 for (Item it: a) {  
 items[i] = it;  
 i++;  
 }  
 return items;  
 }  
 public double costTotal() {  
 double count = 0;  
 for (Item it: a) {  
 count += it.getPrice();  
 }  
 return count;  
 }  
 public int itemQuantity(String itemName){  
 int count = 0;  
 for (Item it: a) {  
 if (it.getName().equals(itemName))  
 count++;  
 }  
 return count;  
 }  
 public String[] dishesNames(){  
 String[] ret = new String[a.size()];  
 int i = 0;  
 for (Item it: a) {  
 ret[i] = it.getName();  
 i++;  
 }  
 return ret;  
 }  
 public Item[] sortedDishesByCostDesc(){  
 Item[] k = getItems();  
 for (int out = a.size() - 1; out >= 1; out--) //Внешний цикл  
 for (int in = 0; in < out; in++) //Внутренний цикл  
 if(k[in].getPrice() < k[in + 1].getPrice()) {  
 Item t = k[in];  
 k[in] = k[in+1];  
 k[in+1] = t;  
 }  
 return k;  
 }  
}

**Класс Order**

package ru.mirea.java.practice16.order;  
  
import ru.mirea.java.practice16.Item.Item;  
  
public interface Order {  
 public boolean add(Item item);  
 public boolean remove(String itemName);  
 public int itemQuantity();  
 public double costTotal();  
 public Item[] getItems();  
 public int itemQuantity(String itemName);  
 public String[] dishesNames();  
 public Item[] sortedDishesByCostDesc();  
}

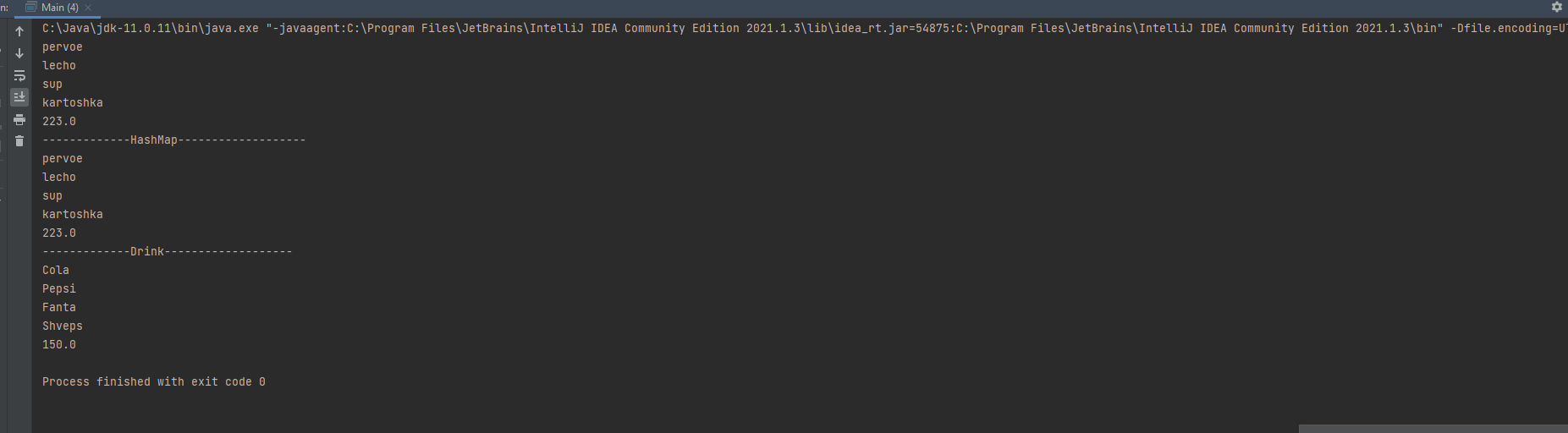
**Класс OrderManager**

package ru.mirea.java.practice16.order;  
  
import ru.mirea.java.practice16.Item.Item;  
import ru.mirea.java.practice16.exeption.IllegalTableNumber;  
  
import java.util.HashMap;  
  
public class OrderManager {  
  
 private Order[] orders = new RestaurantOrder[20];  
 private HashMap<String, Order> hashmap = new HashMap<>();  
  
 public void add(Order order,int tableNumber) {  
 orders[tableNumber] = order;  
 }  
 public void add(String address,Order order) {  
 hashmap.put(address,order);  
 }  
  
 public Order getOrders(int tableNumber) {  
 return orders[tableNumber];  
 }  
 public Order getOrders(String address) {  
 return hashmap.get(address);  
 }  
  
 public void addDish(Item item, int tableNumber) throws IllegalTableNumber {  
 if (tableNumber < 0 || tableNumber >= orders.length) throw new IllegalTableNumber();  
 orders[tableNumber].add(item);  
 }  
 public void addDish(Item item, String adress){  
 hashmap.get(adress).add(item);  
 }  
  
 public void removeOrder(int tableNumber) {  
 orders[tableNumber] = null;  
 }  
 public void removeOrder(String adress) {  
 hashmap.remove(adress);  
 }  
  
 public int freeTableNumber(){  
 for (int i = 0; i < 20;i++){  
 if (orders[i] == null)  
 return i;  
 }  
 return -1;  
 }  
 public int[] freeTabelNumbers(){  
 int[] a = new int[20];  
 int j = 0;  
 for (int i = 0; i < 20;i++)  
 if (orders[i] == null) {  
 a[j] = i;  
 j++;  
 }  
 return a;  
 }  
  
 public Order[] getOrders() {  
 return orders;  
 }  
 public Order[] getHashOrders() {  
 return hashmap.values().toArray(new Order[0]);  
 }  
  
 public double ordersCostSummary() {  
 int count = 0;  
 for (int i = 0; i < 20; i++)  
 if(orders[i] != null)  
 count += orders[i].costTotal();  
 return count;  
 }  
 public double hashCostSummary() {  
 int count = 0;  
 for(Order i: hashmap.values())  
 count += i.costTotal();  
 return count;  
 }  
 public int dishQuantity(String dishName){  
 int count = 0 ;  
 for (int i = 0; i < 20; i++)  
 count += orders[i].itemQuantity(dishName);  
 return count;  
 }  
 public int dishHashQuantity(String dishName){  
 int count = 0 ;  
 for (String i: hashmap.keySet())  
 if (i.equals(dishName)) count += hashmap.get(dishName).itemQuantity(dishName);  
 return count;  
 }  
}

**Класс RestaurantOrder**

package ru.mirea.java.practice16.order;  
  
import ru.mirea.java.practice16.Item.Dish;  
import ru.mirea.java.practice16.Item.Item;  
  
public class RestaurantOrder implements Order {  
 private final int sizeD = 10;  
 private int size = 0;  
 private Dish[] dishes = new Dish[sizeD];  
 public RestaurantOrder(Item[] k) {  
 if (k.length < dishes.length)  
 for (int i = 0; i < k.length;i++) {  
 dishes[i] = (Dish) k[i];  
 size++;  
 }  
 }  
 public boolean add(Item item) {  
 if (size < sizeD) {  
 dishes[size-1] = (Dish)item;  
 size++;  
 return true;  
 } else return false;  
  
 }  
 public boolean remove(String dishName){  
 int i = 0;  
 while (i < size) {  
 if (dishes[i].getName().equals(dishName)) {  
 for (int j = i; j < size-1;j++)  
 dishes[j] = dishes[j+1];  
 dishes[size-1] = null;  
 return true;  
 }  
 i++;  
 }  
 return false;  
 }  
 public int removeAAll(String dishName){  
 int i = 0;  
 int count = 0;  
 while (i < size) {  
 if (dishes[i].getName().equals(dishName)) {  
 for (int j = i; j < size-1;j++)  
 dishes[j] = dishes[j+1];  
 dishes[size-1] = null;  
 count++;  
 }  
 i++;  
 }  
 return count;  
 }  
 public int itemQuantity(){  
 return size;  
 }  
 public int itemQuantity(String dishName) {  
 int count = 0;  
 for (int i = 0;i < size; i++)  
 if (dishes[i].getName().equals(dishName))  
 count++;  
 return count;  
 }  
 public Item[] getItems(){  
 return dishes;  
 }  
 public double costTotal() {  
 double count = 0;  
 for (int i = 0; i < size; i++)  
 count += dishes[i].getPrice();  
 return count;  
 }  
 public String[] dishesNames(){  
 String[] ret = new String[10];  
 for (int i = 0; i < size;i++)  
 ret[i] = dishes[i].getName();  
 return ret;  
 }  
 public Dish[] sortedDishesByCostDesc(){  
 for (int out = size - 1; out >= 1; out--) //Внешний цикл  
 for (int in = 0; in < out; in++) //Внутренний цикл  
 if(dishes[in].getPrice() > dishes[in + 1].getPrice()) {  
 Dish k = dishes[in];  
 dishes[in] = dishes[in+1];  
 dishes[in+1] = k;  
 }  
 return dishes;  
 }  
}

**Тест**



**Выводы**

Я научился строить динамические структуры в java, неизменяемые классы, пользоваться интерфейсами.