|  |
| --- |
|  |
| 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 Зотов Е. 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 Phone**

public class Phone {  
 private String code, city, triplet, quadruple;  
  
 public Phone(String phone){  
 if(phone.startsWith("8")){  
 code = "+7";  
 }else{  
 code = phone.substring(0, phone.length()-10);  
 }  
 city = phone.substring(phone.length()-10, phone.length()-7);  
 triplet = phone.substring(phone.length()-7, phone.length()-4);  
 quadruple = phone.substring(phone.length()-4);  
 }  
  
 @Override  
 public String toString() {  
 return code + city + "-" + triplet + "-" + quadruple;  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println(new Phone("89175655655"));  
 System.*out*.println(new Phone("+79175655655"));  
 System.*out*.println(new Phone("+104289652211"));  
 System.*out*.println(new Phone("+112345678900"));  
 }  
}

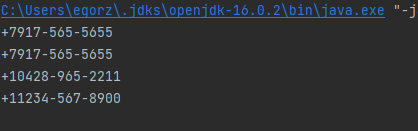
**Задание 2**

**Класс Main**

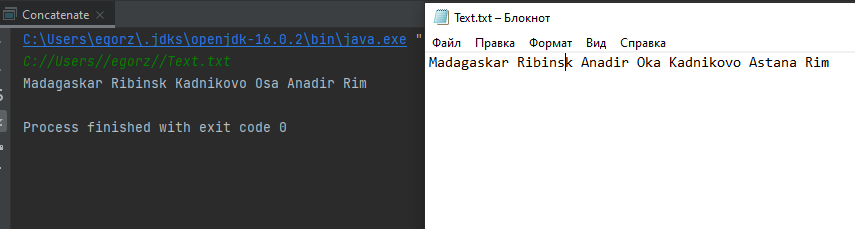
import java.io.FileReader;  
import java.io.IOException;  
import java.util.\*;  
  
public class Concatenate {  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 String path = sc.nextLine();  
 StringBuilder s = new StringBuilder();  
 try(FileReader fr = new FileReader(path)){  
 int c;  
 while((c=fr.read())!=-1){  
 s.append((char) c);  
 }  
 }catch (IOException e){  
 e.printStackTrace();  
 }  
 System.*out*.println(*getLine*(s.toString()));  
 }  
  
 private static String getLine(String s) {  
 Queue<String> toAdd = new ArrayDeque<>(Arrays.*asList*(s.split(" ")));  
 Set<String> added = new HashSet<>();  
 added.add(toAdd.peek());  
 StringBuilder sb = new StringBuilder(toAdd.remove());  
 boolean wasAdded = true;  
  
 while(!toAdd.isEmpty() && wasAdded){  
 wasAdded = false;  
 int i = 0;  
 for (; i < toAdd.size(); i++) {  
 String cur = toAdd.remove();  
 if(added.contains(cur)) continue;  
 if(cur.startsWith(sb.substring(sb.length()-1).toLowerCase()) ||  
 cur.startsWith(sb.substring(sb.length()-1).toUpperCase())){  
 sb.append(" ").append(cur);  
 wasAdded = true;  
 added.add(cur);  
 break;  
 }  
 toAdd.add(cur);  
 }  
 for(; i< toAdd.size(); i++){  
 String cur = toAdd.remove();  
 if(added.contains(cur)) continue;  
 toAdd.add(cur);  
 }  
 }  
 return sb.toString();  
 }  
}

**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.

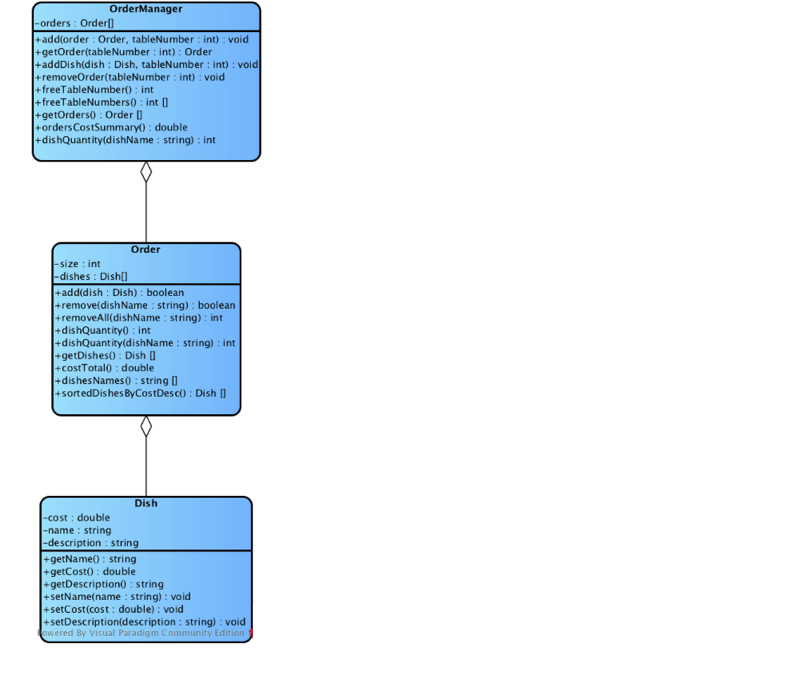
−возвращающий массив заказанных блюд и напитков (*значений null в массиве быть не должно*).

−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**

public class Main {  
 public static void main(String[] args) {  
 OrderManager k = new OrderManager();  
 Dish[] d1 = new Dish[] {new Dish(100,"Суп", "куриный"),  
 new Dish(80,"Борщ", "со сметаной"),  
 new Dish(150,"Стейк", "свиной"),  
 new Dish(120,"Кебаб", "говяжий"),  
 };  
 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();  
 System.*out*.println(" HashMap");  
 System.*out*.println();  
 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();  
 System.*out*.println("Drink");  
 System.*out*.println();  
 Drink[] d2 = new Drink[] {new Drink(40,"Чай", "зеленый"),  
 new Drink(50,"Кофе", "капучино"),  
 new Drink(30,"Пепси", "max"),  
 new Drink(40,"Чай", "черный"),  
 };  
 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 IllegalTableNumberException**

package ru.mirea.practice16;  
  
public class IllegalTableNumberException extends Exception{  
}

**Класс OrderAlreadyAddedException**

public class OrderAlreadyAddedException extends Exception{  
}

**Класс Dish**

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**

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**

public interface Item {  
 int getPrice();  
 String getDescription();  
 String getName();  
}

**Класс InternetOrder**

import java.util.LinkedList;  
  
public class InternetOrder implements Order {  
  
 LinkedList<Item> a = new LinkedList();  
  
 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--)  
 for (int in = 0; in < out; in++)  
 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**

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();  
}

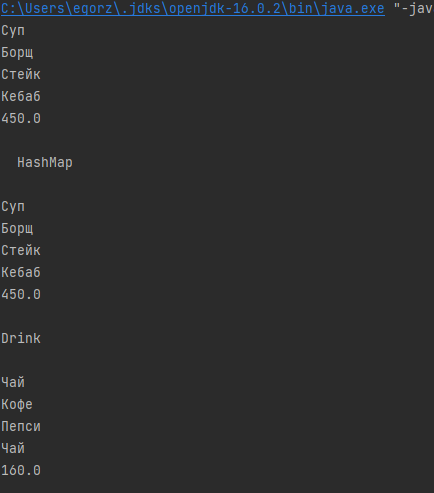
**Класс OrderManager**

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 IllegalTableNumberException {  
 if (tableNumber < 0 || tableNumber >= orders.length) throw new IllegalTableNumberException();  
 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**

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 removeAll(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--)  
 for (int in = 0; in < out; in++)  
 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.