|  |
| --- |
|  |
| 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 ПРАКТИЧЕСКИМ ЗАНЯТИЯМ №15, №16**

Выполнил студент группы ИНБО-02-20 Колмаков Е.Ю.

Принял Степанов П.В.

Практические работы выполнены «\_\_\_»\_\_\_\_\_\_\_2021г.

«\_\_\_\_\_\_\_\_\_\_\_\_\_\_» «\_\_\_»\_\_\_\_\_\_\_2021г.

Отметка о выполнении

**Москва 2021 г.**

**Практическое занятие №15**

**Задание 4**

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>

**Код программы**

**Класс Number**

package Lab12.ex4;  
  
public class Number {  
 private String str, codeCountry, numbers31, numbers32, numbers4;  
 public Number(String num){  
 this.str=num.replace("8","+7");  
 //str.replace("8","+7");  
 codeCountry=str.substring(0,2);  
 numbers31=str.substring(2,5);  
 numbers32=str.substring(5,8);  
 numbers4=str.substring(8,12);  
 }  
 public String toString(){  
 return codeCountry + "-" + numbers31 +"-" + numbers32 + "-" + numbers4;  
 }  
  
 public static void main(String[] args){  
 Number num = new Number("89350920303");  
 System.*out*.println(num.toString());  
 Number num2 = new Number("+19350920303");  
 System.*out*.println(num2.toString());  
 Number num3 = new Number("+39350920303");  
 System.*out*.println(num3.toString());  
  
  
  
 }  
  
}

**Задание 5**

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**

**Класс Main**

package Lab12.ex5;  
  
import java.io.FileReader;  
import java.io.IOException;  
import java.util.\*;  
  
public class Main {  
 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();  
 }  
}

**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.

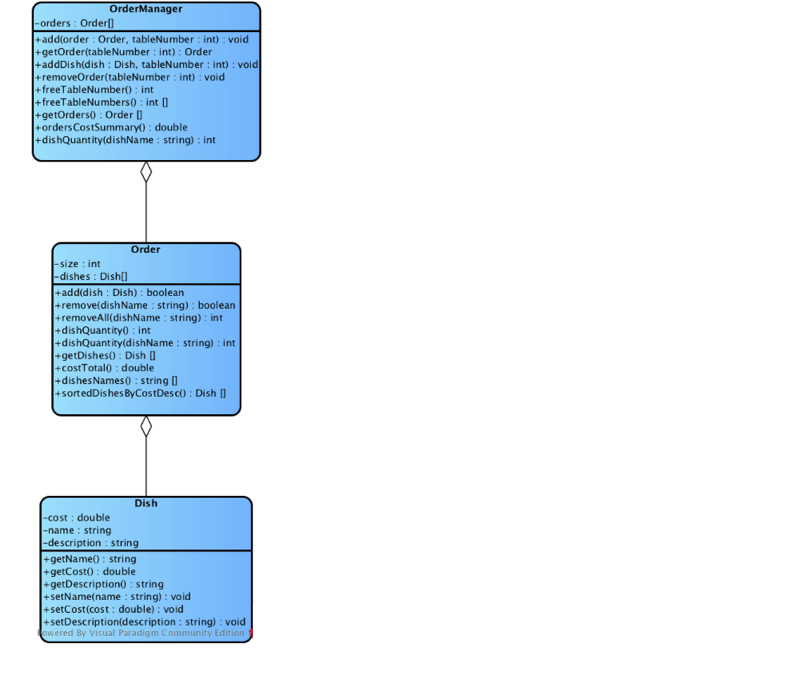
−возвращающий массив заказанных блюд и напитков (*значений null в массиве быть не должно*).

−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**

**Исключение OrderAlreadyAddedException**

package Lab16;  
  
public class OrderAlreadyAddedException extends Exception {  
 public OrderAlreadyAddedException(String message) {  
 super(message);  
 }  
}

**Класс Dish**

package Lab16;  
  
public class Dish {  
}

**Класс Drink**

package Lab16;  
  
public final class Drink implements Item{  
 private int cost;  
 private String name;  
 private Dish description;  
  
 public Drink(int cost, String name, Dish description) {  
 try {  
 this.cost = cost;  
 this.name = name;  
 this.description = description;  
 }  
 catch (IllegalArgumentException ex){  
 if (cost == 0 || name == null || description == null)  
 System.*out*.println(ex);  
 }  
 }  
  
 @Override  
 public int getCost() {  
 return cost;  
 }  
  
 @Override  
 public void setCost(int cost) {  
 this.cost = cost;  
 }  
  
 @Override  
 public String getName() {  
 return name;  
 }  
  
 @Override  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public Dish getDescription() {  
 return description;  
 }  
  
 @Override  
 public void setDescription(Dish description) {  
 this.description = description;  
 }  
}

**Интерфейс Item**

package Lab16;  
  
public interface Item {  
 int getCost();  
 void setCost(int cost);  
 String getName();  
 void setName(String name);  
 Dish getDescription();  
 void setDescription(Dish description);  
}

**Класс InternetOrder**

package Lab16;  
  
public class InternetOrder<Item> {  
 Item info;  
 InternetOrder<Item> next;  
 InternetOrder<Item> last;  
  
 public InternetOrder(Item info) {  
 this.info = info;  
 this.next = null;  
 this.last = null;  
 }  
  
 public InternetOrder(Item info, InternetOrder<Item> next, InternetOrder<Item> last) {  
 this.info = info;  
 this.next = next;  
 this.last = last;  
 }  
}

**Интерфейс Order**

package Lab16;  
  
interface Order {  
 boolean addPosition(Item a);  
 String[] names();  
 int count();  
 int count(String itemName);  
 Item[] getItems();  
 boolean delete(String itemName);  
 int deleteAll(String itemName);  
 Item[] sorted();  
 int cost();  
}

**Класс OrderManager**

package Lab16;  
  
import java.util.HashMap;  
  
public class OrderManager {  
 private HashMap<String, Order> map = new HashMap<>();  
  
 public void addOrder(String address, Order order) throws Exception {  
 if (map.containsKey(address)) {  
 throw new OrderAlreadyAddedException("СТОЛИК ЗАНЯТ!");  
 }  
 map.put(address, order);  
 }  
  
 public Order getOrder(String address) {  
 return map.get(address);  
 }  
  
 public void deleteOrder(String address) {  
 map.remove(address);  
 }  
  
 public void addToOrder(String address, Item a) {  
 map.get(address).addPosition(a);  
 }  
  
 public Order[] getOrdersArray() {  
 return map.values().toArray(new Order[0]);  
 }  
  
 public int getCostt() {  
 int cost = 0;  
 for (Order order : map.values()) {  
 cost += order.cost();  
 }  
 return cost;  
 }  
  
 public int count(String itemName) {  
 int count = 0;  
 for (Order order : map.values()) {  
 count += order.count(itemName);  
 }  
 return count;  
 }  
}

**Класс RestaurantOrder**

package Lab16;  
  
public class RestaurantOrder implements Order {  
 private Item[] items;  
  
 public RestaurantOrder(Item[] items) {  
 this.items = items;  
 }  
  
 @Override  
 public int count() {  
 return items.length;  
 }  
  
 @Override  
 public int count(String itemName) {  
 int count = 0;  
 for (Item item: items) {  
 if (item.getName().equals(itemName)) {  
 count += 1;  
 }  
 }  
 return count;  
 }  
  
 @Override  
 public Item[] getItems() {  
 return items;  
 }  
  
 @Override  
 public int cost() {  
 int cost = 0;  
 for (Item item : items) {  
 cost += item.getCost();  
 }  
 return cost;  
 }  
  
 @Override  
 public boolean addPosition(Item a) {  
 return false;  
 }  
  
 @Override  
 public String[] names() {  
 return new String[0];  
 }  
  
 @Override  
 public boolean delete(String itemName) {  
 return false;  
 }  
  
 @Override  
 public int deleteAll(String itemName) {  
 return 0;  
 }  
  
 @Override  
 public Item[] sorted() {  
 return new Item[0];  
 }  
}

**Выводы**

По ходу данной практической работы у меня получилось построить динамические структуры в java, неизменяемые классы, а также пользоваться интерфейсами.

Git: https://github.com/efim1111/KolmakovJavaINBO-02-20