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 №16

Выполнил студент группы ИНБО-02-20 Лукьяненко Д.В.

Принял Степанов П.В.

Практическая работа выполнена «___»_____2021 г.

«_____» «___»_____2021 г.
Отметка о выполнении

Москва – 2021 г.

СОДЕРЖАНИЕ

Цель работы 3
Задание 3
Репозиторий 5
Выполнение работы 5
Код выполненной работы 7
Вывод 20

Цель работы

Цель данной практической работы – Реализовать ресторан с интернет и личными заказами.

Задание

Задание 1

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.

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).

Задание 2

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).

Задание 3

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 (создаем 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.

Задание 4

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.

Задание 5

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.

Задание 6

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).

Создайте не объявляемое исключение IllegalTableNumber, выбрасываемое в методах, принимающих номер столика в качестве параметра, если столика с таким номером не существует.

Репозиторий

Ссылка: https://github.com/neluckoff/mirea-java-lessons/tree/master/src/ru/luckoff/mirea/practice_16

Выполнение работы

В процессе выполнения поставленных заданий я не стал создавать новые папки с заданиями (task_1, task_2, task_3), как это было в предыдущих моих

работах, а наоборот, изменял уже готовые классы и по ним строилась готовая программа. Результат можно посмотреть в разделе “Код выполненной работы”.

Рисунок 1 – Полученная UML Диаграмма

Код выполненной работы

Здесь в нескольких скриншотах можно увидеть как выглядит код выполненного задания и результат его работы.

```java
public class Dish implements Item {
    private int price;
    private String name;
    private String description;

    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 = 0;
        this.name = name;
        this.description = description;
    }

    @Override
    public int getPrice() { return price; }

    @Override
    public String getDescription() { return description; }

    @Override
    public String getName() { return name; }

    @Override
    public String toString() { return "Блюдо " + name + " стоит $" + getPrice() + ". Описание: " + description; }
}
```

Рисунок 2 – Конечная реализация класса Dish

```java
public class Drink implements Item {
    private int price;
    private String name;
    private String description;

    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 = 0;
        this.name = name;
        this.description = description;
    }

    @Override
    public int getPrice() { return price; }

    @Override
    public String getDescription() { return description; }

    @Override
    public String getName() { return name; }

    @Override
    public String toString() {
        return "Напиток " + name + " стоит $" + getPrice() + ". Описание: " + description;
    }
}
```

Рисунок 3 – Конечная реализация класса Drink

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

Рисунок 4 – Интерфейс Item

```java
public class OrderList<T> {
    private Node<T> front;
    private int size;

    public OrderList() {
        front = null;
        size = 0;
    }

    public void add(T elem) {
        if (isEmpty())
            front = new Node<T>(elem);
        else {
            Node<T> temp = front;
            front = new Node<T>( prev: null, elem, temp);
            front.next.prev = front;
        }
        size++;
    }
```

Рисунок 5 – Класс OrderList и метод add()

```java
public void remove(T elem) {
    if (isEmpty())
        throw new NoSuchElementException("Элемент " + elem.toString() + " не найден");

    if (front.data.equals(elem)) {
        front = front.next;
        return;
    }

    Node<T> current = front;
    while (current != null && !current.data.equals(elem))
        current = current.next;

    if (current == null)
        throw new NoSuchElementException("Элемент " + elem.toString() + " не найден");

    if (current.next != null)
        current.next.prev = current.prev;

    current.prev.next = current.next;
    size--;
}
```

Рисунок 6 – Метод remove() класса OrderList

```java
public boolean isEmpty() {
    return size == 0;
}

public int size() {
    return size;
}

public Node getNext(Node current){
    return current.next;
}

public Node getFront(){
    return front;
}

public static class Node<T> {
    T data;
    Node<T> next;
    Node<T> prev;

    Node(T data) {
        this( prev: null, data,  next: null);
    }

    Node(Node<T> prev, T data, Node<T> next) {
        this.data = data;
        this.next = next;
        this.prev = prev;
    }

    public T getData(){
        return data;
    }
}
```

Рисунок 7 – Оставшиеся методы класса OrderList

```java
public interface Order {
    boolean add(Item item);
    boolean remove(String itemName);
    int itemQuantity();
    double costTotal();
    Item[] getItems();
    int itemQuantity(String itemName);
    String[] itemsNames();
    Item[] sortedItemsByCostDesc();
}
```

Рисунок 8 – Интерфейс Order

```java
public class InternetOrder implements Order {
    OrderList<Item> list = new OrderList<>();

    public InternetOrder() {
    }

    public InternetOrder(Item[] k) {
        for (int i = 0; i < k.length;i++)
            list.add(k[i]);
    }

    public boolean add(Item item) {
        list.add(item);
        return true;
    }

    public boolean remove(String itemName){
        OrderList.Node<Item> current = list.getFront();
        while (current != null) {
            Item it = current.getData();
            if (it.getName().equals(itemName)) {
                list.remove(it);
                return true;
            }
            current = list.getNext(current);
        }
        return false;
    }
```

Рисунок 9 – Класс InternetOrder

```java
public int removeAll(String itemName){
    int count = 0 ;
    OrderList.Node<Item> current = list.getFront();
    while (current != null) {
        Item it = current.getData();
        if (it.getName().equals(itemName)) {
            list.remove(it);
            count++;
        }
        current = list.getNext(current);
    }
    return count;
}

public int itemQuantity() { return list.size(); }

public Item[] getItems(){
    Item[] items = new Item[list.size()];
    int i = 0;
    OrderList.Node<Item> current = list.getFront();
    while (current != null) {
        Item it = current.getData();
        items[i] = it;
        i++;
        current = list.getNext(current);
    }
    return items;
}

public double costTotal() {
    double count = 0;
    OrderList.Node<Item> current = list.getFront();
    while (current != null) {
        Item it = current.getData();
        count += it.getPrice();
        current = list.getNext(current);
    }
    return count;
}
```

Рисунок 10 – Методы класса InternetOrder

```java
public int itemQuantity(String itemName){
    int count = 0;
    OrderList.Node<Item> current = list.getFront();
    while (current != null) {
        Item it = current.getData();
        if (it.getName().equals(itemName))
            count++;
        current = list.getNext(current);
    }
    return count;
}

public String[] itemsNames(){
    String[] ret = new String[list.size()];
    int i = 0;
    OrderList.Node<Item> current = list.getFront();
    while (current != null) {
        Item it = current.getData();
        ret[i] = it.getName();
        i++;
        current = list.getNext(current);
    }
    return ret;
}

public Item[] sortedItemsByCostDesc(){
    Item[] k = getItems();
    for (int out = list.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;
}
```

Рисунок 11 – Оставшиеся методы класса InternetOrder

```java
public class OrderManager {
    private Order[] restaurantOrders = new Order[20];
    private Map<String, Order> internetOrders = new HashMap<>();


    public void add(int tableNumber, Order order) {
        restaurantOrders[tableNumber] = order;
    }

    public void add(String address, Order order) {
        internetOrders.put(address, order);
    }

    public Order getOrders(int tableNumber) {
        return restaurantOrders[tableNumber];
    }

    public Order getOrders(String adress) {
        return internetOrders.get(adress);
    }

    public void addItem(Item item, int tableNumber) throws IllegalTableNumber {
        if (tableNumber < 0 || tableNumber  >= restaurantOrders.length) throw new IllegalTableNumber();
        restaurantOrders[tableNumber].add(item);
    }
```

Рисунок 12 – Класс OrderManager

```java
public void addItem(Item item, String adress){
    internetOrders.get(adress).add(item);
}

public void removeOrder(int tableNumber) {
    restaurantOrders[tableNumber] = null;
}

public void removeOrder(String adress) {
    internetOrders.remove(adress);
}

public int freeTableNumber(){
    for (int i = 0; i < 20;i++){
        if (restaurantOrders[i] == null)
            return i;
    }
    return -1;
}
```

Рисунок 13 – Методы класса OrderManager

```java
public int[] freeTableNumbers(){
    int[] a = new int[20];
    int j = 0;
    for (int i = 0; i < 20;i++)
        if (restaurantOrders[i] == null) {
            a[j] = i;
            j++;
        }
    return a;
}

public Order[] getRestaurantOrders() {
    return restaurantOrders;
}

public Order[] getInternetOrders() {
    return internetOrders.values().toArray(new Order[0]);
}

public double restaurantCostSummary() {
    int count = 0;
    for (int i = 0; i < 20; i++)
        if (restaurantOrders[i] != null) count += restaurantOrders[i].costTotal();
    return count;
}

public double internetCostSummary() {
    int count = 0;
    for(Order i: internetOrders.values())
        count += i.costTotal();
    return count;
}

public int itemRestaurantQuantity(String itemName){
    int count = 0 ;
    for (int i = 0; i < 20; i++)
        if (restaurantOrders[i] != null) {
            count += restaurantOrders[i].itemQuantity(itemName);
        }
    return count;
}
```

Рисунок 14 – Оставшиеся методы класса OrderManager

```java
public class RestaurantOrder implements Order {
    private final int sizeD = 10;
    private int size = 0;
    private Item[] items = new Item[sizeD];

    public boolean add(Item item) {
        if (size < sizeD) {
            items[size] = item;
            size++;
            return true;
        } else return false;

    }

    public boolean remove(String itemName){
        int i = 0;
        while (i < size) {
            if (items[i].getName().equals(itemName)) {
                for (int j = i; j < size - 1;j++)
                    items[j] = items[j + 1];
                items[size-1] = null;
                return true;
            }
            i++;
        }
        return false;
    }
```

Рисунок 15 – Класс RestaurantOrder

```java
public int removeAll(String itemName){
    int i = 0;
    int count = 0;
    while (i < size) {
        if (items[i].getName().equals(itemName)) {
            for (int j = i; j < size - 1;j++)
                items[j] = items[j + 1];
            items[size-1] = null;
            count++;
        }
        i++;
    }
    return count;
}

public int itemQuantity() { return size; }

public int itemQuantity(String itemName) {
    int count = 0;
    for (int i = 0; i < size; i++)
        if (items[i].getName().equals(itemName))
            count++;
    return count;
}

public Item[] getItems() { return items; }

public double costTotal() {
    double count = 0;
    for (int i = 0; i < size; i++)
        count += items[i].getPrice();
    return count;
}
```

Рисунок 16 – Методы класса RestaurantOrder

```java
public String[] itemsNames(){
    String[] ret = new String[size];
    for (int i = 0; i < size; i++)
        ret[i] = items[i].getName();
    return ret;
}

public Item[] sortedItemsByCostDesc(){
    for (int out = size - 1; out >= 1; out--)
        for (int in = 0; in < out; in++)
            if(items[in].getPrice() > items[in + 1].getPrice()) {
                Item k = items[in];
                items[in] = items[in + 1];
                items[in + 1] = k;
            }
    return items;
}
```

Рисунок 17 – Оставшиеся методы класса RestaurantOrder

```java
public class IllegalTableNumber extends Exception {
    public IllegalTableNumber() {
        super("Столика с таким номером не существует");
    }
}
```

Рисунок 18 – Класс IllegalTableNumber

```java
public class OrderAlreadyAddedException extends Exception {
    public OrderAlreadyAddedException() {
        super("Заказ уже добавлен");
    }
}
```

Рисунок 19 – Класс OrderAlreadyAddedException

```java
public class Main {
    public static void main(String[] args) {
        Item drink1 = new Drink( price: 5,  name: "Coca-Cola",  description: "Сладкий газированный напиток");
        Item dish1 = new Dish( price: 25, name: "Суп \"Чучвара\"", description: "Томатный суп с пельменями");
        Item drink2 = new Drink( price: 10,  name: "Sprite",  description: "Сладкий газированный напиток");
        Item drink3 = new Drink( price: 15,  name: "Fanta",  description: "Сладкий газированный напиток");

        OrderManager orderManager = new OrderManager();

        InternetOrder internetOrder = new InternetOrder();
        internetOrder.add(drink1);
        internetOrder.add(drink2);

        RestaurantOrder restaurantOrder = new RestaurantOrder();
        restaurantOrder.add(drink3);
        restaurantOrder.add(dish1);

        RestaurantOrder restaurantOrder2 = new RestaurantOrder();
        restaurantOrder2.add(dish1);
        restaurantOrder2.add(dish1);
        restaurantOrder2.add(dish1);

        orderManager.add( address: "Москва, ул. Тверская, д. 2", internetOrder);
        orderManager.add( address: "Москва, пр. Вернадского, д. 6", internetOrder);
        orderManager.add( tableNumber: 0, restaurantOrder2);
        orderManager.add( tableNumber: 2, restaurantOrder);

        System.out.println("Количество в Ресторанных заказах - Coca-Cola: " + orderManager.itemRestaurantQuantity( itemName: "Coca-Cola"));
        System.out.println("Количество в Ресторанных заказах - Суп \"Чучвара\": " + orderManager.itemRestaurantQuantity( itemName: "Суп \"Чучвара\""));
        System.out.println("Количество в Интернет заказах - Суп \"Чучвара\": " + orderManager.itemInternetQuantity( itemName: "Суп \"Чучвара\""));
        System.out.println("Интернет сумма - $" + orderManager.internetCostSummary());
        System.out.println("Ресторанная сумма - $" + orderManager.restaurantCostSummary());
        System.out.println("Следующий свободный стол - №" + orderManager.freeTableNumber());
        System.out.println("Свободные столы - " + Arrays.toString(orderManager.freeTableNumbers()));
    }
}
```

Рисунок 20 – Полученный класс Main

```
Количество в Ресторанных заказах - Coca-Cola: 0
Количество в Ресторанных заказах - Суп "Чучвара": 4
Количество в Интернет заказах - Суп "Чучвара": 0
Интернет сумма - $30.0
Ресторанная сумма - $115.0
Следующий свободный стол - №1
Свободные столы - [1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 0, 0]

Process finished with exit code 0
```

Рисунок 21 – Результат запуска программы

Вывод

В результате выполнения данной практической работы я смог создать программу, имитирующую работу реального ресторана на языке программирования Java.