|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2** | |
| **по дисциплине** | |
| **«**Объектно-ориентированное программирование**»**  **Тема: «Классы. Диаграммы классов в языке UML»** | |
|  | |
| Выполнил студент группы ИКБО-07-18 | Мишанов А.Б. |
| Принял преподаватель | Колесников С.Н. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_201\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_201\_\_ г. | *(подпись руководителя)* |

**Задание 1**

Разработать приложение со структурой классов варианта

**Требования к заданию:**

1. Реализовать классы в соответствии с предложенной диаграммой.
2. В каждый класс включите конструкторы, необходимые для заполнения атрибутов класса (реализуйте проверку переданных в конструктор параметров).
3. Рекомендации по доработке функциональности программы:
   1. В базовой версии программы все объекты, предусмотренные в ней (пользователи, заявки, вопросы тестов и т.д.) могут только создаваться, но не редактироваться. Однако реальная программа, конечно, должна содержать возможность просматривать и изменять параметры существующих объектов (например, изменить пароль пользователя, поменять заявку до того, как она была обработана и т.д.). Рекомендуется реализовать соответствующую возможность. Для этого нужно разработать интерфейс, позволяющий выбрать один из существующих объектов и изменить его, а также добавить в нужный класс методы для изменения соответствующих атрибутов.
   2. Классы, необходимые для построения пользовательского интерфейса, на предложенных диаграммах отсутствуют. В целом предложенные диаграммы не являются догмой и могут быть усовершенствованы (или вообще изменены).
   3. В каждой программе есть пользователи двух типов: администратор и клиенты. Предполагается, что администратор в программе один. Он должен быть добавлен сразу после первого запуска программы.

**Вариант 2:**

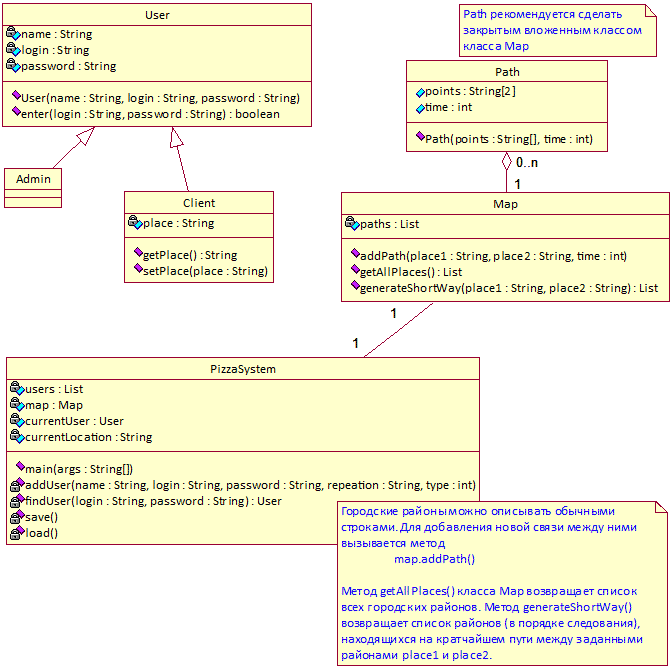
# Обработка заказов на доставку пиццы

Написать программу для составления оптимального маршрута разносчика пиццы. Пользователи, желающие заказать пиццу, оставляют в системе заявку, указывая район своего проживания. Кроме того, система постоянно накапливает информацию о способах передвижения между соседними районами города. Когда поступает заказ, программа должна найти короткий путь от текущего местонахождения вагончика с пиццей до района проживания заказчика.

Основные алгоритмы

*Нахождение кратчайшего маршрута.*

Рекомендуемая диаграмма классов



Интерфейсы:

* Интерфейс для создания «карты» (на самом деле это просто граф) города.
* Интерфейс для входа в систему.
* Интерфейс для регистрации пользователей.
* Интерфейс для заполнения заявки.

Интерфейс для обработки результатов.

Таблицы данных:

* Пользователи
* Карта города (районы и связи между ними)

**Тестирование:**

Было проведено тестирование всех функций приложения. Данное тестирование показало корректность работы всех функции. Программа добавляла маршруты и пользователей, позволяла входить в систему и получать кратчайший маршрут от местоположения курьера до местоположения клиента в системе

**Исходный код**

Класс User

**package** com.company;  
  
**import** java.io.Serializable;  
  
**public class** User **implements** Serializable {  
 **private** String **name**;  
 **private** String **login**;  
 **private** String **password**;  
 User(){  
 **name**=**null**;**login**=**null**;**password**=**null**;  
 }  
 User(String name, String login, String password){  
 **this**.**name**=name; **this**.**login**=login; **this**.**password**=password;  
 }  
 **public boolean** enter(String login, String password){  
 **return** (**this**.**login**==login & **this**.**password**==password);  
 }  
 **public** String getLogin() {  
 **return login**;  
 }  
 **public** String getPassword() {  
 **return password**;  
 }  
 **public** String getPlace(){ **return null**; }  
}  
  
**class** Client **extends** User{  
 **private** String **place**;  
 Client(String n, String l, String p, String place){  
 **super**(n,l,p);  
 **this**.**place**=place;  
 }  
 @Override  
 **public** String getPlace(){  
 **return place**;  
 }  
 **public void** setPlace(String s){  
 **place**=s;  
 }  
}

*//this.Out();* }  
}

Класс Map

**package** com.company;  
  
**import** java.awt.\*;  
**import** java.util.ArrayList;  
**import** java.util.Arrays;  
**import** java.io.\*;  
**import** java.util.HashMap;  
  
**public class** Map **implements** Serializable{  
 *//Вложенный класс путь* **private class** Path **implements** Serializable{  
 **private** String [] **points**;  
 **private int time**;  
 Path(String points[],**int** time){  
 **this**.**points**=points;  
 **this**.**time**=time;  
 }  
 **public** String showPath(){  
 **return** (**points**[0]+**" "**+**points**[1]+**" "**+**time**);  
 }  
 }  
 **private** ArrayList<Path> **paths**;  
 **private** ArrayList **points**;  
 **private int ways**[][];  
 Map(){  
 **paths**=**new** ArrayList<>(); *//массив путей* **points**=**new** ArrayList(); *//массив мест* **ways**=**new int**[100][100]; *//матрица смежности путей. Отражает, есть ли путь(ребро) между двумя точками. Если путя нет, то равняется бесконечности (в рамках программы - Max\_int)* **for** (**int** i=0;i<100;i++)  
 **for**(**int** j=0;j<100;j++)  
 **ways**[i][j]=Integer.***MAX\_VALUE***;  
 }  
 **public void** addPath(String place1,String place2,**int** time){  
 String[] arr=**new** String[2];  
 arr[0]=place1; arr[1]=place2;  
 Path path=**new** Path(arr,time);  
 **paths**.add(path);  
 *//Проверяем наличие точки в массиве мест, если этого места нет, то добавляем его* **if** (!**points**.contains(place1))  
 **points**.add(place1);  
 **if** (!**points**.contains(place2))  
 **points**.add(place2);  
 *//Добавляем путь в матрицу смежности* **ways**[**points**.indexOf(place1)][**points**.indexOf(place2)]=time;  
 **ways**[**points**.indexOf(place2)][**points**.indexOf(place1)]=time;  
  
 }  
 **public** ArrayList getAllPlace(){  
 **return paths**;  
 }  
  
 **public** ArrayList genShortWay(String place1, String place2){  
  
 **int** MAX=Integer.***MAX\_VALUE***; *//Бесконечность. Нужна для проверки наличия ребра* **int** pointCount=**points**.size(); *//Количетсво мест* **int** D[]=**new int**[pointCount]; *//массив меток мест* **int** P[]=**new int**[pointCount]; *//массив точек* **int** startPoint=**points**.indexOf(place1); *//стартовая точка* **boolean** []visited=**new boolean**[pointCount]; *//массив посещенных точек (индекс соответствует индексу точки в массиве points)  
 //Заполняем D, P и visited значениями по умолчанияю* Arrays.*fill*(D,MAX); Arrays.*fill*(P,-1);  
 Arrays.*fill*(visited,**false**);  
 D[startPoint]=0; *//устанавливаем метку стартовой точки за 0* **for** (**int** i=0;i<pointCount;i++)  
 D[i]=**ways**[startPoint][i]; *//устанавливаем метки на все остальные точки, на основе наличия расстояния до начальной метки (если прямого пути(ребра) нет, то бесконечность)* **int** index=0, u=0, min; *//index - индекс точки, метка которой имеет минимальное значение  
  
 //алгоритм Дейсктры* **for** (**int** i=0;i<pointCount;i++){  
 min=MAX;  
 **for** (**int** j=0;j<pointCount;j++){  
 **if** (!visited[j] && D[j]<min){  
 min=D[j]; index=j;  
 }  
 }  
 u=index;  
 visited[u]=**true**;  
 **for** (**int** j=0;j<pointCount;j++){  
 **if** (!visited[j] && **ways**[u][j]!=MAX && D[u]!=MAX && (D[u]+**ways**[u][j]<D[j])){  
 D[j]=D[u]+**ways**[u][j];  
 P[j]=u;  
 }  
 }  
 }  
 *//расложение маршрута* ArrayList stack=**new** ArrayList();  
 **int** target=**points**.indexOf(place2);  
 **for** (**int** v=target;v!=-1;v=P[v])  
 stack.add(0,v);  
 ArrayList sp=**new** ArrayList(stack.size());  
 **for** (**int** i=0;i<stack.size();i++)  
 sp.add(stack.get(i));  
 ArrayList res= **new** ArrayList();  
 res.add(place1);  
 **for** (**int** i=0; i<sp.size();i++)  
 res.add(**points**.get((Integer)sp.get(i)));  
 **return** res;  
 }  
 **public** String show(){  
 String res=**""**;  
 **for** (**int** i=0; i<**paths**.size();i++){  
 res+=**paths**.get(i).showPath()+**"\n"**;  
 }  
 **return** res;  
 }  
}

Класс PizzaSystem

**package** com.company;  
  
**import** java.io.\*;  
**import** java.sql.SQLOutput;  
**import** java.util.ArrayList;  
**import** java.util.Scanner;  
  
**public class** PizzaSystem {  
 **private static** Map *map*;  
 **private static** ArrayList<User> *users*;  
 **private static** User *currentUser*;  
 **private static** String *currentLocation*;  
 **public static void** main(String[] args)**throws** IOException, ClassNotFoundException {  
 *users* = **new** ArrayList<>();  
 *map* = **new** Map();  
 *currentUser* = **new** User();  
 *currentLocation* = **""**;  
 Client a = **new** Client(**"Alex"**, **"A"**, **"1234"**, **"Москва"**);  
 Client b = **new** Client(**"Ivan"**, **"I"**, **"1234"**, **"Подольск"**);  
   
 *currentLocation* = **"Подольск"**;  
 *currentUser* = a;  
 Scanner in = **new** Scanner(System.***in***);  
 System.***out***.println(**"1.Добавить пользователя"** + **'\n'** + **"2.Добавить путь"** + **'\n'** + **"3.Войти в систему"** + **'\n'** + **"4.Ввести место курьера"** + **'\n'** + **"5.Оформить заказ(получить путь и время)"**);  
 **int** n = 0;  
 String str;  
 **while** (n != 8) {  
 System.***out***.println(**"Выберите действие"**);  
 n = in.nextInt();  
 **switch** (n) {  
 **case** 1:  
 System.***out***.println(**"Введите тип (1-админ и ничего не делает)"**);  
 **int** t = in.nextInt();  
 in.nextLine();  
 System.***out***.println(**"Введите имя"**);  
 String s1 = in.nextLine();  
 System.***out***.println(**"Введите логин"**);  
 String s2 = in.nextLine();  
 System.***out***.println(**"Введите пароль"**);  
 String s3 = in.nextLine();  
 **if** (t != 1) {  
 System.***out***.println(**"Введите место"**);  
 String s4 = in.nextLine();  
 *users* = *addClient*(*users*, s1, s2, s3, s4);  
 } **else** *users* = *addUser*(*users*, s1, s2, s3, s3);  
 **break**;  
 **case** 2:  
 System.***out***.println(**"Введите первое место"**);  
 in.nextLine();  
 String place1 = in.nextLine();  
 System.***out***.println(**"Введите второе место"**);  
 String place2 = in.nextLine();  
 System.***out***.println(**"Введите время"**);  
 **int** time = in.nextInt();  
 *map*.addPath(place1, place2, time);  
 **break**;  
 **case** 3:  
 System.***out***.println(**"Введите логин"**);  
 in.nextLine();  
 String login = in.nextLine();  
 System.***out***.println(**"Введите пароль"**);  
 String password = in.nextLine();  
 **if** (*findUser*(*users*, login, password) != **null**) {  
 *currentUser* = *findUser*(*users*, login, password);  
 System.***out***.println(**"Вы вошли в систему как "** + *currentUser*.getLogin());  
 **if** (*currentUser*.getPlace() != **null**)  
 System.***out***.println(**"Ваше местоположение: "** + *currentUser*.getPlace());  
 } **else** System.***out***.println(**"Неверные данные"**);  
 **break**;  
 **case** 4:  
 System.***out***.println(**"Введите текущую локацию"**);  
 in.nextLine();  
 String loc = in.nextLine();  
 *currentLocation* = loc;  
 **break**;  
 **case** 5:  
 **if** (*currentUser* == **null** || *currentLocation* == **""** || *currentUser*.getPlace() == **null**) {  
 System.***out***.println(**"Необходимо войти в систему и ввести текущее место курьера"**);  
 **break**;  
 }  
 System.***out***.println(*map*.genShortWay(*currentLocation*, *currentUser*.getPlace()).toString());  
 **case** 6:  
 *save*();  
 **break**;  
 **case** 7:  
 *load*();  
 **break**;  
 }  
  
 }  
 }  
  
 **public static** ArrayList<User> addUser(ArrayList<User> users, String name, String login, String password, String rep) {  
 users.add(**new** User(name, login, password));  
 **return** (users);  
 }  
  
 **public static** ArrayList<User> addClient(ArrayList<User> users, String name, String login, String password, String place) {  
 users.add(**new** Client(name, login, password, place));  
 **return** (users);  
 }  
  
 **public static** User findUser(ArrayList<User> users, String login, String password) {  
 **for** (User i : users  
 ) {  
 String s1 = i.getLogin();  
 String s2 = i.getPassword();  
 **if** ((s1.equals(login)) && (s2.equals(password)))  
 **return** i;  
 }  
 **return null**;  
 }  
 **public static void** save()**throws** IOException {  
 File f = **new** File(**"map.txt"**);  
 f.createNewFile();  
 ObjectOutputStream out = **new** ObjectOutputStream(**new** FileOutputStream(**"map.txt"**));  
 out.writeObject(*map*);  
 out = **new** ObjectOutputStream(**new** FileOutputStream(**"users.txt"**));  
 out.writeObject(*users*);  
 out.close();  
 }  
 **public static void** load() **throws** IOException, ClassNotFoundException{  
 ObjectInputStream in = **new** ObjectInputStream(**new** FileInputStream(**"map.txt"**));  
 *map* = (Map) in.readObject();  
 in = **new** ObjectInputStream(**new** FileInputStream(**"users.txt"**));  
 *users* = (ArrayList<User>) in.readObject();  
 in.close();;  
 }  
}