Assignment 5

Problems:

Problem 1: Design a class named User [Easy, 15 marks]

Problem 2: Design a class named Office [Easy, 15 marks]

Problem 3: Design a class named Group [Easy, 20 marks]

Problem 4: Design a class named Log [Easy, 10 marks]

Problem 5: Design a class named Data [medium, 40 marks]

Notes for this assignment

1. Use the language specified (Java) in the lab / problem description.
2. Submit more than one .java files if necessary to invoke methods from other class. You don't need to change the class names to Main. And the class name should be definitely the same as above, while you can also design other class or other data fields and methods in above classes if necessary.
3. DO NOT include package in your submission.
4. Submit solutions before deadline, and don't wait until the last minute. The server might be overloaded and your submission will wait a long time before getting judged.
5. The required method of problem 4 must be implemented, otherwise it will affect your score.
6. Finish problems by yourself and enjoy!(≧▽≦)/

Background:

To prevent COVID-19, a Campus Access Management System need be developed in SUSTech. Everyone to get into the campus and offices should be checked the identity to get the permit and taken the body temperature. It can also show the statistic result of user accessing records, such as the total amount of people accessed the campus, people accessed the office in a day or a week.

The Campus Access Management System runs as following:

(1) Initialize the data, including users, groups, offices, and set the date to write some accessing logs for several days.

(2) When the Campus Access Management System runs, a user prepares to access the campus and office, first to check if he/she is qualified. If true, then write the accessing log.

(a) Access campus for who lives outside: it is allowed only if he/she came back to Shenzhen over 14 days and the body temperature is below 37.3 ℃.

(b) Access campus for who lives in campus: always ok. But if the body temperature is not below 37.3℃, it is allowed to enter the campus and live in separated area but not allowed to access any office.

(c) Access offices: After entered campus, only if he/she is working in the office and it has enough room in office. For example, if it can hold 5 people working in the office, and there is already 5 people now, a new user can not enter the office. Especially if a user has got into the office in a day, and goes out then go back to the office again, it is always allowed in the day.

(3) Get some statistic result from the accessing log.

Problem 1: User

Design a class named **User**:
private String id;// primary key, not null, not duplicate, eg:11911234; for visitors outside campus, it should be national ID card number
private String name;//
private int gender;// 0-female, 1-male, 2-other
private Role role;//enum Role{TEACHER, STAFF, UNDERGRADUATE, GRADUATE,VISITOR}
private ResidentialCollege residentialCollege;// enum ResidentialCollege{ ZHIREN, SHUREN, ZHICHENG, SHUDE, ZHIXIN, SHULI}
private String department;//eg: “ComputerScienceEngineering”
private String district;// “SUSTech” for in campus, other for outside campus
private String domain;//eg: “Lychee Park”
private int buildingNo;//building number, eg: 2
private int roomNo;//room number, eg: 101
private LocalDate dateBackToShenzhen;// the date back to Shenzhen
private boolean isValid=true;// false if the user is deleted
private double bodyTemperature;// temperature today, 0- not checked. A user should only take temperature once per day.
private LocalDateTime temperatureTestTime;

```
public User(String id, String name, int gender)
```

Constructor with id, name, gender

Some get/set methods

```
public void takeTemperature(double temperature, LocalDateTime dateTime)
```

//to set bodyTemperature and temperatureTestTime

```
@Override
public String toString()
```

//contents returned see UserJunitTest.java

Problem 2: Office
Design a class named **Office**. Office is the working area for groups. A group may have one or more offices, and there may be one or more groups in an office. A user working in the office should have at least 2 ㎡, so no one can enter the office if it has already enough people.

private int id;//primary key, not null, not duplicate; increase by 1 from 1 with total count of offices as default when creating an office object
private String domain;//eg:”Lychee Park”
private int buildingNo;//building number, eg: 6
private int roomNo;//room number, eg: 402
private double area;// area of the room
private boolean isValid=true;// false if the office is deleted
private static int count;// total count of offices created, including deleted ones

private int visitorCountToday;// set as 0 for everyday, ++1 when a user entered; not consider user get out. A user gets in office several times in a day should only count once.

```
public Office(String domain, int buildingNo, int roomNo, double area)
```

Constructor with domain, buildingNo, roomNo, area;////to increase Office.id by 1 with count

Some get/set methods

```
@Override
public String toString()
```

Problem 3: Group

Design a class named **Group** for research project's group. There are one PI user and some teachers and students, including undergraduates and graduates. It has one or more offices. A user can be only in one group, or not in any group.

private int id;//increase from 1 by 1 with total count of groups
private String piId;// userId of PI, should not in userList, but can access their offices
public ArrayList<User> userList;// to store all group members
public ArrayList<Office> officeList;
// to store all offices of the group, users in the group can access any office in the list
private boolean isValid=true;// false if the group is deleted
private static int count;// total count of groups created

```
public String addUser(User user)
```

return "user already exists";// if the userId of user to add already exists
return "addUser succeed";// otherwise add user to userList

```
public String deleteUser(String userId)
```

return "user not exists";// if the user to delete not exists
return "deleteUser succeed";// otherwise delete the user from group's userList, not the Data.userList

```
public String addOffice(Office office)
```

return "office already exists";// if the officeId of office to add already exists
return "addOffice succeed";// otherwise add office to officeList

```
public String deleteOffice(int officeId)
```

return "office not exists";// if the office to delete not exists
return "deleteOffice succeed";// otherwise delete the office from group's officeList, not the Data.officeList

```
public Group(String pild)
```

Constructor with piId;

//(1) set this.piId=piId. (2) increase group.id from 1 by 1 with count. (3)Set count ++.(4)set this.isValid=true;

Some get/set methods

```
@Override
public String toString()
```

Problem 4: Log

Design a class named **Log** for accessing records.

private int id;//increase from 1 by 1 with total count of logs

private String userId;

private int officeId; //0 for campus, others for offices

private double temperature;//

private LocalDateTime dateTime;//

private static int count;// total count of logs created

```
public Log( String userId, int officeId, double temperature, LocalDateTime
dateTime)
```

Constructor with userId, officeId, temperature, dateTime;

Some get/set methods

```
@Override
public String toString()
```

Problem 5: Data

Design a class named **Data** to store all data, including users, offices, groups, logs and so on. It is almost the same as a file or database. It can be initialized with some data for testing. When the system runs, it can also add, delete, update, search in the contents of data. Some statistic results can be got from the records.

Attributes:

private static ArrayList<User> userList;

private static ArrayList<Group> groupList;

private static ArrayList<Office> officeList;

private static ArrayList<Log> logList

Methods:

```
public Data()
```

Constructor to initialize 4 lists, only to create ArrayList with no element

```
public static void initialize()
```

// initialize with some data to userList, groupList, officeList and logList

// Codes for the method has been given.

// You can use it directly without any revision, and finish other methods.

// And you should not revise this method to change the initial data, or it may produce errors while testing.

// Of course you may add, delete and update the initial data in/by other methods.

```
public String addUser(User user)
```

return “user already exists”;

// if the userId of user to add already exists, not to add user; but if the user is invalid, set user.isValid=true. It is the same as addOffice, addGroup

return “addUser succeed”;// otherwise add user to userList

```
public String deleteUser(String userId)
```

return “user not exists”;// if the user to delete not exist

return “deleteUser succeed”;// otherwise set user.isValid=false, not delete user from userList

Notice: Deleted user will be set attribute isValid = false, not to delete it directly. An invalid user can be deleted again. It is the same as deleteOffice and deleteGroup.

```
public User getUser(String userId)
```

return User;// if userId exists, even if user.isValid==false.

return null;// if userId not exist.

```
public String addGroup(Group group)
```

return “group already exists”;// if the groupId of group to add already exists, not to add group; but if the group is invalid, set group.isValid=true.

return “addGroup succeed”;// otherwise add group to groupList

```
public String deleteGroup(int groupId)
```

return “group not exist”;// if the group to delete not exist

return “deleteGroup succeed”;

// otherwise set group.isValid=false, not delete group from groupList

```
public Group getGroup(int groupId)
```

return Group;// if groupId exists, even if group.isValid==false.

return null;// if groupId not exist.

```
public String addOffice(Office office)
```

return “office already exists”;

// if the officeId of office to add already exists, not to add office; but if the office is invalid, set office.isValid=true.

return “addOffice succeed”;// otherwise add office to officeList

```
public String deleteOffice(int officeId)
```

return “office not exist”;// if the office to delete not exist

return “deleteOffice succeed”;

// otherwise set office.isValid=false, not delete office from officeList

```
public Office  getOffice(int officeId)
```

return Office;// if officeId exists, even if officeId.isValid=false.

return null;// if officeId not exist.

```
public  boolean canAccessCampus (String userId, LocalDateTime dateTime)
```

// should have taken temperature before invoking this method

// if user with userId invalid or not exist, return false

// a valid user can access campus only if

// (1) temperature should < 37.3 today. It doesn’t matter if >= 37.3 in days before

// (2)user who lives in campus, directly return true

// user living in campus, user.district == “SUSTech”

// (3)user who lives outside campus, should be back to Shenzhen more than 14 days including teachers, visitors and so on. If a user back to Shenzhen on April 1, it is OK on April 15, but not on April 14

```
public boolean canAccessOffice(String userId,int officeId,LocalDateTime
dateTime)
```

// should have taken temperature before invoking this method

// if user with userId or office with officeId invalid or not exist, return false

// not need to consider that a user who lives outside campus to access an office, but has not accessed campus. Also not need to consider that a user who lives outside campus to access an office whether he/she has been back to Shenzhen over 14 days or not.

// a valid user can access the valid office only if

// (1) the user works in the office, including PI

// (2) temperature should < 37.3 today.

// (3)enough room for users to the office

Notice:A user haven’t enter the office can not enter if the office is full. If a user has been working in the office today, and got out later, It is allowed to enter the office directly even though the office is full. Since it not considered user gets out of office.

```
public void addAccessRecord(Log log)
```

// add accessing campus or office record to data.logList

```
public boolean accessCampus (String userId, LocalDateTime dateTime)
```

// when a user access campus

// first judge canAccessCampus: if true, then enter the campus and invoke addAccessRecord() to add accessing campus record to logList; if false, can not enter, and return false

// should have taken temperature before invoking this method

```
public boolean accessOffice(String userId,int officeId,LocalDateTime dateTime)
```
// when a user access an office
// first judge canAccessOffice, if true, then enter the office and invoke addAccessRecord() to add accessing office record to logList; if false, can not enter, and return false
// should have taken temperature before invoking this method

Some statistic method:

You just need to read the logList to get the statistic results, but not consider how they accessed the campus and offices.

```
public int accessCampusUserCount(LocalDateTime date)
```
// number of people who visits the campus on the date. A user lives in campus should not be counted if he/she didn't go out and get in, while it will be counted if he/she goes out and gets in again. A user gets in several times in a day should only count once.

```
public ArrayList<User> accessOfficeUsers(int officeId, LocalDateTime date){
```
// users who visit the office on the date. If a user gets into the office more than once, there should be only one in ArrayList returned.

```
public ArrayList<Integer> accessOfficeTimes(String userId, int officeId,
LocalDateTime startDate, LocalDateTime endDate)
```
// count the accessing times everyday of user who visit the office from startDate to endDate
// store the accessing times to returned ArrayList, if he/she didn't go to the office, it counts 0 on the date.
// Sample returned value: [1,2,0,1,1,0,1]

NOTICE: LocalDateTime, LocalDate need

```
import java.time.*;
```

Important: when deleting a user, office or group, just to set isValid=false, not need to delete it from group's or Data's userList, officeList, groupList. An invalid user cannot access and an invalid office cannot be accessed.

Revision History:

Apr26 v3.5

1. accessOffice: input variance temperature deleted.

```
1.public boolean accessOffice(String userId,int officeId,LocalDateTime
dateTime)
```