Assignment 4 2021 Spring

Contributors:

Designer: WANG Wei, YI Xiang, LU Lan, ZHANG Jiayu

Tester： YI Xiang, ZHANG Jiayu, NIE Qiushi, FU Weibao

1. [Easy] MagicStrings (30 points)

Description

Design a class named `MagicStrings`, which could compare the strings according to the `priority` identified, sort items of `ss` in descending order and provide a way to get the string based on `ss` .

1.1. Attributes

`private int[] priority` :There are 26 items in `priority`, representing the priority order of the 26 **lowercase** letters, with index `0` for `'a'`, index `1` for `'b'` , index `25` for `'z'` and so forth. The priority values are guaranteed to be integers while in range from 1 to 26 and different for different lowercase letters. A larger value indicates higher priority.

NOTE that the strings may contain letters in both uppercase and lowercase and the priority of uppercase letters are **the same as** its lowercase letters.

`private String[] ss` : The array containing the references of Strings to be sorted before `sort` and the references of Strings which have been `sort` . Every item in `ss` is supposed to include only letters in both lowercase and uppercase.

1.2. Constructor

1.2.1. Constructor 1:

```
public MagicStrings(String s)
```

Initialize `priority` with default priorities. Initialize `ss` with method `setSs(String s)`.

The default priorities are list in an array with 26 integer items and following initial values: {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26}

1 is the value of 1st item, 26 is the value of last item, and so forth.

1.2.2. Constructor 2:

```
public MagicStrings(int[] priority, String s)
```

Initialize `priority` with method `setPriority(int[] priority)`. Initialize `ss` with method `setSs(String s)`.

1.2.3. Constructor 3:

```
public MagicStrings(String priority, String s)
```

Initialize `priority` with method `setPriority(String priority)`. Initialize `ss` with method `setSs(String s)`.

1.3 Method setPriority

1.3.1. setPriority 1:

```
public void setPriority(int[] priority)
```

Set `priority` with a given array `int[] priority` which is supposed to be an array with 26 integer items, each item is different from each other and is on a range of 1 to 26.

1.3.2. setPriority 2:

```
public void setPriority(String priority)
```

Set `priority` with a given String in the format of "1 2 3 4 5 6 7 8 ... 23 24 25 26" where the priority values are indicated by numbers split by one blank space.

1.3.3. setPriority 3:

```
public void setPriority(char c, int priority)
```

Change the priority of `c` in data filed `priority` with parameter `priority`. `c` is supposed to be a letter which is among from `'a'` to `'z'` or `'A'` to `'Z'`.

For example, **setPriority('A',26)** means change the value of **priority[0]** to be **26**.
setPriority('z',1) means change the value of **priority[25]** to be **1**.

1.4. Method setSs

```
public void setSs(String input)
```

Set `ss` with a given string in the format of `"SeD a ... Ac abCd"`, this string is supposed to be a group of strings which are split by blank space(s), every string item be set as an array item in `ss`.

Note that the input may contain invalid letter like `numbers`, `"\"`, `"+"`, `"("` and so on , but only letters and blank space would be left. Blank space is supposed as separator between strings.

For example, the `input` is **"A)a bf ce\12#d xy+z"** , after `setSS(input)` , `ss` would be {"**Aa**" , "**bf**" , "**ced**", "**xyz**"}.

1.5. Method stringsNum

```
public int stringsNum()
```

Return the number of strings in `ss`. For example, if the `ss` is "The Input For SS iS inValid", `stringsNum` would return 6. if `ss` is null, return 0.

1.7. Method compareString

```
public int compareString(String a, String b)
```

compare the priority of characters in string `a` and `b`. Both `a` and `b` are supposed to only includes letters both in uppercase or lowercase.

1) Return `0` if the priority of `a` is equal to that of `b` and the length of `a` is equal to that of `b` ;

2) Return `-1` if the priority of `a` is smaller than that of `b`, or the priority of `a` is equal to that of `b` and the length of `a` is smaller than that of `b` ;

3) Return `1` if the priority of `a` is bigger than that of `b`, or the priority of `a` is equal to that of `b` and the length of `a` is bigger than that of `b` ;

For example, while the `priority` is {1,2,3,4,..26} ,

`compareString("abcd","aCaaa")` return **-1**. It is because that the 1st letter of **"abcd"** and **"aCaaa"** has the same priority, the 2nd letter in **"abcd"** is **'b'** whose priority is 2 while the 2nd letter in **"aCaaa"** is **'C'** whose priority is 3, so "abcd" is smaller than "aCaaa".

`compareString("abcd","aBCd")` return 0. Here the priority of first 4 letters in two strings are equal, and the length of "**abcd**" is same that of string "**aBCd**", so return 0.

`compareString("abCDz","abcd")` return 1. Here the priority of first 4 letters in two strings are equal, and the length of "**abCDz**" is bigger than that of string "**abcd**", so return 1.

1.8. Method sortSs

```
public void sortSs()
```

Sort array `ss` into an descending order by invoke `compareString` method .

For example, the `priority` is {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26}, the `ss` is **{"at","Ata","aR","AS"}** , after `sortSs`, `ss` is **{"Ata","at","AS","aR"}**.

1.9. Method getStrings

```
public String getStrings()
```

Build a new string which is composed with every item in `ss` with a blank space as theirs separator.

For example, `ss` is **{"Ata","at","AS","aR"}** , `getStrings` will return a string as **"Ata at AS aR**".

Submission

You are free to design other methods to help you complete the assignment. For this question, submit one java file: `MagicStrings.java`.

Description

The midterm exam is coming so some students want to order discussion rooms to review lessons with their friends.

Now you need to design two classes `Room` and `RoomManager`. `Room` represents discussion room and `RoomManager` is used to manager discussion rooms. With the help of `RoomManager`, you can add and order rooms. Through searching the rooms, you can get rooms on your need and order them. Also you can cancel orders.

We provide you with two enum classes `Library` and `Landmark`. `Library` is used to represent rooms' location. `Landmark` contains some SUSTech landmarks such as teaching building and research building, which is used to search the nearest rooms from the specified location. Now let's start our journey!

NOTICE: This description is for **Question 2** and **Question 3**. Make sure you complete all tasks in **Question 2** before you start **Question 3**.

2. [Medium]Room (30 points)

Design a class named `Room`. `Room` represents discussion room. We provide you with an enum class `Library` to specify the location of the room (it **CANNOT** be modified):

```
public enum Library {
    Lynn, Yidan, LearningNexus
    // 对应：琳恩图书馆，一丹图书馆，涵泳图书馆
}
```

2.1. Attributes:

`private String rid`: Represents room id such as **'R101'**.

`private Library location`: Represents which library this room is located in. **Default**: `Library.Lynn`.

`private int capacity`: A **POSITIVE** integer representing the capacity of the room. **Default**: `3`.

`private boolean hasDisplay`: Represents whether this room has a display. **Default**: `true`.

`private boolean hasWhiteboard`: Represents whether this room has a whiteboard. **Default**: `true`.

`private String[] applicants`: Contains SIDs of applicants of each valid single hour. You can use the indices to represent the start time of each order.

NOTICE:

1. `rid` should strictly follow the format: a character `R` + a 3-digit-number **no less than 100**, e.g. `R101`, `R204`, `R999`.
2. `applicants`: rooms are **only available from 08:00 to 22:00** so it is an array of **length 15** and `applicants[0]` represents the `SID` of the applicant of the order from 08:00 to 09:00 (`null` if room is not ordered in this interval). **Default**: `null`.

   e.g. If the value of applicants[0] is 11911000, it means student whose SID is 11911000 has order this room for use from 08:00-09:00. If it's value is `null` means the room hasn't been ordered, it is valid for applicants.

2.2. Constructors:

2.2.1. Constructor 1:

```
public Room (String rid)
```

A default room constructor with specified `rid`. Set all other attributes to their default value.

2.2.2. Constructor 2:

```
public Room (String rid, Library location, int capacity)
```

An overloaded room constructor with `rid`, `location` and `capacity` specified. Set all other attributes to their default value.

2.2.3. Constructor 3:

```
public Room (String rid, Library library, int capacity, boolean hasDisplay,
boolean hasWhiteboard)
```

An overloaded room constructor with `rid`, `location`, `capacity`, `hasDisplay` and `hasWhiteboard` specified. Set all other attributes to their default value.

2.3. Method toString:

2.3.1. toString 1:

```
public String toString ()
```

Returns the info and all orders in a whole day (from 08:00 to 22:00) of this room object as a string.

The returned string includes 3 lines which should strictly follows the rules stated below:

1. The first line contains basic attributes of the room;
2. The second line contains the timetable of one day available for order. Between every two '|'s there are space for **8** characters. Each time interval is represented as `xx:00` where `xx` is the beginning time of this time interval;
3. The third line also follows the same format as the second line. In each interval, put a string `"EMPTY"` if there is no order. Otherwise, put the `SID` of the applicant of this reservation. When one applicant made an order of 2 hours, omit the second '|' as the following example.

For example, the following string is returned when calling `toString()`:

```
Lynn R101, capacity=6, hasDisplay=true, hasWhiteboard=true\n
|08:00   |09:00   |10:00   |11:00   |12:00   |13:00   |14:00   |15:00   |16:00
|17:00   |18:00   |19:00   |20:00   |21:00   |\n
|11810000          |EMPTY   |11910000|EMPTY   |EMPTY   |EMPTY   |EMPTY   |EMPTY
|EMPTY   |EMPTY   |EMPTY   |EMPTY   |EMPTY   |\n
```

NOTICE:

- "\n"s in the example are merely used to emphasize new lines in that some lines are too long for this document to show.

2.3.2. toString 2:

```
public String toString (int start, int end)
```

Only return room info and order info from start time to end time (8 $\le$ `start` $\le$ 21, 9 $\le$ `end` $\le$ 22). For example, `toString(9, 12)` returns order info from 09:00 to 12:00. If `end` is **no greater than** `start` or either of them is not in their range, return `null`.

For example, the following string is returned when calling `toString(8,16)`:

```
Lynn R101, capacity=6, hasDisplay=true, hasWhiteboard=true
|08:00   |09:00   |10:00   |11:00   |12:00   |13:00   |14:00   |15:00   |
|11810000         |EMPTY   |11910000|EMPTY   |EMPTY   |EMPTY   |EMPTY   |
```

2.4. Method setApplicant

```
public boolean setApplicant(int start, int end, String SID, int
numberOfTeammates)
```

This method is used when an order is tried. `start` and `end` follows the rules stated in **2.3.2**. For example, `setApplicant(8, 9, 11910000, 3)` means applicant with `SID` = 11910000 orders the room from 08:00 to 09:00 with 3 teammates.

NOTICE:

1. All reservation cases only consider the time in one day, that is, you do not need to consider the real time;
2. Reservations are limited to **one hour or two hours** at a time. Rooms are only available from **08:00 to 22:00**;
3. If the sum of `numberOfTeamates` and 1 is greater than the room's capacity or is negative, this order fails. **Notice that teammates do not contain the applicant**.
4. If there has been an order for this room object in the selected time interval, that is, the corresponding applicants on this time interval are not null, the order fails;
5. If the applicant has ordered the room before this order, which means the `SID` has been in the `applicants`, the order fails;
6. If all rules are followed, the order is successful and the method returns true. If order fails, do nothing but return `false`.

   TIPS: the room's `applicants` Must be updated at the same time.

2.5. Method removeApplicant

```
public boolean removeApplicant (String SID)
```

This method will be called to cancel orders of applicant with `SID`: search the SID from the array 'applicants', if SID is found, then reset the related array item(s) as `null` and returns `true`, do nothing but return `false` otherwise.

For example, before removing, `applicants` is:

```
[11810000, null, null, null, null, 11910000, 11910000, null, null, null, null,
null, null, null, null]
```

After calling `removeApplicant("11910000")`, `applicants` is:

```
[11810000, null, null, null, null, null, null, null, null, null, null, null,
null, null, null]
```

2.6. Other methods

You are free to design other methods to help you complete the assignment, Including but not limited to getters and setters.

Submission

For this question, submit one java file: `Library.java` , `Room.java`.

3. [Medium]RoomManager (40 points)

Design a class named `RoomManager`. `RoomManager` manages all rooms for three libraries.

3.1. Attributes:

```
private static ArrayList<Room> roomInfoLynn;
private static ArrayList<Room> roomInfoYidan;
private static ArrayList<Room> roomInfoLearningNexus;
```

Three static attributes used to represent all room info of each library.

You can also use your own design rather than the design above to realize the following methods.

3.2. Static Method addRoom:

3.2.1. addRoom 1:

```
public static boolean addRoom (String rid, Library location, int capacity)
```

Create a room with `rid`, `location` and `capacity` specified and add it to the room list of given library. Other attributes of the room should be assigned by default values as stated in **2.1**.

3.2.2. addRoom 2:

```
public static boolean addRoom (String rid, Library library, int capacity,
boolean hasDisplay, boolean hasWhiteboard)
```

Create a room with `rid`, `location`, `capacity`, `hasDisplay` and `hasWhiteboard` specified and add it to the room list of given library which is specified by `library`. Other attributes of the room should be assigned by default values as stated in **2.1**.

NOTICE:

For both **3.2.1** and **3.2.2**, following rules **MUST** be followed:

- `rid` should follow the rules stated in **2.1**. If rules are not followed, return `false`.
- If there is an existing room in room list of the library having duplicated `id`, this method will only return `false`. Otherwise, create a room, add it to the room list and return `true`.

3.3. Static Method orderRoom:

```
public boolean orderRoom (Library library, String rid, String SID, int start,
int end, int numberOfTeammates)
```

Someone `SID` orders a room `rid` in a `library` from `start` time to `end` time with `numberOfTeammates` teammates together.

NOTICE:

1. Each user can only order one room in each library (but could order up to 3 rooms in 3 different libraries. These orders **may even happen at the same time**, such as an online meeting using rooms in different libraries). But one user **CANNOT** order two rooms in one library as stated in **2.4**.
2. Return `false` if no room has room id equaling to `rid`.
3. Other rules are stated in **2.4**;
4. We ensure that `library` and `SID` are valid;
5. If all rules are followed, the order is successful and the method returns true. Make sure the room with `rid` update its attributes according to this order.

3.4. Static Method cancelOrder:

3.4.1. cancelOrder 1:

```
public static boolean cancelOrder (String SID)
```

If no order is to be canceled, that is, the applicant with `SID` have not order a room in any library before(there is no `SID` in the `applicants` of all the rooms), return `false`. Otherwise cancel all orders in three libraries for applicant with student id equaling `SID`.TIPS： the related attributes of rooms MUST be updated.

3.4.2. cancelOrder 2:

```
public static boolean cancelOrder (String SID, Library location)
```

If no order is to be canceled, return `false`. Otherwise cancel all orders in the library specified by `location` for applicant with student id equaling `SID`. TIPS： the related attributes of rooms MUST be updated.

3.5. Static Method searchRoom:

We provide you with an **INCOMPLETE** enum class named `Landmark` where each landmark has its own priority of three libraries (what it actually means is the order of distances between the landmark and libraries).

e.g, For landmark "**TeachingBuilding**", it's **Library[0]** is `Library.Yidan`, **Library[1]** is `Library.Lynn`, **Library[2]** is `Library.LearningNexus` . which means:

`Library.Yidan` is the nearest for `TeachingBuilding` then `Library.Lynn` and `Library.LearningNexus` follow.

You are asked to complete this enum class with a constructor. You are free to implement other methods to help you complete the assignment.

```
public enum Landmark {
    TeachingBuilding(new Library[]{Library.Yidan, Library.Lynn,
Library.LearningNexus}),
    ResearchBuilding(new Library[]{Library.Lynn, Library.Yidan,
Library.LearningNexus}),
    TaizhouFloor(new Library[]{Library.Lynn, Library.Yidan,
Library.LearningNexus}),
    AdministrativeBuilding(new Library[]{Library.Lynn, Library.Yidan,
Library.LearningNexus}),
    SUSTechCenter(new Library[]{Library.Yidan, Library.Lynn,
Library.LearningNexus}),
    Dormitory(new Library[]{Library.LearningNexus, Library.Yidan,
Library.Lynn}),
    Playground(new Library[]{Library.Yidan, Library.LearningNexus,
Library.Lynn});

    // The library item specified by smaller index in priority means smaller
distance between the library item and the landmark.
    private final Library[] priority;

    //TODO: add your constructor and other methods here
}
```

3.5.1. searchRoom 1:

```
public static ArrayList<Room> searchRoom (Library location, int start, int end,
boolean needDisplay, boolean needWhiteboard)
```

Search a list of rooms with the need of display(identified by `hasDisplay`) and whiteboard (identified by `hasWhiteboard`) which are available for order from `start` time to `end` time in library `location`. You can still get rooms with display even though you set `needDisplay` as `false`. However, you cannot get rooms without display if you set `needDisplay` as `true`. For `needWhiteboard`, it is the same.

3.5.2. searchRoom 2:

```
public static ArrayList<Room> searchRoom (int start, int end)
```

Search a list of rooms which are available for order from `start` time to `end` time in all three libraries.

3.5.3. searchRoom 3:

```
public static ArrayList<Room> searchRoom (int start, int end, Landmark landmark)
```

Search a list of rooms which are available for order from `start` time to `end` time in all three libraries. Put rooms near `landmark` on the front.

Notice:

The following rulers should be followed in **3.5.1, 3.5.2 and 3.5.3**.

- The returned rooms should be in order. First sort rooms in order of libraries (For landmark specified, follow its priority: priority[0], priority[1], and so forth. If the landmark is not specified, use default order: `Yidan`, `Lynn`, `LearningNexus`).

  For example, for `Landmark.Dormitory`, you should list rooms with the order: `Library.LearningNexus`, `Library.Yidan`, `Library.Lynn` according to its priority.

- For rooms in the same library, sort them in **lexicographical order**. For example, `R101`, `R102`, `R201`.

3.6. Static Method showOrder:

```
public static String showOrder (ArrayList<Room> list)
```

Used to get the string of the search result from **3.5**, including the info and order info of all the rooms in this `list`. If `list` is empty which means it's size is 0, return `"No room to show."`.

The returned String would be like:

```
Lynn R101, capacity=6, hasDisplay=true, hasWhiteboard=true\n
|08:00   |09:00   |10:00   |11:00   |12:00   |13:00   |14:00   |15:00   |16:00
|17:00   |18:00   |19:00   |20:00   |21:00   |\n
|11810000         |EMPTY   |11910000|EMPTY   |EMPTY   |EMPTY   |EMPTY   |EMPTY
|EMPTY   |EMPTY   |EMPTY   |EMPTY   |EMPTY   |\n
Lynn R102, capacity=4, hasDisplay=true, hasWhiteboard=true\n
|08:00   |09:00   |10:00   |11:00   |12:00   |13:00   |14:00   |15:00   |16:00
|17:00   |18:00   |19:00   |20:00   |21:00   |\n
|12010000|EMPTY   |11910000|EMPTY   |EMPTY   |EMPTY   |EMPTY   |EMPTY   |EMPTY
|EMPTY   |EMPTY   |EMPTY   |EMPTY   |EMPTY   |\n
LearningNexus R101, capacity=5, hasDisplay=true, hasWhiteboard=true\n
|08:00   |09:00   |10:00   |11:00   |12:00   |13:00   |14:00   |15:00   |16:00
|17:00   |18:00   |19:00   |20:00   |21:00   |\n
|EMPTY   |EMPTY   |EMPTY   |EMPTY   |EMPTY   |EMPTY   |11840000|EMPTY   |EMPTY
|EMPTY   |EMPTY   |EMPTY   |EMPTY   |EMPTY   |\n
```

NOTICE:

- "\n"s in the example are merely used to emphasize new lines in that some lines are too long for this document to show.

3.7. Other methods

You are free to implement other methods to help you complete the assignment.

Submission

For this question, submit three java files: `Library.java`, `Room.java`, `Landmark.java` and `RoomManager.java`.