Website

After being shown how to convert images to grayscale and compare the minimum differences between them in Lab #01, Rar the Cat now wants to make a website that allows a group of users to upload their photos.

Rar the Cat has designed the website to support 5 operations:

1. Upload photos operation (UPLOAD)
2. Tag users operation (TAG)
3. Tagged photos query operation (TAGGED_PHOTOS)
4. Self-tag statistics query operation (SELF_TAG)
5. Server statistics query operation (SERVER_STATS)

The website will be backed by an Application Programming Interface (API) which the front-end website will interact with in order to perform these functionalities correctly. In addition, Rar the Cat has already drafted the operations that the API should support, as below.

Rar the Cat is too busy that he does not have time to write the backend for his designed API. However, he has heard from Mr Panda that CS1020 students are very good with Objected Oriented Programming (OOP) and data structures. Hence, he decided to enlist you, a fellow CS1020 student, to code his API.

To simulate queries to the API, Rar the Cat has provided an input file to test your API with. There will be a total of **N** operations to the API, with each operation taking up a single line of input. Details of the input structure will be detailed together with the API specifications in subsequent pages.

Input

The first line of input will contain a single integer **N**, the number of operations to the API. It is guaranteed that $0 < N \leq 200$.

N lines of input will follow. Each line of input will correspond to one operation to the API.
The details and input format of these operations will be detailed in subsequent pages.

Among these, it is guaranteed that all **usernames** will only contain lowercase alphabets ('a' to 'z') up to 20 characters long. In addition, all **photoID** will be an integer between 0 and 1,000,000,000.

Output

For each operation, print the corresponding output denoted in subsequent pages.

1. Upload photo operation

Description

This operation will be provided with 2 parameters: a **username** and a distinct integer **photoID**, indicating that a user with **username** has uploaded a new photo, denoted by the integer **photoID**. It is *guaranteed* that no users will upload 2 photo with the same **photoID**.

Response and Error Handling

- If there is no existing user with **username**, the API is supposed to create such a new user and output “New user [**username**] created.” on a single line, before *continuing* with the operation.
- Subsequently, output “Photo **[photoID]** uploaded successfully by **[username]**.”

Input/Output Format

In your input, this operation will be in the following format: `UPLOAD [username] [photoID]`
Do note that output corresponding to alternate lines of input are **bolded** in the table below for clarity.

Sample Input (**website1.in**)	Sample Output (**website1.out**)
`4`	**`New user rarthecat created.`**
`UPLOAD rarthecat 23`	**`Photo 23 uploaded successfully by rarthecat.`**
`UPLOAD panda 39`	`New user panda created.`
`UPLOAD panda 57`	`Photo 39 uploaded successfully by panda.`
`UPLOAD hellokitty 0`	**`Photo 57 uploaded successfully by panda.`**
	`New user hellokitty created.`
	`Photo 0 uploaded successfully by hellokitty.`

2. Tag user operation

Description

This operation will be provided with 2 parameters: a **photoID**, followed by a **username**, indicating that a user with **username** has been tagged in the photo denoted by the integer **photoID**.

Response and Error Handling

- If there is no existing user with **username**, the API is supposed to create such a new user and output “New user [**username**] created.” on a single line, before *continuing* with the operation.
- If there is no photo previously uploaded with **photoID**, the API is supposed to *terminate* the operation and output “No photo [**photoID**] exists.”
- If **username** is already tagged in the photo denoted by **photoID**, the API is supposed to *ignore* the operation and output “User **[username]** is already tagged in photo **[photoID]**.”
- Lastly, if the operation has not been *terminated* or *ignored*, output “User **[username]** tagged successfully in photo **[photoID]**.”

To clarify, if the operation is called with a new **username** and a **photoID** that was not uploaded before, the API is *still* supposed to create a new user but then *reject* the tagging thereafter.

Input/Output Format

In your input, this operation will be in the following format: `TAG [photoID] [username]`
Do note that output corresponding to alternate lines of input are **bolded** in the table below for clarity.

Sample Input (**website2.in**)	Sample Output (**website2.out**)
`6`	**`New user panda created.`**
`UPLOAD panda 23`	**`Photo 23 uploaded successfully by panda.`**
`TAG 23 rarthecat`	`New user rarthecat created.`
`TAG 23 panda`	`User rarthecat tagged successfully in photo 23.`
`TAG 23 rarthecat`	**`User panda tagged successfully in photo 23.`**
`TAG 999 rarthecat`	`User rarthecat is already tagged in photo 23.`
`TAG 999 miao`	**`No photo 999 exists.`**
	`New user miao created.`
	`No photo 999 exists.`

3. Tagged photos query operation

Description

This operation will be provided with 1 parameter only: a single username. Then, this operation should return the number of distinct photos the user with username is tagged in, number_of_tagged_photos.

Response and Error Handling

- If there is no existing user with username, the API is supposed to *terminate* the operation and output “No user [username] exists.”
- Otherwise the API should compute the number of tagged photos the user is tagged in and output “User [username] is tagged in [number_of_tagged_photos] photo(s).”

To clarify, users that upload photos might not necessarily be automatically tagged in their own photos. In addition, if the user is not tagged in any photos, this operation should return 0.

Input/Output Format

In your input, this operation will be in the following format: `TAGGED_PHOTOS [username]`
Do note that output corresponding to alternate lines of input are bolded in the table below for clarity.

Sample Input (website3.in)	Sample Output (website3.out)

`8`
`UPLOAD hellokitty 2`
`TAG 2 rarthecat`
`TAGGED_PHOTOS rarthecat`
`UPLOAD panda 3`
`TAG 3 rarthecat`
`TAGGED_PHOTOS rarthecat`
`TAGGED_PHOTOS hellokitty`
`TAGGED_PHOTOS invalidusername`
	`New user hellokitty created.`
	`Photo 2 uploaded successfully by hellokitty.`
	`New user rarthecat created.`
	`User rarthecat tagged successfully in photo 2.`
	`User rarthecat is tagged in 1 photo(s).`
	`New user panda created.`
	`Photo 3 uploaded successfully by panda.`
	`User rarthecat tagged successfully in photo 3.`
	`User rarthecat is tagged in 2 photo(s).`
	`User hellokitty is tagged in 0 photo(s).`
	`No user invalidusername exists.`

4. Self-tag statistics query operation

Description

This operation will be provided with no parameters and it should calculate the total number of users that have uploaded at least one photo which have themselves tagged in it.
For example, if user rarthecat uploaded photo 17 and is also tagged in photo 17, then we will count rarthecat as one such user whom was tagged in their own photo. Do note that we will only count each user once, even if the user has uploaded multiple photos that have him/herself tagged in it.
In another example, if another user panda has uploaded photos 1, 2 and 3 but none of these 3 photos were tagged with panda, then we will not count panda as one such user.

Response and Error Handling

- Compute the number of users which as tagged themselves in at least one photo which they uploaded as [number_of_self_tag_users].
- Output the result in a single line as such: “There are [number_of_self_tag_users] user(s) that have tagged themselves.”

Input/Output Format

In your input, this operation will be a single word on a single line: `SELF_TAG`
Do note that output corresponding to alternate lines of input are **bolded** in the table below for clarity.

Sample Input (**website4.in**)	Sample Output (**website4.out**)

`11`
`UPLOAD rar 17`
`UPLOAD rar 27`
`UPLOAD panda 3`
`SELF_TAG`
`TAG 17 panda`
`SELF_TAG`
`TAG 17 rar`
`TAG 27 rar`
`SELF_TAG`
`TAG 3 panda`
`SELF_TAG`
	`New user rar created.`
	`Photo 17 uploaded successfully by rar.`
	`Photo 27 uploaded successfully by rar.`
	`New user panda created.`
	`Photo 3 uploaded successfully by panda.`
	`There are 0 user(s) that have tagged themselves.`
	`User panda tagged successfully in photo 17.`
	`There are 0 user(s) that have tagged themselves.`
	`User rar tagged successfully in photo 17.`
	`User rar tagged successfully in photo 27.`
	`There are 1 user(s) that have tagged themselves.`
	`User panda tagged successfully in photo 3.`
	`There are 2 user(s) that have tagged themselves.`

5. Server statistics query operation

Description

This operation will be provided with no parameters and the API should output how many users there are in the website as well as how many photos have been uploaded.

Response and Error Handling

- Compute how many users there are in the website as **[number_of_users]** and how many photos have been uploaded in total as **[number_of_photos]**.
- Output the result in a single line as such: "Total: **[number_of_users]** user(s), **[number_of_photos]** photo(s)."

Input/Output Format

In your input, this operation will be a single word on a single line: `SERVER_STATS`
Do note that output corresponding to alternate lines of input are **bolded** in the table below for clarity.

Sample Input (**website5.in**)	Sample Output (**website5.out**)

`10`
`SERVER_STATS`
`UPLOAD rar 1`
`UPLOAD rar 2`
`SERVER_STATS`
`UPLOAD panda 3`
`SERVER_STATS`
`TAG 1 panda`
`SERVER_STATS`
`TAG 1 ivan`
`SERVER_STATS`
	`Total: 0 user(s), 0 photo(s).`
	`New user rar created.`
	`Photo 1 uploaded successfully by rar.`
	`Photo 2 uploaded successfully by rar.`
	`Total: 1 user(s), 2 photo(s).`
	`New user panda created.`
	`Photo 3 uploaded successfully by panda.`
	`Total: 2 user(s), 3 photo(s).`
	`User panda tagged successfully in photo 1.`
	`Total: 2 user(s), 3 photo(s).`
	`New user ivan created.`
	`User ivan tagged successfully in photo 1.`
	`Total: 3 user(s), 3 photo(s).`

Sample Testcase

We have also provided a longer sample testcase that includes all operations and *most* of their error handling cases, with explanation. Do note that output corresponding to alternate lines of input are **bolded** in the table below for clarity. (You do not need to bold your output.)

Sample Input (website6.in)	Sample Output (website6.out)
17	**New user rar created.**
UPLOAD rar 39	**Photo 39 uploaded successfully by rar.**
UPLOAD rar 13	Photo 13 uploaded successfully by rar.
UPLOAD panda 12	**New user panda created.**
TAG 12 panda	**Photo 12 uploaded successfully by panda.**
TAG 13 panda	User panda tagged successfully in photo 12.
TAGGED_PHOTOS panda	**User panda tagged successfully in photo 13.**
TAGGED_PHOTOS rar	User panda is tagged in 2 photo(s).
TAGGED_PHOTOS ivan	**User rar is tagged in 0 photo(s).**
SELF_TAG	No user ivan exists.
SERVER_STATS	**There are 1 user(s) that have tagged themselves.**
TAG 11 panda	Total: 2 user(s), 3 photo(s).
TAG 12 panda	**No photo 11 exists.**
TAG 12 ivan	User panda is already tagged in photo 12.
TAG 13 rar	**New user ivan created.**
TAGGED_PHOTOS ivan	**User ivan tagged successfully in photo 12.**
SELF_TAG	User rar tagged successfully in photo 13.
SERVER_STATS	**User ivan is tagged in 1 photo(s).**
	There are 2 user(s) that have tagged themselves.
	Total: 3 user(s), 3 photo(s).

Explanation

There are a total of 17 API operations in this input.

1. User rar uploaded photo 39. Since user rar did not exist before, it was created and the photo was uploaded successfully.
2. User rar uploaded photo 13. It was uploaded successfully. Note that no new user accounts were created as user rar already has an account.
3. User panda uploaded photo 12. panda is a new user which did not have an account. Hence, a new user panda was created and the upload of photo 12 proceeded successfully after that.
4. User panda is tagged in photo 12.
5. User panda is tagged in photo 13.
6. User panda is tagged in 2 photos in total; photos 12 and 13.
7. User rar is not tagged in any photos yet, although he has uploaded 2 photos.
8. User ivan does not exist (*yet*), API returns an error indicating so.
9. 1 user have themselves tagged in the photos they upload; user panda.
10. There are 2 users in the website so far – rar and panda and 3 photos – 12, 13, 39.
11. Operation was to tag panda in photo 11. Photo 11 does not exist, hence an error is returned.
12. Operation was to tag panda in photo 12. User panda is already tagged in photo 12, hence an error is returned.
13. User ivan is tagged in photo 12. An account was created for ivan as he is a new user which did not have an account previously.
14. User rar is tagged in photo 13, which is one of the photos he uploaded.
15. User ivan is tagged in 1 photo in total; photo 12.
16. 2 users have themselves tagged in the photos they upload; user panda in photo 12 and rar in photo 13.
17. In total, there are 3 users – rar, panda and ivan; 3 photos – photos 12, 13, 39.

Skeleton

You are given the skeleton file ***Website.java***. You should see a file with the following content when you open it, otherwise you might be in the wrong directory.

```java
/**
 * Name         :
 * Matric No.   :
 * PLab Acct.   :
 */

import java.util.*;

public class Website {

    // define your own attributes, constructor, and methods here

    private void run() {

    }

    public static void main(String[] args) {
        Website website = new Website();
        website.run();
    }
}

class User {
    // define your own attributes, constructor, and methods here
}

class Photo {
    // define your own attributes, constructor, and methods here
}
```

Notes:

1. You should develop your program in the subdirectory `ex1` and use the skeleton java file provided. You should not create a new file or rename the file provided.
2. You only need to modify the skeleton file. <u>You do not need to create a new file for each class</u>. All code should be <u>inside the file given to you</u> in the `ex1` directory.
3. If your algorithm is different from the given skeleton, you are free to write a solution according to your own algorithm. You are free to define your own classes besides the ones given in the skeleton file.
4. You <u>must (and need to)</u> use OOP for this sit-in lab.
5. You are free to define your own methods.
6. Please be reminded that the marking scheme is:

Input	: 10%
Output	: 10%
Correctness	: 50%
Programming Style	: 30%, which consists of:

   - Meaningful comments (pre- and post- conditions, comments inside the code): 10%
   - Modularity (incremental programming, proper modifiers [public / private]): 10%
   - Proper Indentation: 5%
   - Meaningful Identifiers (for both method and variable names): 5%

Compilation Error : Deduction of **50% of the total marks obtained**.