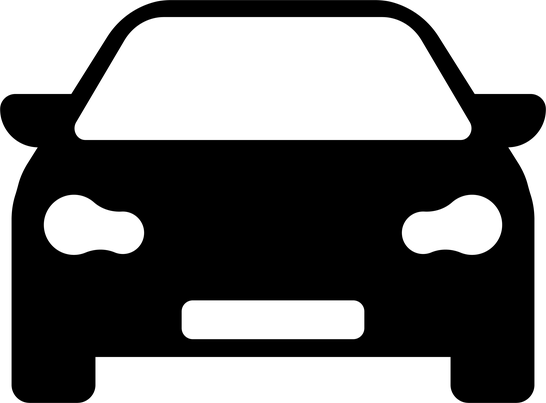
# KAT



*Your task is to optimize the performance of the KAT. By helping them reduce the queue of cars waiting for registration.*

You will be given **two sequences of integers** – one with **license plates** and one with **cars**. Two license plates are required for each car. Your goal is to find the **number of registered cars** and **how many days** they are installed. If there are cars or license plates that have not been used, you should **count** them.

The installation starts from the **first** license plate with the **last** cars. If, after the end of the day, **license plates remain**, they are added at the **end of the sequence** for a new day. If after the end of the first day there are **cars without license plates**, they are added at the **beginning of the sequence** for a new day. The procedure is performed until at least one **sequence** is over.

### Input

* On the **first line**, you will receive the integers representing the number of **license plates, separated** by **", "**.
* On the **second line**, you will receive the integers representing the number of **cars**, **separated** by "**,** ".

### Output

* Print count of registered cars and how many days were needed:
  + **“{count of registered cars} cars were registered for {count of days} days!”**
* If there are any remaining license plates:
  + **“{count of plates} license plates remain!”**
* If any cars are remaining without license plates:
  + **“{count of cars without license plates} cars remain without license plates!”**
* If all cars and license plates were used:
  + **"Good job! There is no queue in front of the KAT!"**

### Constraints

* License plates will be **even** numbers.
* All of the given numbers for license plates will be **valid** integers in the range **[4, 2147483646]**.
* All of the given numbers for cars will be **valid** integers in the range **[2, 2147483647]**.

### Examples

|  |  |
| --- | --- |
| ****Input**** | ****Output**** |
| **30, 50**  **25, 15** | **40 cars were registered for 2 days!**  **Good job! There is no queue in front of the KAT!** |
| ****Comment**** | |
| (Day 1) -> We start with the last cars (15) and the first license plates (30) -> 2 license plates for 1 car -> 2 \* 15 cars = 30 license plates.  Count of registered cars: 15  Sequences:  [50]  [25]  (Day 2) -> 25 cars and 50 license plates -> 2 license plates for 1 car -> 2 \* 25 cars = 50 license plates. **25** cars were registered.  Count of registered cars: 40. | |

|  |  |
| --- | --- |
| ****Input**** | ****Output**** |
| **20, 100, 40, 68**  **33, 50** | **83 cars were registered for 4 days!**  **62 license plates remain!** |
| ****Comment**** | |
| (Day 1) -> We start with the last cars (50) and the first license plates (20) -> 20 license plates for **10** cars -> 10 cars were registered -> The **rest (40)** cars were moved at **beginning** of the sequence.  Count of registered cars: **10**  Sequances:  [100, 40, 68]  [40, 33]  (Day 2) -> 33 cars and 100 license plates -> 66 license plates for 33 cars -> Add **33** cars to count of registered cars -> The **rest (34)** license plates are moved to the **end** of sequence.  Count of registered cars: **43**  Sequences:  [40, 68, **34**]  [40]  (Day 3) -> 40 cars and 40 plates -> 40 plates for 20 cars -> Add 20 cars to count of registered cars -> The **rest (20)** cars were moved in **beginning** of sequance.  Count of registered cars: **63**  Sequences:  [68, 34]  [**20**]  (Day 4) -> 20 cars and 68 plates -> 40 plates for 20 cars -> Add **20** cars to count of registered cars -> The **rest (28)** plates are moved to the **end** of sequence.  Count of registered cars: **83**  The sequence of license plates: [34, 28] | |

# 02. Treasure hunt

*Shape

Description automatically generated with medium confidence*

*You are a treasure hunter. You have found a long-lost secret map, showing the way to a hidden treasure. But is the map real or fake…*

You will be given **N** and **M** – **integers**, indicating the **field's dimensions**, shown on the secret map. On the next **N** lines, you will receive the **rows** of the field. You will be placed in a **random position**, marked with the letter '**Y**'. On random positions, there will be trees marked with the letter '**T**'. The treasure is marked with the letter '**X**'. **All of the empty positions** will be marked with **'-'**.

You will receive a few lines with commands representing which direction you need to move. The possible directions are **up, down, right,** and **left.**

If you go out of the field, you need to stay in the last possible position inside the field.

If you step on a tree (position marked with '**T**'), **go one step back** to the direction you came from (not make a move).

If the given command is **"Finish"** you need to check the position you are standing on. If it is marked with '**X**' this means you have found the treasure, and you have to print the following message: **"I've found the treasure!".** Then print the correct directions you went to in order to find the treasure.

Otherwise, print: **"The map is fake!"**.

### Input

* On the first line, you'll receive the **field dimensions** in the format: **"N M"**, where **N** is the number of **rows**, and **M** is the number of **columns**. They'll be separated by a single **space (" ")**.
* On the next N lines, you will receive a string representing the **respective row** of the **field**. The **positions** in every string will be **separated** by a single **space (" ").**
* On the next few lines, until you receive the command **"Finish"**, you will be given directions (**up, down, right, left**).

### Output

* There are two types of output:
* If you have **found** the treasure (the **last step** is on a position marked with '**X**') print the following output: **"I've found the treasure!".**

On the next line, print the correct directions you went to find the treasure (do not include the directions that made you **go out of the field** or **step on a tree**). The directions must be separated by a **comma** and **space** **(", ").** It should look like this:

**"The right path is {direction1}, {direction2}, …".**

* If you have **not found** the treasure, print the following message: **"The map is fake!".**

### Constraints

* The **field size** will be a 32-bit integer in the range **[0 … 2 147 483 647]**.
* The field will always have only one '**X**' and only one '**Y**'.
* If the steps are **invalid**, do **not include** them in the result.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 5 8  - - - T - - - T  - X - T T - - -  - - - - - - T -  - - Y - T - - T  - - - T - - - -  up  up  left  Finish | I've found the treasure!  The right path is up, up, left | **1. up 2. up 3. left**  - - - T - - - T - - - T - - - T - - - T - - - T  - X - T T - - - - X Y T T - - - - X/Y - T T - - -  - - Y - - - T - - - - - - - T - - - - - - - T -  - - - - T - - T - - - - T - - T - - - - T - - T  - - - T - - - - - - - T - - - - - - - T - - - - |
| 4 7  T - - T - T T  - - - - - X -  T - - - - - -  Y - - - - - T  left  right  right  up  Finish | The map is fake! | **1. left**  goes outside the field - stays in the same position.  **2. right** **3. right**  **4. up**  T - - T - T T T - - T - T T T - - T - T T  - - - - - X - - - - - - X - - - - - - X -  T - - - - - - T - - - - - - T - Y - - - -  - Y - - - - T - - Y - - - T - - - - - - T |

# 03. Elephant Sanctuary

A picture containing text

Description automatically generated

## Preparation

Download the skeleton provided in Judge. **Do not** change the **packages**!

**Pay attention to the name of the package (sanctuary), all the classes, their fields, and methods the same way they are presented in the following document. It is also important to keep the project structure as described.**

## Problem description

Your task is to create a repository that stores items by creating the classes described below.

First, write a Java class **Elephant** with the following fields:

* **name: String**
* **age: int**
* **retiredFrom: String**

The class **constructor** should receive a **name, age,** and **the place where the elephant was retired from.** You need to create the appropriate **getters and setters**. The class should override the **toString()** method in the following format:

**"{name} {age} - {retiredFrom}"**

**Next**, write a Java class **Habitat** that has **data** (a collection, which stores the Elephants in the Sanctuary). All entities inside the repository have the **same fields**. Also, the **Habitat** class should have those fields:

* **capacity: int**

The class **constructor** should receive **capacity**. Also, it should initialize the **data** with a new collection instance**.** Implement the following features:

* Field **data** – **List** that holds added elephants
* Method add(Elephant elephant) – **adds** an **entity** to the data **if** **there** **is** an **empty space** for the elephant.
* Method remove(String name) – removes the elephant by **given name,** if such **exists**, and **returns boolean**.
* Method **getElephant(String retiredFrom)** – returns the elephant **retired from the given place** or **null if no such elephant exists**.
* Method getOldestElephant() – returns the oldest Elephant.
* Getter getAllElephants() – **returns** the **number** of elephants.
* **getReport()** – **returns** a **String** in the following **format**:

**"Saved elephants in the park:  
 {name} came from: {retiredFrom}  
 {name} came from: {retiredFrom}  
 (…)**"

## Constraints

* The **age** of the elephants will always be a **positive number**.

## Examples

This is an example of how the **Habitat** class is **intended to be used**.

|  |
| --- |
| Sample code usage |
| // Initialize the repository Habitat park = new Habitat(10);  // Initialize entity Elephant firstElephant = new Elephant("Bobby", 10, "Thailand Zoo"); // Print Elephant System.*out*.println(firstElephant); //Bobby 10 - Thailand Zoo  // Add Elephant park.add(firstElephant);  // Remove Elephant System.*out*.println(park.remove("Bobby")); //true System.*out*.println(park.remove("Lola")); //false Elephant secondElephant = new Elephant("Bibi", 5, "Private Zoo"); Elephant thirdElephant = new Elephant("Lola", 7, "National Circus of Thailand"); park.add(secondElephant); park.add(thirdElephant);  // Get Oldest Elephant Elephant oldest = park.getOldestElephant(); System.*out*.println(oldest); //Lola 7 - National Circus of Thailand  Elephant elephant = park.getElephant("Private Zoo"); System.*out*.println(elephant); //Bibi 5 - Private Zoo  // All elephants in the park System.*out*.println(park.getAllElephants()); //2  // Information Report System.*out*.println(park.getReport());  //Saved elephants in the park:  //Bibi came from: Private Zoo  //Lola came from: National Circus of Thailand |

## Submission

Submit a **single .zip file** containing the **sanctuary package, with the classes inside (Elephant, Habitat, and the Main class)**, and there is no specific content required inside the Main class e. g. you can do any local testing of your program there. However, there should be a **main(String[] args)** method inside