# Autumn Cocktails



*Summer is over, autumn has come. For this purpose, we have prepared several cocktails that we think you will like.*

First, you will receive a sequence of **integers**, representing the number of ingredients in a single bucket. After that, you will be given another sequence of **integers** - the freshness level of the ingredients.

Your task is to **mix** them so you can produce the cocktails, listed in the table below with the **exact** freshness level.

|  |  |
| --- | --- |
| **Cocktail** | **Freshness Level needed** |
| Pear Sour | 150 |
| The Harvest | 250 |
| Apple Hinny | 300 |
| High Fashion | 400 |

To mix a cocktail, you have to take the **first** **bucket of** **ingredients** and the **last freshness level value**. The total freshness level is calculated by their **multiplication**. If the product of this operation **equals** one of the levels described in the table, you make the cocktail and **remove both** buckets with ingredients and freshness value. **Otherwise,** you should **remove the freshness level**, **increase** the ingredient value by **5, then remove it from the first position and add it at the end**. In case you have an ingredient with a value of **0** you have to **remove it** and continue mixing the cocktails.  
You need to stop making cocktails when you **run out** of buckets with ingredients **or** freshness level values.

Your task is considered done if you make at least **four** cocktails - **one of each type**.

## Input

* The first line of input will represent the values of buckets with ingredients - **integers**, separated by a **single space**.
* On the second line, you will be given the freshness values - **integers** again, separated by a **single space**.

## Output

* On the first line of output - print whether you've succeeded in preparing the cocktails
* "**It's party time! The cocktails are ready!**"**.**
* "**What a pity! You didn't manage to prepare all cocktails.**".
* On the next output line - print the **sum** of the ingredients **only if they are left** **any**
  + "**Ingredients left: {sum of the left ingredients}**".
* On the last few lines, you have to print the cocktails you **have made at least once,** ordered **alphabetically** in the format:

**" # {cocktail name} --> {amount}"**.

## Constraints

* All of the ingredients' values and freshness level values will be **integers** in the range **[0, 100].**
* We can have **more than one** mixed cocktail of the types specified in the table above.

## Examples

|  |  |  |
| --- | --- | --- |
| ****Input**** | ****Output**** | ****Comment**** |
| **10 10 12 8 10 12**  **25 15 50 25 25 15** | **It's party time! The cocktails are ready!**  **# Apple Hinny --> 2**  **# High Fashion --> 1**  **# Pear Sour --> 2**  **# The Harvest --> 1** | First, you take the **first** ingredient and the **last** freshness level value and **multiply** them - the result is 150 so we **make** a Pear Sour cocktail. Next, we have a product of 250 and The Harvest cocktail is **ready**. Then we **mix** the Apple Hinny cocktail by multiplying 12 and 25. The product of next ingredient value and freshness level value is 400 and we **make** High Fashion cocktails. The next pair is 10 and 15, we multiply them and mix one more Pear Sour. The last multiplication of 12 and 25 equals 300 and we make one more Apple Hinny. There are **no more ingredients and freshness values** so we stop mixing cocktails, but we have **one of each** cocktail type and print the **proper** message. |
| **12 20 0 6 19**  **12 12 25** | **What a pity! You didn't manage to prepare all cocktails.**  **Ingredients left: 55**  **# Apple Hinny --> 1** | **The first pair is 12 and 25, we mix the Apple Hinny cocktail and remove both of them.  Next, we take 20 and 12 - the product is 240 - we can't mix a cocktail, so we remove the freshness level value, increase the ingredient value by 5, remove it from the beginning of the buckets sequence and add it at the end. The next ingredient has a value of 0 - we remove it and continue.  The next pair is 6 and 12 - again we can't make a cocktail. After that we don't have more freshness level values, so we stop mixing drinks. The rest of the ingredients are 19, 25, and 11 with a sum of 55.** |

# Mouse and cheese



You will be given an integer **n** for the **size** of the mouse territory with a **square** shape. On the next **n** lines, you will receive the **rows** of the territory. The mouse will be placed in a **random position**, marked with the letter '**M**'. On random positions, there will be cheese, marked with **'c'**. There may also be a **bonus** on the territory. There will always be only one bonus. It will be **marked** with the **letter** - '**B**'. **All of the empty positions** will be marked with **'-'**.

Each turn, you will be given a **command** for the **mouse’s movement**.

The commands will be: "**up**", "**down**", "**left**", "**right**", "**end**".

If the mouse **moves** to **cheese**, it eats the cheese and increases the cheese it has eaten by one.

If it goes to a **bonus**, the mouse gets a bonus one move forward and then the bonus **disappears**. If the mouse **goes out** she can't return and the program ends. If the mouse receives the "**end**" command the program ends. The mouse needs **at least 5 eaten cheeses**.

### Input

* On the first line, you are given the integer **n** – the size of the **square** matrix.
* The **next n lines** hold the values for every **row**.
* On each of the next lines, until you receive the "**end**" command, you will receive a move command.

### Output

* On the first line:
  + If the mouse goes out of its territory print: **"Where is the mouse?"**.
* On the second line:
  + If the mouse couldn’t eat enough cheeses print: **"The mouse couldn't eat the cheeses, she needed {needed} cheeses more."**.
  + If the mouse has successfully eaten enough cheeses print: **"Great job, the mouse is fed {eaten cheeses} cheeses!"**.
* At the end print the matrix.

### Constraints

* The size of the **square** matrix will be between **[2…10]**.
* There will always be only one bonus, marked with '**B**.
* The mouse position will be marked with '**M**'.
* There won't be a case where a bonus moves the mouse out of its territory.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 3  M--  ccc  ---  right  right  down  down  left  end | The mouse couldn't eat the cheeses, she needed 4 cheeses more.  ---  cc-  -M- | 1) right 2) right 3) down 5) down  -M- --M --- ---  ccc ccc ccM cc-  --- --- --- --M    6) left  ---  cc-  -M- |
| 5  Mcc--  --B--  c-c-c  -----  ccccc  right  down  left  down  right  down  left  left  end | Where is the mouse?  The mouse couldn't eat the cheeses, she needed 3 cheeses more.  --c--  --B--  --c-c  -----  ccccc |  |

# Hotel



## Preparation

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

**Pay attention to the name of the package hotel, 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 people by creating the classes described below.

**Person**

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

* **name: String**
* **id: int**
* **age: int**
* **hometown: String – "n/a" by default**

The class **constructor** should receive the **name, id, age, and hometown**. You need to create the appropriate **getters and setters**. Override the **toString()** method in the following format:

**"Person {name}: {id}, Age: {age}, Hometown: {hometown}"**

**Hotel**

**Next**, write a Java class **Hotel** that has **a roster** (a collection that stores **Person** entities). All entities inside the repository have the **same fields**. Also, the **Hotel** class should have those **fields**:

* **name: String**
* **capacity: int**

The class **constructor** should receive the **name** and **capacity**, also it should initialize the **roster** with a new instance of the collection.Implement the following features:

* Method add(Person person) - **adds** an **entity** to the roster **if** **there** **is** **room** for it
* Method remove(String name) - removes a person by **given name,** if such **exists**, and **returns boolean**
* Method **getPerson(String name, String hometown)** – returns the people with the **given name** and **hometown** or **null** if there is no such person.
* Getter getCount() – **returns** the **number** of people.
* **getStatistics()** – **returns** a **String** in the following **format**:
  + **"The people in the hotel {name} are:  
    {Person1}  
    {Person2}  
    (…)**"

## Constraints

* The **names** of the people will be **always unique**.
* You will always have a person added before receiving methods manipulating the Hotel's people.

## Examples

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

|  |
| --- |
| Sample code usage |
| package hotel;  public class Main {  public static void main(String[] args) {  //Initialize the repository (hotel)  Hotel hotel = new Hotel("Hilton", 30);  //Initialize entity  Person person = new Person("Mark", 11111, 41, "Sofia");  //Print person  System.*out*.println(person);  //Person Mark: 11111, Age: 41, Hometown: Sofia   //Add person  hotel.add(person);  System.*out*.println(hotel.getCount()); //1  System.*out*.println(hotel.remove("Anna")); //false   Person firstPerson = new Person("Alice", 22121, 18, "London");  Person secondPerson= new Person("Lizzy", 31311, 24, "Varna");  Person thirdPerson = new Person("Lucy", 54122, 31, "Birmingham");  Person fourthPerson = new Person("Maria", 66611, 41, "Sofia");    //Add people  hotel.add(firstPerson);  hotel.add(secondPerson);  hotel.add(thirdPerson);  hotel.add(fourthPerson);  // Get person  Person personForGet = hotel.getPerson("Lucy", "Birmingham");  Person personForGet1 = hotel.getPerson("Anna", "Burgas");  System.out.println(personForGet); // Person Lucy: 54122, Age: 31, Hometown: Birmingham  System.out.println(personForGet1); // null  //Count  System.out.println(hotel.getCount()); // 5  //Remove Person  System.*out*.println(hotel.remove("Alice")); //true  // Get Statistics  System.out.println(hotel.getStatistics());  // The people in the hotel Hilton are:  // Person Mark: 11111, Age: 41, Hometown: Sofia  // Person Lizzy: 31311, Age: 24, Hometown: Varna  // Person Lucy: 54122, Age: 31, Hometown: Birmingham  // Person Maria: 66611, Age: 41, Hometown: Sofia  } } |