# Exam Preparation – 13 February 2023

## Bombs

**Link:** [**https://judge.softuni.org/Contests/Practice/Index/2471#0**](https://judge.softuni.org/Contests/Practice/Index/2471#0)

*Ezio still is learning to make bombs. With their help, he will save civilization. We should help Ezio to make his perfect bombs.*

You will be given **two sequences of integers, representing bomb effects** and **bomb casings**.

You need to start from the **first bomb effect** and try to mix it with the **last bomb casing**. If the **sum** of their values is **equal** to **any of the materials in the table below** – **create the bomb** **corresponding** to the **value** and **remove** **both** bomb materials. Otherwise, just **decrease** the value of the **bomb casing** **by 5**. You need to **stop** combining when you have **no more bomb effects** of **bomb casings,** or you successfully filled the bomb pouch.

**Bombs:**

* **Datura Bombs: 40**
* **Cherry Bombs: 60**
* **Smoke Decoy Bombs: 120**

To fill the bomb pouch, Ezio needs **three of each** of the **bomb types**.

### Input

* On the **first line**, you will receive the integers representing the **bomb effects**, **separated** by **", "**.
* On the **second line**, you will receive the integers representing the **bomb casing**, **separated** by "**,** ".

### Output

* On the **first** line of output – print one of these rows according to whether Ezio succeeded to fulfill the bomb pouch:
  + "**Bene! You have successfully filled the bomb pouch!**"
  + "You don't have enough materials to fill the bomb pouch."
* On the **second** line - print all bomb effects left:
  + If there are no bomb effects: "**Bomb Effects: empty**"
  + If there are effects: "**Bomb Effects: {bombEffect1}, {bombEffect2},** **(…)**"
* On the **third** line - print all bomb casings left:
  + If there are no bomb casings: "**Bomb Casings: empty**"
  + If there are casings: "**Bomb Casings: {bombCasing1}, {bombCasing2},** **(…)"**
* Then**,** you need to print **all** created bombs and the **count you have of them**, ordered **alphabetically**:
  + **"Cherry Bombs: {count}"**
  + **"Datura Bombs: {count}"**
  + **"Smoke Decoy Bombs: {count}"**

### Constraints

* All of the given numbers will be valid integers in the range **[0, 120]**.
* Don't have a situation with negative material.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | | **Output** |
| **5, 25, 50, 115**  **5, 15, 25, 35** | | **You don't have enough materials to fill the bomb pouch.**  **Bomb Effects: empty**  **Bomb Casings: empty**  **Cherry Bombs: 1**  **Datura Bombs: 2**  **Smoke Decoy Bombs: 1** |
| **Comment** | | |
| 1) 5 + 35 = 40 -> Datura Bomb. Remove both.  2) 25 + 25 = 50 -> can't create bomb. Bomb casing should be decreased with 5 -> 20  3) 25 + 20 = 45 -> can't create bomb. Bomb casing should be decreased with 5 -> 15  4) 25 + 15 = 40 -> Datura Bomb. Remove both  … | | |
| **Input** | **Output** | |
| **30, 40, 5, 55, 50, 100, 110, 35, 40, 35, 100, 80**  **20, 25, 20, 5, 20, 20, 70, 5, 35, 0, 10** | **Bene! You have successfully filled the bomb pouch!**  **Bomb Effects: 100, 80**  **Bomb Casings: 20**  **Cherry Bombs: 3**  **Datura Bombs: 4**  **Smoke Decoy Bombs: 3** | |
| **Comment** | | |
| **…**  After creating a bomb with bomb effect 35 and bomb casing 25, have created 3 Cherry bombs, 4 Datura bombs, and 3 Smoke Decoy bombs. All of the bomb types have 3 bombs and the program ends. | | |
| **Input** | O**utput** | |
| **10, 20**  **40, 35** | **You don't have enough materials to fill the bomb pouch.**  **Bomb Effects: empty**  **Bomb Casings: empty**  **Cherry Bombs: 1**  **Datura Bombs: 1**  **Smoke Decoy Bombs: 0** | |

## Treasure Hunt

**Link:** [**https://judge.softuni.org/Contests/Practice/Index/3529#1**](https://judge.softuni.org/Contests/Practice/Index/3529#1)

*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 |

## Rabbits

**Link:** [**https://judge.softuni.org/Contests/Practice/Index/1857#2**](https://judge.softuni.org/Contests/Practice/Index/1857#2)

### Preparation

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

**Pay attention to name the package (rabbits), 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 which stores departments by creating the classes described below.

### Rabbit

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

* **name: String**
* **species: String**
* **available: boolean - true by default**

The class **constructor** should receive **(name, species)**.

The class should also have the following methods:

* **getName()**
* **getSpecies()**
* **isAvailable()**
* **setAvailable()**
* Override the **toString()** method in the format:

**"Rabbit ({species}): {name}"**

### Cage

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

* **name: String**
* **capacity: int**
* **data**: **List<Rabbit>** that holds added rabbits

The class **constructor** should receive **(name**, **capacity)**, also it should initialize the **data** with a new instance of the collection.

Implement the following features:

* **getName()**
* **getCapacity()**
* add(Rabbit rabbit) method - **adds** an **entity** to the data **if** **there** **is** **room** for it
* removeRabbit(String name) method - removes a rabbit by **given name,** if such **exists**, and **returns boolean**
* removeSpecies(String species) method - removes **all rabbits** by given **species**
* sellRabbit(String name) method - **sell** (**set** its available **property** to **false** without removing it from the collection) the **first rabbit** with the **given name**, also **return** the **rabbit**
* sellRabbitBySpecies(String species) method - sells and returns **all rabbits** from that **species as a List**
* count() - **returns** the **number** of rabbits
* **report()** - **returns** a **String** in the following **format, including only not sold rabbits**:
  + **"**Rabbits **available at {cageName}:  
    {**Rabbit **1}  
    {**Rabbit **2}  
    (…)**"

### Constraints

* The **names** of the rabbits will be **always unique**.
* You will always have a rabbit added before receiving methods manipulating the Cage’s rabbits.

### Examples

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

|  |
| --- |
| Sample code usage |
| //Initialize the repository (Cage)  Cage cage = new Cage("Wildness", 20);  //Initialize entity  Rabbit rabbit = new Rabbit("Fluffy", "Blanc de Hotot");  //Print Rabbit  System.out.println(rabbit); // Rabbit (Blanc de Hotot): Fluffy  //Add Rabbit  cage.add(rabbit);  System.out.println(cage.count()); //1  //Remove Rabbit  cage.removeRabbit("Rabbit Name"); //false  Rabbit secondRabbit = new Rabbit("Bunny", "Brazilian");  Rabbit thirdRabbit = new Rabbit("Jumpy", "Cashmere Lop");  Rabbit fourthRabbit = new Rabbit("Puffy", "Cashmere Lop");  Rabbit fifthRabbit = new Rabbit("Marlin", "Brazilian");    //Add Rabbits  cage.add(secondRabbit);  cage.add(thirdRabbit);  cage.add(fourthRabbit);  cage.add(fifthRabbit);    //Sell Rabbit by name  System.out.println(cage.sellRabbit("Bunny")); //Rabbit (Brazilian): Bunny  //Sell Rabbit by species  List<Rabbit> soldSpecies = cage.sellRabbitBySpecies(("Cashmere Lop");  soldSpecies.forEach(f-> {  System.out.println(f.getName());   });  //Jumpy  //Puffy  System.out.println(cage.report());  //Rabbits available at Wildness:  //Rabbit (Blanc de Hotot): Fluffy  //Rabbit (Brazilian): Marlin |

### Submission

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