# Exam Preparation – 11 February 2022

## Bouquets

**Link:** [**https://judge.softuni.org/Contests/Practice/Index/2826#0**](https://judge.softuni.org/Contests/Practice/Index/2826#0)

*You want to go on a bouquets competition but to participate you have to make at least 5* *bouquets.*

You will be given **two sequences of integers, representing daffodils and tulips.** You need to start making bouquets **knowing that one bouquets needs 15 flowers**. Your goal is to make **at least 5 bouquets.**

You will start crafting from the **last tulips** and the **first daffodils**. If the **sum** of their values is **equal** to **15** – **create one bouquet** and **remove** them. While the sum is **bigger than 15**, keep **decreasing** the value of the **tulips by 2**. If the sum **is less than 15 you have to store them for later and remove them**. You need to **stop** combining when you have **no more daffodils or tulips**. In the end, if you have **any stored flowers you should make as many bouquets as you can with them.**

### Input

* On the **first line**, you will receive the integers representing the **tulips**, **separated** by **", "**.
* On the **second line**, you will receive the integers representing the **daffodils**, **separated** by "**,** ".

### Output

* Print whether you have succeeded making **at least 5 bouquets**:
  + **"You made it! You go to the competition with {count of bouquets} bouquets!"**
  + **"You failed... You need more {number} bouquets."**

### Constraints

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

### Examples

|  |  |
| --- | --- |
| ****Input**** | ****Output**** |
| **10, 15, 2, 7, 9, 13**  **2, 10, 8, 12, 0, 5** | **You made it! You go to the competition with 5 bouquets!** |
| ****Comment**** | |
| We start with the last tulips (13) and the first daffodils (2) -> 13 + 2 = 15 -> 15 = 15 So we create one bouquet and remove them bouth.  Next we have 9 + 10 = 19 -> 19 > 15 so we decrease the tulips by 2 -> 7 + 10 = 17 and we decrease the tulips by 2 -> 5 + 10 = 15 and we create one more bouquet and remove them.  Next, we have 7 + 8 = 15. We create one more bouquet and remove them.  Next, we have 2 + 12 = 14 -> 14 < 15 so we have to store theire sum for later and remove theme.  Next, we have 15 + 0 = 15 so we create one more bouquet.  And last we have 10 + 5 = 15, we create one more bouquet and stop mixing because we don’t have any flowers left.  Now we have a tottal of 5 bouquets and we also have 14 flowers left but we cant create bouquet because 14 < 15. | |

|  |  |
| --- | --- |
| ****Input**** | ****Output**** |
| **10, 5, 3, 7, 8**  **5, 10, 8, 7, 6** | **You failed... You need more 1 bouquets.** |
| ****Comment**** | |
| We start with 8 + 5 = 13 -> 13 < 15 -> we have to store their sum for later and remove them.  Next, we have 7 + 10 = 17 -> we decrease the tulips by 2 -> 5 + 10 = 15 -> 15 = 15 and we create one bouquet.  Next, we have 3 + 8 = 11 -> 11 < 15 -> we store their sum for later and remove them.  Next, we have 5 +7 = 12 -> we store their sum for later and remove them.  Next, we have 10 + 6 = 16 -> 16 > 15 we decrease the tulips by 2 -> 8 + 6 = 14 and we store their sum for later and remove them.  We stop crafting because we don’t have any flowers left and we have 1 bouquet and 50 stored flowers. We create 3 more bouquets because 3 \* 15 = 45 -> 50 – 45 = 5 -> 5 < 15. | |

## Selling

**Link:** [**https://judge.softuni.org/Contests/Practice/Index/2704#1**](https://judge.softuni.org/Contests/Practice/Index/2704#1)

*You successfully started your cooking journey, so now you need to sell the products from your basket in the bakery in order to collect your price.*

You will be given an integer **n** for the **size** of the bakery with **square** shape. On the next **n** lines, you will receive the **rows** of the bakery. You will be placed on a **random position**, marked with the letter '**S**'. On random positions there will be clients, marked with a **single digit**. There **may** also be **pillars**. Their **count** will be either **0** or **2** and they are **marked** with the **letter** - '**O**'. **All of the empty positions** will be marked with **'-'**.

Each turn, you will be given **commands** for the **your movement**. Move commands will be: "**up**", "**down**", "**left**", "**right**". If you **move** to a **client**, you **collects** **the price** **equal** to the **digit** **there** and the client **disappears**. If you move to a **pillar**, you move on the **position** of the **other pillar** and then **both** pillars **disappear**. If you **go** **out** of the bakery, you **disappear** from the bakery and you are out of there. You need **at least** **50 dollars** to rent your own Bakery

When **you are out of the bakery or you collect enough money,** the program **ends**.

### Input

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

### Output

* On the first line:
  + If the player goes to the void, print: "**Bad news, you are out of the bakery.**"
  + If the player collects enough star power, print: "**Good news! You succeeded in collecting enough money!**"
* On the second line print all star power collected: "**Money: {money}**"
* In the end print the matrix.

### Constraints

* The size of the **square** matrix will be between **[2…10].**
* There will **always** be **0** or **2** pillars, marked with the **letter** - '**O**'.
* Your position will be marked with '**S**'.
* You will **always** go out of the bakery or collect enough money.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 5  SO---  -----  -----  -----  ----O  right  right | Bad news, you are out of the bakery.  Money: 0  -----  -----  -----  -----  ----- | The first command is right. You move to **one of the pillars** and then **appears** on the other side of it (4,4).  The bakery looks like this after the first command:  -----  -----  -----  -----  ----S  The second command is right. You go of the bakery. |
| 6  S98---  99----  666666  ------  --77--  -6-6-6  right  right  down  left  left  down  right  right | Good news! You succeeded in collecting enough money!  Money: 53  ------  ------  --S666  ------  --77--  -6-6-6 | Here we have **no** pillars and bakery rich of clients.  You manage to collect **enough** money **without** **going out** of the bakery.  The clients you have selled food to have disappeared and we can see where you were when you collected the last neeeded money (2,2). |

## Grooming Salon

**Link:** [**https://judge.softuni.org/Contests/Practice/Index/2997#2**](https://judge.softuni.org/Contests/Practice/Index/2997#2)

**Preparation**

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

**Pay attention to name the package groomingSalon, 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 items by creating the classes described below.

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

* **name: String**
* **age: int**
* **owner: String**

The class **constructor** should receive **all fields.** You need to create the appropriate **getters and setters**. The class should override the **toString()** method in the following format:

**"{name} {age} - ({owner})"**

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

* **capacity: int**

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

* Field **data** – **List** that holds added pets
* Method add(Pet pet) – **adds** an **entity** to the data **if** **there** **is** an **empty place** inthegrooming salon for the pet.
* Method remove(String name) – removes the pet by **given name,** if such **exists**, and **returns boolean**.
* Method **getPet(String name, String owner)** – returns the pet with the **given name** and **owner** or **null if no such pet exists**.
* Getter getCount – **returns** the **number** of pets.
* **getStatistics()** – **returns** a **String** in the following **format**:
  + **"** **The grooming salon has the following clients:  
    {name} {owner}  
    {name} {owner}**

**(…)**"

## Constraints

* The **combinations** of **names** and **owners** will **always be unique**.
* The **age** of the pets will always be **positive**.

## Examples

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

|  |
| --- |
| Sample code usage |
| // Initialize the repository  GroomingSalon salon = new GroomingSalon(20);  // Initialize entity  Pet dog = new Pet("Ellias", 5, "Tim");  // Print Pet  System.out.println(dog); // Ellias 5 - (Tim)  // Add Pet  salon.add(dog);  // Remove Pet  System.out.println(salon.remove("Ellias")); // true  System.out.println(salon.remove("Pufa")); // false  Pet cat = new Pet("Bella", 2, "Mia");  Pet bunny = new Pet("Zak", 4, "Jon");  salon.add(cat);  salon.add(bunny);  // Get Pet  Pet pet = salon.getPet("Bella", "Mia");  System.out.println(pet); // Bella 2 - (Mia)  // Count  System.out.println(salon.getCount()); // 2  // Get Statistics  System.out.println(salon.getStatistics());  // The grooming salon has the following clients:  //Bella Mia  //Zak Jon |

## Submission

Zip all the files in the project folder except **bin** and **obj** folders

Submit **single .zip file**, containing **groomingSalon package, with the classes inside (Pet, GroomingSalon 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 you program there. However there should be **main(String[] args)** method inside.