# Exam Preparation – 10 February 2022

## Autumn Cocktails

**Link:** [**https://judge.softuni.org/Contests/Practice/Index/3203#0**](https://judge.softuni.org/Contests/Practice/Index/3203#0)

*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 cocktail. 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 types 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 with 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, 11 with the sum of 55.** |

## Cooking journey

**Link:** [**https://judge.softuni.org/Contests/Practice/Index/2826#1**](https://judge.softuni.org/Contests/Practice/Index/2826#1)

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

You will be given an integer **n** for the **size** of the pastry shop with **square** shape. On the next **n** lines, you will receive the **rows** of the pastry shop. 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** - '**P**'. **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 pastry shop, you **disappear** from the pastry shop and you are out of there. You need **at least** **50 dollars** to rent your own Pastry shop.

When **you are out of the pastry shop** **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 pastry shop.**"
  + 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** - '**P**'.
* Your position will be marked with '**S**'.
* You will **always** go out of the pastry shop or collect enough money.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 5  SP---  -----  -----  -----  ----P  right  right | Bad news, you are out of the pastry shop.  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 pastry shop looks like this after the first command:  -----  -----  -----  -----  ----S  The second command is right. You go of the pastry shop. |
| 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 pastry shop rich of clients.  You manage to collect **enough** money **without** **going out** of the pastry shop.  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). |

## Car Dealership

**Link:** [**https://judge.softuni.org/Contests/Practice/Index/2805#2**](https://judge.softuni.org/Contests/Practice/Index/2805#2)

**Preparation**

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

**Pay attention to name the package dealership, 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.

**Car**

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

* **manufacturer: String**
* **model: String**
* **year: int**

The class **constructor** should receive **manufacturer, model** and **year**.You need to create the appropriate **getters and setters**. Override the **toString()** method in the following format:

**"{manufacturer} {model} ({year})"**

**Dealership**

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

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

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

* Field **data** – **Collection** that holds added cars
* Method add(Car car) – **adds** an **entity** to the data **if** **there** **is** an **empty cell** for the car.
* Method buy(String manufacturer, String model) – removes the car by **given manufacturer and model,** if such **exists**, and **returns boolean**.
* Method getLatestCar() – returns the **latest** car (by year) or null if have no cars.
* Method **getCar(String manufacturer, String model)** – returns the car with the **given manufacturer** and **model** or **null** if there is no such car.
* Getter getCount() – **returns** the **number** of cars.
* **getStatistics()** – **returns** a **String** in the following **format**:
  + **"** **The cars are in a car dealership {name}:  
    {Car1}  
    {Car2}  
    (…)**"

## Constraints

* The **combinations** of **manufacturers** and **models** will be **always unique**.
* The **year** of the cars will always be **positive**.
* There won't be cars with the same years.

## Examples

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

|  |
| --- |
| Sample code usage |
| // Initialize the repository  Dealership dealership = new Dealership("Autofest", 5);  // Initialize entity  Car volvo = new Car("Volvo", "XC70", 2010);  // Print Car  System.out.println(volvo); // Volvo XC70 (2010)  // Add Car  dealership.add(volvo);  // Remove Car  System.out.println(dealership.buy("Volvo", "XC90")); // false  System.out.println(dealership.buy("Volvo", "XC70")); // true  Car peugeot = new Car("Peugeot", "307", 2011);  Car audi = new Car("Audi", "S4", 2005);  dealership.add(peugeot);  dealership.add(audi);  // Get Latest Car  Car latestCar = dealership.getLatestCar();  System.out.println(latestCar); // Peugeot 307 (2011)  // Get Car  Car audiS4 = dealership.getCar("Audi", "S4");  System.out.println(audiS4); // Audi S4 (2005)  // Count  System.out.println(dealership.getCount()); // 2  // Get Statistics  System.out.println(dealership.getStatistics());  // The cars are in a car dealership Autofest:  // Peugeot 307 (2011)  // Audi S4 (2005) |

## Submission

Submit **single .zip file**, containing **dealership package, with the classes inside (Car, Dealership 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.