# Bombs

*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**** | ****Output**** |
| **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** |

# Snake

*Everyone remembers the old snake games. Now is the time to create our own snake game.*

You will be given an integer **n** for the **size** of the territory with a **square** shape. On the next **n** lines, you will receive the **rows** of the territory. The snake will be placed in a **random position**, marked with the letter '**S**'. There will also be food on random positions, marked with **'\*'**. The territory may have a **lair**. The lair will have two burrows **marked** with the **letter** - '**B**'. **All of the empty positions** will be marked with **'-'**.

Each turn, you will be given a **command** for the **snake’s movement**. When the snake moves it leaves a trail marked with **'.'**

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

If the snake **moves** to **food**, it will eat the food, which will increase food quantity with one.

If it goes inside a **burrow**, it **goes out** on the **position** of the **other burrow** and then **both** burrows **disappear**. If the snake **goes out** of its territory, the game is over. The snake needs **at least** **10 food quantities** to be fed.

If **the snake** **goes outside of its territory or has eaten enough food**, the game should **end**.

### 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, you will get a move command.

### Output

* On the first line:
  + If the snake goes out of its territory, print: "**Game over!**"
  + If the snake eats enough food, print: "**You won! You fed the snake.**"
* On the second line print, all food is eaten: "**Food eaten: {food quantity}**"
* At the end print the matrix.

### Constraints

* The size of the **square** matrix will be between **[2…10].**
* There will **always** be **0** or **2** burrows, marked with - '**B**'.
* The snake position will be marked with '**S**'.
* The snake will **always** either go out of its territory or eat enough food.
* There will be no case in which the snake will go through itself.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 6  -----S  ----B-  ------  ------  --B---  --\*---  left  down  down  down  left | Game over!  Food eaten: 1  ----..  ----.-  ------  ------  --.---  --.--- | 1) left 2) down 3) down 5) down  ----S. ----.. ----.. ----..  ----B- ----.- ----.- ----.-  ------ ------ ------ ------  ------ ------ ------ ------  --B--- --S--- --.--- --.---  --\*--- --\*--- --S--- --.---  3) eat the food: '\*' (5, 2)  5) the snake goes out of its territory and the program ends |
| 7  --\*\*\*S-  --\*----  --\*\*\*--  ---\*\*--  ---\*---  ---\*---  ---\*---  left  left  left  down  down  right  right  down  left  down | You won! You fed the snake.  Food eaten: 10  --....-  --.----  --...--  ---..--  ---S---  ---\*---  ---\*--- |  |

# Parking

*Parking games are also among the popular games. Let's create one.*

## Preparation

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

**Pay attention to name the package parking, 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 fields:

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

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

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

**Parking**

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

* **type: String**
* **capacity: int**

The class **constructor** should receive **type** 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 remove(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 parked in {parking type}:  
    {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 of the same year.

## Examples

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

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

## Submission

Submit a **single .zip file**, containing the **parking package, with the classes inside (Car, Parking, 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 your program there. However, there should be a **main(String[] args)** method inside.