# Exam Preparation – 9 June 2023

## Meeting

**Link:** [**https://judge.softuni.org/Contests/Practice/Index/3251#0**](https://judge.softuni.org/Contests/Practice/Index/3251#0)

The first line will give you **a sequence of integers representing males**. Afterward, you will be given another **sequence of integers representing females**.

You have to start from the **first female** and try to match it with the **last male**.

* If their **values** are **equal**, you have to **match them** and **remove both** of them. Otherwise, you should **remove only the female** and **decrease** the **value** of the **male** by **2**.
* If someone’s value is **equal to or below** **0**, you should **remove him/her** from the records **before** trying to **match** him/her with anybody.
* Special case - if someone’s **value is divisible by 25** **without remainder**, you should **remove** **him/her** **and** the **next person** of the **same gender**.

You need to **stop** **matching** people when you have **no more females or males**.

### Input / Constraints

* On the **first line**, you will receive the integers, representing the **males**, **separated** by a **single space**.
* On the **second line,** you will receive the integers, representing the **females**, **separated** by a **single space**.
* All of the given numbers will be valid integers in the range **[-100, 100]**.

### Output

* On the first line - print the number of successful matches:
  + "**Matches: {matchesCount}"**
* On the second line - print all males left:
  + If there are no males: "**Males left: none**"
  + If there are males: "**Males left: {male1}, {male2}, {male3}, (…)**"
* On the third line - print all females left:
  + If there are no females: "**Females left: none**"
  + If there are females: "**Females left: {female1}, {female2}, {female3}, (…)**"

### Examples

|  |  |  |
| --- | --- | --- |
| ****Input**** | ****Output**** | ****Comment**** |
| **3 6 9 12**  **12 9 6 1 25 25** | **Matches: 3**  **Males left: 1**  **Females left: none** | The first pair is the **first female** with a value of 12 and the **last male** of value 12, their **values are equal**, so we **match them,** therefore - **remove them** from the **records**. Then we have **two more matches** (9 == 9 and 6 == 6). But the value of the **next male is 3** and the value of the **next female is 1**, it’s **not a match** and we **remove** the **female** and **reduce** the **male’s value** by 2. We have a **female** whose **value** is **25** and we have to **remove** **her** and the **next female**. Then, we **print** the desired **output**. |
| **3 0 3 6 9 0 12**  **12 9 6 1 2 3 15 13 4** | **Matches: 4**  **Males left: none**  **Females left: 15, 13, 4** |  |

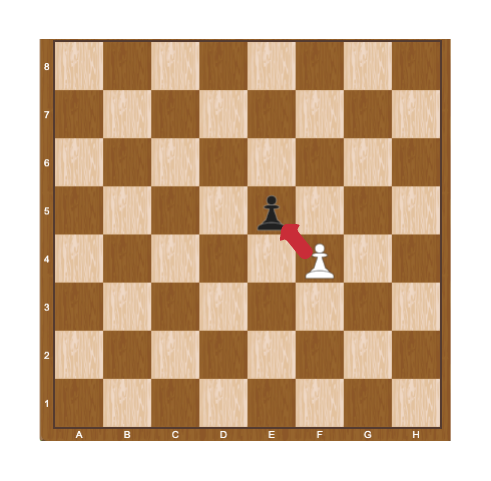
## Pawn Wars

**Link:** [**https://judge.softuni.org/Contests/Practice/Index/3350#1**](https://judge.softuni.org/Contests/Practice/Index/3350#1)

A chessboard has 8 rows and 8 columns. Rows also called ranks, are marked from number 1 to 8, and columns are marked from **a** to **h**. We have a total of 64 squares, each square is represented by a combination of letters and a number (a1, b1, c1, etc.). In this problem colors of the board will be ignored.

We will play the game with two pawns **white (w)** and **black (b)**, where they can:

* Only move forward:
  + - White (**w**) moves from the 1st rank to the 8th rank direction.
    - Black (**b**) moves from 8th rank to the 1st rank direction.
* Can move only 1 square at a time.
* Can capture another pawn only diagonally:



When a pawn reaches the **last rank**, for **white this is the 8th** rank, and **for black, this is the 1st** rank, can be **promoted** to a queen.

Two pawns (**w** and **b**) will be placed on two random squares of the bord. The **first** move is always made by the **white pawn** (**w**), then black moves (b), then white (w) again, and so on. When **a pawn marches forward**, the **previous position** is marked by "-" (dash).

Some rules will be applied when moving paws:

* If the two pawns interact diagonally, the player, in turn, must capture the opponent’s pawn. When a pawn capture another pawn the game is over and "**Game over! {White/Black} capture on {coordinates}.**" is printed to the console.

**Example:**

White pawn is on the move and captures black in "**e5**". We print "**Game over! White capture on e5.**"

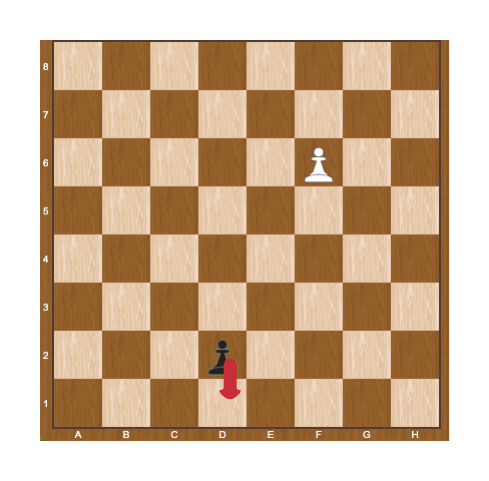
A picture containing text, checker

Description automatically generated

* If no capture is possible, the pawns keep on moving until one of them reaches the last rank. When one of the pawns reaches the last rank we print: "**Game over! {White/Black} pawn is promoted to a queen at {coordinates}.**"

**Example:**

It is black**'**s turn and the pawn reaches the d1 square, we print "**Game over! Black pawn is promoted to a queen at d1.**"



### Constraints

* The input will be always valid.
* The matrix will always be 8x8.
* There will be no case where two pawns are placed on the same square.
* There will be no case where two pawns are placed on the same column.
* There will be no case where black/white will be placed on the last rank.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| ------b-  --------  --------  --------  --------  -w------  --------  -------- | Game over! White pawn is promoted to a queen at b8. | We start by pushing the white pawn to b4, next, we push the black pawn to g7:  --------  ------b-  --------  --------  -w------  --------  --------  --------  Then white play b5, black play g6:  --------  --------  ------b-  -w------  --------  --------  --------  --------  …  Capturing is not possible here, so after a few more moves, the white pawn is promoted to a queen on b8. |
| --------  --------  --------  --------  --------  b-------  -w------  -------- | Game over! White capture on a3. | Here white captures black on a3 in the first move:  --------  --------  --------  --------  --------  w-------  --------  -------- |

## Easter Basket

Link: [**https://judge.softuni.org/Contests/Practice/Index/3402#2**](https://judge.softuni.org/Contests/Practice/Index/3402#2)

*It’s Easter and we have a basket full of Easter eggs. We want the basket to look beautiful, so we have to do some changings.*

### Preparation

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

**Pay attention to name the package easterBasket, 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 Easter eggs by creating the classes described below.

First, write a class **Egg** with the following properties:

* **color: String**
* **strength: int**
* **shape: String**

The class **constructor** should receive **color, strength and shape**. You need to create the appropriate **getters and setters**. Override the **toString()** method in the following format:  
**"{color} egg, with {strength} strength and {shape} shape."**

**Next**, write a class **Basket** that has **data** (a List which stores the entity **Egg**). All entities inside the repository have the **same properties**. Also, the **Basket** class should have those **properties**:

* **material: String**
* **capacity: int**

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

* **List<Egg> data** - **collection** that holds added eggs
* **Method addEgg(Egg egg)** – **adds** an **entity** to the data **if** **there** **is** **room** for it
* **Method removeEgg(string color)** – removes an egg by **given color,** if such **exists**, and **returns boolean** (true if it is removed, otherwise – false)
* **Method getStrongestEgg()**– **returns the strongest egg**
* **Method getEgg(string color)** – **returns** the **egg** with the **given color**
* **Method getCount** – **returns** the **number** of **eggs**
* **Method report()** – **returns** a **string** in the following **format** (print the eggs in **order of appearance**):
  + **"{material} basket contains:  
    {Egg1}  
    {Egg2}  
    (…)"**

### Constraints

* The **color** and **strength** of the eggs will be **always unique**.
* You will always have an egg added before receiving methods manipulating the Basket’s eggs.

### Examples

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

|  |
| --- |
| **Sample code usage** |
| //Initialize the repository (Basket)Basket basket = **new** Basket(**"**Wood**"**, 20); //Initialize entity (Egg)Egg egg = **new** Egg(**"**Red**"**, 10, **"**oval**"**); //Print EggSystem.***out***.println(egg); //Red egg, 10 strength, oval shape//Add Eggbasket.addEgg(egg);  //Remove EggSystem.***out***.println(basket.removeEgg(**"**Pink**"**)); //FalseEgg secondEgg = **new** Egg(**"**Green**"**, 9, **"**pointy**"**);  //Add Eggbasket.addEgg(secondEgg);  //Get strongest eggEgg strongestEgg = basket.getStrongestEgg();  System.***out***.println(strongestEgg);  //Get eggEgg getEgg = basket.getEgg(**"**Green**"**); //Green egg with 9 strength, pointy shapeSystem.***out***.println(getEgg);   System.***out***.println(basket.report()); //Wood basket contains: //Red egg, with 10 strength and oval shape. //Green egg, with 9 strength and pointy shape. |

### Submission

Submit **single .zip file**, containing **easterBasket** package, **with the classes inside** (**Basket** and **Egg** 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.