# Exam Preparation – 16 October 2023

## Food Finder

**Link:** [**https://judge.softuni.org/Contests/Practice/Index/3350#0**](https://judge.softuni.org/Contests/Practice/Index/3350#0)

You will be given **two sequences of characters, representing vowels and consonants**. Your task is to start checking if the following words could be created:

* **"pear"**
* **"flour"**
* **"pork"**
* **"olive"**

Start by taking the **first character** of the **vowels collection** and the **last character** from the **consonants collection.** Then check if these letters are present in one or more of the given words. If these letters are present, you should store the information. Then process to the next couple of letters until there are no more **consonant** letters left.

A **letter (vowels or consonants) could participate in more than one word, for example:**

The letter **'**o**'** is present in **"flour", "pork",** and **"olive".**

The letter **'**l**'** is present in **"flour",** and **"olive".**

**Keep in mind that:**

* A **vowel** letter is always returned to the collection, whether used or not.
* A **consonant** letter is always removed from the collection, whether used or not.

As a result, you should **check how many of the given words were** found and print:

**"Words found: {numberOfWordsFound}**

**{wordOne}**

**{wordTwo}**

**…"**

**Look at the provided examples for a better understanding of the problem.**

### Input

* On the **first line**, you will receive characters representing the **vowels**, **separated** by a single space (**" "**).
* On the **second line**, you will receive characters representing the **consonants**, **separated** by a single space (**" "**)**.**

### Output

* As a result, print on the first line how many words have been found and on the next N lines, every word:

**"Words found: {numberOfWordsFound}**

**{wordOne}**

**{wordTwo}**

**…"**

**Print words in the same order as in the problem's description.**

### Constraints

* All letters will be lowercase.
* All letters in the given words are unique.
* There will be no case where no word is matched.
* The letter **'**y**' will be always vowel.**

### Examples

|  |  |
| --- | --- |
| ****Input**** | ****Output**** |
| **e a u o**  **p r l x f** | **Words found: 2**  **pear**  **flour** |
| ****Comment**** | |
| We start by taking the first two letters **'e' and 'f'. We have concurrence, 'e' is found into "olive" and "pear", 'f' is found into "flour".**  **We add 'e' to the end of the vowels collection (a u o e) and remove 'f' out of the consonant collection (p r l x).**  Onto the next iteration we continue **'a' and 'x', where 'a' is found in "pear", and 'x' is not located into any word, 'a' is added at the end of the vowels collection (u o e a) and remove 'x' (p r l).**  **Next, we have 'u' and 'l', where both letters are found in "flour" and 'l' is found in "**olive**", we add 'u' back in the collection (o e a u) and remove 'l' (p r).**  **Next, we have 'o' and 'r', 'o' is found in "**flour**"**, **"**pork**"**, and **"**olive**"**, and **'r' is found in "**pear**"**, **"**flour**"**, **"**pork**"**, we add **'o' back in the collection (e a u o) and remove 'r' (p).**  **In this iteration, one word was found: "**flour**"**.  In the last iteration, we take **'e' and 'p'. We have already found all 'e' letters so far, so we only search for 'p', which is found in "**pear**"** and **"**pork**"**.  As a result, we found two words **"**pear**"** and **"**flour**",** so we print the corresponding output.  Final result: **"**pear**"**, **"**flour**"**, **"**pork, **"**olive**"**. | |
|  | |
| **a o y**  **b h p j r n k** | **Words found: 1**  **pork** |

## Throne Conquering

**Link:** [**https://judge.softuni.org/Contests/Practice/Index/3251#1**](https://judge.softuni.org/Contests/Practice/Index/3251#1)

After Paris got into Sparta, he has to fight his way to Helen’s chamber. To do that, he has to walk through the city where dangerous enemies are watching out for threats, but he also has to be careful not to get exhausted and not be able to proceed with his mission. If Paris successfully reaches her chamber, they safely escape from the Spartans.

A standard field of Sparta looks like this:

|  |  |
| --- | --- |
| **Field of Sparta** | **Legend** |
| ------H--- -------S-- --S------- ---------- -----P---- | P 🡺 **Paris**, the player character  S🡺 **Spartan, enemy**  H 🡺 **Helen**  - 🡺 **Empty space** |

Each turn proceeds as follows:

* **First**, Spartan **spawns** on the given indices.
* **Then, Paris** moves in a direction, which **decreases** his energy by 1.
  + It can be "**up**", "**down**", "**left**", "**right**".
  + If Paris tries to move **outside** of the field, he **doesn’t** move but **still** has his energy **decreased**.
* If an enemy is in the **same cell** where Paris moves, Paris fights him, which **decreases** his energy by 2. If Paris’ energy **drops** at 0 or below, he **dies** and you should mark his position with ‘**X**’.
* If Paris **kills** the enemy successfully, the enemy **disappears**.
* If Paris reaches the **index** where **Helen** is, **they both run away** (disappear from the field)**, even if his energy is 0 or below.**

### Input

* On the **first line** of input, you will receive **e** – **the energy** Paris has.
* On the **second line** of input, you will receive **n** – the **number of rows** the field of Sparta will consist of  
  range: **[5-20].**
* On the next **n lines**, you will receive how each row looks.
* Then, **until** Paris dies, or reaches Helen, you will receive a **move command** and **spawn row and column**.

### Output

* If Paris is **running out of energy**, print "Paris died at {row};{col}."
* If Helen is **abducted**, print "Paris has successfully abducted Helen! Energy left: {energy}"
* Then, in all cases, **print** the **final state of the field** on the **console**.

### Constraints

* The field will always be **rectangular**.
* Paris will **always** run out of energy or **reach Helen**.
* There will be **no case** with spawn on **invalid** indices.
* There will be **no case** with **two enemies on the same cell**.
* There will be **no case** with enemy **spawning** on the indices **where Paris is**.
* There will be **no case** with enemy **spawning** on the indices **where Helen is**.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 100  5  --H--  -----  -----  -----  --P--  up 3 0  up 3 1  up 3 2  up 3 3 | Paris has successfully abducted Helen! Energy left: 96  -----  -----  -----  SSSS-  ----- | Turn 1: An enemy spawns at [3;0], Paris moves to [3;2], and his energy decreases by 1.  Turn 2: An enemy spawns at [3;1], Paris moves to [2;2], and his energy decreases by 1.  Turn 3: An enemy spawns at [3;2], Paris moves to [1;2], and his energy decreases by 1.  Turn 4: An enemy spawns at [3;3], Paris moves to [0;2], his energy decreases by 1, but he also moves to the index where Helen is – they both run away. |
| 3  5  --H--  -----  -----  -----  --P--  up 3 2 | Paris died at 3;2.  --H--  -----  -----  --X--  ----- | Turn 1: An enemy spawns at [3;2], Paris moves to [3;2], his energy decreases by 1, and fights the enemy at that index. Paris’ energy is decreased by 2, dropping it to 0 or below => Paris dies. |
| 3  5  --H--  -----  -----  -----  --P--  left 1 0  down 2 0  up 3 0 | Paris died at 3;1.  --H--  S----  S----  SX---  ----- | Turn 1: An enemy spawns at [1;0], Paris moves to [4;1], and his energy decreases by 1.  Turn 2: An enemy spawns at [2;0], Paris tries to move down, but [5;1] is an invalid index, so he stays at [4;1]. His energy still decreases.  Turn 3: An enemy spawns at [3;0], Paris moves to [3;1], his energy drops to 0 and he cannot continue his mission. |

## Workout

**Link:** [**https://judge.softuni.org/Contests/Practice/Index/3482#2**](https://judge.softuni.org/Contests/Practice/Index/3482#2)

### Preparation

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

**Pay attention to the name of the package workout, 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 workout, which stores exercises by creating the classes described below.

### Exercise

First, write a Java class **Exercise** with the following properties:

* **name: String**
* **muscle: String**
* **burnedCalories: int**

The class **constructor** should receive **name, muscle,** and **burnedCalories**. You need to create the appropriate **getters and setters**. Override the **ToString()** method in the following format:

**"Exercise: {name}, {muscle}, {burnedCalories}"**

### Workout

**Next**, write a Java class **Workout** that has **exercises** (**List**, which stores the entity **Exercise**). All entities inside the repository have the **same fields**. Also, the Workout class should have those fields:

* **type: String**
* **exerciseCount: int**

The class **constructor** should receive **type** and **exerciseCount**, also it should initialize the **exercises** with a new instance of the list**.** Implement the following features:

* Field **exercises** – **List** that holds added exercises.
* Method **addExercise(Exercise exercise)** – **adds** an entity to the exercises If there is still **space on** the workout sheet **(exerciseCount).**
* Method removeExercise(String name, String muscle) – removes the exercise by **given name and muscle,** if such **exists**, and **returns boolean**.
* Method **getExercise(String name, String muscle)** – returns the exercise with the given **name** and **muscle** or **null** if there is no such exercise.
* Method getMostBurnedCaloriesExercise() – returns the exercise which is burned the most calories or null if there are no exercises.
* Getter getExerciseCount() – **returns** the **number** of exercises.
* **getStatistics()** – **returns** a **String** in the following **format**:
  + **"Workout type: {workout type}  
    Exercise: {Exercise1}  
    Exercise: {Exercise2}  
    (…)**"

### Constraints

* The **combinations** of **names** and **muscles** will be **always unique**.
* The **burned calories** from the exercises will always be **positive**.
* There won't be exercises with the same burned calories.

### Examples

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

|  |
| --- |
| **Sample code usage** |
| // Initialize the repository  Workout workout = newWorkout("strength", 3);  // Initialize entity  Exercise exercise = newExercise("Bench Press", "chest", 30);  // Print Exercise  System.out.println(exercise); // Exercise: Bench Press, chest, 30  // Add Exercise  workout.addExercise(exercise);  // Remove Exercise  System.out.println(workout.removeExercise("Bench Press", "arms")); // falseSystem.out.println(workout.removeExercise("Bench Press", "chest")); // true  //Get exerciseSystem.out.println(workout.getExercise("Bench Press", "chest")); // null  Exercise secondExercise = newExercise("Deadlift", "back", 50); Exercise thirdExercise = newExercise("Barbell Curl", "biceps", 25); Exercise fourthExercise = newExercise("Squats", "legs", 60); Exercise fifthExercise = newExercise("Deadlift", "legs", 55);  workout.addExercise(secondExercise); workout.addExercise(thirdExercise); workout.addExercise(fourthExercise); workout.addExercise(fifthExercise);  //Get mostBurnedCaloriesExercise  Exercise mostBurnedCaloriesExercise = workout.getMostBurnedCaloriesExercise(); System.out.println(mostBurnedCaloriesExercise); // Exercise: Squats, legs, 60  Exercise exerciseByNameAndMuscle = workout.getExercise("Deadlift", "back"); System.out.println(exerciseByNameAndMuscle); // Exercise: Deadlift, back, 50  // Count  System.out.println(workout.getExerciseCount()); // 3System.out.println(workout.removeExercise("Barbell Curl", "biceps")); // true  // Print Statistics  System.out.println(workout.getStatistics());  //Workout type: strength //Exercise: Deadlift, back, 50 //Exercise: Squats, legs, 60 |

### Submission

Submit a **single .zip file**, containing the **workout package, with the classes inside (Exercise, Workout, 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.