# Exam Preparation – 15 October 2021

## Magic box

**Link:** [**https://judge.softuni.bg/Contests/Practice/Index/2805#0**](https://judge.softuni.bg/Contests/Practice/Index/2805#0)

*Each member of your family has a magic box of items and now it's your turn, so you will buy a few.*

Every purchase gives you two magic boxes and they are represented as a sequence of integers. First, you will be given **a sequence of integers, representing the first magic box**. Afterwards, you will be given another **sequence of integers representing the second magic box**.

You need to start from the **first item** in the first box and **sum** it with the last item in the second box. If the **sum** of their values is **an even number,** add the **summed** item to **your collection of claimed items** and **remove** them **both** from the boxes. Otherwise, remove the last item from the second box and add it at the last position in the first box. You need to **stop** summing items when one of the boxes becomes empty.

If the first magic box becomes empty print:

**"First magic box is empty."**

If the second magic box becomes empty print:

**"Second magic box is empty."**

In the end you need to determine the quality of your claimed items. If the sum of the claimed items is equal to or greater than 90, print:

**"Wow, your prey was epic! Value: {sum of claimed items}"**

Else print:

**"Poor prey... Value: {sum of claimed items}"**

### Input

* On the **first line**, you will receive the integers representing the **first magic box**, **separated** by a **single space**.
* On the **second line**, you will receive the integers representing the **second magic box, separated** by a **single space**.

### Output

* On the **first** line of output – print which box got empty in the format described above.
* On the **second** line – the quality of your prey in the format described above.

### Constraints

* All of the given numbers will be valid integers in the range **[0, 100]**.
* There won’t be a case where both magic boxes become empty at the same time.

### Examples

|  |  |  |
| --- | --- | --- |
| ****Input**** | ****Output**** | ****Comment**** |
| **10 11 8 13 5 6**  **0 4 7 3 6 23 3** | **Second magic box is empty.**  **Poor prey... Value: 42** | First we sum 10 and 3. We get 13, which is not an even number, so we take the last item from the second box and move it to last position in the first box. The current state of the boxes:  10 11 8 13 5 6 3  0 4 7 3 6 23  The next sum is 33 so we do the same again. On the third iteration the sum is 16 which is an even number, so we remove both of the boxes and we add the value to our claimed items. We keep summing items until one of the boxes becomes empty. |
| **20 40 60 80 100**  **10 20 30 40 50 60** | **First magic box is empty.**  **Wow, your prey was epic! Value: 500** |  |

## Python

**Link:** [**https://judge.softuni.bg/Contests/Practice/Index/2997#1**](https://judge.softuni.bg/Contests/Practice/Index/2997#1)

*One of the candidates-intern must solve the task - "Python". He must write a Java code that calculates wheater the python wins or loses the game.*

Here are the rules of the game:

The game starts with a **python length of 1**.

We get as input **the size** of the **screen** in which our python moves. The screen is **always a square**. After that we receive the commands wich represent the directions in wich the python should move.The python **starts** from **s**-position The commands will be: **left/right/up/down.** If the python reaches the side edge of the screen (left or right), it goes to the **opposite side of the same row**. If the python reaches the top/bottom edge of the screen it goes on the **opposite side of the same column** . The possible characters that may appear on the screeen are:

* **\***– that is a regular asterisk; it does nothing
* **e** – represents an enemy.
* **f** – this is the food
* **s** - the place where the **game starts**

Each time you eat a piece of food your **length increases by one**. Keep track of the length, because **in case you win you have to print it**. If you **step on an enemy the game is over (the python stops moving)** and you have to print the output as shown in the output section. After executing all of the commands there are 3 possible outcomes:

* you have eaten all the food and you win
* you get killed by an enemy
* there is still some food to be eaten

Print the corresponding output depending on the case.

### Input

* **Length** of the screen side – an integer number.
* **Commands to move** the python – an array of strings separated by "**,**".

### Output

* There are three types of output:
  + If all of the food is eaten print the following output: **"You win! Final python length is {length}"**
  + If there are no left commands and there is still some food to be eaten: **"You lose! There is still {left food} food to be eaten."**
  + If you step on the enemy the game is over and you print **"You lose! Killed by an enemy!"**

### Constraints

* The **input numbers** will be a 32-bit integer in the range [0 … 2 147 483 647].
* Allowed working time for your program: 0.1 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 5  up, right, right, right, up  \* \* e \* \*  \* \* \* f \*  \* f \* \* \*  s \* \* \* \*  \* \* e \* \* | You win! Final python length is 3 | After executing all of the commands the python has eaten all of the food and it is still alive. |
| 4  right, right, right, right, right, down, right  \* s \* \*  \* \* e \*  \* f \* f  \* \* \* f | You lose! Killed by an enemy! | The python moves 2 times to the right, then it goes off screen and appears on left side. Then it makes 2 more moves to the right, it goes down, steps on an enemy and the game is over. |
| 6  down, left, left, down, right, right, right, right, right  \* \* \* \* s \*  \* e f \* \* \*  f \* \* \* \* \*  \* \* \* f e \*  \* e \* \* \* \*  \* \* \* \* \* \* | You lose! There is still 1 food to be eaten. | The python survived but there is still 1 piece of food that the python could not eat. |

## Guild

**Link:** [**https://judge.softuni.bg/Contests/Practice/Index/2036#2**](https://judge.softuni.bg/Contests/Practice/Index/2036#2)

### Preparation

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

**Pay attention to name the package guild, 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 players by creating the classes described below.

### Player

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

* **name: String**
* **clazz: String**
* **rank: String – "Trial" by default**
* **description: String – "n/a" by default**

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

**"Player {name}: {clazz}**

**Rank: {rank}**

**Description: {description}"**

### Guild

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

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

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

* Method addPlayer(Player player) - **adds** an **entity** to the roster **if** **there** **is** **room** for it
* Method removePlayer(String name) - removes a player by **given name,** if such **exists**, and **returns boolean**
* Method promotePlayer(String name) - **promote** (**set his rank to "Member"**) the **first player** with the **given name.** If the player is **already** a "Member", **do nothing.**
* Method **demotePlayer(String name)- demote (set his rank to "Trial")** the first player with the **given** name. If the player is **already** a "Trial",  **do nothing.**
* Method kickPlayersByClass(String clazz) - removes all the players by the given class and returns **all removed players** from that **class as an array**
* Method count() - **returns** the **number** of players
* Method **report()** - **returns** a **String** in the following **format:**
  + **"Players in the guild: {guildName}:  
    {Player1}  
    {Player2}  
    (…)**"

### Constraints

* The **names** of the players will be **always unique**.
* You will always have a player added before receiving methods manipulating the Guild's players.

### Examples

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

|  |
| --- |
| Sample code usage |
| package guild;  public class Main {  public static void main(String[] args) {  //Initialize the repository (guild)  Guild guild = new Guild("Weekend Raiders", 20);  //Initialize entity  Player player = new Player("Mark", "Rogue");  //Print player  System.*out*.println(player);  //Player Mark: Rogue  //Rank: Trial  //Description: n/a   //Add player  guild.addPlayer(player);  System.*out*.println(guild.count()); //1  System.*out*.println(guild.removePlayer("George")); //false   Player firstPlayer = new Player("Pep", "Warrior");  Player secondPlayer = new Player("Lizzy", "Priest");  Player thirdPlayer = new Player("Mike", "Rogue");  Player fourthPlayer = new Player("Marlin", "Mage");   //Add description to player  secondPlayer.setDescription("Best healer EU");   //Add players  guild.addPlayer(firstPlayer);  guild.addPlayer(secondPlayer);  guild.addPlayer(thirdPlayer);  guild.addPlayer(fourthPlayer);   //Promote player  guild.promotePlayer("Lizzy");   //Remove Player  System.*out*.println(guild.removePlayer("Pep")); //true   Player[] kickedPlayers = guild.kickPlayersByClass("Rogue");  for (Player kickedPlayer : kickedPlayers) {  System.*out*.print(kickedPlayer.getName() + " ");  }  //Mark Mike   System.*out*.println(guild.report());  //Players in the guild: Weekend Raiders:  //Player Lizzy: Priest  //Rank: Member  //Description: Best healer EU  //Player Marlin: Mage  //Rank: Trial  //Description: n/a  } } |