Lab: String Processing

This document defines the exercises for "Java Advanced" course @ Software University. Please submit your solutions (source code) of all below described problems in Judge.

1. Student's Results

Write a program that reads one line, containing a student's name and his results in format **{name} - {firstResult}, {secondResult}, {thirdResult}**

Print a table on the console. Each row must contain:

- JAdv - first result, aligned right, rounded to a **precision of 2**
- OOP - second result, aligned right, rounded to a **precision of 2**
- AdvOOP - third result, aligned right, rounded to a **precision of 2**
- Average – average result, rounded to a **precision of 4**
- Columns have a **width** of **7 characters** and must be separated with **"|"**
- Don't forget the heading row

Examples

Input	Output
`Gosho - 3.33333, 4.4444, 5.555`	`Name	JAdv|JavaOOP| AdvOOP|Average|` `Gosho	3,33| 4,44| 5,56| 4,4442|`
`Mara - 5, 4, 3`	`Name	JAdv|JavaOOP| AdvOOP|Average|` `Mara	5,00| 4,00| 3,00| 4,0000|`

Hints

It is up to you what type of data structures you will use to solve this problem

- The first row is **easy**, but long.

```
System.out.println(String.format
("%1$-10s|%2$7s|%3$7s|%4$7s|%5$7s|",
"Name", "JAdv", "JavaOOP", "AdvOOP", "Average"));
```

- Data rows are just a little bit more complicated:

```
System.out.println(String.format
("%1$-10s|%2$7.2f|%3$7.2f|%4$7.2f|%5$7.4f|",
student, resuls.get(0), resuls.get(1),
resuls.get(2), average));
```

2. Parse URL

Write a program that parses an URL address given in the format: **[protocol]://[server]/[resource]** and extracts from it the **[protocol]**, **[server]** and **[resource]** elements.

If the URL is not in a correct format, print "`Invalid URL`" on the console.

Examples

Input	Output

`https://softuni.bg/courses/java-advanced`	`Protocol = https`
	`Server = softuni.bg`
	`Resources = courses/java-advance`

`https://www.google.bg/search?q=google&oq=goo&aqs=chrome.0.0j69i60l2://j0j69i57j69i65.2112j0j7&sourceid=chrome&ie=UTF-8`	`Invalid URL`

Hints

- **"://"** is used to show where a protocol name ends. If you have it more than once, the URL will be **invalid**.
- Server name ends with **"/"**, but it is **not** part of **resourses**.
- Resources use the same symbol **"/"** to show that we go deeper in the **folders tree**, so be careful.

Think about the proper operations over the input:

- **`.split()`**
- **`.substring()`**
- **`.indexOf()`**

3. Parse Tags

You are given a text. Write a program that changes the text in all regions surrounded by the tags <**upcase**> and </**upcase**> to upper-case. The tags won't be nested.

Examples

Input	Output
`We are living in a <upcase>yellow submarine</upcase>. We don't have <upcase>anything</upcase> else.`	We are living in a YELLOW SUBMARINE. We don't have ANYTHING else.
`<upcase>StringBuilder</upcase> is <upcase>awesome</upcase>`	`STRINGBUILDER is AWESOME`

Hints

- Be careful when **replacing tags** with **empty** strings.
- Consider that, after replacing a tag, the **indexes** in the string are **not** the **same**.

4. Series of Letters

Read a string from the console and **replace** all series of **consecutive identical letters** with a **single one**.

Examples

Input	Output

aabb	ab
abc	abc
aaaaabbbbbcdddeeeedssaa	abcdedsa

Hints

- Use a quantifier for one or more repetitions **+**, grouping **()** and a backreference construct

5. Vowel Count

Find the **count** of **all vowels** in a given **text** using a regex.

The vowels that you should be looking for are **upper** and **lower** case: **a**, **e**, **i**, **o**, **u** and **y**.

Examples

Input	Output
`Abraham Lincoln`	`Vowels: 5`
`In 1519 Leonardo da Vinci died at the age of 67.`	`Vowels: 15`
`n vwls.`	`Vowels: 0`

Hints

- Read the input using
- Compile the pattern and create a **Matcher** object:

```
Pattern pattern = Pattern.compile("[AEIOUYaeiouy]");
Matcher matcher = pattern.matcher(text);
```

- Count the occurrences:

```
int count = 0;
while (matcher.find()) {
    count++;
}
```

- Finally, print the result:

6. Extract Tags

Read lines until you get the **"END"** command. Extract all **tags** from the given HTML using **RegEx**. If there are **no tags**, don't print anything.

Examples

Input	Output

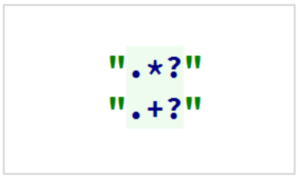
`<!DOCTYPE html>`
`<html lang="en">`
`<head>`
`    <meta charset="UTF-8">`
`    <title>Title</title>`
	`<!DOCTYPE html>`
	`<html lang="en">`
	`<head>`
	`<meta charset="UTF-8">`
	`<title>`

`</head>`
`</html>`
END
	`</title>`
	`</head>`
	`</html>`

`No tags.`
END
	(no output)

Hints

- Use the special character dot "**.**" and one of the regex quantifiers **made lazy**:



```
".*?"
".+?"
```

- Design your own regex to get a complete solution

7. Valid Usernames

Scan through the lines for **valid usernames**.

A valid username:

- Has **length** between 3 and 16 characters
- **Contains** only letters, numbers, hyphens and underscores
- Has **no redundant symbols** before, after or in between

Read the lines until you get the **"END"** command. If there are **no valid usernames**, don’t print anything.

Examples

Input	Output

`sh`
`too_long_username`
`!lleg@l ch@rs`
`jeff_butt`
END
	`invalid`
	`invalid`
	`invalid`
	`valid`

END	*(no output)*

Hints

- Use character classes **[ ]**, quantifiers **{ }** and anchors **^** and **$**