# Exercise: Java Basics

This exercise is part of the [“Databases Frameworks” course @ SoftUni](https://softuni.bg/trainings/1635/databases-frameworks-hibernate-and-spring-data-june-2017).

# Part I: Data Types and Methods

## Variable in Hexadecimal Format

Write a program that reads a number in **hexadecimal format** convert it to **decimal format** and prints it.

|  |  |
| --- | --- |
| **Input** | **Output** |
| FE | 254 |
| 37 | 55 |
| 10 | 16 |

### Hints

* Use [**Integer.parseInt(string, base)**](http://www.tutorialspoint.com/java/number_parseint.htm).

## Boolean Variable

Write a program that reads a **string**, converts it to **Boolean** variable and **prints** “**Yes**”if the variable is true and “**No**” if the variable is false.

|  |  |
| --- | --- |
| **Input** | **Output** |
| True | Yes |
| False | No |

### Hints

* You can parse Booleans.

## Reverse Characters

Write a program to ask the user for **3 letters** and print them in **reversed order**.

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| A  B  C | CBA |  | x  Y  z | zYx |  | G  g  n | ngG |

## Vowel or Digit

Create a program to check if given symbol is **digit**, **vowel** or any **other symbol**.

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| a | vowel |  | 9 | digit |  | g | other |

## Integer to Hex and Binary

Create a program to convert a **decimal number** to **hexadecimal** and **binary** number and print it.

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 10 | A  1010 |  | 420 | 1A4  110100100 |  | 256 | 100  100000000 |

### Hints

* There are methods, that convert Integer to Hex and Binary.

# Part II: Arrays

## Compare Char Arrays

Compare two char arrays lexicographically (letter by letter).

Print them in alphabetical order, each on separate line.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| a b c  d e f | abc  def |
| p e t e r  a n n i e | annie  peter |
| a n n i e  a n | an  annie |
| a b  a b | ab  ab |

### Hints

* Compare the first letter of arr1[] and arr2[], if equal, compare the next letter, etc.
* If all letters are equal, the smaller array is the **shorter**.
* If all letters are equal and the array lengths are the same, the arrays are **equal**.

## Max Sequence of Equal Elements

Write a program that finds the **longest sequence of equal elements** in an array of integers. If several longest sequences exist, print the leftmost one.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 1 1 2 3 3 **2 2 2** 1 | 2 2 2 |
| **1 1 1** 2 3 1 3 3 | 1 1 1 |
| **4 4 4 4** | 4 4 4 4 |
| 0 **1 1** 5 2 2 6 3 3 | 1 1 |

### Hints

* Start with the sequence that consists of the first element: start=0, len=1.
* Scan the elements from left to right, starting at the second element: pos=1…n-1.
  + At each step compare the current element with the element on the left.
    - Same value 🡪 you have found a sequence longer by one 🡪 len++.
    - Different value 🡪 start a new sequence from the current element: start=pos, len=1.
  + After each step remember the sequence it is found to be longest at the moment: bestStart=start, bestLen=len.
* Finally, print the longest sequence by using bestStart and bestLen.

## Max Sequence of Increasing Elements

Write a program that finds the **longest increasing subsequence** in an array of integers. The longest increasing subsequence is a **portion of the array** (subsequence) that is strongly **increasing** and has the **longest possible length**. If several such subsequences exist, find the left most of them.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3 **2 3 4** 2 2 4 | 2 3 4 |
| 4 5 **1 2 3 4 5** | 1 2 3 4 5 |
| **3 4 5 6** | 3 4 5 6 |
| **0 1** 1 2 2 3 3 | 0 1 |

### Hints

* Use the same algorithm like in the previous problem (Max Sequence of Equal Elements).

## Most Frequent Number

Write a program that finds the **most frequent number** in a given sequence of numbers.

* Numbers will be in the range [0…65535].
* In case of multiple numbers with the same maximal frequency, print the left most of them.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Output** |
| **4** 1 1 **4** 2 3 **4 4** 1 2 **4** 9 3 | 4 | The number **4** is the most frequent (occurs 5 times) |
| **2 2 2 2** 1 **2 2 2** | 2 | The number **2** is the most frequent (occurs 7 times) |
| **7 7 7** 0 2 2 2 0 10 10 10 | 7 | The numbers **2**, **7** and **10** have the same maximal frequence (each occurs 3 times). The leftmost of them is **7**. |

## Index of Letters

Write a program that creates an array containing all letters from the alphabet (**a**-**z**). Read a lowercase word from the console and print the **index of each of its letters in the letters array**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| abcz | a -> 0  b -> 1  c -> 2  z -> 25 |
| softuni | s -> 18  o -> 14  f -> 5  t -> 19  u -> 20  n -> 13  i -> 8 |

## Equal Sums

Write a program that determines if there **exists an element in the array** such that the **sum of the elements on its left** is **equal** to the **sum of the elements on its right**. If there are **no elements to the left / right**, their **sum is considered to be 0**. Print the **index** that satisfies the required condition or **“no”** if there is no such index.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 1 2 3 3 | 2 | At a[2] -> left sum = 3, right sum = 3  a[0] + a[1] = a[3] |
| 1 2 | no | At a[0] -> left sum = 0, right sum = 2  At a[1] -> left sum = 1, right sum = 0  No such index exists |
| 1 | 0 | At a[0] -> left sum = 0, right sum = 0 |
| 1 2 3 | no | No such index exists |
| 10 5 5 99 3 4 2 5 1 1 4 | 3 | At a[3] -> left sum = 20, right sum = 20  a[0] + a[1] + a[2] = a[4] + a[5] + a[6] + a[7] + a[8] + a[9] + a[10] |

# Part IV: Strings, Maps and Stream API

## Reverse String

Write a program that reads a string from the console, **reverses** its letters and prints the result back at the console.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| sample | elpmas |
| 24tvcoi92 | 29iocvt42 |

### Hints

* **Variant I**: convert the string to **char array**, **reverse** it, then convert it to **string** again.
* **Variant II**: print the letters of the string in back direction (from the last to the first) in a for-loop.

## Fit String in 20 Chars

Write a program that **reads** from the console a string and **fits the string in 20 characters** as follows:

* If the string has **less than 20 characters**, append some ‘\*’ until it gets length of exactly 20 characters.
* If the string length is **more than 20 characters**, discard all characters after the first 20.

**Print** the result string at the console.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Welcome to SoftUni! | Welcome to SoftUni!\* |
| A "regular expression" (abbreviated regex or regexp) is a sequence of characters that forms a search pattern. | A "regular expressio |
| C# | C#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

### Hints

* If string length < 20, use PadRight(20, '\*').
* If string length > 20, use Substring(0, 20).

## Censor Email Address

You have some text that contains your email address. You are sick of spammers, so you want to **hide** it. You decide to **censor** your email: to **replace all characters** in it with asterisks ('\*') **except the domain**.

Assume your email address will always be in format **[username]@[domain]**. You need to replace the username with asterisks of equal number of letters and keep the domain unchanged.

### Input

* The first line holds your **email** address.
* The second line holds a **text** where the email should be censored.

### Examples

|  |
| --- |
| **Input** |
| pesho.peshev@email.bg  My name is Pesho Peshev. I am from Sofia, my email is: pesho.peshev@email.bg (not pesho.peshev@email.com). Test: pesho.meshev@email.bg, pesho.peshev@email.bg |
| **Output** |
| My name is Pesho Peshev. I am from Sofia, my email is: \*\*\*\*\*\*\*\*\*\*\*\*@email.bg (not pesho.peshev@email.com). Test: pesho.meshev@email.bg, \*\*\*\*\*\*\*\*\*\*\*\*@email.bg |

### Hints

In order to accomplish the task, you may find these steps useful:

* **Split** the email into two parts – **username** and **domain**.
* Create the **replacement** string by duplicating the '\*' character username.length and appending '@' and the **domain**.
* **Replace** all occurrences of your **email** with the **replacement string**.

## URL Parser

Write a program that **parses an URL** given in the following format:

[protocol]://[server]/[resource]

The parsing extracts its parts: protocol, server and resource.

* The **[server]** part is mandatory.
* The **[protocol]** and **[resource]** parts are optional.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| http://www.abc.com/video | [protocol] = "http"  [server] = "www.abc.com"  [resource] = "video" |
| https://www.softuni.bg/Resources/Materials | [protocol] = "https"  [server] = "www.softuni.bg"  [resource] = "Resources/Materials" |
| ftp://www.su.us/TestResource | [protocol] = "ftp"  [server] = "www.su.us"  [resource] = "TestResource" |
| https://softuni.bg | [protocol] = "https"  [server] = "softuni.bg"  [resource] = "" |
| www.nakov.com | [protocol] = ""  [server] = "www.nakov.com"  [resource] = "" |

### Hints

* Find the leftmost occurrence of “://” in the input URL.
  + If **found**, the left side holds the **protocol**, the right side holds the **server + resource**.
  + If **not found**, the protocol is missing, the input string holds **server + resource** only.
* After the “protocol” part is removed from the input URL, find the leftmost occurrence of “/”.
  + If **found**, the left side holds the **server**, the right side holds the **resource**.
  + If **not found**, the resource is missing, the whole string holds the **server**.

## Change to Uppercase

We are given a text. Write a program that **modifies the casing of letters to uppercase** at all places **in the text surrounded** **by** <upcase> **and** </upcase> **tags**. Tags cannot be nested.

### Example

|  |
| --- |
| **Input** |
| Welcome to the **<upcase>Software University</upcase>**. Learn **<upcase>computer programming</upcase>** and start a **<upcase>job</upcase>** in a software company. |
| **Output** |
| Welcome to the **SOFTWARE UNIVERSITY**. Learn **COMPUTER PROGRAMMING** and start a **JOB** in a software company. |

### Hints

* You may find the position of the first <upcase> and the first </upcase>, delete the text between and insert the uppercase version of the text without the tags at the position of <upcase>.
* Repeat the above until no more <upcase> and </upcase> tags are found in the text.

## Phonebook

Write a program that receives some info from the console about **people** and their **phone numbers**. Each **entry** should have just **one name** and **one number** (both of them strings).

On each line you will receive some of the following commands:

* **A {name} {phone}** – adds entry to the phonebook. In case of trying to add a name that is already in the phonebook you should change the existing phone number with the new one provided.
* **S {name}** – searches for a contact by given name and prints it in format "**{name} -> {number}**". In case the contact isn't found, print "**Contact {name} does not exist.**".
* **END** – stop receiving more commands.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| A Nakov 0888080808  S Mariika  S Nakov  END | Contact Mariika does not exist.  Nakov -> 0888080808 |
| A Nakov +359888001122  A RoYaL(Ivan) 666  A Gero 5559393  A Simo 02/987665544  S Simo  S simo  S RoYaL  S RoYaL(Ivan)  END | Simo -> 02/987665544  Contact simo does not exist.  Contact RoYaL does not exist.  RoYaL(Ivan) -> 666 |
| A Misho +359883123  A Misho 02/3123  S Misho  END | Misho -> 02/3123 |

### Hints

* **Parse the commands** by splitting by space. Execute the commands until “**END**” is reached.
* Store the **phonebook entries** in LinkedHashMap<String, String> with key **{name}** and value **{phone number}**.

## Phonebook Upgrade

**Add functionality to the phonebook** from the previous task to **print all contacts ordered lexicographically** when receive the command “ListAll”.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| A Nakov +359888001122  A RoYaL(Ivan) 666  A Gero 5559393  A Simo 02/987665544  ListAll  END | Gero -> 5559393  Nakov -> +359888001122  RoYaL(Ivan) -> 666  Simo -> 02/987665544 |

### Hints

* **Variant I (slower):** Sort all entries in the dictionary by key and print them.
* **Variant II (faster):** Keep the entries in more appropriate data structure that will keep them in sorted order for better performance.