Lab: Lists

Problems for exercises and homework for the ["Programming Fundamentals" course @ SoftUni](https://softuni.bg)

You can check your solutions in [Judge.](https://judge.softuni.bg)

1. Sum Adjacent Equal Numbers

Write a program to **sum all adjacent equal numbers** in a list of decimal numbers, starting from **left to right**.

- After two numbers are summed, the obtained result could be equal to some of its neighbors and should be summed as well (see the examples below).
- Always sum the **leftmost** two equal neighbors (if several couples of equal neighbors are available).

Examples

Input	Output	Explanation
`3 3 6 1`	`12 1`	**3 3** 6 1 → **6 6** 1 → 12 1
`8 2 2 4 8 16`	`16 8 16`	8 **2 2** 4 8 16 → 8 **4 4** 8 16 → **8 8** 8 16 → 16 8 16
`5 4 2 1 1 4`	`5 8 4`	5 4 2 **1 1** 4 → 5 4 **2 2** 4 → 5 **4 4** 4 → 5 8 4
`0.1 0.1 5 -5`	`0.2 5 -5`	**0.1 0.1** 5 -5 → 0.2 5 -5

Solution

Read a list of numbers.

```java
Scanner sc = new Scanner(System.in);

List<Double> numbers =
        Arrays.stream(sc.nextLine().split( regex: " "))
                .map(Double::parseDouble)
                .collect(Collectors.toList());
```

Iterate through the elements. Check if the number at the current index is equal to the next number. If it is, aggregates the numbers and reset the loop otherwise, don't do anything.

```java
if(numbers.get(i).equals(numbers.get(i + 1))){
    numbers.set(i, (numbers.get(i) + numbers.get(i + 1)));
    numbers.remove( index: i + 1);
    i = -1;
}
```

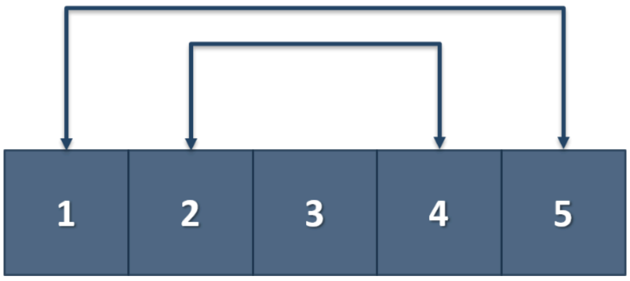
Finally, you have to print the numbers joined by space.

```java
  String output = joinElementsByDelimiter(numbers,  delimiter: " ");
  System.out.println(output);
```

```
static String joinElementsByDelimiter(List<Double> items,
                                      String delimiter) {
    String output = "";
    for (Double item : items)
        output += (new DecimalFormat( pattern: "0.#").format(item)
                + delimiter);
    return output;
}
```

2. Gauss' Trick

Write a program that **sum** all **numbers in a list** in the following order:
first **+** last, first + 1 **+** last - 1, first + 2 **+** last **-** 2**,** ... first + n, last - n.



Example

Input	Output
1 2 3 4 5	6 6 3
1 2 3 4	5 5

3. Merging Lists

You are going to receive two lists with numbers. Create a result list that contains the numbers from both of the lists. The first element should be from the first list, the second from the second list, and so on. If the length of the two lists is not equal, just add the remaining elements at the end of the list.

Examples

Input	Output
3 5 2 43 12 3 54 10 23 76 5 34 2 4 12	3 76 5 5 2 34 43 2 12 4 3 12 54 10 23
76 5 34 2 4 12 3 5 2 43 12 3 54 10 23	76 3 5 5 34 2 2 43 4 12 12 3 54 10 23

Hint

- Read the two lists.
- Create a result list.
- Start looping through them until you reach the end of the smallest one.
- Finally, add the remaining elements (if any) to the end of the list.

4. List Manipulation Basics

Write a program that reads a list of integers. Then until you receive **"end"**, you will be given different **commands:**

- **`Add {number}`** – add a number to the end of the list
- **`Remove {number}`** – remove a number from the list
- **`RemoveAt {index}`** – remove a number at a given index
- **`Insert {number} {index}`** – insert a number at a given index

Note: All the indices will be valid!

When you receive the **"end"** command, print the **final state** of the list (**separated by spaces**).

Examples

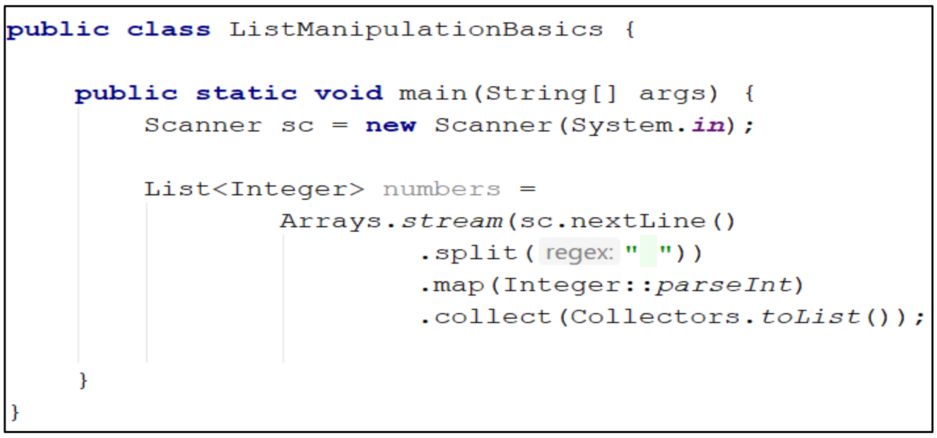
Input	Output

`4 19 2 53 6 43`
`Add 3`
`Remove 2`
`RemoveAt 1`
`Insert 8 3`
`end`
	`4 53 6 8 43 3`

`12 34 100 1 45 2 8`
`Add 30`
`Remove 12`
`Remove 3`
`RemoveAt 3`
`Insert 2 3`
`end`
	`34 100 1 2 2 8 30`

Solution

First, let us read the list from the console.



```java
public class ListManipulationBasics {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        List<Integer> numbers =
                Arrays.stream(sc.nextLine()
                        .split( regex: " "))
                        .map(Integer::parseInt)
                        .collect(Collectors.toList());

    }
}
```

Next, we make the while loop for the commands and make a switch statement for the commands.

```
while (true){
    String line = sc.nextLine();

    if(line.equals("end")){
        break;
    }

    String[] tokens = line.split( regex: " ");
}
```

We break if the line is "**end**", otherwise, we split it into tokens and process the command.

```
String[] tokens = line.split( regex: " ");

switch (tokens[0]){
    case "Add":
        break;
    case "Remove":
        break;
    case "RemoveAt":
        break;
    case "Insert":
        break;
}
```

Now let's implement each command.

```
case "Add":
    int numberToAdd = Integer.parseInt(tokens[1]);
    numbers.add(numberToAdd);
    break;
case "Remove":
    int numberToRemove = Integer.parseInt(tokens[1]);
    numbers.remove(numberToRemove);
    break;
case "RemoveAt":
    int indexToRemove = Integer.parseInt(tokens[1]);
    numbers.remove(indexToRemove);
    break;
case "Insert":
    int numberToInsert = Integer.parseInt(tokens[1]);
    int indexToInsert = Integer.parseInt(tokens[2]);
    numbers.add(indexToInsert, numberToInsert);
    break;
```

For all commands **except** the **"`Insert`", tokens[1]** is the **number/index**. For the **"`Insert`"** command, we receive a **number and an index** (**tokens[1], tokens[2]**).

Finally, we **print** the numbers, joined by **a single space:**

```
System.out.println(numbers.toString()
        .replaceAll( regex: "[\\[\\],]",  replacement: ""));
```

5. List Manipulation Advanced

Now we will implement more complicated list commands. Again, read a list, and until you receive **"end"** read commands:

- **`Contains {number}`** – check if the list contains the number. If **yes,** print **"`Yes`"**, **otherwise,** print **"`No such number`"**
- **`Print even`** – print **all the numbers** that are **even separated by a space**
- **`Print odd`** – print **all the numbers** that are **oddly separated by a space**
- **`Get sum`** – print the **sum of all the numbers**
- **`Filter {condition} {number}`** – print all the numbers that **fulfill that condition**. The condition will be either **'<'**, **'>'**, **">="**, **"<="**

Examples

Input	Output

`2 13 43 876 342 23 543`
`Contains 100`
`Contains 543`
`Print even`
`Print odd`
`Get sum`
`Filter >= 43`
`Filter < 100`
`end`
	`No such number`
	`Yes`
	`2 876 342`
	`13 43 23 543`
	`1842`
	`43 876 342 543`
	`2 13 43 23`

`12 3 123 546 222 45 7`
`Contains 3`
`Contains 121`
`Print even`
`Print odd`
`Get sum`
`Filter >= 100`
`Filter < 45`
`end`
	`Yes`
	`No such number`
	`12 546 222`
	`3 123 45 7`
	`958`
	`123 546 222`
	`12 3 7`

6. List of Products

Read a number **n** and **n lines of products**. Print a **numbered list** of all the products **ordered by name**.

Examples

Input	Output

`4`
`Potatoes`
`Tomatoes`
`Onions`
`Apples`
	`1.Apples`
	`2.Onions`
	`3.Potatoes`
	`4.Tomatoes`

3 Orange Grape Strawberry	1.Grape 2.Orange 3.Strawberry

Solution

First, we need to read the number **n** from the console.

```
import java.util.Scanner;

public class ListOfProducts {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = Integer.parseInt(sc.nextLine());
    }
}
```

Then we need to create our **list of strings** because the **products are strings.**

```
public class ListOfProducts {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = Integer.parseInt(sc.nextLine());
        List<String> products = new ArrayList<>();
    }
}
```

Then we need to iterate **n times** and **read products**.

```
for (int i = 0; i < n; i++) {
    String currentProduct = sc.nextLine();
}
```

The next step is to add the current product to the list.

```
for (int i = 0; i < n; i++) {
    String currentProduct = sc.nextLine();
    products.add(currentProduct);
}
```

After we finish reading the products, we **sort our list alphabetically.**

```
Collections.sort(products);
```

The **sort method** sorts the list in ascending order.

Finally, we have to **print our sorted** list. To do that we **loop through the list**.

```
for (int i = 0; i < products.size(); i++) {
    System.out.printf("%d.%s%n", i + 1, products.get(i));
}
```

We use **i + 1** because we want to **start counting from 1**.

7. Remove Negatives and Reverse

Read a **list of integers**, **remove all negative numbers** from it and print the remaining elements in **reversed order**. In case of no elements left in the list, print "**empty**".

Examples

Input	Output
`10 -5 7 9 -33 50`	`50 9 7 10`
`7 -2 -10 1`	`1 7`
`-1 -2 -3`	`empty`

Solution

Read a list of integers.

```
Scanner sc = new Scanner(System.in);

List<Integer> numbers =
        Arrays.stream(sc.nextLine().split( regex: " "))
                .map(Integer::parseInt)
                .collect(Collectors.toList());
```

Remove all negative numbers and reverse the collection.

```
numbers.removeIf(n -> n < 0);
Collections.reverse(numbers);
```

If the list is empty, print "**empty**", otherwise, print all numbers joined by space.

```
if (numbers.isEmpty()) {
    System.out.println("empty");
} else {
    System.out.println(numbers.toString().replaceAll( regex: "[\\[\\],]",  replacement: ""));
}
```