Some fact About Java Programming

① > sc.nextLine()
↳ No matter String. String input or
- integer
- double
- array
- object

② After taking the input ↳ Anything
↳ Convert into input array

How,

```
String str = sc.nextLine();
String[] input = str.split("\\s+");
```

Don't change it is regular expression.

Some code justify the point 1st

```
import java.util.*;

class Main {
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        int t=5;

        while(t-->0){
            System.out.print("Input given ");
        String str=sc.nextLine();
        System.out.println("Output given "+str);
        }
    }
}
```

Output

```
Input given 5
Output given 5
Input given [1,2,3]
Output given [1,2,3]
Input given {gfds}fgsda   dshdf
Output given {gfds}fgsda   dshdf
Input given "shailendr"
Output given "shailendr"
Input given 85.5455gsdfsd
Output given 85.5455gsdfsd

=== Code Execution Successful ===
```

① (integer)
② (arrays)
③ (combination of two)
④ (string)
⑤ (random thing)

2nd points :

```java
import java.util.*;

class Main {
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        int t=5;

        while(t-->0){
    System.out.print("Input given ");
    String str=sc.nextLine();
    System.out.println("Output given "+str);
    String[] input =str.split("\\s+");
    System.out.println("Arrays corresponding to it "+Arrays.toString(input));

        }

    }
}
```

```
Input given "shailendra Mishra From somewhere"
Output given "shailendra Mishra From somewhere"
Arrays corresponding to it ["shailendra, Mishra, From, somewhere"]
Input given 1        2       3
Output given 1        2       3
Arrays corresponding to it [1, 2, 3]
Input given A b cd     t
Output given A b cd     t
Arrays corresponding to it [A, b, cd, t]
Input given 5f t3  6s    " , 5
Output given 5f t3  6s    " , 5
Arrays corresponding to it [5f, t3, 6s, ", ,, 5]
Input given 2# %    h
Output given 2# %    h
Arrays corresponding to it [2#, %, h]

=== Code Execution Successful ===
```

- \\s - a whitespace character (space, tab, newline, etc.)

→ Concept related input skipping :-

A common issue in Java when using ***Scanner*** to read different types of input sequentially. This behavior occurs due to how Scanner methods ***handle newline characters***.

```
7         Scanner scanner = new Scanner(System.in);
8
9   System.out.print("Enter an integer: ");
10  int num = scanner.nextInt();  // Reads integer but leaves newline
11
12  System.out.print("Enter a string: ");
13  String text = scanner.nextLine();  // Reads the leftover newline
14
15  // The string input seems to be skipped!
16  System.out.print("Enter a double: ");
17  double value = scanner.nextDouble();
18
19
20  System.out.println("Integer"+num);
21  System.out.println("text"+text);
22  System.out.println("double"+ value);
```

Output

```
Enter an integer: 5
Enter a string: Enter a double: 8
Integer5
text
double8.0

=== Code Execution Successful ===
```

Issue in the code ... you don't get the chances to inset the string

Problem if insert the next string:

```
Enter an integer: 5
Enter a string: Enter a double: "shailenda"
ERROR!
Exception in thread "main" java.util.InputMismatchException
    at java.base/java.util.Scanner.throwFor(Scanner.java:947)
    at java.base/java.util.Scanner.next(Scanner.java:1602)
    at java.base/java.util.Scanner.nextDouble(Scanner.java:2573)
    at Main.main(Main.java:17)

=== Code Exited With Errors ===
```

Input Mismatch Exception : this problem

Why This Happens

1. nextInt() reads the integer but doesn't consume the newline (\n) from pressing Enter
2. The subsequent nextLine() immediately consumes this leftover newline
3. This makes it appear as if the string input was skipped

Solutions

Solution 1: Consume the leftover newline

Java

```
int num = scanner.nextInt();
scanner.nextLine();  // Consume the leftover newline
String text = scanner.nextLine();
```

Solution 2: Use consistent input methods

java

```
// Read everything as String and parse
int num = Integer.parseInt(scanner.nextLine());
String text = scanner.nextLine();
double value = Double.parseDouble(scanner.nextLine());
```

Solution 3: Clear the buffer (more robust)

java

```
int num = scanner.nextInt();
// Clear the buffer
while (scanner.hasNextLine() && scanner.nextLine().isEmpty()) {}
String text = scanner.nextLine();
```

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

    System.out.print("Enter an integer: ");
    int num = scanner.nextInt();
    scanner.nextLine();  // Consume newline

    System.out.print("Enter a string: ");
    String text = scanner.nextLine();

    System.out.print("Enter a double: ");
    double value = scanner.nextDouble();

    System.out.println("\nResults:");
    System.out.println("Integer: " + num);
    System.out.println("String: " + text);
    System.out.println("Double: " + value);

    scanner.close();
}
```

Methods That **Don't** Consume Newlines

These methods leave the newline (`\n`) character in the buffer:

- `nextInt()`
- `nextDouble()`
- `nextFloat()`
- `nextLong()`
- `nextShort()`
- `nextByte()`
- `nextBoolean()`
- `next()` (reads single token)

Methods That **Do** Consume Newlines

These methods read and consume the newline:

- `nextLine()`
- Any method when you press Enter after input

Why This Matters

When you mix these methods, you get unexpected behavior:

```java
Scanner sc = new Scanner(System.in);
int num = sc.nextInt();    // Reads number, leaves \n
String s = sc.nextLine();  // Reads the leftover \n immediately
// s will be empty!
```

Complete Solution Approach

Option 1: Always use `nextLine()` and parse

```java
int num = Integer.parseInt(sc.nextLine());
double d = Double.parseDouble(sc.nextLine());
String s = sc.nextLine();
```

Option 2: Consume newlines manually

```java
int num = sc.nextInt();
sc.nextLine();  // Consume the leftover newline
String s = sc.nextLine();
```

Option 3: Pattern for mixed input

```java
System.out.print("Enter age: ");
int age = sc.nextInt();
sc.nextLine();  // Clear buffer

System.out.print("Enter name: ");
String name = sc.nextLine();

System.out.print("Enter price: ");
double price = sc.nextDouble();
sc.nextLine();  // Clear buffer
```

Special Cases

1. **Multiple numbers then string**:

```java
int x = sc.nextInt();
int y = sc.nextInt();
sc.nextLine();  // Only need one newline consumption
String s = sc.nextLine();
```

2. **Alternating numbers and strings**:

```java
while(...) {
    int num = sc.nextInt();
    sc.nextLine();  // Consume newline after number

    String text = sc.nextLine();
    // No need for extra newline consumption
}
```

This behavior is consistent across all numeric `nextXxx()` methods in Java's Scanner class. The key is to remember that any non-line method leaves the newline in the buffer, and you need to handle it explicitly when switching to line-based input.