

Ways to read input from console in Java

Difficulty Level : Easy Last Updated : 30 Apr, 2021

In Java, there are four different ways for reading input from the user in the command line environment(console).

1.Using Buffered Reader Class

This is the Java classical method to take input, Introduced in JDK1.0. This method is used by wrapping the System.in (standard input stream) in an InputStreamReader which is wrapped in a BufferedReader, we can read input from the user in the command line.

- The input is buffered for efficient reading.
- The wrapping code is hard to remember.

Implementation:

```
// Java program to demonstrate BufferedReader
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Test {
    public static void main(String[] args)
        throws IOException
    {
        // Enter data using BufferReader
        BufferedReader reader = new BufferedReader(
            new InputStreamReader(System.in));

        // Reading data using readLine
        String name = reader.readLine();

        // Printing the read line
        System.out.println(name);
    }
}
```

Input:

Geek

Output:

Geek

Note:

To read other types, we use functions like `Integer.parseInt()`, `Double.parseDouble()`. To read multiple values, we use `split()`.

2. Using Scanner Class

This is probably the most preferred method to take input. The main purpose of the Scanner class is to parse primitive types and strings using regular expressions, however, it is also can be used to read input from the user in the command line.

- Convenient methods for parsing primitives (`nextInt()`, `nextFloat()`, ...) from the tokenized input.
- Regular expressions can be used to find tokens.
- The reading methods are not synchronized

To see more differences, please see [this](#) article.

```
// Java program to demonstrate working of Scanner in Java
import java.util.Scanner;

class GetInputFromUser {
    public static void main(String args[])
    {
        // Using Scanner for Getting Input from User
        Scanner in = new Scanner(System.in);

        String s = in.nextLine();
        System.out.println("You entered string " + s);

        int a = in.nextInt();
        System.out.println("You entered integer " + a);

        float b = in.nextFloat();
        System.out.println("You entered float " + b);

        // closing scanner
        in.close();
    }
}
```

Input:

GeeksforGeeks

12

3.4

Output:

You entered string GeeksforGeeks

You entered integer 12

You entered float 3.4

3. Using Console Class

It has been becoming a preferred way for reading user's input from the command line. In addition, it can be used for reading password-like input without echoing the characters entered by the user; the format string syntax can also be used (like `System.out.printf()`).

Advantages:

- Reading password without echoing the entered characters.
 - Reading methods are synchronized.
 - Format string syntax can be used.
 - Does not work in non-interactive environment (such as in an IDE).
-

```
// Java program to demonstrate working of System.console()
// Note that this program does not work on IDEs as
// System.console() may require console
public class Sample {
    public static void main(String[] args)
    {
        // Using Console to input data from user
        String name = System.console().readLine();

        System.out.println("You entered string " + name);
    }
}
```

Input:

Output:

You entered string GeeksforGeeks

4. Using Command line argument

Most used user input for competitive coding. The command-line arguments are stored in the String format. The parseInt method of the Integer class converts string argument into Integer. Similarly, for float and others during execution. The usage of args[] comes into existence in this input form. The passing of information takes place during the program run. The command line is given to args[]. These programs have to be run on cmd.

Code:

```
// Program to check for command line arguments
class Hello {
    public static void main(String[] args)
    {
        // check if length of args array is
        // greater than 0
        if (args.length > 0) {
            System.out.println(
                "The command line arguments are:");

            // iterating the args array and printing
            // the command line arguments
            for (String val : args)
                System.out.println(val);
        }
        else
            System.out.println("No command line "
                               + "arguments found.");
    }
}
```

Command Line Arguments:

```
javac GFG1.java  
java Main Hello World
```

Output:

```
The command line arguments are:  
Hello  
World
```

Please refer [this](#) for more faster ways of reading input.

This article is contributed by **D Raj Ranu**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.