

# Java String

## Introduction

The string is one of the most used variable types in Java. Strings are frequently used in Java to represent text-based data. They are a collection of letters and numbers encased in double quotations.

You can use the **Scanner** class, which is a component of the **java.util** package to accept string input from the user. Utilizing the **Scanner** class is the most commonly used method, but you can also use the **BufferedReader** class and other methods.

## Overview

This tutorial will explore the techniques, best practices, and commonly asked questions related to accepting **string input in Java**. By mastering this fundamental skill, you'll be well-equipped to create dynamic and interactive Java programs that effectively handle textual data and user interactions.

## How to Take String Input in Java

To take string input in Java, we can use the **java.util.Scanner** class that provides methods to read input from various sources, including the standard input (keyboard) and files. We can also use the **BufferedReader** class or Command Line Arguments.

Let us use the **Scanner** class to explain the general process and syntax to take string input in Java.

1. Import the necessary package (**java.util.Scanner**):

```
import java.util.Scanner; // We are using the Scanner class for this program
```

2. Create a **Scanner** object:

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

3. Create provisions for user input:

```
System.out.print("Enter a string: ");
```

4. Read the input as a string:

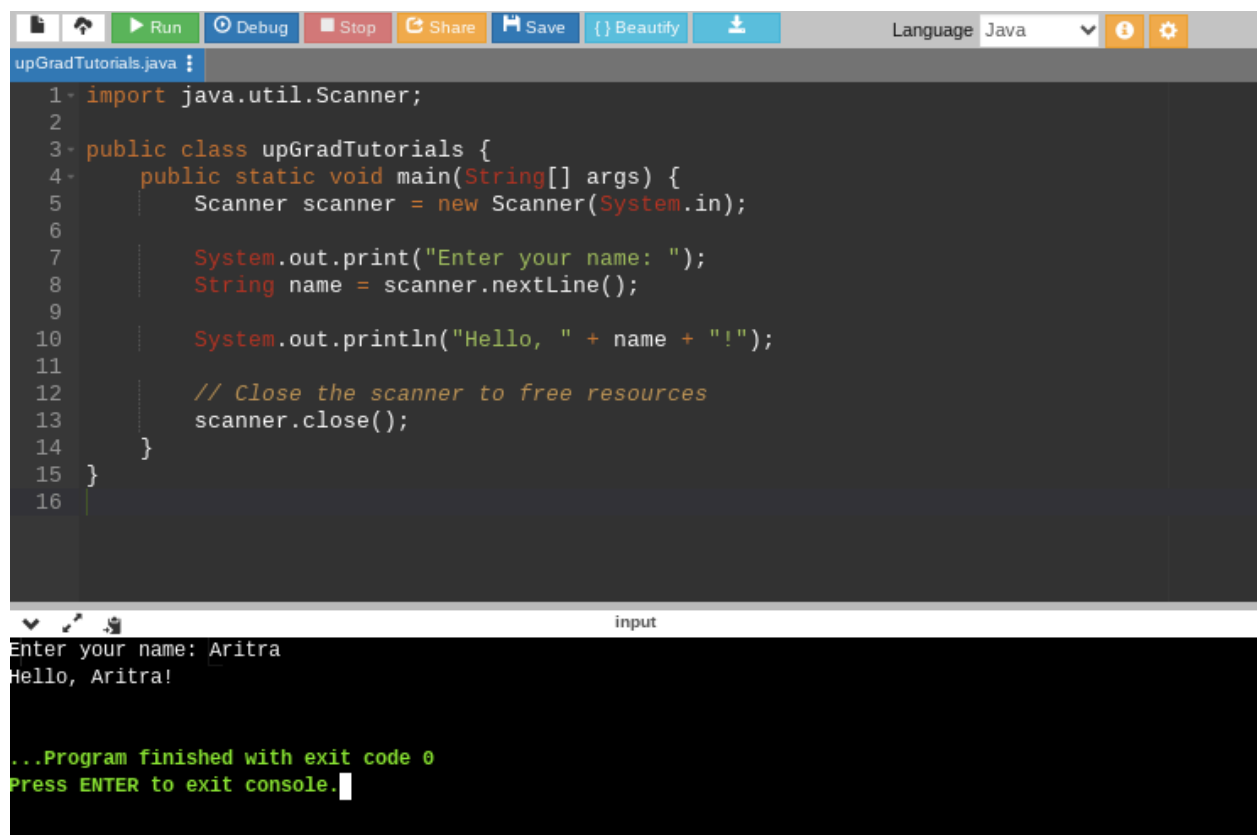
***String inputString = scanner.nextLine();***

The ***nextLine()*** method reads the entire input line, including any spaces or special characters, and returns it as a string.

## Method - 1: By Using Java Scanner Class

From the above section, we now have a general idea about using the ***Scanner*** class in Java. Let us look at some examples of taking string input in Java with two essential methods from the ***Scanner*** class:

### Java nextLine() Method

A screenshot of an IDE window titled 'upGradTutorials.java'. The code is as follows:

```
1 import java.util.Scanner;
2
3 public class upGradTutorials {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         System.out.print("Enter your name: ");
8         String name = scanner.nextLine();
9
10        System.out.println("Hello, " + name + "!");
11
12        // Close the scanner to free resources
13        scanner.close();
14    }
15 }
16
```

The IDE has a toolbar with 'Run', 'Debug', 'Stop', 'Share', 'Save', and 'Beautify' buttons. The language is set to 'Java'. Below the code editor, the console output is shown:

```
Enter your name: Aritra
Hello, Aritra!

...Program finished with exit code 0
Press ENTER to exit console.
```

```
import java.util.Scanner;
```

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

```
        System.out.print("Enter your name: ");
```

```
        String name = scanner.nextLine();
```

```
System.out.println("Hello, " + name + "!");
```

```
// Close the scanner to free resources
```

```
scanner.close();
```

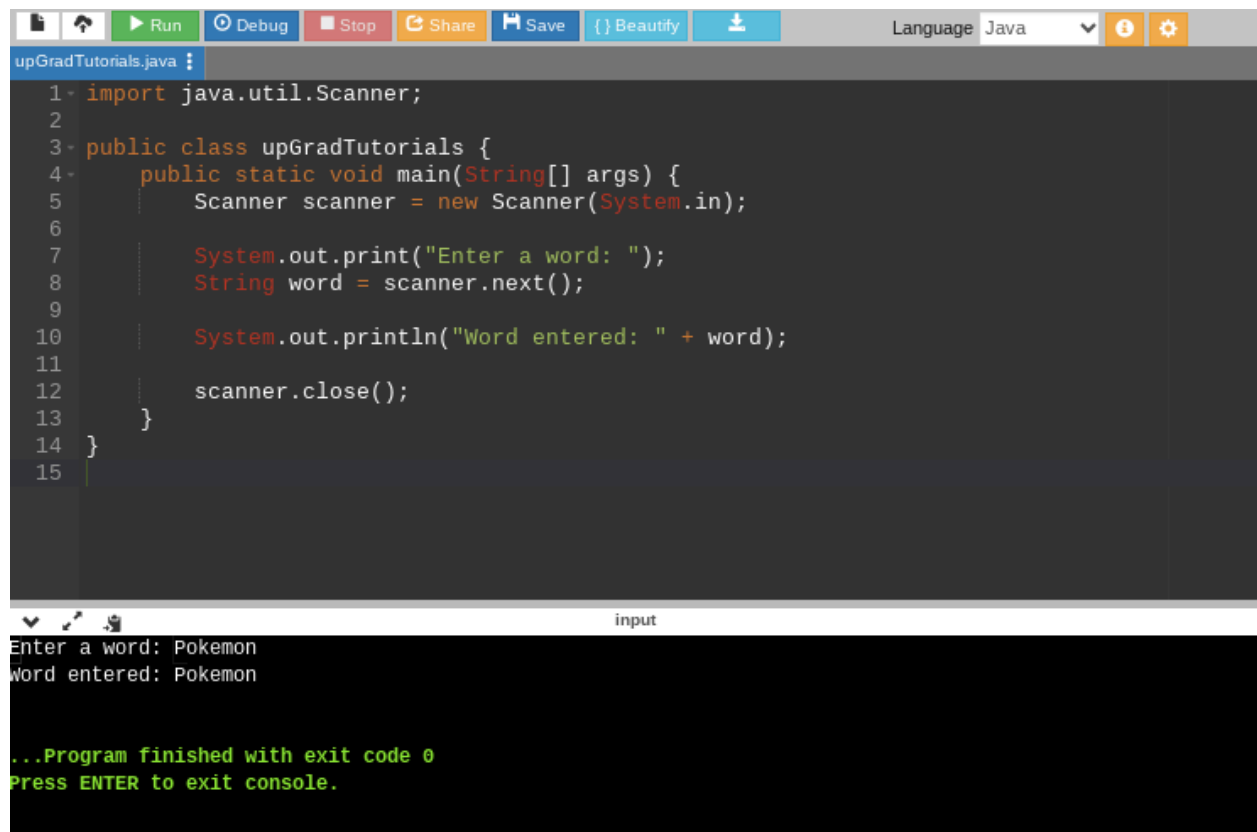
```
}
```

```
}
```

The above program starts by importing the **java.util.Scanner** class, which is used to read user input. The program then creates a **Scanner** object, **scanner**, to read input from the standard input (keyboard). It prompts the user to enter their name by displaying the message **"Enter your name: "**.

The **nextLine()** method of the **Scanner** object reads the entire line of input as a string and stores it in the **name** variable. The program then displays a greeting message using the entered name by concatenating it with the string **"Hello, "** and **"!"**. Finally, the **scanner.close()** method is called to close the **Scanner** object and release any associated resources.

## Java next() Method



The screenshot shows an IDE window titled 'upGradTutorials.java'. The code is as follows:

```
1 import java.util.Scanner;
2
3 public class upGradTutorials {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         System.out.print("Enter a word: ");
8         String word = scanner.next();
9
10        System.out.println("Word entered: " + word);
11
12        scanner.close();
13    }
14 }
15
```

Below the code editor, the console output is visible:

```
Enter a word: Pokemon
Word entered: Pokemon

...Program finished with exit code 0
Press ENTER to exit console.
```

```
import java.util.Scanner;
```

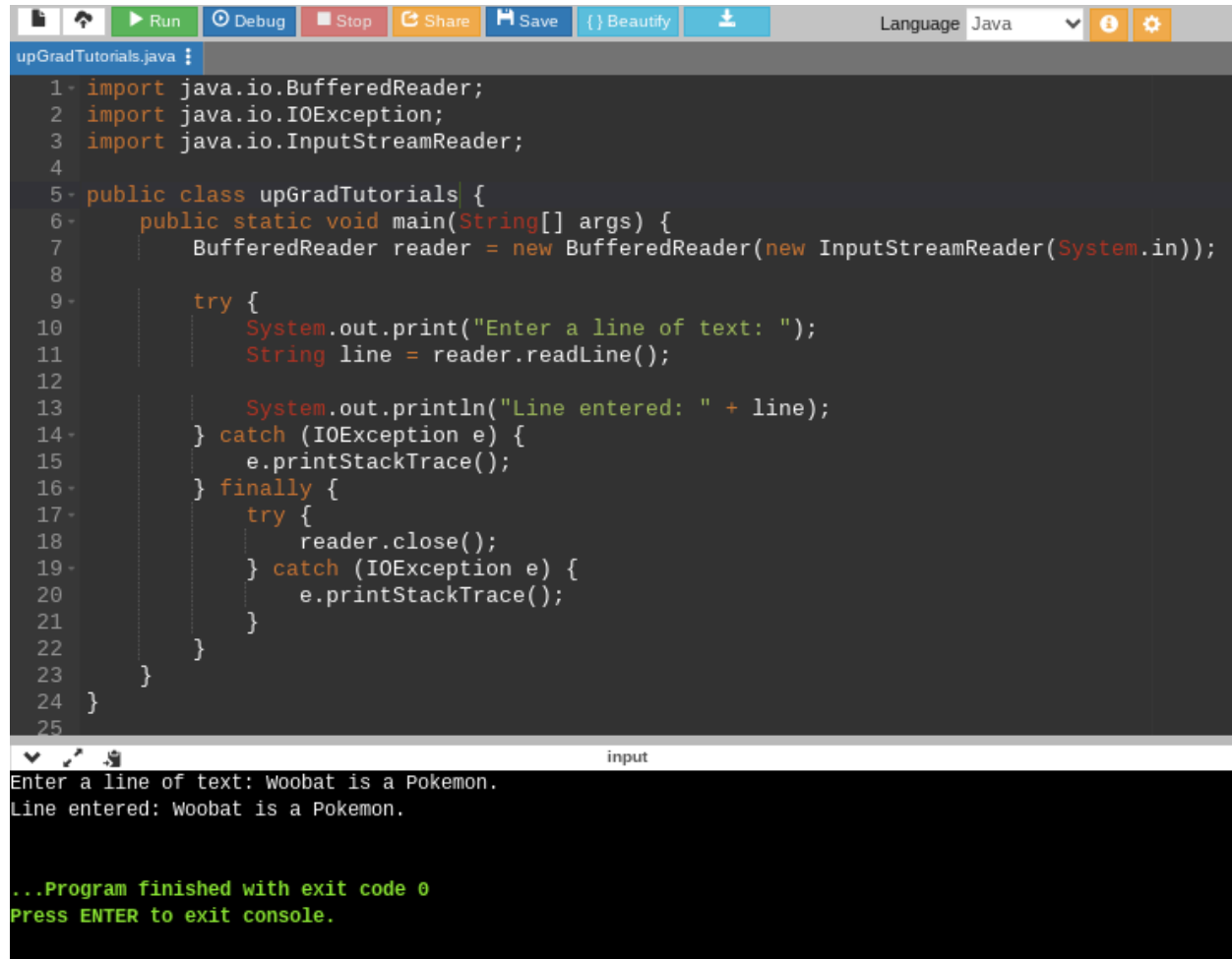
```
public class upGradTutorials {
```

```
public static void main(String[] args) {  
    Scanner scanner = new Scanner(System.in);  
  
    System.out.print("Enter a word: ");  
    String word = scanner.next();  
  
    System.out.println("Word entered: " + word);  
  
    scanner.close();  
}  
}
```

This program demonstrates the ***next()*** method in Java to read a single word as input from the user. The ***Scanner*** class is imported to enable reading user input. A ***Scanner*** object named ***scanner*** is created to read input from the standard input (keyboard). The program prompts the user to enter a word by displaying the message "***Enter a word:***".

The ***next()*** method of the ***Scanner*** object is used to read the next word from the input as a string and store it in the variable ***word***. The program then displays the word entered by concatenating it with the string "***Word entered:***". Finally, the ***scanner.close()*** method is called to close the ***Scanner*** object and release any associated resources.

## Method - 2: By Using Java BufferedReader Class



```
1- import java.io.BufferedReader;
2- import java.io.IOException;
3- import java.io.InputStreamReader;
4-
5- public class upGradTutorials {
6-     public static void main(String[] args) {
7-         BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
8-
9-         try {
10-             System.out.print("Enter a line of text: ");
11-             String line = reader.readLine();
12-
13-             System.out.println("Line entered: " + line);
14-         } catch (IOException e) {
15-             e.printStackTrace();
16-         } finally {
17-             try {
18-                 reader.close();
19-             } catch (IOException e) {
20-                 e.printStackTrace();
21-             }
22-         }
23-     }
24- }
25
```

input

Enter a line of text: Woobat is a Pokemon.  
Line entered: Woobat is a Pokemon.

...Program finished with exit code 0  
Press ENTER to exit console.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
public class upGradTutorials {
    public static void main(String[] args) {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
```

```
        try {
            System.out.print("Enter a line of text: ");
            String line = reader.readLine();

            System.out.println("Line entered: " + line);
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
```

```

    try {
        reader.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
}
}
}

```

The above program demonstrates the usage of the **BufferedReader** class in Java to read a line of text as input from the user. The **BufferedReader** class, along with **InputStreamReader** and **System.in**, is imported to enable reading user input. A **BufferedReader** object named **reader** is created by wrapping an **InputStreamReader** object around **System.in**. This allows reading input from the standard input (keyboard).

The program prompts the user to enter a line of text by displaying the message "**Enter a line of text:**". The **readLine()** method of the **BufferedReader** object is used to read the entire line of input as a string and store it in the variable **line**. The program then displays the entered line by concatenating it with the string "**Line entered:**".

In case of any input/output errors, the program catches and prints the exception stack trace. Finally, the **close()** method is called on the **BufferedReader** object to close it and release any associated resources.

## Method - 3: By Using the Command Line Arguments

```

1- public class upGradTutorials {
2-     public static void main(String[] args) {
3-         if (args.length > 0) {
4-             System.out.println("Command line arguments:");
5-             for (int i = 0; i < args.length; i++) {
6-                 System.out.println(args[i]);
7-             }
8-         } else {
9-             System.out.println("No command line arguments provided.");
10-        }
11-    }
12- }
13-

```

input

No command line arguments provided.

...Program finished with exit code 0  
Press ENTER to exit console.

```

public class upGradTutorials {

```

```

public static void main(String[] args) {
    if (args.length > 0) {
        System.out.println("Command line arguments:");
        for (int i = 0; i < args.length; i++) {
            System.out.println(args[i]);
        }
    } else {
        System.out.println("No command line arguments provided.");
    }
}
}

```

In the above program, the **main** method accepts an array of strings, **args**, as command line arguments. The program first checks if any command line arguments are provided by checking the length of the **args** array. If there are command line arguments, it enters a loop and displays each argument on a separate line using a **for** loop.

If no command line arguments are provided, it prints the message "**No command line arguments provided.**" This program allows you to pass arguments to the Java program when executing it from the command line. The provided command line arguments are then accessed and processed within the program.

If command line arguments are provided when running the program, the program will display the following:

```

Command line arguments:

```

We must run the program with the command line arguments (**apple**, **banana** and **orange**): **java upGradTutorials apple banana orange**

```

java upGradTutorials apple banana orange

```

After we provide the command line arguments, this will be the output:

```

Command line arguments:
apple
banana
orange

```

## Conclusion

When working with text-based data and user interactions, receiving string input is a crucial component of Java programming. The **Scanner** class makes reading string input from files or the console simple. Remember to handle exceptions and validate the input as necessary to ensure your program behaves appropriately.

You can make Java programs more interactive and user-friendly by being familiar with standard methods for validating, transforming, and processing string input. You can supply text-based data through string input, which can then be processed, changed, or stored to meet your program's needs.

## FAQs

### 1. Is it possible for me to accept several strings on a single line?

Yes, the **next()** method of the **Scanner** class can take several strings as input on a single line. This method reads the subsequent token from the input, a string of characters mostly separated by spaces. To read several strings entered on the same line, call the **next()** method multiple times.

### 2. Can I use a file instead of the console to input strings?

Yes, by constructing a **Scanner** object with a **File** object corresponding to the file we wish to read, we may use the **Scanner** class to read string input from a file. We would supply the **File** object to the **Scanner** constructor instead of **System.in**. Using this technique, we can read strings from a file and process them accordingly.

### 3. How can I change string inputs in Java to another data type?

We can use the methods the corresponding wrapper classes offer if we need to convert string input to any other data type, such as an integer or a double. For instance, we can use **Double.parseDouble()** to turn a string into a double or **Integer.parseInt()** to turn a string into an integer.