Command - Line Arguments

- A command-line argument is the information that directly follows the program's name on the command line when it is executed

- The command-line arguments are stored as strings in the String array passed to main( )

```
class CommandLineExample
{
 public static void main(String args[]){
 System.out.println("Your first argument is: "+args[0]);
 }
}
```

Command - line Arguments In Java

```
compile by > javac CommandLineExample.java
run by > java CommandLineExample Amit
```

```
Output: Your first argument is: Amit
```

Command - line Arguments In Java

Example of command-line argument that prints all the values

```
class A{
public static void main(String args[]){

for(int i=0;i<args.length;i++)
System.out.println(args[i]);
  }  }

compile by > javac A.java
run by > java A  Amit 1 3 abc
```

Command - line Arguments In Java

Output:

```
Amit
1
3
abc
```

Read Input from console in Java

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.

Advantages :

The input is buffered for efficient reading.

Drawback:

The wrapping code is hard to remember.

Command - line Arguments In Java

```
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);
    }}
```

Command - line Arguments In Java

Input:
Amit
Output:
Amit

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

Read Input from console in Java

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.

Advantages :

- Convenient methods for parsing primitives (nextInt(), nextFloat(), …) from the tokenized input.
- Regular expressions can be used to find tokens.

Drawback:

The reading methods are not synchronized

Read Input from console in Java

```
// Java program to demonstrate working of Scanner in
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);
}}
```

Read Input from console in Java

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.**

Drawback:

Does not work in non-interactive environment

Read Input from console in Java

```
// 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(name);
    }
}
```