ShreeSoft INFORMATICS

Address: Tirumala Plaza, Upendra Nagar, Cidco, Nashik.

Join us on Telegram:
https://t.me/ShreeSoftinformatics

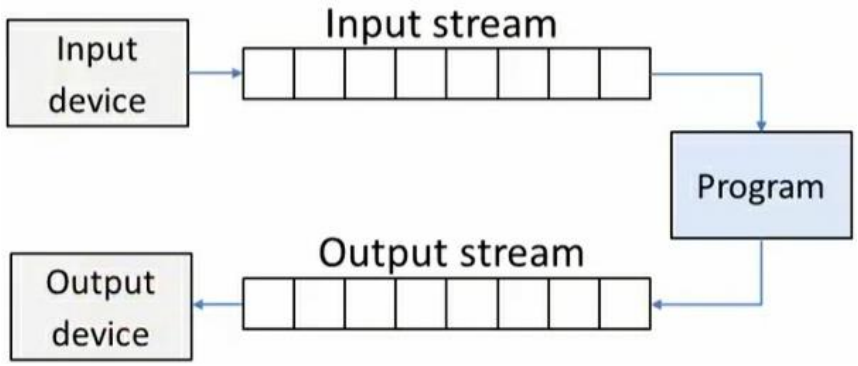
Join us on WhatsApp:
https://chat.whatsapp.com/BrLM0ZoWMbpA8nXi9Mrocl

Contact: +91 8308341531

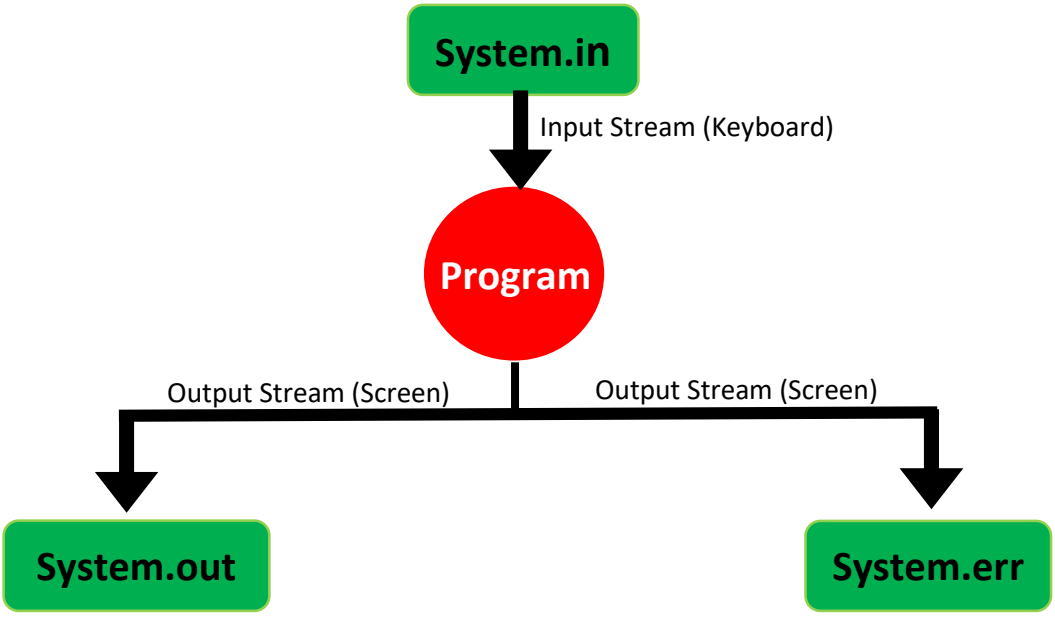
Basic Input Output in Java

What is input, output and Stream?

- Java input and output is an essential concept while working on java programming. It consists of elements such as input, output and stream.
- The input is the data that we give to the program.
- The output is the data what we receive from the program in the form of result.
- Stream represents flow of data or the sequence of data.
- To give input we use the **input stream** and to give output we use the **output stream**.



Before exploring various input and output streams lets look at **3 standard or default streams** that Java has to provide which are also most common in use:



Output Stream (Screen):

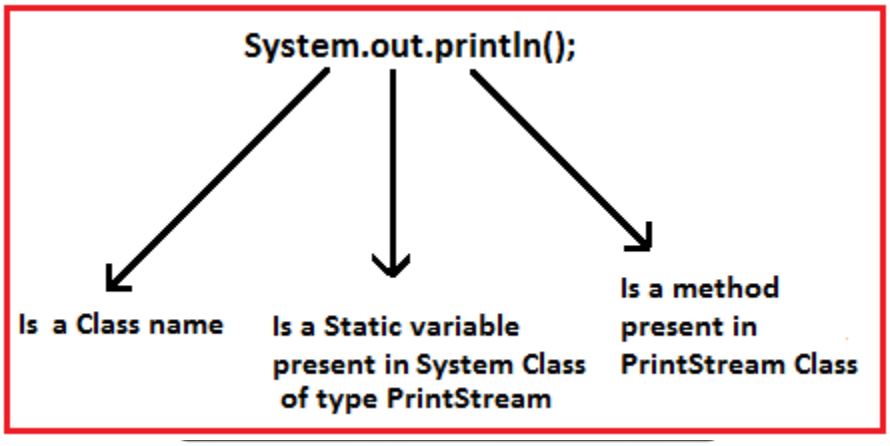
1) System.out:

This is standard output stream. System is a class that provides methods related to the "system" or environment where programs run. **out** is an object of **PrintStream** class which is a public and static member field of the **System** class and **println()** or **print()** or **printf()** is a public method of the **PrintStream** class which can be accessed by the object **out**.

- **System.out.println()** - This statement is used to print text or values in separate lines on the screen. This statement breaks the line after printing text or values on the screen.

- **System.out.print()** - This statement is used to print text or values in the same line on the screen. This statement does not break the line after printing text or values on the screen.

- **System.out.printf()** – The method is part of the java.io.PrintStream class and provides String formatting similar to the printf( ) function in C.



2) System.err:

It also represents PrintStream object, which by default represents a monitor. Normally, we use it to display error messages.

System.err is a PrintStream. System.err works like System.out except it is normally only used to output error texts. Some programs (like Eclipse) will show the output to System.err in red text, to make it more obvious that it is error text.

Example:

```
System.err.println("File not found error"); // File not found error
```

Input Stream (Keyboard):

3) System.in:

This field represents an InputStream object, which is a standard input device, known as a keyboard by default.

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

1. Using Command Line Argument
2. Using BufferedReader Class
3. Using Scanner Class
4. Using Console Class

Using Command Line Argument

Command Line Argument in Java is the information that is passed to the program when it is executed. The information passed is stored in the string array passed to the main( ) method and it is stored as a string. It is the information that directly follows the program's name on the command line when it is running.

As we said that above 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.

Example 1:

```
public class InputOperations {
	public static void main(String[] args) {
		int i;
		System.out.println("Arguments Length:"+args.length);
		for(i=0;i<args.length;i++) {
		System.out.println("args["+i+"] = "+ args[i]);
		}
	}
}
```

Example 2:

```
public class InputOperations {
	public static void main(String[] args) {
		int x,y;
		if(args.length == 2) {
			x = Integer.parseInt(args[0]);
			y = Integer.parseInt(args[1]);
			System.out.println("Addition = "+(x+y));
			System.out.println("Sbtraction = "+(x-y));
		}
		else {
			System.out.println("Invalid Number of Arguments");
		}
	}
}
```

Using BufferedReader 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.

Example 1:

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

public class InputOperations {
	public static void main(String[] args) {
		int x,y;
		BufferedReader bufferedReader = new BufferedReader(new
		InputStreamReader(System.in));
		try {
			System.out.println("Enter character:");
			int ch = bufferedReader.read();

			String read = bufferedReader.readLine();

			System.out.println("Enter String:");
			String str = bufferedReader.readLine();

			System.out.println("Your Entered Character:"+(char)ch);
			System.out.println("Your Entered String:"+str);

		} catch (IOException e) {
		e.printStackTrace();
		}
	}
}
```

Example 2:

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

public class InputOperations {
	public static void main(String[] args) {
		int x,y;

		BufferedReader bufferedReader = new BufferedReader(new
		InputStreamReader(System.in));

		try {
			System.out.println("Enter 2 Number:");
			x = Integer.parseInt(bufferedReader.readLine());
```

```
                y = Integer.parseInt(bufferedReader.readLine());

                System.out.println("Addition = "+(x+y));
                System.out.println("Sbtraction = "+(x-y));

            } catch (NumberFormatException | IOException e) {
                e.printStackTrace();
            }
        }
}
```

Using Scanner Class

- **Scanner in Java** is a predefined class that reads or scans the data dynamically from the keyboard or a text file. **Scanner class** in Java is found in the **java.util** package.

- In other words, Scanner class allows the user to read all types of numeric values, strings, and other types of data in java, whether it comes from a disk file, keyboard, or another source.

- It is the fastest and easiest way to receive the input from the user in java than InputStreamReader and BufferedReader.

- Scanner is a new util class that was added in Java 1.5 to simplify the task of getting input from the user. It is present in java.util.Scanner package.

- Scanner is a new util class that was added in Java 1.5 to simplify the task of getting input from the user. It is present in java.util.Scanner package.

- The Java Scanner class breaks the input into tokens using a delimiter which is whitespace by default. It provides many methods to read and parse various primitive values.

- The **Java Scanner class** is widely used to parse text for strings and primitive types using a regular expression. It is the simplest way to get input in Java. With the help of Scanner in Java, we can get input from the user in primitive types as well as strings such as int, long, double, byte, float, short, strings, etc.

Constructor: There are various constructor of Scanner class but we can learn in this chapter 2 only because others are learn in File handling chapter.

1) **Scanner(InputStream is):** This constructor creates a Scanner object with the specified InputStream is as a source for input.
2) **Scanner(String str):** This form of constructor creates an object of Scanner with the specified string str as a source for input.

Syntax to Create Scanner Object:

Scanner sc = new Scanner(System.in);

Here, System.in is an object of type InputStream.
The sc object reads the user input from System.in (i.e. keyboard).

Methods to read Input from Keyboard using Scanner class:

Method	Description
nextInt()	reads an int value from the user
nextFloat()	reads a float value form the user
nextBoolean()	reads a boolean value from the user
nextLine()	reads a line of text from the user
next()	reads a word from the user
nextByte()	reads a byte value from the user
nextDouble()	reads a double value from the user
nextShort()	reads a short value from the user
nextLong()	reads a long value from the user

Example:

```
import java.util.Scanner;
public class InputOperations {
	public static void main(String[] args) {
		byte b;
		short s;
		int i;
		long l;
		float f;
		double d;
		String str;
		char ch;
		boolean bool;
		Scanner scanner = new Scanner(System.in);
		System.out.println("Enter byte value:");
		b = scanner.nextByte();
		System.out.println("Enter short value:");
		s = scanner.nextShort();
		System.out.println("Enter int value:");
		i = scanner.nextInt();
		System.out.println("Enter long value:");
		l = scanner.nextLong();
		System.out.println("Enter float value:");
		f = scanner.nextFloat();
		System.out.println("Enter double value:");
		d = scanner.nextDouble();
		System.out.println("Enter String value:");
		str = scanner.next();
		System.out.println("Enter character value:");
		ch = scanner.next().charAt(0);
```

```
            System.out.println("Enter boolean value:");
            bool = scanner.nextBoolean();

            System.out.println("Your Entered byte value:"+b);
            System.out.println("Your Entered short value:"+s);
            System.out.println("Your Entered int value:"+i);
            System.out.println("Your Entered long value:"+l);
            System.out.println("Your Entered float value:"+f);
            System.out.println("Your Entered double value:"+d);
            System.out.println("Your Entered String value:"+str);
            System.out.println("Your Entered Character:"+ch);
            System.out.println("Your Entered boolean value"+bool);
            scanner.close();
        }
}
```

String Tokenization using the Scanner class

Java Scanner class is also used to parse the string into tokens and perform operations on them. There are some useful methods provided by the Scanner class to do this. These methods are:

Method	Description
Stream<String> tokens()	This method gives a stream of delimiter-separated tokens from the Scanner object which is in use.
String toString()	This method returns the string representation of Scanner using.
Scanner useDelimiter()	This method sets the delimiting pattern of the Scanner which is in use to the specified pattern.
void close()	This method closes this scanner.

Notes:

- Token: A *token* is a series of characters that ends with whitespace characters like blank, a tab character, and a carriage return.
- Delimeter: A delimiter is a character (, ; , :, #, @, $ etc.) that is used to separate items of data stored on a computer. It tells the computer to finish processing one item of data and move on to the next one.

Example 1:

```
import java.util.Scanner;
public class StringTokenizer1 {
      public static void main(String args[]) {
            String token = "ShreeSoft#Informatics#Java#Batch#Scanner#programs#2023";

            //Initialize Scanner object
            Scanner sc = new Scanner(token);

            //Initialize the string delimiter
            sc.useDelimiter("#");
```

```
            //Printing the tokenized Strings
            while (sc.hasNext()) {
            System.out.println(sc.next());
            }
            sc.close();
      }
}
```

Example 2:

```
import java.util.Scanner;
      public class StringTokenizer2 {
            public static void main(String args[]) {
            String input = "7 Java 13 Java ShreeSoft Java Informatics";

            Scanner sc = new Scanner(input).useDelimiter("\\s*Java\\s*");
            // \\s* means 0 or more repetitions of any whitespace character

            //Java is the pattern to find
            System.out.println(sc.nextInt()); // prints: 7
            System.out.println(sc.nextInt()); // prints: 13
            System.out.println(sc.next()); // prints: ShreeSoft
            System.out.println(sc.next()); // prints: Informatics

            sc.close();
      }
}
```

Advantages of Scanner class in Java

Scanner class can perform all works that a BufferedReader class does with the same efficiency. There are some advantages of using Scanner class in Java as

1) No need to use InputStreamReader and BufferedReader to accept data as input from user.
2) The main advantage of using Scanner class is to read primitive type values and string from the keyboard (System.in).
3) Scanner class helps to tokenize the underlying stream with delimiter of our choice.
4) Scanner class is better than BufferedReader to parse some files. BufferedReader is good enough for a simple read operation.
5) Scanner is not thread-safe, but BufferedReader is thread-safe.

Using Console Class:

Console in Java is a utility class in java.io package that provides access to the system console associated with JVM.

Console class was introduced in Java 1.6 version. It is mainly used for reading data from the console and writing data on the console. All read and write operations are synchronized.

Here, console refers to the character input device (keyboard) and character display device (screen display).

This utility class provides the various methods to access character-based console device connected with JVM (Java Virtual Machine).

Constructor of Console class

Console class does not provide any constructor in Java. We can create an object reference to the console using **System.console( )** method if JVM is connected to the console.

The syntax to get object reference of console class is as

```
Console c = System.console();
```

Here, System is a class that provides a static method console(). If JVM is not connected with any console, this method will return null.

Methods of Console class in Java

1) **void flush():** This method flushes the console and forces all buffered output to be written immediately.

2) **Console format(String fmtString, Object… args):** This method writes a formatted string to this console's output stream according to the specified format string fmtString and arguments args.

3) **Console printf(String format, Object… args):** This method writes a formatted string to this console's output stream using the specified format string and arguments.

4) **Reader reader():** It is used to retrieve the Reader object reference associated with this console. That is, this method returns a reference to a Reader connected to the console.

5) **String readLine():** The readLine() method is used to read a single line of text from the console. It reads and returns a string entered from the keyboard or user.

6) Input ends when the user presses ENTER. Null is returned if the end of the console input stream has been reached. On failure, an IOError is thrown.

7) **String readLine(String fmtString, Object… args):** This method displays a prompting string formatted using specified fmtString and args, and then reads a single line of text from the console.

8) Input ends when the user presses ENTER. Null is returned if the end of the console input stream has been reached. On failure, an IOError is thrown.

9) **char[ ] readPassword():** This method reads a password from the console with echoing.

10) **char[ ] readPassword(String fmtString, Object… args):** This overloaded method displays a prompting string formatted as specified by fmtString and args, and then reads a password from the console with echoing disabled.

11) **PrintWriter writer():** The writer() method is used to retrieve the **PrintWriter** object reference associated with this console. That is, it returns an object reference to a Writer connected to the console.

Example 1:

```
import java.io.Console;
public class ConsoleClassExample1 {
	public static void main(String[] args) {
		// Create the console object
		Console cnsl = System.console();

		//check console is null or not
		if (cnsl == null) {
		System.out.println("No console available");
		return;
		}

		String str = cnsl.readLine("Enter string : ");
		System.out.println(str);
	}
}
```

Example 2:

```
import java.io.Console;
public class ConsoleClassExample {
	public static void main(String[] args) {
		// Create a reference to the console.
		Console con = System.console();

		// Checking the console is available or not.
		if (con == null) {
			System.out.printf("Console is not available");
			return;
		}

		System.out.println("Enter your password: ");
		char[] ch = con.readPassword(); // Reading password.

		// Converting an array of char into string
		String pass = String.valueOf(ch);

		System.out.println("Password is: " + pass);
	}
}
```

Example 3:

```
import java.io.Console;
public class ConsoleClassExample {
	public static void main(String[] args) {
		Console cnsl = null;
		try {
			cnsl = System.console();
			if (cnsl != null) {
				String fmt = "%1$4s %2$10s %3$10s%n";
				// format
				cnsl.format(fmt, "Items", "Quanity", "Price");
				cnsl.format(fmt, "-----", "-----", "-----");
				cnsl.format(fmt, "Tomato", "1Kg", "15");
				cnsl.format(fmt, "Potato", "5Kg", "50");
				cnsl.format(fmt, "Onion", "2Kg", "30");
				cnsl.format(fmt, "Apple", "4Kg", "80");
			}
		} catch(Exception ex) {
			ex.printStackTrace();
		}
	}
}
```