

Chapter 4

Standard Input and Output

Introduction

A stream is required to accept input from the keyboard. A stream represents flow of data from one place to another place. It is just like a water-pipe where water flows. Like a water-pipe carries water from one place to another, a stream carries data from one place to another place. A stream can carry data from keyboard to memory or from memory to monitor/printer or from memory to a file. A stream is always required if we want to move data from one place to another.

Basically, there are two types of streams: input streams and output streams. Input streams are those streams which receive or read data coming from some other place (keyboard/ file/Network socket). Output streams are those streams which send or write data to some other place (monitor/printer/file/Network socket).

All streams are represented by classes in **java.io** (input and output) package. This package contains a lot of classes, all of which can be classified into two categories: input streams and output streams.

Keyboard is represented by a field, called in **System** class. When we write **System.in**, we are representing a standard input device, i.e. keyboard, by default. System class is found in **java.lang** (language) package and has three fields. These three streams are created automatically for us as shown below. All these fields represent some type of stream:

1. **System.in**: This represents **InputStream** object, which by default represents standard input device, i.e. keyboard.
2. **System.out**: This represents **PrintStream** object, which by default represents standard output device, i.e. monitor.
3. **System.err**: This field also represents **PrintStream** object, which by default represents standard output device, i.e. monitor.

Note that both **System.out** and **System.err** can be used to represent the monitor and hence any of these two can be used to send data to the monitor. **System.out** is used to display normal messages and results, whereas **System.err** is used to display error messages.

The **in**, **out** and **err** are static variables of **System** class. System class is stored in **java.lang** package which is imported in java program automatically. Static

member of any class is always access by *classname.static member* , that why in, out and err is always written as:

- **System.in**
- **System.out** and
- **System.err**

Let's understand the working of Java OutputStream and InputStream by the figure given below.

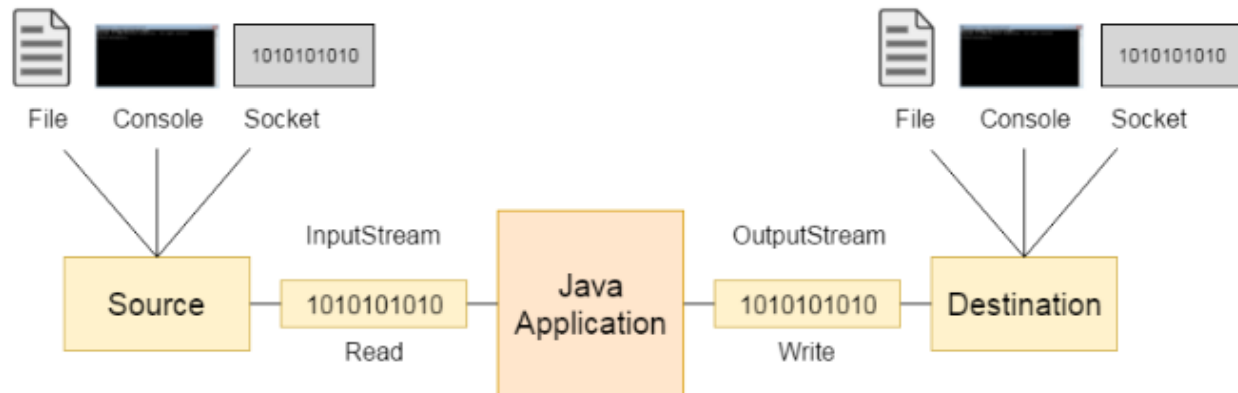


Figure 4.1

Reading data from keyboard:

There are many ways to read data from the keyboard. For example:

- **InputStreamReader**
- **Console**
- **Scanner**
- **Command Line Argument**

InputStreamReader class:

To accept the data from keyboard, i.e. **System.in**, we need to connect it to an input stream as some input stream is needed to read data. Figure 4.2 shows the working detail.

- Connects the keyboard to an input stream object. Here we can use **InputStreamReader** that can read data from the keyboard.

```
InputStreamReader isr=new InputStreamReader(System.in);
```

In this statement, we are creating **InputStreamReader** object and connecting the keyboard (**System.in**) to it.

- Connect **InputStreamReader** to **BufferedReader**, which is another input type of stream. We are using **BufferedReader** as it has got methods to read data properly, coming from the stream.

```
BufferedReader br = new BufferedReader(isr);
```

Here, we are creating **BufferedReader** object (br) and connecting the **InputStreamReader** object(isr) to it.

In the above two steps, we got **BufferedReader** object (br). These two steps can be combined and rewritten in a single statement as:

```
BufferedReader br = new BufferedReader(new  
InputStreamReader(System.in));
```

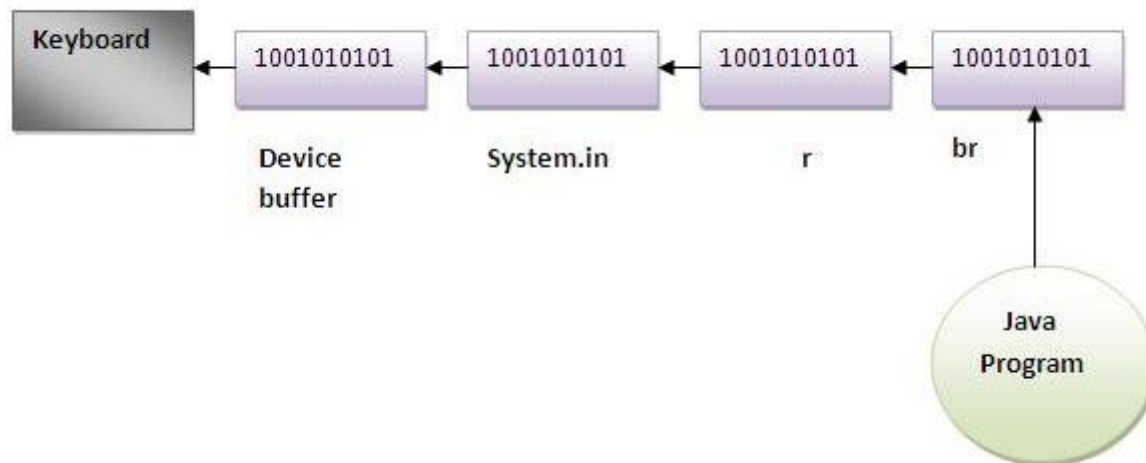


Figure 4.2

- Now, we can read the data coming from the keyboard using **read()** and **readLine()** methods available in **BufferedReader** class.

Advantages

- The input is buffered for efficient reading.

Drawback:

- The wrapping code is hard to remember.

Accepting a Single Character from the Keyboard

- Create a **BufferedReader** class object (br).
- Then, read a single character from the keyboard using read() method as:

```
char ch = br.read();
```

Here, the **read()** method reads a single character from the keyboard but it returns its ASCII number, which is an integer. Since, this integer number cannot be stored into character type variable **ch**, we should convert it into char type by writing (**char**) before the method as:

```
char ch = (char) br.read();
```

if read() method could not accept a character due to some reasons (like insufficient memory or illegal character), then it gives rise to a runtime error which is called by the name **IOException**, where IO stands for input/output and **Exception** represents runtime error.

So, when **read()** method is giving **IOException**, as a programmer, it is our duty to do something in case of the exceptions, let us throw this exception without handling it by writing throws **IOException** at the side of the method where read is used.

Program1: Accepting Single character from keyboard

```
import java.io.*;  
import java.lang.*;  
class Program1  
{  
    public static void main(String args[]) throws IOException  
    {  
        String str;  
        InputStreamReader isr= new InputStreamReader(System.in);  
        BufferedReader br = new BufferedReader(isr);  
        System.out.println("Enter the value for A");  
        char ch = (char) br.read();  
    }  
}
```

Accepting a String from the Keyboard

Observe the statement, where **readLine()** method is called using **BufferedReader** object (br)

```
String str = br.readLine()
```

Program2: In this example, we are connecting the **BufferedReader** stream with the **InputStreamReader** stream for reading the line by line data from the keyboard.

```
import java.io.*;
class Program2
{
    public static void main(String args[])throws Exception
    {
        InputStreamReader r=new InputStreamReader(System.in);
        BufferedReader br=new BufferedReader(r);
        System.out.print("Enter your name");
        String name=br.readLine();
        System.out.println("Welcome "+name);
    }
}
```

Output:

```
Enter your name Amit
Welcome Amit
```

Another Example of reading data from keyboard by InputStreamReader and BufferedReader class until the user writes stop

In this example, we are reading and printing the data until the user prints stop.

```
import java.io.*;
class Program3
{
    public static void main(String args[])throws Exception
    {
        InputStreamReader r=new InputStreamReader(System.in);
        BufferedReader br=new BufferedReader(r);

        String name="";
```

```

while(name.equals("stop"))
{
    System.out.println("Enter data: ");
    name=br.readLine();
    System.out.println("data is: "+name);
}
br.close();
r.close();
}
}

```

Output:

Enter data: Amit
 data is: Amit
 Enter data: 10
 data is: 10
 Enter data: stop
 data is: stop

Accepting an Number Value from the Keyboard

Now, let us follow these steps to accept an number(byte,short,int,long,float, double) from the keyboard.

First, we should accept the integer number from the keyboard as a string, using **readLine()** as:

```
String str = br.readLine();
```

Now, the number is in **str**, i.e. in form of a **String**. This should be converted into an **int** by using **parseInt()** method of **Integer** class as:

```
int n = Integer.parseInt(br.readLine()) // String to int conversion
```

Simirlarly

```
float f = Float.parseFloat(br.readLine()) // String to float conversion
```

```
double d = Double.parseDouble(br.readLine())//String to double conversion
```

```
long L = Long.parseLong(br.readLine())//String to Long conversion
```

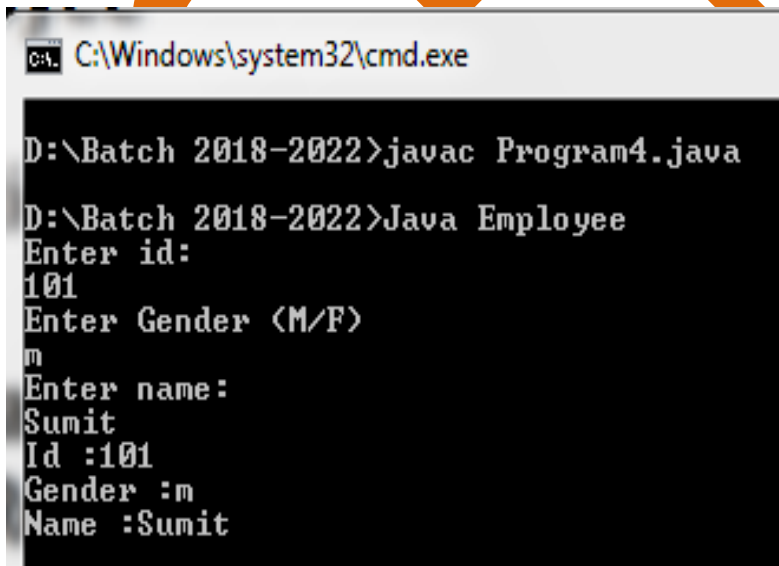
```
short s = Short.parseShort(br.readLine())//String to short conversion
```

```
byte b = Short.parseByte(br.readLine())//String to Byte conversion
```

Program4: Accepting and displaying employee details

```
import java.io.*;
class Employee
{
    public static void main(String args[])throws IOException
    {
        InputStreamReader r=new InputStreamReader(System.in);
        BufferedReader br=new BufferedReader(r);
        System.out.println("Enter id: ");
        int id = Integer.parseInt(br.readLine());
        System.out.println("Enter Gender (M/F) ");
        char gender= (char) br.read();
        br.skip(2);
        //char gender= br.readLine().charAt(0);
        System.out.println("Enter name: ");
        String name = br.readLine();
        System.out.println("Id :" + id);
        System.out.println("Gender :" + gender);
        System.out.println("Name :" + name);
    }
}
```

Output:



```
C:\Windows\system32\cmd.exe

D:\Batch 2018-2022>javac Program4.java

D:\Batch 2018-2022>Java Employee
Enter id:
101
Enter Gender (M/F)
m
Enter name:
Sumit
Id :101
Gender :m
Name :Sumit
```

Reading Input With java.util.Scanner class

There are various ways to read input from the keyboard, the **java.util.Scanner** class is one of them. The 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.

Commonly used methods of Scanner class:

There is a list of commonly used Scanner class methods:

- **public String next():** to read a string
- **public String nextLine():** to read a string till the end of the line..
- **public byte nextByte():** to read byte value.
- **public short nextShort():** to read short value.
- **public int nextInt():** to read int value.
- **public long nextLong():** to read long value.
- **public float nextFloat():** to read float value.
- **public double nextDouble():** to read double value.
- **public String next().charAt(0):** to read a single character

To read input from keyboard, we can use Scanner class as:

```
Scanner sc = new Scanner(System.in)
```

Now, If the user has given an integer value from the keyboard, it is stored into the Scanner object (sc) as a token. To retrieve that token, we can use the method **sc.nextInt()**. Following program will make this concept clear.

Advantages:

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

Drawback:

The reading method are not synchronized.

Example of java.util.Scanner class:

Let's see the simple example of the Scanner class which reads the int, string and double value as an input:

Program5:

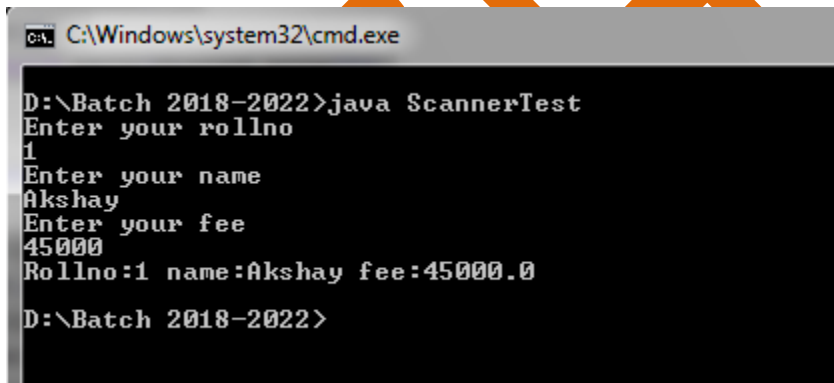
```
import java.util.Scanner;
class ScannerTest{
    public static void main(String args[]){

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter your rollno");
        int rollno=sc.nextInt();
        System.out.println("Enter your name");
        String name=sc.next();
        System.out.println("Enter your fee");
        double fee=sc.nextDouble();

        System.out.println("Rollno:"+rollno+" name:"+name+" fee:"+fee);
    }
}
```

Output:



```
C:\Windows\system32\cmd.exe

D:\Batch 2018-2022>java ScannerTest
Enter your rollno
1
Enter your name
Akshay
Enter your fee
45000
Rollno:1 name:Akshay fee:45000.0
D:\Batch 2018-2022>
```

Console class (I/O)

It has been becoming a preferred way for reading user's input from the Keyboard. 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 (such as in an IDE).

How to get the object of Console class?

System class provides a static method named `console()` that returns the unique instance of Console class.

Syntax:

```
1. public static Console console(){}
```

Commonly used methods of Console class:

1) public String readLine(): is used to read a single line of text from the console.

2) public String readLine(String fmt, Object... args): it provides a formatted prompt then reads the single line of text from the console.

3) public char[] readPassword(): is used to read password that is not being displayed on the console.

4) public char[] readPassword(String fmt, Object... args): it provides a formatted prompt then reads the password that is not being displayed on the console.

Example of Console class that reads name of user:

```
import java.io.*;
class A{
    public static void main(String args[]){

        Console c=System.console();

        System.out.println("Enter your name");
        String n=c.readLine();
        System.out.println("Welcome "+n);

    }
}
```

Example of Console class that reads password:

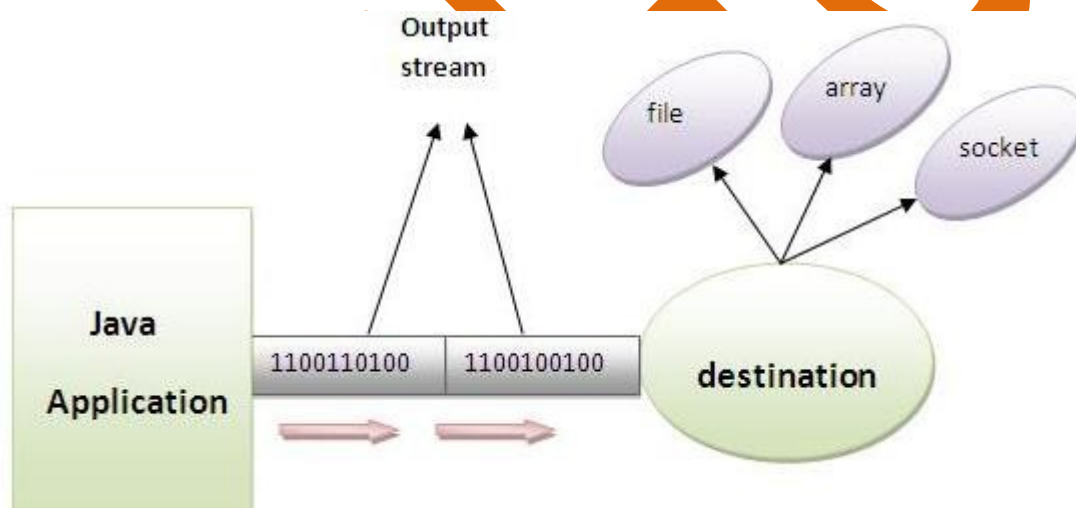
```

import java.io.*;
class A{
public static void main(String args[])
{
    Console c=System.console();
    System.out.println("Enter password");
    char[] ch=c.readPassword();
    System.out.println("Password is");
    for(char ch2:ch)
        System.out.print(ch2);
}
}

```

OutputStream

Java application uses an output stream to write data to a destination, it may be a file, an array, peripheral device or socket.



java.io.PrintStream class:

The PrintStream class provides methods to write data to another stream. The PrintStream class automatically flushes the data so there is no need to call flush() method. Moreover, its methods don't throw IOException.

Commonly used methods of PrintStream class:

There are many methods in PrintStream class. Let's see commonly used methods of PrintStream class:

- **public void print(boolean b):** it prints the specified boolean value.
- **public void print(char c):** it prints the specified char value.
- **public void print(char[] c):** it prints the specified character array values.
- **public void print(int i):** it prints the specified int value.
- **public void print(long l):** it prints the specified long value.
- **public void print(float f):** it prints the specified float value.
- **public void print(double d):** it prints the specified double value.
- **public void print(String s):** it prints the specified string value.
- **public void print(Object obj):** it prints the specified object value.
- **public void println(boolean b):** it prints the specified boolean value and terminates the line.
- **public void println(char c):** it prints the specified char value and terminates the line.
- **public void println(char[] c):** it prints the specified character array values and terminates the line.
- **public void println(int i):** it prints the specified int value and terminates the line.
- **public void println(long l):** it prints the specified long value and terminates the line.
- **public void println(float f):** it prints the specified float value and terminates the line.
- **public void println(double d):** it prints the specified double value and terminates the line.
- **public void println(String s):** it prints the specified string value and terminates the line.
- **public void println(Object obj):** it prints the specified object value and terminates the line.
- **public void println():** it terminates the line only.
- **public void printf(Object format, Object... args):** it writes the formatted string to the current stream.
- **public void printf(Locale l, Object format, Object... args):** it writes the formatted string to the current stream.
- **public void format(Object format, Object... args):** it writes the formatted string to the current stream using specified format.
- **public void format(Locale l, Object format, Object... args):** it writes the formatted string to the current stream using specified format.

Displaying Output with System.out.printf()

To format and display the output, **printf()** method is available in **PrintStream** class. This method works similar to printf() function in C. We know that **System.out** returns **PrintStream** class object, so to call printf() method, we can use:

System.out.printf()

The following format character can be used in printf():

- %s - string
- %c - char
- %d - decimal integer
- %f - float number
- %n - new line character

An example of using printf() is given below:

Let's see the simple example of printing integer value by format specifier.

```
class PrintStreamTest
{
    public static void main(String args[])
    {
        int a=10;
        System.out.printf("%d",a); // Note, out is the object of PrintStream class
    }
}
```

Output:10

Program6

```
class Program6
{
    public static void main(String args[])
    {
        String s1 = "Hello";
        int n = 65;
        float f = 15.1234f;
        System.out.printf("String = %s %n num = %d %n float = %f", s1,n,f);
    }
}
```