Stream

A stream can be defined as a sequence of data. There are two kinds of Streams −

InPutStream − The InputStream is used to read data from a source.

OutPutStream − The OutputStream is used for writing data to a destination.

Byte Streams

Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are, **FileInputStream** and **FileOutputStream**. Following is an example which makes use of these two classes to copy an input file into an output file −

Example

```
import java.io.*;public class CopyFile {

    public static void main(String args[]) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;

        try {
            in = new FileInputStream("input.txt");
            out = new FileOutputStream("output.txt");

            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
        }finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }}
```

Now let's have a file input.txt with the following content −

This is test for copy file.

As a next step, compile the above program and execute it, which will result in creating output.txt file with the same content as we have in input.txt. So let's put the above code in CopyFile.java file and do the following −

```
$javac CopyFile.java
$java CopyFile
```

Character Streams

Java **Byte** streams are used to perform input and output of 8-bit bytes, whereas Java **Character** streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are, **FileReader** and **FileWriter**. Though internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but here the major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.

We can re-write the above example, which makes the use of these two classes to copy an input file (having unicode characters) into an output file −

Example

```
import java.io.*;

public class CopyFile {

    public static void main(String args[]) throws IOException {

        FileReader in = null;

        FileWriter out = null;

        try {

            in = new FileReader("input.txt");

            out = new FileWriter("output.txt");

                        int c;

            while ((c = in.read()) != -1) {

                out.write(c);

            }

        }finally {

            if (in != null) {

                in.close();

            }

            if (out != null) {
```

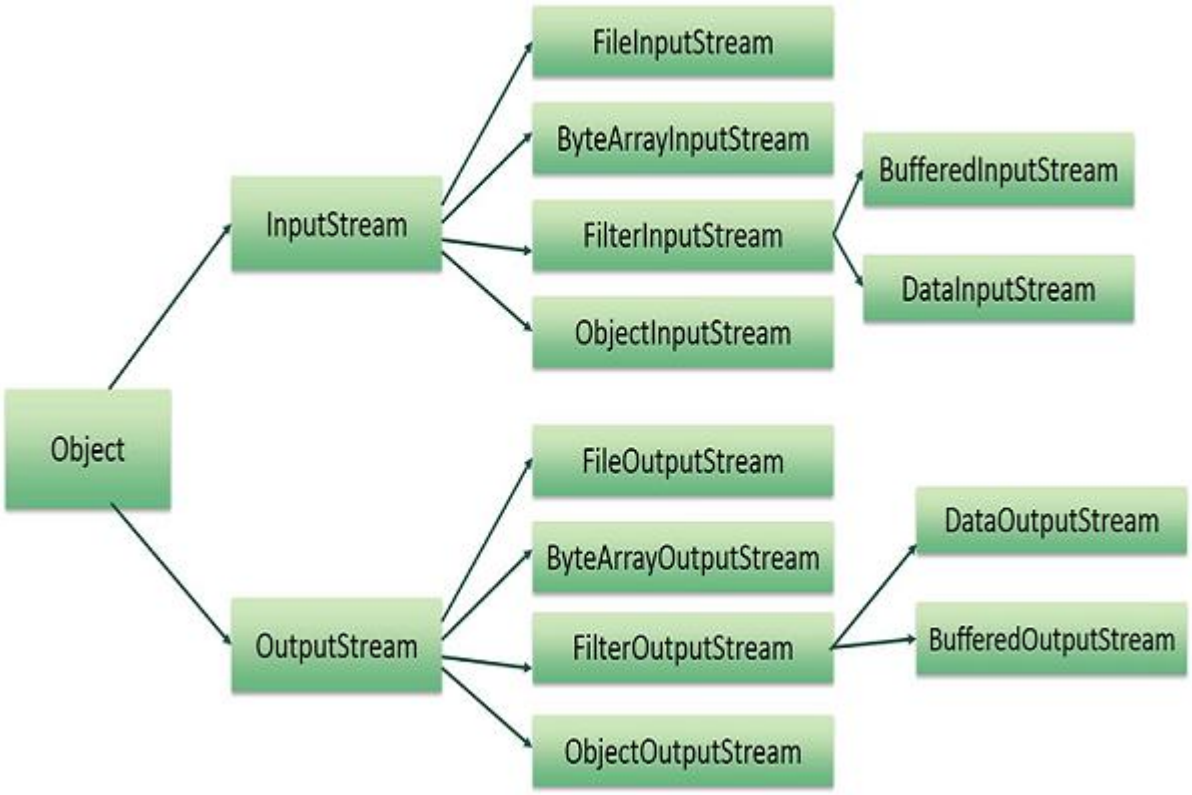
```
            out.close();
         }
      }
   }}
```

Now let's have a file input.txt with the following content −

This is test for copy file.
As a next step, compile the above program and execute it, which will result in creating output.txt file with the same content as we have in input.txt. So let's put the above code in CopyFile.java file and do the following −

```
$javac CopyFile.java
$java CopyFile
```

Here is a hierarchy of classes to deal with Input and Output streams.



Java BufferedReader Class

Java BufferedReader class is used to read the text from a character-based input stream. It can be used to read data line by line by readLine() method. It makes the performance fast. It inherits Reader class.

Java BufferedReader class declaration

```
public class BufferedReader extends Reader
```

Java BufferedReader class constructors

Constructor	Description
BufferedReader(Reader rd)	It is used to create a buffered character input stream that uses the default size for an input buffer.
BufferedReader(Reader rd, int size)	It is used to create a buffered character input stream that uses the specified size for an input buffer.

Java BufferedReader class methods

Method	Description
int read()	It is used for reading a single character.
int read(char[] cbuf, int off, int len)	It is used for reading characters into a portion of an array.
boolean markSupported()	It is used to test the input stream support for the mark and reset method.
String readLine()	It is used for reading a line of text.
boolean ready()	It is used to test whether the input stream is ready to be read.
long skip(long n)	It is used for skipping the characters.
void reset()	It repositions the stream at a position the mark method was last called on this input stream.
void mark(int readAheadLimit)	It is used for marking the present position in a stream.
void close()	It closes the input stream and releases any of the system resources associated with the stream.

Java BufferedReader Example

In this example, we are reading the data from the text file **testout.txt** using Java BufferedReader class.

```
import java.io.*;
public class BufferedReaderExample {
```

```
    public static void main(String args[])throws Exception{
        FileReader fr=new FileReader("D:\\testout.txt");
        BufferedReader br=new BufferedReader(fr);

        int i;
        while((i=br.read())!=-1){
        System.out.print((char)i);
        }
        br.close();
        fr.close();
    }
}
```

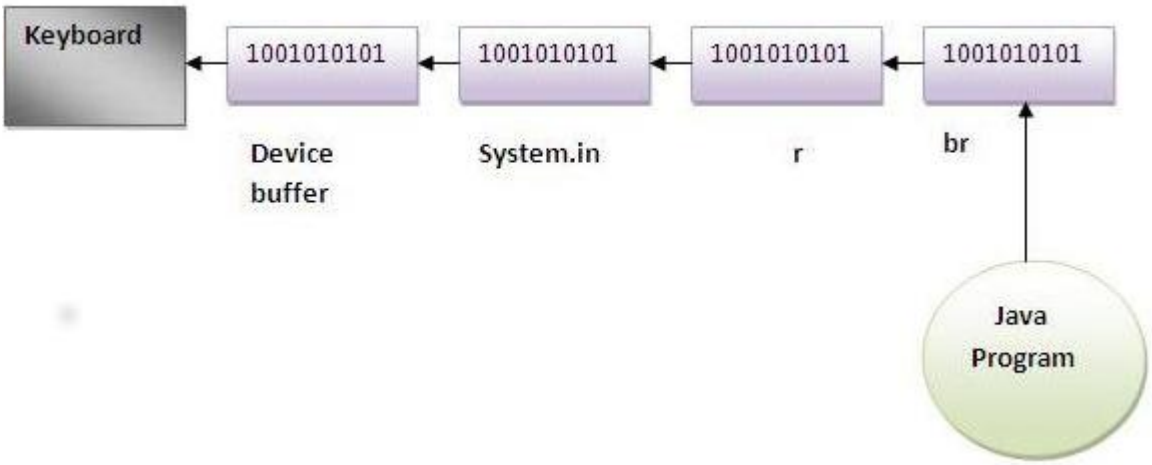
Reading data from console by InputStreamReader and BufferedReader

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

Output:

```
Enter your name
BCA Guys
Welcome BCA Guys
```



Another example of reading data from console until user writes stop

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

```
import java.io.*;
public class BufferedReaderExample{
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: BCA
data is: BCA
Enter data: 12
data is: 12
Enter data: stop
data is: stop
```