Filing in java

Abdul Haseeb

Java I/O

- **Java I/O** (Input and Output) is used *to process the input* and *produce the output*.
- Java uses the concept of a stream to make I/O operation fast.
- A stream is a sequence of data. In Java, a stream is composed of bytes:
  - Stream of River

java.io

- The java.io package contains all the classes required for input and output operations.

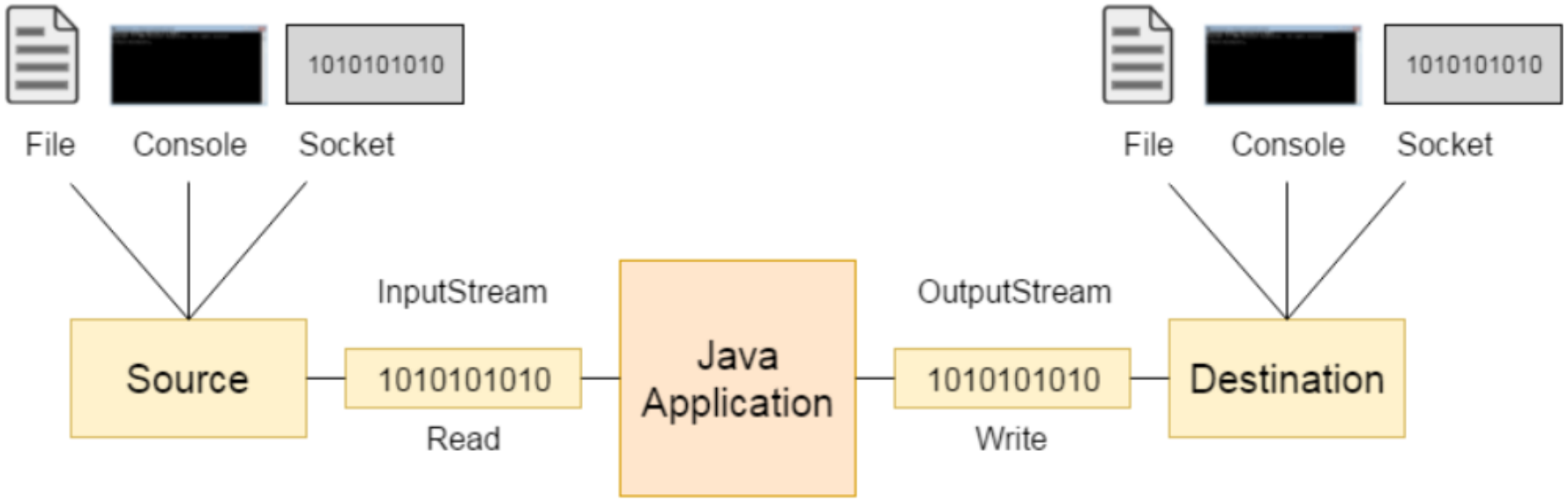
Streams

- In Java, 3 streams are created for us automatically. All these streams are attached with the console.
- **1) System.out:** standard output stream
- **2) System.in:** standard input stream
- **3) System.err:** standard error stream

Input/Output Stream

- Input stream to read data from a source; it may be a file, an array, peripheral device
- An output stream accepts output bytes and sends them to some sink.

Input Stream vs Output Stream



FileOutputStream Class

- A child of **OutputStream** class
- If you have to write primitive values into a file:
  - Use **FileOutputStream** class.
  - For writing Raw Bytes (unencoded bytes)

FileOutputStream class methods

Method	Description
protected void finalize()	It is used to clean up the connection with the file output stream.
void write(byte[] ary)	It is used to write **ary.length** bytes from the byte array to the file output stream.
void write(byte[] ary, int off, int len)	It is used to write **len** bytes from the byte array starting at offset **off** to the file output stream.
void write(int b)	It is used to write the specified byte to the file output stream.
FileChannel getChannel()	It is used to return the file channel object associated with the file output stream.
FileDescriptor getFD()	It is used to return the file descriptor associated with the stream.
void close()	It is used to closes the file output stream.

Java FileOutputStream Example 1: write byte

```
import java.io.FileOutputStream;
public class FileOutputStreamExample {
    public static void main(String args[]){
          try{
            FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
            fout.write(65);
            fout.close();
            System.out.println("success...");
          }catch(Exception e){System.out.println(e);}
      }
}
```

Java FileOutputStream example 2: write string

```
import java.io.FileOutputStream;
public class FileOutputStreamExample {
    public static void main(String args[]){
          try{
            FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
            String s="Welcome to javaTpoint.";
            byte b[]=s.getBytes();//converting string into byte array
            fout.write(b);
            fout.close();
            System.out.println("success...");
           }catch(Exception e){System.out.println(e);}
      }
}
```

Code to append data

```
try{
FileOutputStream fs=new FileOutputStream( name:"D://f1.txt",  append:true);
fs.write( b:'D');
fs.close();
}

catch(Exception e){
    System.out.println( x:"Not FOUND");
}
```

FileWriter

- For character-oriented data, it is preferred to use FileWriter than **FileOutputStream**.
- It automatically performs character encoding
- Can write strings directly

FileWriter Code

```
try{
FileWriter fs=new FileWriter( fileName:"D://hello.txt", append:true);
fs.append( csq:"Bye");
fs.close();
}

catch(Exception e){
    System.out.println( x:"Not FOUND");
}
```

FileInputStream

- Java FileInputStream class obtains input bytes from a file
- For reading character streams, FileReader can be used

Method	Description
int available()	It is used to return the estimated number of bytes that can be read from the input stream.
int read()	It is used to read the byte of data from the input stream.
int read(byte[] b)	It is used to read up to **b.length** bytes of data from the input stream.
int read(byte[] b, int off, int len)	It is used to read up to **len** bytes of data from the input stream.
long skip(long x)	It is used to skip over and discards x bytes of data from the input stream.
FileChannel getChannel()	It is used to return the unique FileChannel object associated with the file input stream.
FileDescriptor getFD()	It is used to return the FileDescriptor object.
protected void finalize()	It is used to ensure that the close method is call when there is no more reference to the file input stream.
void close()	It is used to closes the stream.

Java FileInputStream example 1: read single character

```
import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
          FileInputStream fin=new FileInputStream("D:\\testout.txt");
          int i=fin.read();
          System.out.print((char)i);

          fin.close();
        }catch(Exception e){System.out.println(e);}
       }
      }
```

Java FileInputStream example 2: read all characters

```
package com.javatpoint;

import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
          FileInputStream fin=new FileInputStream("D:\\testout.txt");
          int i=0;
          while((i=fin.read())!=-1){
           System.out.print((char)i);
          }
          fin.close();
        }catch(Exception e){System.out.println(e);}
       }
      }
```

```
import java.io.FileReader;
public class FileReaderExample {
    public static void main(String args[])throws Exception{
        FileReader fr=new FileReader("D:\\testout.txt");
        int i;
        while((i=fr.read())!=-1)
        System.out.print((char)i);
        fr.close();
    }
}
```

Buffered Output and Buffered Input Stream

- FileStreams read a block of data, if file is large there are many blocks to read
- Buffered Streams read larger chunks of data, and store them inside buffers
- Calls to system are reduced
- Improves performance

Code:

```
FileOutputStream fout= new FileOutputStream(“F://f1.txt”);
BufferedOutputStream b1=new BufferedOutputStream(fout);
string line=“Working with Buffered Streams…”;
byte b[]=s.getbytes();
b1.write(b);
b1.close();
fout.close();
```

Code:

```
FileInputStream fin= new FileInputStream(“F://f1.txt”);
BufferedInputStream b1=new BufferedInputStream(fout);
string line=“Working with Buffered Streams…”;
int ch;
while(ch=b1.read()!=-1)
        S.O.P((char)ch)
b1.close();
fin.close();
```

```
try {
FileReader fileReader = new FileReader(filePath);
BufferedReader bufferedReader = new BufferedReader(fileReader); String line;
while ((line = bufferedReader.readLine()) != null) {
        System.out.println(line); }
bufferedReader.close();
fileReader.close(); }

catch (IOException e)
{ System.out.println("Error reading file: " + e.getMessage());}
```