**JAVA IO Handling: Chapter 7**

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

We can perform **file handling in Java** by Java I/O API.

Stream

A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

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

Let's see the code to print **output and an error** message to the console.

1. System.out.println("simple message");
2. System.err.println("error message");

Let's see the code to get **input** from console.

1. **int** i=System.in.read();//returns ASCII code of 1st character
2. System.out.println((**char**)i);//will print the character

## OutputStream vs InputStream

The explanation of OutputStream and InputStream classes are given below:

### OutputStream

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

### InputStream

Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.

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



## OutputStream class

OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

### Useful methods of OutputStream

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public void write(int)throws IOException | is used to write a byte to the current output stream. |
| 2) public void write(byte[])throws IOException | is used to write an array of byte to the current output stream. |
| 3) public void flush()throws IOException | flushes the current output stream. |
| 4) public void close()throws IOException | is used to close the current output stream. |

### OutputStream Hierarchy



## InputStream class

InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes.

### Useful methods of InputStream

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public abstract int read()throws IOException | reads the next byte of data from the input stream. It returns -1 at the end of the file. |
| 2) public int available()throws IOException | returns an estimate of the number of bytes that can be read from the current input stream. |
| 3) public void close()throws IOException | is used to close the current input stream. |

### InputStream Hierarchy



Java FileOutputStream Example 1: write byte

1. **import** java.io.FileOutputStream;
2. **public** **class** FileOutputStreamExample {
3. **public** **static** **void** main(String args[]){
4. **try**{
5. FileOutputStream fout=**new** FileOutputStream("D:\\testout.txt");
6. fout.write(65);
7. fout.close();
8. System.out.println("success...");
9. }**catch**(Exception e){System.out.println(e);}
10. }
11. }

Output:

Success...

The content of a text file **testout.txt** is set with the data **A**.

testout.txt

A

Java FileOutputStream example 2: write string

1. **import** java.io.FileOutputStream;
2. **public** **class** FileOutputStreamExample {
3. **public** **static** **void** main(String args[]){
4. **try**{
5. FileOutputStream fout=**new** FileOutputStream("D:\\testout.txt");
6. String s="Welcome to javaTpoint.";
7. **byte** b[]=s.getBytes();//converting string into byte array
8. fout.write(b);
9. fout.close();
10. System.out.println("success...");
11. }**catch**(Exception e){System.out.println(e);}
12. }
13. }

Output:

Success...

Java FileInputStream example 1: read single character

1. **import** java.io.FileInputStream;
2. **public** **class** DataStreamExample {
3. **public** **static** **void** main(String args[]){
4. **try**{
5. FileInputStream fin=**new** FileInputStream("D:\\testout.txt");
6. **int** i=fin.read();
7. System.out.print((**char**)i);
8.
9. fin.close();
10. }**catch**(Exception e){System.out.println(e);}
11. }
12. }

**Note:** Before running the code, a text file named as **"testout.txt"**is required to be created. In this file, we are having following content:

Welcome to javatpoint.

After executing the above program, you will get a single character from the file which is 87 (in byte form). To see the text, you need to convert it into character.

Output:

W

Java FileInputStream example 2: read all characters

1. **package** com.javatpoint;
2.
3. **import** java.io.FileInputStream;
4. **public** **class** DataStreamExample {
5. **public** **static** **void** main(String args[]){
6. **try**{
7. FileInputStream fin=**new** FileInputStream("D:\\testout.txt");
8. **int** i=0;
9. **while**((i=fin.read())!=-1){
10. System.out.print((**char**)i);
11. }
12. fin.close();
13. }**catch**(Exception e){System.out.println(e);}
14. }
15. }

Output:

Welcome to javaTpoint

Java FileWriter Example

In this example, we are writing the data in the file testout.txt using Java FileWriter class.

1. **package** com.javatpoint;
2. **import** java.io.FileWriter;
3. **public** **class** FileWriterExample {
4. **public** **static** **void** main(String args[]){
5. **try**{
6. FileWriter fw=**new** FileWriter("D:\\testout.txt");
7. fw.write("Welcome to javaTpoint.");
8. fw.close();
9. }**catch**(Exception e){System.out.println(e);}
10. System.out.println("Success...");
11. }
12. }

Output:

Success...

testout.txt:

Welcome to javaTpoint.

Java FileReader Example

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

1. **package** com.javatpoint;
2.
3. **import** java.io.FileReader;
4. **public** **class** FileReaderExample {
5. **public** **static** **void** main(String args[])**throws** Exception{
6. FileReader fr=**new** FileReader("D:\\testout.txt");
7. **int** i;
8. **while**((i=fr.read())!=-1)
9. System.out.print((**char**)i);
10. fr.close();
11. }
12. }

Here, we are assuming that you have following data in "testout.txt" file:

Welcome to javaTpoint.

Output:

Welcome to javaTpoint.