

## Lecture – 16 (File I/O)

**Java I/O** (Input and Output) is used to process the input and produce the output.

Java uses the concept of stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

### 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 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 error** message to the console.

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

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

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

### Java FileOutputStream Class

Java FileOutputStream is an output stream used for writing data to a file.

If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.

## FileOutputStream class declaration

Let's see the declaration for `Java.io.FileOutputStream` class:

```
public class FileOutputStream extends OutputStream
```

## 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);}
    }
}
```

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

```
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 R-Creation.";
        }
    }
}
```

```
        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);
    }
}
```

Output:

Success...

The content of a text file **testout.txt** is set with the data **Welcome to R-Creation.**

testout.txt

Welcome to R-Creation.

## Java FileInputStream Class

Java `FileInputStream` class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use `FileReader` class.

## Java FileInputStream class declaration

Let's see the declaration for `java.io.FileInputStream` class:

```
public class FileInputStream extends InputStream
```

## 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);
}
}
```

**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 R-Creation.

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

```
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);
        }
    }
}
```

```
}  
}
```

Output:

Welcome to R-Creation

## Java BufferedOutputStream Class

Java BufferedOutputStream class is used for buffering an output stream. It internally uses buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.

For adding the buffer in an OutputStream, use the BufferedOutputStream class. Let's see the syntax for adding the buffer in an OutputStream:

```
OutputStream os= new BufferedOutputStream(new FileOutputStream("D:\\IO Package\\testout.txt"));
```

## Java BufferedOutputStream class declaration

Let's see the declaration for Java.io.BufferedOutputStream class:

```
public class BufferedOutputStream extends FilterOutputStream
```

## Example of BufferedOutputStream class:

In this example, we are writing the textual information in the BufferedOutputStream object which is connected to the FileOutputStream object. The flush() flushes the data of one stream and send it into another. It is required if you have connected the one stream with another.

```
import java.io.*;  
public class BufferedOutputStreamExample{  
    public static void main(String args[])throws Exception{  
        FileOutputStream fout=new FileOutputStream("D:\\testout.txt");  
        BufferedOutputStream bout=new BufferedOutputStream(fout);  
        String s="Welcome to R-Creation.";  
        byte b[]=s.getBytes();
```

```
bout.write(b);  
bout.flush();  
bout.close();  
fout.close();  
System.out.println("success");  
}  
}
```

Output:

Success

testout.txt

Welcome to R-Creation.

## Java BufferedInputStream Class

Java BufferedInputStream class is used to read information from stream. It internally uses buffer mechanism to make the performance fast.

The important points about BufferedInputStream are:

- When the bytes from the stream are skipped or read, the internal buffer automatically refilled from the contained input stream, many bytes at a time.
- When a BufferedInputStream is created, an internal buffer array is created.

## Java BufferedInputStream class declaration

Let's see the declaration for Java.io.BufferedInputStream class:

```
public class BufferedInputStream extends FilterInputStream
```

## Example of Java BufferedInputStream

Let's see the simple example to read data of file using BufferedInputStream:

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

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

R-Creation

Output:

R-Creation

## Java ByteArrayOutputStream Class

Java ByteArrayOutputStream class is used to **write common data** into multiple files. In this stream, the data is written into a byte array which can be written to multiple streams later.

The ByteArrayOutputStream holds a copy of data and forwards it to multiple streams.

The buffer of ByteArrayOutputStream automatically grows according to data.

## Java ByteArrayOutputStream class declaration

Let's see the declaration for `Java.io.ByteArrayOutputStream` class:

```
public class ByteArrayOutputStream extends OutputStream
```

## Example of Java `ByteArrayOutputStream`

Let's see a simple example of java `ByteArrayOutputStream` class to write common data into 2 files: `f1.txt` and `f2.txt`.

```
import java.io.*;
public class DataStreamExample {
    public static void main(String args[])throws Exception{
        FileOutputStream fout1=new FileOutputStream("D:\\f1.txt");
        FileOutputStream fout2=new FileOutputStream("D:\\f2.txt");
        ByteArrayOutputStream bout=new ByteArrayOutputStream();
        bout.write(65);
        bout.writeTo(fout1);
        bout.writeTo(fout2);
        bout.flush();
        bout.close();//has no effect
        System.out.println("Success...");
    }
}
```

Output:

Success...

f1.txt:

A

f2.txt:

A

## Example of Java `ByteArrayInputStream`



---

Let's see a simple example of java `ByteArrayInputStream` class to read byte array as input stream.

```
import java.io.*;
public class ReadExample {
    public static void main(String[] args) throws IOException {
        byte[] buf = { 35, 36, 37, 38 };
        // Create the new byte array input stream
        ByteArrayInputStream byt = new ByteArrayInputStream(buf);
        int k = 0;
        while ((k = byt.read()) != -1) {
            //Conversion of a byte into character
            char ch = (char) k;
            System.out.println("ASCII value of Character is:" + k + "; Special character is: " + ch);
        }
    }
}
```

Output:

```
ASCII value of Character is:35; Special character is: #
ASCII value of Character is:36; Special character is: $
ASCII value of Character is:37; Special character is: %
ASCII value of Character is:38; Special character is: &
```

## Java `DataOutputStream` Class

Java `DataOutputStream` class allows an application to write primitive Java data types to the output stream in a machine-independent way.

Java application generally uses the data output stream to write data that can later be read by a data input stream.

## Java `DataOutputStream` class declaration

Let's see the declaration for java.io.`DataOutputStream` class:

```
public class DataOutputStream extends FilterOutputStream implements DataOutput
```

## Example of DataOutputStream class

In this example, we are writing the data to a text file **testout.txt** using DataOutputStream class.

```
import java.io.*;
public class OutputExample {
    public static void main(String[] args) throws IOException {
        FileOutputStream file = new FileOutputStream(D:\\testout.txt);
        DataOutputStream data = new DataOutputStream(file);
        data.writeInt(65);
        data.flush();
        data.close();
        System.out.println("Success...");
    }
}
```

Output:

Success...

testout.txt:

A

## Java DataInputStream Class

Java DataInputStream class allows an application to read primitive data from the input stream in a machine-independent way.

Java application generally uses the data output stream to write data that can later be read by a data input stream.

## Java DataInputStream class declaration

Let's see the declaration for java.io.DataInputStream class:

```
public class DataInputStream extends FilterInputStream implements DataInput
```

---

## Example of DataInputStream class

In this example, we are reading the data from the file testout.txt file.

```
import java.io.*;
public class DataStreamExample {
    public static void main(String[] args) throws IOException {
        InputStream input = new FileInputStream("D:\\testout.txt");
        DataInputStream inst = new DataInputStream(input);
        int count = input.available();
        byte[] ary = new byte[count];
        inst.read(ary);
        for (byte bt : ary) {
            char k = (char) bt;
            System.out.print(k+"-");
        }
    }
}
```

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

JAVA

Output:

J-A-V-A

## Java FileWriter Class

Java FileWriter class is used to write character-oriented data to a file. It is character-oriented class which is used for file handling in java.

Unlike FileOutputStream class, you don't need to convert string into byte array because it provides method to write string directly.

## Java FileWriter class declaration

Let's see the declaration for Java.io.FileWriter class:

```
public class FileWriter extends OutputStreamWriter
```

## Java FileWriter Example

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

```
import java.io.FileWriter;
public class FileWriterExample {
    public static void main(String args[]){
        try{
            FileWriter fw=new FileWriter("D:\\testout.txt");
            fw.write("Welcome to R-Creation.");
            fw.close();
        }catch(Exception e){
            System.out.println(e);
        }
        System.out.println("Success...");
    }
}
```

Output:

Success...

testout.txt:

Welcome to R-Creation.

## Java FileReader Class

Java FileReader class is used to read data from the file. It returns data in byte format like FileInputStream class.

It is character-oriented class which is used for file handling in java.

## Java FileReader class declaration

Let's see the declaration for Java.io.FileReader class:

```
public class FileReader extends InputStreamReader
```

## Java FileReader Example

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

```
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();
    }
}
```

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

Welcome to R-Creation.

Output:

Welcome to R-Creation.

## Java BufferedWriter Class

Java BufferedWriter class is used to provide buffering for Writer instances. It makes the performance fast. It inherits Writer class. The buffering characters are used for providing the efficient writing of single arrays, characters, and strings.

## Class declaration

Let's see the declaration for Java.io.BufferedWriter class:

```
public class BufferedWriter extends Writer
```

## Example of Java BufferedWriter

Let's see the simple example of writing the data to a text file **testout.txt** using Java BufferedWriter.

```
import java.io.*;
public class BufferedWriterExample {
    public static void main(String[] args) throws Exception {
        FileWriter writer = new FileWriter("D:\\testout.txt");
        BufferedWriter buffer = new BufferedWriter(writer);
        buffer.write("Welcome to R-Creation.");
        buffer.close();
        System.out.println("Success");
    }
}
```

Output:

success

testout.txt:

Welcome to R-Creation.

## 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

Let's see the declaration for `Java.io.BufferedReader` class:

```
public class BufferedReader extends Reader
```

## 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();
    }
}
```

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

Welcome to R-Creation.

Output:

Welcome to R-Creation.

## 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);
```



*track of IT*

215/318, Hasina Monjil (1<sup>st</sup> Floor), Muradpur, Panchlaish, Chittagong.  
Website: [www.rcreation-bd.com](http://www.rcreation-bd.com), Contact: +8801722-964303

---

```
System.out.println("Enter your name");  
String name=br.readLine();  
System.out.println("Welcome "+name);  
}  
}
```

Output:

```
Enter your name  
Jon Kabir  
Welcome Jon Kabir
```