

## File Handling

The java.io package contains nearly every class you might ever need to perform input and output (I/O) in Java. All these streams represent an input source and an output destination. The stream in the java.io package supports many data such as primitives, Object, localized characters, etc. A stream can be defined as a sequence of data. The InputStream is used to read data from a source and the OutputStream is used for writing data to a destination.

Examples to Read Write from File are listed below

### Reading Character from File

```
import java.io.*;
class char_read_file
{
    public static void main(String args[])
    {
        File infile = new File("File.txt");
        FileReader ins = null;
        try
        {
            ins = new FileReader(infile);
            int ch;
            while((ch=ins.read())!=-1)
            {
                System.out.print((char)ch);
            }
        }
        catch(IOException e)
        {
            System.out.println(e);
        }
        finally
        {
            try
            {
                ins.close();
            }
            catch(IOException e)
            {}
        }
    }
}
```

Declare and Create Input File  
Used to Read Characters from File

*read()* method returns the integer value of the character present in Text File and returns "-1" to indicate End of File.  
To display character on screen use (char) Integer Value for typecasting

## Output

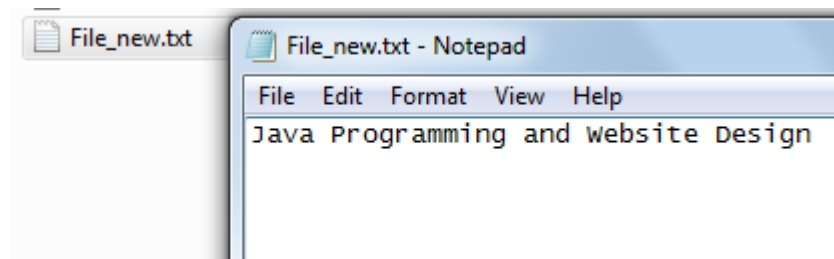
```
C:\Achin Jain\File>javac char_read_file.java
C:\Achin Jain\File>java char_read_file
Achin Jain
Assistant Professor
CSE Department
NIEC
C:\Achin Jain\File>_
```

## Writing Character to File

```
import java.io.*;
class char_write_file
{
    public static void main(String args[])
    {
        File outfile = new File("File_new.txt");
        FileWriter outs = null;
        try
        {
            outs = new FileWriter(outfile);
            int ch;
            String s = "Java Programming and Website Design";
            outs.write(s);
        }
        catch(IOException e)
        {
            System.out.println(e);
        }
        finally
        {
            try
            {
                outs.close();
            }
            catch(IOException e)
            {}
        }
    }
}
```

*write()* method is used to write the contents to the File

## Output



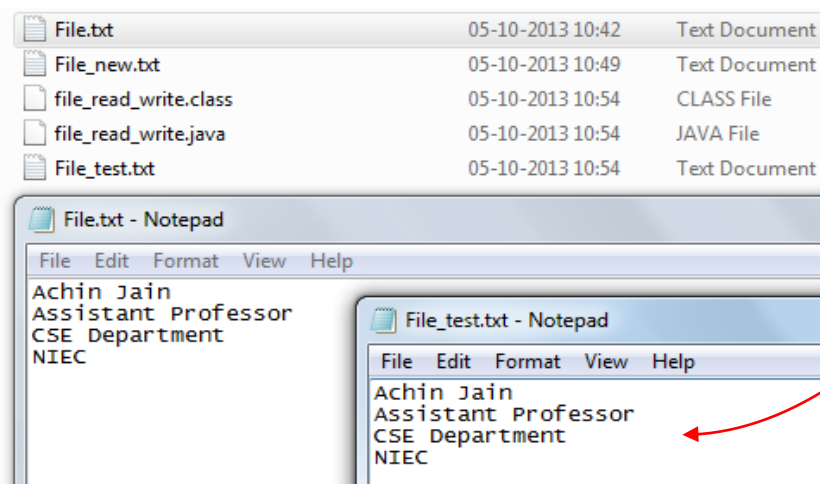
## Read and Write to File

```

class file_read_write
{
    public static void main(String args[])
    {
        File infile = new File("File.txt");
        File outfile = new File("File_test.txt");
        FileReader ins = null;
        FileWriter outs = null;
        try
        {
            ins = new FileReader(infile);
            outs = new FileWriter(outfile);
            int ch;
            while((ch=ins.read())!=-1)
            {
                outs.write(ch);
            }
        }
        catch(IOException e)
        {
            System.out.println(e);
        }
        finally
        {
            try
            {
                ins.close();
                outs.close();
            }
            catch(IOException e)
            {}
        }
    }
}

```

## Output



## Reading Bytes from File

```
import java.io.*;
class read_byte
{
    public static void main(String args[])
    {
        FileInputStream infile = null;
        try
        {
            infile = new FileInputStream("File.txt");
            int b;
            while((b = infile.read()) != -1)
            {
                System.out.print((char)b);
            }
            infile.close();
        }
        catch(IOException e)
        {
            System.out.println(e);
        }
    }
}
```

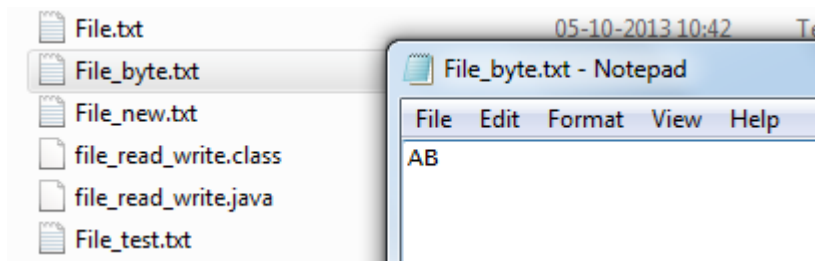
## Output

```
C:\Achin Jain\File>javac read_byte.java
C:\Achin Jain\File>java read_byte
Achin Jain
Assistant Professor
CSE Department
NIEC
C:\Achin Jain\File>_
```

## Writing Byte to File

```
import java.io.*;
class write_byte
{
    public static void main(String args[])
    {
        FileOutputStream outfile = null;
        byte b1[]={ 'A', 'B' };
        try
        {
            outfile = new FileOutputStream("File_byte.txt");
            outfile.write(b1);
            outfile.close();
        }
        catch(IOException e)
        {
            System.out.println(e);
        }
    }
}
```

## Output




## Reading Console Input

Java input console is accomplished by reading from **System.in**. To obtain a character-based stream that is attached to the console, you wrap **System.in** in a **BufferedReader** object, to create a character stream. Once **BufferedReader** is obtained, we can use `read()` method to reach a character or `readLine()` method to read a string from the console.

### 1 Reading Characters

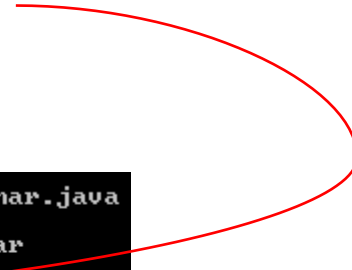
To read a character from a **BufferedReader**, use `read()` method. Each time that `read()` is called, it reads a character from the input stream and returns it as an integer value. It returns `.1` when the end of the stream is encountered. As you can see, it can throw an **IOException**.

```
import java.io.*;
class console_read_char
{
    public static void main(String args[]) throws IOException
    {
        char c;
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter Characters");
        System.out.println((char)br.read());
    }
}
```



### Output

```
C:\Achin Jain\File>javac console_read_char.java
C:\Achin Jain\File>java console_read_char
Enter Characters
a
a
```



### 2 Reading Strings

```
import java.io.*;
class console_read_string
{
    public static void main(String args[]) throws IOException
    {
        String str;
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter String");
        System.out.println(br.readLine());
    }
}
```

## Output

```
C:\Achin Jain\File>javac console_read_string.java
C:\Achin Jain\File>java console_read_string
Enter String
Achin Jain
Achin Jain
```

## Writing Console Output

Console output is most easily accomplished with **print()** and **println()**. These methods are defined by the class **PrintStream** which is the type of the object referenced by **System.out**. Even though **System.out** is a byte stream, using it for simple program output is still acceptable. Because **PrintStream** is an output stream derived from **OutputStream**, it also implements the low-level method **write()**. Thus, **write()** can be used to write to the console. The simplest form of **write()** defined by **PrintStream** is shown below:

```
import java.io.*;
public class WriteDemo {
    public static void main(String args[]) {
        int b;
        b = 'A';
        System.out.write(b);
        System.out.write('\n');
    }
}
```

## Output

```
C:\Achin Jain\File>javac WriteDemo.java
C:\Achin Jain\File>java WriteDemo
A
```

## Networking

The term network programming refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.

The java.net package provides support for the two common network protocols:

1. **TCP:** TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.
2. **UDP:** UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

## Example or URL Class

URL class represents a URL which is acronym for Uniform Resource Locator and points to a resource on the World Wide Web. There are different information that we can parse from a URL and in the example below methods defined in URL class of Java are used to fetch information from a test URL.

```
import java.io.*;
import java.net.*;

public class URLEDemo
{
    public static void main(String[] args)
    {
        try
        {
            URL url=new URL("http://www.achin_jain_test:80/index.html");
            System.out.println("Protocol: "+url.getProtocol());
            System.out.println("Host Name: "+url.getHost());
            System.out.println("Port Number: "+url.getPort());
            System.out.println("File Name: "+url.getFile());
        }
        catch(Exception e){System.out.println(e);}
    }
}
```



## Output

```
C:\Achin Jain\networking>javac URLDemo.java
C:\Achin Jain\networking>java URLDemo
Protocol: http
Host Name: www.achin_jain_test
Port Number: 80
File Name: /index.html
```

## URL Connection Class

URL Connection class represents a communication link between URL and an application.

This class can be used to read/write data to the specified resource referred by the URL.

```
import java.io.*;
import java.net.*;
public class read_from_URL
{
    public static void main(String[] args)
    {
        try
        {
            URL url=new URL("http://localhost/niec/wp-content/uploads/2013/10/URLConnectionClass.txt");
            URLConnection urlcon=url.openConnection();
            InputStream stream=urlcon.getInputStream();
            int i;
            while((i=stream.read())!=-1)
            {
                System.out.print((char)i);
            }
        }
        catch(Exception e){System.out.println(e);}
    }
}
```

## Output

```
C:\Achin Jain\networking>javac read_from_URL.java
C:\Achin Jain\networking>java read_from_URL
This is an example of URLConnectionClass
C:\Achin Jain\networking>
```

## Example of InetAddress Class

InetAddress class represents an IP address and provides methods to get the IP of any host name.

```
import java.io.*;
import java.net.*;

public class InetDemo
{
    public static void main(String[] args)
    {
        try
        {
            InetAddress ip=InetAddress.getByName("www.technokarak.com");
            System.out.println("Host Name: "+ip.getHostName());
            System.out.println("IP Address: "+ip.getHostAddress());
        }
        catch(Exception e){System.out.println(e);}
    }
}
```

## Output

```
C:\Achin Jain\networking>java InetDemo
Host Name: www.technokarak.com
IP Address: 141.101.117.136
C:\Achin Jain\networking>
```

## Example of Socket Programming

Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.

When the connection is made, the server creates a socket object on its end of the communication. The client and server can now communicate by writing to and reading from the socket.

The `java.net.Socket` class represents a socket, and the `java.net.ServerSocket` class provides a mechanism for the server program to listen for clients and establish connections with them.

The following steps occur when establishing a TCP connection between two computers using sockets:

1. The server instantiates a `ServerSocket` object, denoting which port number communication is to occur on.
2. The server invokes the `accept()` method of the `ServerSocket` class. This method waits until a client connects to the server on the given port.
3. After the server is waiting, a client instantiates a `Socket` object, specifying the server name and port number to connect to.
4. The constructor of the `Socket` class attempts to connect the client to the specified server and port number. If communication is established, the client now has a `Socket` object capable of communicating with the server.
5. On the server side, the `accept()` method returns a reference to a new socket on the server that is connected to the client's socket.

After the connections are established, communication can occur using I/O streams. Each socket has both an `OutputStream` and an `InputStream`. The client's `OutputStream` is

connected to the server's InputStream, and the client's InputStream is connected to the server's OutputStream.

TCP is a twoway communication protocol, so data can be sent across both streams at the same time. There are following usefull classes providing complete set of methods to implement sockets.

## Socket Server

```
import java.io.*;
import java.net.*;
public class MyServer
{
    public static void main(String[] args)
    {
        try
        {
            ServerSocket ss=new ServerSocket(6666);
            Socket s=ss.accept();//establishes connection
            DataInputStream dis=new DataInputStream(s.getInputStream());
            String str=(String)dis.readUTF();
            System.out.println("message= "+str);
            ss.close();
        }
        catch(Exception e){System.out.println(e);}
    }
}
```

## Socket Client

```
import java.io.*;
import java.net.*;
public class MyClient
{
    public static void main(String[] args)
    {
        try
        {
            Socket s=new Socket("localhost",6666);
            DataOutputStream dout=new DataOutputStream(s.getOutputStream());
            dout.writeUTF("Hello Server");
            dout.flush();
            dout.close();
            s.close();
        }
        catch(Exception e){System.out.println(e);}
    }
}
```

## Output

```

C:\Windows\system32\cmd.exe
D<document, 'script', 'facebook-jssdk'></script>
</div>
</body>
</html>
C:\Achin Jain\networking>javac read_from_URL.java
C:\Achin Jain\networking>java read_from_URL
This is an example of URLConnectionClass
C:\Achin Jain\networking>javac InetDemo.java
C:\Achin Jain\networking>java InetDemo
Host Name: www.technokarak.com
IP Address: 141.101.117.136
C:\Achin Jain\networking>javac MyServer.java
C:\Achin Jain\networking>javac MyClient.java
C:\Achin Jain\networking>java MyClient
java.net.ConnectException: Connection refused: connect
C:\Achin Jain\networking>java MyServer
message= Hello Server
C:\Achin Jain\networking>

C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation

C:\Users\achin>cd/
C:\>cd achin jain
C:\Achin Jain>cd networking
C:\Achin Jain\networking>java MyClient
C:\Achin Jain\networking>
    
```