Java I/O Tutorial

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.

Java performs I/O through **Streams**. A Stream is linked to a physical layer by java I/O system to make input and output operation in java. In general, a stream means continuous flow of data. Streams are clean way to deal with input/output without having every part of your code understand the physical.

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.

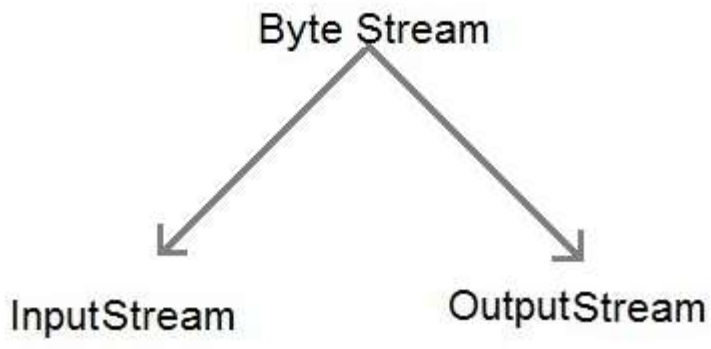
```
1. int i=System.in.read();//returns ASCII code of 1st character
2. System.out.println((char)i);//will print the character
```

Java encapsulates Stream under **java.io** package. Java defines two types of streams. They are,

1. **Byte Stream :** It provides a convenient means for handling input and output of byte.
2. **Character Stream :** It provides a convenient means for handling input and output of characters. Character stream uses Unicode and therefore can be internationalized.

Byte Stream Classes

Byte stream is defined by using two abstract class at the top of hierarchy, they are InputStream and OutputStream.



These two abstract classes have several concrete classes that handle various devices such as disk files, network connection etc.

Some important Byte stream classes.

Stream class	Description
BufferedInputStream	Used for Buffered Input Stream.
BufferedOutputStream	Used for Buffered Output Stream.
DataInputStream	Contains method for reading java standard datatype
DataOutputStream	An output stream that contain method for writing java standard data type
FileInputStream	Input stream that reads from a file
FileOutputStream	Output stream that write to a file.

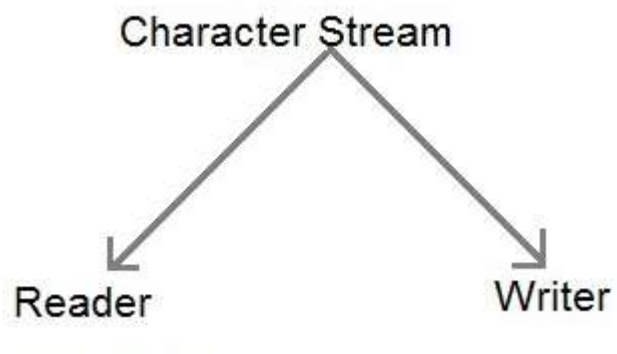
InputStream	Abstract class that describe stream input.
OutputStream	Abstract class that describe stream output.
PrintStream	Output Stream that contain `print()` and `println()` method

These classes define several key methods. Two most important are

1. `read()` : reads byte of data.
2. `write()` : Writes byte of data.

Character Stream Classes

Character stream is also defined by using two abstract class at the top of hierarchy, they are Reader and Writer.



These two abstract classes have several concrete classes that handle unicode character.

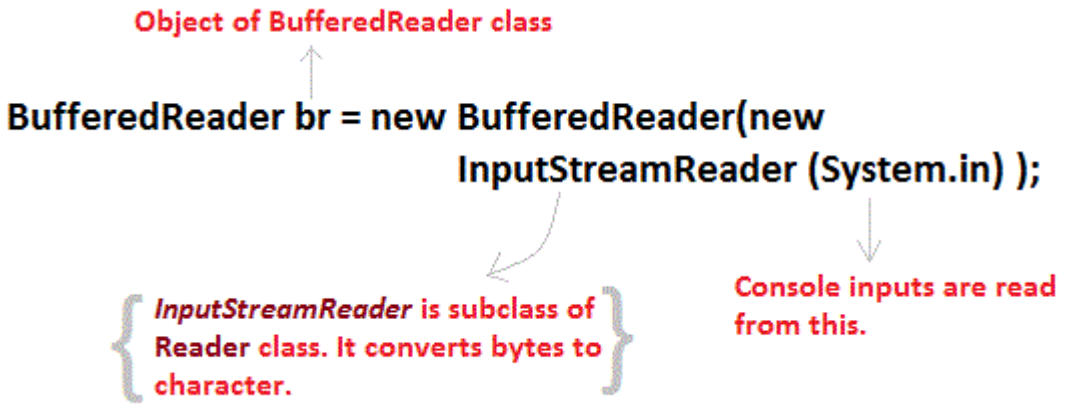
Some important Character stream classes.

Stream class	Description
BufferedReader	Handles buffered input stream.
BufferedWriter	Handles buffered output stream.

FileReader	Input stream that reads from file.
FileWriter	Output stream that writes to file.
InputStreamReader	Input stream that translate byte to character
OutputStreamReader	Output stream that translate character to byte.
PrintWriter	Output Stream that contain `print()` and `println()` method.
Reader	Abstract class that define character stream input
Writer	Abstract class that define character stream output

Reading Console Input

We use the object of BufferedReader class to take inputs from the keyboard.



Reading Characters

`read()` method is used with BufferedReader object to read characters. As this function returns integer type value, we need to use typecasting to convert it into **char** type.

```
int read() throws IOException
```

Below is a simple example explaining character input.

```
class CharRead
{
 public static void main( String args[])
 {
  BufferedReader br = new Bufferedreader(new InputstreamReader(System.in));
  char c = (char)br.read();        //Reading character
 }
}
```

Reading Strings

To read string we have to use `readLine()` function with BufferedReader class's object.

```
String readLine() throws IOException
```

Program to take String input from Keyboard in Java

```
import java.io.*;
class MyInput
{
 public static void main(String[] args)
 {
  String text;
  InputStreamReader isr = new InputStreamReader(System.in);
  BufferedReader br = new BufferedReader(isr);
  text = br.readLine();           //Reading String
  System.out.println(text);
 }
}
```

Program to read from a file using BufferedReader class

```
import java. Io *;
class ReadTest
{
 public static void main(String[] args)
 {
  try
```

```
  {
   File fl = new File("d:/myfile.txt");
   BufferedReader br = new BufferedReader(new FileReader(fl)) ;
   String str;
   while ((str=br.readLine())!=null)
   {
    System.out.println(str);
   }
   br.close();
   fl.close();
  }
  catch (IOException  e)
  { e.printStackTrace(); }
 }
}
```

Program to write to a File using FileWriter class

```
import java. Io *;
class WriteTest
{
 public static void main(String[] args)
 {
  try
  {
   File fl = new File("d:/myfile.txt");
   String str="Write this string to my file";
   FileWriter fw = new FileWriter(fl) ;
   fw.write(str);
   fw.close();
   fl.close();
  }
  catch (IOException  e)
  { e.printStackTrace(); }
 }
}
```

OutputStream vs InputStream

The explanation of OutputStream and InputStream classes are given below:

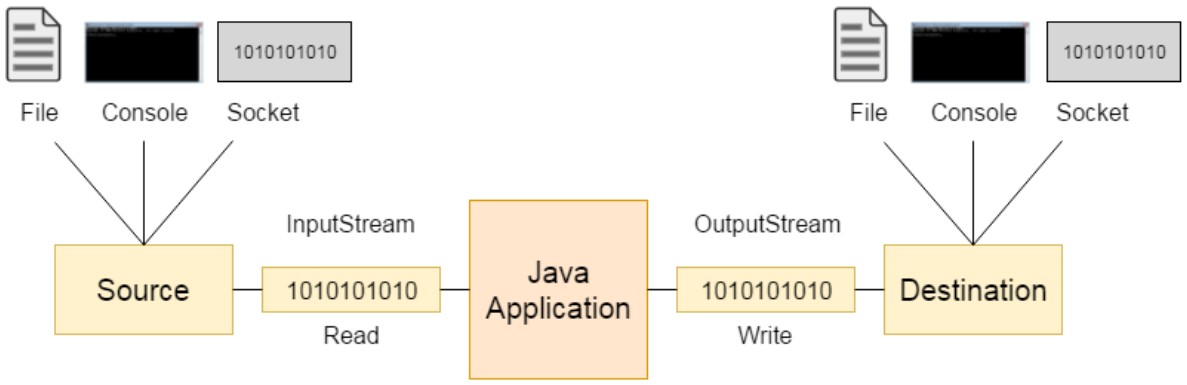
OutputStream

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

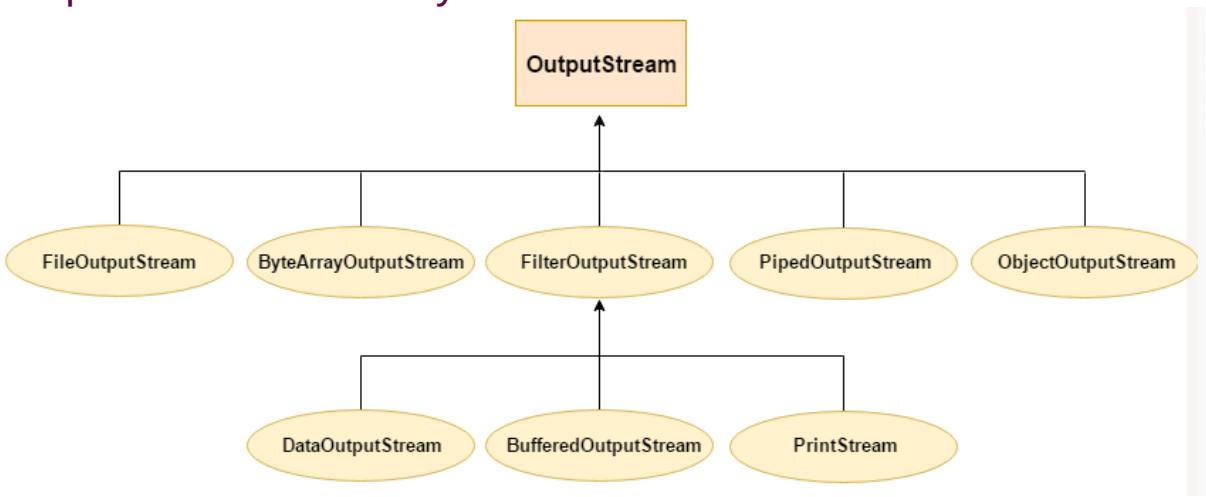
OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

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



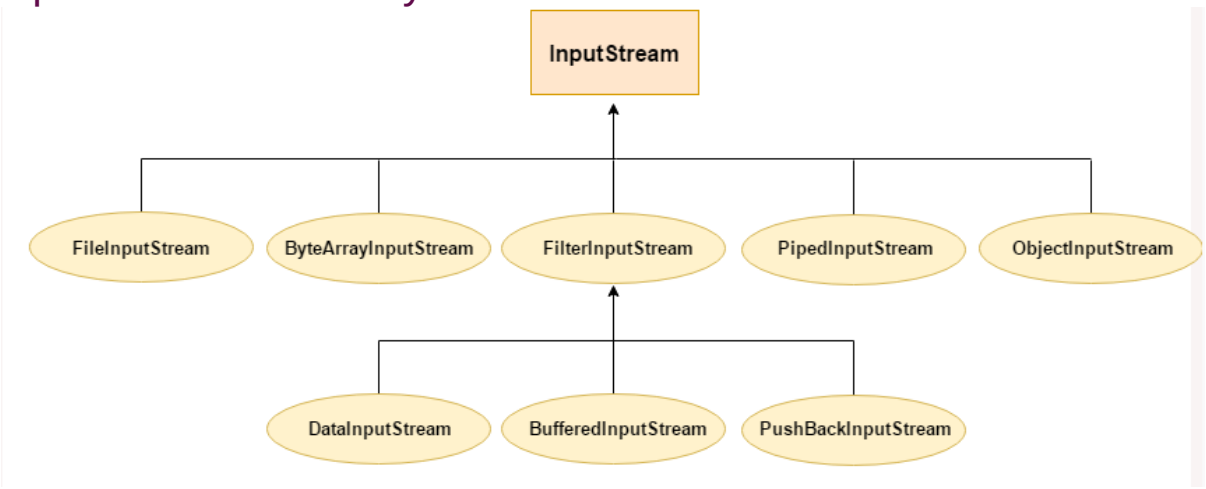
InputStream class

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 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

```
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...
```

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

testout.txt

```
Welcome to javaTpoint.
```

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

```
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 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:

```
1. 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.

```
1. package com.javatpoint;
2. import java.io.*;
3. public class BufferedOutputStreamExample{
4. public static void main(String args[])throws Exception{
5.     FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
6.     BufferedOutputStream bout=new BufferedOutputStream(fout);
7.     String s="Welcome to javaTpoint.";
8.     byte b[]=s.getBytes();
9.     bout.write(b);
10.    bout.flush();
11.    bout.close();
12.    fout.close();
13.    System.out.println("success");
14. }
15. }
```

Output:

```
Success
```

testout.txt

```
Welcome to javaTpoint.
```

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:

```
1.  package com.javatpoint;
2.
3.  import java.io.*;
4.  public class BufferedInputStreamExample{
5.   public static void main(String args[]){
6.    try{
7.      FileInputStream fin=new FileInputStream("D:\\testout.txt");
8.      BufferedInputStream bin=new BufferedInputStream(fin);
9.      int i;
10.     while((i=bin.read())!=-1){
11.      System.out.print((char)i);
12.     }
13.     bin.close();
14.     fin.close();
15.   }catch(Exception e){System.out.println(e);}
16. }
17. }
```

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

```
javaTpoint
```

Output:

```
javaTpoint
```

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.io.DataOutputStream in Java

A data output stream lets an application write primitive Java data types to an output stream in a portable way. An application can then use a data input stream to read the data back in.

Implementations of some of the important methods
Program:

```
//Java program to demonstrate DataOutputStream


import java.io.*;
class DataOutputStreamDemo
{
    public static void main(String args[]) throws IOException
    {
        //writing the data using DataOutputStream
        try ( DataOutputStream dout =
                    new DataOutputStream(new FileOutputStream("file.dat")) )
        {
            dout.writeDouble(1.1);
            dout.writeInt(55);
            dout.writeBoolean(true);
            dout.writeChar('4');
        }
        catch(FileNotFoundException ex)
        {
            System.out.println("Cannot Open the Output File");
            return;
        }

        // reading the data back using DataInputStream
        try ( DataInputStream din =
                    new DataInputStream(new FileInputStream("file.dat")) )
        {
            double a = din.readDouble();
            int b = din.readInt();
            boolean c = din.readBoolean();
            char d=din.readChar();
```

```
            System.out.println("Values: " + a + " " + b + " " + c+" " + d);
        }
        catch(FileNotFoundException e)
        {
            System.out.println("Cannot Open the Input File");
            return;
        }
    }
}
```

Output:

```
Values: 1.1 55 true 4
```

Important Points:

- DataOutputStream and DataInputStream are often used together.
- When a DataOutputStream is closed (by calling close( )), the underlying stream specified by out is also closed automatically.

Java Console Class

The Java Console class is be used to get input from console. It provides methods to read texts and passwords.

If you read password using Console class, it will not be displayed to the user.

The java.io.Console class is attached with system console internally. The Console class is introduced since 1.5.

Let's see a simple example to read text from console.

```
1. String text=System.console().readLine();
2. System.out.println("Text is: "+text);
```

Java Console class declaration

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

public final class Console **extends** Object **implements** Flushable

How to get the object of Console

System class provides a static method console() that returns the singleton instance of Console class.

```
1. public static Console console(){}
```

Let's see the code to get the instance of Console class.

```
1. Console c=System.console();
```

Java Console Example

```
1. import java.io.Console;
2. class ReadStringTest{
3. public static void main(String args[]){
4. Console c=System.console();
5. System.out.println("Enter your name: ");
6. String n=c.readLine();
7. System.out.println("Welcome "+n);
8. }
9. }
```

Output

```
Enter your name: Nakul Jain
Welcome Nakul Jain
```

Java Console Example to read password

```
1. import java.io.Console;
2. class ReadPasswordTest{
3. public static void main(String args[]){
4. Console c=System.console();
5. System.out.println("Enter password: ");
6. char[] ch=c.readPassword();
7. String pass=String.valueOf(ch);//converting char array into string
8. System.out.println("Password is: "+pass);
9. }
10. }
```

Output

```
Enter password:
Password is: 123
```

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.

```
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 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.

```
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.
```

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.

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

Output:

```
success
```

testout.txt:

```
Welcome to javaTpoint.
```

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.

```
1. package com.javatpoint;
2. import java.io.*;
3. public class BufferedReaderExample {
4.     public static void main(String args[])throws Exception{
5.         FileReader fr=new FileReader("D:\\testout.txt");
6.         BufferedReader br=new BufferedReader(fr);
```

```
7.
8.          int i;
9.          while((i=br.read())!=-1){
10.         System.out.print((char)i);
11.         }
12.         br.close();
13.         fr.close();
14.     }
15. }
```

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

```
Welcome to javaTpoint.
```

Output:

```
Welcome to javaTpoint.
```

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.

```
1. package com.javatpoint;
2. import java.io.*;
3. public class BufferedReaderExample{
4. public static void main(String args[])throws Exception{
5.     InputStreamReader r=new InputStreamReader(System.in);
6.     BufferedReader br=new BufferedReader(r);
7.     System.out.println("Enter your name");
8.     String name=br.readLine();
9.     System.out.println("Welcome "+name);
10. }
11. }
```

Output:

```
Enter your name
Nakul Jain
Welcome Nakul Jain
```

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.

```
1. package com.javatpoint;
2. import java.io.*;
3. public class BufferedReaderExample{
4. public static void main(String args[])throws Exception{
5.      InputStreamReader r=new InputStreamReader(System.in);
6.      BufferedReader br=new BufferedReader(r);
7.      String name="";
8.      while(!name.equals("stop")){
9.       System.out.println("Enter data: ");
10.      name=br.readLine();
11.      System.out.println("data is: "+name);
12.     }
13.    br.close();
14.    r.close();
15.    }
16.    }
```

Output:

```
Enter data: Nakul
data is: Nakul
Enter data: 12
data is: 12
Enter data: stop
data is: stop
```

Java - RandomAccessFile

This class is used for reading and writing to random access file. A random access file behaves like a large array of bytes. There is a cursor implied to the array called file pointer, by moving the cursor we do the read write operations. If end-of-file is reached before the desired number of byte has been read than EOFException is thrown. It is a type of IOException.

Example

```
1. import java.io.IOException;
2. import java.io.RandomAccessFile;
3.
4. public class RandomAccessFileExample {
5.     static final String FILEPATH ="myFile.TXT";
```

```
6.      public static void main(String[] args) {
7.          try {
8.              System.out.println(new String(readFromFile(FILEPATH, 0, 18)));
9.              writeToFile(FILEPATH, "I love my country and my people", 31);
10.         } catch (IOException e) {
11.             e.printStackTrace();
12.         }
13.     }
14.     private static byte[] readFromFile(String filePath, int position, int size)
15.             throws IOException {
16.         RandomAccessFile file = new RandomAccessFile(filePath, "r");
17.         file.seek(position);
18.         byte[] bytes = new byte[size];
19.         file.read(bytes);
20.         file.close();
21.         return bytes;
22.     }
23.     private static void writeToFile(String filePath, String data, int position)
24.             throws IOException {
25.         RandomAccessFile file = new RandomAccessFile(filePath, "rw");
26.         file.seek(position);
27.         file.write(data.getBytes());
28.         file.close();
29.     }
30. }
```

The myFile.TXT contains text "This class is used for reading and writing to random access file."

after running the program it will contain

This class is used for reading I love my country and my people