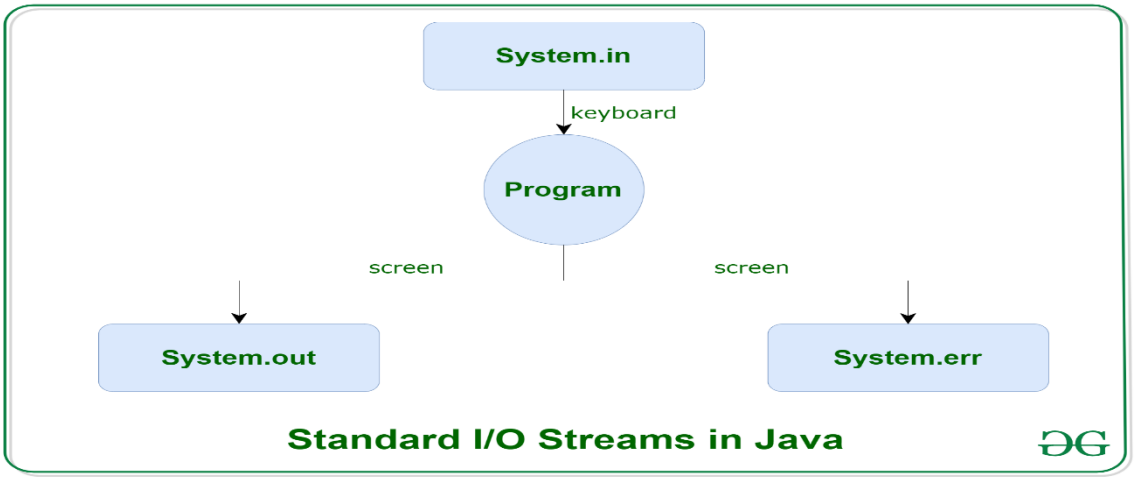
CONTENT

- INPUT OUTPUT STREAM
- BUFFERED STREAM
- STREAM FILTERS
- DATA INPUT OUTPUT STREAM
- PRINT STREAM
- RANDOM ACCESS FILE

Input/ Output Streams in JAVA

Java brings various Streams with its I/O package that helps the user to perform all the input-output operations. These streams support all the types of objects, data-types, characters, files etc to fully execute the I/O operations.



1. System.in: This is the **standard input stream** that is used to read characters from the keyboard or any other standard input device.
2. System.out: This is the **standard output stream** that is used to produce the result of a program on an output device like the computer screen.

Here is a list of the various print functions that we use to output statements:

- print(): This method in Java is used to display a text on the console. This text is passed as the parameter to this method in the form of String. This method prints the text on the console and the cursor remains at the end of the text at the console. The next printing takes place from just here.
  Syntax:

System.out.print(*parameter*);

Example:

```
// Java code to illustrate print()
import java.io.*;
  class Demo_print {
    public static void main(String[] args)
    {     // using print()
        // all are printed in the
        // same line
    System.out.print("IITM ");
        System.out.print("IITM! ");
        System.out.print("IITM ");
    }
}
```

Output:

IITM IITM IITM

- println(): This method in Java is also used to display a text on the console. It prints the text on the console and the cursor moves to the start of the next line at the console. The next printing takes place from the next line.
  Syntax:

System.out.println(*parameter*);

Example:

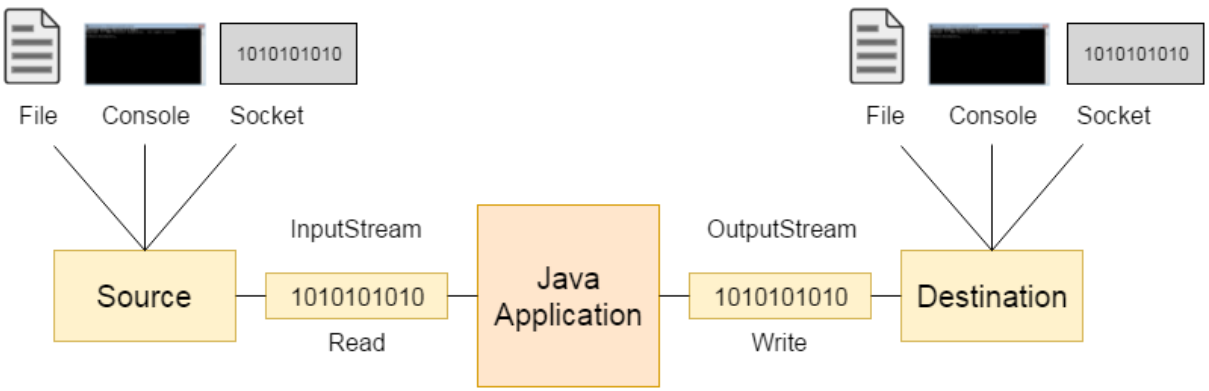
```
// Java code to illustrate println()
 import java.io.*;
class Demo_print {
    public static void main(String[] args)
```

```
    {  // using println()
        // all are printed in the
        // different line
        System.out.println("GfG! ");
        System.out.println("GfG! ");
        System.out.println("GfG! ");
    }
}
```

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.



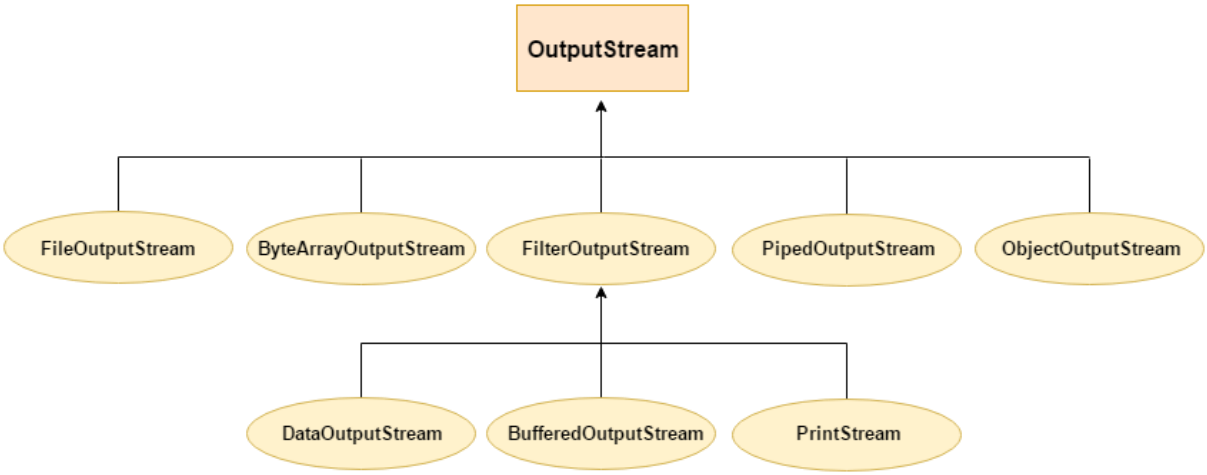
OutputStream class

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

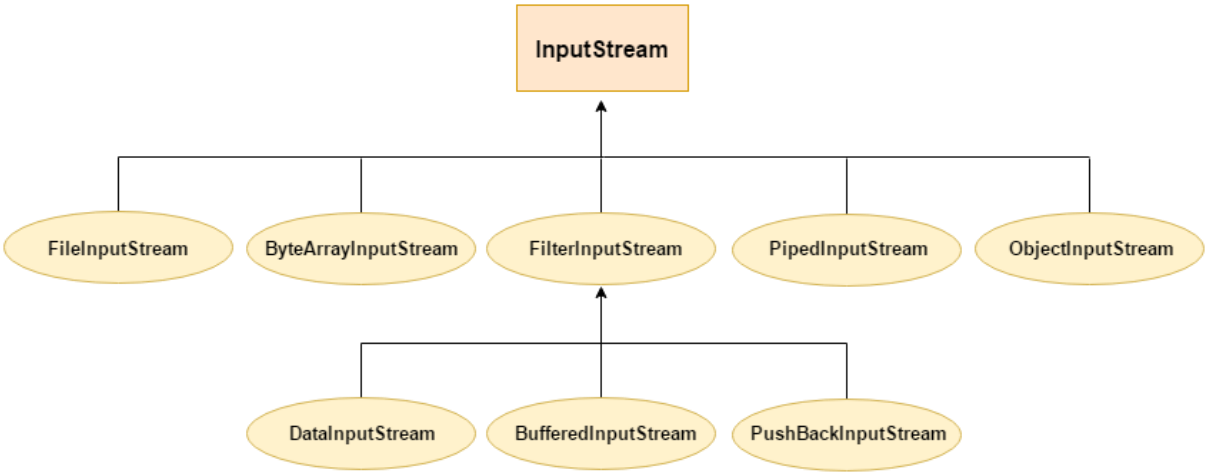
InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes.

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

declaration for Java.io.FileOutputStream class:

```
1. public class FileOutputStream extends OutputStream
```

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.

```
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...
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 IITM.";
         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);}
    }
}
```

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:

```
1. public class FileInputStream extends InputStream
```

Java FileInputStream class methods

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.

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 iitm
```

```
package com.IITM;

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 IITM
```

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:

1. public class BufferedOutputStream extends FilterOutputStream

Java BufferedOutputStream class constructors

Constructor	Description
BufferedOutputStream(OutputStream os)	It creates the new buffered output stream which is used for writing the data to the specified output stream.
BufferedOutputStream(OutputStream os, int size)	It creates the new buffered output stream which is used for writing the data to the specified output stream with a specified buffer size.

Java BufferedOutputStream class methods

Method	Description
void write(int b)	It writes the specified byte to the buffered output stream.
void write(byte[] b, int off, int len)	It write the bytes from the specified byte-input stream into a specified byte array, starting with the given offset
void flush()	It flushes the buffered output stream.

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.

```
package com.iitm;
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 iitm.";
        byte b[]=s.getBytes();
        bout.write(b);
        bout.flush();
        bout.close();
        fout.close();
        System.out.println("success");
    }
    }
```

Output:

```
Success
```

testout.txt

```
Welcome to iitm
```

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:

1. `public class BufferedInputStream extends FilterInputStream`

Java BufferedInputStream class constructors

Constructor	Description
BufferedInputStream(InputStream IS)	It creates the BufferedInputStream and saves it argument, the input stream IS, for later use.
BufferedInputStream(InputStream IS, int size)	It creates the BufferedInputStream with a specified buffer size and saves it argument, the input stream IS, for later use.

Java BufferedInputStream class methods

Method	Description
int available()	It returns an estimate number of bytes that can be read from the input stream without blocking by the next invocation method for the input stream.
int read()	It read the next byte of data from the input stream.
int read(byte[] b, int off, int ln)	It read the bytes from the specified byte-input stream into a specified byte array, starting with the given offset.
void close()	It closes the input stream and releases any of the system resources associated with the stream.
void reset()	It repositions the stream at a position the mark method was last called on this input stream.
void mark(int readlimit)	It sees the general contract of the mark method for the input stream.
long skip(long x)	It skips over and discards x bytes of data from the input stream.

boolean markSupported()	

```
1. package com.iitm;
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:

```
iitm
```

Output:

```
iitm
```

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:

1. **public class** DataOutputStream **extends** FilterOutputStream **implements** DataO utput

   Java DataOutputStream class methods

Method	Description
int size()	It is used to return the number of bytes written to the data output stream.
void write(int b)	It is used to write the specified byte to the underlying output stream.
void write(byte[] b, int off, int len)	It is used to write len bytes of data to the output stream.
void writeBoolean(boolean v)	It is used to write Boolean to the output stream as a 1-byte value.
void writeChar(int v)	It is used to write char to the output stream as a 2-byte value.
void writeChars(String s)	It is used to write string to the output stream as a sequence of characters.
void writeByte(int v)	It is used to write a byte to the output stream as a 1-byte value.
void writeBytes(String s)	It is used to write string to the output stream as a sequence of bytes.
void writeInt(int v)	It is used to write an int to the output stream
void writeShort(int v)	It is used to write a short to the output stream.

void writeShort(int v)	It is used to write a short to the output stream.
void writeLong(long v)	It is used to write a long to the output stream.
void writeUTF(String str)	It is used to write a string to the output stream using UTF-8 encoding in portable manner.
void flush()	It is used to flushes the data output stream.

```
1. package com.IITM;
2.
3. import java.io.*;
4. public class OutputExample {
5.     public static void main(String[] args) throws IOException {
6.         FileOutputStream file = new FileOutputStream(D:\\testout.txt);
7.         DataOutputStream data = new DataOutputStream(file);
8.         data.writeInt(65);
9.         data.flush();
10.        data.close();
11.        System.out.println("Succcess...");
12.    }
13.}
```

Output:

```
Succcess...
```

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:

1. **public class** DataInputStream **extends** FilterInputStream **implements** DataInpu
   t

   Java DataInputStream class Methods

Method	Description
int read(byte[] b)	It is used to read the number of bytes from the input stream.
int read(byte[] b, int off, int len)	It is used to read **len** bytes of data from the input stream.
int readInt()	It is used to read input bytes and return an int value.
byte readByte()	It is used to read and return the one input byte.
char readChar()	It is used to read two input bytes and returns a char value.
double readDouble()	It is used to read eight input bytes and returns a double value.
boolean readBoolean()	It is used to read one input byte and return true if byte is non zero, false if byte is zero.
int skipBytes(int x)	It is used to skip over x bytes of data from the input stream.
String readUTF()	It is used to read a string that has been encoded using the UTF-8 format.
void readFully(byte[] b)	It is used to read bytes from the input stream and store them into the buffer array.

void readFully(byte[] b, int off, int len)	It is used to read **len** bytes from the input stream.

```
1. package com.IITM;
2. import java.io.*;
3. public class DataStreamExample {
4.   public static void main(String[] args) throws IOException {
5.     InputStream input = new FileInputStream("D:\\testout.txt");
6.     DataInputStream inst = new DataInputStream(input);
7.     int count = input.available();
8.     byte[] ary = new byte[count];
9.     inst.read(ary);
10.    for (byte bt : ary) {
11.      char k = (char) bt;
12.      System.out.print(k+"-");
13.    }
14.  }
15.}
```

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

```
JAVA
```

Output:

```
J-A-V-A
```

Java FilterOutputStream Class

Java FilterOutputStream class implements the OutputStream class. It provides different sub classes such as BufferedOutputStream and DataOutputStream to provide additional functionality. So it is less used individually.

Java FilterOutputStream class declaration

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

```
1. public class FilterOutputStream extends OutputStream
```

Java FilterOutputStream class Methods

Method	Description
void write(int b)	It is used to write the specified byte to the output strea
void write(byte[] ary)	It is used to write ary.length byte to the output stream.
void write(byte[] b, int off, int len)	It is used to write len bytes from the offset off to the stream.
void flush()	It is used to flushes the output stream.
void close()	It is used to close the output stream.

Example of FilterOutputStream class

```
1. import java.io.*;
2. public class FilterExample {
3.     public static void main(String[] args) throws IOException {
4.         File data = new File("D:\\testout.txt");
5.         FileOutputStream file = new FileOutputStream(data);
6.         FilterOutputStream filter = new FilterOutputStream(file);
7.         String s="Welcome to IITM.";
8.         byte b[]=s.getBytes();
9.         filter.write(b);
10.        filter.flush();
11.        filter.close();
12.        file.close();
13.        System.out.println("Success...");
14.    }
15.}
16.
```

Output:

```
Success...
```

testout.txt

Welcome to IITM.

Java FilterInputStream Class

Java FilterInputStream class implements the InputStream. It contains different sub classes as BufferedInputStream, DataInputStream for providing additional functionality. So it is less used individually.

Java FilterInputStream class declaration

Let's see the declaration for java.io.FilterInputStream class

1. **public class** FilterInputStream **extends** InputStream

Java FilterInputStream class Methods

Method	Description
int available()	It is used to return an estimate number of bytes that can be read from the input stream.
int read()	It is used to read the next byte of data from the input stream.
int read(byte[] b)	It is used to read up to byte.length bytes of data from the input stream.
long skip(long n)	It is used to skip over and discards n bytes of data from the input stream.
boolean markSupported()	It is used to test if the input stream support mark and reset method.
void mark(int readlimit)	It is used to mark the current position in the input stream.
void reset()	It is used to reset the input stream.

void close()	It is used to close the input stream.

Example of FilterInputStream class

```
import java.io.*;
public class FilterExample {
   public static void main(String[] args) throws IOException {
      File data = new File("D:\\testout.txt");
      FileInputStream  file = new FileInputStream(data);
      FilterInputStream filter = new BufferedInputStream(file);
      int k =0;
      while((k=filter.read())!=-1){
         System.out.print((char)k);
      }
      file.close();
      filter.close();
   }
}
```

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

```
Welcome to IITM
```

Java PrintStream Class

The PrintStream class provides methods to write data to another stream. The PrintStream class automatically flushes the data so there is no need to call flush() method. Moreover, its methods don't throw IOException.

Class declaration

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

1. **public class** PrintStream **extends** FilterOutputStream **implements** Closeable. Ap
   pendable

Methods of PrintStream class

Method	Description
void print(boolean b)	It prints the specified boolean value.
void print(char c)	It prints the specified char value.
void print(char[] c)	It prints the specified character array values.
void print(int i)	It prints the specified int value.
void print(long l)	It prints the specified long value.
void print(float f)	It prints the specified float value.
void print(double d)	It prints the specified double value.
void print(String s)	It prints the specified string value.
void print(Object obj)	It prints the specified object value.
void println(boolean b)	It prints the specified boolean value and terminates the line.
void println(char c)	It prints the specified char value and terminates the line.
void println(char[] c)	It prints the specified character array values and terminates the line.
void println(int i)	It prints the specified int value and terminates the line.

void println(long l)	It prints the specified long value and terminates the line.
void println(float f)	It prints the specified float value and terminates the line.
void println(double d)	It prints the specified double value and terminates the line.
void println(String s)	It prints the specified string value and terminates the line.
void println(Object obj)	It prints the specified object value and terminates the line.
void println()	It terminates the line only.
void printf(Object format, Object... args)	It writes the formatted string to the current stream.
void printf(Locale l, Object format, Object... args)	It writes the formatted string to the current stream.
void format(Object format, Object... args)	It writes the formatted string to the current stream using specified format.
void format(Locale l, Object format, Object... args)	It writes the formatted string to the current stream using specified format.

Example of java PrintStream class

In this example, we are simply printing integer and string value.

```
package com.IITM;

import java.io.FileOutputStream;
```

```
import java.io.PrintStream;
public class PrintStreamTest{
 public static void main(String args[])throws Exception{
   FileOutputStream fout=new FileOutputStream("D:\\testout.txt ");
   PrintStream pout=new PrintStream(fout);
   pout.println(2016);
   pout.println("Hello Java");
   pout.println("Welcome to Java");
   pout.close();
   fout.close();
  System.out.println("Success?");
 }
}
```

Output

```
Success...
```

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

```
2016
Hello Java
Welcome to Java
```

Example of printf() method using java PrintStream class:

Let's see the simple example of printing integer value by format specifier using **printf()** method of **java.io.PrintStream** class.

```
1. class PrintStreamTest{
2.  public static void main(String args[]){
3.    int a=19;
4.    System.out.printf("%d",a); //Note: out is the object of printstream
5.  }
6. }
```

Output

```
19
```

Java PrintWriter class

Java PrintWriter class is the implementation of Writer class. It is used to print the formatted representation of objects to the text-output stream.

Class declaration

declaration for Java.io.PrintWriter class:

1. `public class PrintWriter extends Writer`

Methods of PrintWriter class

Method	Description
void println(boolean x)	It is used to print the boolean value.
void println(char[] x)	It is used to print an array of characters.
void println(int x)	It is used to print an integer.
PrintWriter append(char c)	It is used to append the specified character to the writer.
PrintWriter append(CharSequence ch)	It is used to append the specified character sequence to the writer.
PrintWriter append(CharSequence ch, int start, int end)	It is used to append a subsequence of specified character to the writer.
boolean checkError()	It is used to flushes the stream and check its error state.

protected void setError()	It is used to indicate that an error occurs.
protected void clearError()	It is used to clear the error state of a stream.
PrintWriter format(String format, Object... args)	It is used to write a formatted string to the writer using specified arguments and format string.
void print(Object obj)	It is used to print an object.
void flush()	It is used to flushes the stream.
void close()	It is used to close the stream.

Java PrintWriter Example

Let's see the simple example of writing the data on a **console** and in a **text file testout.txt** using Java PrintWriter class.

```
package com.IITM;

import java.io.File;
import java.io.PrintWriter;
public class PrintWriterExample {
   public static void main(String[] args) throws Exception {
         //Data to write on Console using PrintWriter
     PrintWriter writer = new PrintWriter(System.out);
     writer.write("IITM provides tutorials of all technology.");
 writer.flush();
     writer.close();
//Data to write in File using PrintWriter
     PrintWriter writer1 =null;
       writer1 = new PrintWriter(new File("D:\\testout.txt"));
```

```
        writer1.write("Like Java, Spring, Hibernate, Android, PHP etc.");

                writer1.flush();
        writer1.close();
    }
}
```

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.

Constructor

Constructor	Description
RandomAccessFile(File file, String mode)	Creates a random access file stream to read from, and optionally to write to, the file specified by the File argument.
RandomAccessFile(String name, String mode)	Creates a random access file stream to read from, and optionally to write to, a file with the specified name.

Method

Modifier and Type	Method	Method
void	close()	It closes this random access file stream and releases any system resources associated with the stream.
FileChannel	getChannel()	It returns the unique FileChannel object associated with this file.
int	readInt()	It reads a signed 32-bit integer from this file.

String	readUTF()	It reads in a string from this file.
void	seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.
void	writeDouble(double v)	It converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the file as an eight-byte quantity, high byte first.
void	writeFloat(float v)	It converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the file as a four-byte quantity, high byte first.
void	write(int b)	It writes the specified byte to this file.
int	read()	It reads a byte of data from this file.
long	length()	It returns the length of this file.
void	seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.

Example

```
import java.io.IOException;
import java.io.RandomAccessFile;

public class RandomAccessFileExample {
static final String FILEPATH ="myFile.TXT";
public static void main(String[] args) {
try {
System.out.println(new String(readFromFile(FILEPATH, 0, 18)));
```

```
writeToFile(FILEPATH, "I love my country and my people", 31);
} catch (IOException e) {
e.printStackTrace();
}
}
private static byte[] readFromFile(String filePath, int position, int size)
        throws IOException {
    RandomAccessFile file = new RandomAccessFile(filePath, "r");
    file.seek(position);
    byte[] bytes = new byte[size];
    file.read(bytes);
    file.close();
    return bytes;
  }
  private static void writeToFile(String filePath, String data, int position)
        throws IOException {
    RandomAccessFile file = new RandomAccessFile(filePath, "rw");
    file.seek(position);
    file.write(data.getBytes());
    file.close();
  }
}
```

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