**Lecture Handouts (I/O Streams)**
**Course**: Web Engineering
**Instructor**: Wasim Ahmad Khan

# Input and Output in Java

**Input and Output (I/O)** is used to process the input and produce the output based on the input. Java uses the concept of stream to make I/O operations fast. **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's like a stream of water that continues to flow.

**Three streams are created for us automatically:**

- **1) System.out:** standard output stream
- **2) System.in:** standard input stream
- **3) System.err:** standard error

Do You Know ?

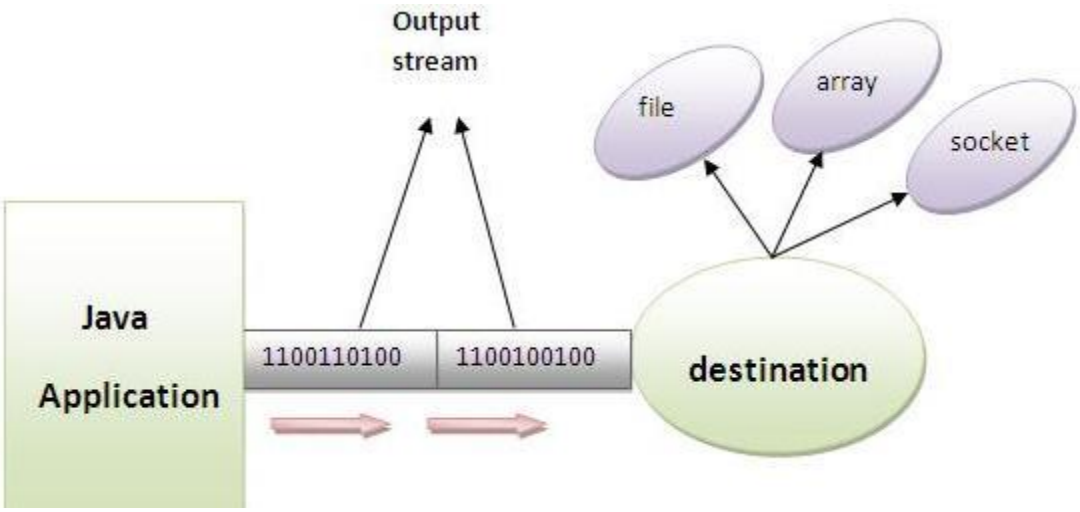
- How to write a common data to multiple files using single stream only ?
- How can we access multiple files by single stream ?
- How can we improve the performance of Input and Output operation ?
- How many ways can we read data from the keyboard?
- What is console class ?
- How to compress and uncompress the data of a file?

---

## OutputStream

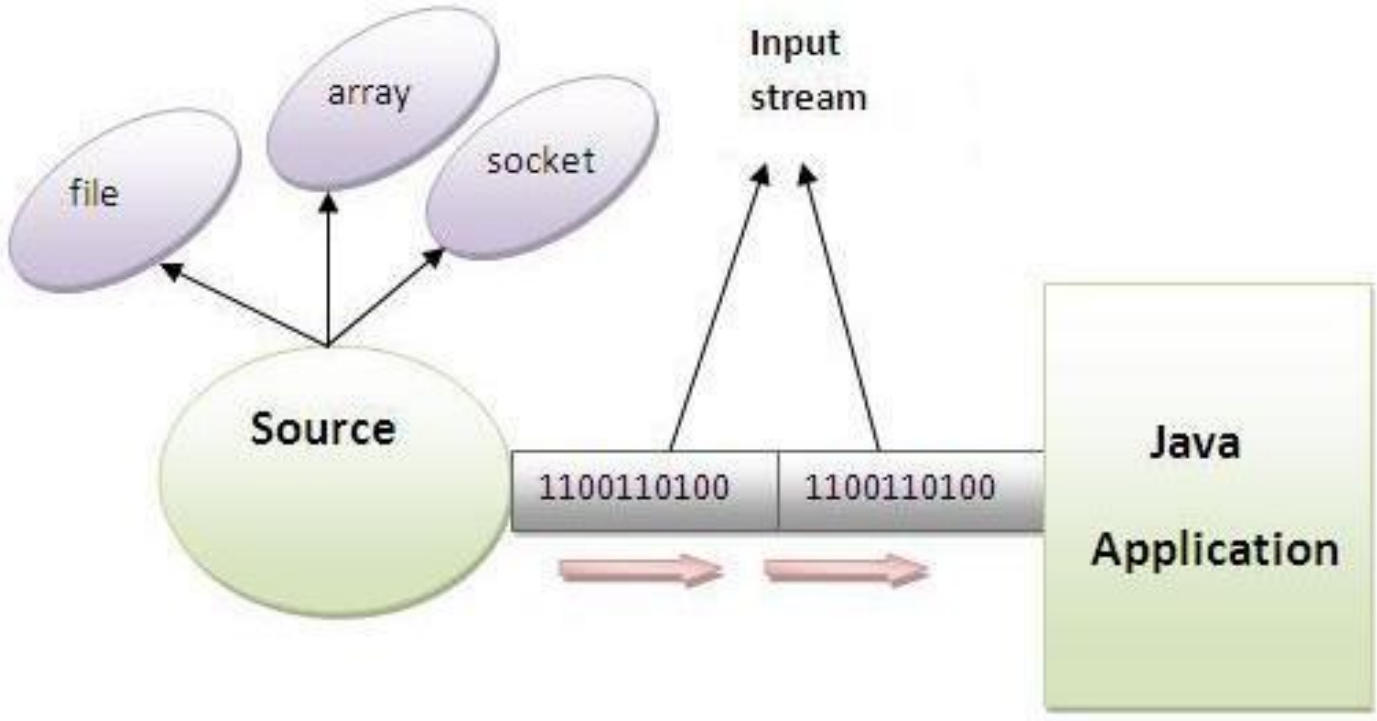
Java application uses an output stream to write data to a destination, it may be a file,an array,peripheral device or socket.

**Lecture Handouts (I/O Streams)**
**Course**: Web Engineering
**Instructor**: Wasim Ahmad Khan




## InputStream

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




---

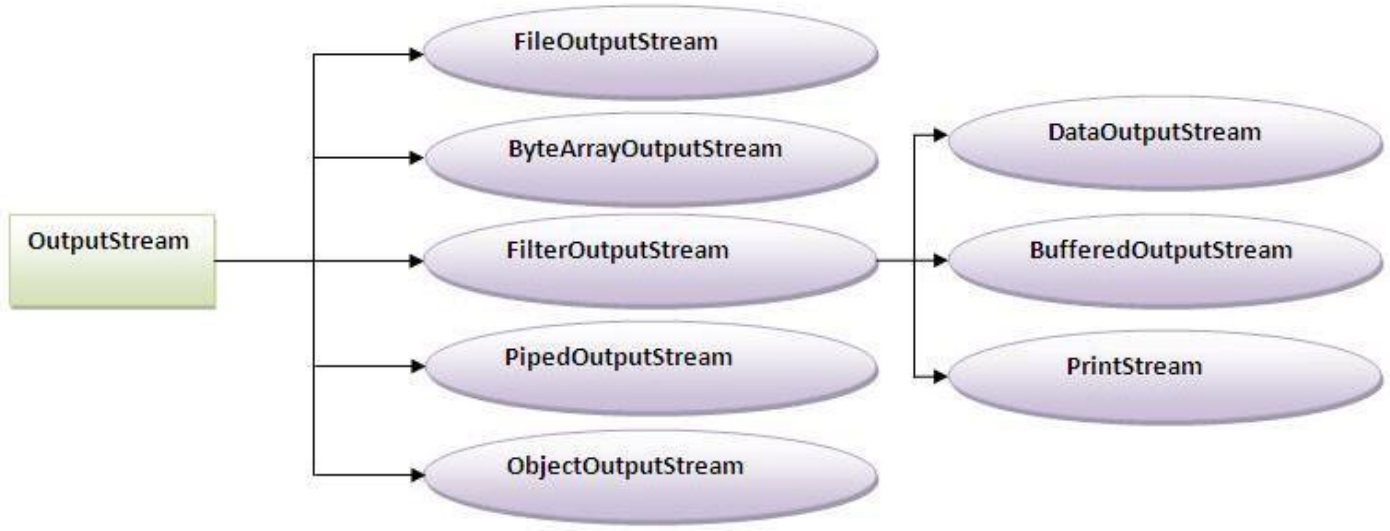
## OutputStream class

OutputStream class ia 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.

## Commonly used methods of OutputStream class

| 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. |




---

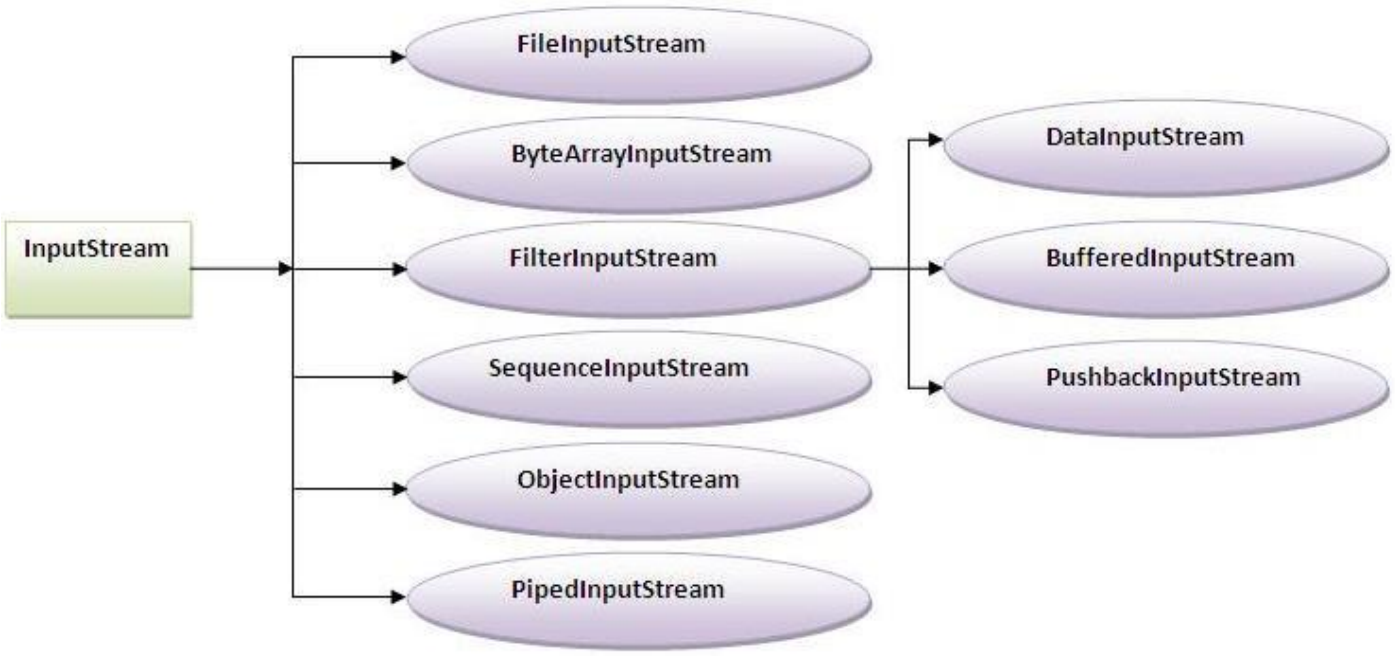
## InputStream class

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

## Commonly used methods of InputStream class

| 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 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. |

**Lecture Handouts (I/O Streams)**
**Course**: Web Engineering
**Instructor**: Wasim Ahmad Khan




# FileInputStream and FileOutputStream (File Handling):

FileInputStream and FileOutputStream classes are used to read and write data in file. In another words, they are used for file handling in java.

---

## FileOutputStream class:

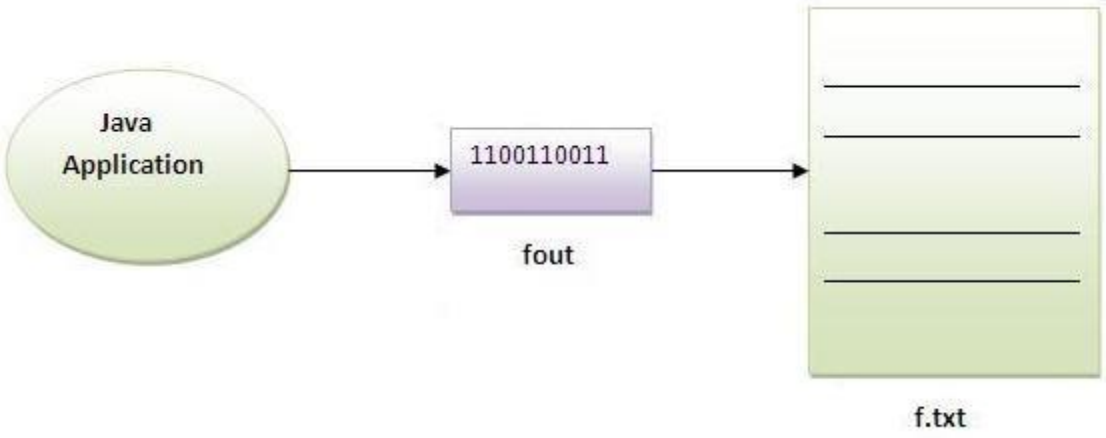
A FileOutputStream is an output stream for writing data to a file.
If you have to write primitive values then use FileOutputStream.Instead, for character-oriented data, prefer FileWriter.But you can write byte-oriented as well as character-oriented data.

## Example of FileOutputStream class:

```
//Simple program of writing data into the file


import java.io.*;
class Test{
  public static void main(String args[]){
   try{
     FileOutputstream fout=new FileOutputStream("abc.txt");
     String s="Sachin Tendulkar is my favourite player";

     byte b[]=s.getBytes();
```

```
      fout.write(b);

      fout.close();

      System.out.println("success...");
     }catch(Exception e){system.out.println(e);}
   }
}
```

**Output:**success...




---

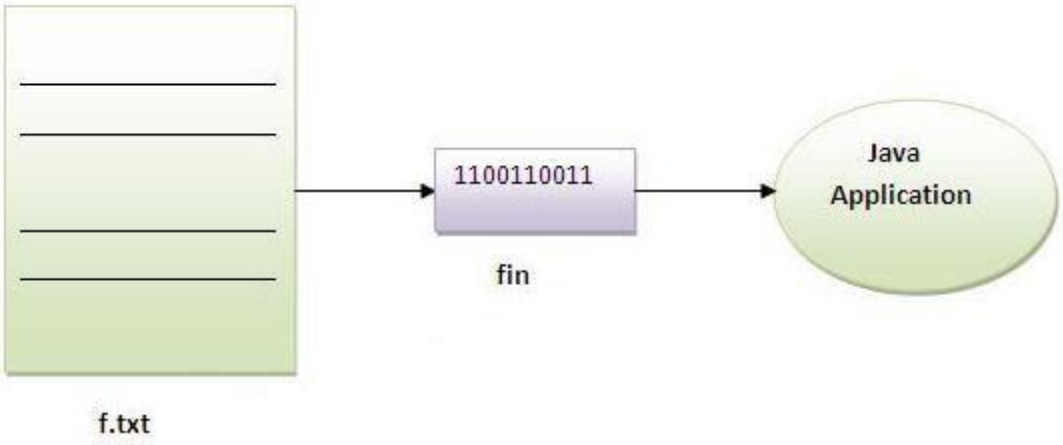
## FileInputStream class:

A FileInputStream obtains input bytes from a file.It is used for reading streams of raw bytes such as image data. For reading streams of characters, consider using FileReader.
It should be used to read byte-oriented data.For example, to read image etc.

## Example of FileInputStream class:

```
//Simple program of reading data from the file

import java.io.*;
class SimpleRead{
 public static void main(String args[]){
  try{
    FileInputStream fin=new FileInputStream("abc.txt");
    int i;
    while((i=fr.read())!=-1)
     System.out.println((char)i);

    fin.close();
  }catch(Exception e){system.out.println(e);}
 }
}
```

**Output:**Sachin is my favourite player.




---

### Example of Reading the data of current java file and writing it into another file

We can read the data of any file using the FileInputStream class whether it is java file, image file, video file etc. In this example, we are reading the data of C.java file and writing it into another file M.java.

```
import java.io.*;

class C{
public static void main(String args[])throws Exception{

FileInputStream fin=new FileInputStream("C.java");
FileOutputStream fout=new FileOutputStream("M.java");

int i=0;
while((i=fin.read())!=-1){
fout.write((byte)i);
}

fin.close();
}
}
```

# ByteArrayOutputStream class:

In this stream, the data is written into a byte array. The buffer automatically grows as data is written to it.
Closing a `ByteArrayOutputStream` has no effect.

### Commonly used Constructors of ByteArrayOutputStream class:

**1) ByteArrayOutputStream():**creates a new byte array output stream with the initial capacity of 32 bytes, though its size increases if necessary.

**2) ByteArrayOutputStream(int size):**creates a new byte array output stream, with a buffer capacity of the specified size, in bytes.

## Commonly used Methods of ByteArrayOutputStream class:

**1) public synchronized void writeTo(OutputStream out) throws IOException:** writes the complete contents of this byte array output stream to the specified output stream.

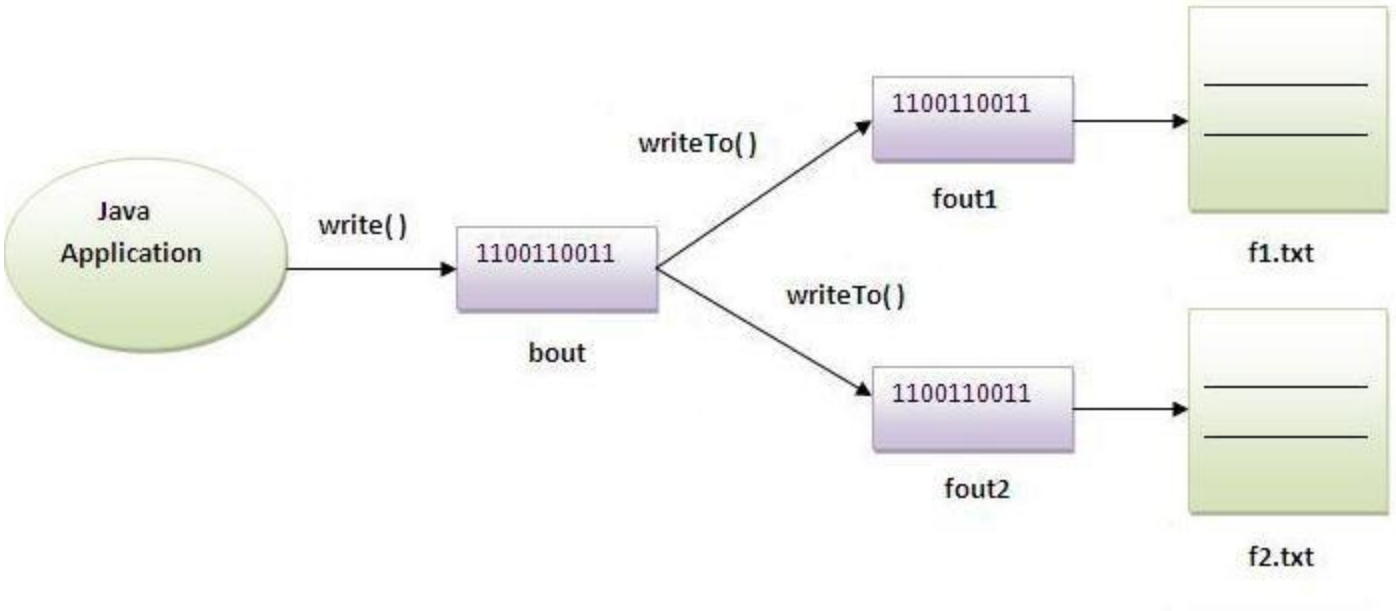
---

## Example of ByteArrayOutputStream class:

```
//Simple program of writing data by ByteArrayOutputStream class


import java.io.*;
class S{
 public static void main(String args[])throws Exception{

  FileOutputStream fout1=new FileOutputStream("f1.txt");
  FileOutputStream fout2=new FileOutputStream("f2.txt");

  ByteArrayOutputStream bout=new ByteArrayOutputStream();
  bout.write(239);

  bout.writeTo(fout1);
  bout.writeTo(fout2);

  bout.flush();

  bout.close();//has no effect
  System.out.println("success...");
 }
}
```

**Output:**success...

**Lecture Handouts (I/O Streams)**
**Course**: Web Engineering
**Instructor**: Wasim Ahmad Khan




# SequenceInputStream class:

SequenceInputStream class is used to read data from multipule streams.

### Constructors of SequenceInputStream class:

**1) SequenceInputStream(InputStream s1, InputStream s2):**creates a new input stream by reading the data of two input stream in order, first s1 and then s2.

**2) SequenceInputStream(Enumeration e):**creates a new input stream by reading the data of an enumeration whose type is InputStream.

---

### Simple example of SequenceInputStream class

In this example, we are printing the data of two files f1.txt and f2.txt.

```
//Program of SequenceInputStream that reads data of 2 files

import java.io.*;
class Simple{
  public static void main(String args[])throws Exception{

   FileinputStream fin1=new FileinputStream("f1.txt");
   FileinputStream fin2=new FileinputStream("f2.txt");

   SequenceinputStream sis=new SequenceinputStream(fin1,fin2);
   int i;
   while((i=sis.read())!=-1)
   {
```

```
    System.out.println((char)i);
   }
  }
}
```

---

## Example of SequenceInputStream class that reads the data from two files and write it into another

In this example, we are writing the data of two files f1.txt and f2.txt into another file named f3.txt.

```
//reading data of 2 files and writing it into one file

import java.io.*;
class Simple{
  public static void main(String args[])throws Exception{

   FileinputStream fin1=new FileinputStream("f1.txt");
   FileinputStream fin2=new FileinputStream("f2.txt");

   FileOutputStream fout=new FileOutputStream("f3.txt");

   SequenceinputStream sis=new SequenceinputStream(fin1,fin2);
   int i;
   while((i.sisread())!=-1)
   {
     fout.write(i);
   }
   sis.close();
   fout.close();
   fin.close();
   fin.close();

  }
}
```

---

## Example of SequenceInputStream class that reads the data from multiple files using enumeration

If we need to read the data from more than two files, we need to have these information in the Enumeration object. Enumeration object can be get by calling elements method of the Vector class. Let's see the simple example where we are reading the data from the 4 files.

```
import java.io.*;
import java.util.*;

class B{
public static void main(String args[])throws IOException{

//creating the FileInputStream objects for all the files
FileInputStream fin=new FileInputStream("A.java");
```

```
FileInputStream fin2=new FileInputStream("abc2.txt");
FileInputStream fin3=new FileInputStream("abc.txt");
FileInputStream fin4=new FileInputStream("B.java");

//creating Vector object to all the stream
Vector v=new Vector();
v.add(fin);
v.add(fin2);
v.add(fin3);
v.add(fin4);

//creating enumeration object by calling the elements method
Enumeration e=v.elements();

//passing the enumeration object in the constructor
SequenceInputStream bin=new SequenceInputStream(e);
int i=0;

while((i=bin.read())!=-1){
System.out.print((char)i);
}

bin.close();
fin.close();
fin2.close();
}
}
```

# BufferedOutputStream class:

BufferedOutputStream used an internal buffer. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.

---

### 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.*;
class Test{
 public static void main(String args[])throws Exception{

   FileOutputStream fout=new FileOutputStream("f1.txt");
   BufferedOutputStream bout=new BufferedOutputStream(fout);

   String s="Sachin is my favourite player";
   byte b[]=s.getBytes();
   bout.write(b);
```

```
   bout.flush();
   bout.close();
   System.out.println("success");
 }
}
```
**Output:**success...

---

### Example of BufferedInputStream class:

```
//Simple program of reading data from the file using buffer

import java.io.*;
class SimpleRead{
 public static void main(String args[]){
  try{

    FileInputStream fin=new FileInputStream("f1.txt");
    BufferedInputStream bin=new BufferedInputStream(fin);
    int i;
    while((i=bin.read())!=-1)
     System.out.println((char)i);

    fin.close();
  }catch(Exception e){system.out.println(e);}
 }
}
```
**Output:**Sachin is my favourite player

# FileWriter class:

FileWriter class is used to write character-oriented data to the file. Sun Microsystem has suggested not to use the FileInputStream and FileOutputStream classes if you have to read and write the textual information.

### Example of FileWriter class:

In this example, we are writing the data in the file abc.txt.

```
import java.io.*;
class Simple{
 public static void main(String args[]){
  try{
   FileWriter fw=new FileWriter("abc.txt");
   fw.write("my name is sachin");
   fw.flush();

   fw.close();
  }catch(Exception e){System.out.println(e);}
  System.out.println("success");
```

```
 }
}
```
**Output:**success...

---

## FileReader class:

FileReader class is used to read data from the file.

## Example of FileReader class:

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

```
import java.io.*;
class Simple{
 public static void main(String args[])throws Exception{

  FileReader fr=new FileReader("abc.txt");
  int i;
  while((i=fr.read())!=-1)
  System.out.println((char)i);

  fr.close();
 }
}
```
**Output:**my name is sachin

## CharArrayWriter class:

The CharArrayWriter class can be used to write data to multiple files. This class implements the Appendable interface. Its buffer automatically grows when data is written in this stream. Calling the close() method on this object has no effect.

## Example of CharArrayWriter class:

In this example, we are writing a common data to 4 files a.txt, b.txt, c.txt and d.txt.

```
import java.io.*;
class Simple{
 public static void main(String args[])throws Exception{

  CharArrayWriter out=new CharArrayWriter();
  out.write("my name is");

  FileWriter f1=new FileWriter("a.txt");
  FileWriter f2=new FileWriter("b.txt");
  FileWriter f3=new FileWriter("c.txt");
  FileWriter f4=new FileWriter("d.txt");

  out.writeTo(f1);
```

```
  out.writeTo(f2);
  out.writeTo(f3);
  out.writeTo(f4);


  f1.close();
  f2.close();
  f3.close();
  f4.close();
 }
}
```

# Reading data from keyboard:

There are many ways to read data from the keyboard. For example:

- InputStreamReader
- Console
- Scanner
- DataInputStream etc.

## InputStreamReader class:

InputStreamReader class can be used to read data from keyboard.It performs two tasks:

- connects to input stream of keyboard
- converts the byte-oriented stream into character-oriented stream

## BufferedReader class:

BufferedReader class can be used to read data line by line by readLine() method.

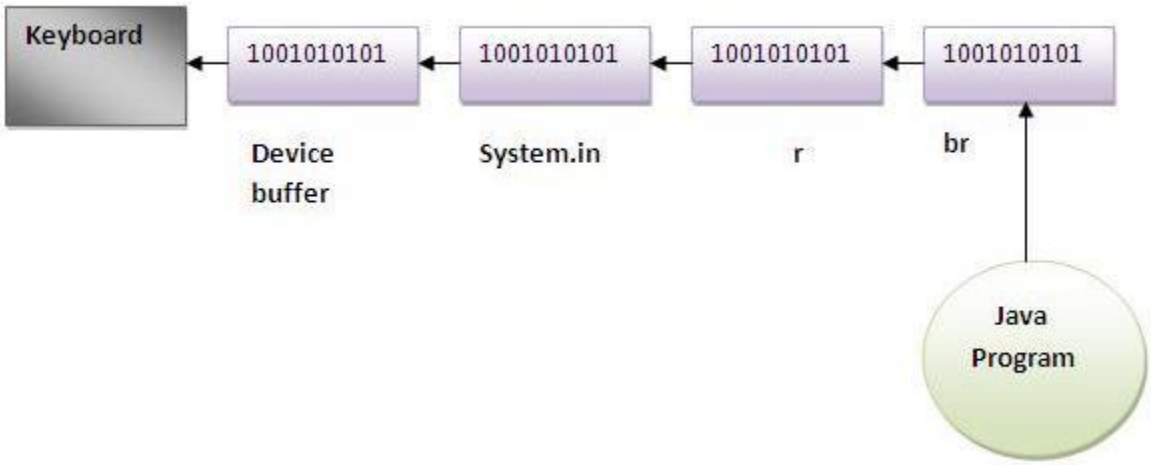
## Example of reading data from keyboard by InputStreamReader and BufferdReader class:

In this example, we are connecting the BufferedReader stream with the InputStreamReader stream for reading the line by line data from the keyboard.

```
//Program of reading data

import java.io.*;
class G5{
public static void main(String args[])throws Exception{

InputStreamReader r=new InputStreamReader(System.in);
BufferedReader br=new BufferedReader(r);
```

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

**Output:**
```
Enter ur name
Amit
Welcome Amit
```




---

## Another Example of reading data from keyboard by InputStreamReader and BufferdReader class until the user writes stop

In this example, we are reading and printing the data until the user prints stop.

```
import java.io.*;
class G5{
public static void main(String args[])throws Exception{

 InputStreamReader r=new InputStreamReader(System.in);
 BufferedReader br=new BufferedReader(r);

 String name="";

  while(name.equals("stop")){
   System.out.println("Enter data: ");
   name=br.readLine();
   System.out.println("data is: "+name);
  }

 br.close();
 r.close();
 }
}
```

**Output:**
```
Enter data: Amit
data is: Amit
Enter data: 10
```

```
data is: 10
Enter data: stop
data is: stop
```

# Console class (I/O)

The Console class can be used to get input from the keyboard.

---

### How to get the object of Console class?

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

### Syntax:

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

---

### Commonly used methods of Console class:

**1) public String readLine()**: is used to read a single line of text from the console.
**2) public String readLine(String fmt,Object... args)**: it provides a formatted prompt then reads the single line of text from the console.
**3) public char[] readPassword()**: is used to read password that is not being displayed on the console.
**4) public char[] readPassword(String fmt,Object... args)**: it provides a formatted prompt then reads the password that is not being displayed on the console.

### Example of Console class that reads name of user:

```
import java.io.*;
class A{
public static void main(String args[]){

Console c=System.console();

System.out.println("Enter ur name");
String n=c.readLine();
System.out.println("Welcome "+n);

}
}
```

---

### Example of Console class that reads password:

```
import java.io.*;
class A{
public static void main(String args[]){

Console c=System.console();

System.out.println("Enter password");
char[] ch=c.readPassword();

System.out.println("Password is");
for(char ch2:ch)
System.out.print(ch2);

}
}
```

# java.util.Scanner class:

There are various ways to read input from the keyboad, the java.util.Scanner class is one of them. The Scanner class breaks the input into tokens using a delimiter which is whitespace bydefault. It provides many methods to read and parse various primitive values.

### Commonly used methods of Scanner class:

There is a list of commonly used Scanner class methods:

- **public String next():** it returns the next token from the scanner.
- **public String nextLine():** it moves the scanner position to the next line and returns the value as a string.
- **public byte nextByte():** it scans the next token as a byte.
- **public short nextShort():** it scans the next token as a short value.
- **public int nextInt():** it scans the next token as an int value.
- **public long nextLong():** it scans the next token as a long value.
- **public float nextFloat():** it scans the next token as a float value.
- **public double nextDouble():** it scans the next token as a double value.

### Example of java.util.Scanner class:

Let's see the simple example of the Scanner class which reads the int, string and double value as an input:

```
import java.util.Scanner;
class ScannerTest{
 public static void main(String args[]){
```

```
   Scanner sc=new Scanner(System.in);

   System.out.println("Enter your rollno");
   int rollno=sc.nextInt();
   System.out.println("Enter your name");
   String name=sc.next();
   System.out.println("Enter your fee");
   double fee=sc.nextDouble();

   System.out.println("Rollno:"+rollno+" name:"+name+" fee:"+fee);

 }
}
```

download this scanner example

```
Output:Enter your rollno
       111
       Enter your name
       Ratan
       Enter
       450000
       Rollno:111 name:Ratan fee:450000
```

# java.io.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.

### Commonly used methods of PrintStream class:

There are many methods in PrintStream class. Let's see commonly used methods of PrintStream class:

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

terminates the line.

- **public void println(int i):** it prints the specified int value and terminates the line.
- **public void println(long l):** it prints the specified long value and terminates the line.
- **public void println(float f):** it prints the specified float value and terminates the line.
- **public void println(double d):** it prints the specified double value and terminates the line.
- **public void println(String s):** it prints the specified string value and terminates the line./li>
- **public void println(Object obj):** it prints the specified object value and terminates the line.
- **public void println():** it terminates the line only.
- **public void printf(Object format, Object... args):** it writes the formatted string to the current stream.
- **public void printf(Locale l, Object format, Object... args):** it writes the formatted string to the current stream.
- **public void format(Object format, Object... args):** it writes the formatted string to the current stream using specified format.
- **public void format(Locale l, Object format, Object... args):** it writes the formatted string to the current stream using specified format.

---

## Example of java.io.PrintStream class:

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

```
import java.io.*;
class PrintStreamTest{
 public static void main(String args[])throws Exception{

   FileOutputStream fout=new FileOutputStream("mfile.txt");
   PrintStream pout=new PrintStream(fout);
   pout.println(1900);
   pout.println("Hello Java");
   pout.println("Welcome to Java");
   pout.close();
   fout.close();

 }
}
```

download this PrintStream example

---

## Example of printf() method of java.io.PrintStream class:

Let's see the simple example of printing integer value by format specifier.

```
class PrintStreamTest{
 public static void main(String args[]){
```

```
   int a=10;
   System.out.printf("%d",a);//Note, out is the object of PrintStream class

 }
}
```

**Output:**10

# Compressing and Uncompressing File

The DeflaterOutputStream and InflaterInputStream classes provide mechanism to compress and uncompress the data in the **deflate compression format**.

---

## DeflaterOutputStream class:

The DeflaterOutputStream class is used to compress the data in the deflate compression format. It provides facility to the other compression filters, such as GZIPOutputStream.

## Example of Compressing file using DeflaterOutputStream class

In this example, we are reading data of a file and compressing it into another file using DeflaterOutputStream class. You can compress any file, here we are compressing the Deflater.java file.

```
import java.io.*;
import java.util.zip.*;

class Compress{
public static void main(String args[]){

try{
FileInputStream fin=new FileInputStream("Deflater.java");

FileOutputStream fout=new FileOutputStream("def.txt");
DeflaterOutputStream out=new DeflaterOutputStream(fout);

int i;
while((i=fin.read())!=-1){
out.write((byte)i);
out.flush();
}

fin.close();
out.close();

}catch(Exception e){System.out.println(e);}
System.out.println("rest of the code");
}
}
```

download this example

### InflaterInputStream class:

The InflaterInputStream class is used to uncompress the file in the deflate compression format. It provides facility to the other uncompression filters, such as GZIPInputStream class.

### Example of uncompressing file using InflaterInputStream class

In this example, we are decompressing the compressed file def.txt into D.java .

```
import java.io.*;
import java.util.zip.*;

class UnCompress{
public static void main(String args[]){

try{
FileInputStream fin=new FileInputStream("def.txt");
InflaterInputStream in=new InflaterInputStream(fin);

FileOutputStream fout=new FileOutputStream("D.java");

int i;
while((i=in.read())!=-1){
fout.write((byte)i);
fout.flush();
}

fin.close();
fout.close();
in.close();

}catch(Exception e){System.out.println(e);}
System.out.println("rest of the code");
}
}
```

download this example

# PipedInputStream and PipedOutputStream classes

The PipedInputStream and PipedOutputStream classes can be used to read and write data simultaneously. Both streams are connected with each other using the connect() method of the PipedOutputStream class.

# Example of PipedInputStream and PipedOutputStream classes using threads

Here, we have created two threads t1 and t2. The **t1** thread writes the data using the PipedOutputStream object and the **t2** thread reads the data from that pipe using the PipedInputStream object. Both the piped stream object are connected with each other.

```
import java.io.*;

class PipedWR{
public static void main(String args[])throws Exception{

final PipedOutputStream pout=new PipedOutputStream();
final PipedInputStream pin=new PipedInputStream();

pout.connect(pin);//connecting the streams

//creating one thread t1 which writes the data
Thread t1=new Thread(){
public void run(){
for(int i=65;i<=90;i++){
try{

pout.write(i);
Thread.sleep(1000);

}catch(Exception e){}
}
}
};

//creating another thread t2 which reads the data
Thread t2=new Thread(){
public void run(){
try{
for(int i=65;i<=90;i++)
System.out.println(pin.read());
}catch(Exception e){}
}
};

//starting both threads
t1.start();
t2.start();

}}
```