Java I/O Tutorial

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

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

We can perform **file handling in java** by java IO API.

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.

In java, 3 streams are created for us automatically. All these streams are attached with console.

**1) System.out:**standard output stream

**2) System.in:**standard input stream

**3) System.err:**standard error stream

Let's see the code to print **output and error** message to the console.

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

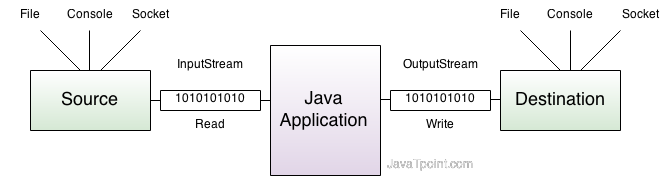
OutputStream

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 working of Java OutputStream and InputStream by the figure given below.

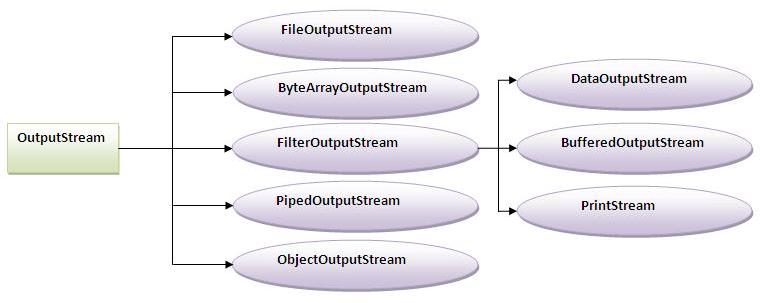


OutputStream class

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.

**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. |

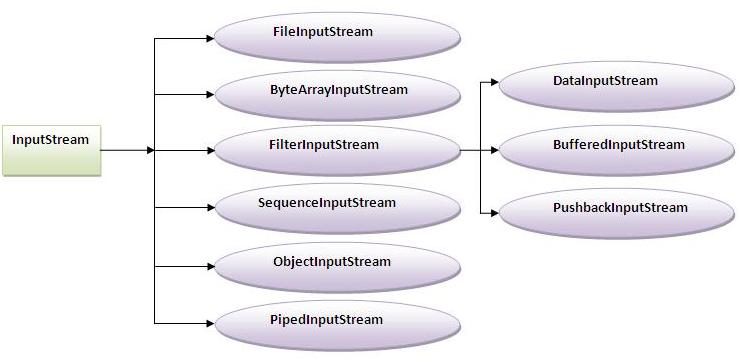


**InputStream class**

InputStream class is 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. |



FileInputStream and FileOutputStream (File Handling)

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

Java FileOutputStream class

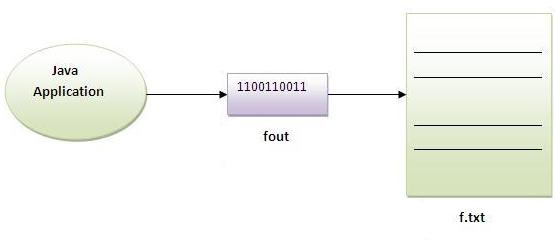
Java FileOutputStream is an output stream for writing data to a file.

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 Java FileOutputStream class

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



Java FileInputStream class

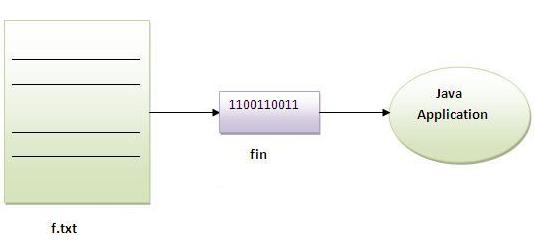
Java FileInputStream class obtains input bytes from a file.It is used for reading streams of raw bytes such as image data. For reading streams of characters, consider using FileReader.

It should be used to read byte-oriented data for example to read image, audio, video etc.

Example of FileInputStream class

1. **import** java.io.\*;
2. **class** SimpleRead{
3. **public** **static** **void** main(String args[]){
4. **try**{
5. FileInputStream fin=**new** FileInputStream("abc.txt");
6. **int** i=0;
7. **while**((i=fin.read())!=-1){
8. System.out.println((**char**)i);
9. }
10. fin.close();
11. }**catch**(Exception e){system.out.println(e);}
12. }
13. }

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.

1. **import** java.io.\*;
2. **class** C{
3. **public** **static** **void** main(String args[])**throws** Exception{
4. FileInputStream fin=**new** FileInputStream("C.java");
5. FileOutputStream fout=**new** FileOutputStream("M.java");
6. **int** i=0;
7. **while**((i=fin.read())!=-1){
8. fout.write((**byte**)i);
9. }
10. fin.close();
11. }
12. }

# Java ByteArrayOutputStream class

Java ByteArrayOutputStream class is used to write data into multiple files. In this stream, the data is written into a byte array that can be written to multiple stream.

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

The buffer of ByteArrayOutputStream automatically grows according to data.

#### Closing the ByteArrayOutputStream has no effect.

### Constructors of ByteArrayOutputStream class

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| ByteArrayOutputStream() | creates a new byte array output stream with the initial capacity of 32 bytes, though its size increases if necessary. |
| ByteArrayOutputStream(int size) | creates a new byte array output stream, with a buffer capacity of the specified size, in bytes. |

### Methods of ByteArrayOutputStream class

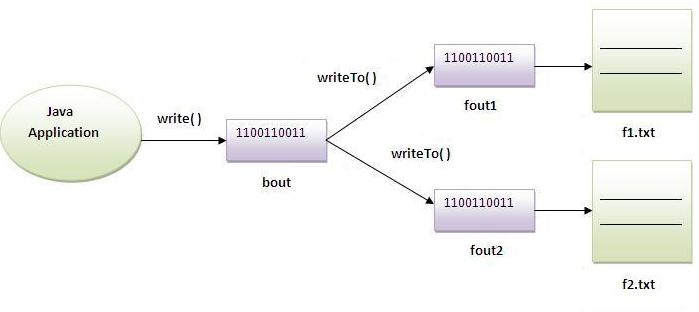
|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public synchronized void writeTo(OutputStream out) throws IOException | writes the complete contents of this byte array output stream to the specified output stream. |
| 2) public void write(byte b) throws IOException | writes byte into this stream. |
| 3) public void write(byte[] b) throws IOException | writes byte array into this stream. |
| 4) public void flush() | flushes this stream. |
| 5) public void close() | has no affect, it doesn't closes the bytearrayoutputstream. |

### Java ByteArrayOutputStream Example

Let's see a simple example of java ByteArrayOutputStream class to write data into 2 files.

1. **import** java.io.\*;
2. **class** S{
3. **public** **static** **void** main(String args[])**throws** Exception{
4. FileOutputStream fout1=**new** FileOutputStream("f1.txt");
5. FileOutputStream fout2=**new** FileOutputStream("f2.txt");
7. ByteArrayOutputStream bout=**new** ByteArrayOutputStream();
8. bout.write(139);
10. bout.writeTo(fout1);
11. bout.writeTo(fout2);
13. bout.flush();
14. bout.close();//has no effect
15. System.out.println("success...");
16. }
17. }

success...



# Java ByteArrayOutputStream class

Java ByteArrayOutputStream class is used to write data into multiple files. In this stream, the data is written into a byte array that can be written to multiple stream.

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

The buffer of ByteArrayOutputStream automatically grows according to data.

#### Closing the ByteArrayOutputStream has no effect.

### Constructors of ByteArrayOutputStream class

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| ByteArrayOutputStream() | creates a new byte array output stream with the initial capacity of 32 bytes, though its size increases if necessary. |
| ByteArrayOutputStream(int size) | creates a new byte array output stream, with a buffer capacity of the specified size, in bytes. |

### Methods of ByteArrayOutputStream class

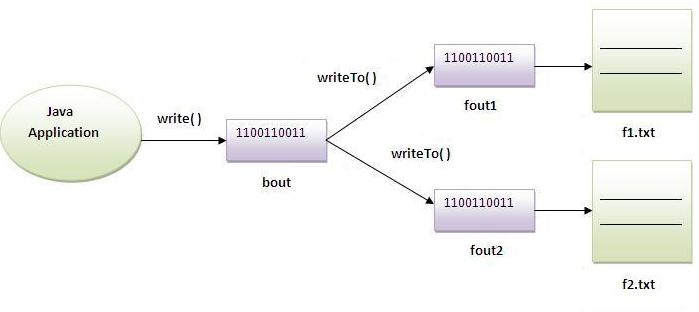
|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public synchronized void writeTo(OutputStream out) throws IOException | writes the complete contents of this byte array output stream to the specified output stream. |
| 2) public void write(byte b) throws IOException | writes byte into this stream. |
| 3) public void write(byte[] b) throws IOException | writes byte array into this stream. |
| 4) public void flush() | flushes this stream. |
| 5) public void close() | has no affect, it doesn't closes the bytearrayoutputstream. |

### Java ByteArrayOutputStream Example

Let's see a simple example of java ByteArrayOutputStream class to write data into 2 files.

1. **import** java.io.\*;
2. **class** S{
3. **public** **static** **void** main(String args[])**throws** Exception{
4. FileOutputStream fout1=**new** FileOutputStream("f1.txt");
5. FileOutputStream fout2=**new** FileOutputStream("f2.txt");
7. ByteArrayOutputStream bout=**new** ByteArrayOutputStream();
8. bout.write(139);
9. bout.writeTo(fout1);
10. bout.writeTo(fout2);
12. bout.flush();
13. bout.close();//has no effect
14. System.out.println("success...");
15. }
16. }

success...



Java SequenceInputStream class

Java SequenceInputStream class is used to read data from multiple streams. It reads data of streams one by one.

Constructors of SequenceInputStream class:

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| **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.

1. **import** java.io.\*;
2. **class** Simple{
3. **public** **static** **void** main(String args[])**throws** Exception{
4. FileinputStream fin1=**new** FileinputStream("f1.txt");
5. FileinputStream fin2=**new** FileinputStream("f2.txt");
7. SequenceinputStream sis=**new** SequenceinputStream(fin1,fin2);
8. **int** i;
9. **while**((i=sis.read())!=-1){
10. System.out.println((**char**)i);
11. }
12. sis.close();
13. fin1.close();
14. fin2.close();
15. }
16. }

Example of SequenceInputStream that reads the data from two files

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

1. //reading data of 2 files and writing it into one file
3. **import** java.io.\*;
4. **class** Simple{
5. **public** **static** **void** main(String args[])**throws** Exception{
7. FileinputStream fin1=**new** FileinputStream("f1.txt");
8. FileinputStream fin2=**new** FileinputStream("f2.txt");
10. FileOutputStream fout=**new** FileOutputStream("f3.txt");
12. SequenceinputStream sis=**new** SequenceinputStream(fin1,fin2);
13. **int** i;
14. **while**((i.sisread())!=-1)
15. {
16. fout.write(i);
17. }
18. sis.close();
19. fout.close();
20. fin.close();
21. fin.close();
23. }
24. }

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. |

1. **import** java.io.\*;
2. **import** java.util.\*;
4. **class** B{
5. **public** **static** **void** main(String args[])**throws** IOException{
7. //creating the FileInputStream objects for all the files
8. FileInputStream fin=**new** FileInputStream("A.java");
9. FileInputStream fin2=**new** FileInputStream("abc2.txt");
10. FileInputStream fin3=**new** FileInputStream("abc.txt");
11. FileInputStream fin4=**new** FileInputStream("B.java");
13. //creating Vector object to all the stream
14. Vector v=**new** Vector();
15. v.add(fin);
16. v.add(fin2);
17. v.add(fin3);
18. v.add(fin4);
20. //creating enumeration object by calling the elements method
21. Enumeration e=v.elements();
23. //passing the enumeration object in the constructor
24. SequenceInputStream bin=**new** SequenceInputStream(e);
25. **int** i=0;
27. **while**((i=bin.read())!=-1){
28. System.out.print((**char**)i);
29. }
31. bin.close();
32. fin.close();
33. fin2.close();
34. }
35. }

Java BufferedOutputStream and BufferedInputStream

Java BufferedOutputStream class

Java BufferedOutputStream class uses an internal buffer to store data. 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.

1. **import** java.io.\*;
2. **class** Test{
3. **public** **static** **void** main(String args[])**throws** Exception{
4. FileOutputStream fout=**new** FileOutputStream("f1.txt");
5. BufferedOutputStream bout=**new** BufferedOutputStream(fout);
6. String s="Sachin is my favourite player";
7. **byte** b[]=s.getBytes();
8. bout.write(b);
10. bout.flush();
11. bout.close();
12. fout.close();
13. System.out.println("success");
14. }
15. }

Output:

success...

Java BufferedInputStream class

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

Example of Java BufferedInputStream

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

1. **import** java.io.\*;
2. **class** SimpleRead{
3. **public** **static** **void** main(String args[]){
4. **try**{
5. FileInputStream fin=**new** FileInputStream("f1.txt");
6. BufferedInputStream bin=**new** BufferedInputStream(fin);
7. **int** i;
8. **while**((i=bin.read())!=-1){
9. System.out.println((**char**)i);
10. }
11. bin.close();
12. fin.close();
13. }**catch**(Exception e){system.out.println(e);}
14. }
15. }

Output:

Sachin is my favourite player

Java FileWriter and FileReader (File Handling in java)

Java FileWriter and FileReader classes are used to write and read data from text files. These are character-oriented classes, used for file handling in java.

Java has suggested not to use the FileInputStream and FileOutputStream classes if you have to read and write the textual information.

Java FileWriter class

Java FileWriter class is used to write character-oriented data to the file.

Constructors of FileWriter class

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| FileWriter(String file) | creates a new file. It gets file name in string. |
| FileWriter(File file) | creates a new file. It gets file name in File object. |

Methods of FileWriter class

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public void write(String text) | writes the string into FileWriter. |
| 2) public void write(char c) | writes the char into FileWriter. |
| 3) public void write(char[] c) | writes char array into FileWriter. |
| 4) public void flush() | flushes the data of FileWriter. |
| 5) public void close() | closes FileWriter. |

Java FileWriter Example

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

1. **import** java.io.\*;
2. **class** Simple{
3. **public** **static** **void** main(String args[]){
4. **try**{
5. FileWriter fw=**new** FileWriter("abc.txt");
6. fw.write("my name is sachin");
7. fw.close();
8. }**catch**(Exception e){System.out.println(e);}
9. System.out.println("success");
10. }
11. }

Output:

success...

Java FileReader class

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

Constructors of FileWriter class

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| FileReader(String file) | It gets filename in string. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException. |
| FileReader(File file) | It gets filename in file instance. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException. |

Methods of FileReader class

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public int read() | returns a character in ASCII form. It returns -1 at the end of file. |
| 2) public void close() | closes FileReader. |

Java FileReader Example

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

1. **import** java.io.\*;
2. **class** Simple{
3. **public** **static** **void** main(String args[])**throws** Exception{
4. FileReader fr=**new** FileReader("abc.txt");
5. **int** i;
6. **while**((i=fr.read())!=-1)
7. System.out.println((**char**)i);
9. fr.close();
10. }
11. }

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.

1. **import** java.io.\*;
2. **class** Simple{
3. **public** **static** **void** main(String args[])**throws** Exception{
5. CharArrayWriter out=**new** CharArrayWriter();
6. out.write("my name is");
8. FileWriter f1=**new** FileWriter("a.txt");
9. FileWriter f2=**new** FileWriter("b.txt");
10. FileWriter f3=**new** FileWriter("c.txt");
11. FileWriter f4=**new** FileWriter("d.txt");
13. out.writeTo(f1);
14. out.writeTo(f2);
15. out.writeTo(f3);
16. out.writeTo(f4);

19. f1.close();
20. f2.close();
21. f3.close();
22. f4.close();
23. }
24. }

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.

1. **import** java.io.\*;
2. **class** G5{
3. **public** **static** **void** main(String args[])**throws** Exception{
5. InputStreamReader r=**new** InputStreamReader(System.in);
6. BufferedReader br=**new** BufferedReader(r);
8. System.out.println("Enter your name");
9. String name=br.readLine();
10. System.out.println("Welcome "+name);
11. }
12. }

Output:Enter your 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.

1. **import** java.io.\*;
2. **class** G5{
3. **public** **static** **void** main(String args[])**throws** Exception{
5. InputStreamReader r=**new** InputStreamReader(System.in);
6. BufferedReader br=**new** BufferedReader(r);
8. String name="";
10. **while**(!name.equals("stop")){
11. System.out.println("Enter data: ");
12. name=br.readLine();
13. System.out.println("data is: "+name);
14. }
16. br.close();
17. r.close();
18. }
19. }

Output:Enter data: Amit

data is: Amit

Enter data: 10

data is: 10

Enter data: stop

data is: stop

Java Console class

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

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);

Methods of Console class

Let's see the commonly used methods of Console class.

|  |  |
| --- | --- |
| **Method** | **Description** |
| 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. |

How to get the object of Console

System class provides a static method console() that returns the unique 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.\*;
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: james gosling

Welcome james gosling

Java Console Example to read password

1. **import** java.io.\*;
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: sonoo

Java Scanner class

There are various ways to read input from the keyboard, the java.util.Scanner class is one of them.

The **Java Scanner** class breaks the input into tokens using a delimiter that is whitespace bydefault. It provides many methods to read and parse various primitive values.

Java Scanner class is widely used to parse text for string and primitive types using regular expression.

Java Scanner class extends Object class and implements Iterator and Closeable interfaces.

Commonly used methods of Scanner class

There is a list of commonly used Scanner class methods:

|  |  |
| --- | --- |
| **Method** | **Description** |
| 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. |

Java Scanner Example to get input from console

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

1. **import** java.util.Scanner;
2. **class** ScannerTest{
3. **public** **static** **void** main(String args[]){
4. Scanner sc=**new** Scanner(System.in);
6. System.out.println("Enter your rollno");
7. **int** rollno=sc.nextInt();
8. System.out.println("Enter your name");
9. String name=sc.next();
10. System.out.println("Enter your fee");
11. **double** fee=sc.nextDouble();
12. System.out.println("Rollno:"+rollno+" name:"+name+" fee:"+fee);
13. sc.close();
14. }
15. }

[download this scanner example](http://www.javatpoint.com/src/io/scanner.zip)

Output:

Enter your rollno

111

Enter your name

Ratan

Enter

450000

Rollno:111 name:Ratan fee:450000

Java Scanner Example with delimiter

Let's see the example of Scanner class with delimiter. The \s represents whitespace.

1. **import** java.util.\*;
2. **public** **class** ScannerTest2{
3. **public** **static** **void** main(String args[]){
4. String input = "10 tea 20 coffee 30 tea buiscuits";
5. Scanner s = **new** Scanner(input).useDelimiter("\\s");
6. System.out.println(s.nextInt());
7. System.out.println(s.next());
8. System.out.println(s.nextInt());
9. System.out.println(s.next());
10. s.close();
11. }}

Output:

10

tea

20

Coffee

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. |

1. **import** java.io.\*;
2. **class** PrintStreamTest{
3. **public** **static** **void** main(String args[])**throws** Exception{
5. FileOutputStream fout=**new** FileOutputStream("mfile.txt");
6. PrintStream pout=**new** PrintStream(fout);
7. pout.println(1900);
8. pout.println("Hello Java");
9. pout.println("Welcome to Java");
10. pout.close();
11. fout.close();
13. }
14. }

[download this PrintStream example](http://www.javatpoint.com/src/io/printstream1.zip)

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

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

1. **class** PrintStreamTest{
2. **public** **static** **void** main(String args[]){
3. **int** a=10;
4. System.out.printf("%d",a);//Note, out is the object of PrintStream class
6. }
7. }

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

1. **import** java.io.\*;
2. **import** java.util.zip.\*;
4. **class** Compress{
5. **public** **static** **void** main(String args[]){
7. **try**{
8. FileInputStream fin=**new** FileInputStream("Deflater.java");
10. FileOutputStream fout=**new** FileOutputStream("def.txt");
11. DeflaterOutputStream out=**new** DeflaterOutputStream(fout);
13. **int** i;
14. **while**((i=fin.read())!=-1){
15. out.write((**byte**)i);
16. out.flush();
17. }
19. fin.close();
20. out.close();
22. }**catch**(Exception e){System.out.println(e);}
23. System.out.println("rest of the code");
24. }
25. }

[download this example](http://www.javatpoint.com/src/io/compress.zip)

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 .

1. **import** java.io.\*;
2. **import** java.util.zip.\*;
4. **class** UnCompress{
5. **public** **static** **void** main(String args[]){
7. **try**{
8. FileInputStream fin=**new** FileInputStream("def.txt");
9. InflaterInputStream in=**new** InflaterInputStream(fin);
11. FileOutputStream fout=**new** FileOutputStream("D.java");
13. **int** i;
14. **while**((i=in.read())!=-1){
15. fout.write((**byte**)i);
16. fout.flush();
17. }
19. fin.close();
20. fout.close();
21. in.close();
23. }**catch**(Exception e){System.out.println(e);}
24. System.out.println("rest of the code");
25. }
26. }

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.

1. **import** java.io.\*;
2. **class** PipedWR{
3. **public** **static** **void** main(String args[])**throws** Exception{
4. **final** PipedOutputStream pout=**new** PipedOutputStream();
5. **final** PipedInputStream pin=**new** PipedInputStream();
7. pout.connect(pin);//connecting the streams
8. //creating one thread t1 which writes the data
9. Thread t1=**new** Thread(){
10. **public** **void** run(){
11. **for**(**int** i=65;i<=90;i++){
12. **try**{
13. pout.write(i);
14. Thread.sleep(1000);
15. }**catch**(Exception e){}
16. }
17. }
18. };
19. //creating another thread t2 which reads the data
20. Thread t2=**new** Thread(){
21. **public** **void** run(){
22. **try**{
23. **for**(**int** i=65;i<=90;i++)
24. System.out.println(pin.read());
25. }**catch**(Exception e){}
26. }
27. };
28. //starting both threads
29. t1.start();
30. t2.start();
31. }}

|  |
| --- |
|  |