Java SequenceInputStream Class

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

Java SequenceInputStream Class declaration

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

```
public class SequenceInputStream extends InputStream
```

Constructors of SequenceInputStream class

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

Methods of SequenceInputStream class

Method	Description
int read()	It is used to read the next byte of data from the input stream.
int read(byte[] ary, int off, int len)	It is used to read len bytes of data from the input stream into the array of bytes.
int available()	It is used to return the maximum number of byte that can be read from an input stream.
void close()	It is used to close the input stream.

Java SequenceInputStream Example

In this example, we are printing the data of two files testin.txt and testout.txt.

```
package com.javatpoint;

import java.io.*;
class InputStreamExample {
  public static void main(String args[])throws Exception{
   FileInputStream input1=new FileInputStream("D:\\testin.txt");
   FileInputStream input2=new FileInputStream("D:\\testout.txt");
   SequenceInputStream inst=new SequenceInputStream(input1, input2);
   int j;
   while((j=inst.read())!=-1){
    System.out.print((char)j);
   }
   inst.close();
   input1.close();
   input2.close();
  }
}
```

Here, we are assuming that you have two files: testin.txt and testout.txt which have following information:

testin.txt:

```
Welcome to Java IO Programming.
```

testout.txt:

```
It is the example of Java SequenceInputStream class.
```

After executing the program, you will get following output:

Output:

```
Welcome to Java IO Programming. It is the example of Java SequenceInputStream class.
```

Example that reads the data from two files and writes into another file

In this example, we are writing the data of two files **testin1.txt** and **testin2.txt** into another file named **testout.txt.**

```
package com.javatpoint;

import java.io.*;
class Input1{
  public static void main(String args[])throws Exception{
   FileInputStream fin1=new FileInputStream("D:\\testin1.txt");
   FileInputStream fin2=new FileInputStream("D:\\testin2.txt");
   FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
   SequenceInputStream sis=new SequenceInputStream(fin1,fin2);
   int i;
   while((i=sis.read())!=-1)
   {
     fout.write(i);
   }
   sis.close();
   fout.close();
   fin1.close();
   fin2.close();
   System.out.println("Success..");
  }
}
```

Output:

```
Succeess...
```

testout.txt:

```
Welcome to Java IO Programming. It is the example of Java SequenceInputStream class.
```

SequenceInputStream example that reads data using enumeration

If we need to read the data from more than two files, we need to use Enumeration. Enumeration object can be obtained by calling elements() method of the Vector class. Let's see the simple example where we are reading the data from 4 files: a.txt, b.txt, c.txt and d.txt.

```
package com.javatpoint;
import java.io.*;
import java.util.*;
class Input2{
public static void main(String args[])throws IOException{
//creating the FileInputStream objects for all the files
FileInputStream fin=new FileInputStream("D:\\a.txt");
FileInputStream fin2=new FileInputStream("D:\\b.txt");
FileInputStream fin3=new FileInputStream("D:\\c.txt");
FileInputStream fin4=new FileInputStream("D:\\d.txt");
//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();
}
}
```

The a.txt, b.txt, c.txt and d.txt have following information:

a.txt:

```
Welcome
```

b.txt:

```
to
```

c.txt:

```
java
```

d.txt:


```
programming
```

Output:

```
Welcometojavaprogramming
```

← Prev Next →

Java ByteArrayOutputStream Class

Java ByteArrayOutputStream class is used to **write common data** into multiple files. In this stream, the data is written into a byte array which can be written to multiple streams later.

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

The buffer of ByteArrayOutputStream automatically grows according to data.

Java ByteArrayOutputStream class declaration

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

```
public class ByteArrayOutputStream extends OutputStream
```

Java ByteArrayOutputStream class constructors

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.

Java ByteArrayOutputStream class methods

Method	Description
int size()	It is used to returns the current size of a buffer.
byte[] toByteArray()	It is used to create a newly allocated byte array.
String toString()	It is used for converting the content into a string decoding bytes using a platform default character set.
String toString(String charsetName)	It is used for converting the content into a string decoding bytes using a specified charsetName.

void write(int b)	It is used for writing the byte specified to the byte array output stream.
void write(byte[] b, int off, int len	It is used for writing **len** bytes from specified byte array starting from the offset **off** to the byte array output stream.
void writeTo(OutputStream out)	It is used for writing the complete content of a byte array output stream to the specified output stream.
void reset()	It is used to reset the count field of a byte array output stream to zero value.
void close()	It is used to close the ByteArrayOutputStream.

Example of Java ByteArrayOutputStream

Let's see a simple example of java ByteArrayOutputStream class to write common data into 2 files: f1.txt and f2.txt.

```
package com.javatpoint;
import java.io.*;
public class DataStreamExample {
public static void main(String args[])throws Exception{
    FileOutputStream fout1=new FileOutputStream("D:\\f1.txt");
    FileOutputStream fout2=new FileOutputStream("D:\\f2.txt");

    ByteArrayOutputStream bout=new ByteArrayOutputStream();
    bout.write(65);
    bout.writeTo(fout1);
    bout.writeTo(fout2);

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

```
}
```

Output:

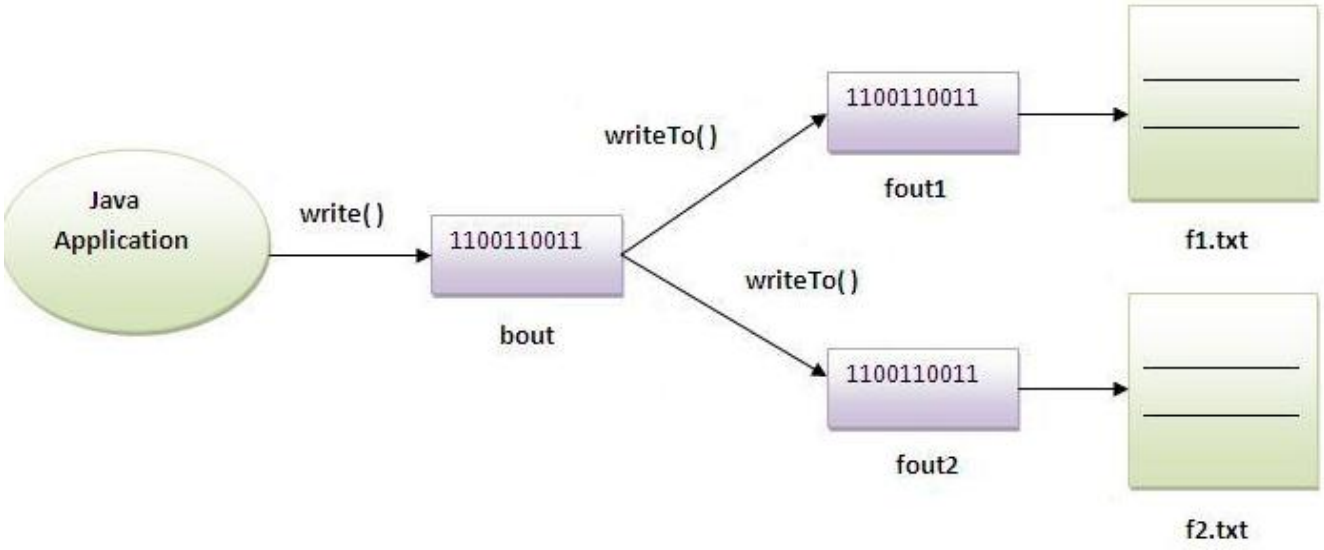
```
Success...
```

f1.txt:

```
A
```

f2.txt:

```
A
```



← Prev Next →

Java ByteArrayInputStream Class

The ByteArrayInputStream is composed of two words: ByteArray and InputStream. As the name suggests, it can be used to read byte array as input stream.

Java ByteArrayInputStream class contains an internal buffer which is used to **read byte array** as stream. In this stream, the data is read from a byte array.

The buffer of ByteArrayInputStream automatically grows according to data.

Java ByteArrayInputStream class declaration

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

```
public class ByteArrayInputStream extends InputStream
```

Java ByteArrayInputStream class constructors

Constructor	Description
ByteArrayInputStream(byte[] ary)	Creates a new byte array input stream which uses **ary** as its buffer array.
ByteArrayInputStream(byte[] ary, int offset, int len)	Creates a new byte array input stream which uses **ary** as its buffer array that can read up to specified **len** bytes of data from an array.

Java ByteArrayInputStream class methods

Methods	Description
int available()	It is used to return the number of remaining 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[] ary, int off, int len)	It is used to read up to len bytes of data from an array of bytes in the input stream.

boolean markSupported()	It is used to test the input stream for mark and reset method.
long skip(long x)	It is used to skip the x bytes of input from the input stream.
void mark(int readAheadLimit)	It is used to set the current marked position in the stream.
void reset()	It is used to reset the buffer of a byte array.
void close()	It is used for closing a ByteArrayInputStream.

Example of Java ByteArrayInputStream

Let's see a simple example of java ByteArrayInputStream class to read byte array as input stream.

```
package com.javatpoint;
import java.io.*;
public class ReadExample {
  public static void main(String[] args) throws IOException {
    byte[] buf = { 35, 36, 37, 38 };
    // Create the new byte array input stream
    ByteArrayInputStream byt = new ByteArrayInputStream(buf);
    int k = 0;
    while ((k = byt.read()) != -1) {
      //Conversion of a byte into character
      char ch = (char) k;
      System.out.println("ASCII value of Character is:" + k + "; Special character is: " + ch);
    }
  }
}
```

Output:

```
ASCII value of Character is:35; Special character is: #
ASCII value of Character is:36; Special character is: $
```

```
ASCII value of Character is:37; Special character is: %
ASCII value of Character is:38; Special character is: &
```

← Prev Next →

Youtube For Videos Join Our Youtube Channel: Join Now

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

Learn Latest Tutorials

Splunk tutorial Splunk
SPSS tutorial SPSS
Swagger tutorial Swagger
T-SQL tutorial Transact-SQL
Tumblr tutorial Tumblr
React tutorial ReactJS
Regex tutorial Regex
Reinforcement learning tutorial

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:

```
public class DataOutputStream extends FilterOutputStream implements DataOutput
```

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.

Example of DataOutputStream class

In this example, we are writing the data to a text file **testout.txt** using DataOutputStream class.

```
package com.javatpoint;

import java.io.*;
public class OutputExample {
    public static void main(String[] args) throws IOException {
        FileOutputStream file = new FileOutputStream(D:\\testout.txt);
        DataOutputStream data = new DataOutputStream(file);
        data.writeInt(65);
        data.flush();
        data.close();
        System.out.println("Succcess...");
    }
}
```

Output:

```
Succcess...
```

testout.txt:

```
A
```

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:

```
public class DataInputStream extends FilterInputStream implements DataInput
```

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.

Example of DataInputStream class

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

```
package com.javatpoint;
import java.io.*;
public class DataStreamExample {
  public static void main(String[] args) throws IOException {
    InputStream input = new FileInputStream("D:\\testout.txt");
    DataInputStream inst = new DataInputStream(input);
    int count = input.available();
    byte[] ary = new byte[count];
    inst.read(ary);
    for (byte bt : ary) {
      char k = (char) bt;
      System.out.print(k+"-");
    }
  }
}
```

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:

```
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 stream.
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 output 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

```
import java.io.*;
public class FilterExample {
   public static void main(String[] args) throws IOException {
      File data = new File("D:\\testout.txt");
      FileOutputStream file = new FileOutputStream(data);
      FilterOutputStream filter = new FilterOutputStream(file);
      String s="Welcome to javaTpoint.";
      byte b[]=s.getBytes();
      filter.write(b);
```

```
        filter.flush();
        filter.close();
        file.close();
        System.out.println("Success...");
    }
}
```

Output:

```
Success...
```

testout.txt

```
Welcome to javaTpoint.
```

← Prev Next →

Feedback

- Send your Feedback to feedback@javatpoint.com

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

```
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 javatpoint
```

Output:

```
Welcome to javatpoint
```

← Prev Next →

Java - ObjectStreamClass

ObjectStreamClass act as a Serialization descriptor for class. This class contains the name and serialVersionUID of the class.

Fields

Modifier and Type	Field	Description
static ObjectStreamField[]	NO_FIELDS	serialPersistentFields value indicating no serializable fields

Methods

Modifier and Type	Method	Description
Class<?>	forClass()	It returns the class in the local VM that this version is mapped to.
ObjectStreamField	getField(String name)	It gets the field of this class by name.
ObjectStreamField[]	getFields()	It returns an array of the fields of this serialization class.
String	getName()	It returns the name of the class described by this descriptor.
long	getSerialVersionUID()	It returns the serialVersionUID for this class.
Static ObjectStreamClass	lookup(Class<?> cl)	It finds the descriptor for a class that can be serialized.
Static ObjectStreamClass	lookupAny(Class<?> cl)	It returns the descriptor for any class, regardless of whether it implements Serializable.
String	toString()	It returns a string describing this ObjectStreamClass.

Example

```
import java.io.ObjectStreamClass;
import java.util.Calendar;

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

      // create a new object stream class for Integers
      ObjectStreamClass osc = ObjectStreamClass.lookup(SmartPhone.class);

      // get the value field from ObjectStreamClass for integers
      System.out.println("" + osc.getField("price"));

      // create a new object stream class for Calendar
      ObjectStreamClass osc2 = ObjectStreamClass.lookup(String.class);

      // get the Class instance for osc2
      System.out.println("" + osc2.getField("hash"));

   }
}
```

Output:

```
I price
null
```

← Prev Next →

Java ObjectStreamField class

A description of a Serializable field from a Serializable class. An array of ObjectStreamFields is used to declare the Serializable fields of a class.

The java.io.ObjectStreamClass.getField(String name) method gets the field of this class by name.

Constructor

Constructor	Description
ObjectStreamField(String name, Class<?> type)	It creates a Serializable field with the specified type.
ObjectStreamField(String name, Class<?> type, boolean unshared)	It creates an ObjectStreamField representing a serializable field with the given name and type.

Methods

Modifier and Type	Method	Description
int	compareTo(Object obj)	It compares this field with another ObjectStreamField.
String	getName()	It gets the name of this field.
int	GetOffset()	Offset of field within instance data.
Class<?>	getType()	It get the type of the field.
char	getTypeCode()	It returns character encoding of field type.
String	getTypeString()	It return the JVM type signature.
boolean	isPrimitive()	It return true if this field has a primitive type.
boolean	isUnshared()	It returns boolean value indicating whether or not the serializable field represented by this ObjectStreamField instance is unshared.

protected void	setOffset(int offset)	Offset within instance data.
String	toString()	It return a string that describes this field.

public char getTypeCode()

Returns character encoding of field type. The encoding is as follows:

B	byte
C	char
D	double
F	float
I	int
J	long
L	class or interface
S	short
Z	boolean
[	array

Returns:

the typecode of the serializable field

Example:

```
import java.io.ObjectStreamClass;
import java.util.Calendar;

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

        // create a new object stream class for Integers
        ObjectStreamClass osc = ObjectStreamClass.lookup(String.class);
```

```
        // get the value field from ObjectStreamClass for integers
        System.out.println("" + osc.getField("value"));

        // create a new object stream class for Calendar
        ObjectStreamClass osc2 = ObjectStreamClass.lookup(Calendar.class);

        // get the Class instance for osc2
        System.out.println("" + osc2.getField("isTimeSet"));

    }
}
```

Output:

```
I value
Z isTimeSet
```

← Prev Next →

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.

```
String text=System.console().readLine();
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
```

Java Console class methods

Method	Description
Reader reader()	It is used to retrieve the reader object associated with the console
String readLine()	It is used to read a single line of text from the console.
String readLine(String fmt, Object... args)	It provides a formatted prompt then reads the single line of text from the console.
char[] readPassword()	It is used to read password that is not being displayed on the console.
char[] readPassword(String fmt, Object... args)	It provides a formatted prompt then reads the password that is not being displayed on the console.

Console format(String fmt, Object... args)	It is used to write a formatted string to the console output stream.
Console printf(String format, Object... args)	It is used to write a string to the console output stream.
PrintWriter writer()	It is used to retrieve the PrintWriter object associated with the console.
void flush()	It is used to flushes the console.

How to get the object of Console

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

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

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

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


Java Console Example

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

Output

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

Java Console Example to read password

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

Output

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

← Prev Next →

Java FilePermission Class

Java FilePermission class contains the permission related to a directory or file. All the permissions are related with path. The path can be of two types:

1) **D:\\IO\\-**: It indicates that the permission is associated with all sub directories and files recursively.

2) **D:\\IO***: It indicates that the permission is associated with all directory and files within this directory excluding sub directories.

Java FilePermission class declaration

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

```
public final class FilePermission extends Permission implements Serializable
```

Methods of FilePermission class

Method	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.

Java FilePermission class methods

Method	Description
int hashCode()	It is used to return the hash code value of an object.
String getActions()	It is used to return the "canonical string representation" of an action.
boolean equals(Object obj)	It is used to check the two FilePermission objects for equality.

boolean implies(Permission p)	It is used to check the FilePermission object for the specified permission.
PermissionCollection newPermissionCollection()	It is used to return the new PermissonCollection object for storing the FilePermission object.

Java FilePermission Example

Let's see the simple example in which permission of a directory path is granted with read permission and a file of this directory is granted for write permission.

```
package com.javatpoint;

import java.io.*;
import java.security.PermissionCollection;
public class FilePermissionExample{
    public static void main(String[] args) throws IOException {
     String srg = "D:\\IO Package\\java.txt";
     FilePermission file1 = new FilePermission("D:\\IO Package\\-", "read");
     PermissionCollection permission = file1.newPermissionCollection();
     permission.add(file1);
         FilePermission file2 = new FilePermission(srg, "write");
         permission.add(file2);
         if(permission.implies(new FilePermission(srg, "read,write"))) {
         System.out.println("Read, Write permission is granted for the path "+srg );
         }else {
        System.out.println("No Read, Write permission is granted for the path "+srg);          }
    }
}
```

Output

```
Read, Write permission is granted for the path D:\IO Package\java.txt
```

Java Writer

It is an abstract class for writing to character streams. The methods that a subclass must implement are write(char[], int, int), flush(), and close(). Most subclasses will override some of the methods defined here to provide higher efficiency, functionality or both.

Fields

Modifier and Type	Field	Description
protected Object	lock	The object used to synchronize operations on this stream.

Constructor

Modifier	Constructor	Description
protected	Writer()	It creates a new character-stream writer whose critical sections will synchronize on the writer itself.
protected	Writer(Object lock)	It creates a new character-stream writer whose critical sections will synchronize on the given object.

Methods

Modifier and Type	Method	Description
Writer	append(char c)	It appends the specified character to this writer.
Writer	append(CharSequence csq)	It appends the specified character sequence to this writer
Writer	append(CharSequence csq, int start, int end)	It appends a subsequence of the specified character sequence to this writer.
abstract void	close()	It closes the stream, flushing it first.
abstract void	flush()	It flushes the stream.
void	write(char[] cbuf)	It writes an array of characters.

abstract void	write(char[] cbuf, int off, int len)	It writes a portion of an array of characters.
void	write(int c)	It writes a single character.
void	write(String str)	It writes a string.
void	write(String str, int off, int len)	It writes a portion of a string.

Java Writer Example

```
import java.io.*;
public class WriterExample {
    public static void main(String[] args) {
        try {
            Writer w = new FileWriter("output.txt");
            String content = "I love my country";
            w.write(content);
            w.close();
            System.out.println("Done");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
Done
```

output.txt:

```
I love my country
```

← Prev Next →

Java Reader

Java Reader is an abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods to provide higher efficiency, additional functionality, or both.

Some of the implementation class are BufferedReader, CharArrayReader, FilterReader, InputStreamReader, PipedReader, StringReader

Fields

Modifier and Type	Field	Description
protected Object	lock	The object used to synchronize operations on this stream.

Constructor

Modifier	Constructor	Description
protected	Reader()	It creates a new character-stream reader whose critical sections will synchronize on the reader itself.
protected	Reader(Object lock)	It creates a new character-stream reader whose critical sections will synchronize on the given object.

Methods

Modifier and Type	Method	Description
abstract void	close()	It closes the stream and releases any system resources associated with it.
void	mark(int readAheadLimit)	It marks the present position in the stream.
boolean	markSupported()	It tells whether this stream supports the mark() operation.
int	read()	It reads a single character.
int	read(char[] cbuf)	It reads characters into an array.

abstract int	read(char[] cbuf, int off, int len)	It reads characters into a portion of an array.
int	read(CharBuffer target)	It attempts to read characters into the specified character buffer.
boolean	ready()	It tells whether this stream is ready to be read.
void	reset()	It resets the stream.
long	skip(long n)	It skips characters.

Example

```
import java.io.*;
public class ReaderExample {
   public static void main(String[] args) {
      try {
         Reader reader = new FileReader("file.txt");
         int data = reader.read();
         while (data != -1) {
            System.out.print((char) data);
            data = reader.read();
         }
         reader.close();
      } catch (Exception ex) {
         System.out.println(ex.getMessage());
      }
   }
}
```

file.txt:

```
I love my country
```

Output:

```
I love my country
```

← Prev Next →

For Videos Join Our Youtube Channel: Join Now

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

Learn Latest Tutorials

Splunk

SPSS

Swagger

Transact-SQL

Tumblr

ReactJS

Regex

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
```

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
void write(String text)	It is used to write the string into FileWriter.
void write(char c)	It is used to write the char into FileWriter.
void write(char[] c)	It is used to write char array into FileWriter.
void flush()	It is used to flushes the data of FileWriter.
void close()	It is used to close the FileWriter.

Java FileWriter Example

In this example, we are writing the data in the file testout.txt using Java FileWriter class.

```
package com.javatpoint;
import java.io.FileWriter;
public class FileWriterExample {
    public static void main(String args[]){
        try{
          FileWriter fw=new FileWriter("D:\\testout.txt");
          fw.write("Welcome to javaTpoint.");
          fw.close();
         }catch(Exception e){System.out.println(e);}
         System.out.println("Success...");
    }
}
```

Output:

```
Success...
```

testout.txt:

```
Welcome to javaTpoint.
```

← Prev Next →

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
```

Constructors of FileReader 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
int read()	It is used to return a character in ASCII form. It returns -1 at the end of file.
void close()	It is used to close the FileReader class.

Java FileReader Example

In this example, we are reading the data from the text file **testout.txt** using Java FileReader class.

```
package com.javatpoint;

import java.io.FileReader;
public class FileReaderExample {
   public static void main(String args[])throws Exception{
        FileReader fr=new FileReader("D:\\testout.txt");
        int i;
        while((i=fr.read())!=-1)
        System.out.print((char)i);
        fr.close();
   }
}
```

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

```
Welcome to javaTpoint.
```

Output:

```
Welcome to javaTpoint.
```

← Prev Next →

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
```

Class constructors

Constructor	Description
BufferedWriter(Writer wrt)	It is used to create a buffered character output stream that uses the default size for an output buffer.
BufferedWriter(Writer wrt, int size)	It is used to create a buffered character output stream that uses the specified size for an output buffer.

Class methods

Method	Description
void newLine()	It is used to add a new line by writing a line separator.
void write(int c)	It is used to write a single character.
void write(char[] cbuf, int off, int len)	It is used to write a portion of an array of characters.
void write(String s, int off, int len)	It is used to write a portion of a string.
void flush()	It is used to flushes the input stream.
void close()	It is used to closes the input stream

Example of Java BufferedWriter

Let's see the simple example of writing the data to a text file **testout.txt** using Java BufferedWriter.

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

Output:

```
success
```

testout.txt:

```
Welcome to javaTpoint.
```

← Prev Next →

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 class constructors

Constructor	Description
BufferedReader(Reader rd)	It is used to create a buffered character input stream that uses the default size for an input buffer.
BufferedReader(Reader rd, int size)	It is used to create a buffered character input stream that uses the specified size for an input buffer.

Java BufferedReader class methods

Method	Description
int read()	It is used for reading a single character.
int read(char[] cbuf, int off, int len)	It is used for reading characters into a portion of an array.
boolean markSupported()	It is used to test the input stream support for the mark and reset method.
String readLine()	It is used for reading a line of text.
boolean ready()	It is used to test whether the input stream is ready to be read.

long skip(long n)	It is used for skipping the characters.
void reset()	It repositions the stream at a position the mark method was last called on this input stream.
void mark(int readAheadLimit)	It is used for marking the present position in a stream.
void close()	It closes the input stream and releases any of the system resources associated with the stream.

Java BufferedReader Example

In this example, we are reading the data from the text file **testout.txt** using Java BufferedReader class.

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

        int i;
        while((i=br.read())!=-1){
        System.out.print((char)i);
        }
        br.close();
        fr.close();
   }
}
```

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

```
Welcome to javaTpoint.
```

Output:

```
Welcome to javaTpoint.
```

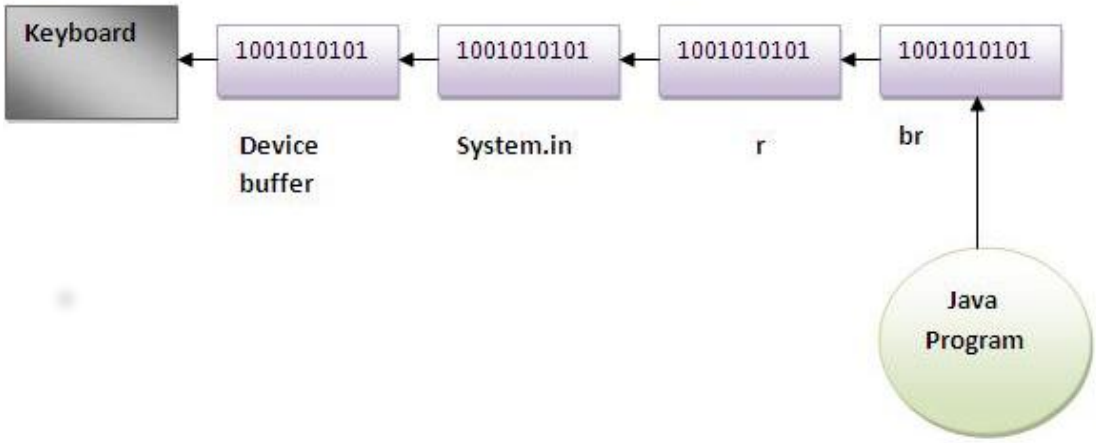
Reading data from console by InputStreamReader and BufferedReader

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

```
package com.javatpoint;
import java.io.*;
public class BufferedReaderExample{
public static void main(String args[])throws Exception{
    InputStreamReader r=new InputStreamReader(System.in);
    BufferedReader br=new BufferedReader(r);
    System.out.println("Enter your name");
    String name=br.readLine();
    System.out.println("Welcome "+name);
}
}
```

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.

```
package com.javatpoint;
import java.io.*;
public class BufferedReaderExample{
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: Nakul
data is: Nakul
Enter data: 12
data is: 12
Enter data: stop
data is: stop
```

← Prev Next →

Java CharArrayReader Class

The CharArrayReader is composed of two words: CharArray and Reader. The CharArrayReader class is used to read character array as a reader (stream). It inherits Reader class.

Java CharArrayReader class declaration

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

```
public class CharArrayReader extends Reader
```

Java CharArrayReader class methods

Method	Description
int read()	It is used to read a single character
int read(char[] b, int off, int len)	It is used to read characters into the portion of an array.
boolean ready()	It is used to tell whether the stream is ready to read.
boolean markSupported()	It is used to tell whether the stream supports mark() operation.
long skip(long n)	It is used to skip the character in the input stream.
void mark(int readAheadLimit)	It is used to mark the present position in the stream.
void reset()	It is used to reset the stream to a most recent mark.
void close()	It is used to closes the stream.

Example of CharArrayReader Class:

Let's see the simple example to read a character using Java CharArrayReader class.

```
package com.javatpoint;

import java.io.CharArrayReader;
public class CharArrayExample{
```

```
  public static void main(String[] ag) throws Exception {
    char[] ary = { 'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't' };
    CharArrayReader reader = new CharArrayReader(ary);
    int k = 0;
    // Read until the end of a file
    while ((k = reader.read()) != -1) {
      char ch = (char) k;
      System.out.print(ch + " : ");
      System.out.println(k);
    }
  }
}
```

Output

```
j : 106
a : 97
v : 118
a : 97
t : 116
p : 112
o : 111
i : 105
n : 110
t : 116
```

← Prev Next →

Java CharArrayWriter Class

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

Java CharArrayWriter class declaration

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

```
public class CharArrayWriter extends Writer
```

Java CharArrayWriter class Methods

Method	Description
int size()	It is used to return the current size of the buffer.
char[] toCharArray()	It is used to return the copy of an input data.
String toString()	It is used for converting an input data to a string.
CharArrayWriter append(char c)	It is used to append the specified character to the writer.
CharArrayWriter append(CharSequence csq)	It is used to append the specified character sequence to the writer.
CharArrayWriter append(CharSequence csq, int start, int end)	It is used to append the subsequence of a specified character to the writer.
void write(int c)	It is used to write a character to the buffer.
void write(char[] c, int off, int len)	It is used to write a character to the buffer.
void write(String str, int off, int len)	It is used to write a portion of string to the buffer.
void writeTo(Writer out)	It is used to write the content of buffer to different character stream.

void flush()	It is used to flush the stream.
void reset()	It is used to reset the buffer.
void close()	It is used to close the stream.

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.

```
package com.javatpoint;

import java.io.CharArrayWriter;
import java.io.FileWriter;
public class CharArrayWriterExample {
public static void main(String args[])throws Exception{
        CharArrayWriter out=new CharArrayWriter();
        out.write("Welcome to javaTpoint");
        FileWriter f1=new FileWriter("D:\\a.txt");
        FileWriter f2=new FileWriter("D:\\b.txt");
        FileWriter f3=new FileWriter("D:\\c.txt");
        FileWriter f4=new FileWriter("D:\\d.txt");
        out.writeTo(f1);
        out.writeTo(f2);
        out.writeTo(f3);
        out.writeTo(f4);
        f1.close();
        f2.close();
        f3.close();
        f4.close();
        System.out.println("Success...");
       }
      }
```

Output

```
Success...
```

After executing the program, you can see that all files have common data: Welcome to javaTpoint.

a.txt:

```
Welcome to javaTpoint
```

b.txt:

```
Welcome to javaTpoint
```

c.txt:

```
Welcome to javaTpoint
```

d.txt:

```
Welcome to javaTpoint
```

← Prev Next →

Feedback

- Send your Feedback to feedback@javatpoint.com

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:

```
public class PrintStream extends FilterOutputStream implements Closeable. Appendable
```

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.javatpoint;

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.

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

```
}
```

Output

```
19
```

← Prev Next →

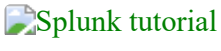
For Videos Join Our Youtube Channel: Join Now

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

Learn Latest Tutorials



Splunk

SPSS

Swagger

Transact-SQL

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

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

```
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.javatpoint;

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("Javatpoint 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();
    }
}
```

Outpt

```
Javatpoint provides tutorials of all technology.
```

The content of a text file **testout.txt** is set with the data **Like Java, Spring, Hibernate, Android, PHP etc.**

Java OutputStreamWriter

OutputStreamWriter is a class which is used to convert character stream to byte stream, the characters are encoded into byte using a specified charset. write() method calls the encoding converter which converts the character into bytes. The resulting bytes are then accumulated in a buffer before being written into the underlying output stream. The characters passed to write() methods are not buffered. We optimize the performance of OutputStreamWriter by using it with in a BufferedWriter so that to avoid frequent converter invocation.

Constructor

Constructor	Description
OutputStreamWriter(OutputStream out)	It creates an OutputStreamWriter that uses the default character encoding.
OutputStreamWriter(OutputStream out, Charset cs)	It creates an OutputStreamWriter that uses the given charset.
OutputStreamWriter(OutputStream out, CharsetEncoder enc)	It creates an OutputStreamWriter that uses the given charset encoder.
OutputStreamWriter(OutputStream out, String charsetName)	It creates an OutputStreamWriter that uses the named charset.

Methods

Modifier and Type	Method	Description
void	close()	It closes the stream, flushing it first.
void	flush()	It flushes the stream.
String	getEncoding()	It returns the name of the character encoding being used by this stream.
void	write(char[] cbuf, int off, int len)	It writes a portion of an array of characters.
void	write(int c)	It writes a single character.

void	write(String str, int off, int len)	It writes a portion of a string.

Example

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

      try {
         OutputStream outputStream = new FileOutputStream("output.txt");
         Writer outputStreamWriter = new OutputStreamWriter(outputStream);

         outputStreamWriter.write("Hello World");

         outputStreamWriter.close();
      } catch (Exception e) {
         e.getMessage();
      }
   }
}
```

Output:

```
output.txt file will contains text  "Hello World"
```

Java InputStreamReader

An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified charset. The charset that it uses may be specified by name or may be given explicitly, or the platform's default charset may be accepted.

Constructor

Constructor name	Description
InputStreamReader(InputStream in)	It creates an InputStreamReader that uses the default charset.
InputStreamReader(InputStream in, Charset cs)	It creates an InputStreamReader that uses the given charset.
InputStreamReader(InputStream in, CharsetDecoder dec)	It creates an InputStreamReader that uses the given charset decoder.
InputStreamReader(InputStream in, String charsetName)	It creates an InputStreamReader that uses the named charset.

Method

Modifier and Type	Method	Description
void	close()	It closes the stream and releases any system resources associated with it.
String	getEncoding()	It returns the name of the character encoding being used by this stream.
int	read()	It reads a single character.
int	read(char[] cbuf, int offset, int length)	It reads characters into a portion of an array.
boolean	ready()	It tells whether this stream is ready to be read.

Example

```
public class InputStreamReaderExample {
   public static void main(String[] args) {
      try  {
         InputStream stream = new FileInputStream("file.txt");
         Reader reader = new InputStreamReader(stream);
         int data = reader.read();
         while (data != -1) {
            System.out.print((char) data);
            data = reader.read();
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

Output:

```
I love my country

The file.txt contains text "I love my country" the InputStreamReader
reads Character by character from the file
```

← Prev Next →

Java PushbackInputStream Class

Java PushbackInputStream class overrides InputStream and provides extra functionality to another input stream. It can unread a byte which is already read and push back one byte.

Class declaration

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

```
public class PushbackInputStream extends FilterInputStream
```

Class Methods

It is used to test if the input stream support mark and reset method.

Method	Description
int available()	It is used to return the 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.
boolean markSupported()	
void mark(int readlimit)	It is used to mark the current position in the input stream.
long skip(long x)	It is used to skip over and discard x bytes of data.
void unread(int b)	It is used to pushes back the byte by copying it to the pushback buffer.
void unread(byte[] b)	It is used to pushes back the array of byte by copying it to the pushback buffer.
void reset()	It is used to reset the input stream.
void close()	It is used to close the input stream.

Example of PushbackInputStream class

```
import java.io.*;
public class InputStreamExample {
public static void main(String[] args)throws Exception{
        String srg = "1##2#34###12";
        byte ary[] = srg.getBytes();
        ByteArrayInputStream array = new ByteArrayInputStream(ary);
        PushbackInputStream push = new PushbackInputStream(array);
        int i;
           while( (i = push.read())!= -1) {
              if(i == '#') {
                 int j;
                 if( (j = push.read()) == '#'){
                     System.out.print("**");
                 }else {
                    push.unread(j);
                    System.out.print((char)i);
                 }
              }else {
                        System.out.print((char)i);
              }
           }
  }
}
```

Output:

```
1**2#34**#12
```

← Prev Next →

Java PushbackReader Class

Java PushbackReader class is a character stream reader. It is used to pushes back a character into stream and overrides the FilterReader class.

Class declaration

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

```
public class PushbackReader extends FilterReader
```

Class Methods

Method	Description
int read()	It is used to read a single character.
void mark(int readAheadLimit)	It is used to mark the present position in a stream.
boolean ready()	It is used to tell whether the stream is ready to be read.
boolean markSupported()	It is used to tell whether the stream supports mark() operation.
long skip(long n)	It is used to skip the character.
void unread (int c)	It is used to pushes back the character by copying it to the pushback buffer.
void unread (char[] cbuf)	It is used to pushes back an array of character by copying it to the pushback buffer.
void reset()	It is used to reset the stream.
void close()	It is used to close the stream.

Example of PushbackReader class

```
import java.io.*;
public class ReaderExample{
```

```
public static void main(String[] args) throws Exception {
    char ary[] = {'1','-','-','2','-','3','4','-','-','-','5','6'};
    CharArrayReader reader = new CharArrayReader(ary);
    PushbackReader push = new PushbackReader(reader);
    int i;
      while( (i = push.read())!= -1) {
        if(i == '-') {
          int j;
          if( (j = push.read()) == '-'){
            System.out.print("#*");
          }else {
            push.unread(j); // push back single character
            System.out.print((char)i);
          }
        }else {
            System.out.print((char)i);
        }
      }
  }
}
```

Output

```
1#*2-34#*-56
```

← Prev Next →

For Videos Join Our Youtube Channel: Join Now

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

Learn Latest Tutorials

Splunk

SPSS

Swagger

Transact-SQL

Tumblr

ReactJS

Regex

Reinforcement Learning

RxJS

Java StringWriter Class

Java StringWriter class is a character stream that collects output from string buffer, which can be used to construct a string. The StringWriter class inherits the Writer class.

In StringWriter class, system resources like network sockets and files are not used, therefore closing the StringWriter is not necessary.

Java StringWriter class declaration

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

```
public class StringWriter extends Writer
```

Methods of StringWriter class

Method	Description
void write(int c)	It is used to write the single character.
void write(String str)	It is used to write the string.
void write(String str, int off, int len)	It is used to write the portion of a string.
void write(char[] cbuf, int off, int len)	It is used to write the portion of an array of characters.
String toString()	It is used to return the buffer current value as a string.
StringBuffer getBuffer()	It is used t return the string buffer.
StringWriter append(char c)	It is used to append the specified character to the writer.
StringWriter append(CharSequence csq)	It is used to append the specified character sequence to the writer.
StringWriter append(CharSequence csq, int start, int end)	It is used to append the subsequence of specified character sequence to the writer.

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

Java StringWriter Example

Let's see the simple example of StringWriter using BufferedReader to read file data from the stream.

```
import java.io.*;
public class StringWriterExample {
    public static void main(String[] args) throws IOException {
        char[] ary = new char[512];
        StringWriter writer = new StringWriter();
        FileInputStream input = null;
        BufferedReader buffer = null;
        input = new FileInputStream("D://testout.txt");
        buffer = new BufferedReader(new InputStreamReader(input, "UTF-8"));
        int x;
        while ((x = buffer.read(ary)) != -1) {
                writer.write(ary, 0, x);
        }
        System.out.println(writer.toString());
        writer.close();
        buffer.close();
    }
}
```

testout.txt:

```
Javatpoint provides tutorial in Java, Spring, Hibernate, Android, PHP etc.
```

Output:

```
Javatpoint provides tutorial in Java, Spring, Hibernate, Android, PHP etc.
```

Java StringReader Class

Java StringReader class is a character stream with string as a source. It takes an input string and changes it into character stream. It inherits Reader class.

In StringReader class, system resources like network sockets and files are not used, therefore closing the StringReader is not necessary.

Java StringReader class declaration

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

```
public class StringReader extends Reader
```

Methods of StringReader class

Method	Description
int read()	It is used to read a single character.
int read(char[] cbuf, int off, int len)	It is used to read a character into a portion of an array.
boolean ready()	It is used to tell whether the stream is ready to be read.
boolean markSupported()	It is used to tell whether the stream support mark() operation.
long skip(long ns)	It is used to skip the specified number of character in a stream
void mark(int readAheadLimit)	It is used to mark the mark the present position in a stream.
void reset()	It is used to reset the stream.
void close()	It is used to close the stream.

Java StringReader Example

```
import java.io.StringReader;
public class StringReaderExample {
    public static void main(String[] args) throws Exception {
        String srg = "Hello Java!! \nWelcome to Javatpoint.";
        StringReader reader = new StringReader(srg);
        int k=0;
            while((k=reader.read())!=-1){
                System.out.print((char)k);
            }
        }
}
```

Output:

```
Hello Java!!
Welcome to Javatpoint.
```

← Prev Next →

Feedback

- Send your Feedback to feedback@javatpoint.com

Java - PipedWriter

The PipedWriter class is used to write java pipe as a stream of characters. This class is used generally for writing text. Generally PipedWriter is connected to a PipedReader and used by different threads.

Constructor

Constructor	Description
PipedWriter()	It creates a piped writer that is not yet connected to a piped reader.
PipedWriter(PipedReader snk)	It creates a piped writer connected to the specified piped reader.

Method

Modifier and Type	Method	Method
void	close()	It closes this piped output stream and releases any system resources associated with this stream.
void	connect(PipedReader snk)	It connects this piped writer to a receiver.
void	flush()	It flushes this output stream and forces any buffered output characters to be written out.
void	write(char[] cbuf, int off, int len)	It writes len characters from the specified character array starting at offset off to this piped output stream.
void	write(int c)	It writes the specified char to the piped output stream.

Example

```
import java.io.PipedReader;
```

```
import java.io.PipedWriter;

public class PipeReaderExample2 {
   public static void main(String[] args) {
      try {

         final PipedReader read = new PipedReader();
         final PipedWriter write = new PipedWriter(read);

         Thread readerThread = new Thread(new Runnable() {
            public void run() {
               try {
                  int data = read.read();
                  while (data != -1) {
                     System.out.print((char) data);
                     data = read.read();
                  }
               } catch (Exception ex) {
               }
            }
         });

         Thread writerThread = new Thread(new Runnable() {
            public void run() {
               try {
                  write.write("I love my country\n".toCharArray());
               } catch (Exception ex) {
               }
            }
         });

         readerThread.start();
         writerThread.start();

      } catch (Exception ex) {
         System.out.println(ex.getMessage());
      }
```

```
    }
}
```

Output:

```
I love my country
```

← Prev Next →

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

Learn Latest Tutorials

Java - PipedReader

The PipedReader class is used to read the contents of a pipe as a stream of characters. This class is used generally to read text.

PipedReader class must be connected to the same PipedWriter and are used by different threads.

Constructor

Constructor	Description
PipedReader(int pipeSize)	It creates a PipedReader so that it is not yet connected and uses the specified pipe size for the pipe's buffer.
PipedReader(PipedWriter src)	It creates a PipedReader so that it is connected to the piped writer src.
PipedReader(PipedWriter src, int pipeSize)	It creates a PipedReader so that it is connected to the piped writer src and uses the specified pipe size for the pipe's buffer.
PipedReader()	It creates a PipedReader so that it is not yet connected.

Method

Modifier and Type	Method	Method
void	close()	It closes this piped stream and releases any system resources associated with the stream.
void	connect(PipedWriter src)	It causes this piped reader to be connected to the piped writer src.
int	read()	It reads the next character of data from this piped stream.
int	read(char[] cbuf, int off, int len)	It reads up to len characters of data from this piped stream into an array of characters.
boolean	ready()	It tells whether this stream is ready to be read.

Example

```
import java.io.PipedReader;
import java.io.PipedWriter;

public class PipeReaderExample2 {
   public static void main(String[] args) {
      try {

         final PipedReader read = new PipedReader();
         final PipedWriter write = new PipedWriter(read);

         Thread readerThread = new Thread(new Runnable() {
            public void run() {
               try {
                  int data = read.read();
                  while (data != -1) {
                     System.out.print((char) data);
                     data = read.read();
                  }
               } catch (Exception ex) {
               }
            }
         });

         Thread writerThread = new Thread(new Runnable() {
            public void run() {
               try {
                  write.write("I love my country\n".toCharArray());
               } catch (Exception ex) {
               }
            }
         });

         readerThread.start();
         writerThread.start();
```

```
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }

    }
}
```

Output:

```
I love my country
```

← Prev Next →

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

Java FilterWriter

Java FilterWriter class is an abstract class which is used to write filtered character streams.

The sub class of the FilterWriter should override some of its methods and it may provide additional methods and fields also.

Fields

Modifier	Type	Field	Description
protected	Writer	out	The underlying character-output stream.

Constructors

Modifier	Constructor	Description
protected	FilterWriter(Writer out)	It creates InputStream class Object

Methods

Modifier and Type	Method	Description
void	close()	It closes the stream, flushing it first.
void	flush()	It flushes the stream.
void	write(char[] cbuf, int off, int len)	It writes a portion of an array of characters.
void	write(int c)	It writes a single character.
void	write(String str, int off, int len)	It writes a portion of a string.

FilterWriter Example

```
import java.io.*;
class CustomFilterWriter extends FilterWriter {
```

```
    CustomFilterWriter(Writer out) {
        super(out);
    }
    public void write(String str) throws IOException {
        super.write(str.toLowerCase());
    }
}
public class FilterWriterExample {
    public static void main(String[] args) {
        try {
            FileWriter fw = new FileWriter("Record.txt");
            CustomFilterWriter filterWriter = new CustomFilterWriter(fw);
            filterWriter.write("I LOVE MY COUNTRY");
            filterWriter.close();
            FileReader fr = new FileReader("record.txt");
            BufferedReader bufferedReader = new BufferedReader(fr);
            int k;
            while ((k = bufferedReader.read()) != -1) {
                System.out.print((char) k);
            }
            bufferedReader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
i love my country
```

While running the current program if the current working directory does not contain the file, a new file is created and CustomFileWriter will write the text "I LOVE MY COUNTRY" in lowercase to the file.

Java FilterReader

Java FilterReader is used to perform filtering operation on reader stream. It is an abstract class for reading filtered character streams.

The FilterReader provides default methods that passes all requests to the contained stream. Subclasses of FilterReader should override some of its methods and may also provide additional methods and fields.

Field

Modifier	Type	Field	Description
protected	Reader	in	The underlying character-input stream.

Constructors

Modifier	Constructor	Description
protected	FilterReader(Reader in)	It creates a new filtered reader.

Method

Modifier and Type	Method	Description
void	close()	It closes the stream and releases any system resources associated with it.
void	mark(int readAheadLimit)	It marks the present position in the stream.
boolean	markSupported()	It tells whether this stream supports the mark() operation.
boolean	ready()	It tells whether this stream is ready to be read.
int	read()	It reads a single character.
int	read(char[] cbuf, int off, int len)	It reads characters into a portion of an array.

void	reset()	It resets the stream.
long	skip(long n)	It skips characters.

Example

In this example, we are using "javaFile123.txt" file which contains "India is my country" text in it. Here, we are converting whitespace with question mark '?'.

```
import java.io.*;
class CustomFilterReader extends FilterReader {
    CustomFilterReader(Reader in) {
        super(in);
    }
    public int read() throws IOException {
        int x = super.read();
        if ((char) x == ' ')
            return ((int) '?');
        else
            return x;
    }
}
public class FilterReaderExample {
    public static void main(String[] args) {
        try  {
            Reader reader = new FileReader("javaFile123.txt");
            CustomFilterReader fr = new CustomFilterReader(reader);
            int i;
            while ((i = fr.read()) != -1) {
                System.out.print((char) i);
            }
            fr.close();
            reader.close();
        } catch (Exception e) {
            e.getMessage();
        }
    }
```

```
}
```

Output:

```
India?is?my?country
```

← Prev Next →

For Videos Join Our Youtube Channel: Join Now

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

Learn Latest Tutorials

Splunk

SPSS

Swagger

Transact-SQL

Java File Class

The File class is an abstract representation of file and directory pathname. A pathname can be either absolute or relative.

The File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc.

Fields

Modifier	Type	Field	Description
static	String	pathSeparator	It is system-dependent path-separator character, represented as a string for convenience.
static	char	pathSeparatorChar	It is system-dependent path-separator character.
static	String	separator	It is system-dependent default name-separator character, represented as a string for convenience.
static	char	separatorChar	It is system-dependent default name-separator character.

Constructors

Constructor	Description
File(File parent, String child)	It creates a new File instance from a parent abstract pathname and a child pathname string.
File(String pathname)	It creates a new File instance by converting the given pathname string into an abstract pathname.
File(String parent, String child)	It creates a new File instance from a parent pathname string and a child pathname string.
File(URI uri)	It creates a new File instance by converting the given file: URI into an abstract pathname.

Useful Methods

Modifier and Type	Method	Description
static File	createTempFile(String prefix, String suffix)	It creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name.
boolean	createNewFile()	It atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.
boolean	canWrite()	It tests whether the application can modify the file denoted by this abstract pathname.String[]
boolean	canExecute()	It tests whether the application can execute the file denoted by this abstract pathname.
boolean	canRead()	It tests whether the application can read the file denoted by this abstract pathname.
boolean	isAbsolute()	It tests whether this abstract pathname is absolute.
boolean	isDirectory()	It tests whether the file denoted by this abstract pathname is a directory.
boolean	isFile()	It tests whether the file denoted by this abstract pathname is a normal file.
String	getName()	It returns the name of the file or directory denoted by this abstract pathname.
String	getParent()	It returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory.
Path	toPath()	It returns a java.nio.file.Path object constructed from the this abstract path.
URI	toURI()	It constructs a file: URI that represents this abstract pathname.

File[]	listFiles()	It returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname
long	getFreeSpace()	It returns the number of unallocated bytes in the partition named by this abstract path name.
String[]	list(FilenameFilter filter)	It returns an array of strings naming the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
boolean	mkdir()	It creates the directory named by this abstract pathname.

Java File Example 1

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

      try {
         File file = new File("javaFile123.txt");
         if (file.createNewFile()) {
            System.out.println("New File is created!");
         } else {
            System.out.println("File already exists.");
         }
      } catch (IOException e) {
         e.printStackTrace();
      }

   }
}
```

Output:

```
New File is created!
```

Java File Example 2

```
import java.io.*;
public class FileDemo2 {
   public static void main(String[] args) {

      String path = "";
      boolean bool = false;
      try {
         // createing  new files
         File file  = new File("testFile1.txt");
         file.createNewFile();
         System.out.println(file);
         // createing new canonical from file object
         File file2 = file.getCanonicalFile();
         // returns true if the file exists
         System.out.println(file2);
         bool = file2.exists();
         // returns absolute pathname
         path = file2.getAbsolutePath();
         System.out.println(bool);
         // if file exists
         if (bool) {
            // prints
            System.out.print(path + " Exists? " + bool);
         }
      } catch (Exception e) {
         // if any error occurs
         e.printStackTrace();
      }
   }
}
```

Output:

```
testFile1.txt
/home/Work/Project/File/testFile1.txt
```

```
true
/home/Work/Project/File/testFile1.txt Exists? true
```

Java File Example 3

```
import java.io.*;
public class FileExample {
public static void main(String[] args) {
    File f=new File("/Users/sonoojaiswal/Documents");
    String filenames[]=f.list();
    for(String filename:filenames){
        System.out.println(filename);
    }
}
}
```

Output:

```
"info.properties"
"info.properties".rtf
.DS_Store
.localized
Alok news
apache-tomcat-9.0.0.M19
apache-tomcat-9.0.0.M19.tar
bestreturn_org.rtf
BIODATA.pages
BIODATA.pdf
BIODATA.png
struts2jars.zip
workspace
```


Java File Example 4

```
import java.io.*;
public class FileExample {
public static void main(String[] args) {
    File dir=new File("/Users/sonoojaiswal/Documents");
    File files[]=dir.listFiles();
    for(File file:files){
        System.out.println(file.getName()+" Can Write: "+file.canWrite()+"
        Is Hidden: "+file.isHidden()+" Length: "+file.length()+" bytes");
    }
}
}
```

Output:

```
"info.properties" Can Write: true Is Hidden: false Length: 15 bytes
"info.properties".rtf Can Write: true Is Hidden: false Length: 385 bytes
.DS_Store Can Write: true Is Hidden: true Length: 36868 bytes
.localized Can Write: true Is Hidden: true Length: 0 bytes
Alok news Can Write: true Is Hidden: false Length: 850 bytes
apache-tomcat-9.0.0.M19 Can Write: true Is Hidden: false Length: 476 bytes
apache-tomcat-9.0.0.M19.tar Can Write: true Is Hidden: false Length: 13711360 bytes
bestreturn_org.rtf Can Write: true Is Hidden: false Length: 389 bytes
BIODATA.pages Can Write: true Is Hidden: false Length: 707985 bytes
BIODATA.pdf Can Write: true Is Hidden: false Length: 69681 bytes
BIODATA.png Can Write: true Is Hidden: false Length: 282125 bytes
workspace Can Write: true Is Hidden: false Length: 1972 bytes
```

← Prev Next →

Java FileDescriptor

FileDescriptor class serves as an handle to the underlying machine-specific structure representing an open file, an open socket, or another source or sink of bytes. The handle can be err, in or out.

The FileDescriptor class is used to create a FileInputStream or FileOutputStream to contain it.

Field

Modifier	Type	Field	Description
static	FileDescriptor	err	A handle to the standard error stream.
static	FileDescriptor	in	A handle to the standard input stream.
static	FileDescriptor	out	A handle to the standard output stream.

Constructors

Constructor	Description
FileDescriptor()	Constructs an (invalid) FileDescriptor object.

Method

Modifier and Type	Method	Description
void	sync()	It force all system buffers to synchronize with the underlying device.
boolean	valid()	It tests if this file descriptor object is valid.

Java FileDescriptor Example

```
import java.io.*;
public class FileDescriptorExample {
   public static void main(String[] args) {
      FileDescriptor fd = null;
```

```
        byte[] b = { 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58 };
        try  {
            FileOutputStream fos = new FileOutputStream("Record.txt");
            FileInputStream fis = new FileInputStream("Record.txt");
            fd = fos.getFD();
            fos.write(b);
            fos.flush();
            fd.sync();// confirms data to be written to the disk
            int value = 0;
            // for every available bytes
            while ((value = fis.read()) != -1) {
                char c = (char) value;// converts bytes to char
                System.out.print(c);
            }
            System.out.println("\nSync() successfully executed!!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
0123456789:
Sync() successfully executed!!
```

Record.txt:

```
0123456789:
```

← Prev Next →

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."

after running the program it will contains

This class is used for reading I love my country and my peoplele.

← Prev Next →

Feedback

- Send your Feedback to feedback@javatpoint.com

Java Scanner

Scanner class in Java is found in the java.util package. Java provides 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 which is whitespace by default. It provides many methods to read and parse various primitive values.

The Java Scanner class is widely used to parse text for strings and primitive types using a regular expression. It is the simplest way to get input in Java. By the help of Scanner in Java, we can get input from the user in primitive types such as int, long, double, byte, float, short, etc.

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

The Java Scanner class provides nextXXX() methods to return the type of value such as nextInt(), nextByte(), nextShort(), next(), nextLine(), nextDouble(), nextFloat(), nextBoolean(), etc. To get a single character from the scanner, you can call next().charAt(0) method which returns a single character.

Java Scanner Class Declaration

```
public final class Scanner
        extends Object
        implements Iterator<String>
```

How to get Java Scanner

To get the instance of Java Scanner which reads input from the user, we need to pass the input stream (System.in) in the constructor of Scanner class. For Example:

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

To get the instance of Java Scanner which parses the strings, we need to pass the strings in the constructor of Scanner class. For Example:

```
Scanner in = new Scanner("Hello Javatpoint");
```

Java Scanner Class Constructors

SN	Constructor	Description
1)	Scanner(File source)	It constructs a new Scanner that produces values scanned from the specified file.
2)	Scanner(File source, String charsetName)	It constructs a new Scanner that produces values scanned from the specified file.
3)	Scanner(InputStream source)	It constructs a new Scanner that produces values scanned from the specified input stream.
4)	Scanner(InputStream source, String charsetName)	It constructs a new Scanner that produces values scanned from the specified input stream.
5)	Scanner(Readable source)	It constructs a new Scanner that produces values scanned from the specified source.
6)	Scanner(String source)	It constructs a new Scanner that produces values scanned from the specified string.
7)	Scanner(ReadableByteChannel source)	It constructs a new Scanner that produces values scanned from the specified channel.
8)	Scanner(ReadableByteChannel source, String charsetName)	It constructs a new Scanner that produces values scanned from the specified channel.
9)	Scanner(Path source)	It constructs a new Scanner that produces values scanned from the specified file.
10)	Scanner(Path source, String charsetName)	It constructs a new Scanner that produces values scanned from the specified file.

Java Scanner Class Methods

The following are the list of Scanner methods:

SN	Modifier & Type	Method	Description
1)	void	close()	It is used to close this scanner.
2)	pattern	delimiter()	It is used to get the Pattern which the Scanner class is currently using to match delimiters.
3)	Stream<MatchResult>	findAll()	It is used to find a stream of match results that match the provided pattern string.
4)	String	findInLine()	It is used to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.
5)	string	findWithinHorizon()	It is used to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.
6)	boolean	hasNext()	It returns true if this scanner has another token in its input.
7)	boolean	hasNextBigDecimal()	It is used to check if the next token in this scanner's input can be interpreted as a BigDecimal using the nextBigDecimal() method or not.
8)	boolean	hasNextBigInteger()	It is used to check if the next token in this scanner's input can be interpreted as a BigDecimal using the nextBigDecimal() method or not.
9)	boolean	hasNextBoolean()	It is used to check if the next token in this scanner's input can be interpreted as a Boolean using the nextBoolean() method or not.
10)	boolean	hasNextByte()	It is used to check if the next token in this scanner's input can be interpreted as a Byte using the nextBigDecimal() method or not.

11)	boolean	hasNextDouble()	It is used to check if the next token in this scanner's input can be interpreted as a BigDecimal using the nextByte() method or not.
12)	boolean	hasNextFloat()	It is used to check if the next token in this scanner's input can be interpreted as a Float using the nextFloat() method or not.
13)	boolean	hasNextInt()	It is used to check if the next token in this scanner's input can be interpreted as an int using the nextInt() method or not.
14)	boolean	hasNextLine()	It is used to check if there is another line in the input of this scanner or not.
15)	boolean	hasNextLong()	It is used to check if the next token in this scanner's input can be interpreted as a Long using the nextLong() method or not.
16)	boolean	hasNextShort()	It is used to check if the next token in this scanner's input can be interpreted as a Short using the nextShort() method or not.
17)	IOException	ioException()	It is used to get the IOException last thrown by this Scanner's readable.
18)	Locale	locale()	It is used to get a Locale of the Scanner class.
19)	MatchResult	match()	It is used to get the match result of the last scanning operation performed by this scanner.
20)	String	next()	It is used to get the next complete token from the scanner which is in use.
21)	BigDecimal	nextBigDecimal()	It scans the next token of the input as a BigDecimal.
22)	BigInteger	nextBigInteger()	It scans the next token of the input as a BigInteger.

23)	boolean	nextBoolean()	It scans the next token of the input into a boolean value and returns that value.
24)	byte	nextByte()	It scans the next token of the input as a byte.
25)	double	nextDouble()	It scans the next token of the input as a double.
26)	float	nextFloat()	It scans the next token of the input as a float.
27)	int	nextInt()	It scans the next token of the input as an Int.
28)	String	nextLine()	It is used to get the input string that was skipped of the Scanner object.
29)	long	nextLong()	It scans the next token of the input as a long.
30)	short	nextShort()	It scans the next token of the input as a short.
31)	int	radix()	It is used to get the default radix of the Scanner use.
32)	void	remove()	It is used when remove operation is not supported by this implementation of Iterator.
33)	Scanner	reset()	It is used to reset the Scanner which is in use.
34)	Scanner	skip()	It skips input that matches the specified pattern, ignoring delimiters
35)	Stream<String>	tokens()	It is used to get a stream of delimiter-separated tokens from the Scanner object which is in use.
36)	String	toString()	It is used to get the string representation of Scanner using.

37)	Scanner	useDelimiter()	It is used to set the delimiting pattern of the Scanner which is in use to the specified pattern.
38)	Scanner	useLocale()	It is used to sets this scanner's locale object to the specified locale.
39)	Scanner	useRadix()	It is used to set the default radix of the Scanner which is in use to the specified radix.

Example 1

Let's see a simple example of Java Scanner where we are getting a single input from the user. Here, we are asking for a string through in.nextLine() method.

```
import java.util.*;
public class ScannerExample {
public static void main(String args[]){
        Scanner in = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = in.nextLine();
        System.out.println("Name is: " + name);
        in.close();
        }
}
```

Output:

```
Enter your name: sonoo jaiswal
Name is: sonoo jaiswal
```

Example 2

```
import java.util.*;
public class ScannerClassExample1 {
    public static void main(String args[]){
        String s = "Hello, This is JavaTpoint.";
```

```
        //Create scanner Object and pass string in it
        Scanner scan = new Scanner(s);
        //Check if the scanner has a token
        System.out.println("Boolean Result: " + scan.hasNext());
        //Print the string
        System.out.println("String: " +scan.nextLine());
        scan.close();
        System.out.println("--------Enter Your Details-------- ");
        Scanner in = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = in.next();
        System.out.println("Name: " + name);
        System.out.print("Enter your age: ");
        int i = in.nextInt();
        System.out.println("Age: " + i);
        System.out.print("Enter your salary: ");
        double d = in.nextDouble();
        System.out.println("Salary: " + d);
        in.close();
        }
}
```

Output:

```
Boolean Result: true
String: Hello, This is JavaTpoint.
-------Enter Your Details---------
Enter your name: Abhishek
Name: Abhishek
Enter your age: 23
Age: 23
Enter your salary: 25000
Salary: 25000.0
```

Example 3

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

```
public class ScannerClassExample2 {
    public static void main(String args[]){
        String str = "Hello/This is JavaTpoint/My name is Abhishek.";
        //Create scanner with the specified String Object
        Scanner scanner = new Scanner(str);
        System.out.println("Boolean Result: "+scanner.hasNextBoolean());
        //Change the delimiter of this scanner
        scanner.useDelimiter("/");
        //Printing the tokenized Strings
        System.out.println("---Tokenizes String---");
       while(scanner.hasNext()){
          System.out.println(scanner.next());
       }
        //Display the new delimiter
        System.out.println("Delimiter used: " +scanner.delimiter());
        scanner.close();
        }
}
```

Output:

```
Boolean Result: false
---Tokenizes String---
Hello
This is JavaTpoint
My name is Abhishek.
Delimiter used: /
```

← Prev Next →

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 Decompressing File

The DeflaterOutputStream and InflaterInputStream classes provide mechanism to compress and decompress 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 decompress the file in the deflate compression format. It provides facility to the other decompression filters, such as GZIPInputStream class.

Example of decompressing 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 DeCompress{
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

<<Prev Next>>

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();
}}
```

download this example

← Prev Next →

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

Learn Latest Tutorials

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.

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.

```
System.out.println("simple message");
System.err.println("error message");
```

Let's see the code to get **input** from console.

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

Do You Know?

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

- How to compress and uncompress the data of a file?

OutputStream vs InputStream

The explanation of OutputStream and InputStream classes are given below:

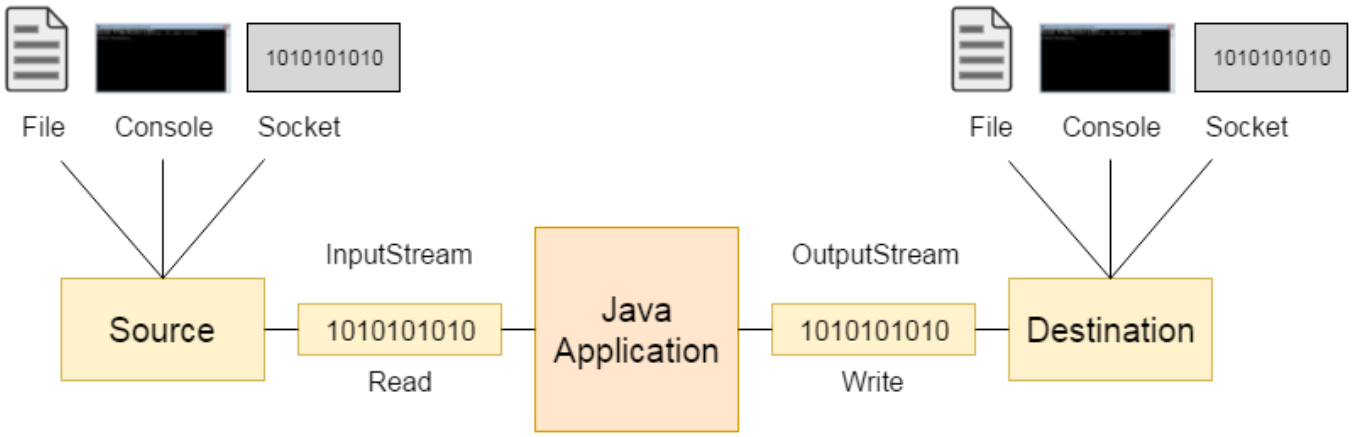
OutputStream

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

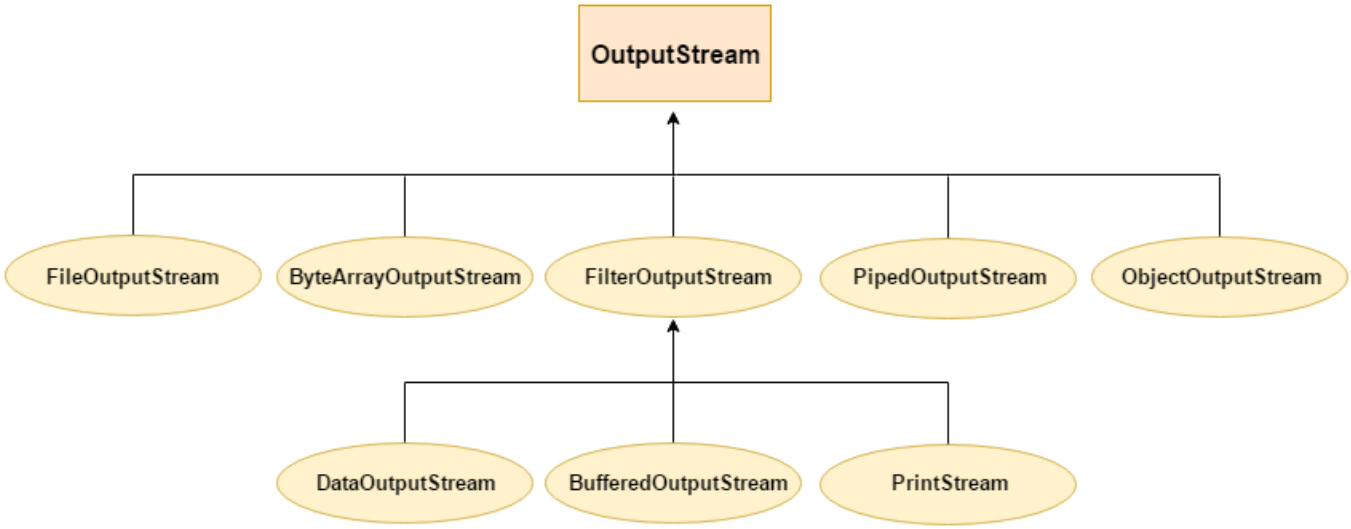
OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

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



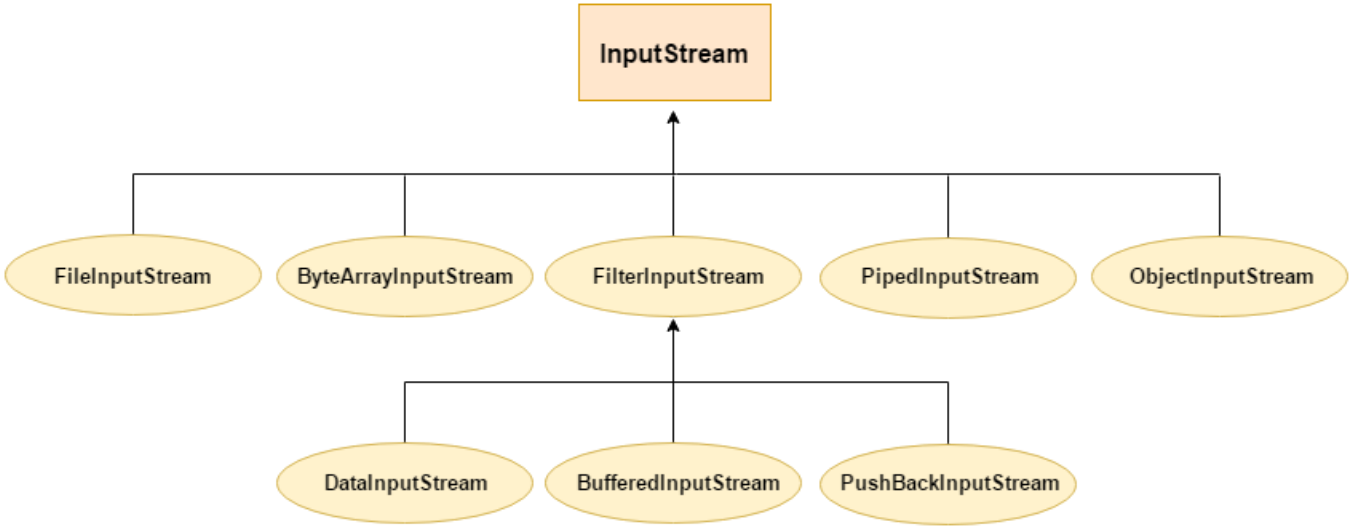
InputStream class

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



← Prev Next →

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

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
```

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.

Java FileOutputStream Example 1: write byte

```
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...
```

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

testout.txt

```
A
```


Java FileOutputStream example 2: write string

```
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 javaTpoint.";
           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);}
      }
```

```
}
```

Output:

```
Success...
```

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

testout.txt

```
Welcome to javaTpoint.
```

← Prev Next →

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

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 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.

Java FileInputStream example 1: 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 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

```
package com.javatpoint;

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 javaTpoint
```

← Prev Next →

Feedback

- Send your Feedback to feedback@javatpoint.com

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:

```
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
```

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

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
```

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()	It tests for the input stream to support the mark and reset methods.

Example of Java BufferedInputStream

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

```
package com.javatpoint;

import java.io.*;
public class BufferedInputStreamExample{
 public static void main(String args[]){
  try{
    FileInputStream fin=new FileInputStream("D:\\testout.txt");
    BufferedInputStream bin=new BufferedInputStream(fin);
    int i;
    while((i=bin.read())!=-1){
     System.out.print((char)i);
    }
    bin.close();
    fin.close();
  }catch(Exception e){System.out.println(e);}
 }
}
```

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

```
javaTpoint
```

Output:

```
javaTpoint
```

← Prev Next →

For Videos Join Our Youtube Channel: Join Now

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

Learn Latest Tutorials

Splunk

SPSS

Swagger

Transact-SQL