java.io

# Class RandomAccessFile

java.lang.Object
    java.io.RandomAccessFile

**All Implemented Interfaces:**

Closeable, DataInput, DataOutput, AutoCloseable

---

```
public class RandomAccessFile
extends Object
implements DataOutput, DataInput, Closeable
```

Instances of this class support both reading and writing to a random access file. A random access file behaves like a large array of bytes stored in the file system. There is a kind of cursor, or index into the implied array, called the *file pointer*; input operations read bytes starting at the file pointer and advance the file pointer past the bytes read. If the random access file is created in read/write mode, then output operations are also available; output operations write bytes starting at the file pointer and advance the file pointer past the bytes written. Output operations that write past the current end of the implied array cause the array to be extended. The file pointer can be read by the `getFilePointer` method and set by the `seek` method.

It is generally true of all the reading routines in this class that if end-of-file is reached before the desired number of bytes has been read, an `EOFException` (which is a kind of `IOException`) is thrown. If any byte cannot be read for any reason other than end-of-file, an `IOException` other than `EOFException` is thrown. In particular, an `IOException` may be thrown if the stream has been closed.

**Since:**

JDK1.0

## Constructor Summary

Constructors

| Constructor and Description |
| --- |
| **`RandomAccessFile(File file, String mode)`**<br>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)`**<br>Creates a random access file stream to read from, and optionally to write to, a file with the specified name. |

## Method Summary

Methods

| Modifier and Type | Method and Description |
| --- | --- |
| `void` | **`close()`**<br>Closes this random access file stream and releases any system resources associated with the stream. |
| `FileChannel` | **`getChannel()`**<br>Returns the unique `FileChannel` object associated with this file. |
| `FileDescriptor` | **`getFD()`**<br>Returns the opaque file descriptor object associated with this stream. |
| `long` | **`getFilePointer()`**<br>Returns the current offset in this file. |

| | |
|---|---|
| long | **length**()<br>Returns the length of this file. |
| int | **read**()<br>Reads a byte of data from this file. |
| int | **read**(byte[] b)<br>Reads up to b.length bytes of data from this file into an array of bytes. |
| int | **read**(byte[] b, int off, int len)<br>Reads up to len bytes of data from this file into an array of bytes. |
| boolean | **readBoolean**()<br>Reads a boolean from this file. |
| byte | **readByte**()<br>Reads a signed eight-bit value from this file. |
| char | **readChar**()<br>Reads a character from this file. |
| double | **readDouble**()<br>Reads a double from this file. |
| float | **readFloat**()<br>Reads a float from this file. |
| void | **readFully**(byte[] b)<br>Reads b.length bytes from this file into the byte array, starting at the current file pointer. |
| void | **readFully**(byte[] b, int off, int len)<br>Reads exactly len bytes from this file into the byte array, starting at the current file pointer. |
| int | **readInt**()<br>Reads a signed 32-bit integer from this file. |
| **String** | **readLine**()<br>Reads the next line of text from this file. |
| long | **readLong**()<br>Reads a signed 64-bit integer from this file. |
| short | **readShort**()<br>Reads a signed 16-bit number from this file. |
| int | **readUnsignedByte**()<br>Reads an unsigned eight-bit number from this file. |
| int | **readUnsignedShort**()<br>Reads an unsigned 16-bit number from this file. |
| **String** | **readUTF**()<br>Reads in a string from this file. |
| void | **seek**(long pos)<br>Sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs. |
| void | **setLength**(long newLength)<br>Sets the length of this file. |
| int | **skipBytes**(int n)<br>Attempts to skip over n bytes of input discarding the skipped bytes. |
| void | **write**(byte[] b)<br>Writes b.length bytes from the specified byte array to this file, starting at the current file pointer. |
| void | **write**(byte[] b, int off, int len)<br>Writes len bytes from the specified byte array starting at offset off to this file. |
| void | **write**(int b)<br>Writes the specified byte to this file. |
| void | **writeBoolean**(boolean v)<br>Writes a boolean to the file as a one-byte value. |
| void | **writeByte**(int v)<br>Writes a byte to the file as a one-byte value. |
| void | **writeBytes**(**String** s) |

| | |
|---|---|
| | Writes the string to the file as a sequence of bytes. |
| void | **writeChar**(int v)<br>Writes a char to the file as a two-byte value, high byte first. |
| void | **writeChars**(String s)<br>Writes a string to the file as a sequence of characters. |
| void | **writeDouble**(double v)<br>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)<br>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 | **writeInt**(int v)<br>Writes an int to the file as four bytes, high byte first. |
| void | **writeLong**(long v)<br>Writes a long to the file as eight bytes, high byte first. |
| void | **writeShort**(int v)<br>Writes a short to the file as two bytes, high byte first. |
| void | **writeUTF**(String str)<br>Writes a string to the file using **modified UTF-8** encoding in a machine-independent manner. |

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

## Constructor Detail

### RandomAccessFile

```
public RandomAccessFile(String name,
                        String mode)
                 throws FileNotFoundException
```

Creates a random access file stream to read from, and optionally to write to, a file with the specified name. A new FileDescriptor object is created to represent the connection to the file.

The mode argument specifies the access mode with which the file is to be opened. The permitted values and their meanings are as specified for the RandomAccessFile(File,String) constructor.

If there is a security manager, its checkRead method is called with the name argument as its argument to see if read access to the file is allowed. If the mode allows writing, the security manager's checkWrite method is also called with the name argument as its argument to see if write access to the file is allowed.

**Parameters:**

name - the system-dependent filename

mode - the access mode

**Throws:**

IllegalArgumentException - if the mode argument is not equal to one of "r", "rw", "rws", or "rwd"

FileNotFoundException - if the mode is "r" but the given string does not denote an existing regular file, or if the mode begins with "rw" but the given string does not denote an existing, writable regular file and a new regular file of that name cannot be created, or if some other error occurs while opening or creating the file

SecurityException - if a security manager exists and its checkRead method denies read access to the file or the mode is "rw" and the security manager's checkWrite method denies write access to the file

**See Also:**

SecurityException, SecurityManager.checkRead(java.lang.String), SecurityManager.checkWrite(java.lang.String)

## RandomAccessFile

```
public RandomAccessFile(File file,
                String mode)
                 throws FileNotFoundException
```

Creates a random access file stream to read from, and optionally to write to, the file specified by the File argument. A new FileDescriptor object is created to represent this file connection.

The mode argument specifies the access mode in which the file is to be opened. The permitted values and their meanings are:

| Value | Meaning |
|---|---|
| "r" | Open for reading only. Invoking any of the write methods of the resulting object will cause an IOException to be thrown. |
| "rw" | Open for reading and writing. If the file does not already exist then an attempt will be made to create it. |
| "rws" | Open for reading and writing, as with "rw", and also require that every update to the file's content or metadata be written synchronously to the underlying storage device. |
| "rwd" | Open for reading and writing, as with "rw", and also require that every update to the file's content be written synchronously to the underlying storage device. |

The "rws" and "rwd" modes work much like the force(boolean) method of the FileChannel class, passing arguments of true and false, respectively, except that they always apply to every I/O operation and are therefore often more efficient. If the file resides on a local storage device then when an invocation of a method of this class returns it is guaranteed that all changes made to the file by that invocation will have been written to that device. This is useful for ensuring that critical information is not lost in the event of a system crash. If the file does not reside on a local device then no such guarantee is made.

The "rwd" mode can be used to reduce the number of I/O operations performed. Using "rwd" only requires updates to the file's content to be written to storage; using "rws" requires updates to both the file's content and its metadata to be written, which generally requires at least one more low-level I/O operation.

If there is a security manager, its checkRead method is called with the pathname of the file argument as its argument to see if read access to the file is allowed. If the mode allows writing, the security manager's checkWrite method is also called with the path argument to see if write access to the file is allowed.

**Parameters:**

file - the file object

mode - the access mode, as described above

**Throws:**

IllegalArgumentException - if the mode argument is not equal to one of "r", "rw", "rws", or "rwd"

FileNotFoundException - if the mode is "r" but the given file object does not denote an existing regular file, or if the mode begins with "rw" but the given file object does not denote an existing, writable regular file and a new regular file of that name cannot be created, or if some other error occurs while opening or creating the file

SecurityException - if a security manager exists and its checkRead method denies read access to the file or the mode is "rw" and the security manager's checkWrite method denies write access to the file

**See Also:**

SecurityManager.checkRead(java.lang.String), SecurityManager.checkWrite(java.lang.String), FileChannel.force(boolean)

## Method Detail

## getFD

```
public final FileDescriptor getFD()
                           throws IOException
```

Returns the opaque file descriptor object associated with this stream.

**Returns:**

the file descriptor object associated with this stream.

**Throws:**

`IOException` - if an I/O error occurs.

**See Also:**

`FileDescriptor`

## getChannel

```
public final FileChannel getChannel()
```

Returns the unique `FileChannel` object associated with this file.

The position of the returned channel will always be equal to this object's file-pointer offset as returned by the `getFilePointer` method. Changing this object's file-pointer offset, whether explicitly or by reading or writing bytes, will change the position of the channel, and vice versa. Changing the file's length via this object will change the length seen via the file channel, and vice versa.

**Returns:**

the file channel associated with this file

**Since:**

1.4

## read

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

Reads a byte of data from this file. The byte is returned as an integer in the range 0 to 255 (`0x00-0x0ff`). This method blocks if no input is yet available.

Although `RandomAccessFile` is not a subclass of `InputStream`, this method behaves in exactly the same way as the `InputStream.read()` method of `InputStream`.

**Returns:**

the next byte of data, or `-1` if the end of the file has been reached.

**Throws:**

`IOException` - if an I/O error occurs. Not thrown if end-of-file has been reached.

## read

```
public int read(byte[] b,
       int off,
       int len)
         throws IOException
```

Reads up to `len` bytes of data from this file into an array of bytes. This method blocks until at least one byte of input is available.

Although `RandomAccessFile` is not a subclass of `InputStream`, this method behaves in exactly the same way as the `InputStream.read(byte[], int, int)` method of `InputStream`.

**Parameters:**

`b` - the buffer into which the data is read.

`off` - the start offset in array `b` at which the data is written.

`len` - the maximum number of bytes read.

**Returns:**

the total number of bytes read into the buffer, or `-1` if there is no more data because the end of the file has been reached.

**Throws:**

`IOException` - If the first byte cannot be read for any reason other than end of file, or if the random access file has been closed, or if some other I/O error occurs.

`NullPointerException` - If `b` is `null`.

`IndexOutOfBoundsException` - If `off` is negative, `len` is negative, or `len` is greater than `b.length - off`

## read

```
public int read(byte[] b)
         throws IOException
```

Reads up to `b.length` bytes of data from this file into an array of bytes. This method blocks until at least one byte of input is available.

Although `RandomAccessFile` is not a subclass of `InputStream`, this method behaves in exactly the same way as the `InputStream.read(byte[])` method of `InputStream`.

**Parameters:**

`b` - the buffer into which the data is read.

**Returns:**

the total number of bytes read into the buffer, or `-1` if there is no more data because the end of this file has been reached.

**Throws:**

`IOException` - If the first byte cannot be read for any reason other than end of file, or if the random access file has been closed, or if some other I/O error occurs.

`NullPointerException` - If `b` is `null`.

## readFully

```
public final void readFully(byte[] b)
                     throws IOException
```

Reads `b.length` bytes from this file into the byte array, starting at the current file pointer. This method reads repeatedly from the file until the requested number of bytes are read. This method blocks until the requested number of bytes are read, the end of the stream is detected, or an exception is thrown.

**Specified by:**

`readFully` in interface `DataInput`

**Parameters:**

`b` - the buffer into which the data is read.

**Throws:**

`EOFException` - if this file reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

## readFully

```
public final void readFully(byte[] b,
             int off,
             int len)
                    throws IOException
```

Reads exactly `len` bytes from this file into the byte array, starting at the current file pointer. This method reads repeatedly from the file until the requested number of bytes are read. This method blocks until the requested number of bytes are read, the end of the stream is detected, or an exception is thrown.

**Specified by:**

`readFully` in interface `DataInput`

**Parameters:**

`b` - the buffer into which the data is read.

`off` - the start offset of the data.

`len` - the number of bytes to read.

**Throws:**

`EOFException` - if this file reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

## skipBytes

```
public int skipBytes(int n)
              throws IOException
```

Attempts to skip over `n` bytes of input discarding the skipped bytes.

This method may skip over some smaller number of bytes, possibly zero. This may result from any of a number of conditions; reaching end of file before `n` bytes have been skipped is only one possibility. This method never throws an `EOFException`. The actual number of bytes skipped is returned. If `n` is negative, no bytes are skipped.

**Specified by:**

`skipBytes` in interface `DataInput`

**Parameters:**

`n` - the number of bytes to be skipped.

**Returns:**

the actual number of bytes skipped.

**Throws:**

`IOException` - if an I/O error occurs.

## write

```
public void write(int b)
           throws IOException
```

Writes the specified byte to this file. The write starts at the current file pointer.

**Specified by:**

`write` in interface `DataOutput`

**Parameters:**

`b` - the `byte` to be written.

**Throws:**

`IOException` - if an I/O error occurs.

## write

```
public void write(byte[] b)
           throws IOException
```

Writes `b.length` bytes from the specified byte array to this file, starting at the current file pointer.

**Specified by:**

`write` in interface `DataOutput`

**Parameters:**

`b` - the data.

**Throws:**

`IOException` - if an I/O error occurs.

## write

```
public void write(byte[] b,
         int off,
         int len)
           throws IOException
```

Writes `len` bytes from the specified byte array starting at offset `off` to this file.

**Specified by:**

`write` in interface `DataOutput`

**Parameters:**

`b` - the data.

`off` - the start offset in the data.

`len` - the number of bytes to write.

**Throws:**

`IOException` - if an I/O error occurs.

## getFilePointer

```
public long getFilePointer()
                    throws IOException
```

Returns the current offset in this file.

**Returns:**

the offset from the beginning of the file, in bytes, at which the next read or write occurs.

**Throws:**

`IOException` - if an I/O error occurs.

## seek

```
public void seek(long pos)
          throws IOException
```

Sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs. The offset may be set beyond the end of the file. Setting the offset beyond the end of the file does not change the file length. The

file length will change only by writing after the offset has been set beyond the end of the file.

**Parameters:**

`pos` - the offset position, measured in bytes from the beginning of the file, at which to set the file pointer.

**Throws:**

`IOException` - if `pos` is less than `0` or if an I/O error occurs.

## length

```
public long length()
            throws IOException
```

Returns the length of this file.

**Returns:**

the length of this file, measured in bytes.

**Throws:**

`IOException` - if an I/O error occurs.

## setLength

```
public void setLength(long newLength)
               throws IOException
```

Sets the length of this file.

If the present length of the file as returned by the `length` method is greater than the `newLength` argument then the file will be truncated. In this case, if the file offset as returned by the `getFilePointer` method is greater than `newLength` then after this method returns the offset will be equal to `newLength`.

If the present length of the file as returned by the `length` method is smaller than the `newLength` argument then the file will be extended. In this case, the contents of the extended portion of the file are not defined.

**Parameters:**

`newLength` - The desired length of the file

**Throws:**

`IOException` - If an I/O error occurs

**Since:**

1.2

## close

```
public void close()
           throws IOException
```

Closes this random access file stream and releases any system resources associated with the stream. A closed random access file cannot perform input or output operations and cannot be reopened.

If this file has an associated channel then the channel is closed as well.

**Specified by:**

`close` in interface `Closeable`

**Specified by:**

`close` in interface `AutoCloseable`

**Throws:**

IOException - if an I/O error occurs.

## readBoolean

```
public final boolean readBoolean()
                           throws IOException
```

Reads a boolean from this file. This method reads a single byte from the file, starting at the current file pointer. A value of 0 represents false. Any other value represents true. This method blocks until the byte is read, the end of the stream is detected, or an exception is thrown.

**Specified by:**

readBoolean in interface DataInput

**Returns:**

the boolean value read.

**Throws:**

EOFException - if this file has reached the end.

IOException - if an I/O error occurs.

## readByte

```
public final byte readByte()
                    throws IOException
```

Reads a signed eight-bit value from this file. This method reads a byte from the file, starting from the current file pointer. If the byte read is b, where 0 <= b <= 255, then the result is:

```
(byte)(b)
```

This method blocks until the byte is read, the end of the stream is detected, or an exception is thrown.

**Specified by:**

readByte in interface DataInput

**Returns:**

the next byte of this file as a signed eight-bit byte.

**Throws:**

EOFException - if this file has reached the end.

IOException - if an I/O error occurs.

## readUnsignedByte

```
public final int readUnsignedByte()
                           throws IOException
```

Reads an unsigned eight-bit number from this file. This method reads a byte from this file, starting at the current file pointer, and returns that byte.

This method blocks until the byte is read, the end of the stream is detected, or an exception is thrown.

**Specified by:**

readUnsignedByte in interface DataInput

**Returns:**

the next byte of this file, interpreted as an unsigned eight-bit number.

**Throws:**

`EOFException` - if this file has reached the end.

`IOException` - if an I/O error occurs.

## readShort

```
public final short readShort()
                      throws IOException
```

Reads a signed 16-bit number from this file. The method reads two bytes from this file, starting at the current file pointer. If the two bytes read, in order, are `b1` and `b2`, where each of the two values is between `0` and `255`, inclusive, then the result is equal to:

```
(short)((b1 << 8) | b2)
```

This method blocks until the two bytes are read, the end of the stream is detected, or an exception is thrown.

**Specified by:**

`readShort` in interface `DataInput`

**Returns:**

the next two bytes of this file, interpreted as a signed 16-bit number.

**Throws:**

`EOFException` - if this file reaches the end before reading two bytes.

`IOException` - if an I/O error occurs.

## readUnsignedShort

```
public final int readUnsignedShort()
                           throws IOException
```

Reads an unsigned 16-bit number from this file. This method reads two bytes from the file, starting at the current file pointer. If the bytes read, in order, are `b1` and `b2`, where `0 <= b1, b2 <= 255`, then the result is equal to:

```
(b1 << 8) | b2
```

This method blocks until the two bytes are read, the end of the stream is detected, or an exception is thrown.

**Specified by:**

`readUnsignedShort` in interface `DataInput`

**Returns:**

the next two bytes of this file, interpreted as an unsigned 16-bit integer.

**Throws:**

`EOFException` - if this file reaches the end before reading two bytes.

`IOException` - if an I/O error occurs.

## readChar

```
public final char readChar()
                    throws IOException
```

Reads a character from this file. This method reads two bytes from the file, starting at the current file pointer. If the bytes read, in order, are `b1` and `b2`, where `0 <= b1, b2 <= 255`, then the result is equal to:

```
(char)((b1 << 8) | b2)
```

This method blocks until the two bytes are read, the end of the stream is detected, or an exception is thrown.

**Specified by:**

`readChar` in interface `DataInput`

**Returns:**

the next two bytes of this file, interpreted as a `char`.

**Throws:**

`EOFException` - if this file reaches the end before reading two bytes.

`IOException` - if an I/O error occurs.

## readInt

```
public final int readInt()
                  throws IOException
```

Reads a signed 32-bit integer from this file. This method reads 4 bytes from the file, starting at the current file pointer. If the bytes read, in order, are `b1`, `b2`, `b3`, and `b4`, where `0 <= b1, b2, b3, b4 <= 255`, then the result is equal to:

```
(b1 << 24) | (b2 << 16) + (b3 << 8) + b4
```

This method blocks until the four bytes are read, the end of the stream is detected, or an exception is thrown.

**Specified by:**

`readInt` in interface `DataInput`

**Returns:**

the next four bytes of this file, interpreted as an `int`.

**Throws:**

`EOFException` - if this file reaches the end before reading four bytes.

`IOException` - if an I/O error occurs.

## readLong

```
public final long readLong()
                    throws IOException
```

Reads a signed 64-bit integer from this file. This method reads eight bytes from the file, starting at the current file pointer. If the bytes read, in order, are `b1`, `b2`, `b3`, `b4`, `b5`, `b6`, `b7`, and `b8`, where:

```
0 <= b1, b2, b3, b4, b5, b6, b7, b8 <=255,
```

then the result is equal to:

```
((long)b1 << 56) + ((long)b2 << 48)
+ ((long)b3 << 40) + ((long)b4 << 32)
+ ((long)b5 << 24) + ((long)b6 << 16)
+ ((long)b7 << 8) + b8
```

This method blocks until the eight bytes are read, the end of the stream is detected, or an exception is thrown.

**Specified by:**

`readLong` in interface `DataInput`

**Returns:**

the next eight bytes of this file, interpreted as a `long`.

**Throws:**

`EOFException` - if this file reaches the end before reading eight bytes.

`IOException` - if an I/O error occurs.

## readFloat

```
public final float readFloat()
                       throws IOException
```

Reads a `float` from this file. This method reads an `int` value, starting at the current file pointer, as if by the `readInt` method and then converts that `int` to a `float` using the `intBitsToFloat` method in class `Float`.

This method blocks until the four bytes are read, the end of the stream is detected, or an exception is thrown.

**Specified by:**

`readFloat` in interface `DataInput`

**Returns:**

the next four bytes of this file, interpreted as a `float`.

**Throws:**

`EOFException` - if this file reaches the end before reading four bytes.

`IOException` - if an I/O error occurs.

**See Also:**

`readInt()`, `Float.intBitsToFloat(int)`

## readDouble

```
public final double readDouble()
                         throws IOException
```

Reads a `double` from this file. This method reads a `long` value, starting at the current file pointer, as if by the `readLong` method and then converts that `long` to a `double` using the `longBitsToDouble` method in class `Double`.

This method blocks until the eight bytes are read, the end of the stream is detected, or an exception is thrown.

**Specified by:**

`readDouble` in interface `DataInput`

**Returns:**

the next eight bytes of this file, interpreted as a `double`.

**Throws:**

`EOFException` - if this file reaches the end before reading eight bytes.

`IOException` - if an I/O error occurs.

**See Also:**

`readLong()`, `Double.longBitsToDouble(long)`

## readLine

```
public final String readLine()
                       throws IOException
```

Reads the next line of text from this file. This method successively reads bytes from the file, starting at the current file pointer, until it reaches a line terminator or the end of the file. Each byte is converted into a character by taking the byte's value for the lower eight bits of the character and setting the high eight bits of the character to zero. This method does not, therefore, support the full Unicode character set.

A line of text is terminated by a carriage-return character (`'\r'`), a newline character (`'\n'`), a carriage-return character immediately followed by a newline character, or the end of the file. Line-terminating characters are discarded and are not included as part of the string returned.

This method blocks until a newline character is read, a carriage return and the byte following it are read (to see if it is a newline), the end of the file is reached, or an exception is thrown.

**Specified by:**

`readLine` in interface `DataInput`

**Returns:**

the next line of text from this file, or null if end of file is encountered before even one byte is read.

**Throws:**

`IOException` - if an I/O error occurs.

## readUTF

```
public final String readUTF()
                     throws IOException
```

Reads in a string from this file. The string has been encoded using a modified UTF-8 format.

The first two bytes are read, starting from the current file pointer, as if by `readUnsignedShort`. This value gives the number of following bytes that are in the encoded string, not the length of the resulting string. The following bytes are then interpreted as bytes encoding characters in the modified UTF-8 format and are converted into characters.

This method blocks until all the bytes are read, the end of the stream is detected, or an exception is thrown.

**Specified by:**

`readUTF` in interface `DataInput`

**Returns:**

a Unicode string.

**Throws:**

`EOFException` - if this file reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

`UTFDataFormatException` - if the bytes do not represent valid modified UTF-8 encoding of a Unicode string.

**See Also:**

`readUnsignedShort()`

## writeBoolean

```
public final void writeBoolean(boolean v)
                        throws IOException
```

Writes a `boolean` to the file as a one-byte value. The value `true` is written out as the value `(byte)1`; the value `false` is written out as the value `(byte)0`. The write starts at the current position of the file pointer.

**Specified by:**

`writeBoolean` in interface `DataOutput`

**Parameters:**

`v` - a `boolean` value to be written.

**Throws:**

`IOException` - if an I/O error occurs.

### writeByte

```
public final void writeByte(int v)
                     throws IOException
```

Writes a `byte` to the file as a one-byte value. The write starts at the current position of the file pointer.

**Specified by:**

`writeByte` in interface `DataOutput`

**Parameters:**

`v` - a `byte` value to be written.

**Throws:**

`IOException` - if an I/O error occurs.

### writeShort

```
public final void writeShort(int v)
                      throws IOException
```

Writes a `short` to the file as two bytes, high byte first. The write starts at the current position of the file pointer.

**Specified by:**

`writeShort` in interface `DataOutput`

**Parameters:**

`v` - a `short` to be written.

**Throws:**

`IOException` - if an I/O error occurs.

### writeChar

```
public final void writeChar(int v)
                     throws IOException
```

Writes a `char` to the file as a two-byte value, high byte first. The write starts at the current position of the file pointer.

**Specified by:**

`writeChar` in interface `DataOutput`

**Parameters:**

`v` - a `char` value to be written.

**Throws:**

`IOException` - if an I/O error occurs.

### writeInt

```
public final void writeInt(int v)
                    throws IOException
```

Writes an `int` to the file as four bytes, high byte first. The write starts at the current position of the file pointer.

**Specified by:**

`writeInt` in interface `DataOutput`

**Parameters:**

v - an int to be written.

**Throws:**

IOException - if an I/O error occurs.

## writeLong

```
public final void writeLong(long v)
                     throws IOException
```

Writes a long to the file as eight bytes, high byte first. The write starts at the current position of the file pointer.

**Specified by:**

writeLong in interface DataOutput

**Parameters:**

v - a long to be written.

**Throws:**

IOException - if an I/O error occurs.

## writeFloat

```
public final void writeFloat(float v)
                      throws IOException
```

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. The write starts at the current position of the file pointer.

**Specified by:**

writeFloat in interface DataOutput

**Parameters:**

v - a float value to be written.

**Throws:**

IOException - if an I/O error occurs.

**See Also:**

Float.floatToIntBits(float)

## writeDouble

```
public final void writeDouble(double v)
                       throws IOException
```

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. The write starts at the current position of the file pointer.

**Specified by:**

writeDouble in interface DataOutput

**Parameters:**

v - a double value to be written.

**Throws:**

IOException - if an I/O error occurs.

**See Also:**

`Double.doubleToLongBits(double)`

## writeBytes

```
public final void writeBytes(String s)
                      throws IOException
```

Writes the string to the file as a sequence of bytes. Each character in the string is written out, in sequence, by discarding its high eight bits. The write starts at the current position of the file pointer.

**Specified by:**

`writeBytes` in interface `DataOutput`

**Parameters:**

`s` - a string of bytes to be written.

**Throws:**

`IOException` - if an I/O error occurs.

## writeChars

```
public final void writeChars(String s)
                      throws IOException
```

Writes a string to the file as a sequence of characters. Each character is written to the data output stream as if by the `writeChar` method. The write starts at the current position of the file pointer.

**Specified by:**

`writeChars` in interface `DataOutput`

**Parameters:**

`s` - a `String` value to be written.

**Throws:**

`IOException` - if an I/O error occurs.

**See Also:**

`writeChar(int)`

## writeUTF

```
public final void writeUTF(String str)
                    throws IOException
```

Writes a string to the file using modified UTF-8 encoding in a machine-independent manner.

First, two bytes are written to the file, starting at the current file pointer, as if by the `writeShort` method giving the number of bytes to follow. This value is the number of bytes actually written out, not the length of the string. Following the length, each character of the string is output, in sequence, using the modified UTF-8 encoding for each character.

**Specified by:**

`writeUTF` in interface `DataOutput`

**Parameters:**

`str` - a string to be written.

**Throws:**

`IOException` - if an I/O error occurs.

Submit a bug or feature

For further API reference and developer documentation, see Java SE Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.