public abstract class

Summary: Ctors | Methods | Inherited Methods | [Expand All]

**Added in API level 1**

# InputStream

extends Object
implements Closeable

java.lang.Object
↳java.io.InputStream

- Known Direct Subclasses
  AssetManager.AssetInputStream, BackupDataInputStream, ByteArrayInputStream, ChunkedInputStream, ContentLengthInputStream, EofSensorInputStream, FileInputStream, FilterInputStream, IdentityInputStream, ObjectInputStream, PipedInputStream, SequenceInputStream, StringBufferInputStream

- Known Indirect Subclasses
  AssetFileDescriptor.AutoCloseInputStream, Base64InputStream, BufferedInputStream, CheckedInputStream, CipherInputStream, DataInputStream, DeflaterInputStream, DigestInputStream, GZIPInputStream, InflaterInputStream, JarInputStream, LineNumberInputStream, ParcelFileDescriptor.AutoCloseInputStream, PushbackInputStream, ZipInputStream

## Class Overview

A readable source of bytes.

Most clients will use input streams that read data from the file system (`FileInputStream (/reference/java/io/FileInputStream.html)`), the network (`getInputStream() (/reference/java/net/Socket.html#getInputStream())`/`getInputStream() (/reference/java/net/URLConnection.html#getInputStream())`), or from an in-memory byte array (`ByteArrayInputStream (/reference/java/io/ByteArrayInputStream.html)`).

Use `InputStreamReader (/reference/java/io/InputStreamReader.html)` to adapt a byte stream like this one into a character stream.

Most clients should wrap their input stream with `BufferedInputStream (/reference/java/io/BufferedInputStream.html)`. Callers that do only bulk reads may omit buffering.

Some implementations support marking a position in the input stream

and resetting back to this position later. Implementations that don't return false from `markSupported()` (/reference/java/io/InputStream.html#markSupported()) and throw an `IOException` (/reference/java/io/IOException.html) when `reset()` (/reference/java/io/InputStream.html#reset()) is called.

### Subclassing InputStream

Subclasses that decorate another input stream should consider subclassing `FilterInputStream`, which delegates all calls to the source input stream.

All input stream subclasses should override **both** `read()` (/reference/java/io/InputStream.html#read()) and `read(byte[],int,int)` (/reference/java/io/InputStream.html#read(byte[], int, int)). The three argument overload is necessary for bulk access to the data. This is much more efficient than byte-by-byte access.

**See Also**

`OutputStream`

## Summary

**Public Constructors**

InputStream ()
This constructor does nothing.

**Public Methods**

| | |
|---|---|
| | available () |
| int | Returns an estimated number of bytes that can be read or skipped without blocking for more input. |
| void | close () Closes this stream. |
| void | mark (int readlimit) Sets a mark position in this InputStream. |
| | markSupported () |
| boolean | Indicates whether this stream supports the `mark()` and `reset()` methods. |
| | read (byte[] buffer) |
| int | Equivalent to `read(buffer, 0, buffer.length)`. |

| | |
|---|---|
| abstract int | read ()<br>Reads a single byte from this stream and returns it as an integer in the range from 0 to 255. |
| int | read (byte[] buffer, int byteOffset, int byteCount)<br>Reads up to `byteCount` bytes from this stream and stores them in the byte array `buffer` starting at `byteOffset`. |
| synchronized void | reset ()<br>Resets this stream to the last marked location. |
| long | skip (long byteCount)<br>Skips at most n bytes in this stream. |

**Inherited Methods** [Expand]

- From class java.lang.Object
- From interface java.io.Closeable
- From interface java.lang.AutoCloseable

## Public Constructors

public **InputStream** ()

Added in API level 1

This constructor does nothing. It is provided for signature compatibility.

## Public Methods

public int **available** ()

Added in API level 1

Returns an estimated number of bytes that can be read or skipped without blocking for more input.

Note that this method provides such a weak guarantee that it is not very useful in practice.

Firstly, the guarantee is "without blocking for more input" rather than "without blocking": a read may still block waiting for I/O to complete — the guarantee is merely that it won't have to wait indefinitely for data to be written. The result of this method should not be used as a license to do I/O on a thread that shouldn't be blocked.

Secondly, the result is a conservative estimate and may be

significantly smaller than the actual number of bytes available. In particular, an implementation that always returns 0 would be correct. In general, callers should only use this method if they'd be satisfied with treating the result as a boolean yes or no answer to the question "is there definitely data ready?".

Thirdly, the fact that a given number of bytes is "available" does not guarantee that a read or skip will actually read or skip that many bytes: they may read or skip fewer.

It is particularly important to realize that you *must not* use this method to size a container and assume that you can read the entirety of the stream without needing to resize the container. Such callers should probably write everything they read to a `ByteArrayOutputStream` (/reference/java/io/ByteArrayOutputStream.html) and convert that to a byte array. Alternatively, if you're reading from a file, `length()` (/reference/java/io/File.html#length()) returns the current length of the file (though assuming the file's length can't change may be incorrect, reading a file is inherently racy).

The default implementation of this method in `InputStream` always returns 0. Subclasses should override this method if they are able to indicate the number of bytes available.

**Returns**

the estimated number of bytes available

**Throws**

| | |
|---|---|
| *IOException* | if this stream is closed or an error occurs |

## public void **close** ()

Added in API level 1

Closes this stream. Concrete implementations of this class should free any resources during close. This implementation does nothing.

**Throws**

| | |
|---|---|
| *IOException* | if an error occurs while closing this stream. |

## public void **mark** (int readlimit)

Added in API level 1

Sets a mark position in this InputStream. The parameter `readlimit` indicates how many bytes can be read before the mark is invalidated. Sending `reset()` will reposition the stream back to the marked position provided `readLimit` has not been surpassed.

This default implementation does nothing and concrete subclasses

must provide their own implementation.

**Parameters**

| | |
|---|---|
| *readlimit* | the number of bytes that can be read from this stream before the mark is invalidated. |

**See Also**

`markSupported()`
`reset()`

### public boolean **markSupported** ()

Added in API level 1

Indicates whether this stream supports the `mark()` and `reset()` methods. The default implementation returns `false`.

**Returns**

always `false`.

**See Also**

`mark(int)`
`reset()`

### public int **read** (byte[] buffer)

Added in API level 1

Equivalent to `read(buffer, 0, buffer.length)`.

**Throws**

*IOException*

### public abstract int **read** ()

Added in API level 1

Reads a single byte from this stream and returns it as an integer in the range from 0 to 255. Returns -1 if the end of the stream has been reached. Blocks until one byte has been read, the end of the source stream is detected or an exception is thrown.

**Throws**

| | |
|---|---|
| *IOException* | if the stream is closed or another IOException occurs. |

### public int **read** (byte[] buffer, int byteOffset, int byteCount)

Added in API level 1

Reads up to `byteCount` bytes from this stream and stores them in the byte array `buffer` starting at `byteOffset`. Returns the number of bytes actually read or -1 if the end of the stream has been

reached.

**Throws**

| | |
|---|---|
| *IndexOutOfBoundsException* | if `byteOffset < 0 \|\| byteCount < 0 \|\| byteOffset + byteCount > buffer.length`. |
| *IOException* | if the stream is closed or another IOException occurs. |

## public synchronized void **reset** ()

Added in API level 1

Resets this stream to the last marked location. Throws an `IOException` if the number of bytes read since the mark has been set is greater than the limit provided to `mark`, or if no mark has been set.

This implementation always throws an `IOException` and concrete subclasses should provide the proper implementation.

**Throws**

| | |
|---|---|
| *IOException* | if this stream is closed or another IOException occurs. |

## public long **skip** (long byteCount)

Added in API level 1

Skips at most n bytes in this stream. This method does nothing and returns 0 if n is negative, but some subclasses may throw.

Note the "at most" in the description of this method: this method may choose to skip fewer bytes than requested. Callers should *always* check the return value.

This default implementation reads bytes into a temporary buffer. Concrete subclasses should provide their own implementation.

**Parameters**

| | |
|---|---|
| *byteCount* | the number of bytes to skip. |

**Returns**

the number of bytes actually skipped.

**Throws**

| | |
|---|---|
| *IOException* | if this stream is closed or another IOException occurs. |