CS360 Lecture 15
Networking

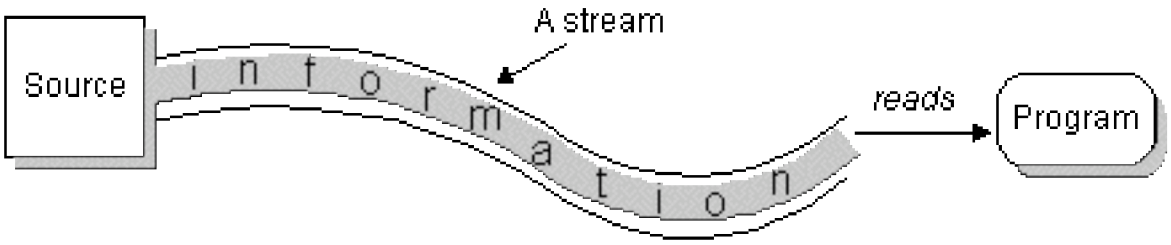
Thursday, March 30, 2004

Reading

Networking: Chapter 18

I/O Streams

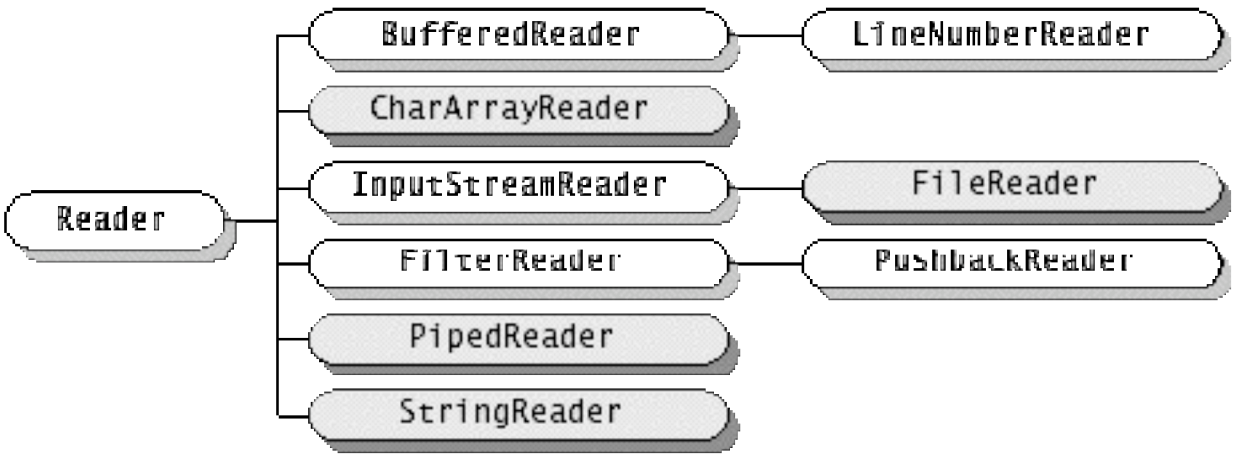
Java programs send and receive information using streams.

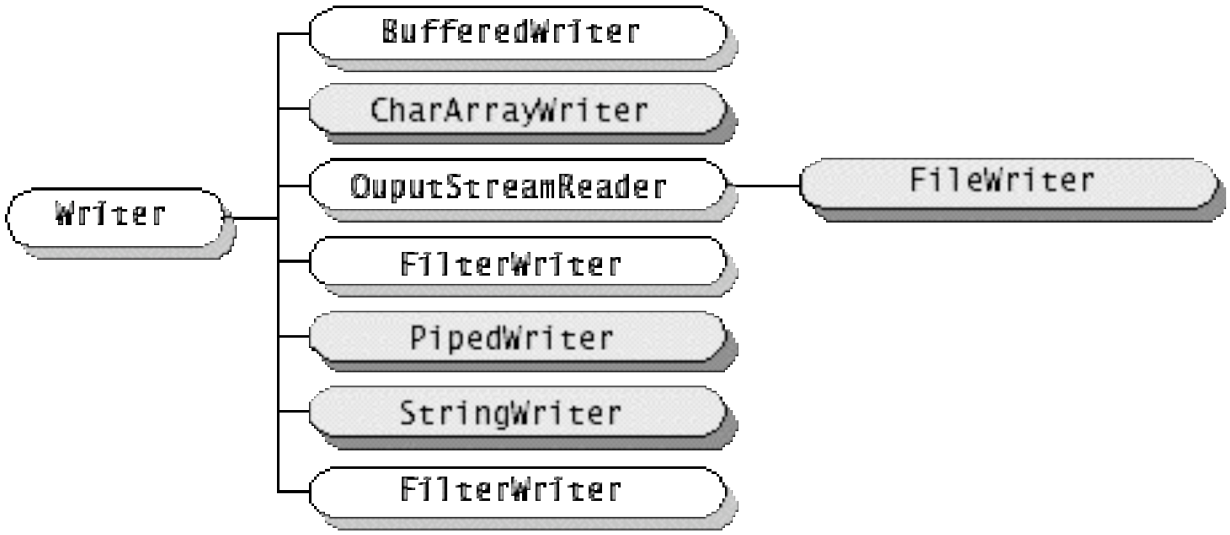


The Java API provides two types of streams in the java.io package. These are the character and byte streams.

Character Streams

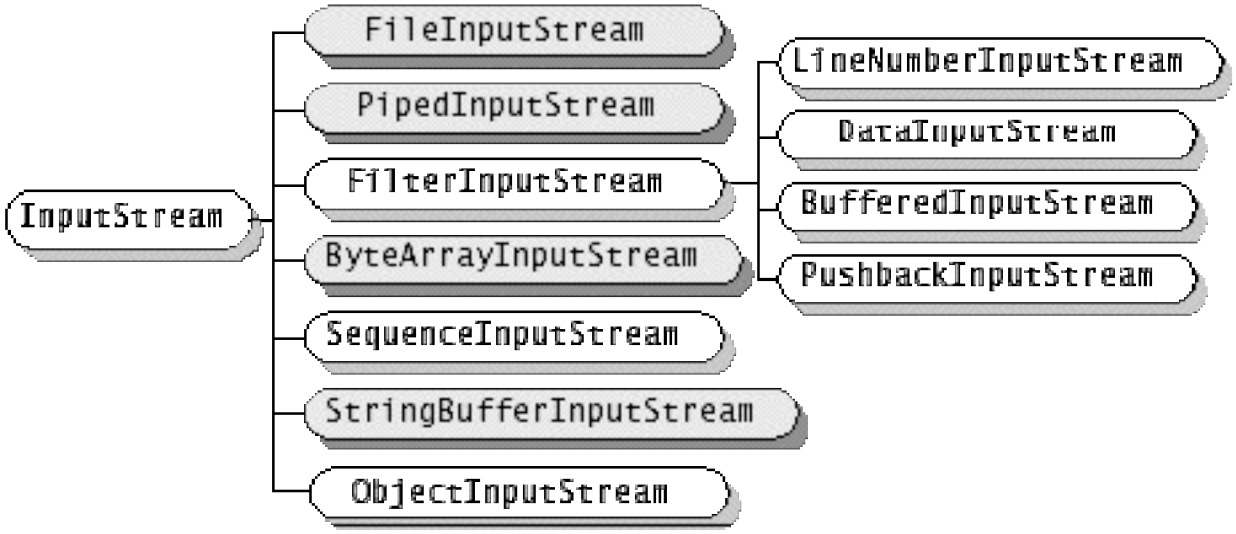
You should use character streams for reading and writing all textual data that you use in your programs. The reason is that character streams read and write 16-bit characters. These cover all the Unicode character set.

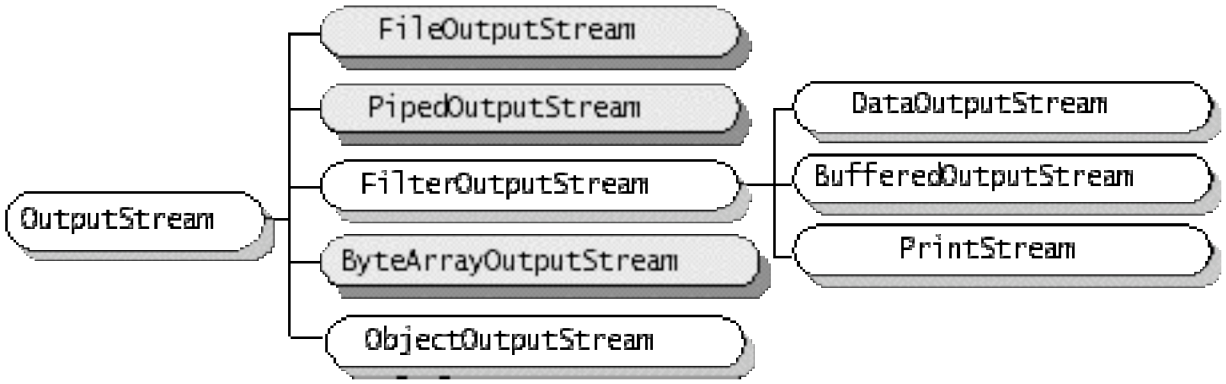




Byte Streams

These read and write 8-bit bytes at a time. These streams are typically used to read and write binary data such as video and audio.





All the streams are automatically opened as soon as they are created. You can close a stream using its close method or the garbage collector will close it once it is no longer referenced.

The following table is taken from the Java tutorial at java.sun.com

Type of I/O	Streams	Description

Memory	`CharArrayReader`
	`CharArrayWriter`
	`ByteArrayInputStream`
	`ByteArrayOutputStream`
		Use these streams to read from and write to memory. You create these streams on an existing array and then use the read and write methods to read from or write to the array.

	`StringReader`
	`StringWriter`
	`StringBufferInputStream`
		`Use StringReader` to read characters from a `String` in memory. Use `StringWriter` to write to a `String`. `StringWriter` collects the characters written to it in a `StringBuffer`, which can then be converted to a `String`. `StringBufferInputStream` is similar to `StringReader`, except that it reads bytes from a `StringBuffer`.

Pipe	`PipedReader`
	`PipedWriter`
	`PipedInputStream`
	`PipedOutputStream`
		Implement the input and output components of a pipe. Pipes are used to channel the output from one thread into the input of another.

File	`FileReader`
	`FileWriter`
	`FileInputStream`
	`FileOutputStream`
		Collectively called file streams, these streams are used to read from or write to a file on the native file system.

Concatenation	*N/A* `SequenceInputStream`	Concatenates multiple input streams into one input stream.

Object Serialization	*N/A*
	`ObjectInputStream`
	`ObjectOutputStream`
		Used to serialize objects.

Data Conversion	*N/A*
	`DataInputStream`
	`DataOutputStream`
		Read or write primitive data types in a machine-independent format.

Counting	`LineNumberReader` `LineNumberInputStream`	Keeps track of line numbers while reading.
Peeking Ahead	`PushbackReader` `PushbackInputStream`	These input streams each have a pushback buffer. When reading data from a stream, it is sometimes useful to peek at the next few bytes or characters in the stream to decide what to do next.
Printing	`PrintWriter` `PrintStream`	Contain convenient printing methods. These are the easiest streams to write to, so you will often see other writable streams wrapped in one of these.

Buffering	`BufferedReader`
	`BufferedWriter`
	`BufferedInputStream`
	`BufferedOutputStream`
		Buffer data while reading or writing, thereby reducing the number of accesses required on the original data source. Buffered streams are typically more efficient than similar nonbuffered streams and are often used with other streams.

Filtering	`FilterReader`
	`FilterWriter`
	`FilterInputStream`
	`FilterOutputStream`
		These abstract classes define the interface for filter streams, which filter data as it's being read or written.

Converting between Bytes and Characters	`InputStreamReader`
	`OutputStreamWriter`
	.
		A reader and writer pair that forms the bridge between byte streams and character streams.
		An `InputStreamReader` reads bytes from an `InputStream` and converts them to characters, using the default character encoding or a character encoding specified by name.
		An `OutputStreamWriter` converts characters to bytes, using the default character encoding or a character encoding specified by name and then writes those bytes to an `OutputStream`.
		You can get the name of the default character encoding by calling `System.getProperty("file.encoding")`

Networking with Sockets

Let’s review socket networking.

Write a Java server that will display the current date when it receives the message (date today) from the client.

Write a Java client that will send the message (date today) to the server and then display the result that the server sends back.

Multithreaded Socket Servers

Modify the client and server programs that you just created so that the server can accept multiple client connections at one time.