CT326 Programming III

LECTURE 11-13 (WEEK 5)

I/O STREAMS

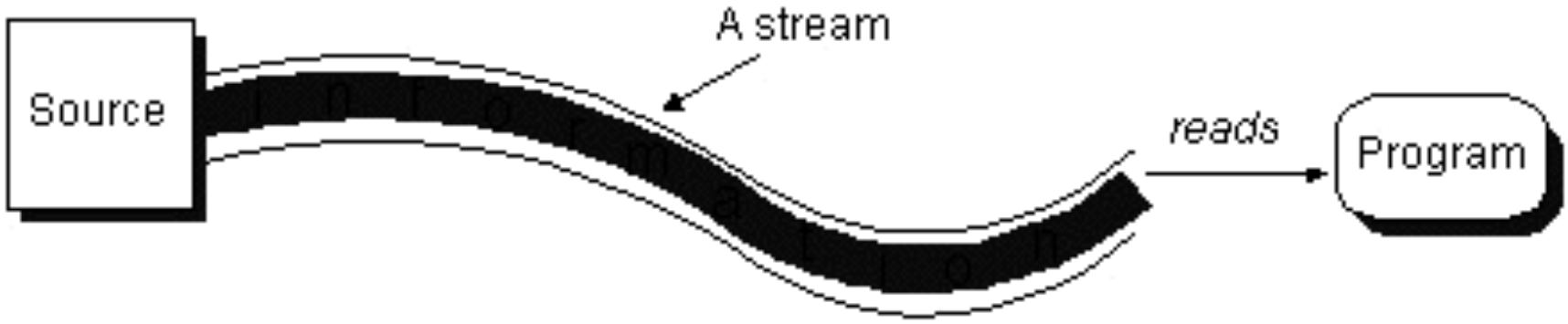
DR ADRIAN CLEAR
SCHOOL OF COMPUTER SCIENCE

Objectives for this week

- Understand the various IO Stream classes that java provides, their organisation and functionality
- Demonstrate the use of some of these
- Illustrate the use of Scanner and Formatting for high-level structuring of character streams

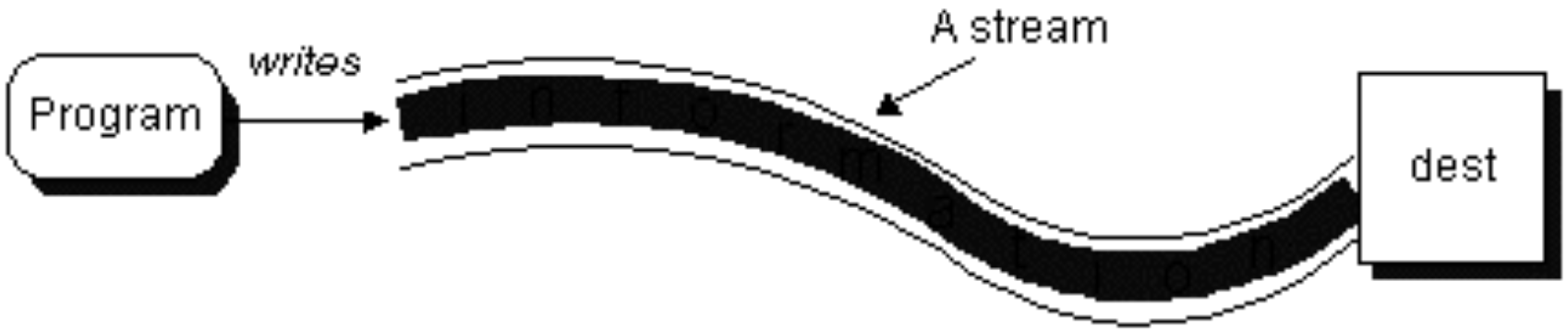
IO Streams

- To bring in information, a program opens a stream on an information source (a file, memory, a socket) and reads the information serially, like this:



IO Streams

- Similarly, a program can send information to an external destination by opening a stream to a destination and writing the information out serially, like this:



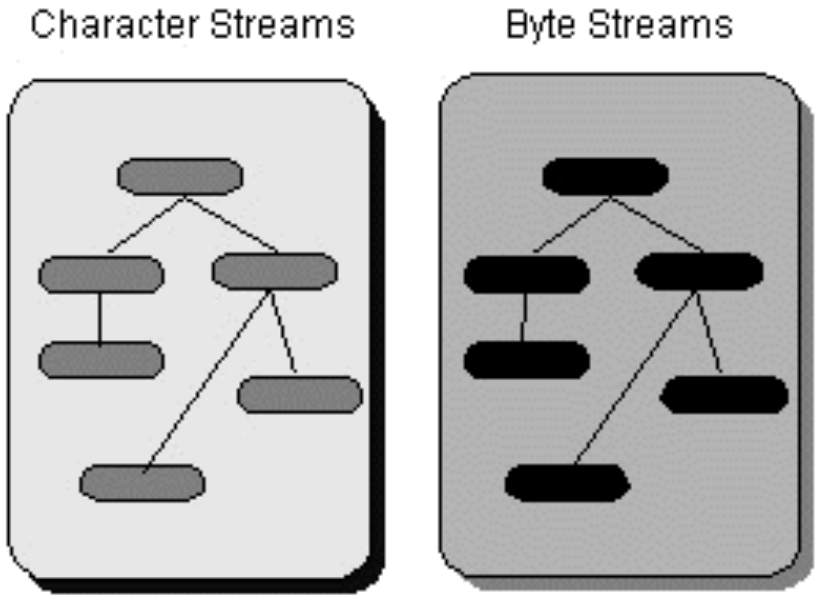
IO Streams

- No matter where the information is coming from or going to and no matter what type of data is being read or written, the algorithms for reading and writing data are usually the same.

```
Reading                          Writing
  open a stream                    open a stream
  while more information           while more information
                read information                 write information
  close the stream                 close the stream
```

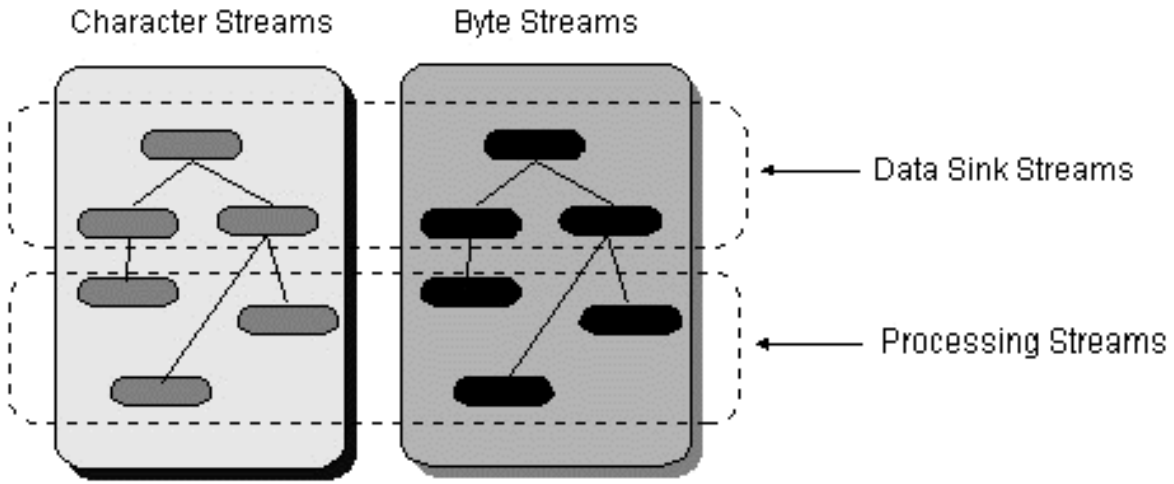
IO Streams

- The java.io package contains a collection of stream classes that support reading and writing streams.
- Divided into two class hierarchies based on the data (either characters or bytes) on which they operate.



IO Streams

- However, it's often more convenient to group the classes based on their purpose rather than on the data type they read and write.
- Thus, we can cross-group the streams by whether they read from and write to data "sinks" or process the information as its being read or written.



Data Sink Streams

- Data sink streams read from or write to specialised data sinks such as strings, files, or pipes.
- Typically, for each reader or input stream intended to read from a specific kind of input source, java.io contains a parallel writer or output stream that can create it.
- Note that both the character stream group and the byte stream group contain parallel pairs of classes that operate on the same data sinks.

Processing Streams

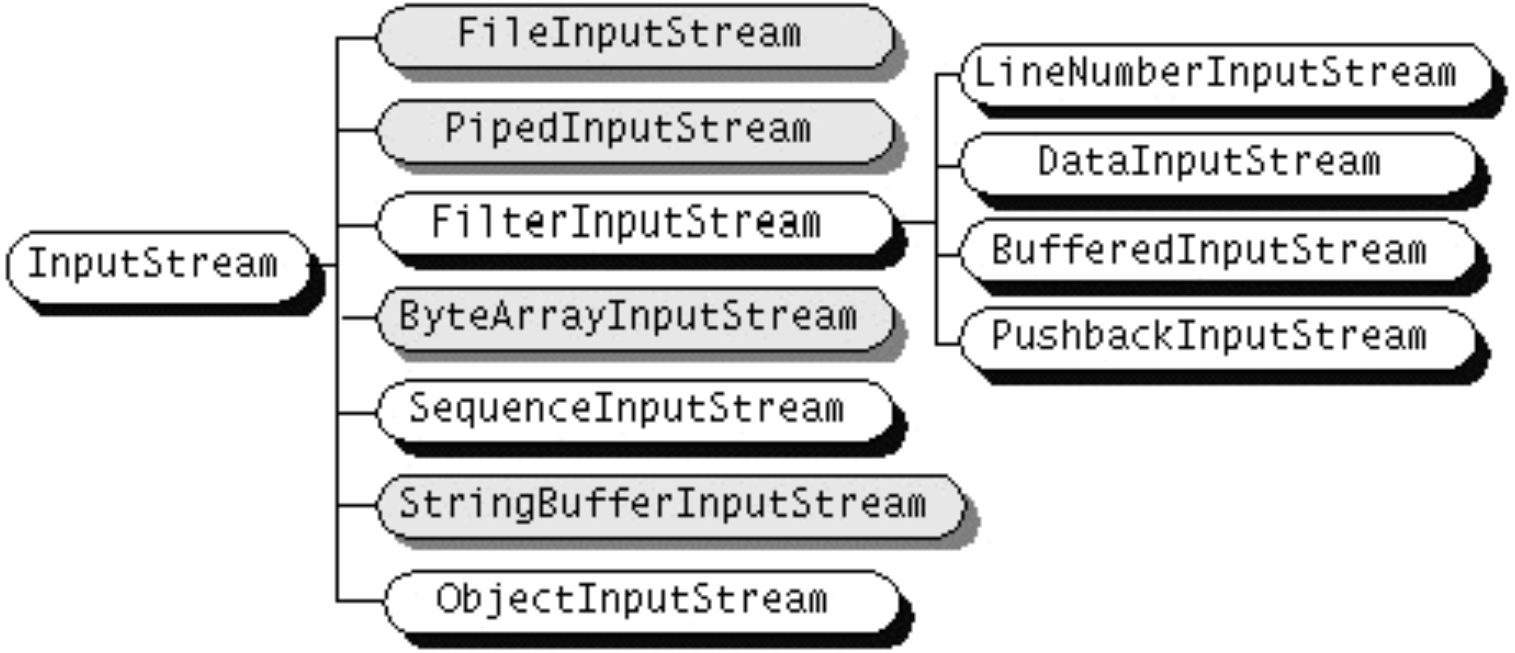
- Processing streams perform some sort of operation, such as buffering or data conversion, as they read and write.
- Like the data sink streams, java.io often contains pairs of streams:
  - One that performs a particular operation during reading and another that performs the same operation (or reverses it) during writing.
  - Note also that in many cases, java.io contains character streams and byte streams that perform the same processing but for the different data type.

Byte Streams

- Programs should use the byte streams, descendants of InputStream and OutputStream, to read and write 8-bit bytes.
- InputStream and OutputStream provide the API and some implementation for input streams (streams that read 8-bit bytes) and output streams (streams that write 8-bit bytes).
- These streams are typically used to read and write binary data such as images and sounds.

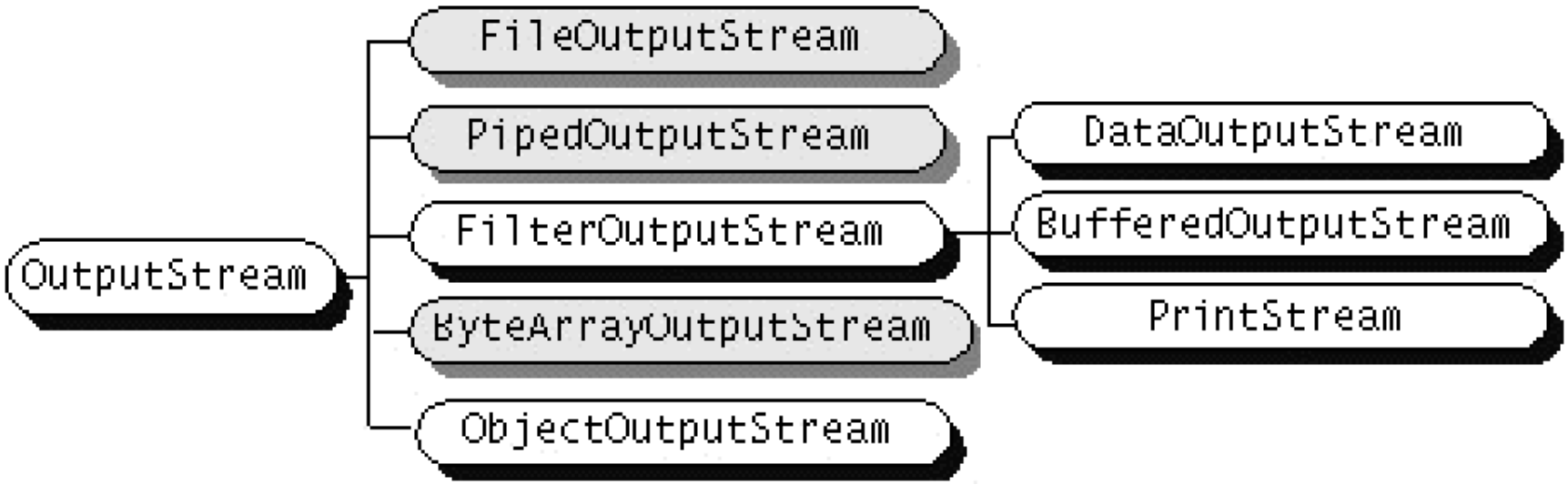
Byte Streams

- Subclasses of InputStream and OutputStream provide specialised I/O. Those that read from or write to data sinks are shown in grey in the following figures, those that perform some sort of processing are shown in white:



Byte Streams

- Note that these also fall into two categories:
  - Data sink streams and processing streams (again shown in grey and white):



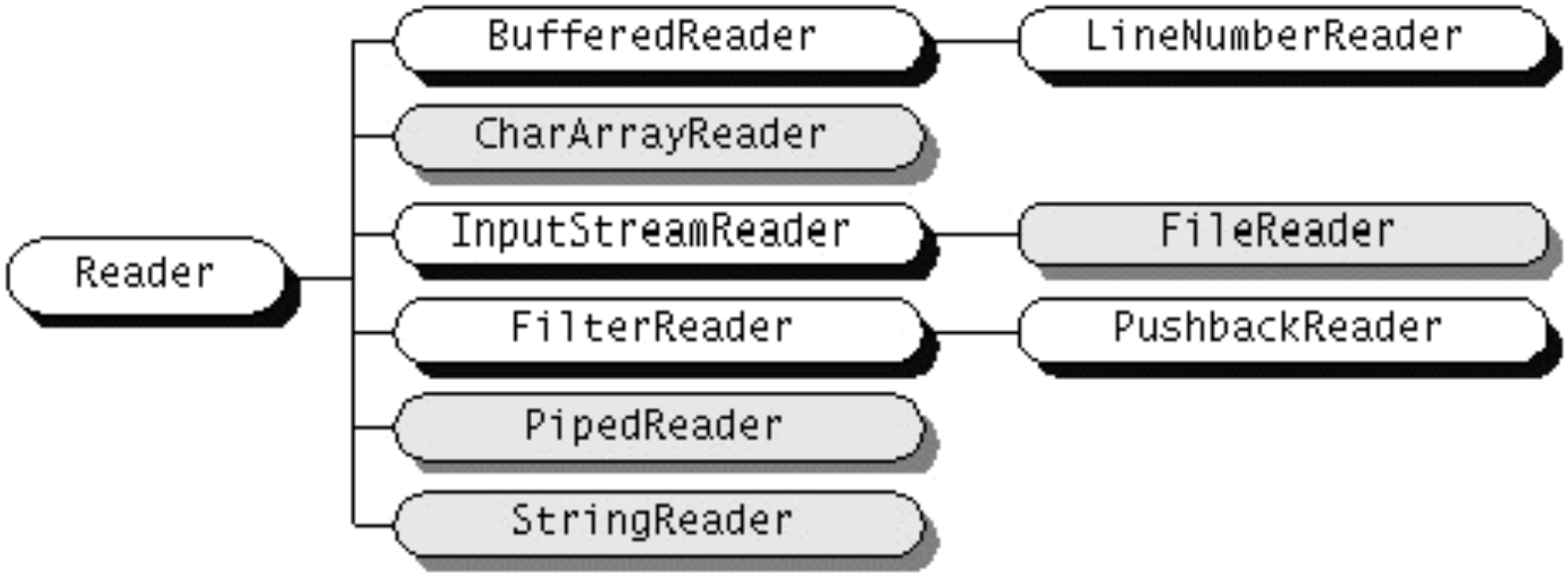
Byte Stream demo

Character Streams

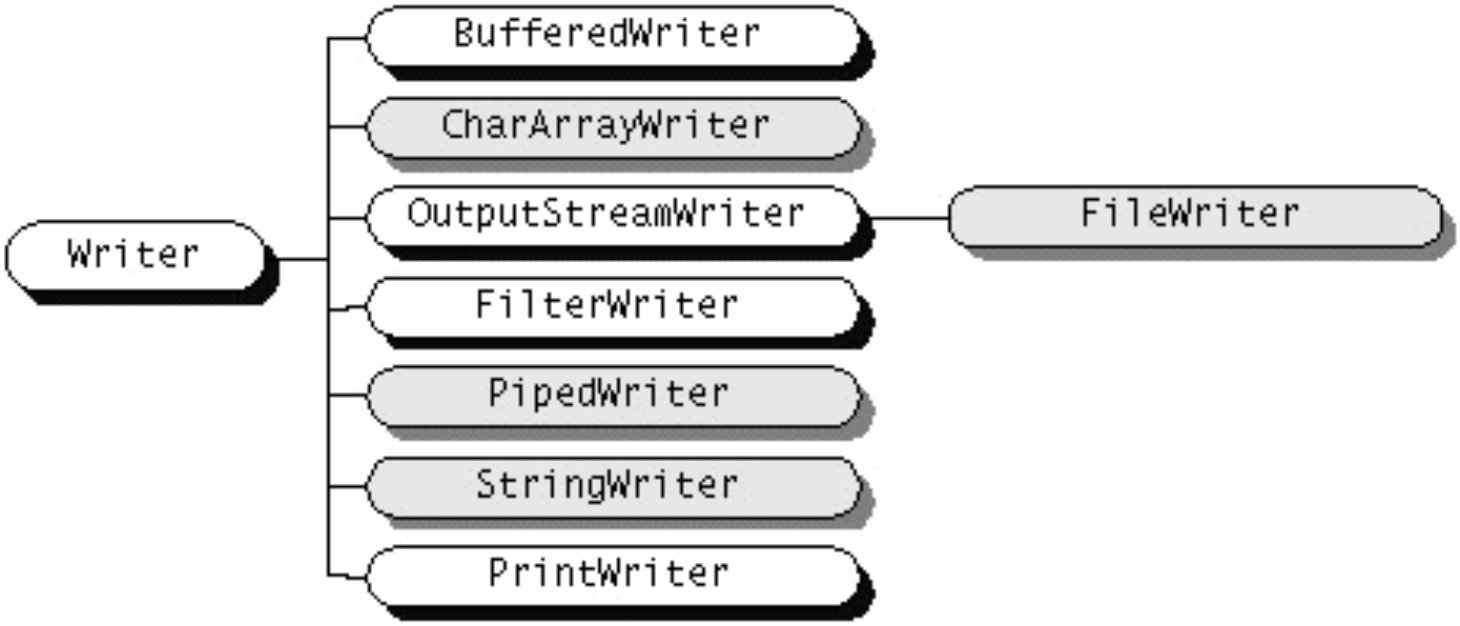
- Reader and Writer are the abstract super-classes for character streams in java.io.*
- Reader provides the API and partial implementation for readers - streams that read 16-bit unicode characters.
- Writer provides the API and partial implementation for writers - streams that write 16-bit characters.
- Subclasses of Reader and Writer implement specialised streams.

Class hierarchies for the Reader and Writer classes

- Those that read from or write to data sinks are shown in grey in the following figures, those that perform some sort of processing are shown in white.



Character Streams



- Most programs should use readers and writers to read and write information.
  - This is because they both can handle any character in the Unicode character set (while the byte streams are limited to ISO-Latin-1 8-bit bytes).

Character Stream demo

Data Sink Streams

Sink Type	Character Streams	Byte Streams
Memory	CharArrayReader	ByteArrayInputStream
	CharArrayWriter	ByteArrayOutputStream
	StringReader	StringBufferInputStream
	StringWriter	
Pipe	PipedReader	PipedInputStream
	PipedWriter	PipedOutputStream
File	FileReader	FileInputStream
	FileWriter	FileOutputStream

Processing Streams

Process	Character Streams	Byte Streams
Buffering	BufferedReader BufferedWriter	BufferedInputStream BufferedOutputStream
Filtering	FilterReader FilterWriter	FilterInputStream FilterOutputStream
Stream Conversion	InputStreamReader OutputStreamWriter	
Concatenation		SequenceInputStream

Processing Streams

Process	Character Streams	Byte Streams
Data Conversion		DataInputStream, DataOutputStream
Counting	LineNumberReader	LineNumberInputStream
Peeking Ahead	PushbackReader	PushbackInputStream
Printing	PrintWriter	PrintStream
Object Serialization	ObjectInputStream	ObjectOutputStream

Scanner

- in the java.util package but can be passed an InputStream as a constructor parameter
  - often used for reading from the console (i.e. the System.in InputStream)
- used for working with input of formatted data consisting of primitive types and strings
- translates input to tokens based on their data type and using a delimiter pattern
- has a collection of “next” methods for different data types

Formatting

- PrintWriter is a processing Character stream
- Includes common methods for formatting
  - print
  - println
  - format
- `format` method formats multiple arguments based on a *format string* that includes format specifiers
  - can be used similar to tokens in Scanner
- See java.util.Formatter class for documentation on format string syntax

Scanner and formatting demo

Next time…

- Object Serialisation