Module 14

Advanced I/O Streams

Objectives

- Describe the main features of the `java.io` package
- Construct node and processing streams, and use them appropriately
- Distinguish readers and writers from streams, and select appropriately between them

Relevance

- What mechanisms are in place within the Java programming language to read and write from sources (or sinks) other than files?
- How are international character sets supported in I/O operations?
- What are the possible sources and sinks of character and byte streams?

I/O Fundamentals

- A *stream* can be thought of as a flow of data from a source or to a sink.
- A *source* stream initiates the flow of data, also called an input stream.
- A *sink* stream terminates the flow of data, also called an output stream.
- Sources and sinks are both *node streams*.
- Types of node streams are files, memory, and pipes between threads or processes.

Fundamental Stream Classes

Stream	Byte Streams	Character Streams
Source streams	`InputStream`	`Reader`
Sink streams	`OutputStream`	`Writer`

Data Within Streams

- Java technology supports two types of streams: character and byte.
- Input and output of character data is handled by readers and writers.
- Input and output of byte data is handled by input streams and output streams:
  - Normally, the term *stream* refers to a byte stream.
  - The terms *reader* and *writer* refer to character streams.

The `InputStream` Methods

- The three basic `read` methods are:

```
int read()
int read(byte[] buffer)
int read(byte[] buffer, int offset, int length)
```

- Other methods include:

```
void close()
int available()
long skip(long n)
boolean markSupported()
void mark(int readlimit)
void reset()
```

The `OutputStream` Methods

- The three basic `write` methods are:

```
void write(int c)
void write(byte[] buffer)
void write(byte[] buffer, int offset, int length)
```

- Other methods include:

```
void close()
void flush()
```

The `Reader` Methods

- The three basic `read` methods are:

```
int read()
int read(char[] cbuf)
int read(char[] cbuf, int offset, int length)
```

- Other methods include:

```
void close()
boolean ready()
long skip(long n)
boolean markSupported()
void mark(int readAheadLimit)
void reset()
```

The `Writer` Methods

- The basic `write` methods are:

```
void write(int c)
void write(char[] cbuf)
void write(char[] cbuf, int offset, int length)
void write(String string)
void write(String string, int offset, int length)
```

- Other methods include:

```
void close()
void flush()
```

Node Streams

Type	Character Streams	Byte Streams
File	`FileReader` `FileWriter`	`FileInputStream` `FileOutputStream`
Memory: array	`CharArrayReader` `CharArrayWriter`	`ByteArrayInputStream` `ByteArrayOutputStream`
Memory: string	`StringReader` `StringWriter`	N/A
Pipe	`PipedReader` `PipedWriter`	`PipedInputStream` `PipedOutputStream`

A Simple Example

This program performs a copy file operation using a manual buffer:

```
java TestNodeStreams file1 file2
```

```
1   import java.io.*;
2
3   public class TestNodeStreams {
4     public static void main(String[] args) {
5       try {
6         FileReader input = new FileReader(args[0]);
7         FileWriter output = new FileWriter(args[1]);
8         char[]     buffer = new char[128];
9         int        charsRead;
10
11        // read the first buffer
12        charsRead = input.read(buffer);
```

A Simple Example

```
13
14          while ( charsRead != -1 ) {
15            // write the buffer out to the output file
16            output.write(buffer, 0, charsRead);
17
18            // read the next buffer
19            charsRead = input.read(buffer);
20          }
21
22          input.close();
23          output.close();
24        } catch (IOException e) {
25          e.printStackTrace();
26        }
27      }
28  }
```

Buffered Streams

This program performs a copy file operation using a built-in buffer:

```
java TestBufferedStreams file1 file2
```

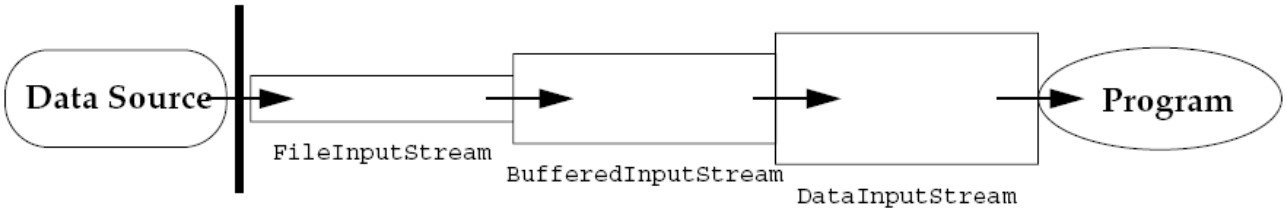
```
1   import java.io.*;
2
3   public class TestBufferedStreams {
4     public static void main(String[] args) {
5       try {
6         FileReader     input     = new FileReader(args[0]);
7         BufferedReader bufInput  = new BufferedReader(input);
8         FileWriter     output    = new FileWriter(args[1]);
9         BufferedWriter bufOutput = new BufferedWriter(output);
10        String line;
11
12        // read the first line
13        line = bufInput.readLine();
```

Buffered Streams

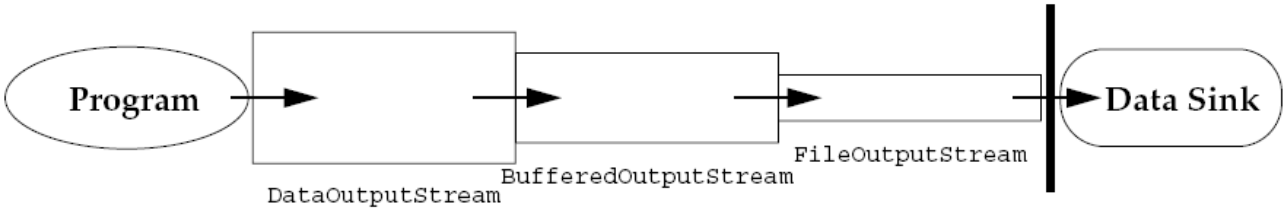
```
14
15          while ( line != null ) {
16            // write the line out to the output file
17            bufOutput.write(line, 0, line.length());
18            bufOutput.newLine();
19            // read the next line
20            line = bufInput.readLine();
21          }
22          bufInput.close();
23          bufOutput.close();
24        } catch (IOException e) {
25          e.printStackTrace();
26        }
27      }
28  }
```

I/O Stream Chaining

Input Stream Chain



Output Stream Chain



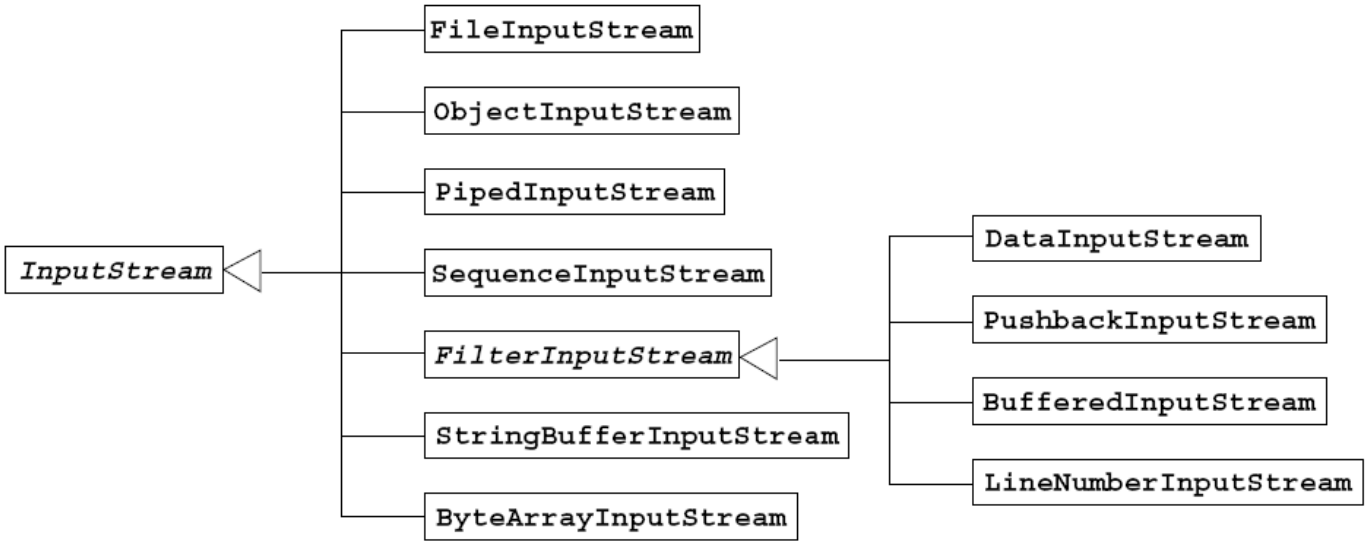
Processing Streams

Type	Character Streams	Byte Streams
Buffering	`BufferedReader` `BufferedWriter`	`BufferedInputStream` `BufferedOutputStream`
Filtering	*`FilterReader`* *`FilterWriter`*	*`FilterInputStream`* *`FilterOutputStream`*
Converting between bytes and character	`InputStreamReader` `OutputStreamWriter`	
Performing object serialization		`ObjectInputStream` `ObjectOutputStream`

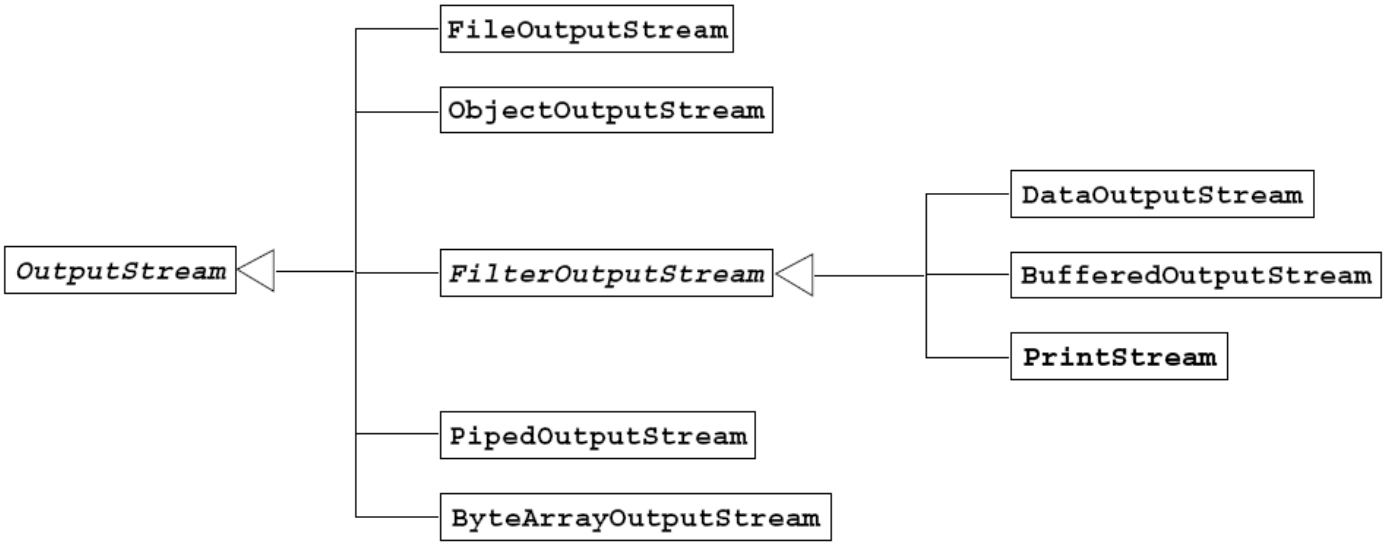
Processing Streams

Type	Character Streams	Byte Streams
Performing data conversion		`DataInputStream` `DataOutputStream`
Counting	`LineNumberReader`	`LineNumberInputStream`
Peeking ahead	`PushbackReader`	`PushbackInputStream`
Printing	`PrintWriter`	`PrintStream`

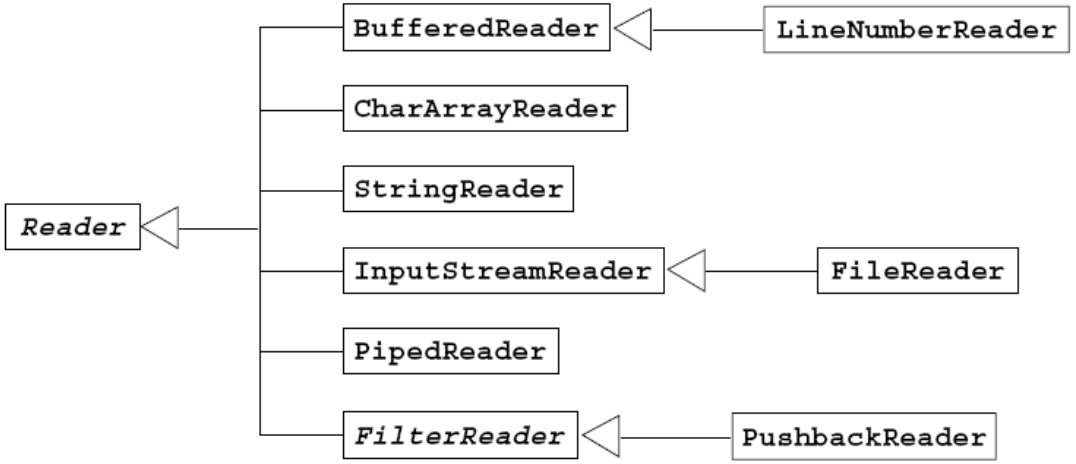
The `InputStream` Class Hierarchy



The `OutputStream` Class Hierarchy



The `Reader` Class Hierarchy



The `Writer` Class Hierarchy