JAC444 / BTP400 Course Object-Oriented Software Development II - Java

Files I/O

Segment 2

Objectives

Upon completion of this lecture, you should be able to:

- Understand Reader / Writer in Java
- Compare CharacterStream and ByteStream
- Work with Buffered Stream
- Design and Develop File I/O programs

Reader/InputStream

- `Reader` and `InputStream` define similar methods, but for different data types.
  - **`Reader` – Reading characters and array of characters.**
    - *`int read()`*
    - *`int read(char[] cbuf)`*
    - *`int read(char[] cbuf, int offset, int length)`*
  - **`InputStream` – Reading bytes and array of bytes.**
    - *`int read()`*
    - *`int read(byte[] cbuf)`*
    - *`int read(byte[] cbuf, int offset, int length)`*

Data Sink Streams

- Data streams read from or write to specializezed sinks:

Sink type	Character Streams	Byte Streams
Memory	*CharArrayReader* *CharArrayWriter*	*ByteArrayInputStream* *ByteArrayOutputStream*
	StringReader *StringWriter*	*StringBufferInputStream*
Pipe	*PipeReader* *PipeWriter*	*PipedInputStream* *PipedOutputStream*
File	*FileReader* *FileWriter*	*FileInputStream* *FileOutputStream*

File Streams Example

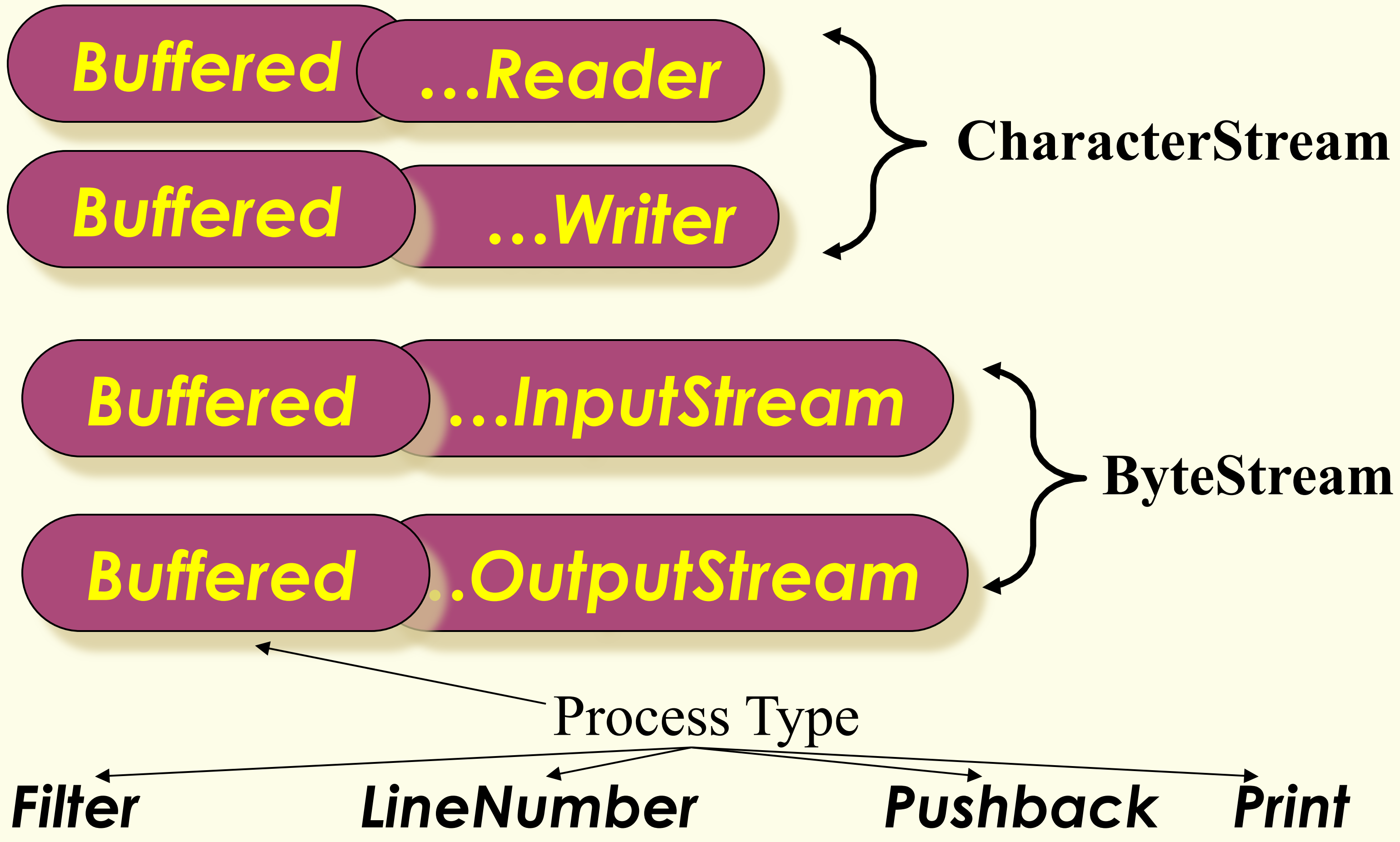
```
import java.io.*;
public class Copy {
   public static void main(String[] args) throws IOException {
       File inputFile = new File("args[0]");    //source
       File outputFile = new File("args[1]");  //destination

       FileReader in = new FileReader(inputFile);
       FileWriter out = new FileWriter(outputFile);
       int c;

       while ((c = in.read()) != -1)
           out.write(c);
       in.close();
       out.close();
    }
}
```

Processing Stream

- Processing streams perform some sort of operation, such as buffering or character encoding, as they read and write.

Process	CharacterStreams	Byte Streams
Buffering	*BufferedReader,* *BufferedWriter*	*BufferedInputStream* *BufferedOutputStream*
Filtering	*FilterReader,* *FilterWriter*	*FilterInputReader* *FilterOutputWriter*
Converting between Bytes and Characters	*InputStreamReader* *OutputStreamWriter*	
Concatenation		*SequenceInputStream*
Object Serialization		*ObjectInputStream* *ObjectOutputStream*
Data Conversion		*DataInputStream* *DataOutputStream*
Counting	*LineNumberReader*	*LineNumberInputStream*
Peeking Ahead	*PushbackReader*	*PushbackInputStream*
Printing	*PrintWriter*	*PrintStream*

Patterns of IO Class Names



Concatenate utility – List of Files

```
public class ListOfFiles implements Enumeration {
   private String[] listOfFiles;
   private int current = 0;
   public ListOfFiles(String[] listOfFiles) {
       this.listOfFiles = listOfFiles;
   }
   public boolean hasMoreElements() {
       if (current < listOfFiles.length) return true; else return false;
   }
   public Object nextElement() {
       InputStream in = null;
       if (!hasMoreElements())
          throw new NoSuchElementException("No more files.");
       else {
          String nextElement = listOfFiles[current];
          current++;
          try {
              in = new FileInputStream(nextElement);
          } catch (FileNotFoundException e) {
              System.err.println("ListOfFiles: Can't open " + nextElement);
          }
       }
       return in;
   }
}
```

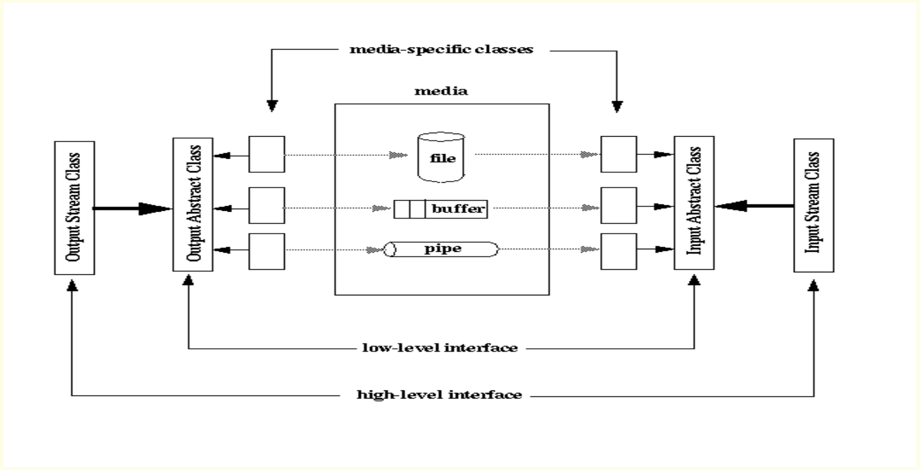
Concatenate utility

```java
import java.io.*;
public class Concatenate {
   public static void main(String[] args) throws IOException {
      ListOfFiles list = new ListOfFiles(args);

      SequenceInputStream s = new SequenceInputStream(list);
      int c;

      while ((c = s.read()) != -1)
         System.out.write(c);

      s.close();
   }
}
```

Conclusions

After completion of this segment you should know:

- How to use Files in Java.
- How to read data to and write data from Java Files
- Use **`java.io`** package for IO data processing .


media-specific classes
media
file
buffer
pipe
Output Stream Class
Output Abstract Class
Input Abstract Class
Input Stream Class
low-level interface
high-level interface