12.3: Byte Stream I/O Hierarchy

Example: FileInputStream & FileOutputStream

```
class CopyFile  {
        FileInputStream fromFile;  FileOutputStream toFile;
        public void init(String arg1, String arg2) {       //pass file names
                try{
                   fromFile = new FileInputStream(arg1);
                   toFile = new FileOutputStream(arg2);
                } catch (Exception fnfe) {…}
      }
      public void copyContents() {// copy bytes
        try {
                int i = fromFile.read();
                while ( i != -1) {                          //check the end of file
                        toFile.write(i);
                        i = fromFile.read(); }
        } catch (IOException ioe) { System.out.println("Exception: " + ioe);}
  }
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 12

The remainder of the code follows:

```
public void closeFiles() {                    //close the file
    try{
            fromFile.close();
            toFile.close();
     } catch (IOException ioe){
            System.out.println("Exception: " + ioe);
            }
}
public static void main(String[] args){
            CopyFile c1 = new CopyFile();
            c1.init(args[0], args[1]);
            c1.copyContents();
            c1.closeFiles();
} }
```

The FileInputStream and FileOutputStream classes define byte input and output streams that are connected to files. Data can only be read or written as a sequence of bytes. The above example demonstrates the use of File InputStream and FileOutputStream.

12.3: Byte Stream I/O Hierarchy

Demo : FileInput/OutputStream

- Execute:
  - ReadKeys.java
  - CopyFile.java program

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 13

12.4: Character Stream Hierarchy

Character Stream Hierarchy

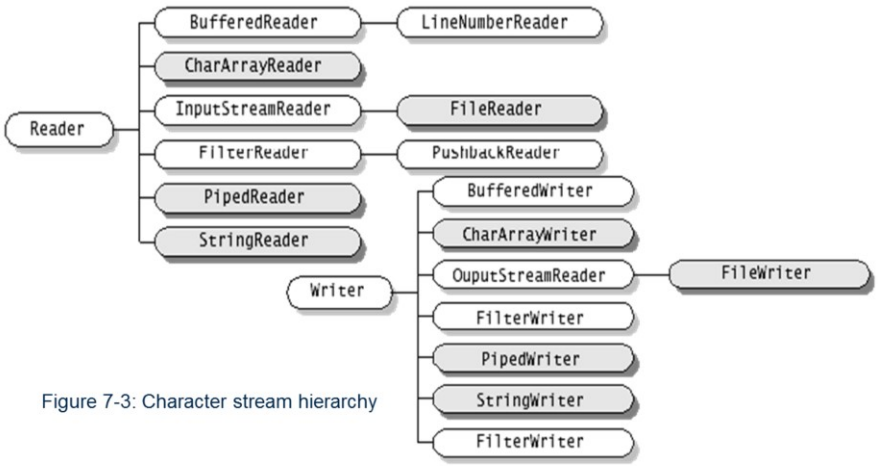

Figure 7-3: Character stream hierarchy

 14

The byte stream classes support only 8-bit byte streams and doesn't handle 16-bit Unicode characters well. A character encoding is a scheme for representing characters. Java represents characters internally in the 16-bit Unicode character encoding, but the host platform might use different character encoding.
The abstract classes Reader and Writer are the roots of the inheritance hierarchies for streams that read and write Unicode characters using a specific character encoding.
A reader is an input character stream that reads a sequence of Unicode characters, and a writer is an output character stream that writes a sequence of Unicode characters.

12.4: Character Stream Hierarchy

Reader Class Methods

Method	Description
int read() throws IOException	reads a byte and returns as an int
int read(char b[])throws IOException	reads into an array of chars b
int read(char b[], int off, int len) throws IOException	reads *len* number of characters into char array *b*, starting from offset *off*
long skip(long n) throws IOException	Can skip n characters.

Table 7-4: Reader Methods

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 15

Note: Refer to Java documentation for more methods.

12.4: Character Stream Hierarchy

Writer Class Methods

Method	Description
void write(int c)throws IOException	writes a byte.
void write(char b[])throws IOException	writes from an array of chars b
void write(char b[], int off, int len) throws IOException	writes len number of characters from char array b, starting from offset off
void write(String b, int off, int len) throws IOException	writes len number of characters from string b, starting from offset off

Table 7-5: Writer Methods

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 16

Note: Refer to Java documentation for more methods.

12.4: Character Stream Hierarchy

Example: FileReader, FileWriter Classes

```
public class CopyCharacters {
   public static void main(String[] args) throws IOException {
 try(FileReader inputStream = new FileReader("sampleinput.txt");
      FileWriter outputStream = newFileWriter("sampleoutput.txt"))        {
          int c;
          while ((c = inputStream.read()) != -1) {
              outputStream.write(c);
          }
 } catch(IOException ex) {
              System.out.println(ex.getMessage());
  }
} }
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 17

12.5: Buffered Stream

Buffered Input Output Stream

- An unbuffered I/O means each read or write request is handled directly by the underlying OS.
  - Makes a program less efficient.
    - Each such request often triggers disk access, network activity, or some other relatively expensive operation.
- Java's buffered I/O Streams reduce this overhead.
  - Buffered streams read/write data from a memory area known as a buffer; the native input API is called only when the buffer is empty.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 18

12.5: Buffered Stream

Using buffered streams

- A program can convert a unbuffered stream into buffered using the *wrapping idiom:*
  - Unbuffered stream object is passed to the constructor of a buffered stream class.
  - Example

```
inputStream = new BufferedReader(new FileReader("input.txt"));
outputStream = new BufferedWriter(new FileWriter("output.txt"));
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 19

There are four buffered stream classes used to wrap unbuffered streams.

BufferedInputStream and BufferedOutputStream - create buffered byte streams.

BufferedReader and BufferedWriter - create buffered character streams.

Flushing Buffered Streams

It often makes sense to write out a buffer at critical points, without waiting for it to fill. This is known as flushing the buffer.

Some buffered output classes support autoflush, specified by an optional constructor argument. When autoflush is enabled, certain key events cause the buffer to be flushed. For example, an autoflush PrintWriter object flushes the buffer on every invocation of println or format.

To flush a stream manually, invoke its flush() method. The flush() method is valid on any output stream, but has no effect unless the stream is buffered.

12.5: Buffered Stream

Example of Buffered stream

```
class LineNumberReaderDemo{
    public static void main(String args[]) {
        String s;
        try(FileReader fr = new FileReader("names.txt");
            BufferedReader br = new BufferedReader(fr);
            LineNumberReader lr = new LineNumberReader(br);)  {
            while((s = lr.readLine()) != null)
                System.out.println(lr.getLineNumber()+" " +s);
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
} }
```

Names.txt contains
Anita
Bindu
Cindy
Diana

Output is:
1 Anita
2 Bindu
3 Cindy
4 Diana

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 20

12.5: Buffered Stream

Demo: File Reader / File Writer

- Execute the Lesson 12
  - LineNumberReaderDemo.java
  - CharEncode.java

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 21

12.6: File class

The File Class

- File class doesn't operate on streams
- Represents the pathname of a file or directory in the host file system
- Used to obtain or manipulate the information associated with a disk file, such as permissions, time, date, directory path etc
- An object of File class provides a handle to a file or directory and can be used to create, rename or delete the entry

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 22

Support for File/Directory Operations are provided by java.io.File. This class makes it easier to write platform-independent code that examines and manipulates files.
Provides methods
To obtain basic information about the file/directory
To Create / Delete Files and Directories