Streams & Files

CS 240 – Advanced Programming Concepts

Ways to Read Files

- Streams
- Scanner Class
- Files Class
- RandomAccessFile Class

Java I/O Streams (overview)

- Writing data to / Reading data from files (or other data sources)
- Two Choices: Binary-Formatted or Text-Formatted Data
  - Binary Formatted: 00 00 04 D2 (4 bytes)
  - Text Formatted: 1234 (4 characters)
- InputStream and OutputStream
  - Reading/writing bytes
  - Reading/writing binary-formatted data
- Reader and Writer
  - Reading/writing characters
  - Reading/writing text-formatted data

Other Java I/O Classes (overview)

- File
  - Represents a file in the file system but not directly used to read it
- Scanner
  - Tokenize stream input (read one token at a time)
- Files
  - Read, copy, etc whole files
- RandomAccessFile
  - Use a file pointer to read from / write to any location in a file

Reading/Writing Bytes

- The InputStream interface is used to read bytes sequentially from a data source
  - FileInputStream
  - PipedInputStream
  - URLConnection.getInputStream()
  - HttpExchange.getRequestBody()
  - ResultSet.getBinaryStream(int columnIndex)
  - Many more examples in the Java API

Filter Input Streams

- There are many features you may want to enable when consuming data from an InputStream
  - Decompress data as it comes out of the stream
  - Decrypt data as it comes out of the stream
  - Compute a “digest” of the stream (a fixed-length value that summarizes the data in the stream)
  - Byte counting
  - Line Counting
  - Buffering

Filter Input Streams

- Open an InputStream on a data source (file, socket, etc.), and then wrap it in one or more “filter input streams” that provide the features you want (decompression, decryption, etc.)

- Filter input streams all implement the InputStream interface, and can be arranged as a pipeline with the real data source at the end

- Each filter input stream reads data from the next InputStream in line, and then performs a transformation or calculation on the data

OutputStream

- Writing bytes works the same way, just in reverse
- The OutputStream interface is used to write bytes sequentially to a data destination
  - FileOutputStream
  - PipedOutputStream
  - URLConnection.getOutputStream()
  - HttpExchange.getResponseBody()
  - Many more examples in the Java API

Filter Output Streams

- There are many features you may want to enable when writing data to an OutputStream
  - Compress data as it goes into the stream
  - Encrypt data as it goes into the stream
  - Compute a “digest” of the stream (a fixed-length value that summarizes the data in the stream)
  - Buffering

Filter Output Streams

- Open an OutputStream on a data destination (file, socket, etc.), and then wrap it in one or more “filter output streams” that provide the features you want (compression, encryption, etc.)

- Filter output streams all implement the OutputStream interface, and can be arranged as a pipeline with the real data destination at the end

- Each filter output stream performs a transformation or calculation on the data, and then writes it to the next OutputStream in line

Filter Stream Example

- Compress a file to GZIP format
  - Compress Example
  - LegacyCompress Example
- Decompress a file from GZIP format
  - Decompress Example
  - LegacyDecompress Example

Reading/Writing Binary-Formatted Data

- Reading/writing bytes is useful, but usually we want to read/write larger data values, such as: float, int, boolean, etc.
- The DataOutputStream class lets you write binary-formatted data values
- The DataOutputStream(OutputStream out) constructor wraps a DataOutputStream around any OutputStream
- The DataInputStream class lets you read binary-formatted data values
- The DataInputStream(InputStream in) constructor wraps a DataInputStream around any InputStream

Reading/Writing Characters

- The Reader interface is used to read characters sequentially from a data source
- The Writer interface is used to write characters sequentially to a data destination
- See CopyFileExample (from Java Fundamentals lecture)
- Convert between streams and readers or writers using InputStreamReader and OutputStreamWriter

```
new InputStreamReader(new FileInputStream("myfile.txt"));

new OutputStreamWriter(new FileOutputStream("myfile.txt"));
```

Reading/Writing Text-Formatted Data

- The PrintWriter class lets you write text-formatted data values (String, int, float, boolean, etc.)
  - See CopyFileExample (from Java Fundamentals lecture)
- The Scanner class lets you read text-formatted data values

Scanner: Tokenize Data Read from a File

```
public void processFile(File file) throws IOException {
    Scanner scanner = new  Scanner(file);
    // Delimit by whitespace and # line comments
    scanner.useDelimiter("((#[^\\n]*\\n)|(\\s+))+");

    while(scanner.hasNext()) {
        String str = scanner.next();
        // Do something with the String
    }
}
```

Files: Read Entire File into a List

```
public List<String> readFile(File file) throws IOException {
    Path path = Paths.get(file.getPath());
    List<String> fileContents = Files.readAllLines(path);
    return fileContents;
}
```

Random Access Files

- The RandomAccessFile class allows "random access" to a file's contents for both reading and writing

- Random Access
  - The "file pointer" represents the current location in the file (similar to an array index)
  - Use the `seek(long)` or `skipBytes(int)` methods to move the "file pointer" to any location in the file
  - Read or write bytes at the current file pointer location using various overloaded read and write methods