12 Advanced I/O Streams

Topics

- General Stream Types
  - Character and Byte Streams
  - Input and Output Streams
  - Node and Filter Streams
- The *File* Class
- *Reader* Classes
  - *Reader* Methods
  - Node *Reader* Classes
  - Filter *Reader* Classes

Topics

- *Writer* Classes
  - *Writer* Methods
  - Node *Writer* Classes
  - Filter *Writer* Classes
- *InputStream* Classes
  - *InputStream* Methods
  - Node *InputStream* Classes
  - Filter *InputStream* Classes

Topics

- *OutputStream* Classes
  - *OutputStream* Methods
  - Node *OutputStream* Classes
  - Filter *OutputStream* Classes
- Serialization
  - The *transient* Keyword
  - Serialization: Writing an Object Stream
  - Deserialization: Reading an Object Stream

General Stream Types

- Streams
  - Abstraction of a file or a device that allows a series of items to be read or written
- General Stream Categories
  - Character and Byte Streams
  - Input and Output Streams
  - Node and Filter Streams

Character and Byte Streams

- Character streams
  - File or device abstractions for Unicode characters
  - Superclass of all classes for character streams:
    - The *Reader* class
    - The *Writer* class
    - Both classes are *abstract*
- Byte streams
  - For binary data
  - Root classes for byte streams:
    - The *InputStream* Class
    - The *OutputStream* Class
    - Both classes are *abstract*

Input and Output Streams

- Input or source streams
  - Can read from these streams
  - Superclasses of all input streams:
    - The *InputStream* Class
    - The *Reader* Class
- Output or sink streams
  - Can write to these streams
  - Root classes of all output streams:
    - The *OutputStream* Class
    - The *Writer* Class

Node and Filter Streams

- Node streams
  - Contain the basic functionality of reading or writing from a specific location
  - Types of node streams include files, memory and pipes
- Filter streams
  - Layered onto node streams between threads or processes
  - For additional functionalities
  - Adding layers to a node stream is called stream chaining

The *File* Class

- Not a stream class
- Important since stream classes manipulate *File* objects
- Abstract representation of actual files and directory pathnames

The *File* Class: Constructors

- Has four constructors

A File Constructor
`File(String pathname)`
Instantiates a *File* object with the specified *pathname* as its filename. The filename may either be absolute (i.e., containes the complete path) or may consists of the filename itself and is assumed to be contained in the current directory.

The *File* Class: Methods

File Methods
`public String getName()`
Returns the filename or the directory name of this *File* object.
`public boolean exists()`
Tests if a file or a directory exists.
`public long length()`
Returns the size of the file.
`public long lastModified()`
Returns the date in milliseconds when the file was last modified.
`public boolean canRead()`
Returns true if it's permissible to read from the file. Otherwise, it returns false.
`public boolean canWrite()`
Returns true if it's permissible to write to the file. Otherwise, it returns false.

The *File* Class: Methods

File Methods
`public boolean isFile()`
Tests if this object is a file, that is, our normal perception of what a file is (not a directory).
`public boolean isDirectory()`
Tests if this object is a directory.
`public String[] list()`
Returns the list of files and subdirectories within this object. This object should be a directory.
`public void mkdir()`
Creates a directory denoted by this abstract pathname.
`public void delete()`
Removes the actual file or directory represented by this *File* object.

The *File* Class: Example

```
1  import java.io.*;
2
3  public class FileInfoClass {
4     public static void main(String args[]) {
5        String fileName = args[0];
6        File fn = new File(fileName);
7        System.out.println("Name: " + fn.getName());
8        if (!fn.exists()) {
9           System.out.println(fileName
10                                + " does not exists.");
11 //continued...
```

The *File* Class: Example

```
12          /* Create a temporary directory instead. */
13          System.out.println("Creating temp
14                                  directory...");
15          fileName = "temp";
16          fn = new File(fileName);
17          fn.mkdir();
18          System.out.println(fileName +
19              (fn.exists()? "exists": "does not exist"));
20          System.out.println("Deleting temp
21                                  directory...");
22          fn.delete();
23  //continued...
```

The *File* Class: Example

```
24          System.out.println(fileName +
25             (fn.exists()? "exists": "does not exist"));
26          return;
27       }  //end of: if (!fn.exists())
28
29       System.out.println(fileName + " is a " +
30                   (fn.isFile()? "file." :"directory."));
31       if (fn.isDirectory()) {
32          String content[] = fn.list();
33          System.out.println("The content of this
34                                directory:");
35 //continued...
```

The *File* Class: Example

```
36            for (int i = 0; i < content.length; i++) {
37               System.out.println(content[i]);
38            }
39         } //end of: if (fn.isDirectory())
40
41         if (!fn.canRead()) {
42            System.out.println(fileName
43                                  + " is not readable.");
44            return;
45         }
46 //continued...
```

The *File* Class: Example

```
47          System.out.println(fileName + " is " + fn.length()
48                                  + " bytes long.");
49          System.out.println(fileName + " is " +
50                             fn.lastModified() + " bytes long.");
51
52          if (!fn.canWrite()) {
53              System.out.println(fileName
54                                      + " is not writable.");
55          }
56      }
57  }
```

The *Reader* Class: Methods

Reader Methods
`public int read(-) throws IOException`
An overloaded method, which has three versions. Reads character(s), an entire character array or a portion of a character array.
`public int read()` - Reads a single character.
`public int read(char[] cbuf)` - Reads characters and stores them in character array *cbuf*.
`public abstract int read(char[] cbuf, int offset, int length)` - Reads up to *length* number of characters and stores them in character array *cbuf* starting at the specified *offset*.
`public abstract void close() throws IOException`
Closes this stream. Calling the other *Reader* methods after closing the stream would cause an *IOException* to occur.

The *Reader* Class: Methods

Reader Methods
`public void mark(int readAheadLimit) throws IOException`
Marks the current position in the stream. After marking, calls to reset() will attempt to reposition the stream to this point. Not all character-input streams support this operation.
`public boolean markSupported()`
Indicates whether a stream supports the mark operation or not. Not supported by default. Should be overidden by subclasses.
`public void reset() throws IOException`
Repositions the stream to the last marked position.

Node *Reader* Classes

Node Reader Classes
`FileReader`
For reading from character files.
`CharArrayReader`
Implements a character buffer that can be read from.
`StringReader`
For reading from a string source.
`PipedReader`
Used in pairs (with a corresponding *PipedWriter*) by two threads that want to communicate. One of these threads reads characters from this source.

Filter *Reader* Classes

Filter Reader Classes
`BufferedReader`
Allows buffering of characters in order to provide for the efficient reading of characters, arrays, and lines.
`FilterReader`
For reading filtered character streams.
`InputStreamReader`
Converts read bytes to characters.
`LineNumberReader`
A subclass of the *BufferedReader* class that is able to keep track of line numbers.
`PushbackReader`
A subclass of the *FilterReader* class that allows characters to be pushed back or unread into the stream.

The *Writer* Class: Methods

Writer Methods
`public void write(-) throws IOException`
An overloaded method with five versions:
`public void write(int c)` – Writes a single character represented by the given integer value.
`public void write(char[] cbuf)` – Writes the contents of the character array *cbuf*.
`public abstract void write(char[] cbuf, int offset, int length)` – Writes *length* number of characters from the *cbuf* array, starting at the specified *offset*.
`public void write(String str)` – Writes the string *string*.
`public void write(String str, int offset, int length)` – Writes *length* number of characters from the string *str*, starting at the specified *offset*.
`public abstract void close() throws IOException`
Closes this stream after flushing any unwritten characters. Invocation of other methods after closing this stream would cause an *IOException* to occur.
`public abstract void flush()`
Flushes the stream (i.e., characters saved in the buffer are immediately written to the intended destination).

Node *Writer* Classes

Node Writer Classes
`FileWriter`
For writing characters to a file.
`CharArrayWriter`
Implements a character buffer that can be written to.
`StringWriter`
For writing to a string source.
`PipedWriter`
Used in pairs (with a corresponding *PipedReader*) by two threads that want to communicate. One of these threads writes characters to this stream.

Filter *Writer* Classes

Filter Writer Classes
`BufferedWriter`
Allows buffering of characters in order to provide for the efficient writing of characters, arrays, and lines.
`FilterWriter`
For writing filtered character streams.
`OutputStreamWriter`
Encodes characters written to it into bytes.
`PrintWriter`
Prints formatted representations of objects to a text-output stream.

Basic *Reader/Writer* Example

```
1  import java.io.*;
2
3  class CopyFile {
4     void copy(String input, String output) {
5        FileReader reader;
6        FileWriter writer;
7        int data;
8        try {
9           reader = new FileReader(input);
10          writer = new FileWriter(output);
11 //continued...
```

Basic *Reader/Writer* Example

```
12            while ((data = reader.read()) != -1) {
13               writer.write(data);
14            }
15            reader.close();
16            writer.close();
17         } catch (IOException ie) {
18            ie.printStackTrace();
19         }
20      }
21 //continued...
```

Basic *Reader/Writer* Example

```
22    public static void main(String args[]) {
23        String inputFile = args[0];
24        String outputFile = args[1];
25        CopyFile cf = new CopyFile();
26        cf.copy(inputFile, outputFile);
27    }
28 }
```

Modified *Reader/Writer* Example

```
1  import java.io.*;
2  class CopyFile {
3     void copy(String input, String output) {
4        BufferedReader reader;
5        BufferedWriter writer;
6        String data;
7        try {
8           reader = new
9                 BufferedReader(new FileReader(input));
10          writer = new
11                BufferedWriter(new FileWriter(output));
12 //continued...
```

Modified *Reader/Writer* Example

```
13            while ((data = reader.readLine()) != null) {
14               writer.write(data, 0, data.length);
15            }
16            reader.close();
17            writer.close();
18         } catch (IOException ie) {
19            ie.printStackTrace();
20         }
21      }
22 //continued...
```

Modified *Reader/Writer* Example

```
23    public static void main(String args[]) {
24        String inputFile = args[0];
25        String outputFile = args[1];
26        CopyFile cf = new CopyFile();
27        cf.copy(inputFile, outputFile);
28    }
29 }
```

The *InputStream* Class: Methods

InputStream Methods
`public int read(-) throws IOException`
An overloaded method, which also has three versions like that of the *Reader* class. Reads bytes.
`public abstract int read()` - Reads the next byte of data from this stream.
`public int read(byte[] bBuf)` - Reads some number of bytes and stores them in the *bBuf* byte array.
`public abstract int read(char[] cbuf, int offset, int length)` - Reads up to *length* number of bytes and stores them in the byte array *bBuf* starting at the specified *offset*.
`public abstract void close() throws IOException`
Closes this stream. Calling the other *InputStream* methods after closing the stream would cause an *IOException* to occur.

The *InputStream* Class: Methods

InputStream Methods
`public void mark(int readAheadLimit) throws IOException`
Marks the current position in the stream. After marking, calls to reset() will attempt to reposition the stream to this point. Not all byte-input streams support this operation.
`public boolean markSupported()`
Indicates whether a stream supports the mark and reset operation. Not supported by default. Should be overidden by subclasses.
`public void reset() throws IOException`
Repositions the stream to the last marked position.

Node *InputStream* Classes

Node InputStream Classes
`FileInputStream`
For reading bytes from a file.
`BufferedArrayInputStream`
Implements a buffer that contains bytes, which may be read from the stream.
`PipedInputStream`
Should be connected to a *PipedOutputStream*. These streams are typically used by two threads wherein one of these threads reads data from this source while the other thread writes to the corresponding *PipedOutputStream*.

Filter *InputStream* Classes

Filter InputStream Classes
`BufferedInputStream`
A subclass of *FilterInputStream* that allows buffering of input in order to provide for the efficient reading of bytes.
`FilterInputStream`
For reading filtered byte streams, which may transform the basic source of data along the way and provide additional functionalities.
`ObjectInputStream`
Used for object serialization. Deserializes objects and primitive data previously written using an *ObjectOutputStream*.
`DataInputStream`
A subclass of *FilterInputStream* that lets an application read Java primitive data from an underlying input stream in a machine-independent way.
`LineNumberInputStream`
A subclass of *FilterInputStream* that allows tracking of the current line number.
`PushbackInputStream`
A subclass of the *FilterInputStream* class that allows bytes to be pushed back or unread into the stream.

The *OutputStream* Class: Methods

OutputStream Methods
`public void write(-) throws IOException`
An overloaded method for writing bytes to the stream. It has three versions:
`public abstract void write(int b)` – Writes the specified byte value *b* to this output stream.
`public void write(byte[] bBuf)` – Writes the contents of the byte array *bBuf* to this stream.
`public void write(byte[] bBuf, int offset, int length)` – Writes *length* number of bytes from the *bBuf* array to this stream, starting at the specified *offset* to this stream.
`public abstract void close() throws IOException`
Closes this stream and releases any system resources associated with this stream. Invocation of other methods after calling this method would cause an *IOException* to occur.
`public abstract void flush()`
Flushes the stream (i.e., bytes saved in the buffer are immediately written to the intended destination).

Node *OutputStream* Classes

Node OutputStream Classes
`FileOutputStream`
For writing bytes to a file.
`BufferedArrayOutputStream`
Implements a buffer that contains bytes, which may be written to the stream.
`PipedOutputStream`
Should be connected to a *PipedInputStream*. These streams are typically used by two threads wherein one of these threads writes data to this stream while the other thread reads from the corresponding *PipedInputStream*.

Filter *OutputStream* Classes

Filter OutputStream Classes
`BufferedOutputStream`
A subclass of *FilterOutputStream* that allows buffering of output in order to provide for the efficient writing of bytes. Allows writing of bytes to the underlying output stream without necessarily causing a call to the underlying system for each byte written.
`FilterOutputStream`
For writing filtered byte streams, which may transform the basic source of data along the way and provide additional functionalities.
`ObjectOutputStream`
Used for object serialization. Serializes objects and primitive data to an *OutputStream*.
`DataOutputStream`
A subclass of *FilterOutputStream* that lets an application write Java primitive data to an underlying output stream in a machine-independent way.
`PrintStream`
A subclass of *FilterOutputStream* that provides capability for printing representations of various data values conveniently.

Basic *InputStream/ OutputStream* Example

```
1  import java.io.*;
2
3  class CopyFile {
4     void copy(String input, String output) {
5        FileInputStream inputStr;
6        FileOutputStream outputStr;
7        int data;
8        try {
9           inputStr = new FileInputStream(input);
10          outputStr = new FileOutputStream(output);
11 //continued...
```

Basic *InputStream*/ *OutputStream* Example

```
12            while ((data = inputStr.read()) != -1) {
13                outputStr.write(data);
14            }
15            inputStr.close();
16            outputStr.close();
17        } catch (IOException ie) {
18            ie.printStackTrace();
19        }
20    }
21 //continued...
```

Basic *InputStream/ OutputStream* Example

```
22    public static void main(String args[]) {
23        String inputFile = args[0];
24        String outputFile = args[1];
25        CopyFile cf = new CopyFile();
26        cf.copy(inputFile, outputFile);
27    }
28 }
```

Modified *InputStream/ OutputStream* Example

```
1  import java.io.*;
2
3  class CopyFile {
4     void copy(String input) {
5        PushbackInputStream inputStr;
6        PrintStream outputStr;
7        int data;
8        try {
9           inputStr = new PushbackInputStream(new
10                            FileInputStream(input));
11          outputStr = new PrintStream(System.out);
12 //continued...
```

Modified *InputStream/ OutputStream* Example

```
13          while ((data = inputStr.read()) != -1) {
14             outputStr.println("read data: " +
15                                (char) data);
16             inputStr.unread(data);
17             data = inputStr.read();
18             outputStr.println("unread data: " +
19                                (char) data);
20          }
21          inputStr.close();
22          outputStr.close();
23 //continued...
```

Modified *InputStream/ OutputStream* Example

```
24          } catch (IOException ie) {
25              ie.printStackTrace();
26          }
27      }
28
29      public static void main(String args[]) {
30          String inputFile = args[0];
31          CopyFile cf = new CopyFile();
32          cf.copy(inputFile);
33      }
34  }
```

Serialization

- Definition:
  - Supported by the Java Virtual Machine (JVM)
  - Ability to read or write an object to a stream
  - Process of "flattening" an object
  - Goal: To save object to some permanent storage or to pass on to another object via the *OutputStream* class
- Writing an object:
  - Its state should be written in a serialized form such that the object can be reconstructed as it is being read
- Persistence
  - Saving an object to some type of permanent storage

Serialization

- Streams for serialization
  - ObjectInputStream
    - For deserializing
  - ObjectOutputStream
    - For serializing

- To allow an object to be serializable:
  - Its class should implement the *Serializable* interface
  - Its class should also provide a default constructor or a constructor with no arguments
  - Serializability is inherited
    - Don't have to implement *Serializable* on every class
    - Can just implement *Serializable* once along the class heirarchy

Non-Serializable Objects

- When an object is serialized:
  - Only the object's data are preserved
  - Methods and constructors are not part of the serialized stream
- Some objects are not serializable
  - Because the data they represent constantly changes
  - Examples:
    - *FileInputStream* objects
    - *Thread* objects
- A *NotSerializableException* is thrown if the serialization fails

The *transient* Keyword

- A class containing a non-serializable object can still be serialized
  - Reference to non-serializable object is marked with the transient keyword
  - Example:

```
1 class MyClass implements Serializable {
2    transient Thread thread;
3    //try removing transient
4    int data;
5    /* some other data */
6 }
```

  - The *transient* keyword prevents the data from being serialized

Serialization: Writing an Object Stream

- Use the *ObjectOutputStream* class

- Use its *writeObject* method

```
public final void writeObject(Object obj)
                                    throws IOException
```

  where,

  - *obj* is the object to be written to the stream

Serialization: Writing an Object Stream

```
1  import java.io.*;
2  public class SerializeBoolean {
3     SerializeBoolean() {
4        Boolean booleanData = new Boolean("true");
5        try {
6           FileOutputStream fos = new
7                       FileOutputStream("boolean.ser");
8           ObjectOutputStream oos = new
9                       ObjectOutputStream(fos);
10          oos.writeObject(booleanData);
11          oos.close();
12 //continued...
```

Serialization: Writing an Object Stream

```
13          } catch (IOException ie) {
14              ie.printStackTrace();
15          }
16      }
17
18      public static void main(String args[]) {
19          SerializeBoolean sb = new SerializeBoolean();
20      }
21  }
```

Deserialization: Reading an Object Stream

- Use the *ObjectInputStream* class

- Use its *readObject* method

```
public final Object readObject()
        throws IOException, ClassNotFoundException
```

  where,

  - *obj* is the object to be read from the stream

- The *Object* type returned should be typecasted to the appropriate class name before methods on that class can be executed

Deserialization: Reading an Object Stream

```
1  import java.io.*;
2  public class UnserializeBoolean {
3      UnserializeBoolean() {
4        Boolean booleanData = null;
5        try {
6           FileInputStream fis = new
7                          FileInputStream("boolean.ser");
8           ObjectInputStream ois = new
9                          ObjectInputStream(fis);
10          booleanData = (Boolean) ois.readObject();
11          ois.close();
12 //continued...
```

Deserialization: Reading an Object Stream

```
13          } catch (Exception e) {
14              e.printStackTrace();
15          }
16          System.out.println("Unserialized Boolean from "
17                             + "boolean.ser");
18          System.out.println("Boolean data: " +
19                             booleanData);
20          System.out.println("Compare data with true: " +
21                  booleanData.equals(new Boolean("true")));
22      }
23  //continued...
```

Deserialization: Reading an Object Stream

```
13    public static void main(String args[]) {
14        UnserializeBoolean usb =
15                                new UnserializeBoolean();
16     }
17 }
```

Summary

- General Stream Types
  - Character and Byte Streams
  - Input and Output Streams
  - Node and Filter Streams

- The *File* Class
  - Constructor

    ```
    File(String pathname)
    ```

  - Methods

Summary

- *Reader* Classes
  - Methods
    - *read*, *close*, *mark*, *markSupported*, *reset*
  - Node *Reader* Classes
    - *FileReader*, *CharArrayReader*, *StringReader*, *PipedReader*
  - Filter *Reader* Classes
    - *BufferedReader*, *FilterReader*, *InputStreamReader*, *LineNumberReader*, *PushbackReader*

Summary

- *Writer* Classes
  - Methods
    - *write*, *close*, *flush*
  - Node *Writer* Classes
    - *FileWriter*, *CharArrayWriter*, *StringWriter*, *PipedWriter*
  - Filter *Writer* Classes
    - *BufferedWriter*, *FilterWriter*, *OutputStreamWriter*, *PrintWriter*

Summary

- *InputStream* Classes
  - Methods
    - *read*, *close*, *mark*, *markSupported*, *reset*
  - Node *InputStream* Classes
    - *FileInputStream*, *BufferedArrayInputStream*, *PipedInputStream*
  - Filter *InputStream* Classes
    - *BufferedInputStream*, *FilterInputStream*, *ObjectInputStream*, *DataInputStream*, *LineNumberInputStream*, *PushbackInputStream*

Summary

- *OutputStream* Classes
  - Methods
    - *write*, *close*, *flush*
  - Node *OutputStream* Classes
    - *FileOutputStream*, *BufferedArrayOutputStream*, *PipedOutputStream*
  - Filter *OutputStream* Classes
    - *BufferedOutputStream*, *FilterOutputStream*, *ObjectOutputStream*, *DataOutputStream*, *PrintStream*

Summary

- Serialization
  - Definition
  - The *transient* Keyword
  - Serialization: Writing an Object Stream
    - Use the *ObjectOutputStream* class
    - Use its *writeObject* method
  - Deserialization: Reading an Object Stream
    - Use the *ObjectInputStream* class
    - Use its *readObject* method