Java I/O

Hsuan-Tien Lin

Department of CSIE, NTU

OOP Class, May 16-17, 2010

Byte OutputStreams

java.io

OutputStream

- ByteArrayOutputStream
- FileOutputStream
- FilterOutputStream
  - BufferedOutputStream

PrintStream

System.out

```
/*
 * Copyright 1994-2004 Sun Microsystems, Inc.  All Rights Reserved.
 * ...
 */

package java.io;

/**
 * This abstract class is the superclass of all classes representing
 * an output stream of bytes. An output stream accepts output bytes
 * and sends them to some sink.
 */
public abstract class OutputStream implements Closeable, Flushable {
```

設數

```
/**
 * Writes the specified byte to this output stream. The general
 * contract for <code>write</code> is that one byte is written
 * to the output stream. The byte to be written is the eight
 * low-order bits of the argument <code>b</code>. The 24
 * high-order bits of <code>b</code> are ignored.
 * <p>
 * Subclasses of <code>OutputStream</code> must provide an
 * implementation for this method.
 *
 * @param      b   the <code>byte</code>.
 * @exception  IOException  if an I/O error occurs. In particular,
 *             an <code>IOException</code> may be thrown if the
 *             output stream has been closed.
 */
public abstract void write(int b) throws IOException;
```

```
/* ... */
public void write(byte b[]) throws IOException {
    write(b, 0, b.length);
}
```

```java
/** ...
 * The general contract for <code>write(b, off, len)</code> is that
 * some of the bytes in the array <code>b</code> are written to the
 * output stream in order; ...
 * The <code>write</code> method of <code>OutputStream</code> calls
 * write of one argument ..., subclasses are encouraged to override
 * this method and provide a more efficient implementation.
 * If <code>b</code> is <code>null</code>, ...
 */
public void write(byte b[], int off, int len) throws IOException {
    if (b == null) {
        throw new NullPointerException();
    } else if ((off < 0) || (off > b.length) || (len < 0) ||
               ((off + len) > b.length) || ((off + len) < 0)) {
        throw new IndexOutOfBoundsException();
    } else if (len == 0) {
        return;
    }
    for (int i = 0 ; i < len ; i++) {
        write(b[off + i]);
    }
}
```

overload: 7

```java
/**
 * Flushes this output stream and forces any buffered output bytes
 * to be written out. The general contract of <code>flush</code> is
 * that calling it is an indication that, if any bytes previously
 * written have been buffered by the implementation of the output
 * stream, such bytes should immediately be written to their
 * intended destination.
 * ...
 */
public void flush() throws IOException {
}
```

```
    /**
     * Closes this output stream and releases any system resources
     * associated with this stream. The general contract of
     * <code>close</code> is that it closes the output stream. A closed
     * stream cannot perform output operations and cannot be reopened.
     * <p>
     * The <code>close</code> method of <code>OutputStream</code> does
         nothing.
     *
     * @exception  IOException  if an I/O error occurs.
     */
    public void close() throws IOException {
    }
}
```

default bevior

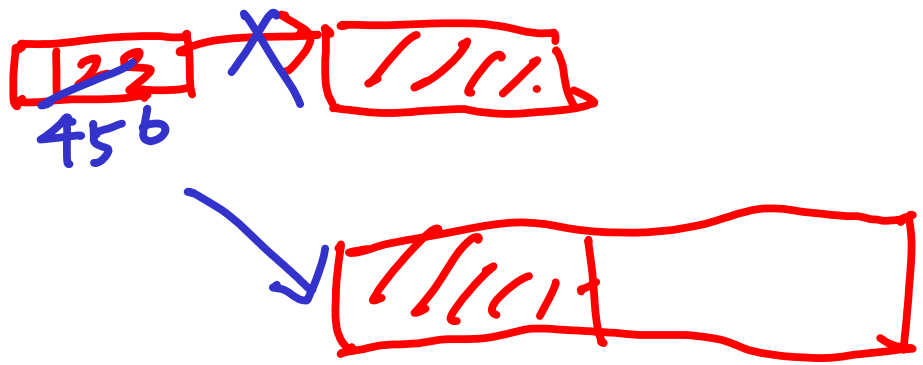
```
/*
 * Copyright 1994-2006 Sun Microsystems, Inc.  All Rights Reserved.
 * ...
 */
package java.io;
import java.util.Arrays;
/**
 * This class implements an output stream in which the data is
 * written into a byte array. The buffer automatically grows as data
 * is written to it.
 * The data can be retrieved using <code>toByteArray()</code> and
 * <code>toString()</code>.
 * ...
 */
public class ByteArrayOutputStream extends OutputStream {
    /**
     * The buffer where data is stored.
     */
    protected byte buf[];
    /**
     * The number of valid bytes in the buffer.
     */
    protected int count;
```

```java
/**
 * Creates a new byte array output stream. The buffer capacity is
 * initially 32 bytes, though its size increases if necessary.
 */
public ByteArrayOutputStream() {
    this(32);
}

/**
 * Creates a new byte array output stream, with a buffer capacity
    of
 * the specified size, in bytes.
 *
 * @param   size   the initial size.
 * @exception  IllegalArgumentException if size is negative.
 */
public ByteArrayOutputStream(int size) {
    if (size < 0) {
        throw new IllegalArgumentException(
          "Negative initial size: " + size);
    }
    buf = new byte[size];
}
```

constructor

RTE

```java
/**
 * Writes the specified byte to this byte array output stream.
 *
 * @param   b   the byte to be written.
 */
public synchronized void write(int b) {
    int newcount = count + 1;
    if (newcount > buf.length) {
        buf = Arrays.copyOf(buf, Math.max(buf.length << 1,
            newcount));
    }
    buf[count] = (byte)b;
    count = newcount;
}
```



```java
/**
 * Writes <code>len</code> bytes from the specified byte array
 * starting at offset <code>off</code> to this byte array output
     stream.
 * ...
 */
public synchronized void write(byte b[], int off, int len) {
    if ((off < 0) || (off > b.length) || (len < 0) ||
        ((off + len) > b.length) || ((off + len) < 0)) {
        throw new IndexOutOfBoundsException();
    } else if (len == 0) {
        return;
    }
    int newcount = count + len;
    if (newcount > buf.length) {
        buf = Arrays.copyOf(buf, Math.max(buf.length << 1,
            newcount));
    }
    System.arraycopy(b, off, buf, count, len);
    count = newcount;
}
```

```
/**
 * Writes the complete contents of this byte array output stream to
 * the specified output stream argument, as if by calling the
     output
 * stream's write method using <code>out.write(buf, 0,
     count)</code>.
 *
 * @param      out   the output stream to which to write the data.
 * @exception  IOException  if an I/O error occurs.
 */
public synchronized void writeTo(OutputStream out)
  throws IOException {
    out.write(buf, 0, count);
}
/**
 * Resets the <code>count</code> field of this byte array output
 * stream to zero, so that all currently accumulated output in the
 * output stream is discarded. The output stream can be used again,
 * reusing the already allocated buffer space.
 */
public synchronized void reset() {
    count = 0;
}
```

```java
/**
 * Creates a newly allocated byte array. Its size is the current
 * size of this output stream and the valid contents of the buffer
 * have been copied into it.
 *
 * @return  the current contents of this output stream, as a byte
     array.
 * @see     java.io.ByteArrayOutputStream#size()
 */
public synchronized byte toByteArray()[] {
    return Arrays.copyOf(buf, count);
}

/**
 * Returns the current size of the buffer.
 *
 * @return  the value of the <code>count</code> field, which is
     the number
 *          of valid bytes in this output stream.
 * @see     java.io.ByteArrayOutputStream#count
 */
public synchronized int size() {
    return count;
}
```

```java
/**
 * Converts the buffer's contents into a string decoding bytes
 * using the platform's default character set. The length of
 * the new <tt>String</tt> is a function of the character set,
 * and hence may not be equal to the size of the buffer.
 * ...
 */
public synchronized String toString() {
    return new String(buf, 0, count);
}

/**
 * Converts the buffer's contents into a string by decoding
 * the bytes using the specified
 * {@link java.nio.charset.Charset charsetName}....
 */
public synchronized String toString(String charsetName)
    throws UnsupportedEncodingException
{
    return new String(buf, 0, count, charsetName);
}
```

```
    /**
     * Creates a newly allocated string. Its size is the
     * current size of the output stream and the valid contents
     * of the buffer have been copied into it. Each character
     * <i>c</i> in the resulting string is constructed from the
     * corresponding element <i>b</i> in the byte array ...
     */
    @Deprecated
    public synchronized String toString(int hibyte) {
        return new String(buf, hibyte, 0, count);
    }

    /**
     * Closing a <tt>ByteArrayOutputStream</tt> has no effect.
     * The methods in this class can be called after the stream
     * has been closed without generating an <tt>IOException</tt>.
     */
    public void close() throws IOException {
    }
}
```

```
/*
 * Copyright 1994-2007 Sun Microsystems, Inc.  All Rights Reserved.
 * ...
 */

package java.io;

/**
 * A file output stream is an output stream for writing data to a
 * <code>File</code> or to a <code>FileDescriptor</code>.
 * ...
 */
public class FileOutputStream extends OutputStream
{
    /**
     * The system dependent file descriptor. The value is
     * 1 more than actual file descriptor. This means that
     * the default value 0 indicates that the file is not open.
     */
    private FileDescriptor fd;
```

```
public FileOutputStream(File file, boolean append)
    throws FileNotFoundException
{
    String name = (file != null ? file.getPath() : null);
    SecurityManager security = System.getSecurityManager();
    if (security != null) {
        security.checkWrite(name);
    }
    if (name == null) {
        throw new NullPointerException();
    }
    fd = new FileDescriptor();
    fd.incrementAndGetUseCount();
    this.append = append;
    if (append) {
        openAppend(name);
    } else {
        open(name);
    }
}
```

throws ...

```
public FileOutputStream(String name) throws FileNotFoundException {
    this(name != null ? new File(name) : null, false);
}
public FileOutputStream(String name, boolean append)
    throws FileNotFoundException
{
    this(name != null ? new File(name) : null, append);
}
public FileOutputStream(File file) throws FileNotFoundException {
    this(file, false);
}
```

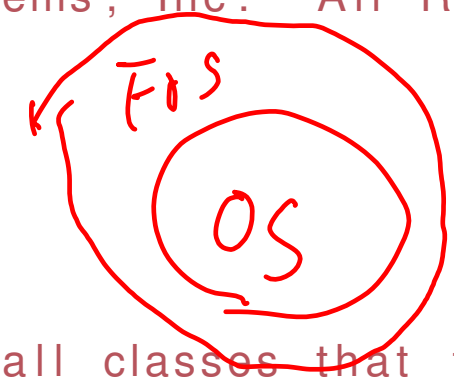
```
private native void open(String name) throws FileNotFoundException;

private native void openAppend(String name) throws
    FileNotFoundException;

public native void write(int b) throws IOException;

private native void writeBytes(byte b[], int off, int len) throws
    IOException;

public void write(byte b[]) throws IOException {
    writeBytes(b, 0, b.length);
}

public void write(byte b[], int off, int len) throws IOException {
    writeBytes(b, off, len);
}
```

```java
/**
 * Closes this file output stream and releases any system resources
 * associated with this stream. This file output stream may
 * no longer be used for writing bytes.
 * ...
 */
public void close() throws IOException {
    synchronized (closeLock) {
        if (closed) { return; }
        closed = true;
    }
    /* ... */
    /* Decrement FD use count associated with this stream */
    int useCount = fd.decrementAndGetUseCount();
    /*
     * If FileDescriptor is still in use by another stream,
     * the finalizer will not close it.
     */
    if ((useCount <= 0) || !isRunningFinalize()) {
        close0();
    }
}

private native void close0() throws IOException;
```

```
/**
 * Cleans up the connection to the file, and ensures that the
 * <code>close</code> method of this file output stream is
 * called when there are no more references to this stream.
 *
 * @exception  IOException  if an I/O error occurs.
 * @see        java.io.FileInputStream#close()
 */
protected void finalize() throws IOException {
    if (fd != null) {
        if (fd == fd.out || fd == fd.err) {
            flush();
        } else {
            runningFinalize.set(Boolean.TRUE);
            try {
                close();
            } finally {
                runningFinalize.set(Boolean.FALSE);
            }
        }
    }
}
```

```
    private static native void initIDs();

    static {
        initIDs();
    }

}
```

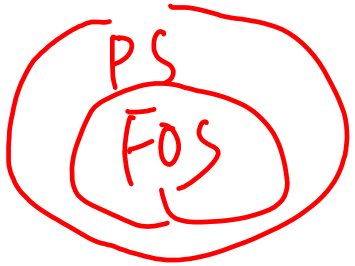
```
/*
 * Copyright 1994-1999 Sun Microsystems, Inc.  All Rights Reserved.
 * ...
 */

package java.io;

/**
 * This class is the superclass of all classes that filter output
 * streams. These streams sit on top of an already existing output
 * stream (the <i>underlying</i> output stream) which it uses as its
 * basic sink of data, but possibly transforming the data along the
 * way or providing additional functionality.
 * <p>
 * The class <code>FilterOutputStream</code> itself simply overrides
 * all methods of <code>OutputStream</code> with versions that pass
 * all requests to the underlying output stream. Subclasses of
 * <code>FilterOutputStream</code> may further override some of these
 * methods as well as provide additional methods and fields.
 *
 * @author  Jonathan Payne
 * @since   JDK1.0
 */
public class FilterOutputStream extends OutputStream {
```

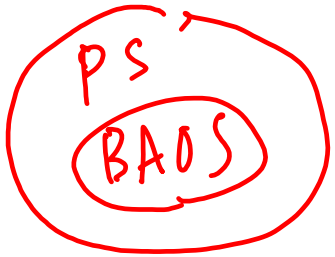


```
/**
 * The underlying output stream to be filtered.
 */
protected OutputStream out;

/**
 * Creates an output stream filter built on top of the specified
 * underlying output stream.
 *
 * @param   out   the underlying output stream to be assigned to
 *                the field <tt>this.out</tt> for later use, or
 *                <code>null</code> if this instance is to be
 *                created without an underlying stream.
 */
public FilterOutputStream(OutputStream out) {
    this.out = out;
}
```

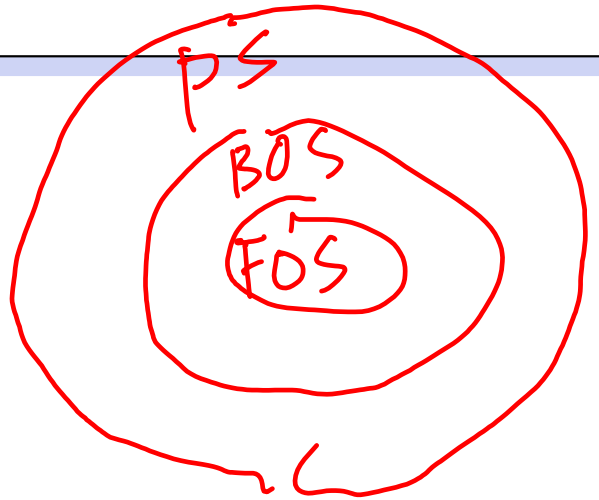
```
public void write(int b) throws IOException {
    out.write(b);
}

public void write(byte b[]) throws IOException {
    write(b, 0, b.length);
}

public void write(byte b[], int off, int len) throws IOException {
    if ((off | len | (b.length - (len + off)) | (off + len)) < 0)
        throw new IndexOutOfBoundsException();

    for (int i = 0 ; i < len ; i++) {
        write(b[off + i]);
    }
}
```

```
    public void flush() throws IOException {
        out.flush();
    }

    public void close() throws IOException {
        try {
          flush();
        } catch (IOException ignored) {
        }
        out.close();
    }
}
```





decorator



```
/*
 * Copyright 1994-2003 Sun Microsystems, Inc.  All Rights Reserved.
 * ...
 */

package java.io;

/**
 * The class implements a buffered output stream. By setting up such
 * an output stream, an application can write bytes to the underlying
 * output stream without necessarily causing a call to the underlying
 * system for each byte written.
 *
 * @author  Arthur van Hoff
 * @since   JDK1.0
 */
public class BufferedOutputStream extends FilterOutputStream {
    /** The internal buffer where data is stored.
     */
    protected byte buf[];
    /** The number of valid bytes in the buffer. ...
     */
    protected int count;
```

```java
/**
 * Creates a new buffered output stream to write data to the
 * specified underlying output stream.
 * ...
 */
public BufferedOutputStream(OutputStream out) {
    this(out, 8192);
}

/**
 * Creates a new buffered output stream to write data to the
 * specified underlying output stream with the specified buffer
 * size.
 */
public BufferedOutputStream(OutputStream out, int size) {
    super(out);
    if (size <= 0) {
        throw new IllegalArgumentException("Buffer size <= 0");
    }
    buf = new byte[size];
}
```

```
/** Flush the internal buffer */
private void flushBuffer() throws IOException {
    if (count > 0) {
        out.write(buf, 0, count);
        count = 0;
    }
}

/**
 * Writes the specified byte to this buffered output stream.
 *
 * @param      b   the byte to be written.
 * @exception  IOException  if an I/O error occurs.
 */
public synchronized void write(int b) throws IOException {
    if (count >= buf.length) {
        flushBuffer();
    }
    buf[count++] = (byte)b;
}
```

```java
/**
 * Writes <code>len</code> bytes from the specified byte array
 * starting at offset <code>off</code> to this buffered output
 * stream. ...
 */
public synchronized void write(byte b[], int off, int len)
  throws IOException {
    if (len >= buf.length) {
        /* If the request length exceeds the size of the output
           buffer, flush the output buffer and then write the
           data directly. In this way buffered streams will
           cascade harmlessly. */
        flushBuffer();
        out.write(b, off, len);
        return;
    }
    if (len > buf.length - count) {
        flushBuffer();
    }
    System.arraycopy(b, off, buf, count, len);
    count += len;
}
```

```
    /**
     * Flushes this buffered output stream. This forces any buffered
     * output bytes to be written out to the underlying output stream.
     *
     * @exception  IOException  if an I/O error occurs.
     * @see        java.io.FilterOutputStream#out
     */
    public synchronized void flush() throws IOException {
        flushBuffer();
        out.flush();
    }
}
```

Byte OutputStreams vs. Char Writers

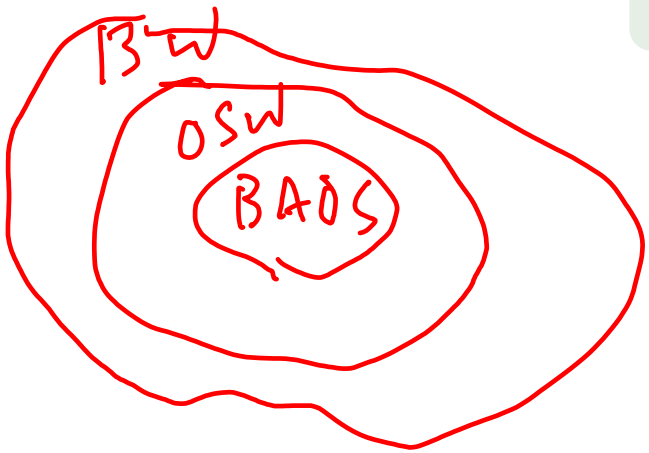
```
FileOutputStream fos = new ...;
BufferedOutputStream bos = new ...(fos);
PrintStream ps = new ...(bos);
OutputStreamWriter osw = new ..(ps);
```

OutputStream

- ByteArrayOutputStream
- FileOutputStream
- FilterOutputStream
  - BufferedOutputStream

Writer

- CharArrayWriter
- FilterWriter (abstract)
- BufferedWriter
- OutputStreamWriter
  - FileWriter



Byte OutputStreams

OutputStream

- ByteArrayOutputStream
- FileOutputStream
- FilterOutputStream
  - BufferedOutputStream
  - PrintStream

System.out

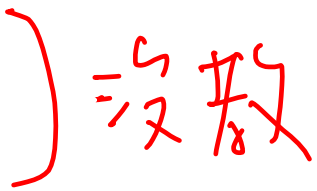
```
/*
 * Copyright 1996-2006 Sun Microsystems, Inc.  All Rights Reserved.
 * ...
 */

package java.io;

/**
 * A <code>PrintStream</code> adds functionality to another output
 * stream, namely the ability to print representations of various
 * data values conveniently.  Two other features are provided as
 * well.  Unlike other output streams, a <code>PrintStream</code>
 * never throws an <code>IOException</code>; instead, exceptional
 * situations merely set an internal flag that can be tested via
 * the <code>checkError</code> method.
 * Optionally, a <code>PrintStream</code> can be created so as to flush
 * automatically; this means that the <code>flush</code> method is
 * automatically invoked after a byte array is written, one of the
 * <code>println</code> methods is invoked, ...
 */

public class PrintStream extends FilterOutputStream
    implements Appendable, Closeable
{
```

```
    private boolean autoFlush = false;
    private boolean trouble = false;

    /**
     * Track both the text- and character-output streams, so that
         their buffers
     * can be flushed without flushing the entire stream.
     */
    private BufferedWriter textOut;
    private OutputStreamWriter charOut;
```

```
/* Initialization is factored into a private constructor
 * (note the swapped parameters so that this one isn't
 * confused with the public one) and a separate init method
 * so that the following two public constructors can share code.
 * ...
 */
private PrintStream(boolean autoFlush, OutputStream out)
{
    super(out);
    if (out == null)
        throw new NullPointerException("Null output stream");
    this.autoFlush = autoFlush;
}
private void init(OutputStreamWriter osw) {
    this.charOut = osw;
    this.textOut = new BufferedWriter(osw);
}
public PrintStream(OutputStream out, boolean autoFlush) {
    this(autoFlush, out);
    init(new OutputStreamWriter(this));
}
public PrintStream(OutputStream out) {
    this(out, false);
}
```

```
public void flush() {
    synchronized (this) {
        try {
            ensureOpen();
            out.flush();
        }
        catch (IOException x) {
            trouble = true;
        }
    }
}
public void close() {
    //similar to flush()
}

public void write(int b) {
    //similar to flush()
}

public void write(byte buf[], int off, int len) {
    //similar to flush()
}
```

```
/*
 * The following private methods on the text- and
 * character-output streams always flush the stream
 * buffers, so that writes to the underlying byte
 * stream occur as promptly as with the original PrintStream.
 */
private void write(String s) {
    try {
        synchronized (this) {
            ensureOpen();
            textOut.write(s);
            textOut.flushBuffer();
            charOut.flushBuffer();
            if (autoFlush && (s.indexOf('\n') >= 0))
                out.flush();
        }
    }
    catch (InterruptedIOException x) {
        Thread.currentThread().interrupt();
    }
    catch (IOException x) {
        trouble = true;
    }
}
```



```
public void print(double d) {
    write(String.valueOf(d));
}

/**
 * Prints an object.
 * ...
 * @see         java.lang.Object#toString()
 */
public void print(Object obj) {
    write(String.valueOf(obj));
}

public void println(Object x) {
    String s = String.valueOf(x);
    synchronized (this) {
        print(s);
        newLine();
    }
}
```

```
    /**
     * @since: 1.5
     */
    public PrintStream printf(String format, Object ... args) {
        return format(format, args);
    }

    /**
     * Writes a formatted string to this output stream
     * using the specified format string and arguments.
     * ...
     */
    public PrintStream format(String format, Object ... args) {
        //some more complicated code with another class
    }
}
```

var. arg list

Byte Input/OutputStreams vs. Char Reader/Writers

OutputStream

- ByteArrayOutputStream
- FileOutputStream
- FilterOutputStream
  - BufferedOutputStream
  - PrintStream

Writer

- CharArrayWriter
- FilterWriter (abstract)
- BufferedWriter
- OutputStreamWriter
  - FileWriter
- PrintWriter

Formatter

InputStream

- ByteArrayInputStream
- FileInputStream
- FilterInputStream
  - BufferedInputStream

Reader

- CharArrayReader
- FilterReader (abstract)
- BufferedReader
- InputStreamReader
  - FileReader

Scanner