Introduction to Interactive Programming
by Lynn Andrea Stein
A Rethinking CS101 Project

Java.io Quick Reference

These appendices are intended to provide a quick reference to that part of Java that is likely to be useful to a student reading this book. No additional reference should be necessary to understand what is contained in the body of this book. However, the class documentation here is not intended to be complete or exhaustive. For a comprehensive listing of the methods and other properties of these classes, read the Java API documentation.

InputStream and Reader

Both java.io.InputStream and java.io.Reader are abstract classes, that is, there are no instances of InputStream (or of Reader) that are not also instances of a subclass of InputStream (or of Reader).

Each InputStream or Reader (generically known as a stream) represents an ordered sequence of readable Things. If a read request is made and no Thing is in the stream, the read request blocks until the next Thing becomes available.

The difference between an InputStream and a Reader is that an InputStream is at base a stream of bytes, while a Reader is at base a stream of chars.

InputStream supports the following methods:

- public int read() throws IOException; *reads and returns the next byte from the stream*

- public int available() throws IOException; *returns the number of bytes that can be read without blocking*
- public void close() throws IOException; *should be called when you are done with the stream*

Reader supports the following methods:

- public int read() throws IOException; *reads and returns the next char from the stream*
- public boolean ready() throws IOException; *returns true if the next read() will not block*
- public void close() throws IOException; *should be called when you are done with the stream*

If you have an InputStream, you can create a Reader using an InputStreamReader.

OutputStream and Writer

Both java.io.OutputStream and java.io.Writer are abstract classes. Each of these classes represents a resource to which Things can be written.

The difference between an InputStream and a Reader is that an InputStream is at base a stream of bytes, while a Reader is at base a stream of chars.

InputStream supports the following methods:

- public void write(int b) throws IOException; *writes a byte to the stream*
- public void flush() throws IOException; *makes sure that any writes have actually happened*
  - *without this method, sometimes writes get queued up....*
- public void close() throws IOException; *should be called when you are done with the stream*
  - *includes a call to flush()*

Writer supports the following methods:

- public void write(int c) throws IOException; *writes a character to the stream*
- public void write(String s) throws IOException; *writes a String to the stream*
- public void flush() throws IOException; *makes sure that any writes have actually happened*
  - *without this method, sometimes writes get queued up....*
- public void close() throws IOException; *should be called when you are done with the stream*
  - *includes a call to flush()*

If you have an OutputStream, you can create a Writer using an OutputStreamWriter.

Sources of Streams

There are several different ways to generate an InputStream. One is to use System.in, the "standard input" stream built in to every running Java program. Other ways involve using some resource to read from.

- java.lang.System.in is an InputStream that is available to every Java program.
- FileInputStream is a class whose constructor opens a file for reading.
- PipedInputStream is a class that can be used to create a stream between two running Java Threads.
- Sockets have InputStreams that (potentially) connect multiple computers.
- Other InputStream types include those that read from a ByteArray or from a Sequence of other InputStreams.

Like an InputStream, a Reader can be generated from a system resource. A Reader can also be generated from an InputStream.

- InputStreamReader's constructor takes an InputStream and creates a Reader that reads from that underlying stream.
- FileReader is an InputStreamReader that can directly produce a Reader from a File or filename.
- PipedReader can create a stream between two Threads.
- Other Reader types include StringReader and CharArrayReader.

There are also several different ways to generate an OutputStream. There are two built-in OutputStreams, System.out, the "standard output", and System.err, the "standard error" stream. Other OutputStreams can be constructed from resources:

- java.lang.System.out and java.lang.System.err are OutputStreams available to every Java program.
- FileOutputStream is a class whose constructor opens a file for writing, creating it if necessary.
- PipedOutputStream is a class that can be used to create a stream between two running Java Threads.
- Sockets have OutputStreams that (potentially) connect multiple computers.
- An OutputStream can also write to a ByteArray.

Writers mimic OutputStreams in the same way that Readers mimic InputStreams. A Writer can be generated from any OutputStream, or from a system resource directly.

- OutputStreamWriter's constructor takes an OutputStream and creates a Writer that reads from that underlying stream.
- FileWriter is an OutputStreamWriter that can directly produce a Writer from a File or filename.
- PipedWriter can create a stream between two Threads.
- Other Writer types include StringWriter and CharArrayWriter.

InputStreamReader and OutputStreamWriter

If you have an underlying byte stream and want a character stream, making one is as simple as calling the appropriate constructor. InputStreamReader has the methods described above for Reader; OutputStreamWriter implements the methods described for Writer.

java.io.InputStreamReader

Makes a Reader out of an InputStream

- constructor
  - public InputStreamReader(InputStream in)

java.io.OutputStreamWriter

Makes a Writer out of an OutputStream

- constructor
  - public OutputStreamWriter (OutputStream out)

Files

The File class is a platform-independent way to refer to directories and Files by name.

The corresponding stream classes are fairly unexceptional. Use the Reader/Writer classes to read and write text, the InputStream/OutputStream to manipulate raw data.

It is generally a good idea to use buffering when reading from or writng to a file. When reading from a file, the value -1 will be returned when the end of the file is reached.

To create a File, use the first constructor of FileOutputStream or FileWriter.

java.io.File

A platform-independent way to refer to directories and files. See also FileOutputStream to create a File.

- implements Serializable
- constructors
  - public File(String path);
  - public File(String path, String name);
  - public File(File directory, String name);
- constants: platform-dependent directory and path separators are preset for you.
  - public static final String pathSeparator;
  - public static final char pathSeparatorChar;
  - public static final String separator;
  - public static final char separatorChar;
- predicates:
  - public boolean canRead();
  - public boolean canWrite();
  - public boolean exists();

        - public boolean isAbsolute();
        - public boolean isDirectory();
        - public boolean isFile();
    - "selectors":
        - public String getName();
        - public String getParent();
        - public String getPath();
    - get information about the file or directory:
        - public long lastModified();
        - public long length();
        - public String[] list(); *// lists the files in the directory*
    - manipulate the file:
        - public boolean delete();
        - public boolean mkdir();
        - public boolean renameTo(File newName);

java.io.FileInputStream

- extends InputStream
- constructors
    - public FileInputStream(String name) throws FileNotFoundException;
    - public FileInputStream(File file) throws FileNotFoundException;

java.io.FileReader

- extends InputStreamReader
- constructors
    - public FileReader(String name) throws FileNotFoundException;
    - public FileReader(File file) throws FileNotFoundException;

java.io.FileOutputStream

- extends OutputStream
- constructors
    - public FileOutputStream(String name) throws IOException;
    - public FileOutputStream(String name, boolean append) throws IOException;
    - public FileOutputStream(File file) throws IOException;

java.io.FileWriter

- extends OutputStream Writer

- constructors
  - public FileWriter(String name) throws IOException;
  - public FileWriter(String name, boolean append) throws IOException;
  - public FileWriter(File file) throws IOException;

Pipes

Pipes are useful for communication between Threads. A PipedInputStream must be connected to a PipedOutputStream, either at construction or using the connect() method. Similarly, a PipedReader must be connected to a PipedWriter.

java.io.PipedInputStream

- extends InputStream
- constructor
  - public PipedInputStream(PipedOutputStream stream) throws IOException;
  - public PipedInputStream();
- additional method
  - public void connect(PipedOutputStream stream) throws IOException;

java.io.PipedOutputStream

- extends OutputStream
- constructor
  - public PipedOutputStream(PipedInputStream stream) throws IOException;
  - public PipedOutputStream();
- additional method
  - public void connect(PipedInputStream stream) throws IOException;

java.io.PipedReader

- extends Reader
- constructor
  - public PipedReader(PipedWriter stream) throws IOException;
  - public PipedReader();
- additional method
  - public void connect(PipedWriter stream) throws IOException;

java.io.PipedWriter

- extends Writer

- constructor
  - public PipedWriter(PipedReader stream) throws IOException;
  - public PipedWriter();
- additional method
  - public void connect(PipedReader stream) throws IOException;

Sockets

The classes Socket and ServerSocket are part of java.net. They are another source of streams, in this case streams that bridge across the network.

java.net.Socket

The class java.net.Socket represents a virtual connection to another machine. A Socket has an InputStream and an OutputStream.

- constructors
  - public Socket(String host, int port) throws UnknownHostException, IOException;
  - public Socket(InetAddress address, int port) throws IOException;
  - public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException;
  - public Socket(InetAddress address, int port, InetAddress localAddress, int localPort) throws IOException;
- stream methods:
  - public InputStream getInputStream();
  - public OutputStream getOutputStream throws IOException();
- when done with the Socket:
  - public synchronized void close() throws IOException;
- other selectors:
  - public InetAddress getInetAddress();
  - public InetAddres getLocalAddress();
  - public int getLocalPort();
  - public int getPort();

java.net.ServerSocket

A Socket can be used to connect to a ServerSocket.

- constructors
  - public Socket(int port) throws IOException;

- listen for a connection:
  - public Socket accept() throws IOException();
- when done with the ServerSocket:
  - public void close() throws IOException;
- other selectors:
  - public InetAddress getInetAddress();
  - public int getLocalPort();

java.net.InetAddress

A constructorless class that represents an internet address.

- pseudo-constructors
  - public static InetAddress getLocalHost() throws UnknownHostException; *// what machine are you running on?*
  - public static InetAddress getByName(String host) throws UnknownHostException; *// what is* host*'s address?*
  - public static InetAddress[] getAllByName(String host) throws UnknownHostException; *// more useful if* host *has many addresses.*
- selectors for humans
  - public String getHostName();
  - public String getHostAddress();

The package Java.net also contains classes for manipulating urls and http (i.e., the web) directly.

Streams that Add Features

The InputStream classes listed above create InputStreams. Other InputStream classes have constructors that take any InputStream and produce a new InputStream with additional functionality. Similar classes exist for Readers, OutputStreams, and Writers.

- You might want to read or write bigger chunks from the stream. This can be done with BufferedInputStream, BufferedReader, BufferedOutputStream or BufferedWriter.
- You might want to read or write a variety of Java primitive types. To do so, use DataInputStream or DataOutputStream.
- You might want to read or write a variety of Java primitive *and* Object types (or simply Object types). The appropriate classes are ObjectInputStream and ObjectOutputStream
- Other stream types (not documented here) include
  - Pushback, which allow you to return something you've read
  - Filter, which provides general infrastructure for only letting part of the stream through.
- See also LineNumberReader and PrintStream, below.

java.io.BufferedInputStream

Reads a larger chunk of data from the underlying stream and stores it in a buffer, then on individual read() calls reads from this buffer as long as data is available there. Often used when reading from disk or from the network.

- extends FilterInputStream
- constructors
  - public BufferedInputStream(InputStream in);
  - public BufferedInputStream(InputStream in, int bufferSize);
- Certain InputStream methods are synchronized:
  - public synchronized int available() throws IOException;
  - public synchronized int read() throws IOException;

java.io.BufferedOutputStream

Writes to a buffer, then when the buffer is full (or when flush() is called) writes the whole thing to the underlying stream at once. Often used when writing to disk.

- extends FilterOutputStream
- constructors
  - public BufferedOutputStream(OutputStream out);
  - public BufferedOutputStream(OutputStream out, int bufferSize);
- Certain OutputStream methods are synchronized:
  - public synchronized void write(int b) throws IOException;
  - public synchronized void flush() throws IOException;

java.io.BufferedReader

Reads a larger chunk of data from the underlying stream and stores it in a buffer, then on individual read() calls reads from this buffer as long as data is available there. Often used when reading from a File.

- extends Reader
- constructors
  - public BufferedReader(Reader in);
  - public BufferedReader(Reader in, int bufferSize);
- Added method for reading a whole line
  - public String readLine() throws IOException;
    - *note: returned String does not include the end-of-line (carriage return or linefeed) character*

java.io.BufferedWriter

Writes to a buffer, then when the buffer is full (or when flush() is called) writes the whole thing to the

- Added method for writing the end-of-line character
  - public void newLine() throws IOException;

Streams that Add Features

The InputStream classes listed above create InputStreams. Other InputStream classes have constructors that take any InputStream and produce a new InputStream with additional functionality. Similar classes exist for Readers, OutputStreams, and Writers.

- You might want to read or write bigger chunks from the stream. This can be done with BufferedInputStream, BufferedReader, BufferedOutputStream or BufferedWriter.
- You might want to read or write a variety of Java primitive types. To do so, use DataInputStream or DataOutputStream.
- You might want to read or write a variety of Java primitive *and* Object types (or simply Object types). The appropriate classes are ObjectInputStream and ObjectOutputStream
- Other stream types (not documented here) include
  - Pushback, which allow you to return something you've read
  - Filter, which provides general infrastructure for only letting part of the stream through.
- See also LineNumberReader and PrintStream, below.

Buffering

Buffering is a way of combining multiple reads or multiple writes into a single action. It is primarily used to increase efficiency, not to obtain additional functionality. However, BufferedReader is independently useful because it has a readLine() method that reads in a whole line of text; BufferedWriter has a corresponding newLine() method.

java.io.BufferedInputStream

Reads a larger chunk of data from the underlying stream and stores it in a buffer, then on individual read() calls reads from this buffer as long as data is available there. Often used when reading from disk or from the network.

- extends FilterInputStream
- constructors
  - public BufferedInputStream(InputStream in);
  - public BufferedInputStream(InputStream in, int bufferSize);
- Certain InputStream methods are synchronized:
  - public synchronized int available() throws IOException;
  - public synchronized int read() throws IOException;

java.io.BufferedOutputStream

Writes to a buffer, then when the buffer is full (or when flush() is called) writes the whole thing to the underlying stream at once. Often used when writing to disk.

- extends FilterOutputStream
- constructors
  - public BufferedOutputStream(OutputStream out);
  - public BufferedOutputStream(OutputStream out, int bufferSize);
- Certain OutputStream methods are synchronized:
  - public synchronized void write(int b) throws IOException;
  - public synchronized void flush() throws IOException;

java.io.BufferedReader

Reads a larger chunk of data from the underlying stream and stores it in a buffer, then on individual read() calls reads from this buffer as long as data is available there. Often used when reading from a File.

- extends Reader
- constructors
  - public BufferedReader(Reader in);
  - public BufferedReader(Reader in, int bufferSize);
- Added method for reading a whole line
  - public String readLine() throws IOException;
    - *note: returned String does not include the end-of-line (carriage return or linefeed) character*

java.io.BufferedWriter

Writes to a buffer, then when the buffer is full (or when flush() is called) writes the whole thing to the underlying stream at once. Often used when writing to a File.

- extends Writer
- constructors
  - public BufferedWriter(Writer out);
  - public BufferedWriter(Writer out, int bufferSize);
- Added method for writing the end-of-line character
  - public void newLine() throws IOException;

Primitive Data

To read and write primitive data types, Java provides two classes with appropriate methods.

java.io.DataInputStream

Useful for reading Java primitve types.

- extends FilterInputStream
- implements DataInput
- constructor
    - public DataInputStream(InputStream in) throws IOException;
- Additional read methods:
    - public boolean readBoolean() throws IOException;
    - public byte readByte() throws IOException;
    - public char readChar() throws IOException;
    - public double readDouble() throws IOException;
    - public float readFloat() throws IOException;
    - public int readInt() throws IOException;
    - public long readLong() throws IOException;
    - public short readShort() throws IOException;

java.io.DataOutputStream

Useful for writing Java primitve types.

- extends FilterOutputStream
- implements DataOutput
- constructor
    - public DataOutputStream(OutputStream out) throws IOException;
- Synchronizes an inherited method:
    - public synchronized void write(int b) throws IOException;
- Additional write methods:
    - public void writeBoolean(boolean b) throws IOException;
    - public void writeByte(int b) throws IOException;
    - public void writeBytes(String s) throws IOException;
    - public void writeChar(int c) throws IOException;
    - public void writeChars(String s) throws IOException;
    - public void writeDouble(double d) throws IOException;
    - public void writeFloat(float f) throws IOException;
    - public void writeInt(int i) throws IOException;
    - public void writeLong(long l) throws IOException;
    - public void writeShort(int s) throws IOException;

Objects

To read and write Objects as well as primitive data types, Java provides two additional classes that support all of the methods of DataInput/OutputStream, plus additional support for Object reading and writing.

Note that an Object to be written or read must implement the Serializable interface. Note also that Object streams do a lot of additional work in packaging/unpackaging Objects to read or write them. If you are using a stream in a time-critical way -- such as to send information between two players of a fast-paced video game -- you may wish to send primitive data, such as ints, rather than Objects encapsulating that data, such as Points.

java.io.ObjectInputStream

Useful for reading Serializable Objects as well as Java primitve types.

- extends InputStream
- implements ObjectInput
- constructor
  - public ObjectInputStream(InputStream in) throws IOException, StreamCorruptedException;
- Object read method:
  - public Object readObject();
- Plus the primitive data methods listed in DataInputStream

java.io.ObjectOutputStream

Useful for writing Serializable Objects as well as Java primitve types.

ava primitve types.

- extends OutputStream
- implements ObjectOutput
- constructor
  - public ObjectOutputStream(OutputStream out) throws IOException;
- Object write method:
  - public void writeObject(Object o);
- Plus the primitive data methods listed in DataOutputStream

Object Streams and Serialization

If you want to read and write Objects as well as primitive data types, you should use Java's ObjectInput and OutputStream classes. These classes support all of the methods of DataInput/OutputStream, plus additional support for Object reading and writing.

Note that an Object to be written or read must implement the Serializable interface.

java.io.Serializable

- Must be implemented by an object to be written to or read from a file.
- Has no methods.
- If the class involves (non-transient) fields with Object types, these fields must be Serializable.
- For complex objects (including those with Threads), additional measures may need to be taken.

java.io.ObjectInputStream

Useful for reading Serializable Objects as well as Java primitve types.

- extends InputStream
- implements ObjectInput
- constructor
  - public ObjectInputStream(InputStream in) throws IOException, StreamCorruptedException;
- Object read method:
  - public Object readObject();
- Plus the primitive data methods listed in DataInputStream

java.io.ObjectOutputStream

Useful for writing Serializable Objects as well as Java primitve types.

ava primitve types.

- extends OutputStream
- implements ObjectOutput
- constructor
  - public ObjectOutputStream(OutputStream out) throws IOException;
- Object write method:
  - public void writeObject(Object o);
- Plus the primitive data methods listed in DataOutputStream

Other Useful Streams

java.io.PrintStream

This is the simplest class for writing. It can write any type, with or without a line terminator following. None of its methods throw exceptions. System.out and System.err are PrintStreams.

- extends FilterOutputStream
- methods to print everthing:

        - public void print(boolean b);
        - public void print(char c);
        - public void print(int i);
        - public void print(long l);
        - public void print(float f);
        - public void print(double d);
        - public void print(String s);
        - public void print(Object obj);
    - method to end the line:
        - public void println();
    - plus a println method identical to each print method, above.
    - and flush(), close(), etc. inherited from OutputStream

java.io.PrintWriter

Unfortunately, you can't make a PrintStream. PrintWriter is similar.

- extends Writer
- constructors
    - public PrintWriter(Writer out);
    - public PrintWriter(Writer out, boolean flushOnPrintln);
    - public PrintWriter(OutputStream out);
    - public PrintWriter(OutputStream out, boolean flushOnPrintln);
- The additional methods of a PrintWriter are as listed above for PrintStream

java.io.LineNumberReader

In case you want to know what number line you're reading.

- extends BufferedReader
- constructor
    - public LineNumberReader(Reader in);
- additional methods:
    - public int getLineNumber();
    - public void setLineNumber();

java.io.SequenceInputStream

This is useful when you want to pick up reading from one stream as soon as another one runs out of input.

- extends InputStream

- constructors
  - public SequenceInputStream(InputStream s1, InputStream s2);
  - public SequenceInputStream(Enumeration e);

IOExceptions

The following classes are defined in the package java.io, and each extends java.io.IOException.

- CharConversionException
- EOFException
- FileNotFoundException
- InterruptedIOException
- InvalidClassException
- InvalidObjectException
- NotActiveException
- NotSerializableException
- ObjectStreamException
- OptionalDataException
- StreamCorruptedException
- SyncFailedException
- UnsupportedEncodingException
- UTFDataFormatException
- WriteAbortedException

The following additional IOExceptions are defined in the package java.net

- BindException
- ConnectException
- MalformedURLException
- NoRouteToHostException
- ProtocolException
- SocketException
- UnknownHostException
- UnknownServiceException

formerly at the MIT AI Lab and the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology.

Questions or comments:
<webmaster@cs101.org>