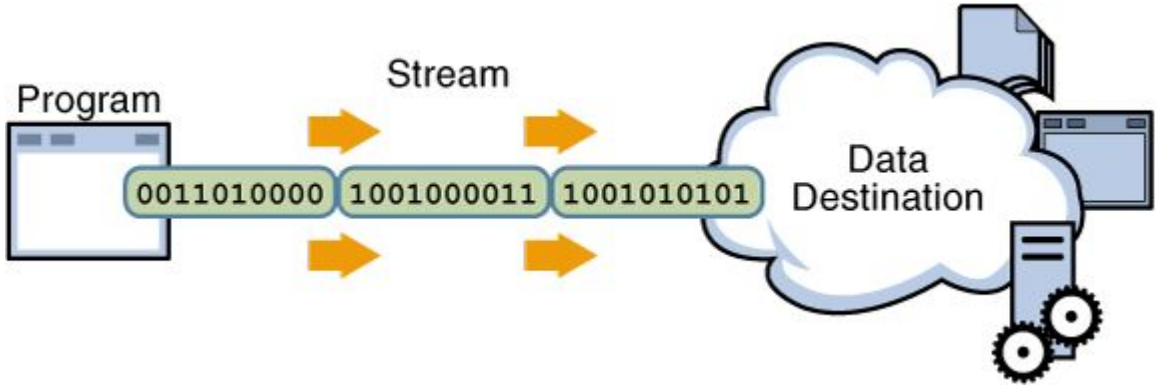
Java Input Output

Java Input/Output

Stream

Streams facilitate transporting data from one place to another. Different streams are needed to send or receive data through different sources such as keyboard, monitor, files etc.



Java Input/Output

The Stream Classes

1. InputStream & OutputStream (Designed for byte streams)

2. Reader & Writer (Designed for character streams)

- In general, we should use the character steam classes when working with characters or strings.
- We should use the byte steam classes when working with bytes or other binary objects like image, audio, video.

Java Input/Output

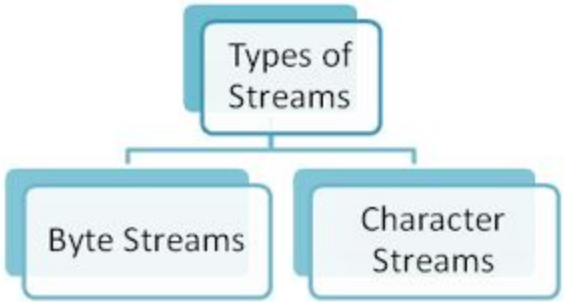
The Stream Classes

1. InputStream & OutputStream (Designed for byte streams)

2. Reader & Writer (Designed for character streams)

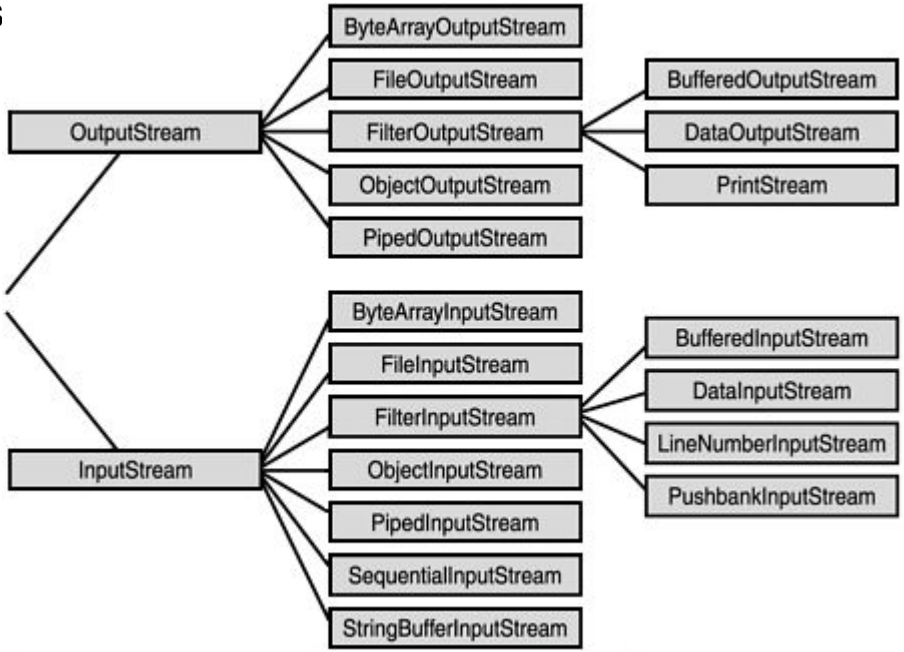
- In general, we should use the character steam classes when working with characters or strings.
- We should use the byte steam classes when working with bytes or other binary objects like image, audio, video.

Java Input/Output



Java Input/Output

Byte Stream Classes

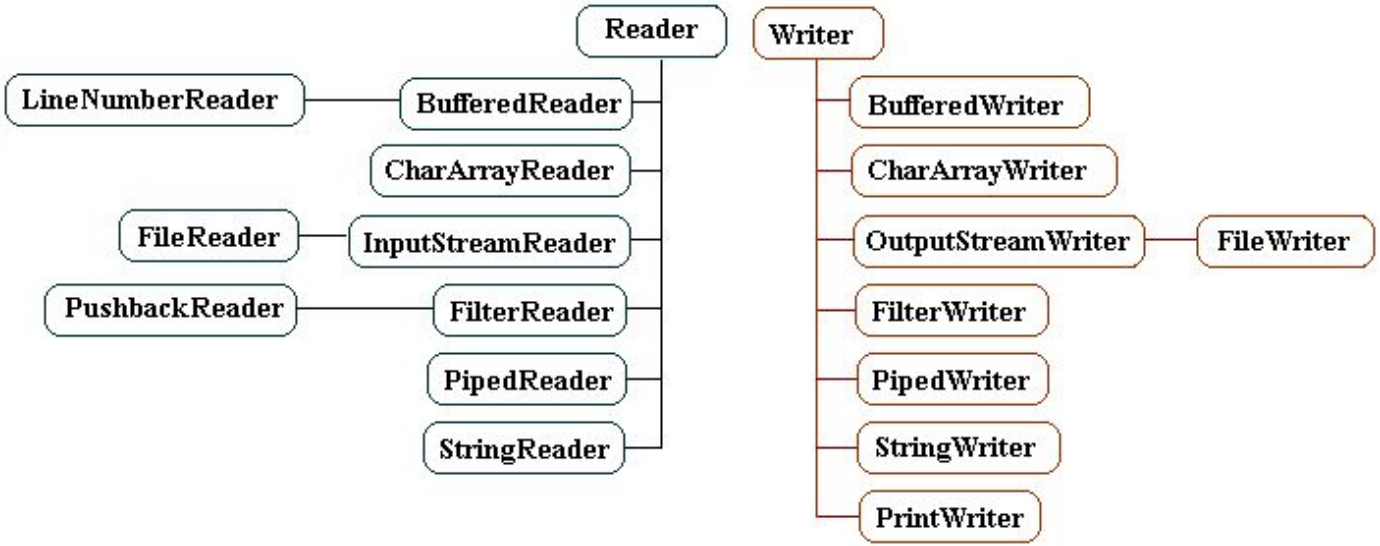


Java Input/Output

Character

Stream

Classes



Java Input/Output

Byte Streams :InputStream Methods

- int available () : Returns the number of bytes of input currently available for reading.
- int read() : Returns integer representation of the next available byte. Returns -1 when the end of file is encountered.
- int read(byte buffer[]) : Reads up to buffer.length bytes into buffer. Returns -1 when the end of file is encountered.
- int read(byte buffer[],int offset,int numBytes) : Read up to numBytes bytes into buffer starting at buffer[offset]. Returns -1 when the end of file is encountered.
- void close(): Closes the input stream.

Java Input/Output

ByteStream: OutputStream Methods

- void write(int b) : Writes a single byte to an output stream.
- void write(byte buffer[]) : Writes a complete array of bytes to an output stream.
- void write(byte buffer[],int offset,int numBytes) : Writes a subrange of numBytes bytes from the array buffer, begining at buffer[offset].
- void close(): Closes the output stream.
- void flush(): It flushes out the output buffers

Java Input/Output

- FileInputStream : To read from file
- ByteArrayInputStream : To read from byte array
- BufferedInputStream : Buffer based reading to make the reading process faster.

- FileOutputStream : To write to the file
- ByteArrayOutputStream : To write to the byte array
- BufferedOutputStream : Buffer based writing to make the writing process faster.

Java Input/Output

- Advantage Of Buffered Stream

In the unbuffered I/O, each read or write request is handled directly by the underlying OS. This can make a program much less efficient, since each such request often triggers disk access, network activity, or some other operation that is relatively expensive.

To reduce this kind of overhead, the Java platform implements buffered I/O streams. Buffered input streams read data from a memory area known as a buffer; **the native input API is called only when the buffer is empty**. Similarly, buffered output streams write data to a buffer, and **the native output API is called only when the buffer is full**.

Java Input/Output

- Flushing Buffered Stream

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. See Formatting for more on these methods.

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.

Java Input/Output

Note:

- FileOutputStream fos = new FileOutputStream("target.txt",**true**); //If we want to append output to the target file
- Buffered classes should be used **always in connection to other stream** classes. For example, BufferedOutputStream can be used along with FileOutputStream to write data into a file.

Java Input/Output

Character Streams

Byte stream classes can not work directly with Unicode characters. Character Stream classes include direct I/O support for characters.

Java Input/Output

Character Streams: Reader Methods

- int read() : Returns integer representation of the next available character. Returns -1 when the end of file is encountered.
- int read(char buffer[]) : Reads up to buffer.length characters into buffer. Returns -1 when the end of file is encountered.
- int read(char buffer[],int offset,int numChars) : Read up to numChars characters into buffer starting at buffer[offset]. Returns -1 when the end of file is encountered.
- void close(): Closes the input source.

Java Input/Output

Character Streams: Writer Methods

- void write(int ch) : Writes a single character to an output stream.
- void write(char buffer[]) : Writes a complete array of characters to an output stream.
- void write(char buffer[],int offset,int numChars) : Writes a subrange of numChars characters from the array buffer, begining at buffer[offset].
- void close(): Closes the output stream.
- void flush(): It flushes out the output buffers
- Writer append(char ch) : Appends a single character to the the end of output stream. Returns a reference to the invoking stream.
- Writer append(CharSequence ch) : Appends characters to the the end of output stream. Returns a reference to the invoking stream.

Java Input/Output

- FileReader : To read from file
- CharArrayReader : To read from byte array
- BufferedReader : Buffer based reading to make the reading process faster.

- FileWriter : To write to the file
- CharArrayWriter : To write to the byte array
- BufferedWriter : Buffer based writing to make the writing process faster.

Note: Without buffering, each invocation of read() or readLine() could cause bytes to be read from the file, converted into characters, and then returned, which can be very inefficient.

Java Input/Output

System.in => Represents keyboard (InputStream class object)

System.out => Represents Monitor (PrintStream class object)

System.err => Represents Monitor (PrintStream class object)

Java Input/Output

Reading input from keyboard

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

char c = (char) br.read(); //casting is required as read() returns integer value of the corresponding character.

String str = br.readLine();

Now convert this str to primitve type using wrapper class parse methods as required.

Java Input/Output

Object Serialization

- Serialization is the process of writing the state of an object to a byte stream. This is useful when we want to save the state of our program to a persistent storage area, such as file. It is also needed to implement Remote Method Invocation (RMI).
- Only an object that implements the Serializable interface can be saved and restored by the serialization facilities. The Serialization interface defines no members.
- Variables that are declared as transient are not saved by the serialization facilities.
- Also, static variables are not saved.

Java Input/Output

Object Serialization

Classes used for Serialization

ObjectOutputStream (OutputStream obj)

ObjectInputStream (InputStream obj)

Methods Used for Serializaiton

void writeObject(Object obj); // Used with ObjectOutputStream object

Object readObject(); // Used with ObjectInputStream

Java Input/Output

Random Access File

RandomAccessFile in order to to read an write data to a File in random positions. The RandomAccessFile class treats the file as an array of Bytes. And you can write your data in any position of the Array. To do that, it uses a pointer that holds the current position (you can think of that pointer like a cursor in a text editor…).

Constructor:

- RandomAccessFile(File file, String mode)
- RandomAccessFile(String name, String mode)

Creates a random access file stream to read from, and optionally to write to, a file with the specified name/file. Mode values: r (read), w (write), rw (read and write both)

Java Input/Output

RandomAccessFile methods :

- long getFilePointer() to get the current position of the pointer
- long length() to get the length of the file
- void seek(int)to set the position of the pointer
- int read(byte[] b) to reads up to b.length bytes of data from the file into an array of bytes
- void write(byte[] b) to write b.length bytes from the specified byte array to the file, starting at the current file pointer

Java Input/Output

File class

File class does not operate on streams.

It describes the properties of the file itself.

It deals directly with files and file system.

Java Input/Output

File class Methods

- getName()
- getPath()
- getAbsolutePath()
- getParent()
- exists()
- canWrite()
- canRead()
- isDierctory()

Java Input/Output

File class Methods

- isFile()
- length()
- lastModified()
- renameTo()
- delete()

```
String[] list = f1.list();          //If f1 is a directory, returns an array of file names

File[] list = f1.listFiles();       //If f1 is a directory, returns an array of File objects
```

Java Input/Output

Java nio package

The NIO (New I/O) system is built on two foundational items: buffers and channels.

A buffer holds data.

A channel represents an open connection to an I/O device.

In general, to use the NIO system, we obtain a channel to an I/O device and a buffer to hold data. We then operate on the buffer, inputting or outputting data as needed.

Java Input/Output

Java nio package

Buffer : Defined in the java.nio package. All bufferes are subclasses of the Buffer class.

Following are the **derived buffer classes** from Buffer:

ByteBuffer, CharBuffer, DoubleBuffer, FloatBuffer, IntBuffer, LongBuffer, ShortBuffer

MappedByteBuffer: Is a subclass of ByteBuffer that is used to map a file to a buffer.

Java Input/Output

Java nio package

- abstract byte get() : Returns the byte at the current position.
- abstract byte get(int idx) : Returns the byte at the index.
- ByteBuffer get(byte vals[]) : Copies the invoking buffer into the array.
- abstract byte put(byte b) : Copies b into the invoking buffer.
- ByteBuffer put(ByteBuffer bb): Copies the elements in bb to the invoking buffer.
- Final ByteBuffer put(byte vals[]): Copies the elements in vals to the invoking buffer.
- Final Buffer rewind( ): Sets the position of the invoking buffer to 0. Returns a reference to the buffer.

Java Input/Output

Java nio package

Channel : Defined in the java.nio.channel package. A channel represents an open connection to an I/O source or destination.

We can obtain a channel by calling getChannel ( ) method on an object that supports channel.

FileInputStream, FileOutputStream, RandomAccessFile all these supports channel and returns a channel of type **FileChannel**.

Java Input/Output

Java nio package

Channel Methods:

- abstract int read(ByteBuffer bb) : Reads bytes from the invoking channel into bb.
- abstract int write(ByteBuffer bb) : Writes bytes from the invoking channel into bb. Returns number of bytes read.

FileChannel map ( ) method

- MappedByteBuffer map (FileChannel.MapMode how, long pos, long size) throws IOException
- MapMode values: MapMode.READ_ONLY, MapMode.READ_WRITE, MapMode.PRIVATE
- PRIVATE causes a private copy of the file to be made, and changes to the buffer do not affect the underlying file.