File IO serialization

Pieter van den Hombergh
Richard van den Ham

Fontys Hogeschool voor Techniek en Logistiek

March 13, 2018

Topics

Streams in Java

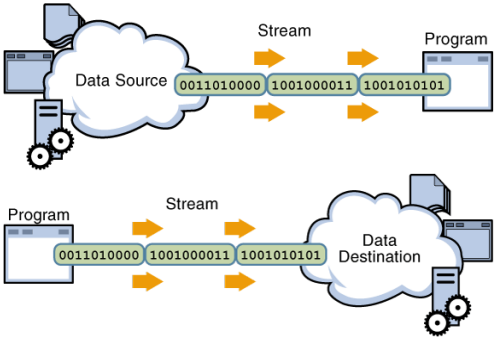


Figure: Taken from the Oracle/Sun Java tutorial

- Read or write information from different sources and types.
  - sources: network, files, devices
  - types: text, picture, sound
- Streams are FIFOs, uni-directional

Two basic stream types

- Byte Streams
  - java.io.InputStream, java.io.OutputStream
  - read and write pdf, mp3, raw...
- Character Streams (16-bit Unicode)
  - java.io.Reader, java.io.Writer
  - simplifies reading and writing characters, character-arrays or Strings

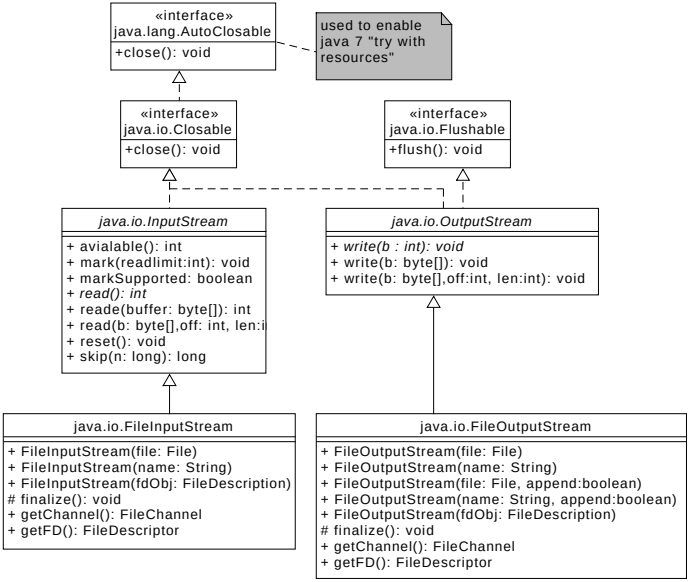
Input Stream Overview

Byte-Stream-Class for Input	Char-Stream-Class for Input
InputStream	Reader
BufferedInputStream	BufferedReader
LineNumberInputStream	LineNumberReader
ByteArrayInputStream	CharArrayReader
(none)	InputStreamReader
DataInputStream	(none)
FilterInputStream	FilterReader
PushbackInputStream	PushbackReader
PipedInputStream	PipedReader
StringBufferInputStream (deprecated under 1.1)	StringReader
SequenceInputStream	(none)

Output Stream Overview

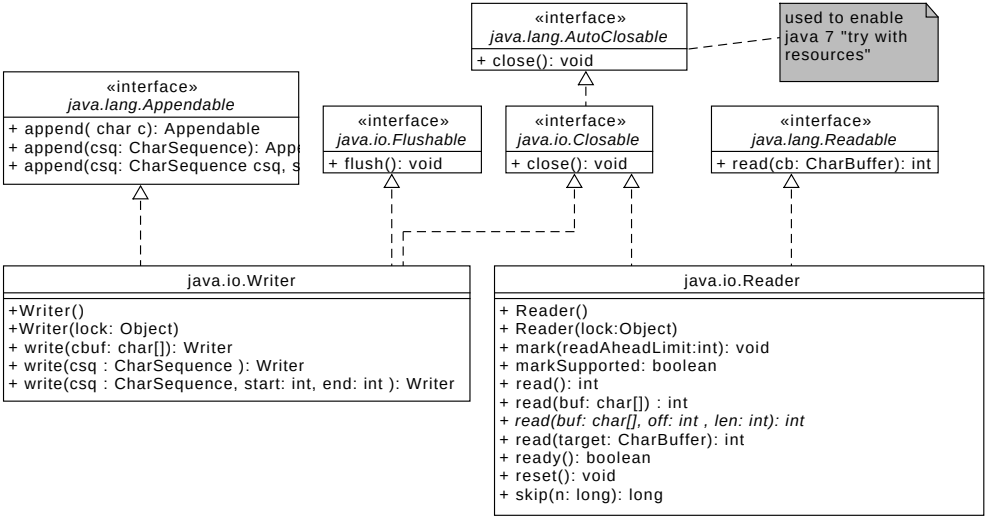
Byte-Stream-Class for Output	Char-Stream-Class for Output
OutputStream	Writer
BufferedOutputStream	BufferedWriter
ByteArrayOutputStream	CharArrayWriter
DataOutputStream	(none)
(none)	OutputStreamWriter
FileOutputStream	FileWriter
PrintStream	PrintWriter
PipedOutputStream	PipedWriter
(none)	StringWriter

ByteStreams - File Input and Output



Both Input and Output are **streams of bytes**.

CharacterStreams - Writer and Reader



- Reader reads character data (text)
- Writer writes character data

Audio Stream in Servlets

- Idea: Use Servlet Output Stream to stream MP3s via HTTP
- set ContentType to “audio/mpeg”
- read MP3 files via InputStream
- write the bytes into the Servlet Output Stream

Wrapping Streams

- Streams can be combined
- Depending on source and type
- useful additions
  - buffering
  - encryption
  - filtering
  - ...

Wrapping Streams - Buffering

- Buffered streams speed up input and output by reducing the number of reads and writes.
- They employ a buffered array of bytes or characters that act as a cache
- If no buffersize is specified, the default size is 512 bytes or characters.
- A buffered output stream calls the write method only when its buffer fills up or when the flush() method is called

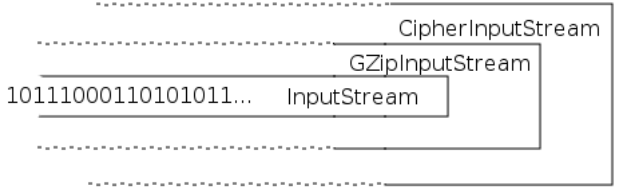
```
FileReader reader =
    new FileReader("/home/hom/test.dat");
BufferedReader bufferedIn =
    new BufferedReader(reader);
```

- Useful additional function:

```
String line = bufferedIn.readLine();
```

Wrapping Streams - On-the-fly encryption

- Java IO streams implement the decorator design pattern
- new/additional behaviour can be added to an existing stream dynamically
- might remind you of the ISO/OSI reference model for network protocols



TIP: The java.nio.file package

- Since Java 1.4 there is a new `java.nio` package, which adds extra buffering capabilities.
- Since Java7 the package `java.nio.file` has been added. The you can find some very useful classes to help you with your daily file hassling code.
- In particular `java.nio.file.Files` has many useful methods.
  - `List<String>Files.readAllLines(Path path)` (since Java8) does what its name says.
  - `Stream<String>Files.lines(Path path)` (since Java8) creates a `Stream` of String. **Streams** are a new Java 8 feature. You can do things like:

```
public static void main(String[] args) throws IOException {
  Path path = Paths.get("/etc/passwd");
  Files.lines(path).forEach(System.out::println);
}
```

Serialization

- Serialization is the process of converting an object into a sequence of bits.
- This is necessary to transport objects via streams.
- Serialization respectively Deserialization is also known as **Mashalling/Unmarshalling** (broader scope).

What makes an Object, data wise

Quiz:

If you imagine some objects you've encountered, what is the closest data structure that describes the structure of all objects?

Serialization in Java

- Not every object is serializable (e.g. `Thread`, or `Socket`).
  - Quiz: Why not? (Think transparency, security).
- In Java, the `Serializable`-Interface "marks" serializable objects
- The stream implementations `ObjectInputStream` and `ObjectOutputStream` can be used to serialize/deserialize serializable objects
  - `final void writeObject( Object obj ) throws IOException`
  - `final Object readObject() throws ClassNotFoundException, IOException`
- To exclude non-serializable attributes of serializable objects, `transient` can be used to mark these fields.
- All `transient` fields are excluded from the serialization.

Serialisation/Marshalling

```
20    private static final String OBJECT_STORE
21            = "/tmp/objects.ser";
22
23    private void marshal() {
24      try ( ObjectOutputStream out
25              = new ObjectOutputStream(
26                      new FileOutputStream( OBJECT_STORE ) ); ) {
27                UserBean user1 = new UserBean( "Uwe",
28                                               "UwePass" );
29                UserBean user2 = new UserBean( "Peter",
30                                               "PeterPass" );
31                UserBean user3 = new UserBean( "Paula",
32                                               "PaulaPass" );
33                user3.setSocket( new Socket() );
34                System.out.println( "BeanCounter =  "
35                        + UserBean.getBeanCounter() );
36
37                out.writeObject( user1 );
38                out.writeObject( user2 );
39                out.writeObject( user3 );
40              } catch ( IOException ex ) {
41                logger.log( Level.SEVERE, null, ex );
42              }
43    }
```

DeSerialisation/UnMarshalling

```
45    private void unmarshal() {
46      try ( ObjectInputStream in
47              = new ObjectInputStream(
48                      new FileInputStream( OBJECT_STORE ) ); ) {
49
50                for ( int i = 0; i < 3; i++ ) {
51                  UserBean user = ( UserBean ) in.readObject();
52                  System.out.println( "user = " + user );
53                }
54
55                //UserBean newUser = new UserBean("Richard", "Richardpw");
56                //System.out.println("BeanCounter =  "+ newUser.getBeanCounter↩
                      ());
57              } catch ( ClassNotFoundException | IOException ex ) {
58                logger.log( Level.SEVERE, null, ex );
59              }
60    }
```

Serialisation formats

- Binary: from Java to Java, (between JVMS even across the network)
- Text based (modern: always UTF-8). Can be produced and consumed by different technologies (e.g. Java at server, Javascript in browser), like PHP or C# at one end, Java at the other
  - XML often via the SOAP protocol, typically **HTTP** based.
  - JSON, also typically **HTTP** based.
- Proto Buffers. Binary. PROTBUF is a modern (de)serialization framework that has a schema.

NEVER DESERIALIZE A JAVA OBJECT STREAM FROM AN UNKNOWN SOURCE.

Deserialization without a schema to validate the input is dangerous at best.
Efective Java 3rd ed item 85.
It is very easy to create a denial of service attack this way.

Javascript Object Notation

- Javascript Object Notation (JSON) is very popular at the moment.
  - It is less verbose and **heavy** on the wire.
  - It is easily generated and parsed (read) by programs and humans(although only for developers during development).
  - Nowadays the format of choice for communication between apps and backend.
  - Can be stored, but also generated from traditional tables in PostgreSQL.
  - Has APIs that help generating (which sounds trivial) and parsing (less so) the wire format. e.g. `javax.json` defines some interfaces.
  - This will be the transport format in PRJ2 this year (2017).

Person (de)serialization demo with GSON

Look ma, No hands

```
public static void main( String[] args ) {
    Gson gson= new Gson();
    Person max = new Person( "Max", LocalDate.of( 1960, 04, 30 ) )↩
        ;
    String jo = gson.toJson( max );
    System.out.println( "jo = " + jo );
    Person otherMax = gson.fromJson( jo, Person.class );
    System.out.println( "fromJson = " + otherMax );
    assert max.equals( otherMax );
    System.out.println( "all  done" );
}
```

Questions and links

Not all understood?
Study
http://docs.oracle.com/javase/tutorial/essential/io/index.html Basic I/O Part I/O Streams.

Questions?

Questions or remarks?