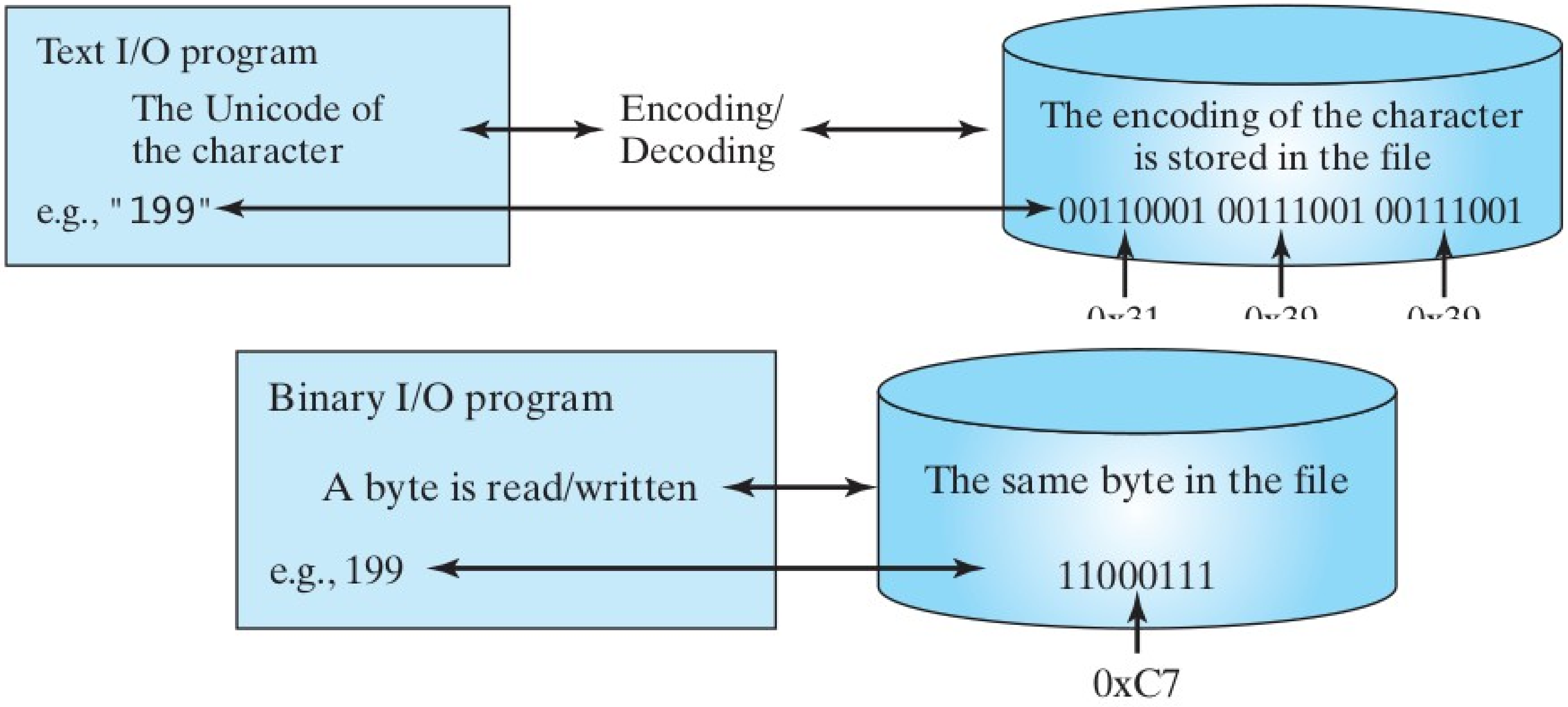
Chapter 17
Binary I/O

Objectives (for video lecture)

- To discover how **I/O** is processed in Java (§17.2).
- To distinguish between **text I/O** and **binary I/O** (§17.3).
- To read and write bytes using **FileInputStream** and **FileOutputStream** (§17.4.1).
- To filter data using the base classes **FilterInputStream** and **FilterOutputStream** (§17.4.2).
- To read and write primitive values and strings using **DataInputStream** and **DataOutputStream** (§17.4.3).
- To store and restore objects using **ObjectOutputStream** and **ObjectInputStream** (§17.6).
- To implement the **Serializable** interface to make objects serializable (§17.6.1).

Text I/O vs Binary I/O

Binary I/O does not involve encoding or decoding and thus is more efficient than text I/O.



Binary I/O

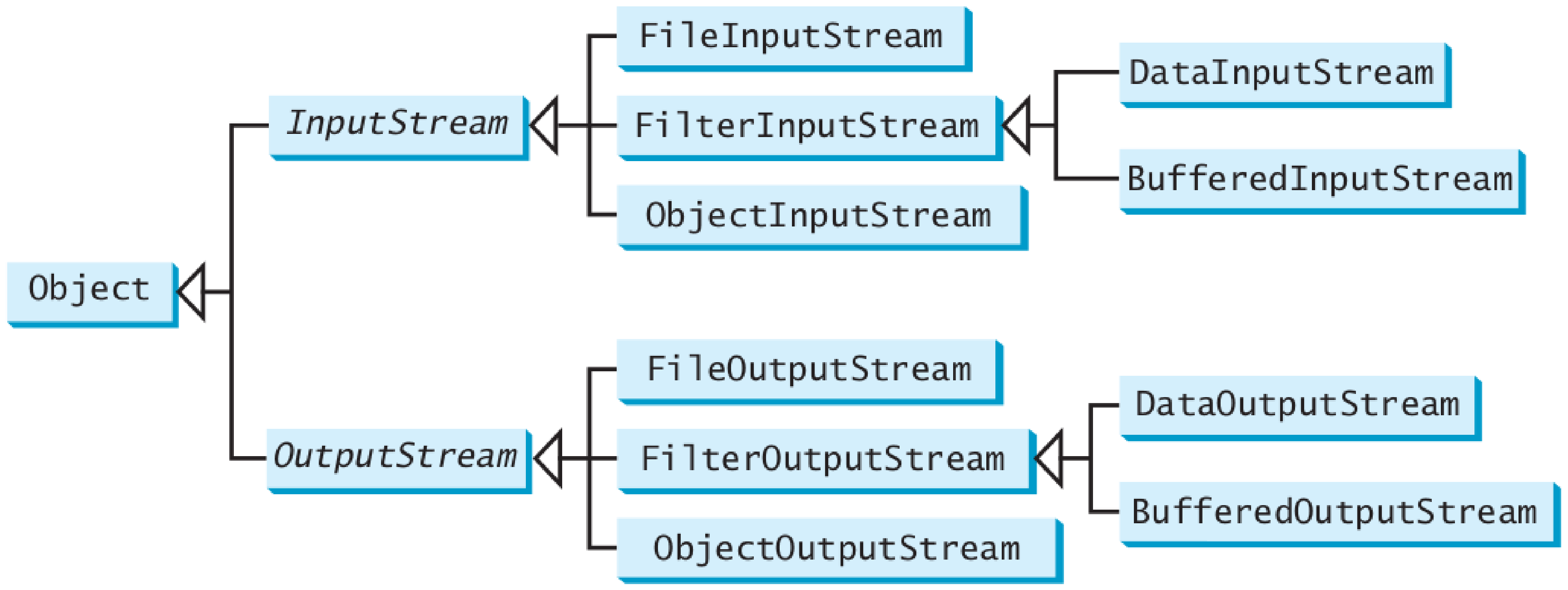


FIGURE 17.3 InputStream, OutputStream, and their subclasses are for performing binary I/O.

File I/O Stream vs Filter I/O Stream vs Data I/O Stream vs Object I/O Stream

FileInputStream / FileOutputStream	$\longrightarrow$	Bytes & characters
FilterInputStream / FilterOutputStream	$\longrightarrow$	Integers, doubles & strings
DataInputStream / DataOutputStream	$\longrightarrow$	Primitive numeric types
ObjectInputStream / ObjectOutputStream	$\longrightarrow$	Java class objects Custom class objects (with serializable)

Serializable interface

Serialization is the conversion of the state of an object into a byte stream; deserialization does the opposite. Stated differently, serialization is the conversion of a Java object into a static stream (sequence) of bytes which can then be saved to a database or transferred over a network.