CS112

Binary I/O
Chapter 17
Lecture 12

الفصل الدراسي الثاني 1443- Spring 2022

College of Computer Science and Engineering

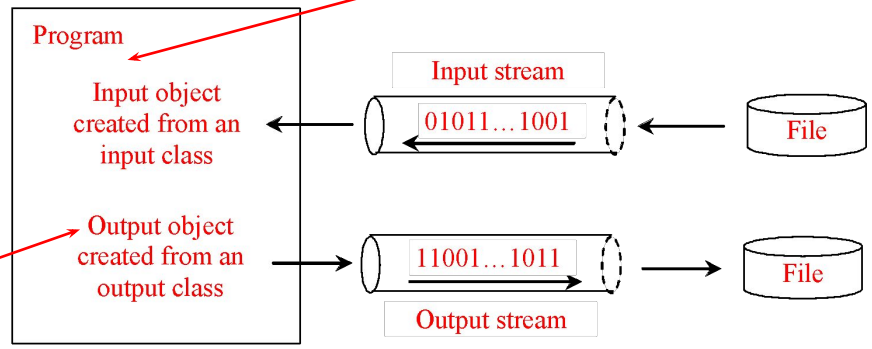
Text File vs. Binary File

- Data stored in a text file is represented in human-readable form.
- Data stored in a **binary file** is represented in binary form.
- You cannot read binary files They are designed to be read by programs.
  - For example, Java source programs are stored in text files and can be read by a text editor, but Java classes are stored in binary files and are read by the JVM.
- The advantage of binary files is that they are more efficient to process than text files.
- Although it is not technically precise and correct, you can imagine that a text file consists of a sequence of characters and a binary file consists of a sequence of bits.
  - For example, the decimal integer 199 is stored as the sequence of three characters: '1', '9', '9' in a text file and the same integer is stored as a byte-type value C7 in a binary file, because decimal 199 equals to hex C7.

How is I/O Handled in Java?

- A File object encapsulates the properties of a file or a path, but **does not contain the methods for reading/writing data from/to a file**.
- In order to perform I/O, you need to create objects using appropriate Java I/O classes.

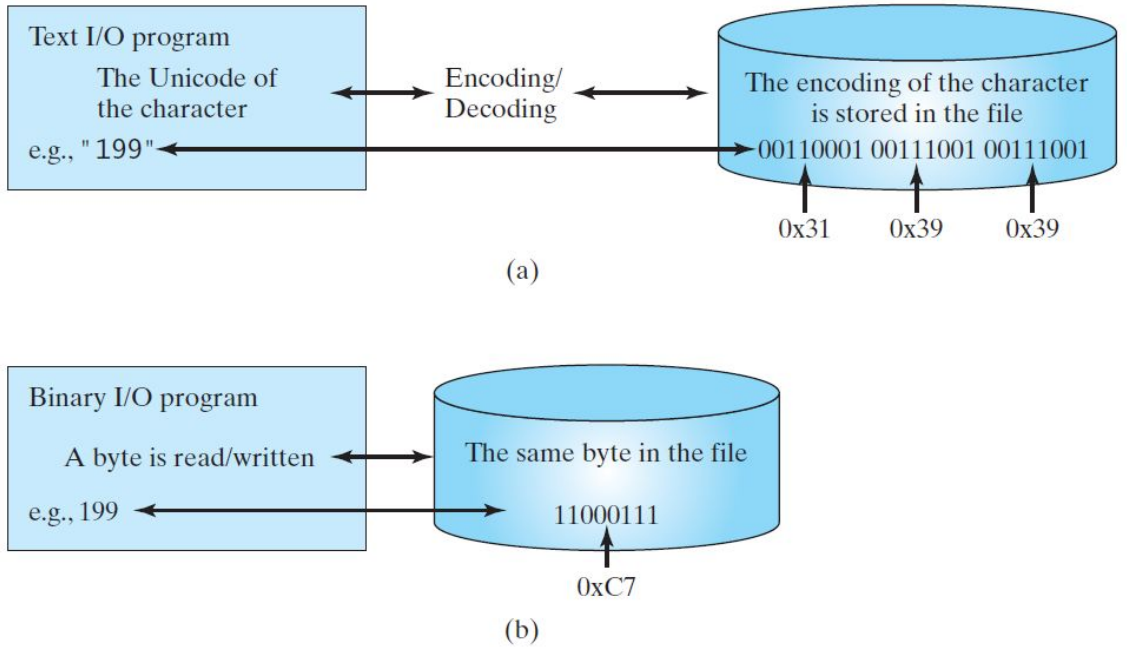
```
Scanner input = new Scanner(new File("temp.txt"));
System.out.println(input.nextLine());
```




```
PrintWriter output = new PrintWriter("temp.txt");
output.println("Java 101");
output.close();
```

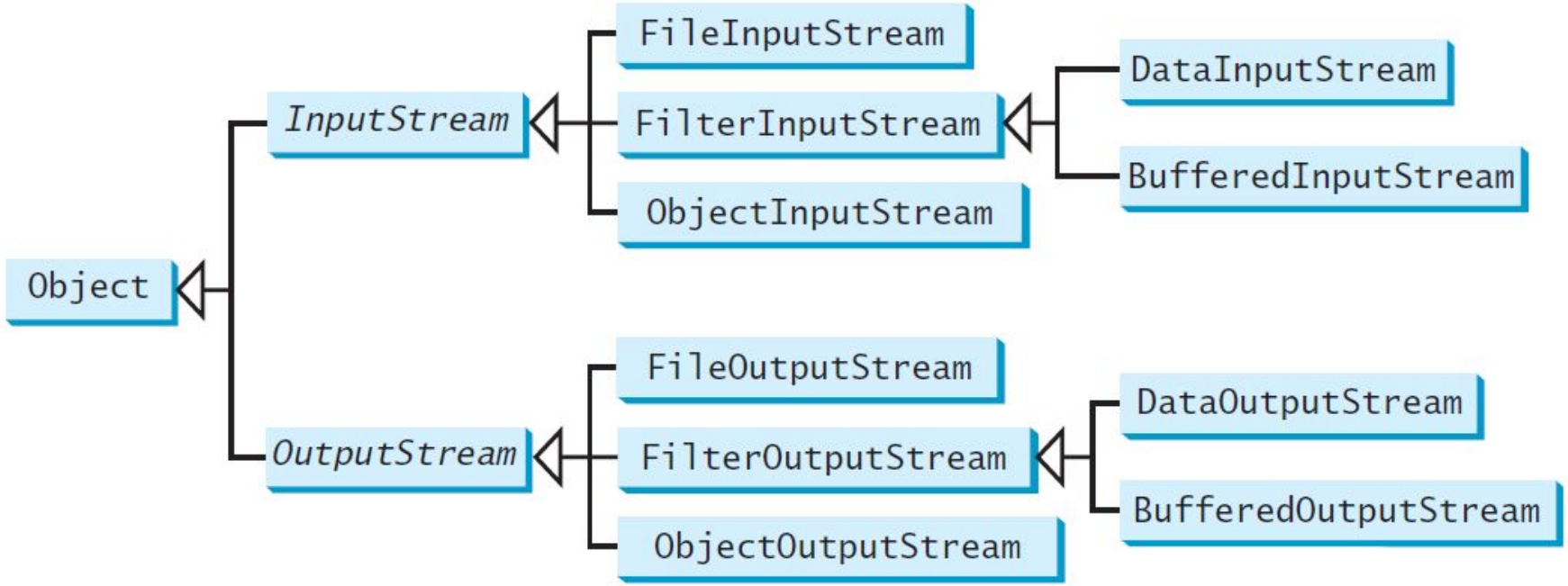
Binary I/O

- Text I/O requires encoding and decoding.
- The JVM converts a Unicode to a file specific encoding when writing a character and coverts a file specific encoding to a Unicode when reading a character.
- Binary I/O does not require conversions. When you write a byte to a file, the original byte is copied into the file. When you read a byte from a file, the exact byte in the file is returned.



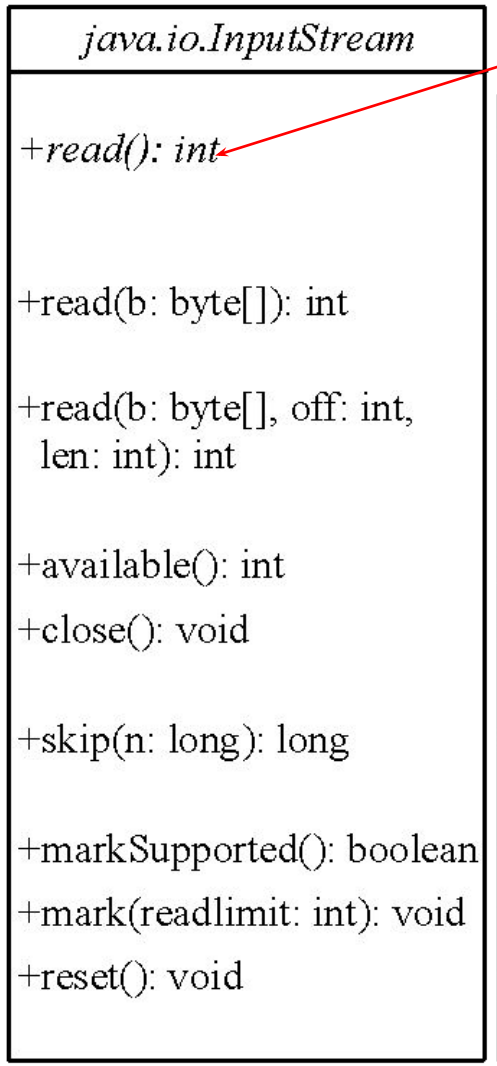
Binary I/O Classes

- The design of the Java I/O classes is a good example of applying inheritance, where common operations are generalized in superclasses, and subclasses provide specialized operations.
- **InputStream** is the root for binary input classes.
- **OutputStream** is the root for binary output classes.



InputStream

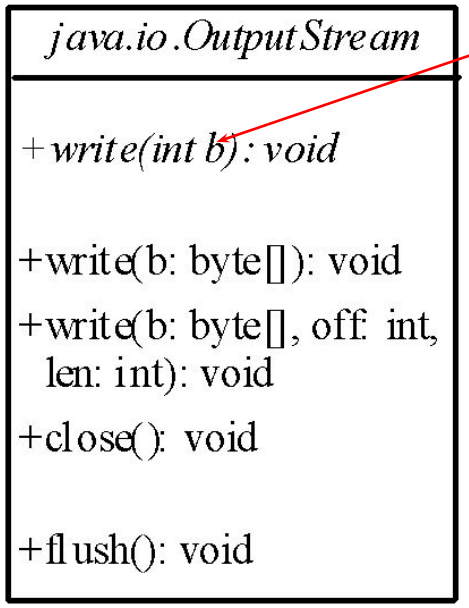
The value returned is a byte as an int type.



+read(): int	Reads the next byte of data from the input stream. The value byte is returned as an int value in the range 0 to 255. If no byte is available because the end of the stream has been reached, the value –1 is returned.
+read(b: byte[]): int	Reads up to b.length bytes into array b from the input stream and returns the actual number of bytes read. Returns -1 at the end of the stream.
+read(b: byte[], off: int, len: int): int	Reads bytes from the input stream and stores into b[off], b[off+1], …, b[off+len-1]. The actual number of bytes read is returned. Returns -1 at the end of the stream.
+available(): int	Returns the number of bytes that can be read from the input stream.
+close(): void	Closes this input stream and releases any system resources associated with the stream.
+skip(n: long): long	Skips over and discards n bytes of data from this input stream. The actual number of bytes skipped is returned.
+markSupported(): boolean	Tests if this input stream supports the mark and reset methods.
+mark(readlimit: int): void	Marks the current position in this input stream.
+reset(): void	Repositions this stream to the position at the time the mark method was last called on this input stream.

OutputStream

The value is a byte as an int type.



Writes the specified byte to this output stream. The parameter b is an int value. (byte)b is written to the output stream.

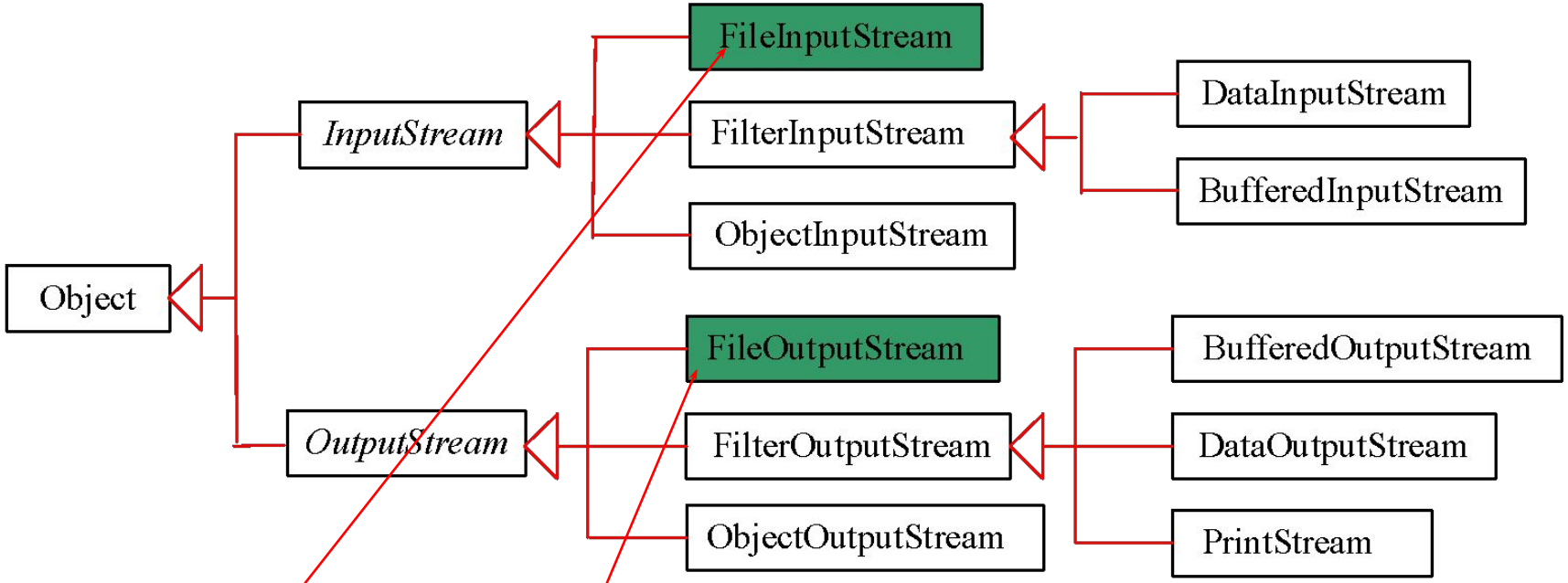
Writes all the bytes in array b to the output stream.

Writes b[off], b[off+1], …, b[off+len-1] into the output stream.

Closes this output stream and releases any system resources associated with the stream.

Flushes this output stream and forces any buffered output bytes to be written out.

FileInputStream/FileOutputStream



FileInputStream/FileOutputStream associates a binary input/output stream with an external file. All the methods in FileInputStream/FileOuptputStream are inherited from its superclasses.

FileInputStream

- To construct a FileInputStream, use the following constructors:

  public FileInputStream(String filename)

  public FileInputStream(File file)

A java.io.FileNotFoundException would occur if you attempt to create a FileInputStream with a nonexistent file.

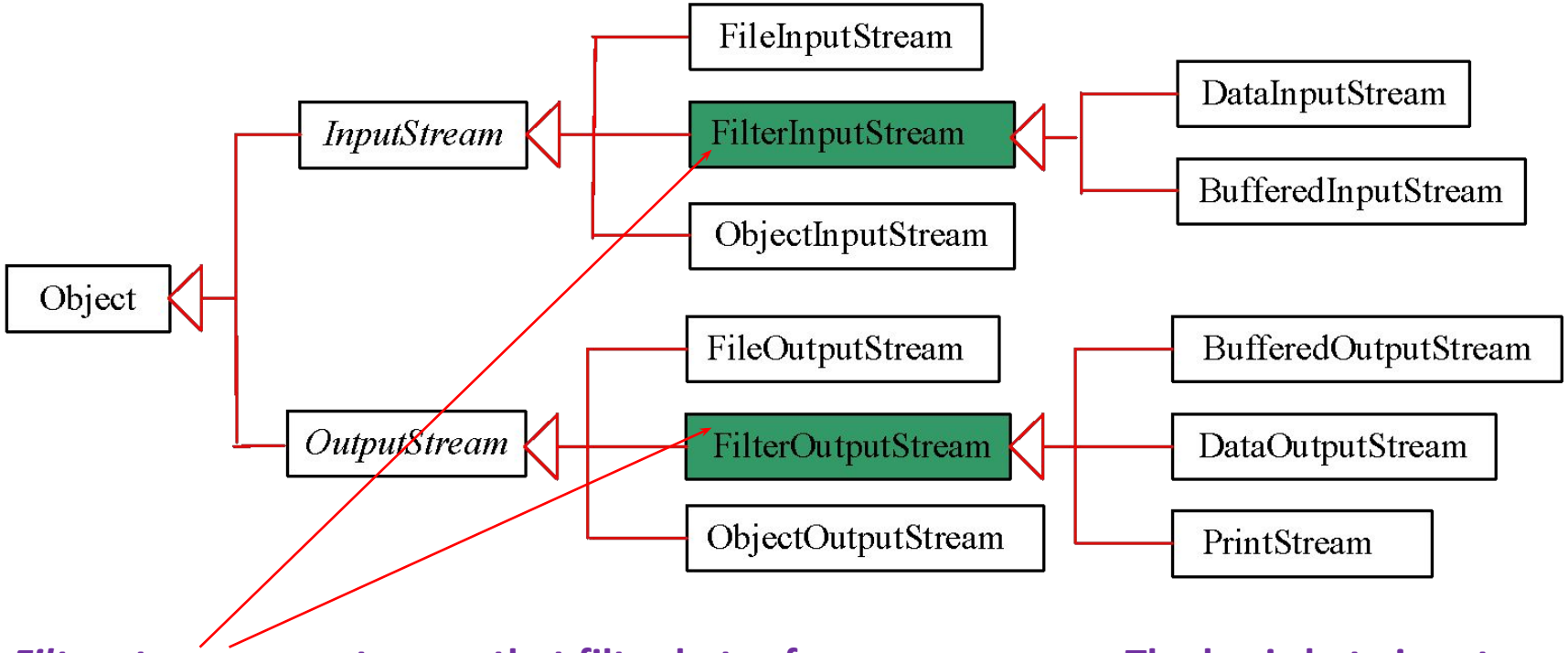
FileOutputStream

- To construct a FileOutputStream, use the following constructors:

```
public FileOutputStream(String filename)
public FileOutputStream(File file)
public FileOutputStream(String filename, boolean append)
public FileOutputStream(File file, boolean append)
```

- If the file does not exist, a new file would be created.
- If the file already exists, the first two constructors would delete the current contents in the file.
- To retain the current content and append new data into the file, use the last two constructors by passing true to the append parameter.
- See TestFileStream.java

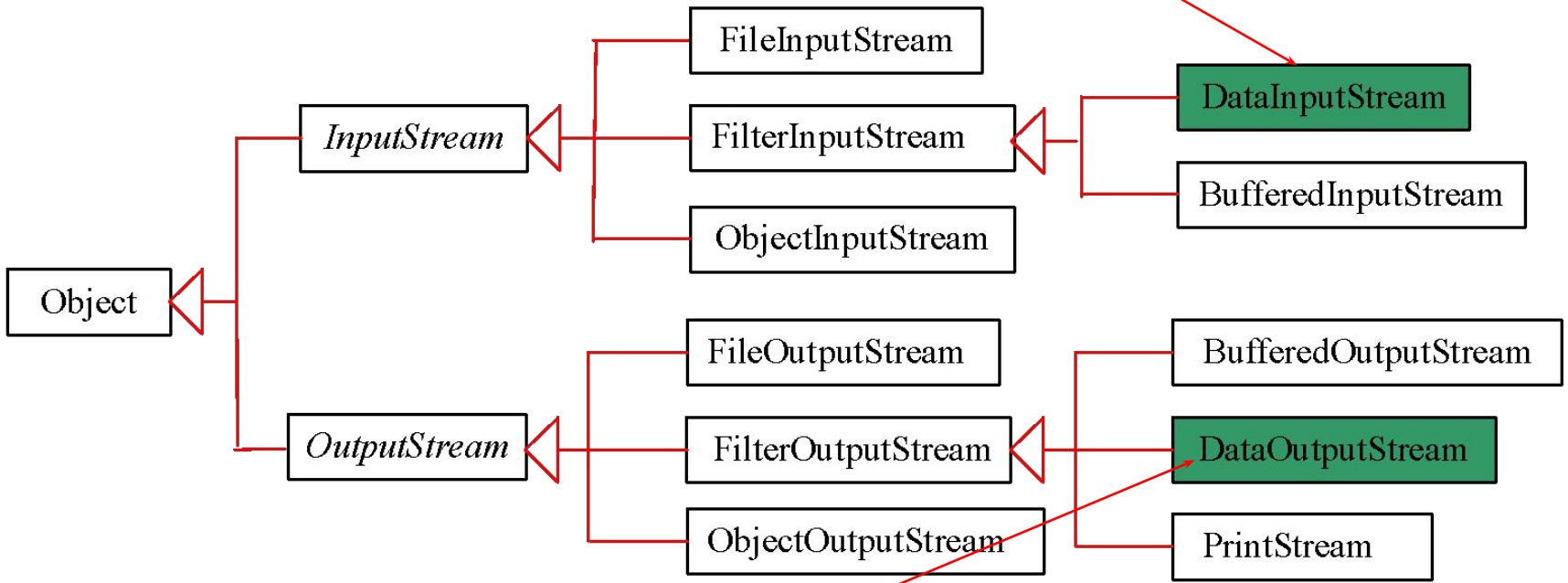
FilterInputStream/FilterOutputStream



Filter streams **are streams that filter bytes for some purpose. The basic byte input stream provides a read method that can only be used for reading bytes. If you want to read integers, doubles, or strings, you need a filter class to wrap the byte input stream. Using a filter class enables you to read integers, doubles, and strings instead of bytes and characters. FilterInputStream and FilterOutputStream are the base classes for filtering data. When you need to process primitive numeric types, use DatInputStream and DataOutputStream to filter bytes.**

DataInputStream/DataOutputStream

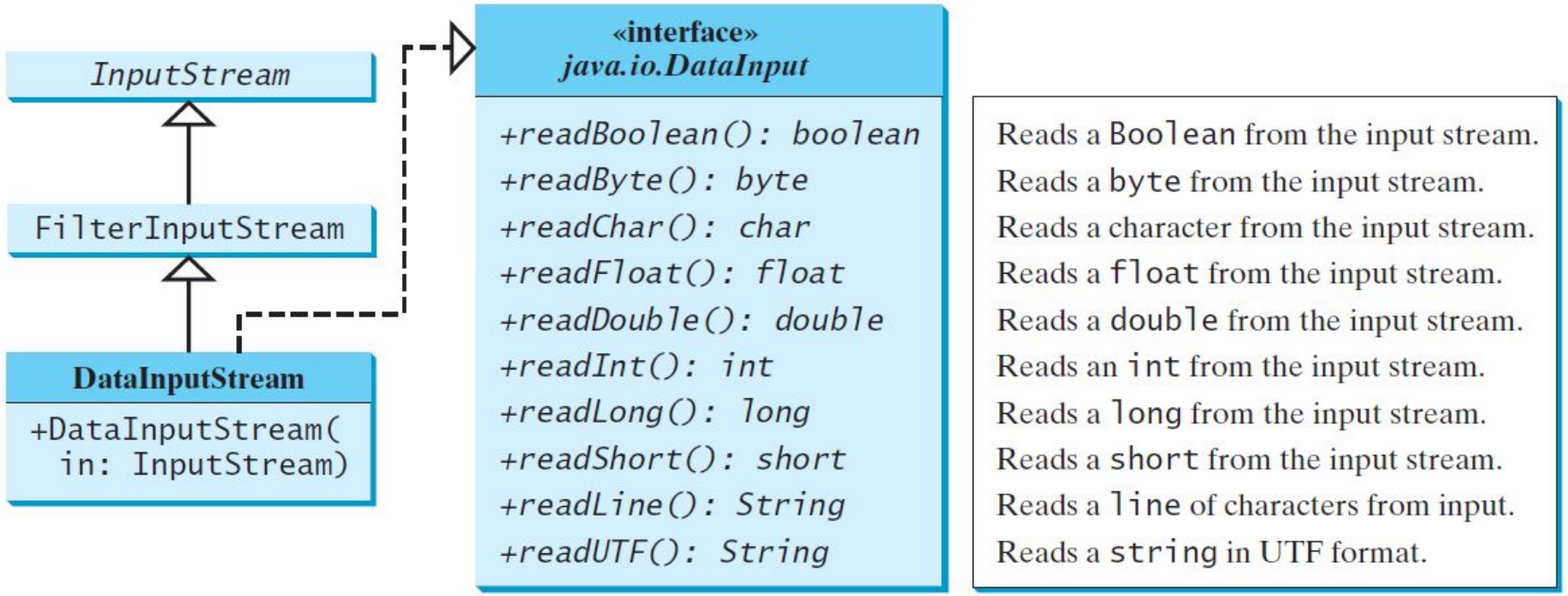
DataInputStream reads bytes from the stream and converts them into appropriate primitive type values or strings.



DataOutputStream converts primitive type values or strings into bytes and output the bytes to the stream.

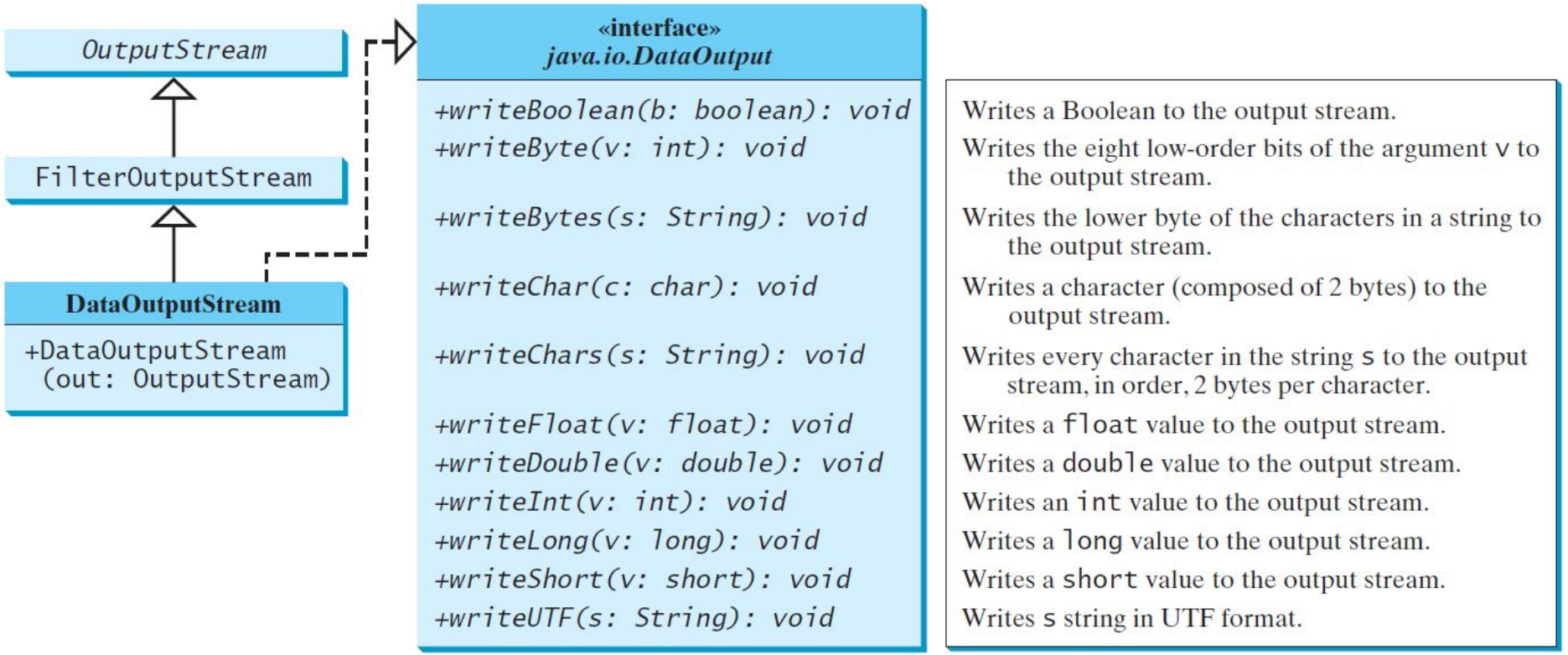
DataInputStream

- DataInputStream extends FilterInputStream and implements the DataInput interface.



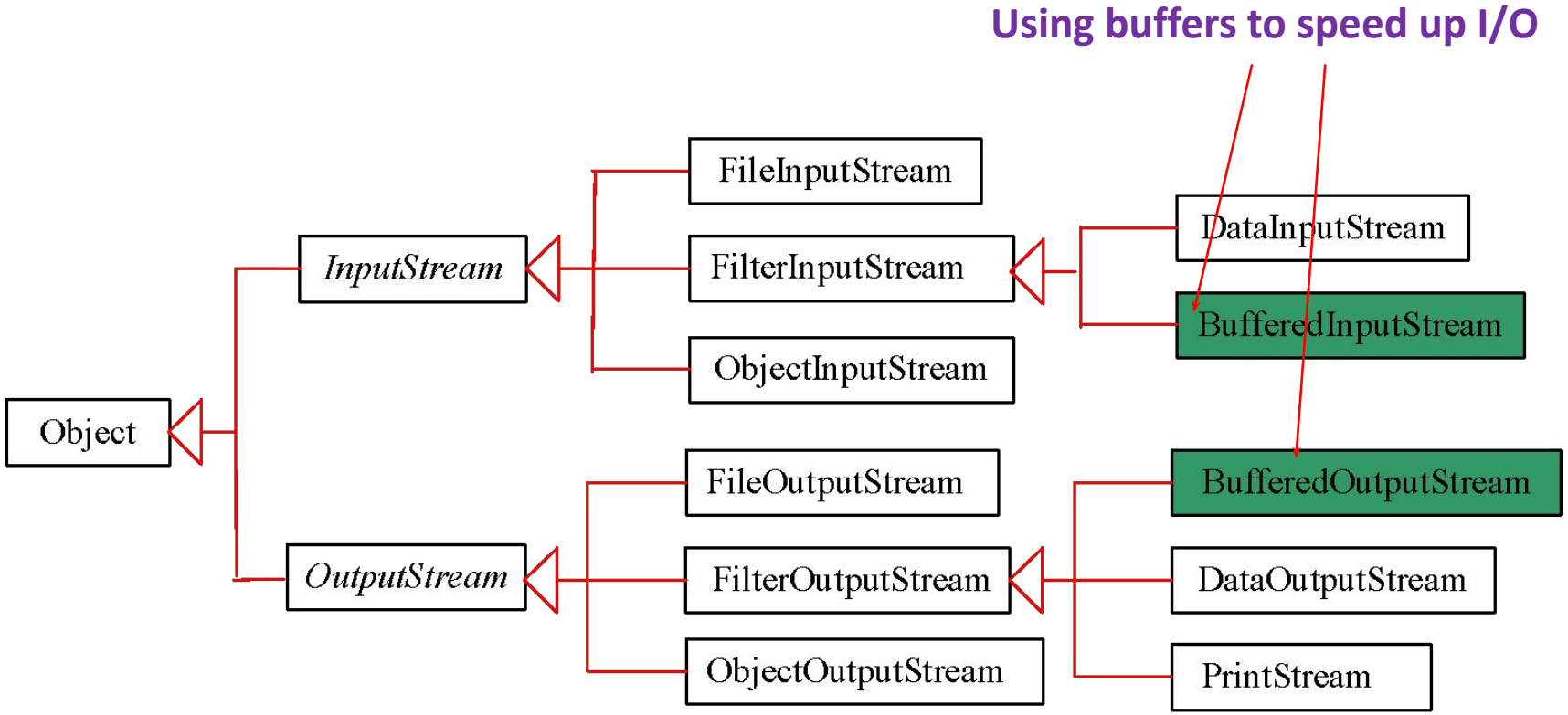
DataOutputStream

- DataOutputStream extends FilterOutputStream and implements the DataOutput interface.



See TestDataStream.java Page 686 in textbook

BufferedInputStream/BufferedOutputStream



BufferedInputStream/BufferedOutputStream does not contain new methods. All the methods BufferedInputStream/BufferedOutputStream are inherited from the InputStream/OutputStream classes.

Recommended Readings

- Object I/O: Section 17.6