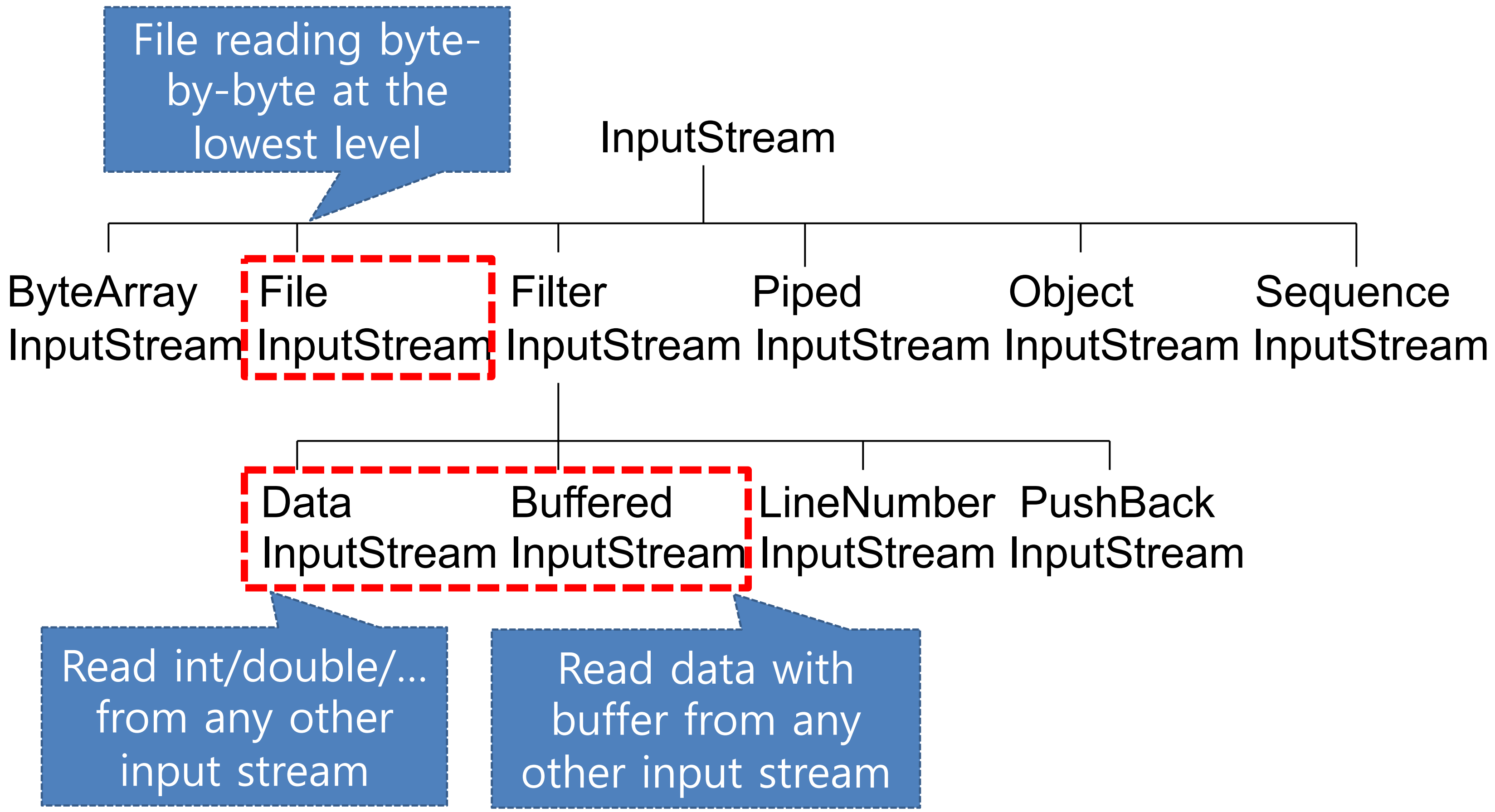
File I/O in JAVA

Younghoon Kim
(nongaussian@hanyang.ac.kr)

Reading Bytes

- Abstract classes provide basic common operations which are used as the foundation for more concrete classes, e.g., InputStream has
  - int **read**( ) - reads a byte and returns it or –1 (end of input)
  - int **available**( ) – num of bytes still to read
  - void close()
- Concrete classes override this method,
  - E.g., *FileInputStream* reads one byte from a file, *System.in* is a subclass of InputStream that allows you to read from the keyboard

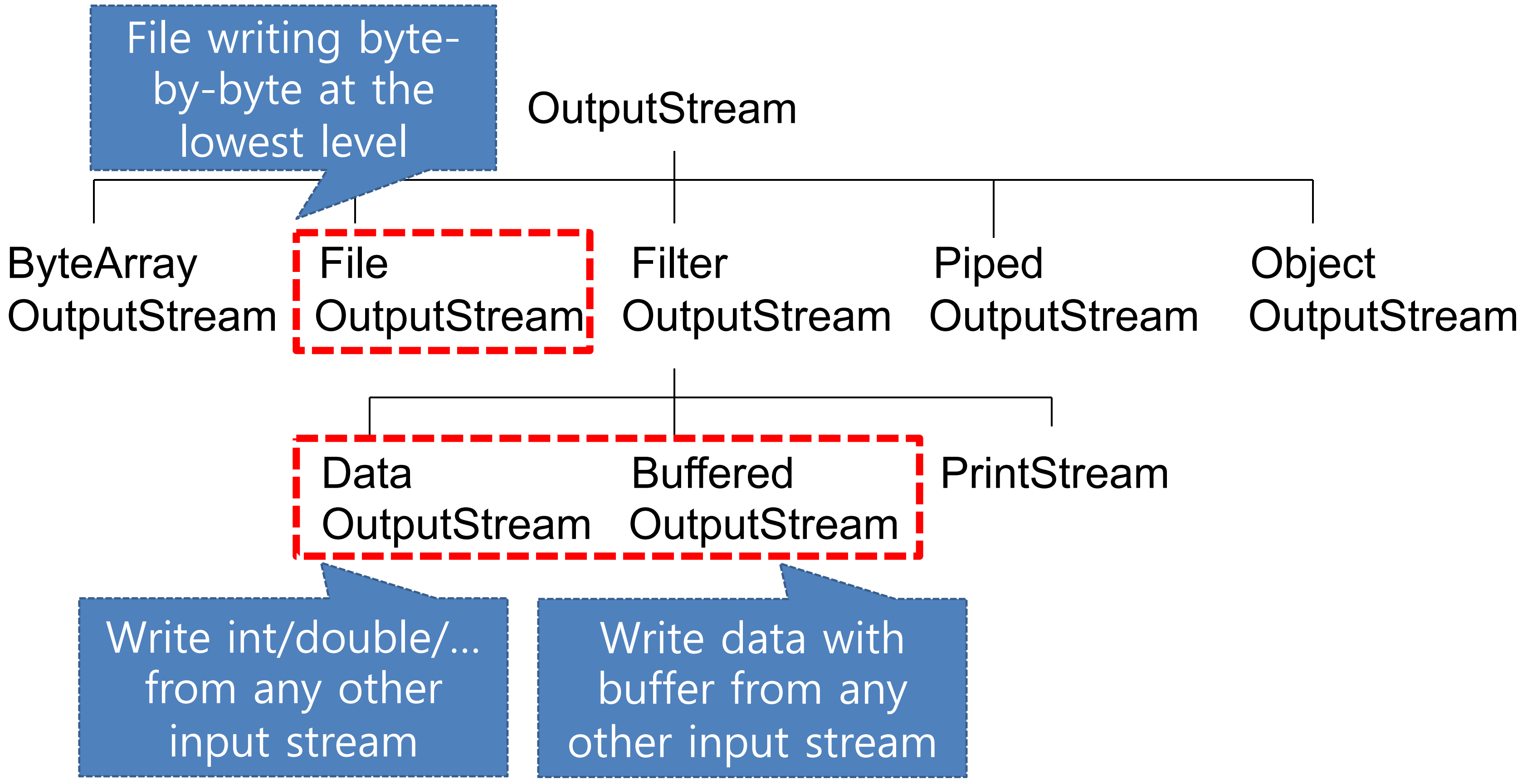
InputStream Hierarchy



Writing Bytes

- OutputStream has
  - void **write**(int b) - writes a single byte to an output location
- Java IO programs involve using concrete versions of these because most data contain numbers, strings and objects rather than individual bytes

OutputStream Hierachy



File Processing

- Typical pattern for file processing is:
  - OPEN A FILE
  - CHECK FILE OPENED SUCCESSFULLY
  - READ/WRITE FROM/TO FILE
  - CLOSE FILE
- Input and Output streams have close method (output may also use flush)

FileInputStream / FileOutputStream

- Handle I/O of raw binary data
  - Using byte streams to perform input and output of 8-bit bytes
  - Unbuffered I/O (each read and write request is handled directly by the underlying OS -> high cost)

```
public static void copyFileStream() throws IOException {

    FileInputStream is =new FileInputStream("./txt/alice.txt");
    FileOutputStream os = new FileOutputStream("./txt/alice-copy.txt");

    File file = new File("./txt/alice.txt");
    int len = (int)file.length();

    try {
        for (int i = 0; i < len; i++) {
            os.write(is.read());
            os.flush();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

File I/O Streams Constructors

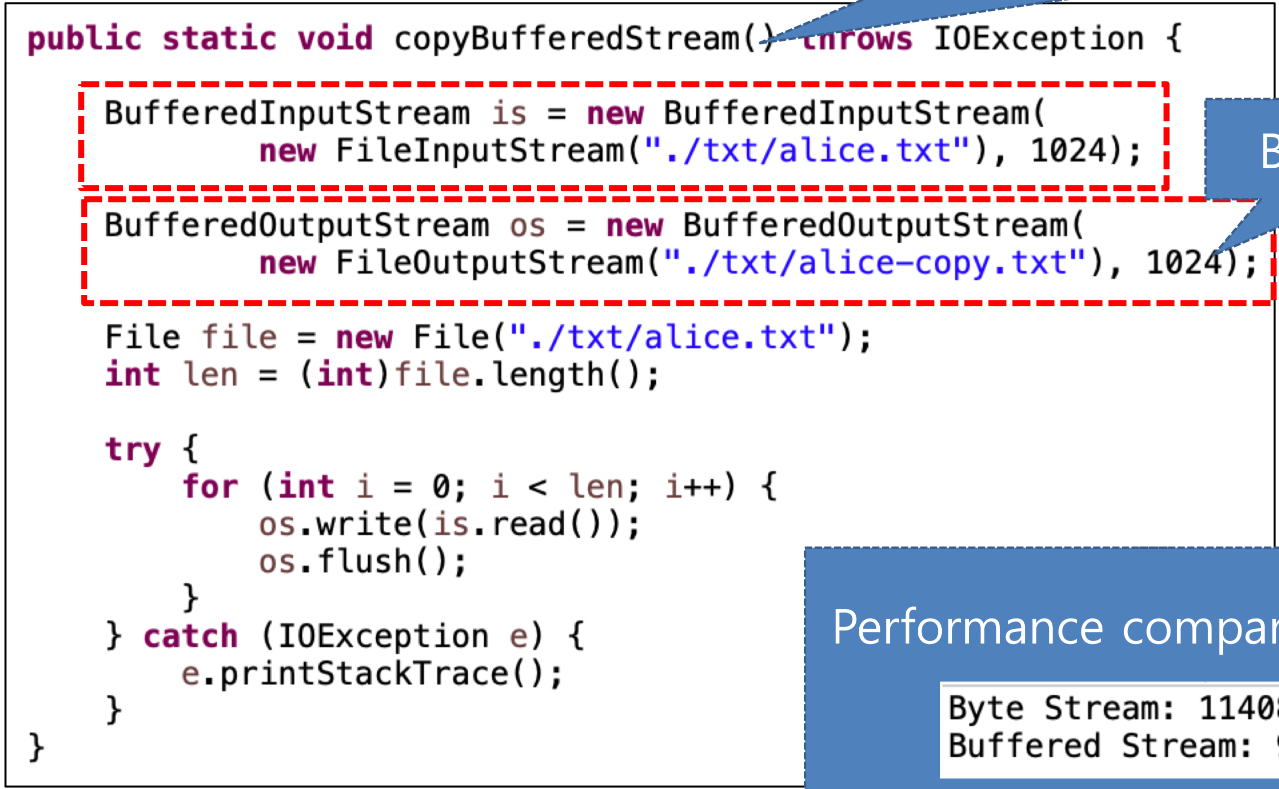
- FileInputStream(String name)
- FileOutputStream(String name)
- BufferedInputStream(InputStream in)
- BufferedOutputStream(OutputStream out)

Buffered I/O Streams

- Buffered input streams read data from a memory area known as a buffer.

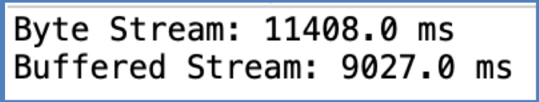
wrapping an unbuffered stream with a buffered stream

Buffer size



```java
public static void copyBufferedStream() throws IOException {

    BufferedInputStream is = new BufferedInputStream(
            new FileInputStream("./txt/alice.txt"), 1024);

    BufferedOutputStream os = new BufferedOutputStream(
            new FileOutputStream("./txt/alice-copy.txt"), 1024);

    File file = new File("./txt/alice.txt");
    int len = (int)file.length();

    try {
        for (int i = 0; i < len; i++) {
            os.write(is.read());
            os.flush();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Performance comparison (on SSD)



```
Byte Stream: 11408.0 ms
Buffered Stream: 9027.0 ms
```

Data I/O Streams

e.g., byte, int, short

- A data input stream lets an application read primitive Java data types from an underlying input stream.

```
public static void copyDataStream() throws IOException {

    DataInputStream is = new DataInputStream(
            new BufferedInputStream(
                    new FileInputStream("./txt/alice.txt"), 1024)
            );

    DataOutputStream os = new DataOutputStream(
            new BufferedOutputStream(
                    new FileOutputStream("./txt/alice-copy.txt"), 1024)
            );

    File file = new File("./txt/alice.txt");
    int len = (int)file.length();

    try {
        for (int i = 0; i < len; i++) {
            os.write(is.readByte());
            os.flush();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

This class is useful to handle given posting data which consists of three integers

Reads primitive data type