File I/O in JAVA

Younghoon Kim
(nongaussian@hanyang.ac.kr)

File I/O In Java

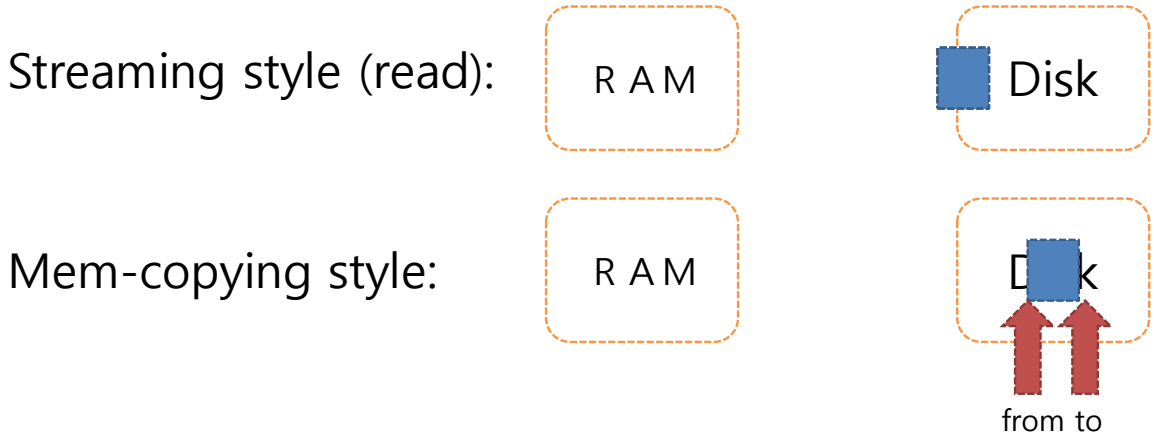
- Sample code

```
import java.io.BufferedInputStream;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.IOException;

public class TestInputStream {
  public static void main(String[] args) throws IOException {
    DataInputStream dis = new DataInputStream(
      new BufferedInputStream(
        new FileInputStream("temp.tmp")));
    int tmp;
    for (int i = 0; i < 10; i++)
      tmp = dis.readInt();
    dis.close();
  }
}
```

Reading Bytes

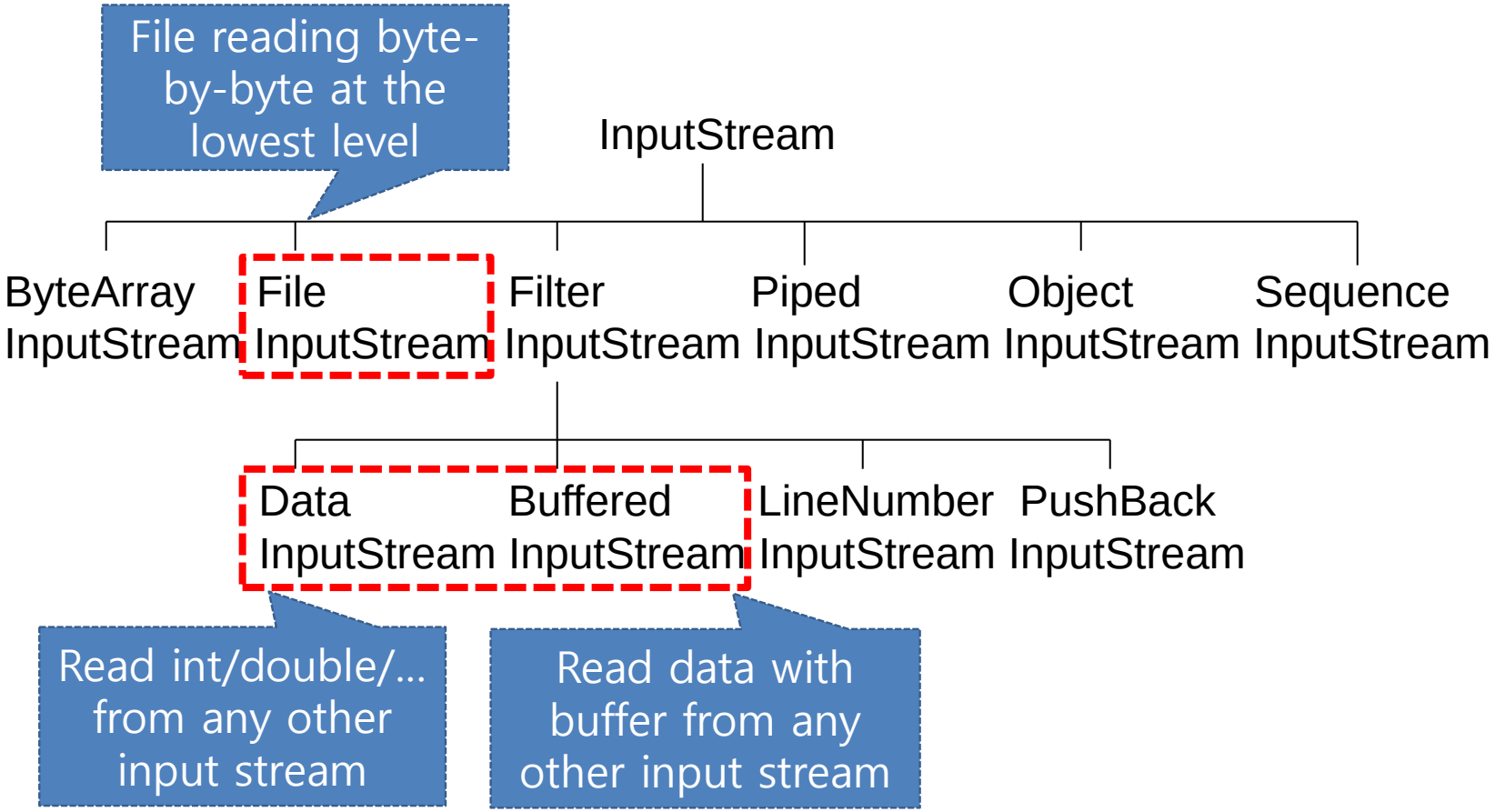
- Abstract classes provide basic common operations which are used as the foundation for more concrete classes, e.g., InputStream has
  - int **read**( ) - reads a byte and returns it or –1 (end of input)
  - int **available**( ) – # of bytes still to read
  - void **close**()



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**( ) – # 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**, *System.out* to print on the **screen**

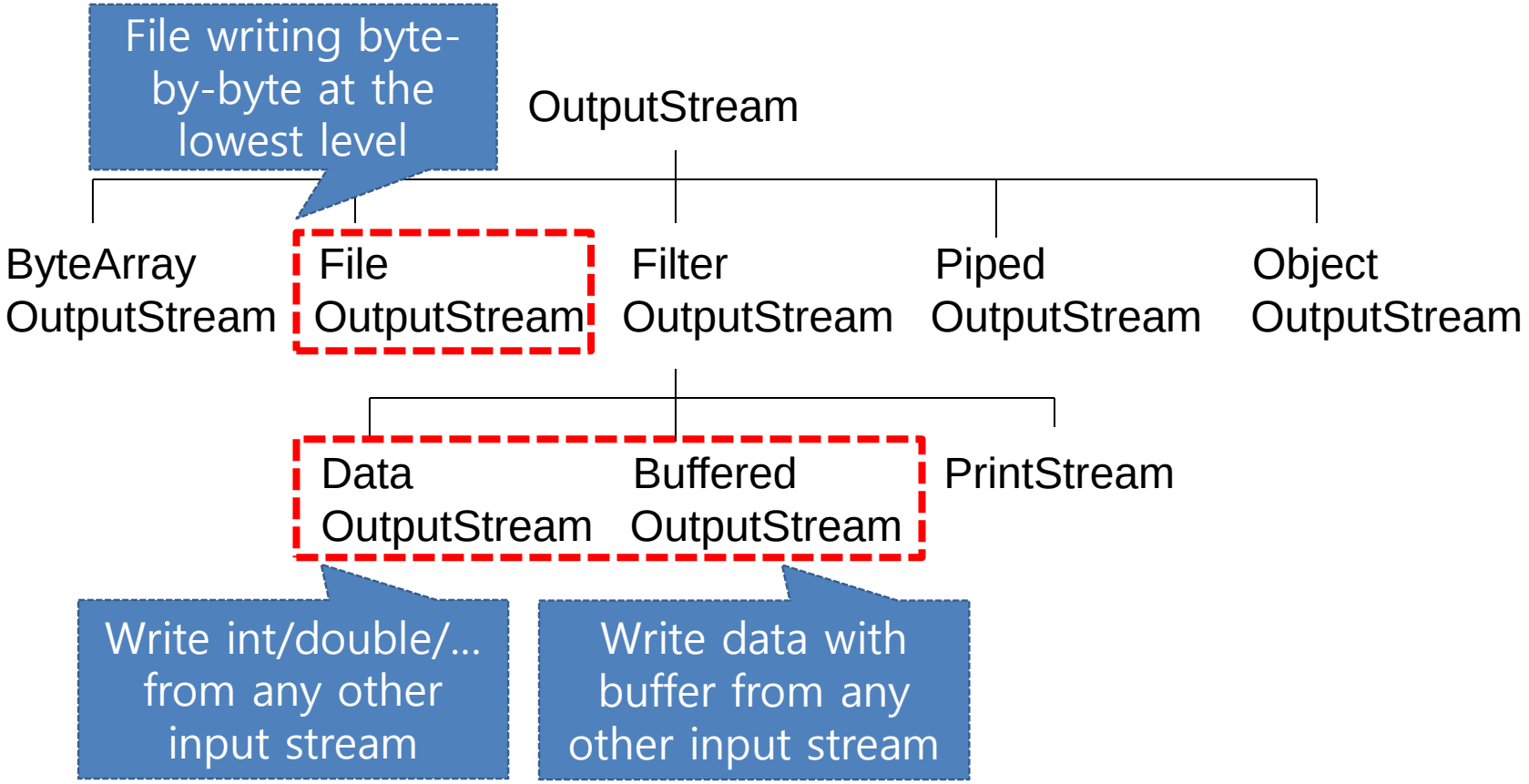
InputStream Hierarchy



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 class TestFileStream {
  public static void main(String[] args) throws IOException {
    FileInputStream is = new FileInputStream("alice.txt");
    FileOutputStream os = new FileOutputStream("alice-output.txt");

    int data = -1;
    while ( (data = is.read()) != -1 ) {
      os.write(data);
    }

    is.close();
    os.close();
  }
}
```

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

```
public class TestBufferedStream {
  public static void main(String[] args) throws IOException {
    BufferedInputStream is = new BufferedInputStream(
      new FileInputStream("alice.txt"), 16*1024);
    BufferedOutputStream os = new BufferedOutputStream(
      new FileOutputStream("alice-output.txt"), 16*1024);

    int data = -1;
    while ( (data = is.read()) != -1 ) {
      os.write(data);
    }

    is.close();
    os.close();
  }
}
```

Buffer size

Data I/O Streams

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

```
public class TestDataStream {
  public static void main(String[] args) throws IOException {
    DataInputStream is = new DataInputStream(
        new BufferedInputStream(
        new FileInputStream("alice.txt"), 16*1024));
    DataOutputStream os = new DataOutputStream(
        new BufferedOutputStream(
        new FileOutputStream("alice-output.txt"), 16*1024));

    try {
      int data = -1;
      while (true) {
        data = is.readByte();
        os.writeByte(data);
      }
    }
    catch (EOFException e) {
    }
```

Reads primitive data type, e.g., byte, int, float

```java
public class TestDataStream {
  public static void main(String[] args) throws IOException {
    DataInputStream is = new DataInputStream(
        new BufferedInputStream(
        new FileInputStream("alice.txt"), 16*1024));
    DataOutputStream os = new DataOutputStream(
        new BufferedOutputStream(
        new FileOutputStream("alice-output.txt"), 16*1024));

    try {
      int data = -1;
      while (true) {
        data = is.readByte();
        os.writeByte(data);
      }
    }
    catch (EOFException e) {
    }

    is.close();
    os.close();
  }
}
```

MODULE 3. EXTERNAL MERGE SORT

FAQ. BLOCKSIZE와 NBLOCKS의 역할

Testing Your Submission

```
public class HanyangSEExternalSort implements ExternalSort {
        public void sort(String infile,  /* input file path */
                         String outfile, /* output file path */
                         String tmpdir,  /* temporary directory
                                            for creating intermediate
                                            runs */
                         int blocksize,  /* 8192 or 16384 bytes */
                         int nblocks) throws IOException {
                                 /* available memory size /  blocksize */
        …
        }
}
```

Called

edu.hanyang.test.ExternalSortEval:

```
sort = new HanyangSEExternalSort();
sort.sort(INPUT_DATA_PATH, OUTPUT_DATA_PATH,
TEMP_DIR_PATH, blocksize, nblocks);
```

Bash Shell Script for Testing Module 2

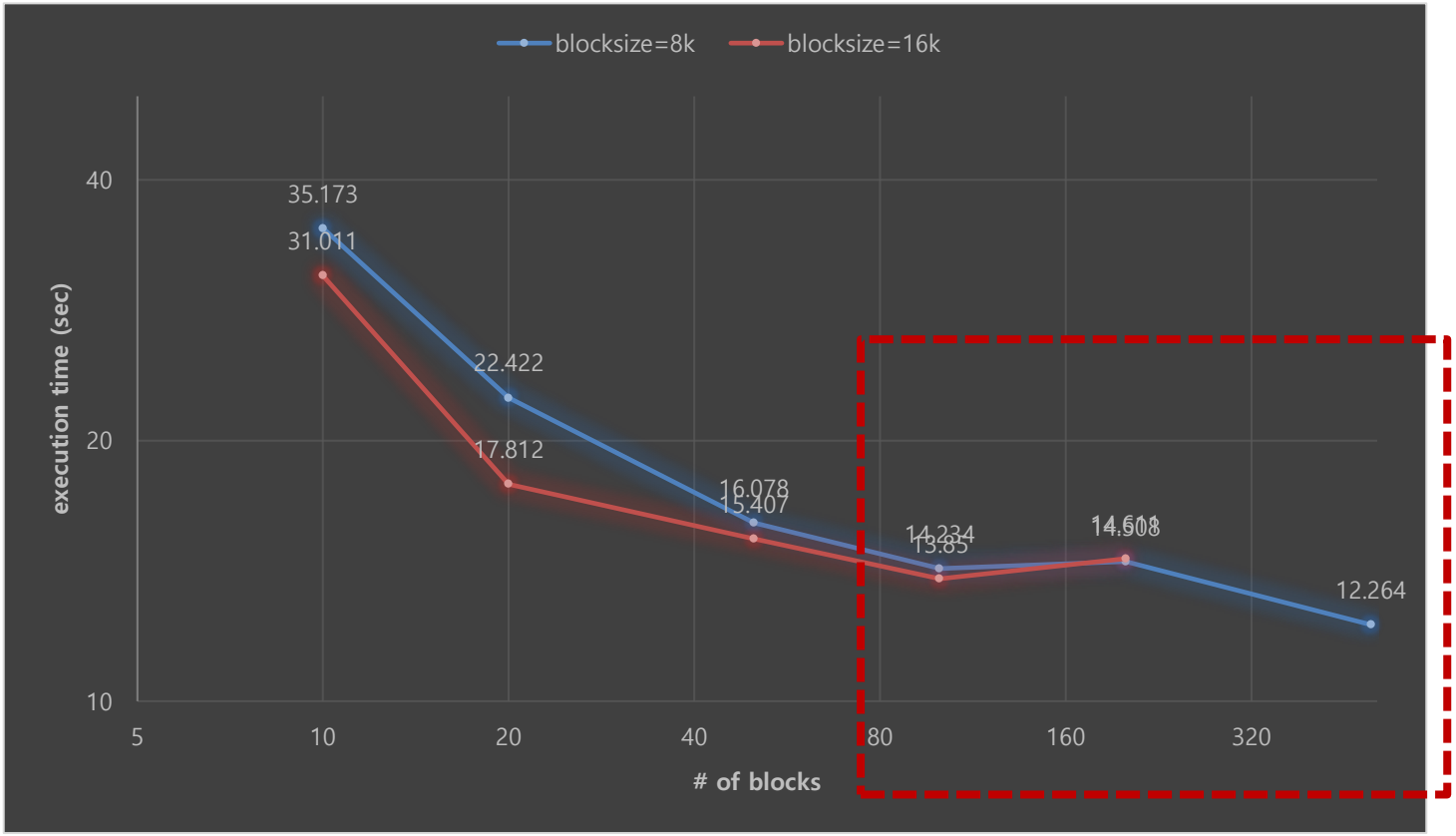
```
#!/usr/bin/bash

nblocks="100 200 500 1000"
bsizes="8192 16384"

rm -f externalsorteval.log

for n in $nblocks; do
  for s in $bsizes; do
    MAVEN_OPTS=-Xmx16m mvn exec:java -
Dexec.mainClass="edu.hanyang.test.ExternalSortEval" -
Dexec.args="$s $n" 2>> externalsorteval.log 1>/dev/null
  done
done
```

Varying Block Size and Number of Blocks



MODULE 3. EXTERNAL MERGE SORT
FAQ. M-WAY MERGE에서 RUN별 BLOCK관리

```java
import org.apache.commons.lang3.tuple.MutableTriple;

public void sort(String infile, String outfile, String tmpdir, int blocksize, int nblocks) throws IOException {
        1) initial phase
        ArrayList<MutableTriple<Integer, Integer, Integer>> dataArr = new ArrayList<>(nElement);
        ...

        2) n-way merge
         _externalMergeSort(tmpdir, outfile, 0);
}

private void _externalMergeSort(String tmpDir, String outputFile, int step) throws IOException {
         File[] fileArr = (new File(tmpDir + File.separator + String.valueOf(prevStep))).listFiles();
         if (fileArr.length <= nblocks - 1) {
                 for (File f : fileArr) {
                         DataInputStream dos = new ... (f.getAbsolutePath(), blocksize);
                 ...
        }
        else {
                 for (File f : fileArr) {
                         ...
                         cnt++;
                         if (cnt == nblocks - 1) {
                                 n_way_merge(...);
                 }
                 _externalMergeSort(tmpDir, outputFile, step+1);
}
```

```java
public void n_way_merge(List<DataInputStream> files, String outputFile) throws IOException {
        PriorityQueue<DataManager> queue = new PriorityQueue<>
                                        (files.size(), new Comparator<DataManager>() {
                public int compare(DataManager o1, DataManager o2) {
                        return o1.tuple.compareTo(o2.tuple);
                }
        });
        while (queue.size() != 0) {
                DataManager dm = queue.poll();
                MutableTriple<Integer, Integer, Integer> tmp = dm.getTuple();
                ...
        }
}
```

```java
class DataManager {
    public boolean isEOF = false;
    private DataInputStream dis = null;
    public MutableTriple<Integer, Integer, Integer> tuple = new MutableTriple<Integer, Integer, Integer>(0, 0, 0);
    public DataManager(DataInputStream dis) throws IOException { … }

    private boolean readNext() throws IOException {
        if (isEOF) return false;
        tuple.setLeft(dis.readInt()); tuple.setMiddle(dis.readInt()); tuple.setRight(dis.readInt());
        return true;
    }

    public void getTuple(MutableTriple<Integer, Integer, Integer> ret) throws IOException {
        ret.setLeft(tuple.getLeft()); ret.setMiddle(tuple.getMiddle()); ret.setRight(tuple.getRight());
        isEOF = (! readNext());
    }
```